intel

MICROCONTROLLER HANDBOOK

1983

Intel Corporation makes no warranty for the use of its products and assumes no responsibility for any errors v
appear in this document nor does it make a commitment to update the information contained herein.

Intel retains the right to make changes to these specifications at any time, without notice.

Contact your local sales office to obtain the latest specifications before placing your order.

The following are trademarks of Intel Corporation and may only be used to identify Intel Products:

BXP, CREDIT, i, ICE, I^2ICE, ICS, iDBP, iDIS, iLBX, i_m, iMMX,
Insite, INTEL, int$_e$l, Intelevision, Intellec, int$_e$ligent Identifier™,
int$_e$lBOS, int$_e$ligent Programming™, Intellink, iOSP, iPDS,
iRMS, iSBC, iSBX, iSDM, iSXM, Library Manager, MCS,
Megachassis, Micromainframe, MULTIBUS, Multichannel™
Plug-A-Bubble, MULTIMODULE, PROMPT, Ripplemode,
RMX/80, RUPI, System 2000, and UPI, and the combination of
ICE, iCS, iRMX, iSBC, MCS, or UPI and a numerical suffix.

MDS is an ordering code only and is not used as a product name or trademark. MDS® is a registered tra
Mohawk Data Sciences Corporation.

* MULTIBUS is a patented Intel bus.

Additional copies of this manual or other Intel literature may be obtained from:

Intel Corporation
Literature Department
3065 Bowers Avenue
Santa Clara, CA 95051

Table of Contents

CHAPTER 1
Introduction
1.0 Introduction to the MCS®-48 Family 1-1
1.1 Introduction to the MCS®-51 1-1

CHAPTER 2
The single component MCS®-48 System
2.0 Introduction 2-1
2.1 Architecture 2-1
2.2 Pin Description 2-16
2.3 Programming, Verifying and Erasing EPROM 2-18
2.4 Reading Internal Program Memory 2-20
2.5 8020H/8021H Functional Specifications 2-20
2.6 8022 Functional Specifications 2-26

CHAPTER 3
Expanded MCS®-48 System
3.0 Introduction 3-1
3.1 Expansion of Program Memory 3-1
3.2 Expansion of Data Memory 3-3
3.3 Expansion of Input/Output 3-4
3.4 Multi-Chip MCS®-48 Systems 3-7
3.5 Memory Bank Switching 3-8
3.6 Control Signal Summary 3-8
3.7 Port Characteristics 3-9

CHAPTER 4
MCS®-48 Instruction Set
4.0 Introduction 4-1
4.1 Instruction Set Description 4-4

Table of Contents (cont.)

CHAPTER 5

MCS®-48 Application Examples

5.0 Introduction 5-1
5.1 Hardware Examples 5-1
5.2 I/O Expansion Techniques 5-11
5.3 8049AH Emulator Circuit Description - 6 MHz 5-18
5.4 Software Examples 5-23
5.6 Single-Chip 8-Bit Microcomputer Fills Gap Between Calculator Types and Powerful Multichip Processors 5-31
5.7 Application Techniques for the MCS®-48 Family 5-38
5.8 Keyboard/Display Scanning with Intel's MCS®-48 Microcomputers 5-62
5.9 Serial I/O and Math Utilities for the 8049 Microcomputer 5-87
5.10 A High-Speed Emulator for Intel MCS®-48 Microcomputers 5-110
5.11 Using the 8049 as an 80 Column Printer Controller 5-213
5.12 Microcontroller Includes A-D Converter for Lowest-Cost Analog Interfacing 5-243
5.13 Microcomputer's On-Chip Functions Ease User's Programming Chores 5-249
5.14 Designing with Intel's 8022 Microcomputer 5-254
5.15 Energy Savings in an Induction Motor Using the 8022 Microcontroller 5-272

CHAPTER 6

MCS®-51 Architecture

6.0 Introduction 6-1
6.1 Memory Organization 6-2
6.2 Oscillator and Clock Circuit 6-3
6.3 CPU Timing 6-4
6.4 Port Structures and Operation 6-6
6.5 Accessing External Memory 6-9
6.6 Timer/Counters 6-11
6.7 Serial Interface 6-13
6.8 Interrupts 6-24
6.9 Single-Step Operation 6-26
6.10 Reset 6-26
6.11 Power Supply and Consumption Features 6-27

Table of Contents (cont.)

CHAPTER 7

MCS®-51 Memory Organization, Addressing Modes and Boolean Processor

7.0 Introduction 7-1
7.1 Memory Organization 7-1
7.2 Addressing Modes 7-1
7.3 Boolean Processor 7-4

CHAPTER 8

MCS®-51 Instruction Set

8.0 Introduction 8-1
8.1 Functional Overview 8-1
8.2 Instruction Definitions 8-3

CHAPTER 9

MCS®-51 Application Examples

9.0 8051 Programming Techniques 9-1
9.1 Peripheral Interfacing Techniques 9-13
9.2 A Microcomputer Optimized for Control .....
9.3 Microcontroller Doubles as Boolean Processor .....
9.4 An Introduction to the Intel MCS®-51 Single-Chip Microcomputer .....
9.5 Using the Intel MCS®-51 Boolean Processing Capabilities .....
9.6 Designing Microcontroller Systems for Electrically Noisy Environments .....
9.7 Microcontrollers Go CMOS Without Slowing Down .....

CHAPTER 10

Design Considerations when using CHMOS

10.0 Introduction 10-1
10.1 Power Supply Considerations 10-1
10.2 Input Considerations 10-1
10.3 CHMOS I/O Port Structure 10-1
10.4 Interfacing Between CHMOS and other Logic Families 10-2

Table of Contents (cont.)

CHAPTER 11
The RUPI™-44 Family
11.0 Introduction 11-1
11.1 Architecture Overview 11-1
11.2 The HDLC/SDLC Protocols 11-1
11.3 RUPI™-44 Design Support 11-5

CHAPTER 12
8044 Architecture
12.0 General 12-1
12.1 Memory Organization Overview 12-1
12.2 Memory Organization Details 12-4
12.3 Reset 12-8
12.4 RUPI™-44 Family Pin Description 12-8

CHAPTER 13
The 8044 Serial Interface Unit
13.0 Serial Interface 13-1
13.1 Data Link Configurations 13-1
13.2 Data Clocking Options 13-1
13.3 Data Rates 13-1
13.4 Operational Modes 13-4
13.5 8044 Frame Format Options 13-7
13.6 HDLC 13-9
13.7 SIU Special Function Registers 13-9
13.8 Operation 13-12
13.9 More Details on SIU Hardware 13-31
13.10 Diagnostics 13-33

CHAPTER 14
8044 Application Examples
14.1 Interfacing the 8044 to a Microprocessor 14-1

Table of Contents (cont.)

CHAPTER 15
MCS®-96
15.1 8096 16-Bit Microcontroller Architectural Specification and Functional Description 15-1
15.2 Controller Chip takes on many industrial, computer uses 15-32

CHAPTER 16
MCS®-48 Data Sheets
16.1 8020H 16-1
16.2 8021H 16-4
16.3 8022 16-8
16.4 Single-Component 8-Bit Microcomputers 16-16
16.5 8243 16-20
16.6 8048AH/8049AH 16-26
16.7 80C48/80C35/80C49/80C39/80C50/80C40 16-39

CHAPTER 17
RUPI™-44 Data Sheets
17.1 8044/8344/8744 17-1

CHAPTER 18
MCS®-51 Data Sheets
18.1 8032/8052 18-1
18.2 8751H 18-13
18.3 8031AH/8051AH 18-29
18.4 80C31/80C51 18-41

CHAPTER 19
General Information
19.1 Application of Intel's 5V EPROM and ROM Family for Microprocessor Systems 19-1
19.2 Crystals: Specifications for Intel Components 19-9

Introduction 1

CHAPTER 1
INTRODUCTION

1.0 INTRODUCTION TO THE MCS®-48 FAMILY

Since 1976 Intel has offered products for the full range of single-chip microcomputer applications. This was accomplished by pushing the 8048's architecture and technology in several directions.

First, the amount of program memory and data memory was doubled with the introduction of the 8049, and doubled again with the recently introduced 8050AH. Second, the MCS®-48 product family has continually been converted to Intel's latest technology. In 1980 the MCS-48 product family was converted to HMOS and in 1982 the family was converted again to HMOS II. Associated with these conversions has been a higher speed product that consumes less power. Intel is now offering CHMOS versions of most members of the MCS-48 family. This transition of processes, the popularity of the architecture and the learning curve phenomena continue to make the MCS-48 family a cost-effective solution for tomorrow's designs.

The popularity of this line of single-chips is also due to the support tools Intel has provided to the marketplace. EPROM microcomputers continue to be indispensable tools in prototyping and limited production. Training courses bring skilled instructors to teach the basics of the MCS-48 family to help you design your end product quickly. In-circuit emulation provides aid between the difficult bridge of software and hardware debugging. The literature, consisting of application notes, software programs, data sheets and user's manuals, is a powerful tool in learning the unique features of the MCS-48 architecture.

1.1 INTRODUCTION TO MCS®-51

The MCS-51 architecture significantly enhances the MCS-48 architecture. It increases the performance of the MCS-48 CPU and the number and variety of on-chip peripherals.

The MCS-51 family of components addresses applications requiring a high degree of on-chip functionality, plus a high speed CPU. The enhanced architecture offers an upward compatible growth path for MCS-48 user's. Table 1-1 outlines these features.

Table 1-1. 8051 Functions/Speed Relative to 8048

- 4X Program Memory (4K Bytes)
- 2X Data Memory (128 Bytes)
- 2X Register Banks (4 vs. 2)
- 2X Timers (Two 16-bit Timers)
- New Full-Duplex Serial I/O Port
- More I/O Pins (32 vs. 27)
- Enhanced MCS-48 Architecture
- 2X to 10X Execution Speed

The MCS-51 family currently consists of the following basic products: the 8051AH, 8031AH, 8751, 8052, and 8032. Of these products the 8051AH and 8031AH are now available in CHMOS. CHMOS is Intel technology which combines the high speed and density characteristics of HMOS with the low power attributes of CMOS. Many of the basic MCS-51 products are also available in industrial-like and military versions.

The 8051AH is a stand-alone high-performance single-chip microcomputer intended for use in sophisticated real-time applications such as instrumentation, industrial control and intelligent computer peripherals. It provides the hardware features, architectural enhancements and instructions that make it a powerful and cost effective controller for applications requiring up to 64K-bytes of program memory and/or up to 64K-bytes of data storage.

The 8031AH is a control-oriented CPU with RAM and I/O. It does not have on-chip program memory. It can address up to 64K-bytes of external program memory and up to 64K-bytes to external data memory.

The 8751 is an 8051AH with 4K-bytes of UV-light-erasable/electrically-programmable program memory.

The 8052 is an 8051AH with double the on-chip memory —8K-bytes of ROM and 256 bytes of RAM. The 8052 also features an additional, more flexible 16-bit timer/counter. The 8032 is an 8052 without the 8K-bytes of internal ROM.

The 8044 integrates an 8051AH core with a high performance HDLC/SDLC serial communication controller called the Serial Interface Unit, SIU. For networking applications, the 8044 provides designers with 8051 real time control capability at a node, and interface that node to a serial network at up to 2.4 million bits per second. Because the SIU offloads the CPU of all network communication tasks, the CPU can concentrate on controlling the node. The 8044 is software compatible with the 8051AH.

For systems requiring extra capability, each member of the MCS-51 family can be expanded using standard memories and most byte oriented MCS-80 and MCS-85 peripherals.

All of the compatible MCS-51 members allow for reduced development problems and provide maximum flexibility. The 8751 is well suited for development, prototyping, low volume production and applications requiring field updates; the 8051, 8052, for low cost high volume production; and the 8031, 8032, for applications desiring the flexibility of external program memory which can be easily modified and updated in the field.

The Single Component MCS®-48 System 2

CHAPTER 2
THE SINGLE COMPONENT MCS®-48 SYSTEM

2.0 INTRODUCTION

Sections 2.1 through 2.4 describe in detail the functional characteristics of the 8748H and 8749H EPROM, 8748H/8049AH/8050AH ROM, and 8035AHL/8039AHL/8040AHL CPU only single component microcomputers. Unless otherwise noted, details within these sections apply to all versions. Section 2.5 describes the operation of the 8020H/8021H and Section 2.6 describes the 8022. This chapter is limited to those functions useful in single-chip implementations of the MCS®-48. Chapter 3 discusses functions which allow expansion of program memory, data memory, and input output capability.

2.1 ARCHITECTURE

The following sections break the MCS-48 Family into functional blocks and describe each in detail. The following description will use the 8048AH as the representative product for the family. See Figure 2-1.

2.1.1 Arithmetic Section

The arithmetic section of the processor contains the basic data manipulation functions of the 8048AH and can be divided into the following blocks:

- Arithmetic Logic Unit (ALU)
- Accumulator
- Carry Flag
- Instruction Decoder

In a typical operation data stored in the accumulator is combined in the ALU with data from another source on the internal bus (such as a register or I/O port) and the result is stored in the accumulator or another register.

The following is more detailed description of the function of each block.

INSTRUCTION DECODER

The operation code (op code) portion of each program instruction is stored in the Instruction Decoder and converted to outputs which control the function of each of the blocks of the Arithmetic Section. These lines control the source of data and the destination register as well as the function performed in the ALU.

ARITHMETIC LOGIC UNIT

The ALU accepts 8-bit data words from one or two sources and generates an 8-bit result under control of the Instruction Decoder. The ALU can perform the following functions:

- Add With or Without Carry
- AND, OR, Exclusive OR
- Increment/Decrement
- Bit Complement
- Rotate Left, Right
- Swap Nibbles
- BCD Decimal Adjust

If the operation performed by the ALU results in a value represented by more than 8 bits (overflow of most significant bit), a Carry Flag is set in the Program Status Word.

ACCUMULATOR

The accumulator is the single most important data register in the processor, being one of the sources of input to the ALU and often the destination of the result of operations performed in the ALU. Data to and from I/O ports and memory also normally passes through the accumulator.

2.1.2 Program Memory

Resident program memory consists of 1024, 2048, or 4096 words eight bits wide which are addressed by the program counter. In the 8748H and the 8749H this memory is user programmable and erasable EPROM; in the 8048AH/8049AH/8050AH the memory is ROM which is mask programmable at the factory. The 8035AHL/8039AHL/8040AHL has no internal program memory and is used with external memory devices. Program code is completely interchangeable among the various versions. To access the upper 2K of program memory in the 8050AH, and other MCS-48 devices, a select memory bank and a JUMP or CALL instruction must be executed to cross the 2K boundary. See Section 2.3 for EPROM programming techniques.

There are three locations in Program Memory of special importance as shown in Figure 2-2.

LOCATION 0
Activating the Reset line of the processor causes the first instruction to be fetched from location 0.

LOCATION 3
Activating the Interrupt input line of the processor (if interrupt is enabled) causes a jump to subroutine at location 3.

LOCATION 7
A timer/counter interrupt resulting from timer/counter overflow (if enabled) causes a jump to subroutine at location 7.

Therefore, the first instruction to be executed after initialization is stored in location 0, the first word of an external interrupt service subroutine is stored in location 2, and the first word of a timer/counter service routines is stored

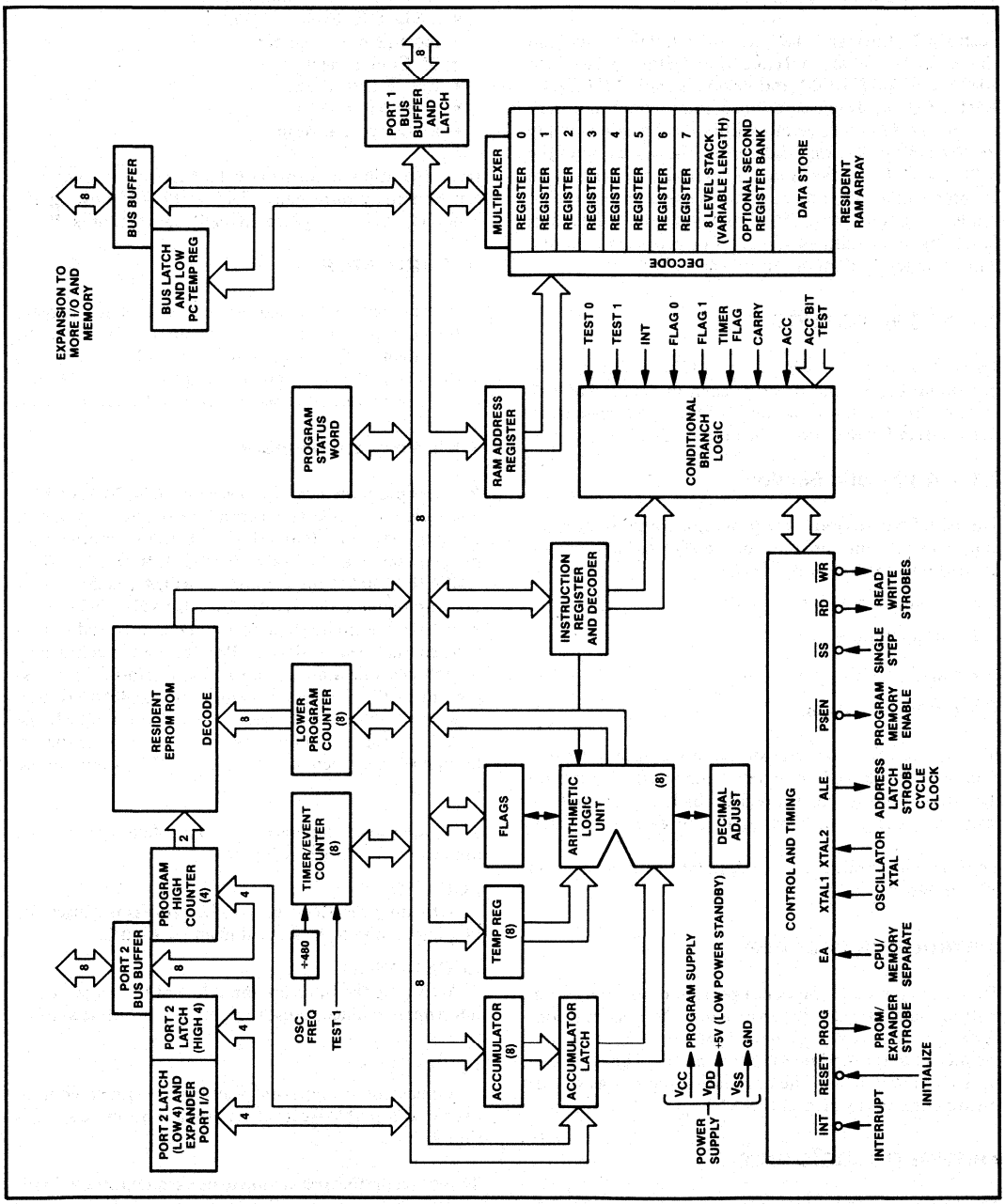


Figure 2-1. 8748H/8048AH/8749H/8049AH/8050AH Block Diagram

in location 7. Program memory can be used to store constants as well as program instructions. Instructions such as MOVP and MOVP3 allow easy access to data "look-up" tables.

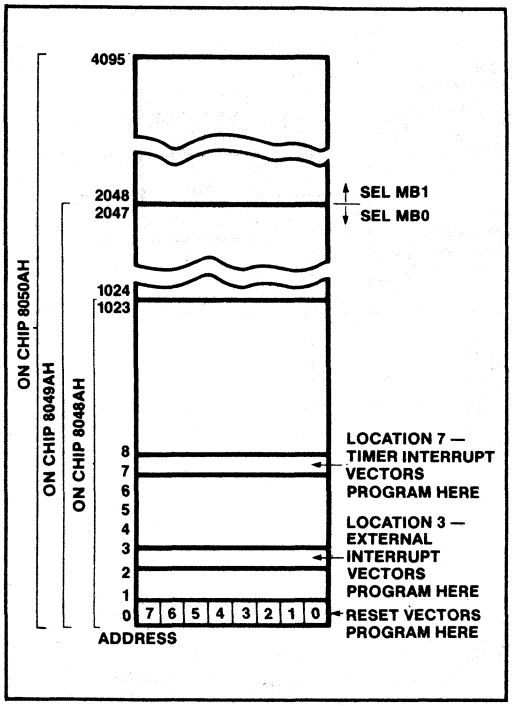


Figure 2-2. Program Memory Map

2.1.3 Data Memory

Resident data memory is organized as 64, 128, or 256 words 8-bits wide in the 8048AH, 8049AH and 8050AH. All locations are indirectly addressable through either of two RAM Pointer Registers which reside at address 0 and 1 of the register array. In addition, as shown in Figure 2-3, the first 8 locations (0–7) of the array are designated as working registers and are directly addressable by several instructions. Since these registers are more easily addressed, they are usually used to store frequently accessed intermediate results. The DJNZ instruction makes very efficient use of the working registers as program loop counters by allowing the programmer to decrement and test the register in a single instruction.

By executing a Register Bank Switch instruction (SEL RB) RAM locations 24–31 are designated as the working registers in place of locations 0–7 and are then directly addressable. This second bank of working registers may be used as an extension of the first bank or reserved for use during interrupt service subroutines allowing the registers of Bank 0 used in the main program to be instantly "saved" by a Bank Switch. Note that if this second bank is not used, locations 24–31 are still addressable as general purpose RAM. Since the two RAM pointer Registers R0 and R1 are a part of the working register array, bank switching effectively creates two more pointer registers (R0′ and R1′) which can be used with R0 and R1 to easily access up to four separate working areas in RAM at one time. RAM locations (8–23) also serve a dual role in that they contain the program counter stack as explained in Section 2.1.6. These locations are addressed by the Stack Pointer during subroutine calls as well as by RAM Pointer Registers R0 and R1. If the level of subroutine nesting is less than 8, all stack registers are not required and can be used as general purpose RAM locations. Each level of subroutine nesting not used provides the user with two additional RAM locations.

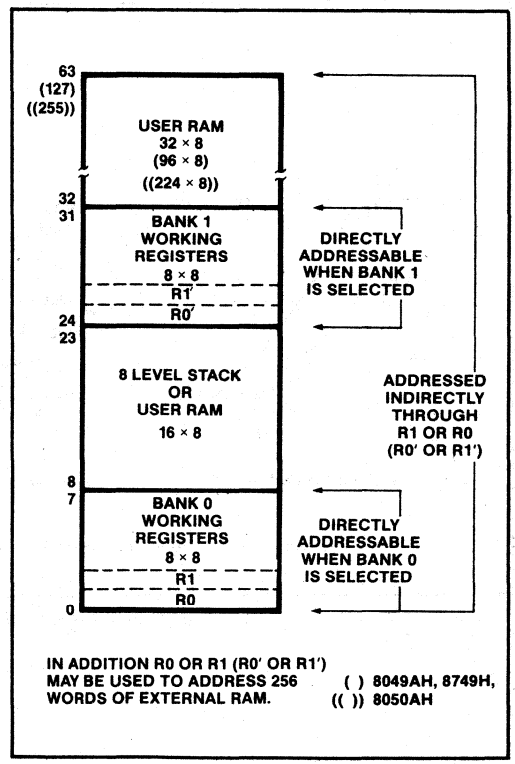


Figure 2-3. Data Memory Map

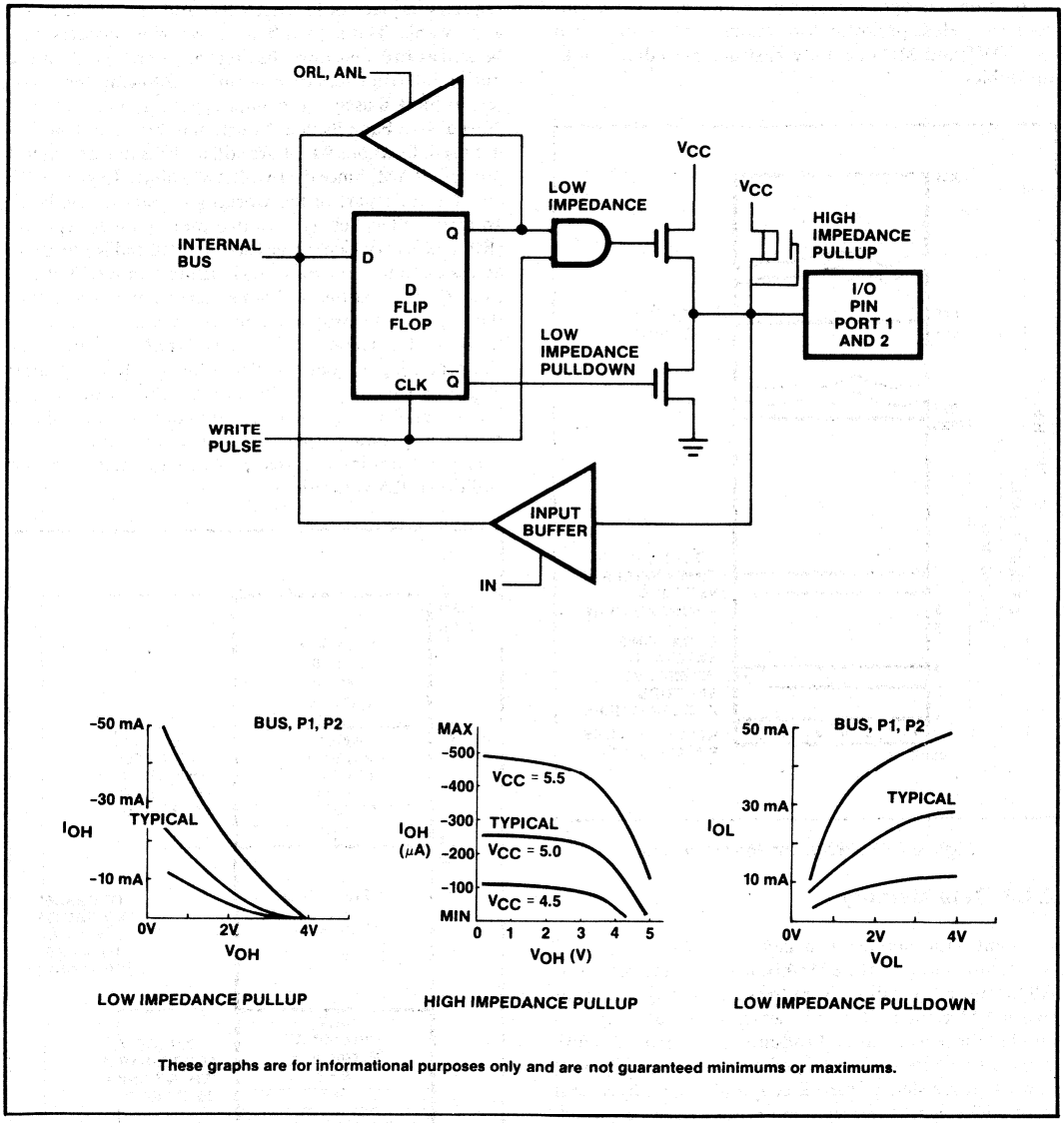


Figure 2-4. "Quasi-bidirectional" Port Structure

2.1.4 Input/Output

The 8048AH has 27 lines which can be used for input or output functions. These lines are grouped as 3 ports of 8 lines each which serve as either inputs, outputs or bi-directional ports and 3 "test" inputs which can alter program sequences when tested by conditional jump instructions.

PORTS 1 AND 2

Ports 1 and 2 are each 8 bits wide and have identical characteristics. Data written to these ports is statically latched and remains unchanged until rewritten. As input ports these lines are non-latching, i.e., inputs must be present until read by an input instruction. Inputs are fully TTL compatible and outputs will drive one standard TTL load.

The lines of ports 1 and 2 are called quasi-bidirectional because of a special output circuit structure which allows each line to serve as an input, and output, or both even though outputs are statically latched. Figure 2-4 shows the circuit configuration in detail. Each line is continuously pulled up to V_{CC} through a resistive device of relatively high impedance.

This pullup is sufficient to provide the source current for a TTL high level yet can be pulled low by a standard TTL gate thus allowing the same pin to be used for both input and output. To provide fast switching times in a "0" to "1" transition a relatively low impedance device is switched in momentarily ($\approx$ 1/5 of a machine cycle) whenever a "1" is written to the line. When a "0" is written to the line a low impedance device overcomes the light pullup and provides TTL current sinking capability. Since the pulldown transistor is a low impedance device a "1" must first be written to any line which is to be used as an input. Reset initializes all lines to the high impedance "1" state.

It is important to note that the ORL and ANL are read/write operations. When executed, the μC "reads" the port, modifies the data according to the instruction, then "writes" the data back to the port. The "writing" (essentially an outL instruction) enables the low impedance pull-up momentarily again even if the data was unchanged from a "1". This specifically applies to configurations that have inputs and outputs mixed together on the same port. See also section 3.7.

BUS

Bus is also an 8-bit port which is a true bidirectional port with associated input and output strobes. If the bidirectional feature is not needed, Bus can serve as either a statically latched output port or non-latching inut port. Input and output lines on this port cannot be mixed however.

As a static port, data is written and latched using the OUTL instruction and inputted using the INS instruction. The INS and OUTL instructions generate pulses on the corresponding $\overline{RD}$ and $\overline{WR}$ output strobe lines; however, in the static port mode they are generally not used. As a bidirectional port the MOVX instructions are used to read and write the port. A write to the port generates a pulse on the $\overline{WR}$ output line and output data is valid at the trailing edge of $\overline{WR}$. A read of the port generates a pulse on the $\overline{RD}$ output line and input data must be valid at the trailing edge of $\overline{RD}$. When not being written or read, the BUS lines are in a high impedance state. See also sections 3.6 and 3.7.

2.1.5 Test and INT Inputs

Three pins serve as inputs and are testable with the conditional jump instruction. These are T0, T1, and $\overline{INT}$. These pins allow inputs to cause program branches without the necessity to load an input port into the accumulator. The T0, T1, and $\overline{INT}$ pins have other possible functions as well. See the pin description in Section 2.2.

2.1.6 Program Counter and Stack

The Program Counter is an independent counter while the Program Counter Stack is implemented using pairs of registers in the Data Memory Array. Only 10, 11, or 12 bits of the Program Counter are used to address the 1024, 2048, or 4096 words of on-board program memory of the 8048AH, 8049AH, or 8050AH, while the most significant bits can be used for external Program Memory fetches. See Figure 2-5. The Program Counter is initialized to zero by activating the Reset line.

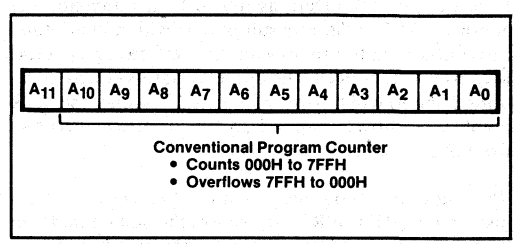


Figure 2-5. Program Counter

An interrupt or CALL to a subroutine causes the contents of the program counter to be stored in one of the 8 register pairs of the Program Counter Stack as shown in Figure 2-6. The pair to be used is determined by a 3-bit Stack Pointer which is part of the Program Status Word (PSW).

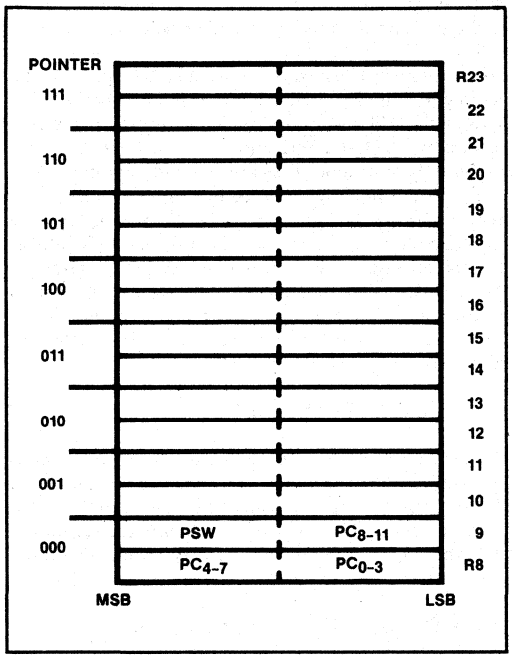


Figure 2-6. Program Counter Stack

Data RAM locations 8–23 are available as stack registers and are used to store the Program Counter and 4 bits of PSW as shown in Figure 2-6. The Stack Pointer when initialized to 000 points to RAM locations 8 and 9. The first subroutine jump or interrupt results in the program counter contents being transferred to locations 8 and 9 of the RAM array. The stack pointer is then incremented by one to point to locations 10 and 11 in anticipation of another CALL. Nesting of subroutines within subroutines can continue up to 8 times without overflowing the stack. If overflow does occur the deepest address stored (locations 8 and 9) will be overwritten and lost since the stack pointer overflows from 111 to 000. It also underflows from 000 to 111.

The end of a subroutine, which is signalled by a return instruction (RET or RETR), causes the Stack Pointer to be decremented and the contents of the resulting register pair to be transferred to the Program Counter.

2.1.7 Program Status Word

An 8-bit status word which can be loaded to and from the accumulator exists called the Program Status Word (PSW). Figure 2-7 shows the information available in the word. The Program Status Word is actually a collection of flip-flops throughout the machine which can be read or written as a whole. The ability to write to PSW allows for easy restoration of machine status after a power down sequence.

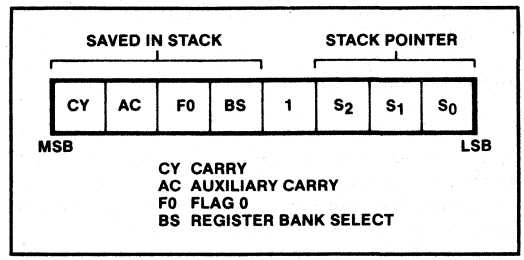


Figure 2-7. Program Status Word (PSW)

The upper four bits of PSW are stored in the Program Counter Stack with every call to subroutine or interrupt vector and are optionally restored upon return with the RETR instruction. The RET return instruction does not update PSW.

The PSW bit definitions are as follows:

Bits 0–2: Stack Pointer bits (S_0, S_1, S_2)

Bit 3: Not used ("1" level when read)

Bit 4: Working Register Bank Switch Bit (BS)
0 = Bank 0
1 = Bank 1

Bit 5: Flag 0 bit (F0) user controlled flag which can be complemented or cleared, and tested with the conditional jump instruction JF0.

Bit 6: Auxiliary Carry (AC) carry bit generated by an ADD instruction and used by the decimal adjust instruction DA A.

Bit 7: Carry (CY) carry flag which indicates that the previous operation has resulted in overflow of the accumulator.

2.1.8 Conditional Branch Logic

The conditional branch logic within the processor enables several conditions internal and external to the processor to be tested by the users program. By using the conditional jump instruction the conditions that are listed in Table 2-1 can effect a change in the sequence of the program execution.

Table 2-1

Device Testable	Jump Conditions (Jump On)	
	All zeros	not all zeros
Accumulator	—	1
Accumulator Bit	0	1
Carry Flag	—	1
User Flags (F0, F1)	—	1
Timer Overflow Flag	0	1
Test Inputs (T0, T1)	0	1
Interrupt Input ($\overline{INT}$)	0	—

2.1.9 Interrupt

An interrupt sequence is initiated by applying a low "0" level input to the INT pin. Interrupt is level triggered and active low to allow "WIRE ORing" of several interrupt sources at the input pin. Figure 2-8 shows the interrupt logic of the 8048AH. The Interrupt line is sampled every instruction cycle and when detected causes a "call to subroutine" at location 3 in program memory as soon as all cycles of the current instruction are complete. On 2-cycle instructions the interrupt line is sampled on the 2nd cycle only. $\overline{INT}$ must be held low for at least 3 machine cycles to ensure proper interrupt operations. As in any CALL to subroutine, the Program Counter and Program Status word are saved in the stack. For a description of this operation see the previous section, Program Counter and Stack. Program Memory location 3 usually contains an unconditional jump to an interrupt service subroutine elsewhere in program memory. The end of an interrupt service subroutine is signalled by the execution of a Return and Restore Status instruction RETR. The interrupt system is single level in that once an interrupt is detected all further interrupt requests are ignored until execution of an RETR reenables the interrupt input logic. This occurs at the beginning of the second cycle of the RETR instruction. This sequence holds true also for an internal interrupt generated by timer overflow. If an internal timer/counter generated interrupt and an external interrupt are detected at the same time, the external source will be recognized. See the following Timer/Counter section for a description of timer interrupt. If needed, a second external interrupt can be created by enabling the timer/counter interrupt, loading FFH in the Counter (one less than terminal count), and enabling the event counter mode. A "1" to "0" transition on the T1 input will then cause an interrupt vector to location 7.

INTERRUPT TIMING

The interrupt input may be enabled or disabled under Program Control using the EN I and DIS I instructions. Interrupts are disabled by Reset and remain so until enabled by the users program. An interrupt request must be removed before the RETR instruction is executed upon return from the service routine otherwise the processor will re-enter the service routine immediately. Many peripheral devices prevent this situation by resetting their interrupt request line whenever the processor accesses (Reads or Writes) the peripherals data buffer register. If the interrupting device does not require access by the processor, one output line of the 8048AH may be designated as an "interrupt acknowledge" which is activated by the service subroutine to reset the interrupt request. The $\overline{INT}$ pin may also be tested using the conditional jump instruction JNI. This instruction may be used to detect the presence of a pending interrupt before interrupts are enabled. If interrupt is left disabled, $\overline{INT}$ may be used as another test input like T0 and T1.

2.1.10 Timer/Counter

The 8048AH contains a counter to aid the user in counting external events and generating accurate time delays without placing a burden on the processor for these functions. In both modes the counter operation is the same, the only difference being the source of the input to the counter. The timer/event counter is shown in Figure 2-9.

COUNTER

The 8-bit binary counter is presettable and readable with two MOV instructions which transfer the contents of the accumulator to the counter and vice versa. The counter content may be affected by Reset and should be initialized by software. The counter is stopped by a Reset or STOP TCNT instruction and remains stopped until started as a timer by a START T instruction or as an event counter by a START CNT instruction. Once started the counter will increment to this maximum count (FF) and overflow to zero continuing its count until stopped by a STOP TCNT instruction or Reset.

The increment from maximum count to zero (overflow) results in the setting of an overflow flag flip-flop and in the generation of an interrupt request. The state of the overflow flag is testable with the conditional jump instruction JTF. The flag is reset by executing a JTF or by Reset. The interrupt request is stored in a latch and then ORed with the external interrupt input INT. The timer interrupt may be enabled or disabled independently of external interrupt by the EN TCNTI and DIS TCNTI instructions. If enabled, the counter overflow will cause a subroutine call to location 7 where the timer or counter service routine may be stored.

If timer and external interrupts occur simultaneously, the external source will be recognized and the Call will be to

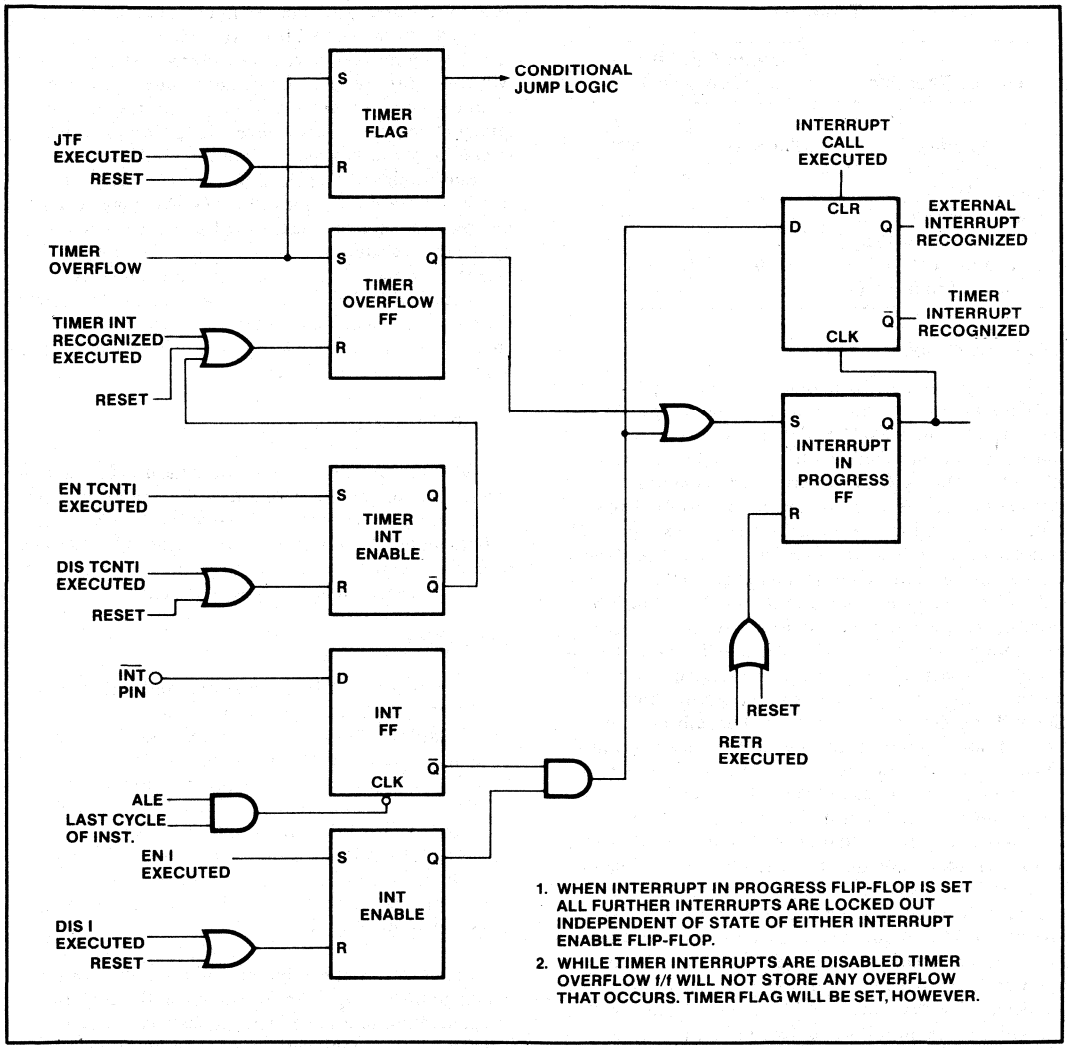


Figure 2-8. Interrupt Logic

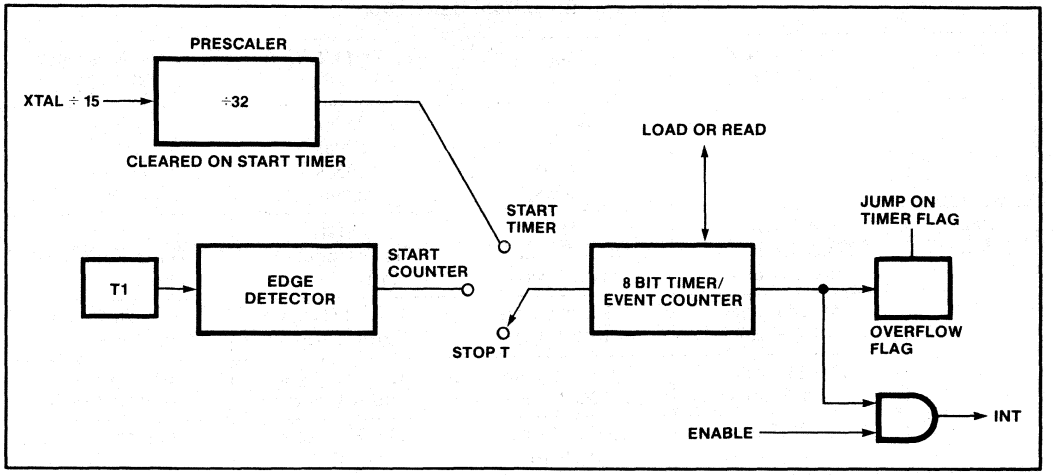


Figure 2-9. Timer/Event Counter

location 3. Since the timer interrupt is latched it will remain pending until the external device is serviced and immediately be recognized upon return from the service routine. The pending timer interrupt is reset by the Call to location 7 or may be removed by executing a DIS TCNTI instruction.

AS AN EVENT COUNTER

Execution of a START CNT instruction connects the T1 input pin to the counter input and enables the counter. The T1 input is sampled at the beginning of state 3 or in later MCS-48 devices in state time 4. Subsequent high to low transitions on T1 will cause the counter to increment. T1 must be held low for at least 1 machine cycle to insure it won't be missed. The maximum rate at which the counter may be incremented is once per three instruction cycles (every 5.7 μsec when using an 8 MHz crystal)—there is no minimum frequency. T1 input must remain stable for at least 1/5 machine cycle after each transition.

AS A TIMER

Execution of a START T instruction connects an internal clock to the counter input and enables the counter. The internal clock is derived by passing the basic machine cycle clock through a ÷ 32 prescaler. The prescaler is reset during the START T instruction. The resulting clock increments the counter every 32 machine cycles. Various delays from 1 to 256 counts can be obtained by presetting the counter and detecting overflow. Times longer than 256 counts may be achieved by accumulating multiple overflows in a register under software control. For time resolution less than 1 count an external clock can be applied to the T1 input and the counter operated in the event counter mode. ALE divided by 3 or more can serve as this external clock. Very small delays or "fine tuning" of larger delays can be easily accomplished by software delay loops.

Often a serial link is desirable in an MCS-48 family member. Table 2-2 lists the timer counts and cycles needed for a specific baud rate given a crystal frequency.

2.1.11 Clock and Timing Circuits

Timing generation for the 8048AH is completely self-contained with the exception of a frequency reference which can be XTAL, ceramic resonator, or external clock source. The Clock and Timing circuitry can be divided into the following functional blocks.

OSCILLATOR

The on-board oscillator is a high gain parallel resonant circuit with a frequency range of 1 to 11 MHz. The X1 external pin is the input to the amplifier stage while X2 is the output. A crystal or ceramic resonator connected between X1 and X2 provides the feedback and phase shift required for oscillation. If an accurate frequency reference is not required, ceramic resonator may be used in place of the crystal.

For accurate clocking, a crystal should be used. An externally generated clock may also be applied to X1–X2 as the frequency source. See the data sheet for more information.

Table 2-2. Baud Rate Generation

	Frequency (MHz)	T_{cy}	T0 Prr(1/5 T_{cy})	Timer Prescaler (32 T_{cy})
	4 6 8 11	3.75μs 2.50μs 1.88μs 1.36μs	750ns 500ns 375ns 275ns	120μs 80μs 60.2μs 43.5μs
Baud Rate	4 MHz Timer Counts + Instr. Cycles	6 MHz Timer Counts + Instr. Cycles	8 MHz Timer Counts + Instr. Cycles	11 MHz Timer Counts + Instr. Cycles
110	75 + 24 Cycles .01% Error	113 + 20 Cycles .01% Error	151 + 3 Cycles .01% Error	208 + 28 Cycles .01% Error
300	27 + 24 Cycles .1% Error	41 + 21 Cycles .03% Error	55 + 13 Cycles .01% Error	76 + 18 Cycles .04% Error
1200	6 + 30 Cycles .1% Error	10 + 13 Cycles .1% Error	13 + 27 Cycles .06% Error	19 + 4 Cycles .12% Error
1800	4 + 20 Cycles .1% Error	6 + 30 Cycles .1% Error	9 + 7 Cycles .17% Error	12 + 24 Cycles .12% Error
2400	3 + 15 Cycles .1% Error	5 + 6 Cycles .4% Error	6 + 24 Cycles .29% Error	9 + 18 Cycles .12% Error
4800	1 + 23 Cycles 1.0% Error	2 + 19 Cycles .4% Error	3 + 14 Cycles .74% Error	4 + 25 Cycles .12% Error

STATE COUNTER

The output of the oscillator is divided by 3 in the State Counter to create a clock which defines the state times of the machine (CLK). CLK can be made available on the external pin T0 by executing an ENTO CLK instruction. The output of CLK on T0 is disabled by Reset of the processor.

CYCLE COUNTER

CLK is then divided by 5 in the Cycle Counter to provide a clock which defines a machine cycle consisting of 5 machine states as shown in Figure 2-10. Figure 2-11 shows the different internal operations as divided into the machine states. This clock is called Address Latch Enable (ALE) because of its function in MCS-48 systems with external memory. It is provided continuously on the ALE output pin.

2.1.12 Reset

The reset input provides a means for initialization for the processor. This Schmitt-trigger input has an internal pull-up device which in combination with an external 1μ fd capacitor provides an internal reset pulse of sufficient length to guarantee all circuitry is reset, as shown in Figure 2-12. If the reset pulse is generated externally the RESET pin must be held low for at least 10 milliseconds after the power supply is within tolerance. Only 5 machine cycles (6.8 μs @ 11 MHz) are required if power is already on and the oscillator has stabilized. ALE and PSEN (if EA = 1) are active while in Reset.

Reset performs the following functions:

1) Sets program counter to zero.
2) Sets stack pointer to zero.
3) Selects register bank 0.
4) Selects memory bank 0.
5) Sets BUS to high impedance state (except when EA = 5V).
6) Sets Ports 1 and 2 to input mode.
7) Disables interrupts (timer and external).
8) Stops timer.
9) Clears timer flag.
10) Clears F0 and F1.
11) Disables clock output from T0.

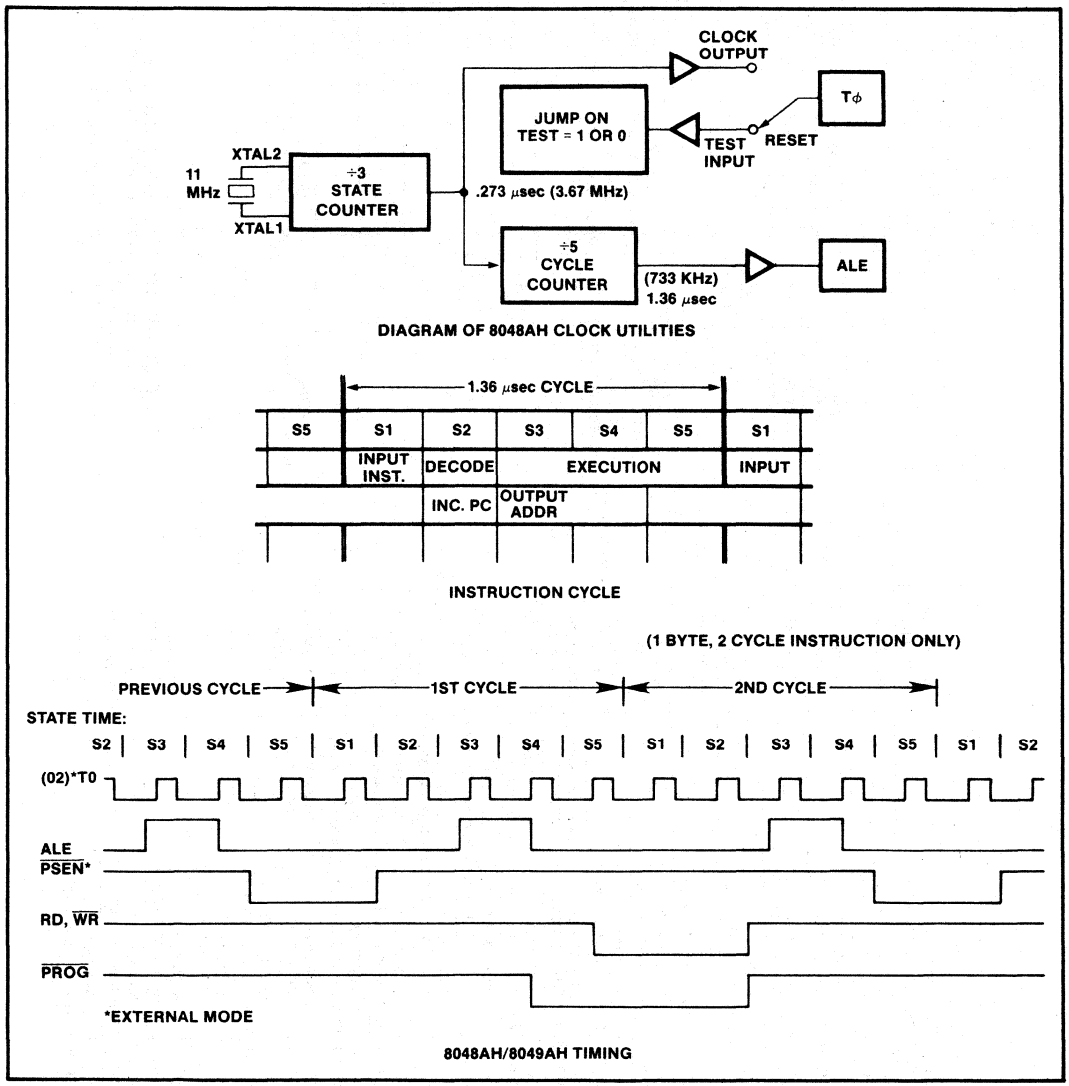


Figure 2-10. MCS®-48 Timing Generation and Cycle Timing

2.1.13 Single-Step

This feature, as pictured in Figure 2-13, provides the user with a debug capability in that the processor can be stepped through the program one instruction at a time. While stopped, the address of the next instruction to be fetched is available concurrently on BUS and the lower half of Port 2. The user can therefore follow the program through each of the instruction steps. A timing diagram, showing the interaction between output ALE and input $\overline{SS}$, is shown. The BUS buffer contents are lost during single step; however, a latch may be added to re-establish the lost I/O capability if needed. Data is valid at the leading edge of ALE.

	CYCLE 1					CYCLE 2				
INSTRUCTION	S1	S2	S3	S4	S5	S1	S2	S3	S4	S5
IN A,P	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	—	*INCREMENT TIMER	—	—	READ PORT	—	* —	—
OUTL P,A	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	—	*INCREMENT TIMER	OUTPUT TO PORT	—	—	—	* —	—
ANL P, = DATA	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	—	*INCREMENT TIMER	READ PORT	FETCH IMMEDIATE DATA	—	INCREMENT PROGRAM COUNTER	*OUTPUT TO PORT	—
ORL P, = DATA	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	—	*INCREMENT TIMER	READ PORT	FETCH IMMEDIATE DATA	—	INCREMENT PROGRAM COUNTER	*OUTPUT TO PORT	—
INS A, BUS	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	—	INCREMENT TIMER	—	—	READ PORT	—	* —	—
OUTL BUS, A	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	—	INCREMENT TIMER	OUTPUT TO PORT	—	—	—	* —	—
ANL BUS, = DATA	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	—	*INCREMENT TIMER	READ PORT	FETCH IMMEDIATE DATA	—	INCREMENT PROGRAM COUNTER	*OUTPUT TO PORT	—
ORL BUS, = DATA	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	—	*INCREMENT TIMER	READ PORT	FETCH IMMEDIATE DATA	—	INCREMENT PROGRAM COUNTER	*OUTPUT TO PORT	—
MOVX @ R,A	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	OUTPUT RAM ADDRESS	INCREMENT TIMER	OUTPUT DATA TO RAM	—	—	—	* —	—
MOVX A,@R	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	OUTPUT RAM ADDRESS	INCREMENT TIMER	—	—	READ DATA	—	* —	—
MOVD A,P_i	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	OUTPUT OPCODE/ADDRESS	INCREMENT TIMER	—	—	READ P2 LOWER	—	* —	—
MOVD P_i,A	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	OUTPUT OPCODE/ADDRESS	INCREMENT TIMER	OUTPUT DATA TO P2 LOWER	—	—	—	* —	—
ANLD P,A	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	OUTPUT OPCODE/ADDRESS	INCREMENT TIMER	OUTPUT DATA	—	—	—	* —	—
ORLD P,A	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	OUTPUT OPCODE/ADDRESS	INCREMENT TIMER	OUTPUT DATA	—	—	—	* —	—
J(CONDITIONAL)	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	SAMPLE CONDITION	*INCREMENT SAMPLE	—	FETCH IMMEDIATE DATA	—	UPDATE PROGRAM COUNTER	* —	—
STRT T STRT CNT	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	—	* —	START COUNTER					
STOP TCNT	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	—	* —	STOP COUNTER					
ENI	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	—	* ENABLE INTERRUPT	—					
DIS I	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER	—	* DISABLE INTERRUPT	—					
ENTO CLK	FETCH INSTRUCTION	INCREMENT PROGRAM COUNTER		* ENABLE CLOCK	—					

***VALID INSTRUCTION ADDRESSES ARE OUTPUT AT THIS TIME IF EXTERNAL PROGRAM MEMORY IS BEING ACCESSED.**

(1) IN LATER MCS-48 DEVICES T1 IS SAMPLED IN S4.

Figure 2-11. 8048AH/8049AH Instruction Timing Diagram

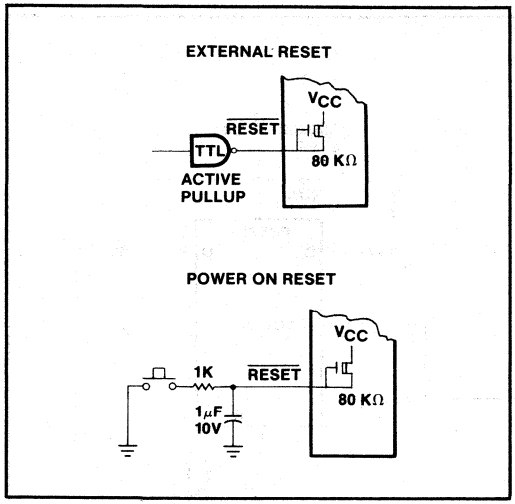


Figure 2-12

TIMING

The 8048AH operates in a single-step mode as follows:

1) The processor is requested to stop by applying a low level on $\overline{SS}$.

2) The processor responds by stopping during the address fetch portion of the next instruction. If a double cycle instruction is in progress when the single step command is received, both cycles will be completed before stopping.

3) The processor acknowledges it has entered the stopped state by raising ALE high. In this state (which can be maintained indefinitely) the address of the next instruction to be fetched is present on BUS and the lower half of port 2.

4) $\overline{SS}$ is then raised high to bring the processor out of the stopped mode allowing it to fetch the next instruction. The exit from stop is indicated by the processor bringing ALE low.

5) To stop the processor at the next instruction $\overline{SS}$ must be brought low again soon after ALE goes low. If $\overline{SS}$ is left high the processor remains in a "Run" mode.

A diagram for implementing the single-step function of the 8748H is shown in Figure 2-13. A D-type flip-flop with preset and clear is used to generate $\overline{SS}$. In the run mode SS is held high by keeping the flip-flop preset (preset has precedence over the clear input). To enter single step, preset is removed allowing ALE to bring $\overline{SS}$ low via the clear input. ALE should be buffered since the clear input of an SN7474 is the equivalent of 3 TTL loads. The processor is now in the stopped state. The next instruction is initiated by clocking a "1" into the flip-flop. This "1" will not appear on $\overline{SS}$ unless ALE is high removing clear from the flip-flop. In response to $\overline{SS}$ going high the processor begins an instruction fetch which brings ALE low resetting $\overline{SS}$ through the clear input and causing the processor to again enter the stopped state.

2.1.14 Power Down Mode (8048AH, 8049AH, 8050AH, 8039AHL, 8035AHL, 8040AHL)

Extra circuitry has been added to the 8048AH/8049AH/8050AH ROM version to allow power to be removed from all but the data RAM array for low power standby operation. In the power down mode the contents of data RAM can be maintained while drawing typically 10% to 15% of normal operating power requirements.

V_{CC} serves as the 5V supply pin for the bulk of circuitry while the V_{DD} pin supplies only the RAM array. In normal operation both pins are a 5V while in standby, V_{CC} is at ground and V_{DD} is maintained at its standby value. Applying Reset to the processor through the $\overline{RESET}$ pin inhibits any access to the RAM by the processor and guarantees that RAM cannot be inadvertently altered as power is removed from V_{CC}.

A typical power down sequence (Figure 2-14) occurs as follows:

1) Imminent power supply failure is detected by user defined circuitry. Signal must be early enough to allow 8048AH to save all necessary data before V_{CC} falls below normal operating limits.

2) Power fail signal is used to interrupt processor and vector it to a power fail service routine.

3) Power fail routine saves all important data and machine status in the internal data RAM array. Routine may also initiate transfer of backup supply to the V_{DD} pin and indicate to external circuitry that power fail routine is complete.

4) Reset is applied to guarantee data will not be altered as the power supply falls out of limits. Reset must be held low until V_{CC} is at ground level.

Recovery from the Power Down mode can occur as any other power-on sequence with an external capacitor on the Reset input providing the necessary delay. See the previous section on Reset.

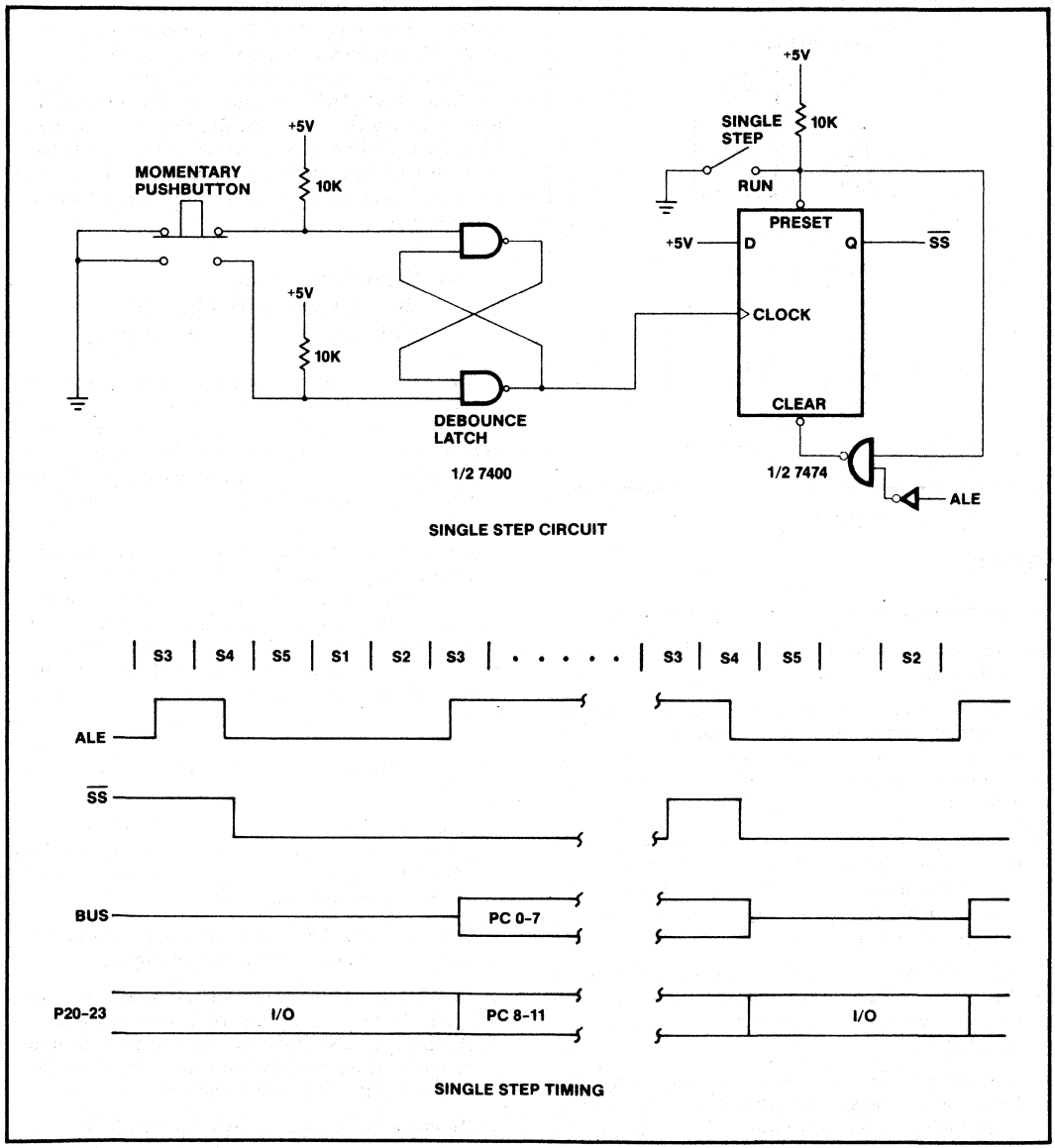


Figure 2-13. Single Step Operation

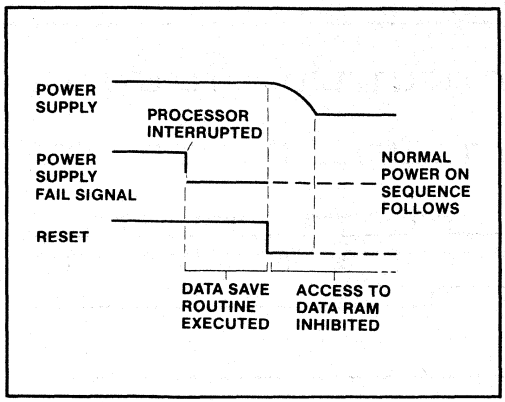


Figure 2-14. Power Down Sequence

2.1.15 External Access Mode

Normally the first 1K (8048AH), 2K (8049AH), or 4K (8050AH) words of program memory are automatically fetched from internal ROM or EPROM. The EA input pin however allows the user to effectively disable internal program memory by forcing all program memory fetches to reference external memory. The following chapter explains how access to external program memory is accomplished.

The External Access mode is very useful in system test and debug because it allows the user to disable his internal applications program and substitute an external program of his choice—a diagnostic routine for instance. In addition, the section on Test and Debug explains how internal program memory can be read externally, independent of the processor. A "1" level on EA initiates the external access mode. For proper operation, Reset should be applied while the EA input is changed.

2.1.16 Sync Mode

The 8048AH, 8049AH, 8050AH has incorporated a new SYNC mode. The Sync mode is provided to ease the design of multiple controller circuits by allowing the designer to force the device into known phase and state time. The SYNC mode may also be utilized by automatic test equipment (ATE) for quick, easy, and efficient synchronizing between the tester and the DUT (device under test).

SYNC mode is enabled when SS′ pin is raised to a high voltage level of +12 volts. To begin synchronization, T0 is raised to 5 volts at least four clocks cycles after SS′. T0 must be high for at least four X1 clock cycles to fully reset the prescaler and time state generators. T0 may then be brought down with the rising edge of X1. Two clock cycles later, with the rising edge of X1, the device enters into Time State 1, Phase 1. SS′ is then brought down to 5 volts 4 clocks later after T0. RESET′ is allowed to go high 5 tCY (75 clocks) later for normal execution of code. See Figure 2-15.

2.1.17 Idle Mode

Along with the standard power down, the 80C48, 80C49, 80C50 has added an IDLE mode instruction (01H) to give even further flexibility and power management. In the IDLE mode, the CPU is frozen while the oscillator, RAM, timer, and the interrupt circuitry remains fully active.

When the IDL instruction (01H) is decoded, the clock to the CPU is stopped. CPU status is preserved in its entirety: the Stack Pointer, Program Counter, Program Status Word, Accumulator, RAM, and all the registers maintain their data throughout idle.

Externally, the following occurs during idle:

1) The ports remain in the logical state they were in when idle was executed.
2) The bus remains in the logical state it was in when idle was executed if the bus was latched.

   If the bus was in a high Z condition or if external program memory if used the bus will remain in the float state.
3) ALE remains in the inactive state (low).
4) RD′, WR′, PROG′, and PSEN′ remains in the inactive state (high).
5) T0 outputs clock if enabled.

There are three ways of exiting idle. Activating any enabled interrupt (external or timer) will cause the CPU to vector to the appropriate interrupt routine. Following a RETR instruction, program execution will resume at the instruction following the address that contained the IDL instruction.

The F0 and F1 flags may be used to give an indication if the interrupt occurred during normal program execution or during idle. This is done by setting or clearing the flags before going into idle. The interrupt service routine can examine the flags and act accordingly when idle is terminated by an interrupt.

Resetting the device can also terminate idle. Since the oscillator is already running, five machine cycles are all that is required to insure proper machine operation.

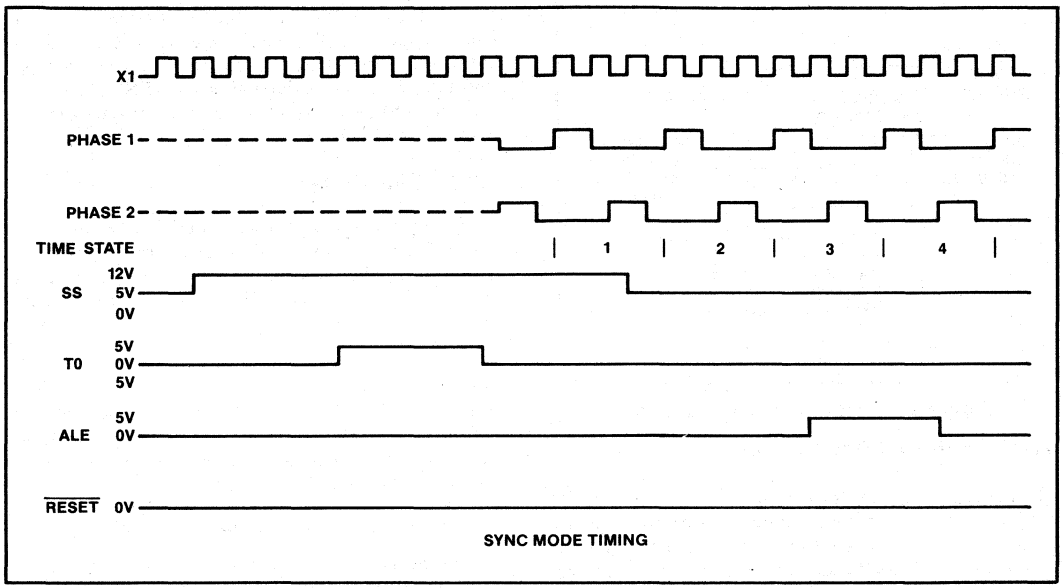


Figure 2-15. Sync Mode Timing

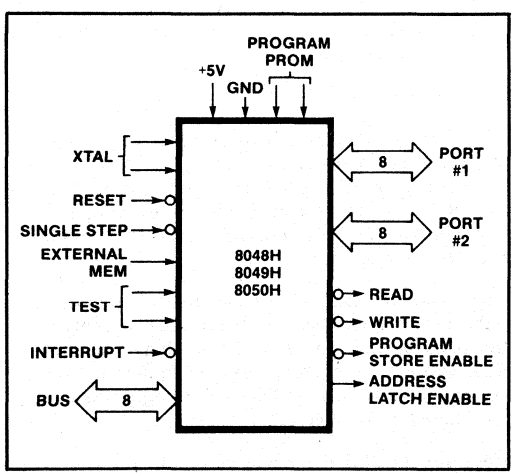


Figure 2-16. 8048AH and 8049AH Logic Symbol

2.2 PIN DESCRIPTION

The MCS-48 processors (except 8021H and 8020H) are packaged in 40 pin Dual In-Line Packages (DIP's). Table 2-3 is a summary of the functions of each pin. Figure 2-16 is the logic symbol for the 8048AH product family. Where it exists, the second paragraph describes each pin's function in an expanded MCS-48 system. Unless otherwise specified, each input is TTL compatible and each output will drive one standard TTL load.

Table 2-3

Designation	Pin Number*	Function
V_{SS}	20	Circuit GND potential
V_{DD}	26	Programming power supply; 21V during program for the 8748H/8749H; +5V during operation for both ROM and PROM. Low power standby pin in 8048AH and 8049AH/8050AH ROM versions.
V_{CC}	40	Main power supply; +5V during operation and during 8748H and 8749H programming.
PROG	25	Program pulse; +18V input pin during 8748H/8749H programming. Output strobe for 8243 I/O expander.
P10–P17 (Port 1)	27-34	8-bit quasi-bidirectional port. (Internal Pullup ≈ 50KΩ)
P20–P27 (Port 2)	21–24 35–38	8-bit quasi-bidirectional port. (Internal Pullup ≈ 50KΩ) P20–P23 contain the four high order program counter bits during an external program memory fetch and serve as a 4-bit I/O expander bus for 8243.
D0–D7 (BUS)	12–19	True bidirectional port which can be written or read synchronously using the $\overline{RD}$, $\overline{WR}$ strobes. The port can also be statically latched. Contains the 8 low order program counter bits during an external program memory fetch, and receives the addressed instruction under the control of $\overline{PSEN}$. Also contains the address and data during an external RAM data store instruction, under control of ALE, $\overline{RD}$, and $\overline{WR}$.
T0	1	Input pin testable using the conditional transfer instructions JT0 and JNT0. T0 can be designated as a clock output using ENTO CLK instruction. T0 is also used during programming and sync mode.
T1	39	Input pin testable using the JT1, and JNT1 instructions. Can be designated the event counter input using the STRT CNT instruction. (See Section 2.1.10)
$\overline{INT}$	6	Interrupt input. Initiates an interrupt if interrupt is enabled. Interrupt is disabled after a reset. (Active low) Interrupt must remain low for at least 3 machine cycles to ensure proper operation.
$\overline{RD}$	8	Output strobe activated during a BUS read. Can be used to enable data onto the BUS from an external device. (Active low) Used as a Read Strobe to External Data Memory.
$\overline{RESET}$	4	Input which is used to initialize the processor. Also used during PROM programming and verification. (Active low) (Internal pullup ≈ 80KΩ)
WR	10	Output strobe during a BUS write. (Active low) Used as write strobe to external data memory.
$\overline{ALE}$	11	Address Latch Enable. This signal occurs once during each cycle and is useful as a clock output. The negative edge of ALE strobes address into external data and program memory.

Table 2-3 (Continued)

Designation	Pin Number*	Function
$\overline{PSEN}$	9	Program Store Enable. This output occurs only during a fetch to external program memory. (Active low)
$\overline{SS}$	5	Single step input can be used in conjunction with ALE to "single step" the processor through each instruction. (Active low) (Internal pullup ≈ 300KΩ) +12V for sync modes (See 2.1.16)
EA	7	External Access input which forces all program memory fetches to reference external memory. Useful for emulation and debug, and essential for testing and program verification. (Active high) +12V for 8048AH/8049AH/8050AH program verification and +18V for 8748H/8749H program verification (Internal pullup ≈ 10MΩ on 8048AH/8049AH/8035AHL/8039AHL/8050AH/8040AHL)
XTAL1	2	One side of crystal input for internal oscillator. Also input for external source.
XTAL2	3	Other side of crystal/external source input.

*Unless otherwise stated, inputs do not have internal pullup resistors. 8048AH, 8748H, 8049AH, 8050AH, 8040AHL

2.3 PROGRAMMING, VERIFYING AND ERASING EPROM

The internal Program Memory of the 8748H and the 8749H may be erased and reprogrammed by the user as explained in the following sections. See also the 8748H and 8749H data sheets.

2.3.1 Programming/Verification

In brief, the programming process consists of: activating the program mode, applying an address, latching the address, applying data, and applying a programming pulse. This programming algorithm applies to both the 8748H and 8749H. Each word is programmed completely before moving on to the next and is followed by a verification step. The following is a list of the pins used for programming and a description of their functions:

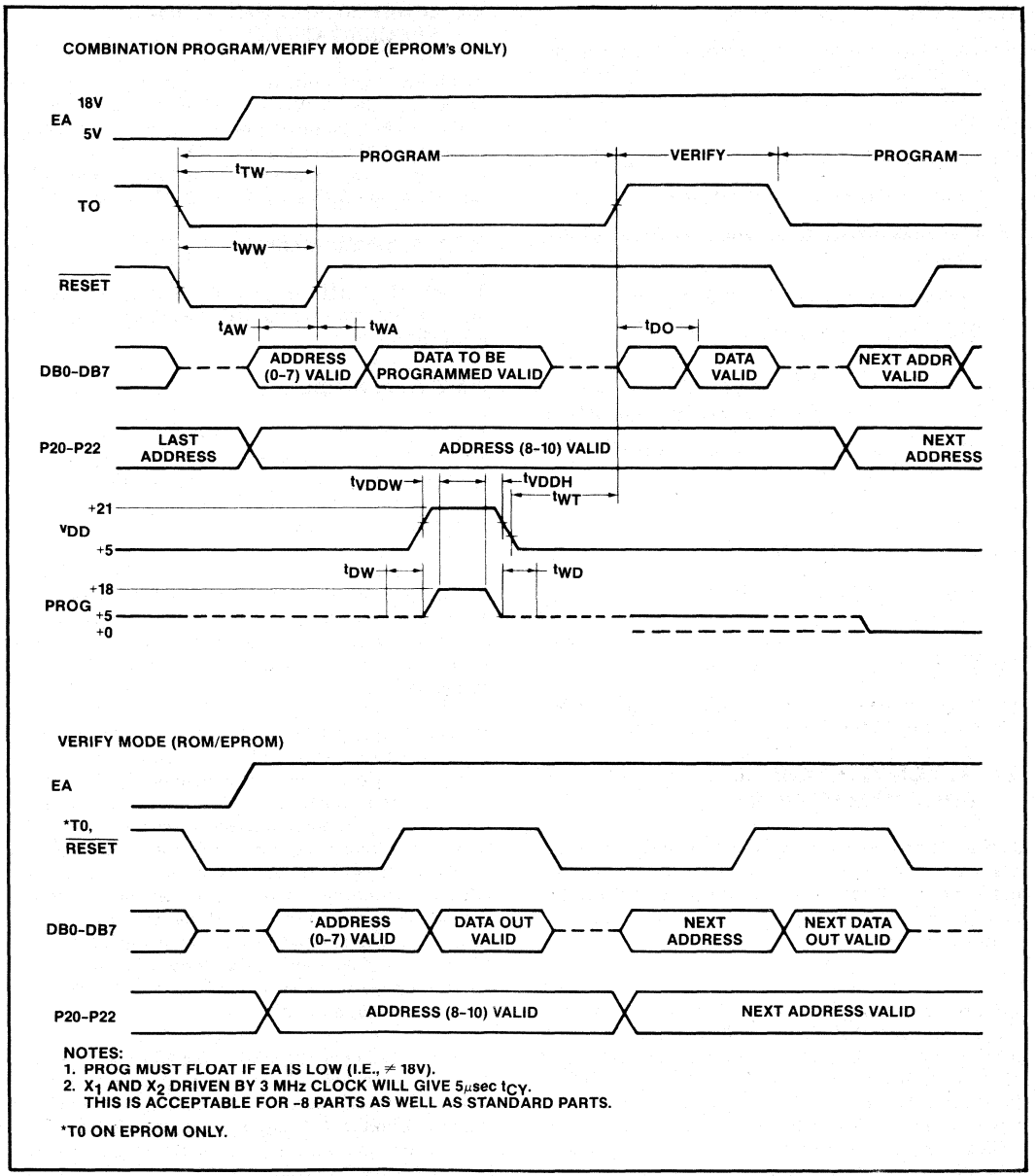
Pin	Function
XTAL 1	Clock Input (3 to 4 MHz)
Reset	Initialization and Address Latching
Test 0	Selection of Program (0V) or Verify (5V) Mode
EA	Activation of Program/Verify Modes
BUS	Address and Data Input Data Output During Verify
P20-1	Address Input for 8748H
P20-2	Address Input for 8749H
V_{DD}	Programming Power Supply
PROG	Program Pulse Input
P10-P11	Tied to ground (8749H only)

8748H AND 8749H ERASURE CHARACTERISTICS

The erasure characteristics of the 8748H and 8749H are such that erasure begins to occur when exposed to light with wavelengths shorter than approximately 4000 Angstroms (A). It should be noted that sunlight and certain types of fluorescent lamps have wavelengths in the 3000–4000A range. Data show that constant exposure to room level fluorescent lighting could erase the typical 8748H and 8749H in approximately 3 years while it would take approximately 1 week to cause erasure when exposed to direct sunlight. If the 8748H or 8749H is to be exposed to these types of lighting conditions for extended periods of time, opaque labels should be placed over the 8748H window to prevent unintentional erasure.

When erased, bits of the 8748H and 8749H Program Memory are in the logic "0" state.

The recommended erasure procedure for the 8748H and 8749H is exposure to shortwave ultraviolet light which has a wavelength of 2537 Angstroms (A). The integrated dose (i.e., UV intensity X exposure time) for erasure should be a minimum of 15W-sec/cm². The erasure time with this dosage is approximately 15 to 20 minutes using an ultraviolet lamp with a 12000μW/cm² power rating. The 8748H and 8749H should be placed within one inch from the lamp tubes during erasure. Some lamps have a filter in their tubes and this filter should be removed before erasure.



NOTES:
1. PROG MUST FLOAT IF EA IS LOW (I.E., ≠ 18V).
2. X_1 AND X_2 DRIVEN BY 3 MHz CLOCK WILL GIVE 5μsec t_{CY}.
   THIS IS ACCEPTABLE FOR -8 PARTS AS WELL AS STANDARD PARTS.

*T0 ON EPROM ONLY.

Figure 2-17. Program/Verify Sequence for 8749H/8748H

The Program/Verify sequence is:

1) $V_{CC} = V_{DD} = 5V$, applied or internal oscillator operating, $\overline{Reset}$ = 0V, Test 0 = 5V, EA = 5V, BUS and PROG floating. P10 and P11 tied to ground (8749H only)
2) Insert 8748H or 8749H in programming socket
3) Test 0 = 0V (Select Program Mode)
4) EA = 18V (Activate Program Mode for 8748H/8749H)
5) Address applied to BUS and P20–1 (2)
6) $\overline{Reset}$ = 5V (Latch Address)
7) Data applied to BUS
8) V_{DD} = 21V (Programming Power for 8748H/8749H)
9) PROG = V_{CC} followed by one 50 msec to 18V for the 8748H/8749H.
10) V_{DD} = 5V
11) TEST 0 = 5V (Verify Mode)
12) Read and Verify Data on BUS
13) TEST 0 = 0V
14) $\overline{Reset}$ = 0V and repeat from Step 5
15) Programmer should be at conditions of Step 1 when 8748H or 8749H is removed from socket.

2.4 READING INTERNAL PROGRAM MEMORY

Just as the processor may be isolated from internal program memory using EA, program memory can be read independent of the processor using the verification mode described in the previous section, Programming/Verification.

The processor is placed in the READ mode by applying a high voltage +18V for the 8749H/8748H and +12V for the 8048AH/8049AH/8050AH) to the EA pin and +5V to the T0 (8748H and 8749H only) input pin. $\overline{RESET}$ must be at 0V when voltage is applied to EA. The address of the location to be read is then applied to the BUS and Port 2. All twelve address bits must be driven. The address is latched by a "0" to "1" transition on $\overline{RESET}$ and a high level on $\overline{RESET}$ causes the contents of the program memory location addressed to appear on the eight lines of BUS. $\overline{RESET}$ must be brought back to 0V before leaving the READ mode.

2.5 8020H/8021H FUNCTIONAL SPECIFICATIONS

The following is a functional description of the major elements of the 8020H and 8021H. It is important to understand at the outset, however, that the 8020H is itself an 8021H but in a 20-pin DIP. This significant pin reduction is accomplished by bonding out only 13 of the 8021H's 21 I/O lines: Port 00–07 and Port 13–17.

2.5.1 Program Memory

The 8020H/8021H contains 1K x 8 of mask programmable ROM. No external ROM expansion capability is provided.

2.5.2 Data Memory

A 64 x 8 dynamic RAM is located on chip for data storage. All locations are indirectly addressable and eight designated locations are directly addressable. Also, included in the memory is the address stack, addressed by a 3-bit stack pointer.

Memory is organized as shown in Figure 2-19. The least significant 8 addresses, 0–7, are directly addressable by any of the 11 direct register instructions. The locations are readily accessible for a variety of operations with the least number of instruction bytes required for their manipulation.

Registers 0 and 1 have another function, in that they can be used to indirectly address all locations in memory, using the indirect register instructions. These indirect RAM address registers, IRAR's, are especially useful for repetitive-type operations on adjacent memory locations. The indirect register instruction specifies which IRAR to use, and the contents of the IRAR is used to address a location in RAM. The contents of the addressed location is used during the execution of the instruction and may be modified. A value larger than 63 should not be present in the IRAR when selected by an indirect register instruction. IRAR's may point to addresses 0–7, if desired.

Locations 8–23 may be used as the address stack. The address stack enables the processor to keep track of the return addresses generated from CALL instructions. A 3-bit stack pointer (SP) supplies the address of the locations to be loaded with the next return address generated. The SP to this pushdown stack is incremented by one after a return address is stored, and decremented by one before an address is fetched during a RET. The unincremented program counter address is stored in the address stack. Before loading the program counter

during a return from the subroutine (RET), the stack contents are incremented. A total of 8 levels of nesting is possible. The SP is initialized to location 8 upon RESET. Since each address is 10-bits long, two bytes must be used to store a single address. The SP is incremented and decremented by one, but each increment and decrement moves the address pointed to by two. Therefore, only even numbered addresses are pointed to. If a particular application does not require 8 levels of nesting, the unused portion of the stack may be used as any other indirectly addressable scratchpad location. For example, if only 3 levels of subroutine nesting are used, then only locations 8–13 need be reserved for the address stack, and locations 14–63 can be used for data storage.

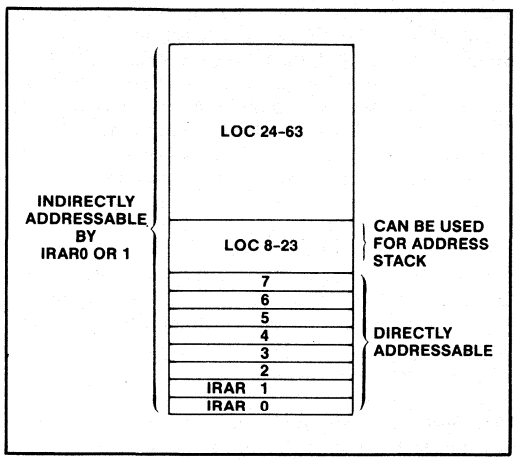


Figure 2-19. Internal RAM Organization

2.5.3 Oscillator and Clock

The 8020H/8021H contains its own on board oscillator and clock circuit, requiring only an external timing control element across pins XTAL1 and XTAL2. The control element can be a clock in, crystal, inductor, or a single resistor. The capacitor normally required with inductor and resistor timing control operations is not necessary with the 8020H/8021H. This capacitor has been integrated directly on chip.

The external timing control element determines all internal time divisions. An instruction cycle consists of 10 states, and each state is a time division of 3 oscillator periods (see Figure 2-20). Therefore, to obtain a 10 μsec instruction cycle, a 3 MHz oscillator frequency is required. This may be obtained with a direct 3 MHz clock in, a 3 MHz crystal, or a 330 μH inductor. A 15K resistor will also yield an oscillator frequency in the range of 3 to 3.6 MHz. Note, however, that the required inductance and resistance may vary and should be adjusted as necessary. The resistance and inductance are inversely proportional to the generated oscillator frequency.

The 8020H/8021H utilizes dynamic RAM and certain other dynamic logic. Due to the clocking required with dynamic circuits, the oscillator frequency must be equal to or greater than 600 kHz, or improper operation may occur.

2.5.4 Timer/Event Counter

The 8020H/8021H has internal timer/event counter circuits that can monitor elapsed time or count external events that occur during program execution. The circuit

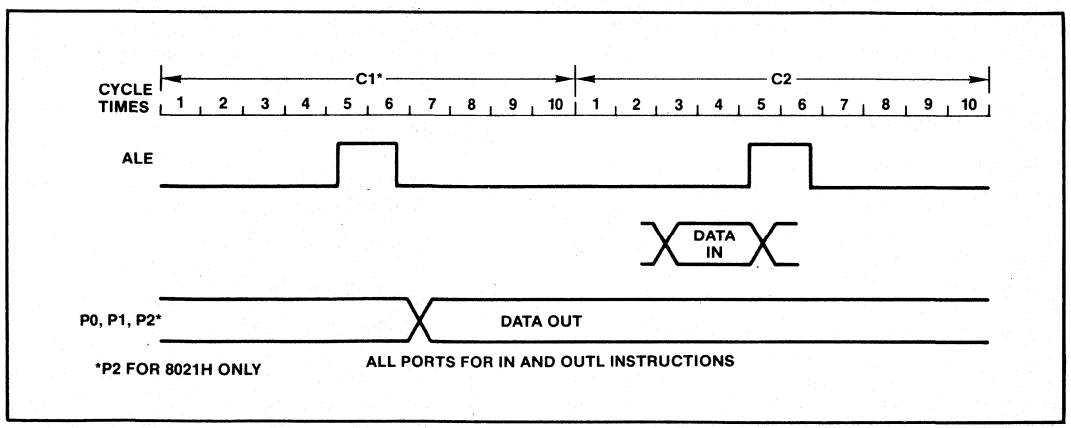


Figure 2-20. 8020H/8021H Timing Diagram

has an 8-bit binary up-counter that is presettable and readable with two MOV instructions. These instructions transfer the contents of the accumulator to the counter and vice versa. The counter is cleared by Reset and can be set by the MOV T,A instruction. The counter is stopped by a RESET or STOP TCNT instruction and remains stopped until started as a timer by a STRT T instruction or as an event counter by a STRT CNT instruction. Once started, the counter increments to its maximum count (FF), and overflows to zero. The count continues until stopped by a STOP TCNT instruction or RESET. The increment from maximum count to zero (overflow) sets an overflow flag. The state of the overflow flag is testable with the conditional jump instruction JTF. The flag is reset by JTF but not by executing a RESET, unlike the 8748H.

At the STRT T command an internal prescaler is zeroed and thereafter increments once each 30 input clocks (once each single cycle instruction, twice each double cycle instruction). The prescaler is a divide by 32. At the (11111) to (00000) transition the timer is incremented. The timer is 8-bits and an overflow (FFH) to (00H) timer flag is set. A conditional branch instruction (JTF) is available for testing this flag, the flag being reset each test. Total count capacity for the timer is $2^8 \times 2^5 = 8192$ or 81.9 msec at a 10 μsec cycle time. Contents of the timer can be moved to the accumulator by the MOV A,T instruction without disturbing the counting process.

Execution of a START CNT instruction connects the T1 input pin to the counter input and enables the counter. The T1 input is sampled at the beginning of state 3. Subsequent high to low transitions on T1 will cause the counter to increment. T1 must be held for at least $1T_{cy}$ to ensure it will not be missed. The maximum rate at which the counter may be incremented is once per three instruction cycles (every 30 μs for a 3 MHz oscillator)—there is no minimum rate. T1 input must remain stable for at least $1/5T_{cy}$ after each transition.

2.5.5 Input/Output Capabilities

The 8020H/8021H I/O configurations are highly flexible. A number of different configurations are possible, tailoring an 8020H/8021H to a given task. Other than the power supply and dedicated pins, all other pins can be used for input, output, or both, depending on the configuration.

All ports are quasi-bidirectional to facilitate stand-alone use. A simplified schematic of the quasi-bidirectional interface is shown in Figure 2-21. This configuration allows buffered outputs, and also allows external input. Data written to these ports is statically latched and remains unchanged until rewritten. As input ports these

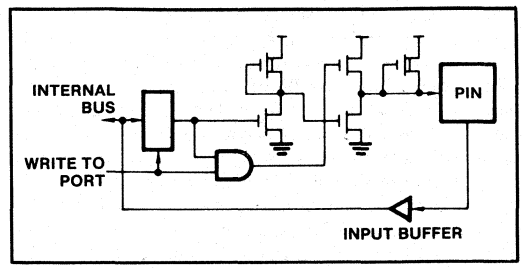


Figure 2-21. Quasi-Bidirectional Port Structure

lines are non-latching, i.e., inputs must be present until read by an input instruction. When writing a "0" or low value to these ports, the low impedance pulldown device sinks an external TTL load. When writing a "1", a large current is supplied through the low impedance pullup device to allow a fast data transfer. After a short time (less than one instruction cycle), the low impedance device is shut off and the high impedance pullup maintains the "1" level indefinitely. However, in this situation, an input device capable of overriding the small amount of sustaining current supplied by the pullup device can be read. (Alternatively, the data written can be read.) So, by writing a "1" to any particular pin, that pin can serve either as a true high-level latched output pin, or as just a pullup resistor on an input. This allows maximum user flexibility in selecting his input or latched output pins, with a minimum of external components.

Port 00–07 is also quasi-bidirectional, except there is no low impedance pullup device. As outputs, this port is essentially open drain.

By mask option the high impedance pullup devices on P00–P07 may be deleted on any pin providing a true open drain output. This is useful in driving analog circuits and certain loads, such as keyboards.

T1 INPUT

The 8020H/8021H T1 input line can be used as an input for the following functions:

- Event Counter (external input)
- Test input for branch instructions
- Zero voltage crossing detection

The operation of T1 as an input to the Event Counter is described in the Timer/Event Counter section. When used as a test input, the JT1 and JNT1 instructions test for 1 and 0 levels, respectively.

The T1 pin can also be used to detect the zero crossing of slowly moving AC signals (60 Hz). The self-biasing circuit shown in Figure 2-22 permits the Test 1 input to detect when the input voltage crosses zero within ±5%; the voltage is then coupled through a 1.0 μf capacitor. Maximum input voltage is 3V peak-to-peak. The zero cross detection is especially useful in SCR control of 60 Hz power and in developing time-of-day and other timing routines. As a ROM mask option there is a pullup device that is useful for switch contact input or standard TTL.

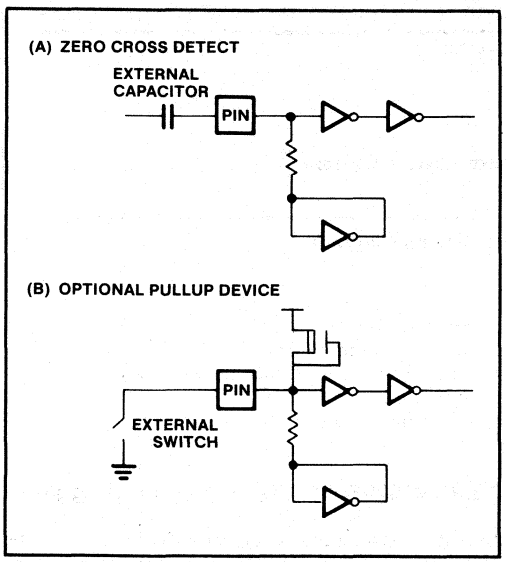


Figure 2-22. Test 1 Pin

HIGH CURRENT OUTPUTS

High current drive is desirable for minimizing external parts required to do high power control. P10 and P11, 8021H only, have been designated high drive outputs capable of sinking 7mA to V_{SS} + 2.5 volts. (For clarity, this is 7mA to V_{SS} with a 2.5 volt drop across the buffer.) These pins may, of course, be paralleled for 14mA drive if the output logic states are always the same.

EXPANDED I/O

The 8021H, not the 8020H however, can be used with the 8243 I/O expander chip, which provides additional I/O capability with a limited number of overhead pins. This chip has 4 directly addressable 4-bit ports. It connects to the PROG pin, which provides a clock, and pins P20–P23, which provide address and data. These ports can be written with a MOVD P,A; ANLD P,A; and ORLD P,A for Ports 4–7. A high to low transition on PROG signifies that address and control are available on P20–P23. The previous data on P20–P23 before an output expander instruction is lost. Therefore, when using an output expander P20–P23 are not useful for general input/output. Reading is via the MOVD A,P. This circuit configuration is shown in Figure 2-23. The timing diagram is shown in Figure 2-24.

Standard TTL can also be used to expand the number of I/O lines on the 8021H as well as the 8020H.

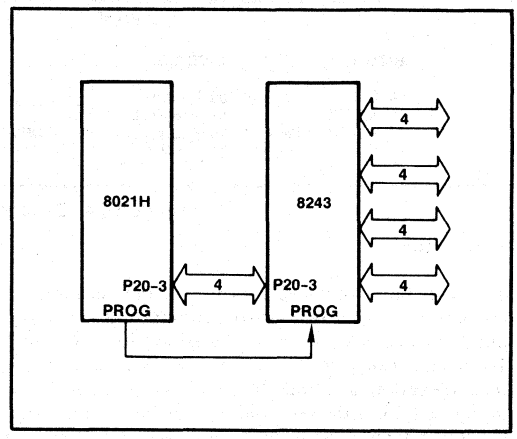


Figure 2-23. I/O Expander Interface

2.5.6 CPU

The 8020H/8021H CPU has arithmetic and logical capability. A wide variety of arithmetic and logic instructions may be exercised, which affect the contents of the accumulator and/or direct or indirect scratchpad locations. Provisions have been made for simplified BCD arithmetic capability using the DAA, SWAP A and XCHD instructions. In addition, MOVP A,@A allows table lookup for display formating and constants. The conditional branch logic within the processor enables several conditions internal and external to the processor to be tested by the users program. Use the conditional jump instructions with the tests listed below to effect a change in the program execution sequence:

Test	Jump Condition	Jump Instructions
Accumulator	A = 0 A ≠ 0	JZ JNZ
Carry Flag	0 1	,JC
Timer Overflow Flag	— 1	JTF
Test Input-T1	0 1	JNT1, JT1

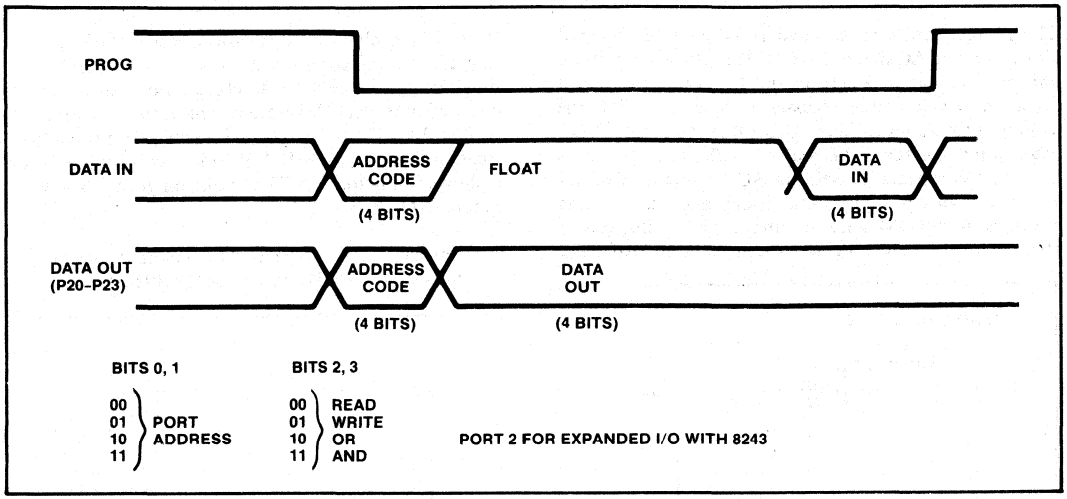


Figure 2-24. Expanded I/O Timing Diagram

2.5.7 Reset

The reset input provides a means for initialization for the processor. This Schmitt-trigger input has an internal pulldown device which in combination with an external 1 μfd capacitor provides an internal reset pulse of sufficient length to guarantee all circuitry is reset, as shown in figure 2-25. If the reset pulse is generated externally, the RESET pin must be held above 3.8V for at least 10 milliseconds after the power supply is within tolerance. Only 3 machine cycles (25 μs @ 3.6MHz) are required if power is already on and the oscillator has stabilized.

Reset performs the following functions:

1) Sets program counter to zero
2) Sets stack pointer to zero
3) Sets Ports to input mode
4) Disables interrupts (timer and external)
5) Stops timer
6) Clears timer register

2.5.8 8020H/8021H Testing and Debugging

To facilitate testing and debug, certain test modes may be activated in the 8020H/8021H by raising combinations of RESET, TEST1 and PROG to 15 volts. Internal ROM is dumped out sequentially for verification. External memory operation is used for CPU checkout.

Reset	Prog	Test 1	Case	Function
5V	X	X		Power On Clear
0V	X	X		Normal Operation
15V	15V	15V	Mode 1a	On each cycle internal ROM is dumped to Port 0—Sequentially after ALE leading edge.
15V	15V	⎍	Mode 1b	On every TEST 1 falling edge the program counter increments, dumps internal ROM to Port 0.
0V	15V	X	Mode 2	Chip will operate from external memory (one page) via Port 0. ALE strobes Address out, memory in.
15V	X	X	Mode 3	Chip accepts op codes into Port 1. Allows Port 0 and 8243 testing.

NOTE
X = Normal mode—between 0V and V_{CC} Test 1 in Mode 1b should be limited to V_{CC} Mode 3 8243 testing for 8021H.

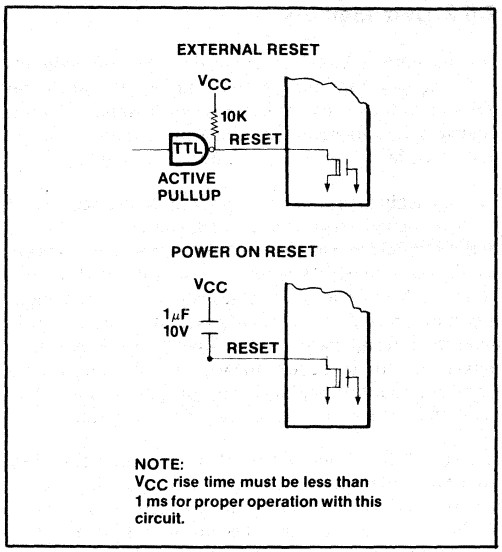


Figure 2-25.

2.5.9 Differences Between the 8021H and the 8748H

Although the 8021H is basically an electrical and functional subset of the 8748H, there are some differences:

1) *Pin Out*—As the 8021H is a 28-pin DIP, some form of interface must be used with the 8748H to simulate the 8021H.

2) *Instruction Time*—The 8021H instruction cycle is 30 clock cycles long, the 8748H instruction cycle is 15 clock cycles long. Where exact timing is important, the 8748H breadboard part should be operated at half the 8021H clock rate.

3) *Test 1*—To facilitate developing time of day routines from 60Hz, and for SCR control, the Test 1 pin without the pullup resistor option will detect zero crossing of a capacitively coupled AC input.

4) *Quasi-Bidirectional Ports*—All 8021H ports are quasi-bidirectional to facilitate stand-alone use. Port 0 has open drain outputs and by mask option it may or may not have pullup devices.

5) *Oscillator*—The 8021H has an on-chip oscillator that is optimized for the single resistor mode. External connection will differ from the 8748H.

6) *Dynamic RAM and Logic*—The 8021H utilizes dynamic RAM and some dynamic logic. Input clocking must be maintained above the minimum rate or improper operation may result.

7) *High Current Outputs*—High current drive is desirable for minimizing external parts required to do high power control. P10 and P11 have been designated high drive outputs capable of sinking 7mA at V_{SS} + 2.5 volts. (For clarity, this is 7mA to V_{SS} with a 2.5 volt drop across the buffer.) These pins may, of course, be paralleled for 14mA drive if the output logic states are always the same.

8) *Timer/Counter*—The 8748H does not increment its timer in the second cycle of a 2-cycle instruction; the 8021H does.

9) *Reset*—Reset has been modified on the 8021H to active high; the 8748H is active low.

10) *Instruction Set*—The instructions below, which are found in the 8748H, have been deleted from the 8021H instruction set:

Data Moves	Registers	Branch	Timer	Control	Input/Output
MOV A,PSW	DEC R	JT0 addr	EN TCNTI	EN I	ANL P,#data
MOV PSW,A	**Flags**	JNT0 addr	DIS TCNTI	DIS I	ORL P,#data
MOVX A,@R		JF0 addr	**Subroutine**	SEL RB0	INS A,BUS*
MOVX @R,A	CLR F0	JF1 addr		SEL RB1	OUTL BUS,A*
MOVP3 A,@A	CPL F0	JNI addr	RETR	SEL MB0	ANL BUS,#data
	CLR F1	JBb addr		SEL MB1	ORL BUS,#data
	CPL F1			ENT0 CLK	

*These instructions have been replaced in the 8021H by IN A,P0 and OUTL P0,A, respectively.

2.6 8022 FUNCTIONAL SPECIFICATIONS

The 8022's architecture is based upon the 8021H, and many functions of the two parts are identical.

2.6.1 Program Memory

The 8022 program memory consists of 2048 words 8 bits wide which are addressed by the program counter. The memory is ROM which is mask programmable at the factory. (See Figure 2-26.) No external ROM expansion capability is provided. There are three locations in program memory of special importance.

Location 0: Activating the RESET line of the processor causes the first instruction to be fetched from location 0.

Location 3: Activating the interrupt input line (T0) of the processor (if interrupt is enabled) causes a jump to subroutine.

Location 7: A timer/event counter interrupt resulting from a timer/counter overflow causes a jump to subroutine (if timer/counter interrupt is enabled).

Therefore, the first instruction to be executed after initialization is stored in location 0. The first word of an external interrupt service routine is stored in location 3, and the first word of a timer/event counter interrupt service routine is stored in location 7.

Program memory can be used to store constants as well as program instructions. The MOVP instruction allows easy table lookup for constants and display formatting.

2.6.2 Data Memory

On-chip data memory is organized as 64 words eight bits wide. All locations are indirectly addressable and eight designated locations are directly addressable. Also included in the data memory is the program counter stack, addressed by a 3-bit stack pointer. (See Figure 2-27.)

The first eight locations (0–7) of the array are designated as working registers and are directly addressable by any of the 11 direct register instructions. These locations are readily accessible for a variety of operations with a minimum number of instruction bytes required for their manipulation. Thus, they are usually used to store frequently accessed intermediate results. The DJNZ instruction makes very efficient use of the working registers as program loop counters by allowing the programmer to decrement and test the register in a single instruction.

Registers 0 and 1 have yet another function in that they can be used to indirectly address all locations in the data memory using the indirect register instructions. These two RAM pointer registers are especially useful for repetitive type operations on adjacent memory locations. The indirect register instruction specifies which register is used to address a location in RAM. The contents of the addressed location are used during the execution of the instruction and may be modified. The pointer registers may also point to registers 0–7, if desired.

Locations 8–23 serve a dual role in that they contain the 8-level program counter stack, two RAM locations per level. The program counter stack enables the processor to keep track of the return addresses generated by interrupts or CALL instructions by storing the contents of the program counter prior to servicing the subroutine.

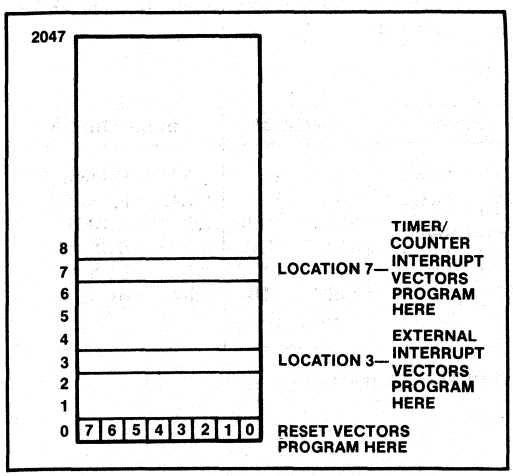


Figure 2-26. Program Memory Map

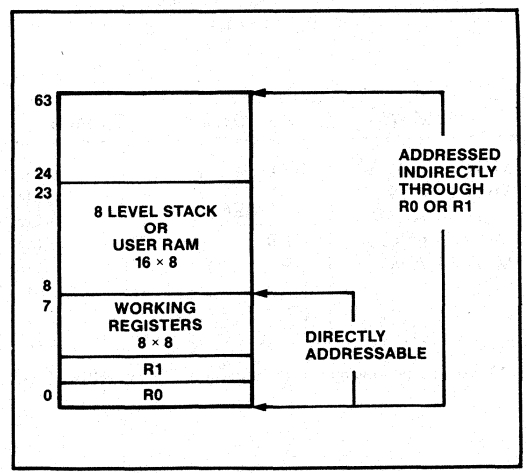


Figure 2-27. Data Memory Map

A 3-bit stack pointer determines which of the program counter stack's eight register pairs will be loaded with the next return address generated. The stack pointer, when initialized to 000 by RESET, points to RAM locations 8 and 9. The first subroutine CALL or interrupt results in the program counter contents being transferred to locations 8 and 9. The stack pointer is then incremented by one and points to locations 10 and 11 in anticipation of another CALL. The end of a subroutine, which is signaled by a return instruction (RET or RETI), causes the stack pointer to be decremented and the contents of the resulting register pair to be transferred to the program counter.

Unlike the 8048AH, in the 8022 the unincremented program counter address is stored in the address stack. The stack contents are then incremented before being loaded into the program counter during a return (RET) from subroutine. However, during a return (RETI) from interrupt, the stack contents are loaded directly into the program counter. This difference makes it imperative to use only RETI's to return from interrupts, and RET's to return from subroutines.

Since the program counter's addresses are 11 bits long, two bytes or registers must be used to store a single address. Thus, the 16-byte program counter stack permits up to a total of 8 levels of subroutine nesting without overflowing the stack. If overflow does occur, the deepest address stored (locations 8 and 9) will be overwritten and lost since the stack pointer overflows from 111 to 000. It also underflows from 000 to 111. If a particular application does not require 8 levels of nesting, the unused portion of the program counter stack may be used as any other indirectly addressable RAM location. For example, if only 3 levels of subroutine nesting are used, then only locations 8–13 need be reserved for the program counter stack, and locations 14–23 can be used for data storage.

2.6.3 Oscillator and Clock

The 8022 contains its own on-board oscillator and clock circuit, requiring only an external timing control element. See Figure 2-28. This control element can be a crystal, inductor, or clock. All internal time slots are derived from the external element, and all outputs are a function of the oscillator frequency. An instruction cycle consists of 10 states, and each state is a time slot of 3 oscillator periods. Therefore, to obtain a 10 μs instruction cycle, a 3 MHz crystal should be used. The minimum instruction cycle time of 8.38 μsec corresponds to a 3.58 MHz crystal.

2.6.4 Timer/Event Counter

Like the other MCS-48 microcomputers, the 8022 has an internal timer/event counter. This circuit can monitor elapsed time or count external events that occur during program execution. See the 8021H description, Section 2.5.4, for a complete explanation.

2.6.5 Input/Output Capabilities

The 8022 has 26 lines which can be used for digital input or output functions. These lines are organized as 3 ports of 8 lines, each of which serve as either inputs, outputs, or bi-directional ports, and 2 test inputs which can alter program sequences when tested by conditional jump instructions.

Ports 1 and 2 have identical operating characteristics and are both quasi-bidirectional. That is, each line may serve as an input, an output, or both. Data written to these ports is statically latched and remains unchanged until rewritten. As inputs, these lines are non-latching, i.e., inputs must be present until read by an input instruction. Inputs are fully TTL compatible and all outputs will drive at least one standard TTL load. See Section 2.1.4 for a more complete description of the quasi-bidirectional structure.

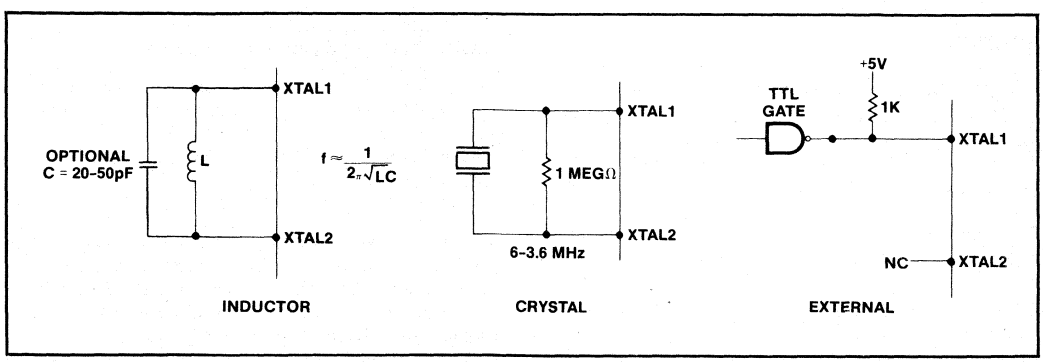


Figure 2-28. Frequency Reference Options

PORT 0 COMPARATOR INPUTS

Port 0 has been modified from the standard quasi-bidirectional structure to allow an optional open drain configuration with comparator inputs. The low impedance pullup device has been eliminated and the high impedance pullup is optional. Thus, the user can choose via a mask programmable selection each line of port 0 to be either quasi-bidirectional with a high impedance or true open drain. The open drain configuration allows the line to sink current through the low impedance pulldown device or to float in the high output state. More importantly, the open drain configuration makes port 0 very easy to drive when it is used as inputs. The input circuitry for each line of port 0 includes a voltage comparator which amplifies the voltage difference between the input port line and the port 0 threshold reference pin (V_{TH}). The voltage gain of the comparator is sufficient to sense a 100 mV input differential within the range V_{SS} to $V_{CC}/2$.

If V_{TH} is allowed to float, it will bias itself to the digital switch point of the other ports, and port 0 behaves as a set of normal digital inputs. However, by biasing V_{TH}, the switch point can be both tightly controlled and adjusted. Common uses for this would include high noise margin inputs ($V_{CC}/2$), unusual logic level inputs as from a diode isolated keyboard, analog channel expansion, and direct capacitive touchpanel interface. The comparator action is automatic and the port is read just as any other port.

HIGH CURRENT OUTPUTS

High current drive is desirable for minimizing external parts required to do high power control. P10 and P11 have been designated high drive outputs capable of sinking 7mA at V_{SS} + 2.5 volts. (For clarity, this is 7mA to V_{SS} with a 2.5 volt drop across the buffer.) These pins may, of course, be paralleled for 14mA drive if the output logic states are always the same.

EXPANDED I/O

In addition to the 26 digital I/O lines contained on-board the 8022, a user can obtain additional I/O lines by utilizing the Intel 8243 I/O expander chip or standard TTL. The 8243 interfaces to 4 port lines of the 8022 (lower half of port 2) and is strobed by the PROG line of the 8022.

The interface procedure is exactly the same as with the 8021H.

2.6.6 Test and Interrupt Inputs

In addition to the 24 general purpose I/O lines which comprise ports 0, 1, and 2, the 8022 has two inputs which are testable via conditional jump instructions, T0 and T1. These pins allow inputs to cause program branches without the necessity to load an input port into the accumulator. T0 and T1 have other functions as well.

The Test 0 pin serves as an external interrupt input as well as a testable input. An interrupt sequence is initiated by applying a low "0" level input to the T0 pin when external interrupt is enabled. Interrupt is level triggered and active low to allow "WIRE ORING" of several interrupt sources at the input pin. When an interrupt is detected, it causes a "jump to subroutine" at location 3 in program memory as soon as all other cycles of the current instruction are complete. At this time, the program counter contents are saved in the program counter stack, but the remaining status of the processor is not. Unlike the 8048AH, the 8022 does not contain a program status word. Thus, when appropriate, the carry and auxiliary carry flags are saved in software, as is the accumulator. The routine shown below saves the accumulator and the carry flags in only four bytes.

Instructions	Bytes	Comments
MOV R6,A	1	;save accumulator
CLR A	1	;clear accumulator
DA A	1	;convert carry flags into sixes
MOV R7,A	1	;save status of carry flags

The end of an interrupt service subroutine is marked by the execution of a Return from Interrupt instruction (RETI). Prior to returning from the interrupt subroutine, however, the status of the accumulator and the carry flags are restored in software. The following routine restores the status of the accumulator and the carry flags, which was previously saved, in five bytes.

Instructions	Bytes	Comments
MOV A,R7	1	;restore carry flags status to
Add A,#0AAH	2	;accumulator and set/clear carry flags
MOV A,R6	1	;restore accumulator
RETI	1	;return

The interrupt system is single level in that once an interrupt is detected, all further interrupt requests are ignored until execution of a RETI re-enables the interrupt input logic. This sequence holds true also for an internal interrupt generated by timer overflow. If an external interrupt and an internal timer/counter generated interrupt are detected at the same time, the external source will be

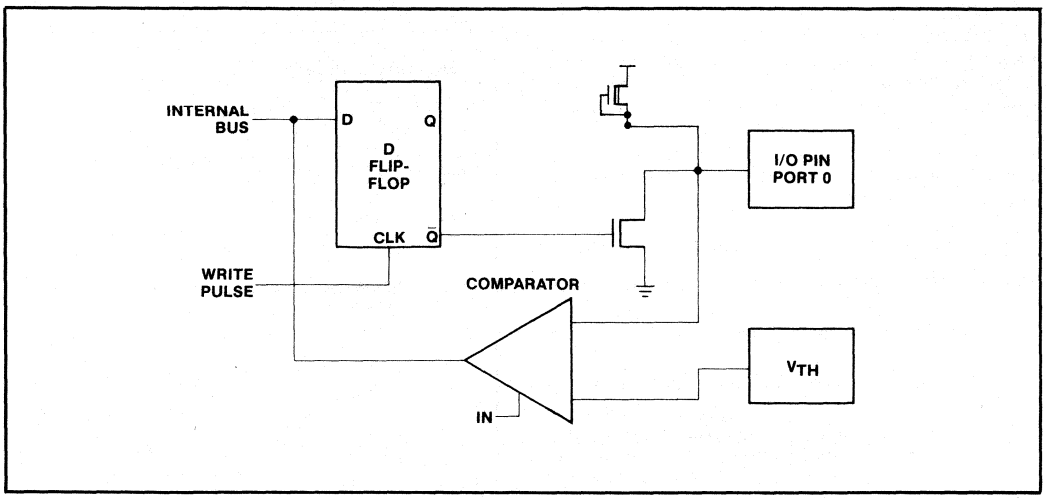


Figure 2-29. Port 0 I/O Structure

recognized. If needed, a second external interrupt can be created by enabling the timer/counter interrupt, loading FFH in the counter (one less than terminal count) and enabling the event counter mode. A low-to-high transition on the T1 input will then cause an interrupt vector to location 7.

The Test 1 pin, in addition to being a testable input, serves two other important functions. It can be used as an input pin to the external event counter, as previously mentioned, and it can be used to detect the zero crossing point of slow moving AC signals. Execution of the STRT CNT instruction puts the T1 pin in the counter input mode by connecting T1 to the counter and enabling the counter. Subsequent low-to-high transitions on T1 will cause the counter to increment. Note that this operation differs from the rest of the MCS-48 devices, which increment the counter on high-to-low transitions. This change was made on the 8022 to take advantage of the accuracy of the rising edge detection on the zero cross circuitry. The maximum rate at which the counter may be incremented is once per three instruction cycles (every 30 μs when using a 3 MHz crystal)—there is no minimum frequency.

In addition to serving as a testable input and as the counter input, the T1 pin has special circuitry to detect when an AC signal crosses its average DC level. When driven directly, this pin responds as a normal digital input. To utilize the zero cross detection mode, an AC signal of approximately 1-3 VAC p-p magnitude and a maximum frequency of 1 kHz is coupled through an external capacitor (1 μF) to the T1 pin.

The internal digital state is sensed as a zero until the rising edge crosses the DC average level, when it becomes a one. This is accomplished by the self-biasing high gain amplifier which is included in the T1 input. This circuit biases the T1 input exactly at its switching point, such that a small change will cause a digital transition to occur. This digital transition takes place within 5 degrees of the zero point. The digital value of T1 remains a one until the falling edge of the AC inputs drops approximately 100 mV below the switching point of the rising edge (100 mV below the zero point, if the digital transition occurred exactly at the zero point). The 100 mV offset is created by hysteresis and eliminates chattering of the internal signal caused by the external noise.

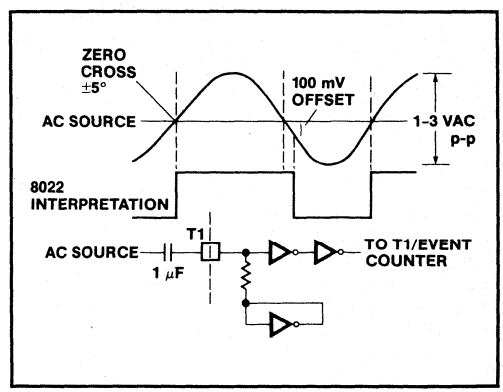


Figure 2-30. Zero-Cross Detection

The zero cross detection capability allows the user to make the 60 Hz power signal the basis for this system timing. All timing routines, including time-of-day, can be implemented using the zero cross detection capability of T1 and its conditional jump instructions. In addition, the zero cross detection feature can be used in conjunction with the timer interrupt to interrupt processing at the zero voltage point. This enables the user to control voltage phase sensitive devices such as triacs and SCRs, and to use the 8022 in applications such as shaft speed and angle measurement.

2.6.7 Analog to Digital Converter

The 8022 contains on-chip a complete hardware implementation of an 8-bit analog to digital (A/D) converter with two multiplexed analog inputs. The A/D converter utilizes a successive approximation technique to provide an updated conversion once every four instruction cycles with a minimum of required software.

The A/D converter consists of four main parts, the input circuitry, a series string of resistors, a voltage comparator, and the successive approximation logic. The two analog inputs are multiplexed on-chip and selected via software by the SEL AN0 and SEL AN1 instructions. Besides selecting one of the analog inputs, these instructions restart the conversion sequence which operates continuously. Restarting a conversion sequence deletes the conversion in progress but does not affect the result of the previous conversion which is stored in the conversion result register. The continuous operation of the A/D converter saves program space and time by allowing the user to obtain multiple readings from a given input with only one select instruction. To obtain a valid conversion reading, the user must provide the analog input signal no later than the beginning of the select instruction cycle. The analog input is then sampled by the A/D converter and maintained internally. This voltage becomes one input to the voltage comparator which amplifies the difference between the analog input and the voltage tap on the series resistor string.

The series resistor string is connected between the A/D reference pin (V_{AREF}) and ground (AV_{SS}). It is comprised of 256 identical resistors which divide the voltage between these two pins into 256 identical voltage steps. This configuration gives the converter its inherent monotonicity. The range of V_{AREF} in which full 8-bit resolution can be provided is between $V_{CC}/2$ and V_{CC}.

Thus, the user is given a minimum voltage range from ground to $V_{CC}/2$ and a maximum range from ground to V_{CC} over which 8-bit resolution is insured.

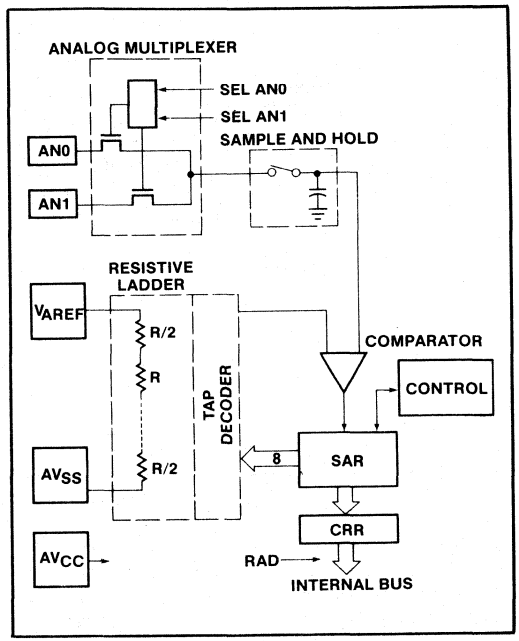


Figure 2-31. A/D Converter Block Diagram

The voltage tap on the series resistor string is selected by the resistor ladder decoder. This decoder is driven by the 8-bit successive approximation register (SAR). Each bit of the SAR is set in succession, MSB to LSB, and a voltage comparison between the selected resistor ladder voltage and the analog input voltage is performed after the setting of each bit. The result of each comparison determines whether the particular bit will remain set or be reset. All comparisons are performed automatically by the on-chip A/D hardware. At the end of 8 comparisons the SAR contains a valid digital result which is then latched into the conversion result register (CRR). The RAD instruction (read (A/D) loads the conversion result from the CRR to the accumulator of the 8022.

As mentioned previously, the software and time required to perform an A/D conversion is optimized by the 8022's on-chip A/D converter configuration. Typical software for reading two sequential A/D conversions and storing them in data memory is shown below:

```
First       SEL AN0     ;Starts conversion of AN0 input
Conversion  MOV R0,#24  ;Set up memory pointer
50μs        RAD         ;First conversion value to
4 bytes                  accumulator
Second      MOV@R0,A    ;Store first conversion value
Conversion  INC R0      ;Increment memory location
40μs        RAD         ;second conversion value to
3 bytes                  accumulator
```

Note that the second conversion occurs without a second select instruction being used. Rather, the continuous operation of the A/D converter provides an updated digital value 4 instruction cycles after the first.

To insure maximum accuracy from the A/D converter, separate power supply pins (AV_{CC} and AV_{SS}) and a substrate pin (SUBST) have been provided. Supplying the power supply pins with a well filtered and regulated voltage supply minimizes the effect of power supply variance and system noise. The substrate pin should be bypassed to ground through a 500 pF to 0.001 μF capacitor.

2.6.8 CPU

The 8022 CPU has arithmetic and logical capability. There is a wide variety of arithmetic and logic instructions which affect the contents of the accumulator, and/or direct or indirect scratchpad locations. Provisions have been made for simplified BCD arithmetic capability using the DAA, SWAP A, and XCHD instructions. In addition, MOVP A,@A allows table lookup for display formating and constants. The conditional branch logic within the processor enables several conditions internal and external to the processor to be tested by the user's program. Use the conditional jump instructions with the tests listed below to effect a change in the program execution sequence.

Test	Jump Condition	Jump Instructions
Accumulator	A = 0 A ≠ 0	JZ JNZ
Carry Flag	0 1	JNC,JC
Timer Overflow Flag	— 1	JTF
Test Input-T1	0 1	JNT1, JT1
Test Input-T0	0 1	JNT0, JT0

2.6.9 8022 Testing and Debugging

To facilitate testing and debug, certain test modes may be activated in the 8022 by raising combinations of RESET, TEST1 and PROG to 15 volts. Internal ROM is dumped out sequentially for verification. External memory operation is used for CPU checkout.

Reset	Prog	Test 1	Case	Function
5V	X	X		Power On Clear
0V	X	X		Normal Operation
15V	15V	15V	Mode 1a	On each cycle internal ROM is dumped to Port 0—Sequentially after ALE leading edge.
15V	15V		Mode 1b	On every TEST 1 falling edge the program counter increments, dumps internal ROM to Port 0.
0V	15V	X	Mode 2	Chip will operate from external memory (one page) via Port 0. ALE strobes Address out, memory in.
15V	X	X	Mode 3	Chip accepts op codes into Port 1. Allows Port 0 and 8243 testing.

NOTE
X = Normal mode—between 0V and V_{CC}
Test 1 in Mode 1b should be limited to V_{CC}

Expanded MCS®-48 System 3

CHAPTER 3
EXPANDED MCS®-48 SYSTEM

3.0 INTRODUCTION

If the capabilities resident on the single-chip 8048AH/8748H/8035AHL/8049AH/8749H/8039AHL are not sufficient for your system requirements, special on-board circuitry allows the addition of a wide variety of external memory, I/O, or special peripherals you may require. The processors can be directly and simply expanded in the following areas:

- Program Memory to 4K words
- Data Memory to 320 words (384 words with 8049AH)
- I/O by unlimited amount
- Special Functions using 8080/8085AH peripherals

By using bank switching techniques, maximum capability is essentially unlimited. Bank switching is discussed later in the chapter. Expansion is accomplished in two ways:

1) Expander I/O —A special I/O Expander circuit, the 8243, provides for the addition of four 4-bit Input/Output ports with the sacrifice of only the lower half (4-bits) of port 2 for inter-device communication. Multiple 8243's may be added to this 4-bit bus by generating the required "chip select" lines.

2) Standard 8085 Bus —One port of the 8048AH/8049AH is like the 8-bit bidirectional data bus of the 8085 microcomputer system allowing interface to the numerous standard memories and peripherals of the MCS®-80/85 microcomputer family.

MCS-48 systems can be configured using either or both of these expansion features to optimize system capabilities to the application.

Both expander devices and standard memories and peripherals can be added in virtually any number and combination required.

3.1 EXPANSION OF PROGRAM MEMORY

Program Memory is expanded beyond the resident 1K or 2K words by using the 8085 BUS feature of the MCS®-48. All program memory fetches from the addresses less than 1024 on the 8048AH and less than 2048 on the 8049AH occur internally with no external signals being generated (except ALE which is always present). At address 1024 on the 8048AH, the processor automatically initiates external program memory fetches.

3.1.1 Instruction Fetch Cycle (External)

As shown in Figure 3-1, for all instruction fetches from addresses of 1024 (2048) or greater, the following will occur:

1) The contents of the 12-bit program counter will be output on BUS and the lower half of port 2.

2) Address Latch Enable (ALE) will indicate the time at which address is valid. The trailing edge of ALE is used to latch the address externally.

3) Program Store Enable ($\overline{\text{PSEN}}$) indicates that an external instruction fetch is in progress and serves to enable the external memory device.

4) BUS reverts to input (floating) mode and the processor accepts its 8-bit contents as an instruction word.

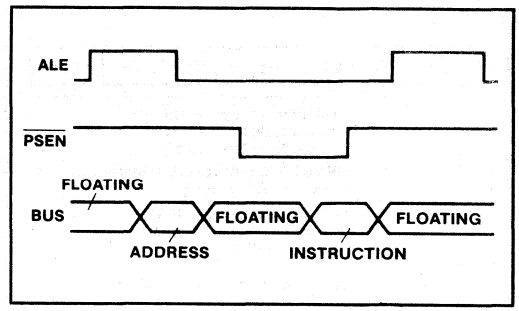


Figure 3-1. Instruction Fetch from External Program Memory

All instruction fetches, including internal addresses, can be forced to be external by activating the EA pin of the 8048AH/8049AH. The 8035AHL/8039AHL processors without program memory always operate in the external program memory mode (EA = 5V).

3.1.2 Extended Program Memory Addressing (Beyond 2K)

For programs of 2K words or less, the 8048AH/8049AH addresses program memory in the conventional manner. Addresses beyond 2047 can be reached by executing a program memory bank switch instruction (SEL MB0, SEL MB1) followed by a branch instruction (JMP or CALL). The bank switch feature extends the range of branch instructions beyond their normal 2K range and at the same time prevents the user from inadvertently crossing the 2K boundary.

PROGRAM MEMORY BANK SWITCH

The switching of 2K program memory banks is accomplished by directly setting or resetting the most significant bit of the program counter (bit 11); see Figure 3-2. Bit 11 is not altered by normal incrementing of the program counter but is loaded with the contents of a special flip-flop each time a JMP or CALL instruction is executed. This special flip-flop is set by executing an SEL MB1

instruction and reset by SEL MB0. Therefore, the SEL MB instruction may be executed at any time prior to the actual bank switch which occurs during the next branch instruction encountered. Since all twelve bits of the program counter, including bit 11, are stored in the stack, when a Call is executed, the user may jump to subroutines across the 2K boundary and the proper bank will be restored upon return. However, the bank switch flip-flop will not be altered on return.

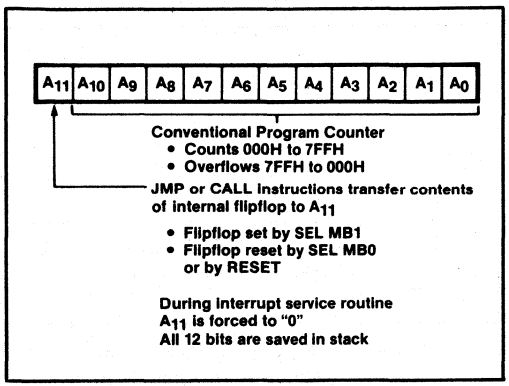


Figure 3-2. Program Counter

INTERRUPT ROUTINES

Interrupts always vector the program counter to location 3 or 7 in the **first** 2K bank, and bit 11 of the program counter is held at "0" during the interrupt service routine. The end of the service routine is signalled by the execution of an RETR instruction. Interrupt service routines should therefore be contained entirely in the lower 2K words of program memory. The execution of a SEL MB0 or SEL MB1 instruction within an interrupt routine is not recommended since it will not alter PC11 while in the routine, but will change the internal flip-flop.

3.1.3 Restoring I/O Port Information

Although the lower half of Port 2 is used to output the four most significant bits of address during an external program memory fetch, the I/O information is still outputed during certain portions of each machine cycle. I/O information is always present on Port 2's lower 4 bits at the rising edge of ALE and can be sampled or latched at this time.

3.1.4 Expansion Examples

Shown in Figure 3-3 is the addition of 2K words of program memory using an 2716A 2K x 8 ROM to give a total of 3K words of program memory. In this case no chip select decoding is required and $\overline{PSEN}$ enables the memory directly through the chip select input. If the system requires only 2K of program memory, the same configuration can be used with an 8035AHL substituted for the 8048AH. The 8049AH would provide 4K of program memory with the same configuration.

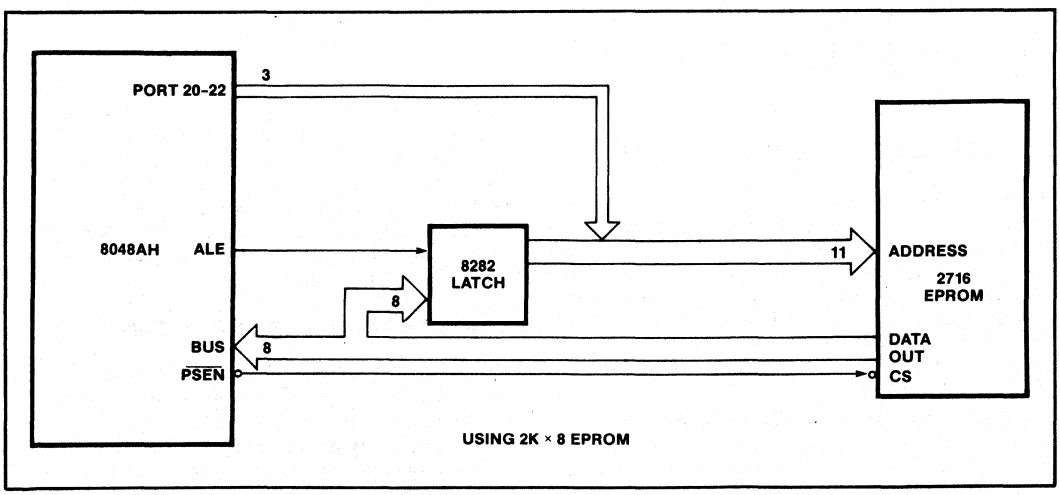


Figure 3-3. Expanding MCS®-48 Program Memory Using Standard Memory Products

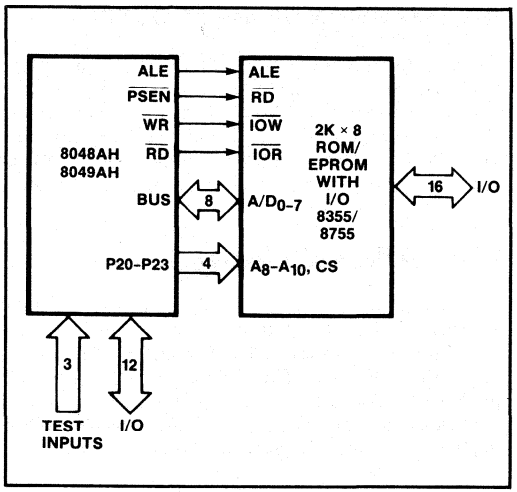


Figure 3-4. External Program Memory Interface

Figure 3-4 shows how the 8755/8355 EPROM/ROM with I/O interfaces directly to the 8048AH without the need for an address latch. The 8755/8355 contains an internal 8-bit address latch eliminating the need for an 8212 latch. In addition to a 2K x 8 program memory, the 8755/8355 also contains 16 I/O lines addressable as two 8-bit ports. These ports are addressed as external RAM; therefore the RD and WR outputs of the 8048AH are required. See the following section on data memory expansion for more detail. The subsequent section on I/O expansion explains the operation of the 16 I/O lines.

3.2 EXPANSION OF DATA MEMORY

Data Memory is expanded beyond the resident 64 words by using the 8085AH type bus feature of the MCS®-48.

3.2.1 Read/Write Cycle

All address and data is transferred over the 8 lines of BUS. As shown in Figure 3-5, a read or write cycle occurs as follows:

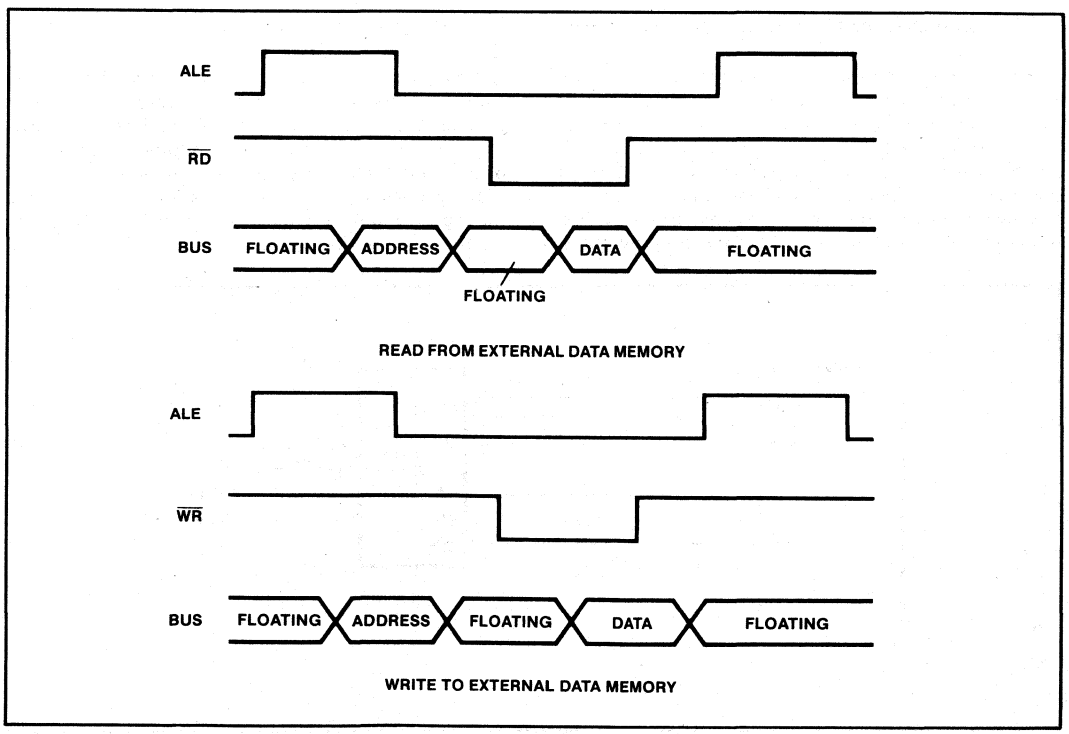


Figure 3-5. External Data Memory Timings

1) The contents of register R0 or R1 is outputed on BUS.
2) Address Latch Enable (ALE) indicates address is valid. The trailing edge of ALE is used to latch the address externally.
3) A read ($\overline{RD}$) or write ($\overline{WR}$) pulse on the corresponding output pins of the 8048AH indicates the type of data memory access in progress. Output data is valid at the trailing edge of $\overline{WR}$ and input data must be valid at the trailing edge of $\overline{RD}$.
4) Data (8 bits) is transferred in or out over BUS.

3.2.2 Addressing External Data Memory

External Data Memory is accessed with its own two-cycle move instructions, MOVX A, @R and MOVX @R, A, which transfer 8 bits of data between the accumulator and the external memory location addressed by the contents of one of the RAM Pointer Registers R0 and R1. This allows 256 locations to be addressed in addition to the resident locations. Additional pages may be added by "bank switching" with extra output lines of the 8048AH.

3.2.3 Examples of Data Memory Expansion

Figure 3-6 shows how the 8048AH can be expanded using the 8155 memory and I/O expanding device. Since the 8155 has an internal 8-bit address latch, it can interface directly to the 8048AH without the use of an external latch. The 8155 provides an additional 256 words of static data memory and also includes 22 I/O lines and a 14-bit timer. See the following section on I/O expansion and the 8155 data sheet for more details on these additional features.

3.3 EXPANSION OF INPUT/OUTPUT

There are four possible modes of I/O expansion with the 8048AH: one using a special low-cost expander, the 8243; another using standard MCS-80/85 I/O devices; and a third using the combination memory/I/O expander devices the 8155, 8355, and 8755. It is also possible to expand using standard TTL devices as shown in Chapter 5.

3.3.1 I/O Expander Device

The most efficient means of I/O expansion for small systems is the 8243 I/O Expander Device which requires only 4 port lines (lower half of Port 2) for communication with the 8048AH. The 8243 contains four 4-bit I/O ports which serve as an extension of the on-chip I/O and are addressed as ports #4–7 (see Figure 3-7). The following operations may be performed on these ports:

- Transfer Accumulator to Port
- Transfer Port to Accumulator
- AND Accumulator to Port
- OR Accumulator to Port

A 4-bit transfer from a port to the lower half of the Accumulator sets the most significant four bits to zero. All communication between the 8048AH and the 8243 occurs over Port 2 lower (P20–P23) with timing provided by an output pulse on the PROG pin of the processor. Each transfer consists of two 4-bit nibbles: The first containing the "op code" and port address, and the second containing the actual 4 bits of data.

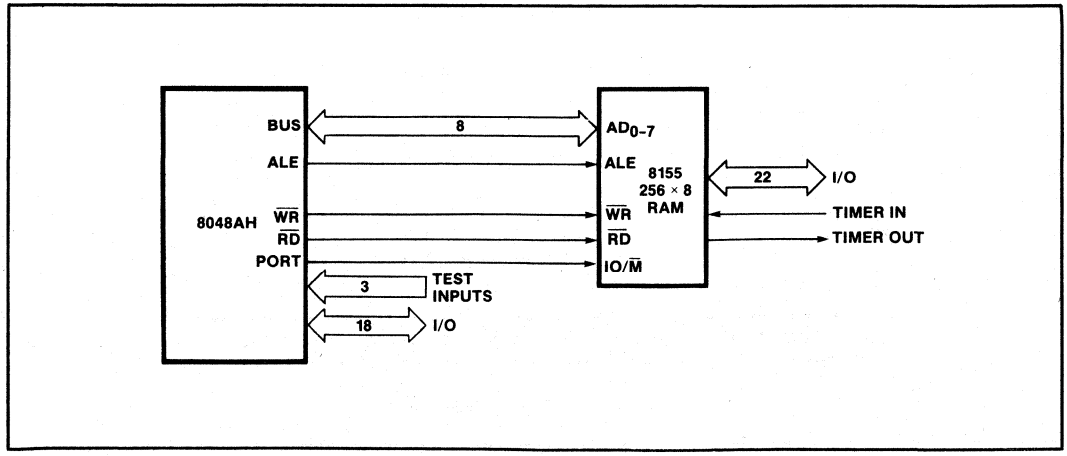


Figure 3-6. 8048AH Interface to 256 × 8 Standard Memories

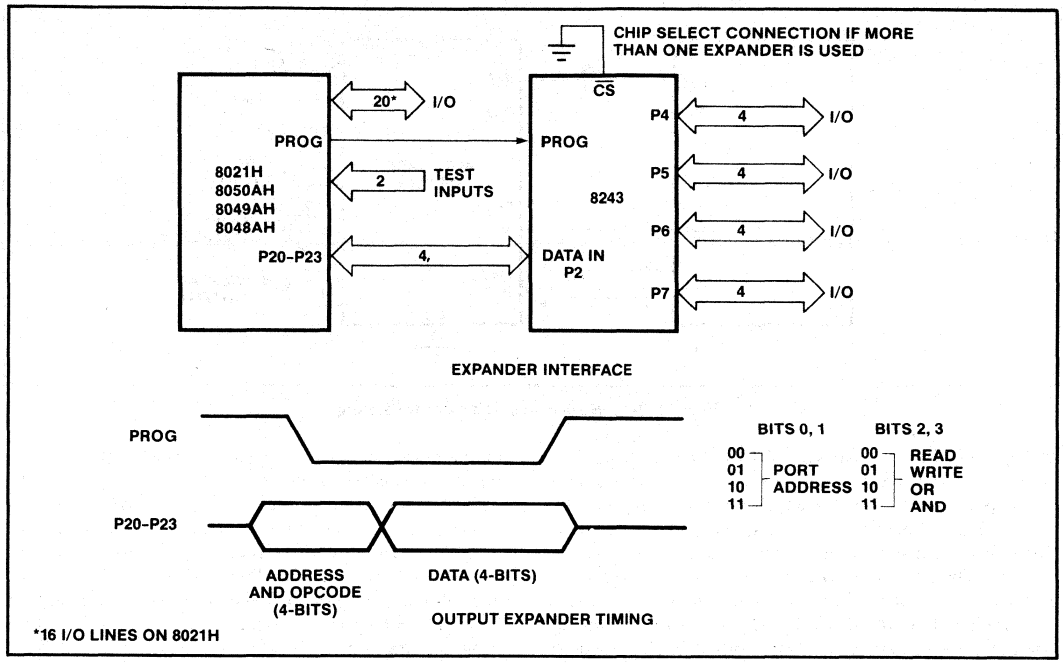
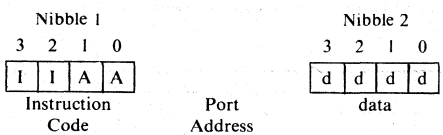


Figure 3-7. 8243 Expander I/O Interface



II	AA
00 Read	00 — Port #4
01 Write	01 — Port #5
10 OR	10 — Port #6
11 AND	11 — Port #7

A high to low transition of the PROG line indicates that address is present, while a low to high transition indicates the presence of data. Additional 8243's may be added to the four-bit bus and chip selected using additional output lines from the 8048AH/8748H.

I/O PORT CHARACTERISTICS

Each of the four 4-bit ports of the 8243 can serve as either input or output and can provide high drive capability in both the high and low state.

3.3.2 I/O Expansion with Standard Peripherals

Standard MCS-80/85 type I/O devices may be added to the MCS®-48 using the same bus and timing used for Data Memory expansion. Figure 3-8 shows an example of how an 8048AH can be connected to an MCS-85 peripheral. I/O devices reside on the Data Memory bus and in the data memory address space and are accessed with the same MOVX instructions. (See the previous section on data memory expansion for a description of timing.) The following are a few of the Standard MCS-80 devices which are very useful in MCS®-48 systems:

- 8214 Priority Interrupt Encoder
- 8251 Serial Communications Interface
- 8255 General Purpose Programmable I/O
- 8279 Keyboard/Display Interface
- 8253 Interval Timer

3.3.3 Combination Memory and I/O Expanders

As mentioned in the sections on program and data memory expansion, the 8355/8755 and 8155 expanders also contain I/O capability.

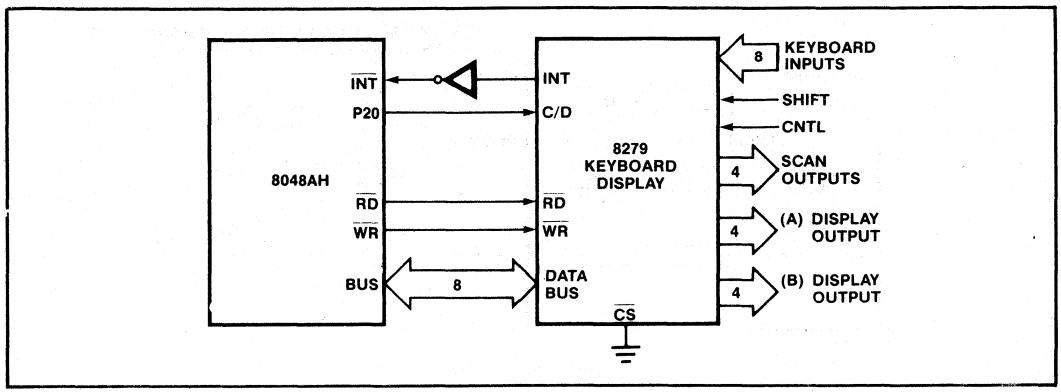


Figure 3-8. Keyboard/Display Interface

8355/8755: These two parts of ROM and EPROM equivalents and therefore contain the same I/O structure. I/O consists of two 8-bit ports which normally reside in the external data memory address space and are accessed with MOVX instructions. Associated with each port is an 8-bit Data Direction Register which defines each bit in the port as either an input or an output. The data direction registers are directly addressable, thereby allowing the user to define under software control each individual bit of the ports as either input or output. All outputs are statically latched and double buffered. Inputs are not latched.

8155/8156: I/O on the 8155/8156 is configured as two 8-bit programmable I/O ports and one 6-bit programmable port. These three registers and a Control/Status register are accessible as external data memory with the MOVX instructions. The contents of the control register determines the mode of the three ports. The ports can be programmed as input or output with or without associated handshake communication lines. In the handshake mode, lines of the six-bit port become input and output strobes for the two 8-bit ports. Also included in the 8155 is a 14-bit programmable timer. The clock input to the timer and the timer overflow output are available on external pins. The timer can be programmed to stop on terminal count or to continuously reload itself. A square wave or pulse output on terminal count can also be specified.

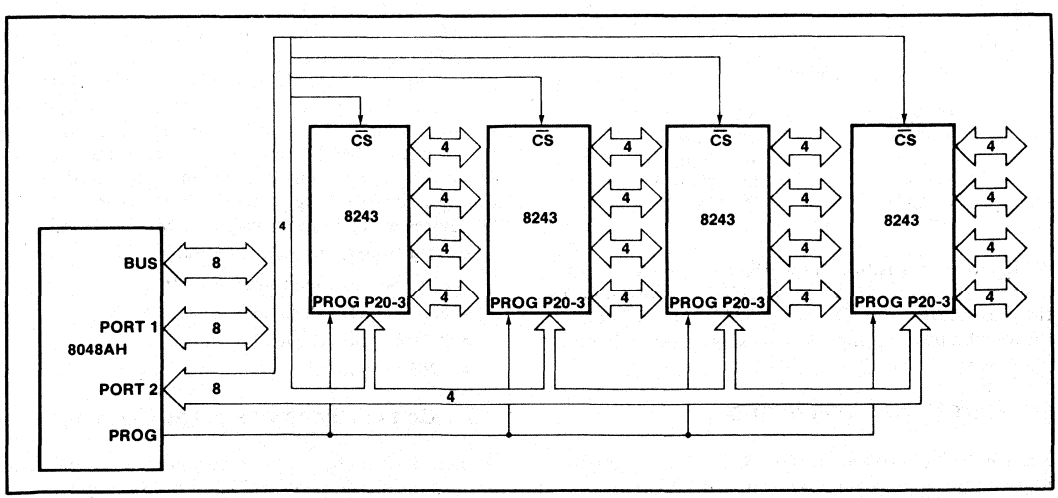


Figure 3-9. Low Cost I/O Expansion

I/O EXPANSION EXAMPLES (SEE ALSO CHAPTER 5)

Figure 3-9 shows the expansion of I/O using multiple 8243's. The only difference from a single 8243 system is the addition of chip selects provided by additional 8048AH output lines. Two output lines and a decoder could also be used to address the four chips. Large numbers of 8243's would require a chip select decoder chip such as the 8205 to save I/O pins.

Figure 3-10 shows the 8048AH interface to a standard MCS®-80 peripheral; in this case, the 8255 Programmable Peripheral Interface, a 40-pin part which provides three 8-bit programmable I/O ports. The 8255 bus interface is typical of programmable MCS®-80 peripherals with an 8-bit bidirectional data bus, a RD and WR input for Read/Write control, a CS (chip select) input used to enable the Read/Write control logic and the address inputs used to select various internal registers.

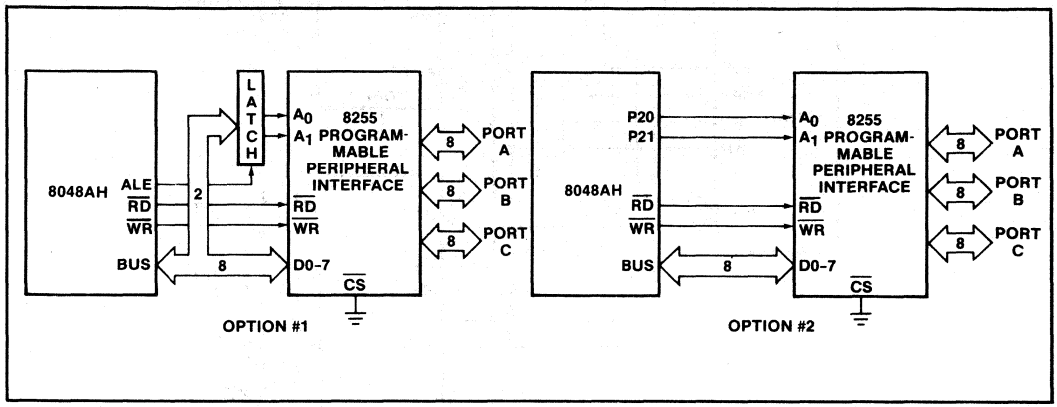


Figure 3-10. Interface to MCS®-80 Peripherals

Interconnection to the 8048AH is very straightforward with BUS, $\overline{RD}$, and $\overline{WR}$ connecting directly to the corresponding pins on the 8255. The only design consideration is the way in which the internal registers of the 8255 are to be addressed. If the registers are to be addressed as external data memory using the MOVX instructions, the appropriate number of address bits (in this case, 2) must be latched on BUS using ALE as described in the section on external data memories. If only a single device is connected to BUS, the 8255 may be continuously selected by grounding $\overline{CS}$. If multiple 8255's are used, additional address bits can be latched and used as chip selects.

A second addressing method eliminates external latches and chip select decoders by using output port lines as address and chip select lines directly. This method, of course, requires the setting of an output port with address information prior to executing a MOVX instruction.

3.4 MULTI-CHIP MCS®-48 SYSTEMS

Figure 3-11 shows the addition of two memory expanders to the 8048AH, one 8355/8755 ROM and one 8156 RAM. The main consideration in designing such a system is the addressing of the various memories and I/O ports. Note that in this configuration address lines A_{10} and A_{11} have been ORed to chip select the 8355. This ensures that the chip is active for all external program memory fetches in the 1K to 3K range and is disabled for all other addresses. This gating has been added to allow the I/O port of the 8355 to be used. If the chip was left selected all the time, there would be conflict between these ports and the RAM and I/O of the 8156. The NOR gate could be eliminated and A_{11} connected directly to the CE (instead of $\overline{CE}$) input of the 8355; however, this would create a 1K word "hole" in the program memory by causing the 8355 to be active in the 2K and 4K range instead of the normal 1K to 3K range.

In this system the various locations are addressed as follows:

- Data RAM —Addresses 0 to 255 when Port 2 Bit 0 has been previously set = 1 and Bit 1 set = 0
- RAM I/O —Addresses 0 to 3 when Port 2 Bit 0 = 1 and Bit 1 = 1
- ROM I/O —Addresses 0 to 3 when Port 2 Bit 2 or Bit 3 = 1

See the memory map in Figure 3-12.

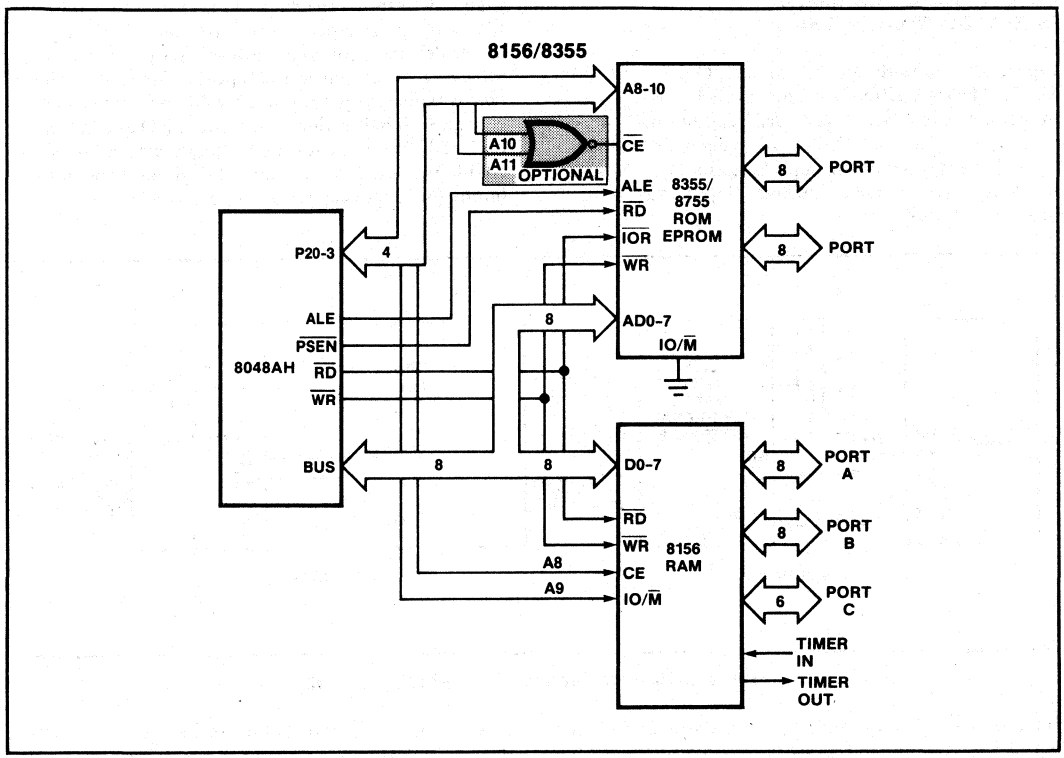


Figure 3-11. The Three-Component MCS®-48 System

3.5 MEMORY BANK SWITCHING

Certain systems may require more than the 4K words of program memory which are directly addressable by the program counter or more than the 256 data memory and I/O locations directly addressable by the pointer registers R0 and R1. These systems can be achieved using "bank switching" techniques. Bank switching is merely the selection of various blocks or "banks" of memory using dedicated output port lines from the processor. In the case of the 8048AH, program memory is selected in blocks of 4K words at a time, while data memory and I/O are enabled 256 words at a time.

The most important consideration in implementing two or more banks is the software required to cross the bank boundaries. Each crossing of the boundary requires that the processor first write a control bit to an output port before accessing memory or I/O in the new bank. If program memory is being switched, programs should be organized to keep boundary crossings to a minimum. Jumping to subroutines across the boundary should be avoided when possible since the programmer must keep track of which bank to return to after completion of the subroutine. If these subroutines are to be nested and accessed from either bank, a software "stack" should be implemented to save the bank switch bit just as if it were another bit of the program counter.

From a hardware standpoint bank switching is very straightforward and involves only the connection of an I/O line or lines as bank enable signals. These enables are ANDed with normal memory and I/O chip select signals to activate the proper bank.

3.6 CONTROL SIGNAL SUMMARY

Table 3-1 summarizes the instructions which activate the various control outputs of the MCS®-48 processors. During all other instructions these outputs are driven to the inactive state.

Table 3-1. MCS®-48 Control Signals

Control Signal	When Active
$\overline{RD}$	During MOVX A, @R or INS Bus
$\overline{WR}$	During MOVX @R, A or OUTL Bus
ALE	Every Machine Cycle
$\overline{PSEN}$	During Fetch of external program memory (instruction or immediate data)
PROG	During MOVD A,P ANLD P,A MOVD P,A ORLD P,A

3.7 PORT CHARACTERISTICS

3.7.1 BUS Port Operations

The BUS port can operate in three different modes: as a latched I/O port, as a bidirectional bus port, or as a program memory address output when external memory is used. The BUS port lines are either active high, active low, or high impedance (floating).

The latched mode (INS, OUTL) is intended for use in the single-chip configuration where BUS is not being used as an expander port. OUTL and MOVX instructions can be mixed if necessary. However, a previously latched output will be destroyed by executing a MOVX instruction and BUS will be left in the high impedance state. INS does not put the BUS in a high impedance state. Therefore, the use of MOVX after OUTL to put the BUS in a high impedance state is necessary before an INS instruction intended to read an external word (as opposed to the previously latched value).

OUTL should never be used in a system with external program memory, since latching BUS can cause the next instruction, if external, to be fetched improperly.

3.7.2 Port 2 Operations

The lower half of Port 2 can be used in three different ways: as a quasi-bidirectional static port, as an 8243 expander port, and to address external program memory.

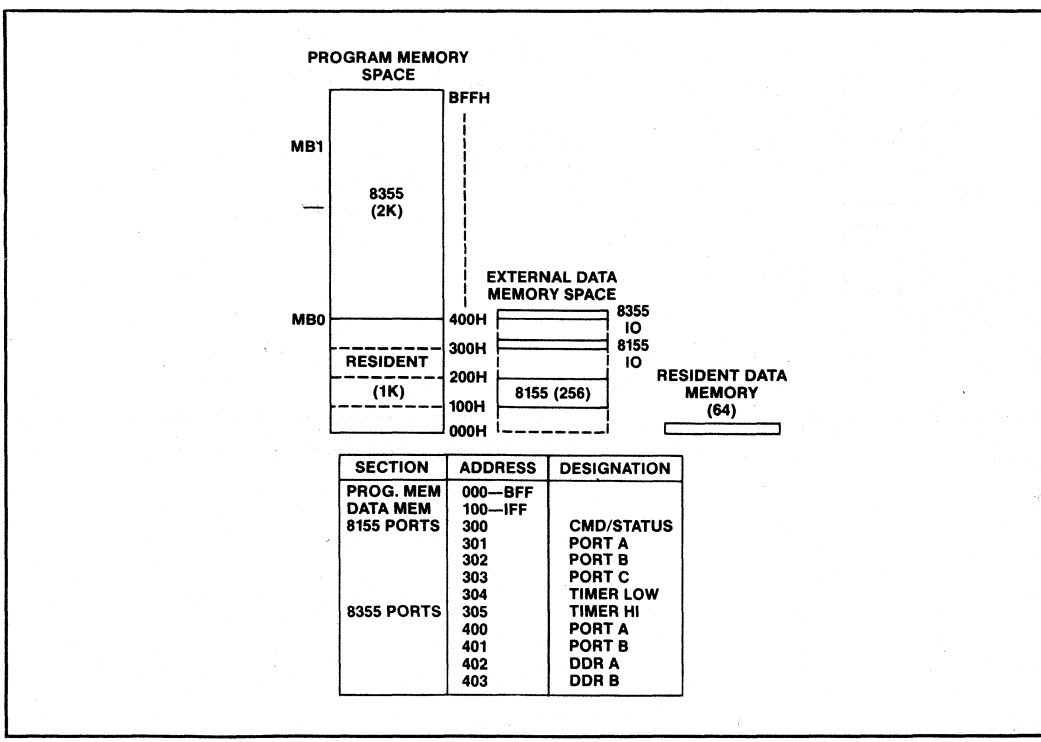


Figure 3-12. Memory Map for Three-Component MCS®-48 Family

In all cases outputs are driven low by an active device and driven high momentarily by a low impedance device and held high by a high impedance device to VCC.

The port may contain latched I/O data prior to its use in another mode without affecting operation of either. If lower Port 2 (P20–3) is used to output address for an external program memory fetch, the I/O information previously latched will be automatically removed temporarily while address is present, then restored when the fetch is complete. However, if lower Port 2 is used to communicate with an 8243, previously latched I/O information will be removed and not restored. After an input from the 8243, P20-3 will be left in the input mode (floating). After an output to the 8243, P20-3 will contain the value written, ANDed, or ORed to the 8243 port.

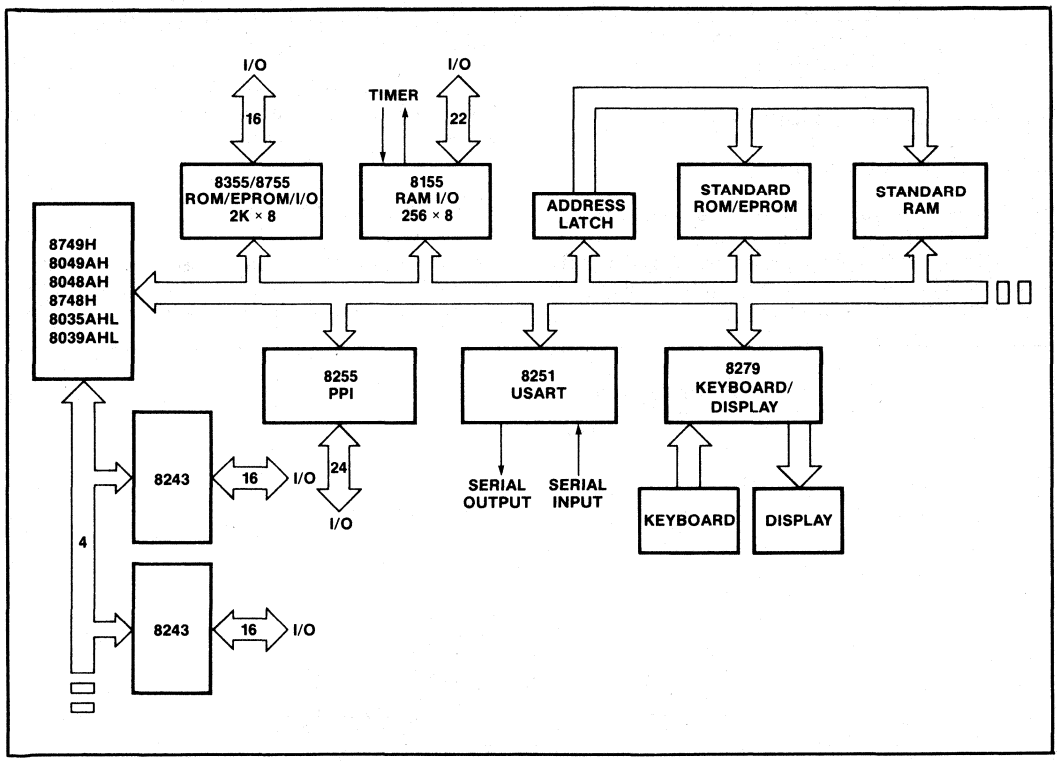


Figure 3-13. MCS®-48 Expansion Capability

MCS®-48 Instruction Set 4

CHAPTER 4
MCS®-48 INSTRUCTION SET

4.0 INTRODUCTION

The MCS®-48 instruction set is extensive for a machine of its size and has been tailored to be straightforward and very efficient in its use of program memory. All instructions are either one or two bytes in length and over 80% are only one byte long. Also, all instructions execute in either one or two cycles and over 50% of all instructions execute in a single cycle. Double cycle instructions include all immediate instructions, and all I/O instructions.

The MCS-48 microcomputers have been designed to handle arithmetic operations efficiently in both binary and BCD as well as handle the single-bit operations required in control applications. Special instructions have also been included to simplify loop counters, table look-up routines, and N-way branch routines.

4.0.1 Data Transfers

As can be seen in Figure 4-1, the 8-bit accumulator is the central point for all data transfers within the 8048. Data can be transferred between the 8 registers of each working register bank and the accumulator directly, i.e., the source or destination register is specified by the instruction. The remaining locations of the internal RAM array are referred to as Data Memory and are addressed indirectly via an address stored in either R0 or R1 of the active register bank. R0 and R1 are also used to indirectly address external data memory when it is present. Transfers to and from internal RAM require one cycle, while transfers to external RAM require two. Constants stored in Program Memory can be loaded directly to the accumulator and to the 8 working registers. Data can also be transferred directly between the accumulator and the on-board timer/

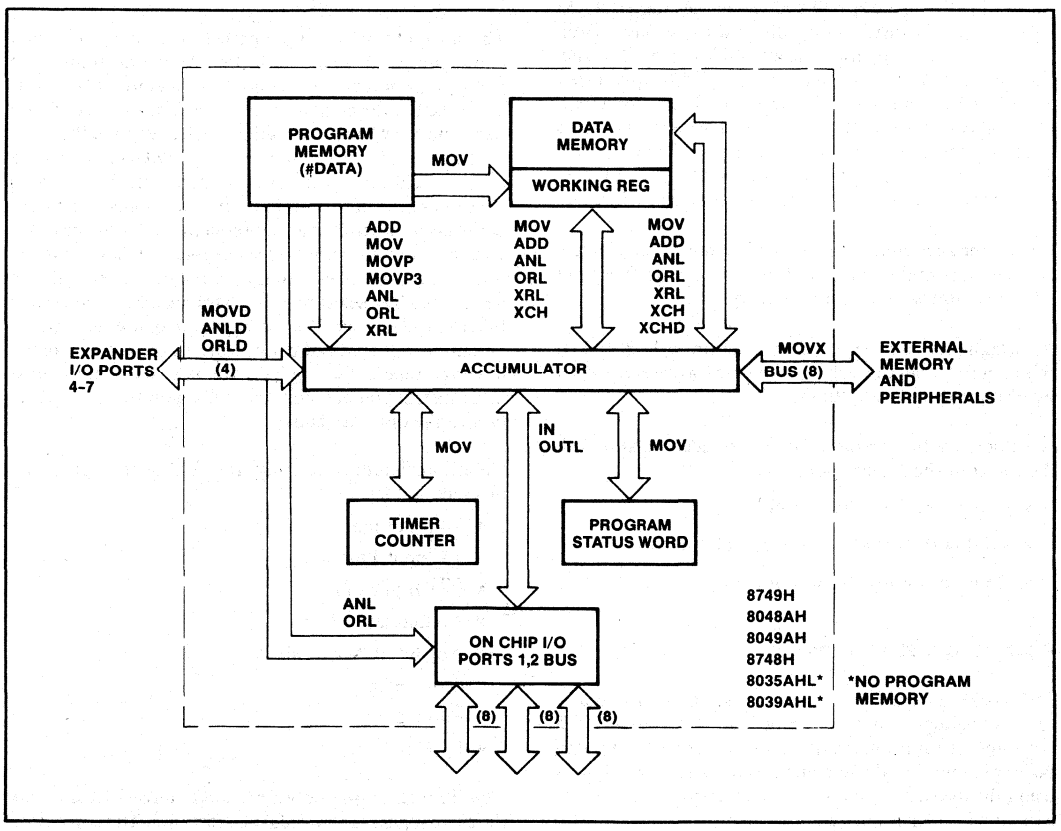


Figure 4-1. Data Transfer Instructions

counter or the accumulator and the Program Status word (PSW). Writing to the PSW alters machine status accordingly and provides a means of restoring status after an interrupt or of altering the stack pointer if necessary.

4.0.2 Accumulator Operations

Immediate data, data memory, or the working registers can be added with or without carry to the accumulator. These sources can also be ANDed, ORed, or Exclusive ORed to the accumulator. Data may be moved to or from the accumulator and working registers or data memory. The two values can also be exchanged in a single operation.

In addition, the lower 4 bits of the accumulator can be exchanged with the lower 4-bits of any of the internal RAM locations. This instruction, along with an instruction which swaps the upper and lower 4-bit halves of the accumulator, provides for easy handling of 4-bit quantities, including BCD numbers. To facilitate BCD arithmetic, a Decimal Adjust instruction is included. This instruction is used to correct the result of the binary addition of two 2-digit BCD numbers. Performing a decimal adjust on the result in the accumulator produces the required BCD result.

Finally, the accumulator can be incremented, decremented, cleared, or complemented and can be rotated left or right 1 bit at a time with or without carry.

Although there is no subtract instruction in the 8048AH, this operation can be easily implemented with three single-byte single-cycle instructions.

A value may be subtracted from the accumulator with the result in the accumulator by:

- Complementing the accumulator
- Adding the value to the accumulator
- Complementing the accumulator

4.0.3 Register Operations

The working registers can be accessed via the accumulator as explained above, or can be loaded immediate with constants from program memory. In addition, they can be incremented or decremented or used as loop counters using the decrement and jump, if not zero instruction, as explained under branch instructions.

All Data Memory including working registers can be accessed with indirect instructions via R0 and R1 and can be incremented.

4.0.4 Flags

There are four user-accessible flags in the 8048AH: Carry, Auxiliary Carry, F0, and F1. Carry indicates overflow of the accumulator, and Auxiliary Carry is used to indicate overflow between BCD digits and is used during decimal-adjust operation. Both Carry and Auxiliary Carry are accessible as part of the program status word and are stored on the stack during subroutines. F0 and F1 are undedicated general-purpose flags to be used as the programmer desires. Both flags can be cleared or complemented and tested by conditional jump instructions. F0 is also accessible via the Program Status word and is stored on the stack with the carry flags.

4.0.5 Branch Instructions

The unconditional jump instruction is two bytes and allows jumps anywhere in the first 2K words of program memory. Jumps to the second 2K of memory (4K words are directly addressable) are made first by executing a select memory bank instruction, then executing the jump instruction. The 2K boundary can only be crossed via a jump or subroutine call instruction, i.e., the bank switch does not occur until a jump is executed. Once a memory bank has been selected all subsequent jumps will be to the selected bank until another select memory bank instruction is executed. A subroutine in the opposite bank can be accessed by a select memory bank instruction followed by a call instruction. Upon completion of the subroutine, execution will automatically return to the original bank; however, unless the original bank is reselected, the next jump instruction encountered will again transfer execution to the opposite bank.

Conditional jumps can test the following inputs and machine status:

- T0 Input pin
- T1 Input Pin
- $\overline{\text{INT}}$ Input Pin
- Accumulator Zero
- Any bit of Accumulator
- Carry Flag
- F0 Flag
- F1 Flag

Conditional jumps allow a branch to any address within the current page (256 words) of execution. The conditions tested are the instantaneous values at the time the conditional jump is executed. For instance, the jump on accumulator zero instruction tests the accumulator itself, not an intermediate zero flag.

The decrement register and jump if not zero instruction combines a decrement and a branch instruction to create an instruction very useful in implementing a loop counter. This instruction can designate any one of the 8 working registers as a counter and can effect a branch to any address within the current page of execution.

A single-byte indirect jump instruction allows the program to be vectored to any one of several different locations based on the contents of the accumulator. The contents of the accumulator points to a location in program memory which contains the jump address. The 8-bit jump address refers to the current page of execution. This instruction could be used, for instance, to vector to any one of several routines based on an ASCII character which has been loaded in the accumulator. In this way ASCII key inputs can be used to initiate various routines.

4.0.6 Subroutines

Subroutines are entered by executing a call instruction. Calls can be made like unconditional jumps to any address in a 2K word bank, and jumps across the 2K boundary are executed in the same manner. Two separate return instructions determine whether or not status (upper 4-bits of PSW) is restored upon return from the subroutine.

The return and restore status instruction also signals the end of an interrupt service routine if one has been in progress.

4.0.7 Timer Instructions

The 8-bit on board timer/counter can be loaded or read via the accumulator while the counter is stopped or while counting. The counter can be started as a timer with an internal clock source or as an event counter or timer with an external clock applied to the T1 input pin. The instruction executed determines which clock source is used. A single instruction stops the counter whether it is operating with an internal or an external clock source. In addition, two instructions allow the timer interrupt to be enabled or disabled.

4.0.8 Control Instructions

Two instructions allow the external interrupt source to be enabled or disabled. Interrupts are initially disabled and are automatically disabled while an interrupt service routine is in progress and re-enabled afterward.

There are four memory bank select instructions, two to designate the active working register bank and two to control program memory banks. The operation of the program memory bank switch is explained in section 3.1.2.

The working register bank switch instructions allow the programmer to immediately substitute a second 8-register working register bank for the one in use. This effectively provides 16 working registers or it can be used as a means of quickly saving the contents of the registers in response to an interrupt. The user has the option to switch or not to switch banks on interrupt. However, if the banks are switched, the original bank will be automatically restored upon execution of a return and restore status instruction at the end of the interrupt service routine.

A special instruction enables an internal clock, which is the XTAL frequency divided by three to be output on pin T0. This clock can be used as a general-purpose clock in the user's system. This instruction should be used only to initialize the system since the clock output can be disabled only by application of system reset.

4.0.9 Input/Output Instructions

Ports 1 and 2 are 8-bit static I/O ports which can be loaded to and from the accumulator. Outputs are statically latched but inputs are not latched and must be read while inputs are present. In addition, immediate data from program memory can be ANDed or ORed directly to Port 1 and Port 2 with the result remaining on the port. This allows "masks" stored in program memory to selectively set or reset individual bits of the I/O ports. Ports 1 and 2 are configured to allow input on a given pin by first writing a "1" out to the pin.

An 8-bit port called BUS can also be accessed via the accumulator and can have statically latched outputs as well. It too can have immediate data ANDed or ORed directly to its outputs, however, unlike ports 1 and 2, all eight lines of BUS must be treated as either input or output at any one time. In addition to being a static port, BUS can be used as a true synchronous bi-directional port using the Move External instructions used to access external data memory. When these instructions are executed, a corresponding $\overline{\text{READ}}$ or $\overline{\text{WRITE}}$ pulse is generated and data is valid only at that time. When data is not being transferred, BUS is in a high impedance state. Note that the OUTL, ANL, and the BRL instructions for the BUS are for use with internal memory only.

The basic three on-board I/O ports can be expanded via a 4-bit expander bus using half of port 2. I/O expander devices on this bus consisit of four 4-bit ports which are addressed as ports 4 through 7. These ports have their own AND and OR instructions like the on-board ports as well as move instructions to transfer data in or out. The expander AND and OR instructions, however, combine

the contents of accumulator with the selected port rather than immediate data as is done with the on-board ports. I/O devices can also be added externally using the BUS port as the expansion bus. In this case the I/O ports become "memory mapped", i.e., they are addressed in the same way as external data memory and exist in the external data memory address space addressed by pointer register R0 or R1.

4.1 INSTRUCTION SET DESCRIPTION

The following pages describe the MCS®-48 instruction set in detail. The instruction set is first summarized with instructions grouped functionally. This summary page is followed by a detailed description listed alphabetically by mnemonic opcode.

The alphabetical listing includes the following information.

- Mnemonic
- Machine Code
- Verbal Description
- Symbolic Description
- Assembly Language Example

The machine code is represented with the most significant bit (7) to the left and two byte instructions are represented with the first byte on the left. The assembly language examples are formulated as follows:

```
Arbitrary
Label: Mnemonic, Operand;
Descriptive Comment
```

8048AH/8748H/8049AH/8749H
Instruction Set Summary

Mnemonic	Description	Bytes	Cycle
Accumulator			
ADD A, R	Add register to A	1	1
ADD A, @R	Add data memory to A	1	1
ADD A, # data	Add immediate to A	2	2
ADDC A, R	Add register with carry	1	1
ADDC A, @R	Add data memory with carry	1	1
ADDC A, # data	Add immediate with carry	2	2
ANL A, R	And register to A	1	1
ANL A, @R	And data memory to A	1	1
ANL A, # data	And immediate to A	2	2
ORL A, R	Or register to A	1	1
ORL A @R	Or data memory to A	1	1
ORL A, # data	Or immediate to A	2	2
XRL A, R	Exclusive Or register to A	1	1
XRL A, @R	Exclusive or data memory to A	1	1
XRL, A, # data	Exclusive or immediate to A	2	2
INC A	Increment A	1	1
DEC A	Decrement A	1	1
CLR A	Clear A	1	1
CPL A	Complement A	1	1
DA A	Decimal adjust A	1	1
SWAP A	Swap nibbles of A	1	1
RL A	Rotate A left	1	1
RLC A	Rotate A left through carry	1	1
RR A	Rotate A right	1	1
RRC A	Rotate A right through carry	1	1
Input/Output			
IN A, P	Input port to A	1	2
OUTL P, A	Output A to port	1	2
ANL P, # data	And immediate to port	2	2
ORL P, # data	Or immediate to port	2	2
*INS A, BUS	Input BUS to A	1	2
*OUTL BUS, A	Output A to BUS	1	2
*ANL BUS, # data	And immediate to BUS	2	2
*ORL BUS, # data	Or immediate to BUS	2	2
MOVD A, P	Input Expander port to A	1	2
MOVD P, A	Output A to Expander port	1	2
ANLD P, A	And A to Expander port	1	2
ORLD P, A	Or A to Expander port	1	2

Mnemonic	Description	Bytes	Cycles
Registers			
INC R	Increment register	1	1
INC @R	Increment data memory	1	1
DEC R	Decrement register	1	1
Branch			
JMP addr	Jump unconditional	2	2
JMPP @A	Jump indirect	1	2
DJNZ R, addr	Decrement register and jump	2	2
JC addr	Jump on carry = 1	2	2
JNC addr	Jump on carry = 0	2	2
JZ addr	Jump on A Zero	2	2
JNZ addr	Jump on A not Zero	2	2
JT0 addr	Jump on T0 = 1	2	2
JNT0 addr	Jump on T0 = 0	2	2
JT1 addr	Jump on T1 = 1	2	2
JNT1 addr	Jump on T1 = 0	2	2
JF0 addr	Jump on F0 = 1	2	2
JF1 addr	Jump on F1 = 1	2	2
JTF addr	Jump on timer flag = 1	2	2
JNI addr	Jump on INT = 0	2	2
JBb addr	Jump on Accumulator Bit	2	2
Subroutine			
CALL addr	Jump to subroutine	2	2
RET	Return	1	2
RETR	Return and restore status	1	2
Flags			
CLR C	Clear Carry	1	1
CPL C	Complement Carry	1	1
CLR F0	Clear Flag 0	1	1
CPL F0	Complement Flag 0	1	1
CLR F1	Clear Flag 1	1	1
CPL F1	Complement Flag 1	1	1
Data Moves			
MOV A, R	Move register to A	1	1
MOV A, @R	Move data memory to A	1	1
MOV A, # data	Move immediate to A	2	2
MOV R, A	Move A to register	1	1
MOV @R, A	Move A to data memory	1	1
MOV R, # data	Move immediate to register	2	2
MOV @R, # data	Move immediate to data memory	2	2
MOV A, PSW	Move PSW to A	1	1
MOV PSW, A	Move A to PSW	1	1

*For use with internal memory only.

8048AH/8748H/8049AH/8749H
Instruction Set Summary (Con't)

Mnemonic	Description	Bytes	Cycle
Data Moves (Cont'd)			
XCH A, R	Exchange A and register	1	1
XCH A, @R	Exchange A and data memory	1	1
XCHD A,@R	Exchange nibble of A and register	1	1
MOVX A, @R	Move external data memory to A	1	2
MOVX @R, A	Move A to external data memory	1	2
MOVP A,@A	Move to A from current page	1	2
MOVP3 A, @	Move to A from Page 3	1	2
Timer/Counter			
MOV A, T	Read Timer/Counter	1	1
MOV T, A	Load Timer/Counter	1	1
STRT T	Start Timer	1	1
STRT CNT	Start Counter	1	1
STOP TCNT	Stop Timer/Counter	1	1
EN TCNTI	Enable Timer/Counter Interrupt	1	1
DIS TCNTI	Disable Timer/Counter Interrupt	1	1

Mnemonic	Description	Bytes	Cycle
Control			
EN I	Enable external Interrupt	1	1
DIS I	Disable external Interrupt	1	1
SEL RB0	Select register bank 0	1	1
SEL RB1	Select register bank 1	1	1
SEL MB0	Select memory bank 0	1	1
SEL MB1	Select memory bank 1	1	1
ENT0 CLK	Enable clock output on T0	1	1
NOP	No Operation	1	1

8021H/8020H
Instruction Set Summary

Mnemonic	Description	Bytes	Cycle
Accumulator			
ADD A, R	Add register to A	1	1
ADD A, @R	Add data memory to A	1	1
ADD A, # data	Add immediate to A	2	2
ADDC A, R	Add with carry	1	1
ADDC A, @R	Add with carry	1	1
ADDC A, # data	Add with carry	2	2
ANL A, R	And register to A	1	1
ANL A, @R	And data memory to A	1	1
ANL A, # data	And immediate to A	2	2
ORL A, R	Or register to A	1	1
ORL A @R	Or data memory to A	1	1
ORL A, # data	Or immediate to A	2	2
XRL A, R	Exclusive or register to A	1	1
XRL A, @R	Exclusive or data memory to A	1	1
XRL, A, # data	Exclusive or immediate to A	2	2
INC A	Increment A	1	1
DEC A	Decrement A	1	1
CLR A	Clear A	1	1
CPL A	Complement A	1	1
DA A	Decimal adjust A	1	1
SWAP A	Swap nibbles of A	1	1
RL A	Rotate A left	1	1
RLC A	Rotate A left through carry	1	1
RR A	Rotate A right	1	1
RRC A	Rotate A right through carry	1	1
Input/Output			
IN A, P	Input port to A	1	2
OUTL P, A	Output A to port	1	2
*MOVD A, P	Input Expander port to A	1	2
*MOVD P, A	Output A to Expander port	1	2
*ANLD P, A	And A to Expander port	1	2
*ORLD P, A	Or A to Expander port	1	2
Registers			
INC R	Increment register	1	1
INC @R	Increment data memory	1	1

Mnemonic	Description	Bytes	Cycles
Branch			
JMP addr	Jump unconditional	2	2
JMPP @A	Jump indirect	1	2
DJNZ R, addr	Decrement register and Jump on R not zero	2	2
JC addr	Jump on carry = 1	2	2
JNC addr	Jump on carry = 0	2	2
JZ addr	Jump on A zero	2	2
JNZ addr	Jump on A not zero	2	2
JT1 addr	Jump on T1 = 1	2	2
JNT1 addr	Jump on T1 = 0	2	2
JTF addr	Jump on timer flag	2	2
Subroutine			
CALL addr	Jump to subroutine	2	2
RET	Return	1	2
Flags			
CLR C	Clear carry	1	1
CPL C	Complement carry	1	1
Data Moves			
MOV A, R	Move register to A	1	1
MOV A, @R	Move data memory to A	1	1
MOV A, # data	Move immediate to A	2	2
MOV R, A	Move A to register	1	1
MOV @R, A	Move A to data memory	1	1
MOV R, # data	Move immediate to register	2	2
MOV @R, # data	Move immediate to data memory	2	2
XCH A, A, R	Exchange A and register	1	1
XCH A, @R	Exchange A and data memory	1	2
XCHD A, @R	Exchange nibble of A and register	1	1
MOVP A, @A	Move to A from current page	1	2
Timer/Counter			
MOV A, T	Read Timer/Counter	1	1
MOV T, A	Load Timer/Counter	1	1
STRT T	Start Timer	1	1
STRT CNT	Start Counter	1	1
STOP TCNT	Stop Timer/Counter	1	1
NOP	No Operation	1	1

*Not included in 8020H.

8021H/8020H
Instruction Set Summary (Con't)

Instruction Set — The following instructions, which are found in the 8748H, have been deleted from the 8021H instruction set.

Data Moves	Registers	Branch	Timer	Control	Input/Output
MOV A, PSW	DEC R	JT0 addr	EN TCNTI	EN I	ANL P, #data
MOV PSW, A		JNT0 addr	DIS TCNTI	DIS I	ORL P, #data
MOVX A, @R	**Flags**	JF0 addr		SEL RB0	INS A, BUS*
MOVX @R, A		JF1 addr	**Subroutine**	SEL RB1	OUTL BUS, A*
MOVP3 A, @A	CLR F0	JN1 addr		SEL MB0	ANL BUS, #data
	CPL F0	JBb addr	RETR	SEL MB1	ORL BUS, #data
	CLR F1			ENT0 CLK	
	CPL F1				

*These Instructions have been replaced in the 8021H by IN A, PO and OUTL PO, A respectively.

8022
Instruction Set Summary

Mnemonic	Description	Bytes	Cycle	Hexadecimal Opcode
Accumulator				
ADD A,R_r	Add register to A	1	1	68-6F
ADD A,@R	Add data memory to A	1	1	60-61
ADD A,#data	Add immediate to A	2	2	03
ADDC A,R_r	Add register with carry	1	1	78-7F
ADDC A,@R	Add data memory with carry	1	1	70-71
ADDC A,#data	Data immediate with carry	2	2	13
ANL A,R_r	And register to A	1	1	58-5F
ANL A,@R	Add data memory to A	1	1	50-51
ANL A,#data	And immediate to A	2	2	53
ORL A,R_r	Or register to A	1	1	48-4F
ORL A,@R	Or data memory to A	1	1	40-41
ORL A,#data	Or immediate to A	2	2	43
XRL A,R_r	Exclusive Or register to A	1	1	D8-DF
XRL A,@R	Exclusive Or data memory to A	1	1	D0-D1
XRL A,#data	Exclusive Or immediate to A	2	2	D3
INC A	Increment A	1	1	17
DEC A	Decrement A	1	1	07
CLR A	Clear A	1	1	27
CPL A	Complement A	1	1	37
DA A	Decimal adjust A	1	1	57
SWAP A	Swap nibbles of A	1	1	47
RL A	Rotate A left	1	1	E7
RLC A	Rotate A left through carry	1	1	F7
RR A	Rotate A right	1	1	77
RRC A	Rotate A right through carry	1	1	67
Input/Output				
IN A, P_p	Input port to A	1	2	08,09,0A
OUTL P_pA	Output A to port	1	2	90,39,3A
MOVD A,P_p	Input expander port to A	1	2	0C-0F
MOVD P_p,A	Output A to expander port	1	2	3C-3F
ANLD $P_{p,A}$	And A to expander port	1	2	9C-9F
ORLD $P_{p,A}$	Or A to expander port	1	2	8C-8F
Registers				
INCR R_r	Increment register	1	1	18-1F
INC @R	Increment data memory	1	1	10-11
Branch				
JMP addr	Jump unconditional	2	2	04,24,44,64 84,A4,C4,E4
JMPP @ A	Jump indirect	1	2	B3
DJNZ R,addr	Decrement register and jump on R not zero	2	2	E8-EF
JC addr	Jump on carry = 1	2	2	F6
JNC addr	Jump on carry = 0	2	2	E6
JZ addr	Jump on A zero	2	2	C6
JNZ addr	Jump on A not zero	2	2	96
JT0	Jump on T0 = 1	2	2	36
JNT0	Jump on T0 = 0	2	2	26
JT1 addr	Jump on T1 = 1	2	2	56
JNT1 addr	Jump on T1 = 0	2	2	46
JTF addr	Jump on timer flag	2	2	16

Mnemonic	Description	Bytes	Cycle	Hexadecimal Opcode
Subroutine				
CALL addr	Jump to subroutine	1	2	14,34,54,74, 94,B4,D4,F4
RET	Return	1	2	83
Flags				
CLR C	Clear carry	1	1k	97
CPL C	Complement carry	1	1	A7
Data Moves				
MOV A,R_r	Move register to A	1	1	F8-FF
MOV A,@R	Move data memory to A	1	1	F0-F1
MOV A, #data	Move immediate to A	2	2	23
MOV R_r,A	Move A to register	1	1	A8-AF
MOV @R,A	Move A to data memory	1	1	A0-A1
MOV R_r, #data	Move immediate to register	2	2	B8-BF
MOV@R, #data	Move immediate to data memory	2	2	B0-B1
XCH A,R_r	Exchange A and register	1	1	28-2F
XCH A,@R	Exchange A and data memory	1	1	20-21
XCHD a,@R	Exchange nibble of A and register	1	1	30-31
MOVP A,@A	Move to A from current page	1	2	A3
Timer/Counter				
MOV A,T	Read timer/counter	1	1	42
MOV T,A	Load timer/counter	1	1	62
STRT T	Start timer	1	1	55
STRT CNT	Start counter	1	1	45
STOP TCNT	Stop timer/counter	1	1	65
A/D Converter				
RAD	Move conversion result register to A	1	2	80
SEL AN0	Select analog input zero	1	1	85
SEL AN1	Select analog input one	1	1	95
Interrupts				
EN I	Enable external interrupt	1	1	05
DIS I	Disable external interrupt	1	1	15
EN TCNTI	Enable timer/counter interrupt	1	1	25
DIS TCNTI	Disable timer/counter interrupt	1	1	35
RET I	Return from interrupt	1	2	93
NOP	No operation	1	1	00

8022
Instruction Set Summary (Con't)

Instruction Set — The following instructions, which are found in the 8748H, have been deleted from the 8022H instruction set.

Data Moves	Registers	Branch		Control	Input/Output

MOV A, PSW
MOV PSW, A
MOVX A, @R
MOVX @R, A
MOVP3 A, @A
	DEC R				ANL P, #data
					ORL P, #data
					INS A, BUS*
					OUTL BUS, A*
					ANL BUS, #data
					ORL BUS, #data

	Flags	JF0 addr JF1 addr JN1 addr JBb addr		SEL RB0 SEL RB1 SEL MB0 SEL MB1 ENT0 CLK	
	CLR F0 CPL F0 CLR F1 CPL F1		**Subroutine** RETR		

*These Instructions have been replaced in the 8022H by IN A, PO and OUTL PO, A respectively.

MCS®-48 INSTRUCTION SET
Symbols and Abbreviations Used

A	Accumulator
AC	Auxiliary Carry
addr	12-Bit Program Memory Address
Bb	Bit Designator (b = 0–7)
BS	Bank Switch
BUS	BUS Port
C	Carry
CLK	Clock
CNT	Event Counter
CRR	Conversion Result Register
D	Mnemonic for 4-Bit Digit (Nibble)
data	8-Bit Number or Expression
DBF	Memory Bank Flip-Flop
F0, F1	Flag 0, Flag 1
I	Interrupt
P	Mnemonic for "in-page" Operation
PC	Program Counter
Pp	Port Designator (p = 1, 2 or 4–7)
PSW	Program Status Word
Ri	Data memory Pointer (i = 0, or 1)
Rr	Register Designator (r = 0, 1 or 0–7)
SP	Stack Pointer
T	Timer
TF	Timer Flag
T0, T1	Test 0, Test 1
X	Mnemonic for External RAM
#	Immediate Data Prefix
@	Indirect Address Prefix
$	Current Value of Program Counter
(X)	Contents of X
((X))	Contents of Location Addressed by X
←	Is Replaced by

ADD A,R_r Add Register Contents to Accumulator

Encoding: | 0 1 1 0 | 1 r r r | 68H-6FH

Description: The contents of register 'r' are added to the accumulator. Carry is affected.

Operation: (A) ← (A) + (Rr) r = 0-7

Example:
```
ADDREG: ADD A,R6      ;ADD REG 6 CONTENTS
                      ;TO ACC
```

ADD A,@R_r Add Data Memory Contents to Accumulator

Encoding; | 0 1 1 0 | 0 0 0 r | 60H-61H

Description: The contents of the resident data memory location addressed by register 'i' bits 0-5 are added to the accumulator. Carry is affected.

Operation: (A) ← (A) + ((Ri)) i = 0-1

Example:
```
ADDM:  MOV R0, #01FH   ;MOVE '1F' HEX TO REG 0
       ADD A, @R0      ;ADD VALUE OF LOCATION
                       ;31 TO ACC
```

ADD A,#data Add Immediate Data to Accumulator

Encoding: | 0 0 0 0 | 0 0 1 1 | d_7 d_6 d_5 d_4 | d_3 d_2 d_1 d_0 | 03H

Description: This is a 2-cycle instruction. The specified data is added to the accumulator. Carry is affected.

Operation: (A) ← (A) + data

Example:
```
ADDID: ADD A,#ADDER:   ;ADD VALUE OF SYMBOL
                       ;ADDER' TO ACC
```

ADDC A,R_r Add Carry and Register Contents to Accumulator

Encoding: | 0 1 1 1 | 1 r r r | 78H-7FH

Description: The content of the carry bit is added to accumulator location 0 and the carry bit cleared. The contents of register 'r' are then added to the accumulator. Carry is affected.

Operation: (A) ← (A) + (Rr) + (C) r = 0-7

Example:
```
ADDRGC: ADDC A,R4      ;ADD CARRY AND REG 4
                       ;CONTENTS TO ACC
```

0-5 in 8048AH
**0-6 in 8049AH
0-7 in 8050AH

ADDC A,@R_i Add Carry and Data Memory Contents to Accumulator

Encoding: | 0 1 1 1 | 0 0 0 i | 70H-71H

Description: The content of the carry bit is added to accumulator location 0 and the carry bit cleared. Then the contents of the resident data memory location addressed by register 'i' bits 0-5* are added to the accumulator. Carry is affected.

Operation: (A) ← (A) + ((Ri)) + (C) i = 0-1

Example:
```
ADDMC:  MOV R1,#40      ;MOVE '40' DEC TO REG 1
        ADDC A,@R1      ;ADD CARRY AND LOCATION 40
                        ;CONTENTS TO ACC
```

ADDC A,@data Add Carry and Immediate Data to Accumulator

Encoding: | 0 0 0 1 | 0 0 1 1 | d_7 d_6 d_5 d_4 | d_3 d_2 d_1 d_0 | 13H

Description: This is a 2-cycle instruction. The content of the carry bit is added to accumulator location 0 and the carry bit cleared. Then the specified data is added to the accumulator. Carry is affected.

Operation: (A) ← (A) + data + (C)

Example:
```
ADDC A,#225             ;ADD CARRY AND '225' DEC
                        ;TO ACC
```

ANL A,R_r Logical AND Accumulator with Register Mask

Encoding: | 0 1 0 1 | 1 r r r | 58H-5FH

Description: Data in the accumulator is logically ANDed with the mask contained in working register 'r'.

Operation: (A) ← (A) AND (Rr) r = 0-7

Example:
```
ANDREG: ANL A,R3        ;'AND' ACC CONTENTS WITH MASK
                        ;IN REG 3
```

ANL A,@R_i Logical AND Accumulator with memory Mask

Encoding: | 0 1 0 1 | 0 0 0 r | 50H-51H

Description: Data in the accumulator is logically ANDed with the mask contained in the data memory location referenced by register 'i' bits 0-5**.

Operation: (A) ← (A) AND ((Ri)) i = 0-1

Example:
```
ANDDM:  MOV R0,#03FH    ;MOVE '3F' HEX TO REG 0
        ANL A, @R0      ;'AND' ACC CONTENTS WITH
                        ;MASK IN LOCATION 63
```

ANL A,#data Logical AND Accumulator with Immediate Mask

Encoding: | 0 1 0 1 | 0 0 1 1 | | d_7 d_6 d_5 d_4 | d_3 d_2 d_1 d_0 | 53H

Description: This is a 2-cycle instruction. Data in the accumulator is logically ANDed with an immediately-specified mask.

Operation: (A) ← (A) AND data

Examples:
```
ANDID: ANL A,#0AFH      ;'AND' ACC CONTENTS
                        ;WITH MASK 10101111
ANL A,#3 + X/Y          ;'AND' ACC CONTENTS
                        ;WITH VALUE OF EXP
                        ;'3 + XY/Y'
```

ANL BUS,#data* Logical AND BUS with Immediate Mask
(Not in 8021H, 8022H)

Encoding: | 1 0 0 1 | 1 0 0 0 | | d_7 d_6 d_5 d_4 | d_3 d_2 d_1 d_0 | 98H

Description: This is a 2-cycle instruction. Data on the BUS port is logically ANDed with an immediately-specified mask. This instruction assumes prior specification of an 'OUTL BUS, A' instruction.

Operation: (BUS) ← (BUS) AND data

Example:
```
ANDBUS: ANL BUS,#MASK   ;'AND' BUS CONTENTS
                        ;WITH MASK EQUAL VALUE
                        ;OF SYMBOL 'MASK'
```

ANL Pp,#data Logical AND Port 1-2 with Immediate Mask
(Not in 8021H, 8022H)

Encoding: | 1 0 0 1 | 1 0 p p | | d_7 d_6 d_5 d_4 | d_3 d_2 d_1 d_0 | 99H-9AH

Description: This is a 2-cycle instruction. Data on port 'p' is logically ANDed with an immediately-specified mask.

Operation: (pp) ← (Pp) AND DATA p = 1-2

Example:
```
ANDP2: ANL P2,#0F0H     ;'AND' PORT 2 CONTENTS
                        ;WITH MASK 'F0' HEX
                        ;(CLEAR P20-23)
```

*For use with internal memory $\overline{\text{ONLY}}$.

ANLD Pp,A Logical AND Port 4-7 with Accumulator Mask

Encoding:

1 0 0 1	1 1 p p

9CH-9FH

Description: This is a 2-cycle instruction. Data on port 'p' is logically ANDed with the digit mask contained in accumulator bits 0-3.

Operation: (Pp) ← (Pp) AND (A0-3) p = 4-7

Note: The mapping of port 'p' to opcode bits 0-1 is as follows:

1 0	Port
0 0	4
0 1	5
1 0	6
1 1	7

Example:

```
ANDP4: ANLD P4,A        ;'AND' PORT 4 CONTENTS
                        ;WITH ACC BITS 0-3
```

CALL address Subroutine Call

Encoding:

a_{10} a_9 a_8 1	0 1 0 0

a_7 a_6 a_5 a_4	a_3 a_2 a_1 a_0

Page	Hex Op Code
0	14
1	34
2	54
3	74
4	94
5	B4
6	D4
7	F4

Description: This is a 2-cycle instruction. The program counter and PSW bits 4-7 are saved in the stack. The stack pointer (PSW bits 0-2) is updated. Program control is then passed to the location specified by 'address'. PC bit 11 is determined by the most recent SEL MB instruction.

A CALL cannot begin in locations 2046-2047 or 4094-4095. Execution continues at the instruction following the CALL upon return from the subroutine.

Operation:

((SP)) ← (PC), (PSW_{4-7})
(SP) ← (SP) + 1
(PC_{8-10}) ← ($addr_{8-10}$)
(PC_{0-7}) ← ($addr_{0-7}$)
(PC_{11}) ← DBF

Example: Add three groups of two numbers. Put subtotals in locations 50, 51 and total in location 52.

```
          MOV R0,#50        ;MOVE '50' DEC TO ADDRESS
                            ;REG 0
BEGADD:   MOV A,R1          ;MOVE CONTENTS OF REG 1
                            ;TO ACC
          ADD A,R2          ;ADD REG 2 TO ACC
          CALL SUBTOT       ;CALL SUBROUTINE 'SUBTOT'
          ADDC A R3         ;ADD REG 3 TO ACC
          ADDC A,R4         ;ADD REG 4 TO ACC
          CALL SUBTOT       ;CALL SUBROUTINE 'SUBTOT'
          ADDC A,R5         ;ADD REG 5 TO ACC
          ADDC A,R6         ;ADD REG 6 TO ACC
          CALL SUBTOT       ;CALL SUBROUTINE 'SUBTOT'
SUBTOT:   MOV @R0,A         ;MOVE CONTENTS OF ACC TO
                            ;LOCATION ADDRESSED BY
                            ;REG 0
          INC R0            ;INCREMENT REG 0
          RET               ;RETURN TO MAIN PROGRAM
```

CLR A Clear Accumulator

Encoding: | 0 0 1 0 | 0 1 1 1 | 27H

Description: The contents of the accumulator are cleared to zero.

Operation: A ← 0

CLR C Clear Carry Bit

Encoding: | 1 0 0 1 | 0 1 1 1 | 97H

Description: During normal program execution, the carry bit can be set to one by the ADD, ADDC, RLC, CPL C, RRC, and DAA insructions. This instruction resets the carry bit to zero.

Operation: C ← 0

CLR F1 Clear Flag 1
(Not in 8021H, 8022H)

Encoding: | 1 0 1 0 | 0 1 0 1 | A5H

Description: Flag 1 is cleared to zero.

Operation: (F1) ← 0

CLR F0 Clear Flag 0
(Not in 8021H, 8022H)

Encoding: | 1 0 0 0 | 0 1 0 1 | 85H

Description: Flag 0 is cleared to zero.

Operation: (F0) ← 0

CPL A Complement Accumulator

Encoding: | 0 0 1 1 | 0 1 1 1 | 37H

Description: The contents of the accumulator are complemented. This is strictly a one's complement. Each one is changed to zero and vice-versa.

Operation: (A) ← NOT (A)

Example: Assume accumulator contains 01101010.

```
CPLA: CPL A          ;ACC CONTENTS ARE COMPLE-
                     ;MENTED TO 10010101
```

CPL C Complement Carry Bit

Encoding: | 1 0 1 0 | 0 1 1 1 | A7H

Description: The setting of the carry bit is complemented; one is changed to zero, and zero is changed to one.

Operation: (C) ← NOT (C)

Example: Set C to one; current setting is unknown.

```
CTO1: CLR C          ;C IS CLEARED TO ZERO
      CPL C          ;C IS SET TO ONE
```

CPL F0 Complement Flag 0
(Not in 8021H, 8022H)

Encoding: | 1 0 0 1 | 0 1 0 1 | 95H

Description: The setting of flag 0 is complemented; one is changed to zero, and zero is changed to one.

Operation: F0 ← NOT (F0)

CPL F1 Complement Flag 1
(Not in 8021H, 8022H)

Encoding: | 1 0 1 1 | 0 1 0 1 | B5H

Description: The setting of flag 1 is complemented; one is changed to zero, and zero is changed to one.

Operation: (F1) ← NOT (F1)

DA A Decimal Adjust Accumulator

Encoding: | 0 1 0 1 | 0 1 1 1 | 57H

Description: The 8-bit accumulator value is adjusted to form two 4-bit Binary Coded Decimal (BCD) digits following the binary addition of BCD numbers. The carry bit C is affected. If the contents of bits 0-3 are greater than nine, or if AC is one, the accumulator is incremented by six.

The four high-order bits are then checked. If bits 4-7 exceed nine, or if C is one, these bits are increased by six. If an overflow occurs, C is set to one.

Example: Assume accumulator contains 10011011.

```
DA A                              ;ACC Adjusted to 00000001
                                  ;WITH C SET

C AC 7     4 3     0
0  0 1 0 0 1 1 0 1 1
     0 0 0 0 0 1 1 0              ADD SIX TO BITS 0-7
0  1 1 0 1 0 0 0 0 1
     0 1 1 0                      ADD SIX TO BITS 4-7
1  0 0 0 0 0 0 0 0 1            OVERFLOW TO C
```

DEC A Decrement Accumulator

Encoding: | 0 0 0 0 | 0 1 1 1 | 07H

Description: The contents of the accumulator are decremented by one. The carry flag is not affected.

Operation: (A) ← (A) -1

Example: Decrement contents of external data memory location 63.

```
MOV R0,#3FH                       ;MOVE '3F' HEX TO REG 0
MOVX A, @R0                       ;MOVE CONTENTS OF
                                  ;LOCATION 63 TO ACC
DEC A                             ;DECREMENT ACC
MOVX @R0,A                        ;MOVE CONTENTS OF ACC TO
                                  ;LOCATION 63 IN EXPANDED
                                  ;MEMORY
```

DEC Rr Decrement Register

(Not in 8021H, 8022H)

Encoding: | 1 1 0 0 | 1 r r r | C8H-CFH

Description: The contents of working register 'r' are decremented by one.

Operation: (Rr) ← (Rr) -1 r = 0-7

Example:

```
DECR1: DEC R1                     ;DECREMENT CONTENTS OF REG 1
```

DIS I External Interrupt
(Not in 8021H)

Encoding: | 0 0 0 1 | 0 1 0 1 | 15H

Description: External interrupts are disabled. A low signal on the interrupt input pin has no effect.

DIS TCNTI Disable Timer/Counter Interrupt
(Not in 8021H)

Encoding: | 0 0 1 1 | 0 1 0 1 | 35H

Description: Timer/counter interrupts are disabled. Any pending timer interrupt request is cleared. The interrupt sequence is not initiated by an overflow, but the timer flag is set and time accumulation continues.

DJNZ R_r, address Decrement Register and Test

Encoding: | 1 1 1 0 | 1 r r r | | a_7 a_6 a_5 a_4 | a_3 a_2 a_1 a_0 | E8H-EFH

Description: This is a 2-cycle instruction. Register 'r' is decremented, then tested for zero. If the register contains all zeros, program control falls through to the next instruction. If the register contents are not zero, control jumps to the specified 'address'.

The address in this case must evaluate to 8-bits, that is, the jump must be to a location within the current 256-location page.

Example: (Rr) ← (Rr) -1 r = 0-7
If Rr not 0
(PC_{0-7}) ← addr
Note: A 12-bit address specification does not cause an error if the DJNZ instruction and the jump target are on the same page. If the DJNZ instruction begins in location 255 of a page, it must jump to a target address on the following page.

Example: Increment values in data memory locations 50-54.

```
        MOV R0,#50          ;MOVE '50' DEC TO ADDRESS
                            ;REG 0
        MOV R3,#5           ;MOVE '5' DEC TO COUNTER
                            ;REG 3
INCRT:  INC @R0             ;INCREMENT CONTENTS OF
                            ;LOCATION ADDRESSED BY
                            ;REG 0
        INC R0              ;INCREMENT ADDRESS IN REG 0
        DJNZ R3, INCRT      ;DECREMENT REG 3 — JUMP TO
                            ;'INCRT' IF REG 3 NONZERO
        NEXT —              ;'NEXT' ROUTINE EXECUTED
                            ;IF R3 IS ZERO
```

EN I Enable External Interrupt
(not in 8021H)

Encoding: | 0 0 0 0 | 0 1 0 1 | 05H

Description: External interrupts are enabled. A low signal on the interrupt input pin initiates the interrupt sequence.

EN TCNTI Enable Timer/Counter Interrupt
(not in 8021H)

Encoding: | 0 0 1 0 | 0 1 0 1 | 25H

Description: Timer/counter interrupts are enabled. An overflow of the timer/counter initiates the interrupt sequence.

ENT0 CLK Enable Clock Output
(not in 8021H, 8022H)

Encoding: | 0 1 1 1 | 0 1 0 1 | 75H

Description: The test 0 pin is enabled to act as the clock output. This function is disabled by a system reset.

Example:
```
EMTST0: ENT0 CLK          ;ENABLE T0 AS CLOCK OUTPUT
```

IN A,Pp Input Port or Data to Accumulator

Encoding: | 0 0 0 0 | 1 0 p p | 09H-0AH

Description: This is a 2-cycle instruction. Data present on port 'p' is transferred (read) to the accumulator. In the 8021 IN A,P2 inputs P_{20}-P_{23} to A_0-A_3 while A_4-A_7 is set to zero.

Operation: (A) ← (Pp)

```
                                  p = 1-2
INP12: IN A,P1          ;INPUT PORT 1 CONTENTS TO ACC
       MOV R6,A         ;MOVE ACC CONTENTS TO REG 6
       IN A,P2          ;INPUT PORT 2 CONTENTS TO ACC
       MOV R7,A         ;MOVE ACC CONTENTS TO REG 7
```

INC A Increment Accumulator

Encoding: | 0 0 0 1 | 0 1 1 1 | 17H

Description: The contents of the accumulator are incremented by one. Carry is not affected.

Operation: (A) ← (A) +1

Example: Increment contents of location 100 in external data memory.

```
INCA: MOV R0,#100      ;MOVE '100' DEC TO ADDRESS REG 0
      MOVX A,@R0       ;MOVE CONTENTS OF LOCATION
                       ;100 TO ACC
      INC A            ;INCREMENT A
      MOVX @R0,A       ;MOVE ACC CONTENTS TO
                       ;LOCATION 101
```

INC R_r Increment Register

Encoding: | 0 0 0 1 | 1 r r r | 18H-1FH

Description: The contents of working register 'r' are incremented by one.

Operation: (Rr) ← (Rr) + 1 r = 0-7

Example:
```
INCR0: INC R0          ;INCREMENT CONTENTS OF REG 0
```

INC @R_r Increment Data Memory Location

Encoding: | 0 0 0 1 | 0 0 0 r | 10H-11H

Description: The contents of the resident data memory location addressed by register 'i' bits 0-5* are incremented by one.

Operation: ((Ri)) ← ((Ri)) + 1 i = 0-1

Example:
```
INCDM: MOV R1,#03FH    ;MOVE ONES TO REG 1
       INC @R1         ;INCREMENT LOCATION 63
```

IN A,P0 Input of Port 0 Data to Accumulator
(8021H, 8022H Only)

Same as INS A, BUS except no RD pulse generated. 80H

INS A,BUS Strobed Input of BUS Data to Accumulator

Encoding: | 0 0 0 0 | 1 0 0 0 | 08H

Description: This is a 2-cycle instruction. Data present on the BUS port is transferred (read) to the accumulator when the RD pulse is dropped. (Refer to section on programming memory expansion for details.)

Operation: (A) ← (BUS)

Example:
```
INPBUS: INS A,BUS      ;INPUT BUS CONTENTS TO ACC
```

JBb address **Jump If Accumulator Bit Is Set**
(Not in 8021H, 8022H)

Encoding: | b_2 b_1 b_0 1 0 0 1 0 | a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0 |

Accumulator Bit	Hex Op Code
0	12
1	32
2	52
3	72
4	92
5	B2
6	D2
7	F2

Description: This is a 2-cycle instruction. Control passes to the specified address if accumulator bit 'b' is set to one.

Operation:

	b = 0–7
$(PC_{0-7}) \leftarrow$ addr	If Bb = 1
(PC) = (PC) + 2	If Bb = 0

Example:

```
JB4IS1: JB4 NEXT        ;JUMP TO 'NEXT' ROUTINE
                        ;IF ACC BIT 4 = 1
```

JC address **Jump If Carry Is Set**

Encoding: | 1 1 1 1 0 1 1 0 | a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0 | F6H

Description: This is a 2-cycle instruction. Control passes to the specified address if the carry bit is set to one.

Operation:

$(PC_{0-7}) \leftarrow$ addr	If C = 1
(PC) = (PC) + 2	If C = 0

Example:

```
JC1: JC OVFLOW          ;JUMP TO 'OVFLOW' ROUTINE
                        ;IF C = 1
```

JF0 address **Jump If Flag 0 Is Set**
(Not in 8021H, 8022H)

Encoding: | 1 0 1 1 0 1 1 0 | | a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0 | B6H

Description: This is a 2-cycle instruction. Control passes to the specified address if flag 0 is set to one.

Operation:

$(PC_{0-7}) \leftarrow$ addr	If F0 = 1
(PC) =(PC) + 2	If F0 = 0

Example:

```
JF0IS1: JF0 TOTAL       ;JUMP TO 'TOTAL' ROUTINE IF F0 = 1
```

JF1 address **Jump If Flag 1 Is Set**
(Not in 8021H, 8022H)

Encoding: | 0 1 1 1 | 0 1 1 0 | a_7 a_6 a_5 a_4 | a_3 a_2 a_1 a_0 | 76H

Description: This is a 2-cycle instruction. Control passes to the specified address if flag 1 is set to one.

Operation:
$(PC_{0-7}) \leftarrow$ addr — If F1 = 1
(PC) = (PC + 2) — If F1 = 0

Example:
```
JF1IS1: JF1 FILBUF      ;JUMP TO 'FILBUF'
                        ;ROUTINE IF F1 = 1
```

JMP address **Direct Jump within 2K Block**

Encoding: | a_{10} a_9 a_8 0 | 0 1 0 0 | a_7 a_6 a_5 a_4 | a_3 a_2 a_1 a_0

Page	Hex Op Code
0	04
1	24
2	44
3	64
4	84
5	A4
6	C4
7	E4

Description: This is a 2-cycle instruction. Bits 0-10 of the program counter are replaced with the directly-specified address. The setting of PC bit 11 is determined by the most recent SELECT MB instruction.

Operation:
$(PC_{8-10}) \leftarrow$ addr 8-10
$(PC_{0-7}) \leftarrow$ addr 0-7
$(PC_{11}) \leftarrow$ DBF

Example:
```
JMP SUBTOT      ;JUMP TO SUBROUTINE 'SUBTOT'
JMP $-6         ;JUMP TO INSTRUCTION SIX
                ;LOCATIONS BEFORE CURRENT
                ;LOCATION
JMP 2FH         ;JUMP TO ADDRESS '2F' HEX
```

JMPP @A **Indirect Jump within Page**

Encoding: | 1 0 1 1 | 0 0 1 1 | B3H

Description: This is a 2-cycle insruction. The contents of the program memory location pointed to by the accumulator are substituted for the 'page' portion of the program counter (PC bits 0-7).

Operation: $(PC_{0-7}) \leftarrow ((A))$

Example: Assume accumulator contains 0FH.

```
JMPPAG: JMPP @A          ;JUMP TO ADDRESS STORED IN
                         ;LOCATION 15 IN CURRENT PAGE
```

JNC address Jump If Carry Is Not Set

Encoding: | 1 1 1 0 | 0 1 1 0 | | a_7 a_6 a_5 a_4 | a_3 a_2 a_1 a_0 | E6H

Description: This is a 2-cycle instruction. Control passes to the specified address if the carry bit is not set, that is, equals zero.

Operation: $(PC_{0-7}) \leftarrow$ addr If C = 0
(PC) = (PC) + 2 If C = 1

Example:

```
JC0: JNC NOVFLO          ;JUMP TO 'NOVFLO' ROUTINE
                         ;IF C = 0
```

JNI address Jump If Interrupt Input Is Low
(Not in 8021H, 8022H)

Encoding: | 1 0 0 0 | 0 1 1 0 | | a_7 a_6 a_5 a_4 | a_3 a_2 a_1 a_0 | 86H

Description: This is a 2-cycle instruction. Control passes to the specified address if the interrupt input signal is low (= 0), that is, an external interrupt has been signaled. (This signal initiates an interrupt service sequence if the external interrupt is enabled.)

Operation: $(PC_{0-7}) \leftarrow$ addr If I = 0
(PC) = (PC) + 2 If I = 1

Example:

```
LOC 3: JNI EXTINT        ;JUMP TO 'EXTINT' ROUTINE
                         ;IF I = 0
```

JNT0 address Jump If Test 0 Is Low
(Not in 8021H)

Encoding: 0 0 1 0 0 1 1 0 | a_7 a_6 a_5 a_4 | a_3 a_2 a_1 a_0 | 26H

Description: This is a 2-cycle instruction. Control passes to the specified address, if the test 0 signal is low.

Operation: $(PC_{0-7}) \leftarrow$ addr If T0 = 0
(PC) = (PC) + 2 If T0 = 1

Example:

```
JT0LOW: JNT0 60          ;JUMP TO LOCATION 60 DEC
                         ;IF T0 = 0
```

JNT1 address Jump If Test 1 Is Low

Encoding: | 0 1 0 0 | 0 1 1 0 | a_7 a_6 a_5 a_4 | a_3 a_2 a_1 a_0 | 46H

Description: This is a 2-cycle instruction. Control passes to the specified address, if the test 1 signal is low.

Operation: $(PC_{0-7}) \leftarrow$ addr — If T1 = 0
(PC) = (PC) + 2 — If T1 = 1

JNZ Address Jump If Accumulator Is Not Zero

Encoding: | 1 0 0 1 | 0 1 1 0 | a_7 a_6 a_5 a_4 | a_3 a_2 a_1 a_0 | 96H

Description: This is a 2-cycle instruction. Control passes to the specified address if the accumulator contents are nonzero at the time this instruction is executed.

Operation: $(PC_{0-7}) \leftarrow$ addr — If A $\neq$ 0
(PC) = (PC) + 2 — If A = 0

Example:
```
JACCN0: JNZ 0ABH      ;JUMP TO LOCATION 'AB' HEX
                      ;IF ACC VALUE IS NONZERO
```

JTF address Jump If Timer Flag Is Set

Encoding: | 0 0 0 1 | 0 1 1 0 | a_7 a_6 a_5 a_4 | a_3 a_2 a_1 a_0 | 16H

Description: This is a 2-cycle instruction. Control passes to the specified address if the timer flag is set to one, that is, the timer/counter register has overflowed. Testing the timer flag resets it to zero. (This overflow initiates an interrupt service sequence if the timer-overflow interrupt is enabled.)

Operation: $(PC_{0-7}) \leftarrow$ addr — If TF = 1
(PC) = (PC) + 2 — If TF = 0

Example:
```
JTF1: JTF TIMER       ;JUMP TO 'TIMER' ROUTINE
                      ;IF TF = 1
```

JT0 address Jump If Test 0 Is High
(not in 8021H)

Encoding: | 0 0 1 1 | 0 1 1 0 | a_7 a_6 a_5 a_4 | a_3 a_2 a_1 a_0 | 36H

Description: This is a 2-cycle instruction. Control passes to the specified address if the test 0 signal is high (= 1).

Operation: $(PC_{0-7}) \leftarrow$ addr — If T0 = 1
(PC) = (PC) + 2 — If T0 = 0

Example:
```
JT0HI: JT0 53         ;JUMP TO LOCATION 53 DEC
                      ;IF T0 = 1
```

JT1 address Jump If Test 1 Is High

Encoding: | 0 1 0 1 | 0 1 1 0 | a_7 a_6 a_5 a_4 | a_3 a_2 a_1 a_0 | 56H

Description: This is a 2-cycle instruction. Control passes to the specified address if the test 1 signal is high (= 1).

Operation: $(PC_{0-7}) \leftarrow$ addr If T1 = 1
(PC) = (PC) + 2 If T1 = 0

Example:
```
JT1HI: JT1 COUNT        ;JUMP TO 'COUNT' ROUTINE
                        ;IF T1 = 1
```

JZ address Jump If Accumulator Is Zero

Encoding: | 1 1 0 0 | 0 1 1 0 | a_7 a_6 a_5 a_4 | a_3 a_2 a_1 a_0 | C6H

Description: This is a 2-cycle instruction. Control passes to the specified address if the accumulator contains all zeros at the time this instruction is executed.

Operation: $(PC_{0-7}) \leftarrow$ addr If A = 0
(PC) = (PC) + 2 If A $\neq$ 1

Example:
```
JACCO: JZ 0A3H          ;JUMP TO LOCATION 'A3' HEX
                        ;IF ACC VALUE IS ZERO
```

MOV A,#data Move Immediate Data to Accumulator

Encoding: | 0 0 1 0 | 0 0 1 1 | a_7 a_6 a_5 a_4 | a_3 a_2 a_1 a_0 | 23H

Description: This is a 2-cycle instruction. The 8-bit value specified by 'data' is loaded in the accumulator.

Operation: (A) $\leftarrow$ data

Example:
```
MOV A,#0A3H             ;MOVE 'A3' HEX TO ACC
```

MOV A,PSW Move PSW Contents to Accumulator
(Not in 8021H, 8022H)

Encoding: | 1 1 0 0 | 0 1 1 1 | C7H

Description: The contents of the program status word are moved to the accumulator.

Operation: (A) $\leftarrow$ (PSW)

Example: Jump to 'RB1SET' routine if PSW bank switch, bit 4, is set.
```
BSCHK: MOV A,PSW        ;MOVE PSW CONTENTS TO ACC
       JB4 RB1SET       ;JUMP TO 'RB1SET' IF ACC BIT 4 = 1
```

MOV A,R_r Move Register Contents to Accumulator

Encoding: | 1 1 1 1 | 1 r r r | F8H-FFH

Description: 8-bits of data are removed from working register 'r' into the accumulator.

Operation: (A) ← (Rr) r = 0-7

Example:
```
MAR: MOV A,R3          ;MOVE CONTENTS OF REG 3 TO ACC
```

MOV A,@R_i Move Data Memory Contents to Accumulator

Encoding | 1 1 1 1 | 0 0 0 i | F0H-F1H

Description: The contents of the resident data memory location addressed by bits 0-5* of register 'r' are moved to the accumulator. Register 'r' contents are unaffected.

Operation: (A) ← ((Ri)) i = 0-1

Example: Assume R1 contains 00110110.
```
MADM: MOV A,@R1        ;MOVE CONTENTS OF DATA MEM
                       ;LOCATION 54 TO ACC
```

MOV A,T Move Timer/Counter Contents to Accumulator

Encoding: | 0 1 0 0 | 0 0 1 0 | 42H

Description: The contents of the timer/event-counter register are moved to the accumulator.

Operation: (A) ← (T)

Example: Jump to "EXIT" routine when timer reaches '64', that is, when bit 6 set—assuming initialization 64,
```
TIMCHK: MOV A,T        ;MOVE TIMER CONTENTS TO ACC
        JB6 EXIT       ;JUMP TO 'EXIT' IF ACC BIT 6 = 1
```

MOV PSW,A Move Accumulator Contents to PSW
(Not in 8021H, 8022H)

Encoding: | 1 1 0 1 | 0 1 1 1 | D7H

Description: The contents of the accumulator are moved into the progam status word. All condition bits and the stack pointer are affected by this move.

Operation: (PSW) ← (A)

Example: Move up stack pointer by two memory locations, that is, increment the pointer by one.
```
INCPTR: MOV A,PSW      ;MOVE PSW CONTENTS TO ACC
        INC A          ;INCREMENT ACC BY ONE
        MOV PSW,A      ;MOVE ACC CONTENTS TO PSW
```

MOV R_r,A Move Accumulator Contents to Register

Encoding: | 1 0 1 0 | 1 r r r | A8H-AFH

Description: The contents of the accumulator are moved to register 'r'.

Operation: (Rr) ← (A) r = 0-7

Example:
```
MRA: MOV R0,A          ;MOVE CONTENTS OF ACC TO REG 0
```

MOV R_r,#data Move Immediate Data to Register

Encoding: | 1 0 1 1 | 1 r_2 r_1 r_0 | | d_7 d_6 d_5 d_4 | d_3 d_2 d_1 d_0 | B8H-BFH

Description: This is a 2-cycle instruction. The 8-bit value specified by 'data' is moved to register 'r'.

Operation: (Rr) ← data r = 0-7

Examples:
```
MIR4: MOV R4,#HEXTEN     ;THE VALUE OF THE SYMBOL
                         ;'HEXTEN' IS MOVED INTO REG 4
MIR 5: MOV R5,#PI*(R*R)  ;THE VALUE OF THE EXPRESSION
                         ;'PI*(R*R)' IS MOVED INTO REG 5
MIR 6: MOV R6, #0ADH     ;'AD' HEX IS MOVED INTO REG 6
```

MOV @ R_i,A Move Accumulator Contents to Data Memory

Encoding: | 1 0 1 0 | 0 0 0 i | A0H-A1H

Description: The contents of the accumulator are moved to the resident data memory location whose address is specified by bits 0-5** of register 'i'. Register 'i' contents are unaffected.

Operation: ((Ri)) ← (A) i = 0-1

Example: Assume R0 contains 00000111.
```
MDMA: MOV @R0,A          ;MOVE CONTENTS OF ACC TO
                         ;LOCATION 7 (REG 7)
```

MOV @ R_i,#data Move Immediate Data to Data memory

Encoding: | 1 0 1 1 | 0 0 0 i | | d_7 d_6 d_5 d_4 | d_3 d_2 d_1 d_0 | B0H-B1H

Description: This is a 2-cycle instruction. The 8-bit value specified by 'data' is moved to the resident data memory location addressed by register 'i', bits 0-5**.

Operation: ((Ri)) ← data i = 0-1

Examples: Move the hexadecimal value AC3F to locations 62-63.
```
MIDM: MOV R0,#62         ;MOVE '62' DEC TO ADDR REG 0
      MOV @R0,#0ACH      ;MOVE 'AC' HEX TO LOCATION 62
      INC R0             ;INCREMENT REG 0 to '63'
      MOV @R0,#3FH       ;MOVE '3F' HEX TO LOCATION 63
```

MOV T,A Move Accumulator Contents to Timer/Counter

Encoding: | 0 1 1 0 | 0 0 1 0 | 62H

Description: The contents of the accumulator are moved to the timer/event-counter register.

Operation: (T) ← (A)

Example: Initialize and start event counter.

```
INITEC: CLR A          ;CLEAR ACC TO ZEROS
        MOV T,A        ;MOVE ZEROS TO EVENT COUNTER
        START CNT      ;START COUNTER
```

MOVD A,Pp Move Port 4-7 Data to Accumulator

Encoding: | 0 0 0 0 | 1 1 p p | 0CH-0FH

Description: This is a 2-cycle instruction. Data on 8243 port 'p' is moved (read) to accumulator bits 0-3. Accumulator bits 4-7 are zeroed.

Operation: (0-3) ← (Pp) p = 4-7
(4-7) ← 0

Note: Bits 0-7 of the opcode are used to represent ports 4-7. If you are coding in binary rather than assembly language, the mapping is as follows:

Bits 1 0	Port
0 0	4
0 1	5
1 0	6
1 1	7

Example:

```
INPPT5: MOVD A,P5      ;MOVE PORT 5 DATA TO ACC
                       ;BITS 0-3, ZERO ACC BITS 4-7
```

MOVD Pp,A Move Accumulator Data to Port 4-7

Encoding: | 0 0 1 1 | 1 1 p p | 3CH-3FH

Description: This is a 2-cycle instruction. Data in accumulator bits 0-3 is moved (written) to 8243 port 'p'. Accumulator bits 4-7 are unaffected. (See NOTE above regarding port mapping.)

Operation: (Pp) ← (A_{0-3}) P = 4-7

Example: Move data in accumulator to ports 4 and 5.

```
OUTP45: MOVD P4,A      ;MOVE ACC BITS 0-3 TO PORT 4
        SWAP A         ;EXCHANGE ACC BITS 0-3 and 4-7
        MOVD P5,A      ;MOVE ACC BITS 0-3 TO PORT 5
```

MOVP A,@A Move Current Page Data to Accumulator

Encoding: | 1 0 1 0 | 0 0 1 1 | A3H

Description: The contents of the program memory location addressed by the accumulator are moved to the accumulator. Only bits 0-7 of the program counter are affected, limiting the program memory reference to the current page. The program counter is restored *following* this operation.

Operation: $(PC_{0-7}) \leftarrow (A)$
$(A) \leftarrow ((PC))$
Note: This is a 1-byte, 2-cycle instruction. If it appears in location 255 of a program memory page, @A addresses a location in the *following* page.

Example:
```
MOV128: MOV A,#128      ;MOVE '128' DEC TO ACC
        MOVP A,@A       ;CONTENTS OF 129th LOCATION IN
                        ;CURRENT PAGE ARE MOVED TO ACC
```

MOVP3 A,@A Move Page 3 Data to Accumulator
(Not in 801H, 8022H)

Encoding: | 1 1 1 0 | 0 0 1 1 | E3H

Description: This is a 2-cycle instruction. The contents of the program memory location (within page 3) addressed by the accumulator are moved to the accumulator. The program counter is restored following this operation.

Operation: $(PC_{0-7}) \leftarrow (A)$
$(PC_{8-11}) \leftarrow 0011$
$(A) \leftarrow ((PC))$

Example: Look up ASCII equivalent of hexadecimal code in table contained at the beginning of page 3. Note that ASCII characters are designated by a 7-bit code; the eighth bit is always reset.

```
TABSCH: MOV A,#0B8H     ;MOVE 'B8' HEX TO ACC (10111000)
        ANL A,#7FH      ;LOGICAL AND ACC TO MASK BIT
                        ;7 (00111000)
        MOVP3 A,@A      ;MOVE CONTENTS OF LOCATION '38'
                        ;HEX IN PAGE 3 TO ACC (ASCII '8')
```

Access contents of location in page 3 labelled TAB1.
Assume current program location is not in page 3.

```
TABSCH: MOV A,#LOW TAB 1 ;ISOLATE BITS 0-7 OF LABEL
                         ;ADDRESS VALUE
        MOVP3 A,@A       ;MOVE CONTENTS OF PAGE 3
                         ;LOCATION LABELED 'TAB1' TO ACC
```

MOVX A,@R_I Move External-Data-Memory Contents to Accumulator
(Not in 8021H, 8022H)

Encoding: | 1 0 0 0 | 0 0 0 i | 80H-81H

Description: This is a 2-cycle instruction. The contents of the external data memory location addressed by register 'i' are moved to the accumulator. Register 'i' contents are unaffected. A read pulse is generated.

Operation: (A) ← ((Ri)) i = 0-1

Example: Assume R1 contains 01110110.

```
MAXDM:  MOVX A,@R1      ;MOVE CONTENTS OF LOCATION
                        ;118 TO ACC
```

MOVX @R_i,A Move Accumulator Contents to External Data Memory
(Not in 8021H, 8022H)

Encoding: | 1 0 0 1 | 0 0 0 i | 90H-91H

Description: This is a 2-cycle instruction. The contents of the accumulator are moved to the external data memory location addressed by register 'i'. Register 'i' contents are unaffected. A write pulse is generated.

Operation: ((Ri)) ← A i = 0-1

Example: Assume R0 contains 11000111.

```
MXDMA:  MOVX @R0,A      ;MOVE CONTENTS OF ACC TO
                        ;LOCATION 199 IN EXPANDED
                        ;DATA MEMORY
```

NOP The NOP Instruction

Encoding: | 0 0 0 0 | 0 0 0 0 | 00H

Description: No operation is performed. Execution continues with the following instruction.

ORL A,R_r Logical OR Accumulator With Register Mask

Encoding: | 0 1 0 0 | 1 r r r | 48H-4FH

Description: Data in the accumulator is logically ORed with the mask contained in working register 'r'.

Operation: (A) ← (A) OR (Rr) r = 0-7

Example:

```
ORREG:  ORL A,R4        ;'OR' ACC CONTENTS WITH
                        ;MASK IN REG 4
```

ORL A,@R_i Logical OR Accumulator With Memory Mask

Encoding: | 0 1 0 0 | 0 0 0 i | 40H-41H

Description: Data in the accumulator is logically ORed with the mask containd in the resident data memory location referenced by register 'r', bits 0-5*.

Operation: (A) ← (A) OR ((Ri)) i = 0-1

Example:
```
ORDM: MOV R0,#3FH      ;MOVE '3F' HEX TO REG 0
      ORL A,@R0        ;'OR' AC CONTENTS WITH MASK
                       ;IN LOCATION 63
```

ORL A,#data Logical OR Accumulator With Immediate Mask

Encoding: | 0 1 0 0 | 0 0 1 1 | d_7 d_6 d_5 d_4 | d_3 d_2 d_1 d_0 | 43H

Description: This is a 2-cycle instruction. Data in the accumulator is logically ORed with an immediately-specified mask.

Operation: (A) ← (A) OR data

Example:
```
ORID: ORL A,#'X'       ;'OR' ACC CONTENTS WITH MASK
                       ;01011000 (ASCII VALUE OF 'X')
```

ORL BUS,#data* Logical OR BUS With Immediate Mask
(Not in 8021H, 8022H)

Encoding: | 1 0 0 0 | 1 0 0 0 | d_7 d_6 d_5 d_4 | d_3 d_2 d_1 d_0 | 88H

Description: This is a 2-cycle instruction. Data on the BUS port is logically ORed with an immediately-specified mask. This instruction assumes prior specification on an 'OUTL BUS,A' instruction.

Operation: (BUS) ← (BUS) OR data

Example:
```
ORBUS: ORL BUS,#HEXMSK  :'OR' BUS CONTENTS WITH MASK
                        ;EQUAL VALUE OF SYMBOL 'HEXMSK'
```

ORL Pp, #data Logical OR Port 1 or 2 With Immediate Mask
(Not in 8021H, 8022H)

Encoding: | 1 0 0 0 | 1 0 p p | d_7 d_6 d_5 d_4 | d_3 d_2 d_1 d_0 | 89H-8AH

Description: This is a 2-cycle instruction. Data on port 'p' is logically ORed with an immediately-specified mask.

Operation: (Pp) ← (Pp) OR data p = 1-2

Example:
```
ORP1: ORL P1, #0FFH     ;'OR' PORT 1 CONTENTS WITH MASK
                        ;'FF' HEX (SET PORT 1 TO ALL ONES)
```

*For use with internal memory ONLY.

ORLD Pp,A Logical OR Port 4-7 With Accumulator Mask

Encoding: | 1 0 0 0 | 1 1 p p | 8CH-8FH

Description: This is a 2-cycle instruction. Data on port 'p' is logically ORed with the digit mask contained in accumulator bits 0-3.

Operation: (Pp) ← (Pp) OR (A_{0-3}) p = 4-7

Example:
```
ORP7: ORLD P7,A      ;'OR' PORT 7 CONTENTS WITH ACC
                     ;BITS 0-3
```

OUTL P0,A Output Accumulator Data to Port 0
(8021H, 8022H Only)

Encoding: | 1 0 0 0 | 0 0 0 0 | 90H

OUTL BUS,A* Output Accumulator Data to BUS
(Not in 8021H, 8022H)

Encoding: | 0 0 0 0 | 0 0 1 0 | 02H

Description: This is a 2-cycle instruction. Data residing in the accumulator is transferred (written) to the BUS port and latched. The latched data remains valid until altered by another OUTL instruction. Any other instruction requiring use of the BUS port (except INS) destroys the contents of the BUS latch. This includes expanded memory operations (such as the MOVX instruction). Logical operations on BUS data (AND, OR) assume the OUTL BUS,A instruction has been issued previously.

Does not apply for OUTL P0,A OF 8021H, 8022H

Operation: (BUS) ← (A)

Example:
```
OUTLBP: OUTL BUS, A      ;OUTPUT ACC CONTENTS TO BUS
```

OUTL Pp,A Output Accumulator Data to Port 1 or 2

Encoding: | 0 0 1 1 | 1 0 p p | 39H-3AH

Description: This is a 2-cycle instruction. Data residing in the accumulator is transferred (written) to port 'p' and latched.

Operation: (Pp) ← (A) p = 1-2

Example:
```
OUTLP: MOV A,R7      ;MOVE REG 7 CONTENTS TO ACC
       OUTL P2,A     ;OUTPUT ACC CONTENTS TO PORT 2
       MOV A, R6     ;MOV REG 6 CONTENTS TO ACC
       OUTL P1,A     ;OUTPUT ACC CONTENTS TO PORT 1
```

*For use with internal memory ONLY.

RAD Move Conversion Result Register to Accumulator
(8022H Only)

Encoding: | 1 0 0 0 | 0 0 0 0 | 80H

Description: This is a 2-cycle instruction. The contents of the A/D conversion result register are moved to the accumulator.

Operation: (A) ← (CRR)

RET Return Without PSW Restore

Encoding: | 1 0 0 0 | 0 0 1 1 | 83H

Description: This is a 2-cycle instruction. The stack pointer (PSW bits 0-2) is decremented. The program counter is then restored from the stack. PSW bits 4-7 are not restored.

Operation: (SP) ← (SP)-1
(PC) ← ((SP))

RETI Return From Interrupt
(8022H Only)

Encoding: | 1 0 0 1 | 0 0 1 1 | 93H

Description: This is a 2-cycle instruction. The stack pointer is decremented and the program counter is restored from the stack. Interrupt input logic is re-enabled.

Operation: (SP) ← (SP)-1
(PC) ← ((SP)) + 1

RETR Return with PSW Restore
(Not in 8021H, 8022H)

Encoding: | 1 0 0 1 | 0 0 1 1 | 93H

Description: This is a 2-cycle instruction. The stack pointer is decremented. The program counter and bits 4-7 of the PSW are then restored from the stack. Note that RETR should be used to return from an interrupt, but should not be used within the interrupt service routine as it signals the end of an interrupt routine by resetting the Interrupt in Progress flip-flop.

Operation: (SP) ← (SP)-1
(PC) ← ((SP))
(PSW 4-7) ← ((SP))

RL A Rotate Left without Carry

Encoding: | 1 1 1 0 | 0 1 1 1 | E7H

Description: The contents of the accumulator are rotated left one bit. Bit 7 is rotated into the bit 0 position.

Operation: (An + 1) ← (An)
(A0) ← (A7) n = 0-6

Example: Assume accumulator contains 10110001.

```
RLNC: RL A          ;NEW ACC CONTENTS ARE 01100011
```

RLC A Rotate Left through Carry

Encoding: | 1 1 1 1 | 0 1 1 1 | F7H

Description: The contents of the accumulator are rotated left one bit. Bit 7 replaces the carry bit; the carry bit is rotatd into the bit 0 position.

Operation: (An + 1) ← (An)
n = 0-6
(A0) ← (C)
(C) ← (A7)

Example: Assume accumulator contains a 'signed' number; isolate sign without changing value.

```
RLTC: CLR C         ;CLEAR CARRY TO ZERO
      RLC A         ;ROTATE ACC LEFT, SIGN
                    ;BIT (7) IS PLACED IN CARRY
      RR A          ;ROTATE ACC RIGHT — VALUE
                    ;(BITS 0-6) IS RESTORED,
                    ;CARRY UNCHANGED, BIT 7
                    ;IS ZERO
```

RR A Rotate Right without Carry

Encoding: | 0 1 1 1 | 0 1 1 1 | 77H

Description: The contents of the accumulator are rotated right one bit. Bit 0 is rotated into the bit 7 position.

Operation: (An) ← (An + 1) n = 0-6
(A7) ← (A0)

Example: Assume accumulator contains 10110001.

```
RRNC: RR A          ;NEW ACC CONTENTS ARE 11011000
```

RRC A Rotate Right through Carry

Encoding: | 0 1 1 0 | 0 1 1 1 | 67H

Description: The contents of the accumulator are rotated right one bit. Bit 0 replaces the carry bit; the carry bit is rotated into the bit 7 position.

Operation: (An) ← (An + 1) n = 0-6
(A7) ← (C)
(C) ← (A_0)

Example: Assume carry is not set and accumulator contains 10110001.

```
RRTC: RRC A          ;CARRY IS SET AND ACC
                     ;CONTAINS 01011000
```

SEL AN0 Select Analog Input Zero
(8022H Only)

Encoding: | 1 0 0 1 | 0 1 0 1 | 95H

SEL AN1 Select Analog Input One
(8022H Only)

Encoding: | 1 0 0 0 | 0 1 0 1 | 85H

Description: One of the two analog inputs to the A/D converter is selected. The conversion process is started. Restarting a sequence deletes the sequence in progress.

SEL MB0 Select Memory Bank 0
(Not in 8021H, 8022H)

Encoding: | 1 1 1 0 | 0 1 0 1 | E5H

Description: PC bit 11 is set to zero on next JMP or CALL instruction. All references to program memory addresses fall within the range 0-2047.

Operation: (DBF) ← 0

Example: Assume program counter contains 834 Hex.

```
SEL MB0              ;SELECT MEMORY BANK 0
JMP $+20             ;JUMP TO LOCATION 58 HEX
```

SEL MB1 Select Memory Bank 1
(Not in 8021H, 8022H)

Encoding: | 1 1 1 1 | 0 1 0 1 | F5H

Description: PC bit 11 is set to one on next JMP or CALL instruction. All references to program memory addresses fall within the range 2048-4095.

Operation: (DBF) ← 1

SEL RB0 Select Register Bank 0
(Not in 8021H, 8022H)

Encoding: | 1 1 0 0 | 0 1 0 1 | C5H

Description: PSW bit 4 is set to zero. References to working registers 0-7 address data memory locations 0-7. This is the recommended setting for normal program execution.

Operation: (BS) ← 0

SEL RB1 Select Register Bank 1
(Not in 8021H, 8022H)

Encoding: | 1 1 0 1 | 0 1 0 1 | D5H

Description: PSW bit 4 is set to one. References to working registers 0-7 address data memory locations 24-31. This is the recommended setting for interrupt service routines, since locations 0-7 are left intact. The setting of PSW bit 4 in effect at the time of an interrupt is restored by the RETR instruction when the interrupt service routine is completed.

Operation: (BS) ← 1

Example: Assume an external interrupt has occurred, control has passed to program memory location 3, and PSW bit 4 was zero before the interrupt.

Operation:

```
LOC3: JNI INIT          ;JUMP TO ROUTINE 'INIT' IF
                        ;INTERRUPT INPUT IS ZERO
INIT: MOV R7,A          ;MOVE ACC CONTENTS TO
                        ;LOCATION 7
      SEL RB1           ;SELECT REG BANK 1
      MOV R7,#0FAH      ;MOVE 'FA' HEX TO LOCATION 31
      .
      .
      .
      SEL RB0           ;SELECT REG BANK 0
      MOV A,R7          ;RESTORE ACC FROM LOCATION 7
      RETR              ;RETURN — RESTORE PC AND PSW
```

STOP TCNT Stop Timer/Event-Counter

Encoding: | 0 1 1 0 | 0 1 0 1 | 65H

Description: This instruction is used to stop both time accumulation and event counting.

Example: Disable interrupt, but jump to interrupt routine after eight overflows and stop timer. Count overflows in register 7.

```
START: DIS TCNTI          ;DISABLE TIMER INTERRUPT
       CLR A              ;CLEAR ACC TO ZEROS
       MOV T,A            ;MOVE ZEROS TO TIMER
       MOV R7,A           ;MOVE ZEROS TO REG 7
       STRT T             ;START TIMER
MAIN:  JTF COUNT          ;JUMP TO ROUTINE 'COUNT'
                          ;IF TF = 1 AND CLEAR TIMER FLAG
       JMP MAIN           ;CLOSE LOOP
COUNT: INC R7             ;INCREMENT REG 7
       MOV A,R7           ;MOVE REG 7 CONTENTS TO ACC
       JB3 INT            ;JUMP TO ROUTINE 'INT' IF ACC
                          ;BIT 3 IS SET (REG 7 = 8)
       JMP MAIN           ;OTHERWISE RETURN TO ROUTINE
                          ;MAIN
       .
       .
       .
INT:   STOP TCNT          ;STOP TIMER
       JMP 7H             ;JUMP TO LOCATION 7 (TIMER)
                          ;INTERRUPT ROUTINE
```

STRT CNT Start Event Conter

Encoding: | 0 1 0 0 | 0 1 0 1 | 45H

Description: The test 1 (T1) pin is enabled as the event-counter input and the counter is started. The event-counter register is incremented with each high-to-low transition on the T1 pin.

Example: Initialize and start event counter. Assume overflow is desired with first T1 input.

```
STARTC: EN TCNTI          ;ENABLE COUNTER INTERRUPT
        MOV A,#0FFH       ;MOVE 'FF'HEX (ONES) TO ACC
        MOV T,A           ;MOVES ONES TO COUNTER
        STRT CNT          ;ENABLE T1 AS COUNTER
                          ;INPUT AND START
```

STRT T Start Timer

Encoding: | 0 1 0 1 | 0 1 0 1 | 55H

Description: Timer accumulation is initiated in the timer register. The register is incremented every 32 instruction cycles. The prescaler which counts the 32 cycles is cleared but the timer register is not.

Example: Initialize and start timer.

```
STARTT: CLR A          ;CLEAR ACC TO ZEROS
        MOV T,A        ;MOVE ZEROS TO TIMER
        EN TCNTI       ;ENABLE TIMER INTERRUPT
        STRT T         ;START TIMER
```

SWAP A Swap Nibbles within Accumulator

Encoding: | 0 1 0 0 | 0 1 1 1 | 47H

Description: Bits 0-3 of the accumulator are swapped with bits 4-7 of the accumulator.

Operation: $(A_{4-7}) \leftrightarrows (A_{0-3})$

Example: Pack bits 0-3 of locations 50-51 into location 50.

```
PCKDIG: MOV R0, #50     ;MOVE '50' DEC TO REG 0
        MOV R1, #51     ;MOVE '51' DEC TO REG 1
        XCHD A,@R0      ;EXCHANGE BITS 0-3 OF ACC
                        ;AND LOCATION 50
        SWAP A          ;SWAP BITS 0-3 AND 4-7 OF ACC
        XCHD A,@R1      ;EXCHANGE BITS 0-3 OF ACC AND
                        ;LOCATION 51
        MOV @R0,A       ;MOVE CONTENTS OF ACC TO
                        ;LOCATION 50
```

XCH A,R_r Exchange Accumulator-Register Contents

Encoding: | 0 0 1 0 | 1 r r r | 28H-2FH

Description: The contents of the accumulator and the contents of working register 'r' are exchanged.

Operation: (A) $\leftrightarrows$ (Rr) r = 0-7

Example: Move PSW contents to Reg 7 without losing accumulator contents.

```
XCHAR7: XCH A,R7        ;EXCHANGE CONTENTS OF REG 7
                        ;AND ACC
        MOV A, PSW      ;MOVE PSW CONTENTS TO ACC
        XCH A,R7        ;EXCHANGE CONTENTS OF REG 7
                        'AND ACC AGAIN
```

XCH A,@R_i Exchange Accumulator and Data Memory Contents

Encoding: | 0 0 1 0 | 0 0 0 r | 20H-21H

Description: The contents of the accumulator and the contents of the resident data memory location addressed by bits 0-5** of register 'i' are exchanged. Register 'i' contents are unaffected.

Operation: (A) ⇆ ((Ri)) i = 0-1

Example: Decrement contents of location 52.

```
DEC52: MOV R0,#52      ;MOVE '52' DEC TO ADDRESS REG 0
       XCH A,@R0       ;EXCHANGE CONTENTS OF ACC
                       ;AND LOCATION 52
       DEC A           ;DECREMENT ACC CONTENTS
       XCH A,@R0       ;EXCHANGE CONTENTS OF ACC
                       ;AND LOCATION 52 AGAIN
```

XCHD A,@R_i Exchange Accumulator and Data Memory 4-Bit Data

Encoding: | 0 0 1 1 | 0 0 0 i | 30H-31H

Description: This instruction exchanges bits 0-3 of the accumulator with bits 0-3 of the data memory location addressed by bits 0-5** of register 'i'. Bits 4-7 of the accumulator, bits 4-7 of the data memory location, and the contents of register 'i' are unaffected.

Operation: (A_{0-3}) ⇆ ((Ri0-3)) i = 0-1

Example: Assume program counter contents have been stacked in locations 22-23.

```
XCHNIB: MOV R0,#23     ;MOVE '23' DEC TO REG 0
        CLR A          ;CLEAR ACC TO ZEROS
        XCHD A,@R0     ;EXCHANGE BITS 0-3 OF ACC AND
                       ;LOCATION 23 (BTS 8-11 OF PC ARE
                       ;ZEROED, ADDRESS REFERS
                       :TO PAGE 0)
```

XRL A,R_r Logical XOR Accumulator With Register Mask

Encoding: | 1 1 0 1 | 1 r r r | D8H-DFH

Description: Data in the accumulator is EXCLUSIVE ORed with the mask contained in working register 'r'.

Operation: (A) ← (A) XOR (Rr) r = 0-7

Example:

```
XORREG: XRL A,R5       ;'XOR' ACC CONTENTS WITH
                       ;MASK IN REG 5
```

XRL A,@R_i Logical XOR Accumulator With Memory Mask

Encoding: | 1 1 0 1 | 0 0 0 i | D0H-D1H

Description: Data in the accumulator is EXCLUSIVE ORed with the mask contained in the data memory location addressed by register 'i', bits 0–5*.

Operation: (A) ← (A) XOR ((Ri)) i = 0-1

Example:
```
XORDM:  MOV R1,#20H     ;MOVE '20' HEX TO REG 1
        XRL A,@R1       ;'XOR' ACC CONTENTS WITH MASK
                        ;IN LOCATION 32
```

XRL A,#data Logical XOR Accumulator With Immediate Mask

Encoding: | 1 1 0 1 | 0 0 1 1 | d_7 d_6 d_5 d_4 | d_3 d_2 d_1 d_0 | D3H

Description: This is a 2-cycle instruction. Data in the accumulator is EXCLUSIVE ORed with an immediately-specified mask.

Operation: (A) ← (A) XOR data

Example:
```
XORID:  XOR A,#HEXTEN   ;XOR CONTENTS OF ACC WITH MASK
                        ;EQUAL VALUE OF SYMBOL 'HEXTEN'
```

MCS®-48 Application Examples 5

CHAPTER 5
MCS®-48 APPLICATION EXAMPLES

5.0 INTRODUCTION

This chapter is organized in two sections, Hardware and Software. The hardware section gives examples of some typical configurations of MCS®-48 components, while the software section gives assembly language listings of some common applications routines.

5.1 HARDWARE EXAMPLES

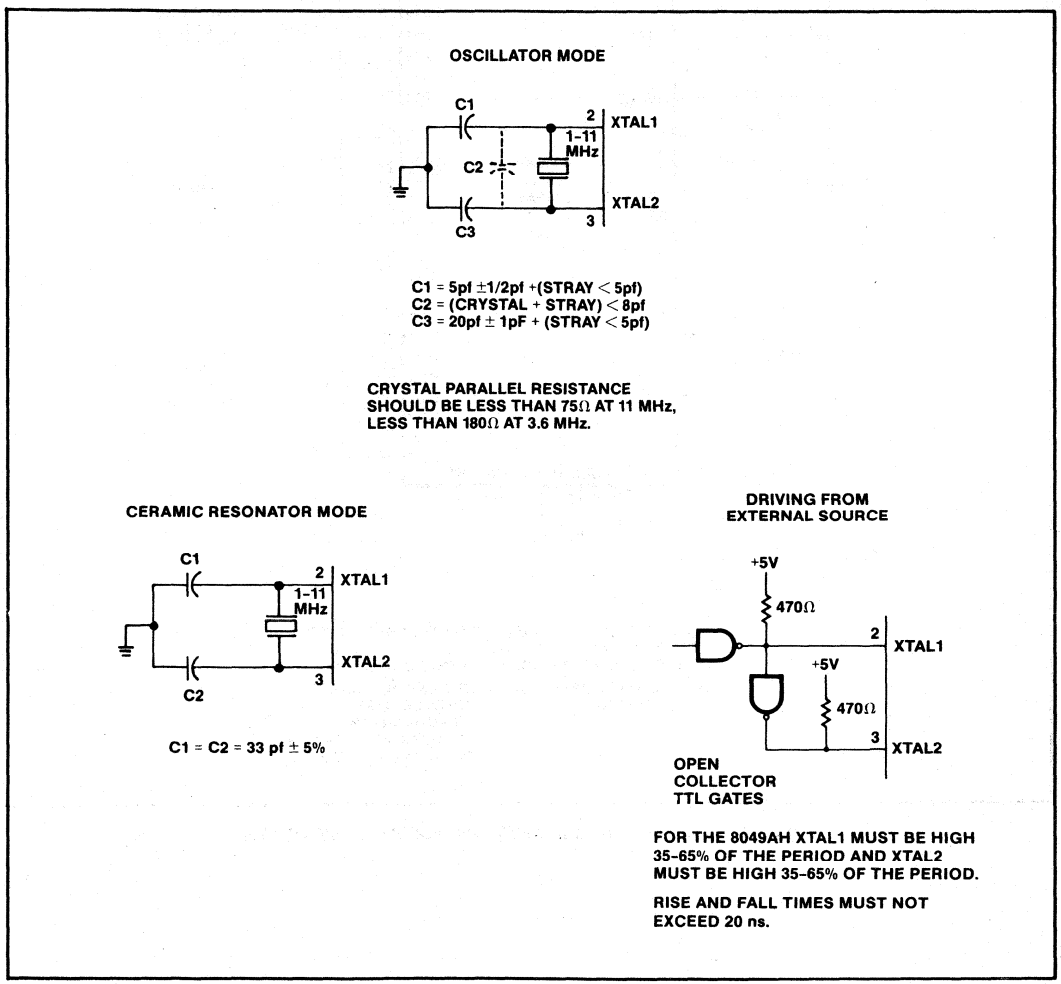
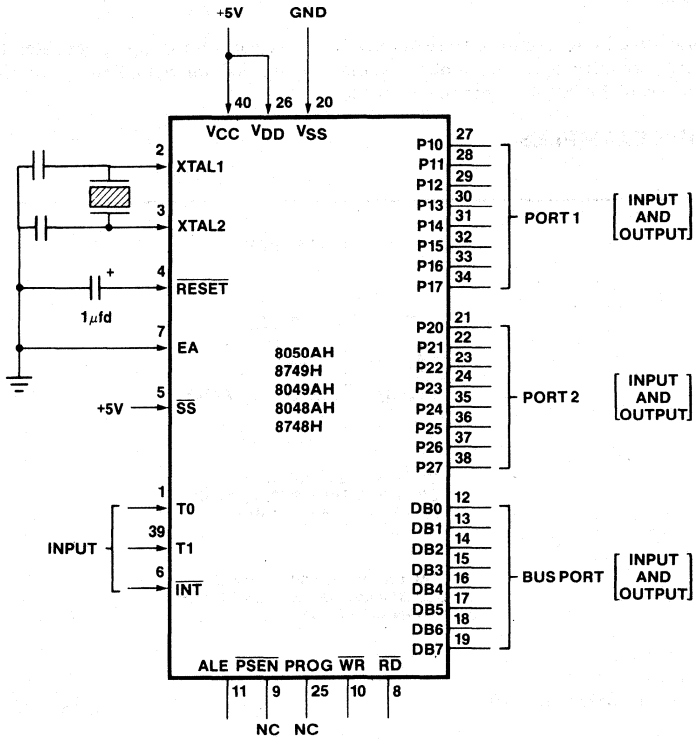


Figure 5-1. Suggested Circuits



- ALL INPUTS AND OUTPUTS (EXCEPT RESET, X1, X2) STANDARD TTL COMPATIBLE
- P1 AND P2 OUTPUTS DRIVE 5V CMOS DIRECTLY

XTAL: PARALLEL RESONANT
1 TO 11 MHz

CTS KNIGHTS MP060
CRYSTEK CY6B
OR EQUIVALENT
OR STANDARD 3.58 MHz
TV COLOR BURST XTAL

Figure 5-2. The Stand Alone 8048AH/8049AH/8748H/8749H

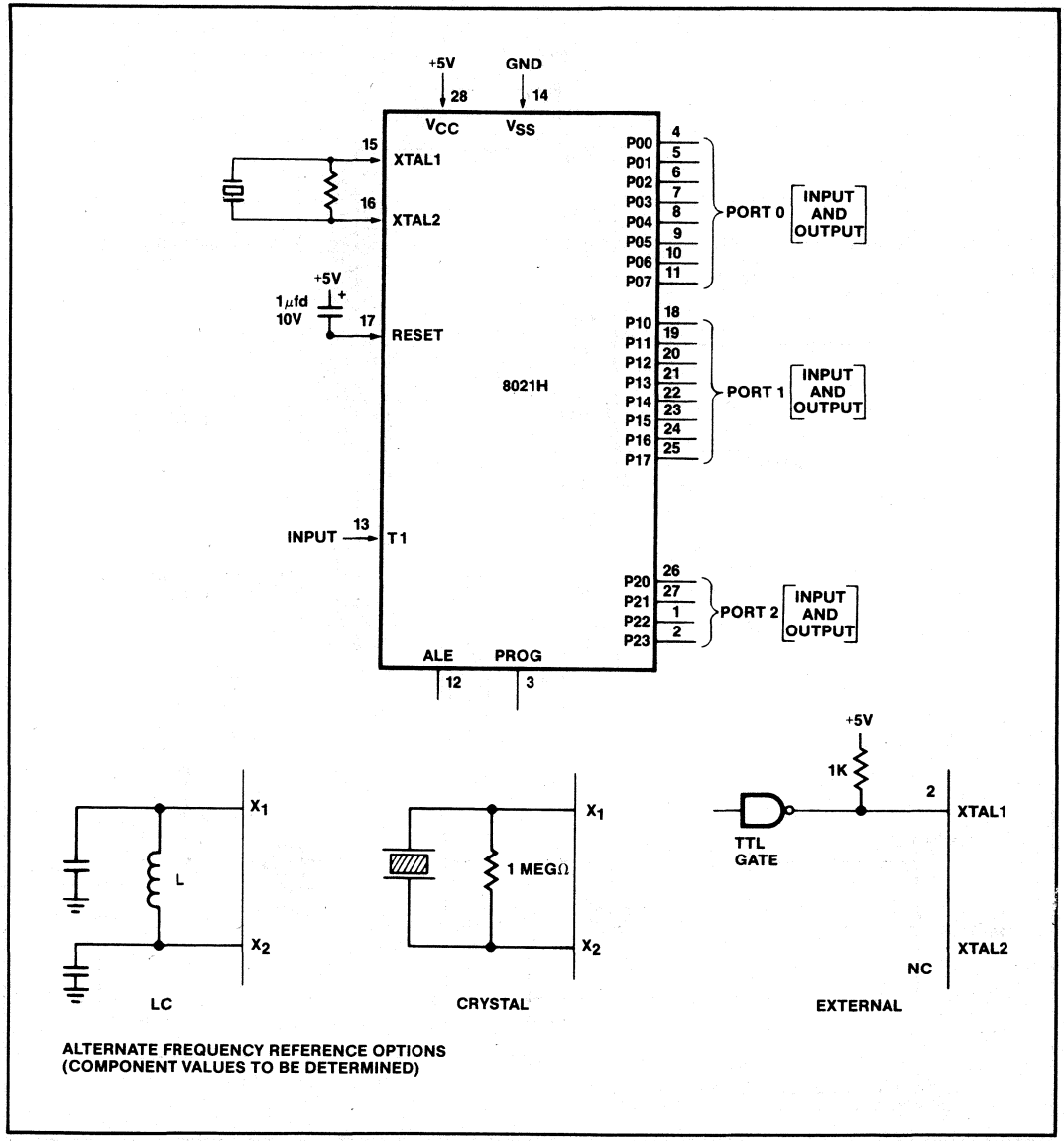


Figure 5-3. The Stand Alone 8021H

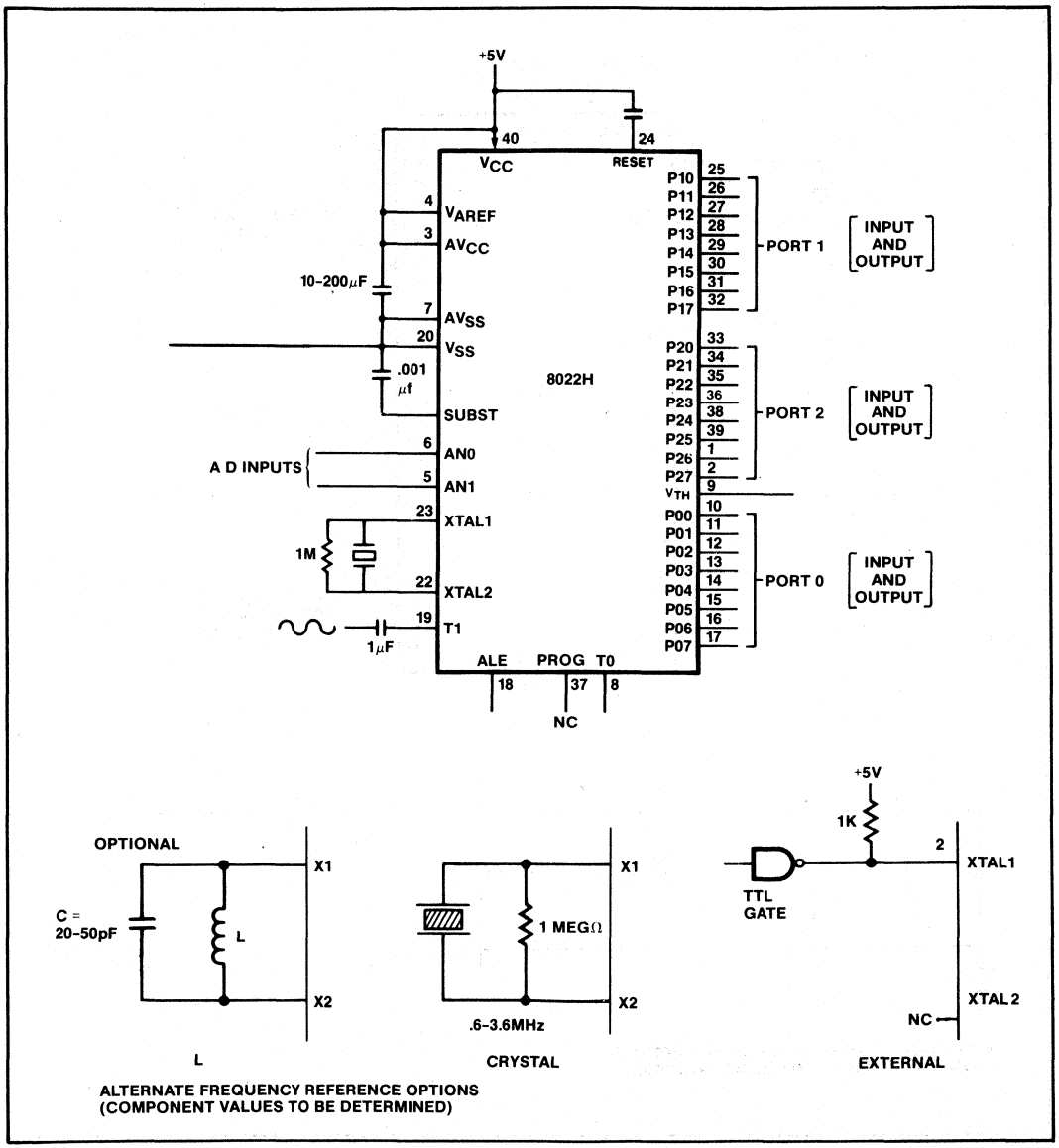


Figure 5-4. The Stand Alone 8022H

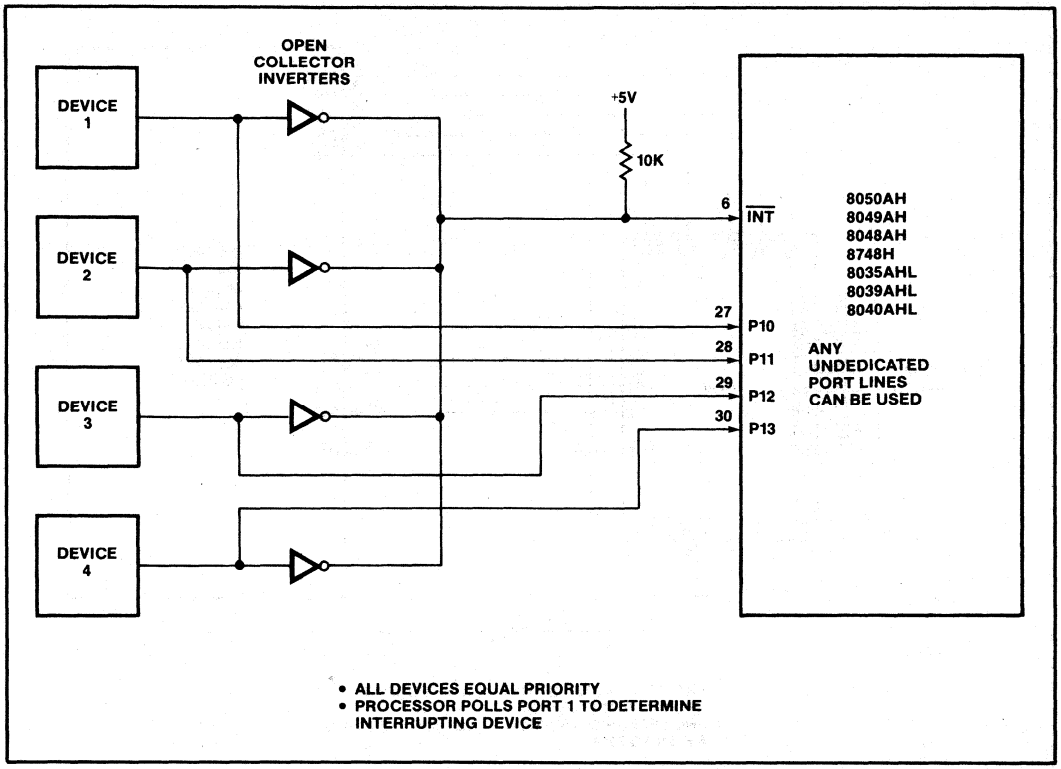


Figure 5-5. Multiple Interrupt Sources

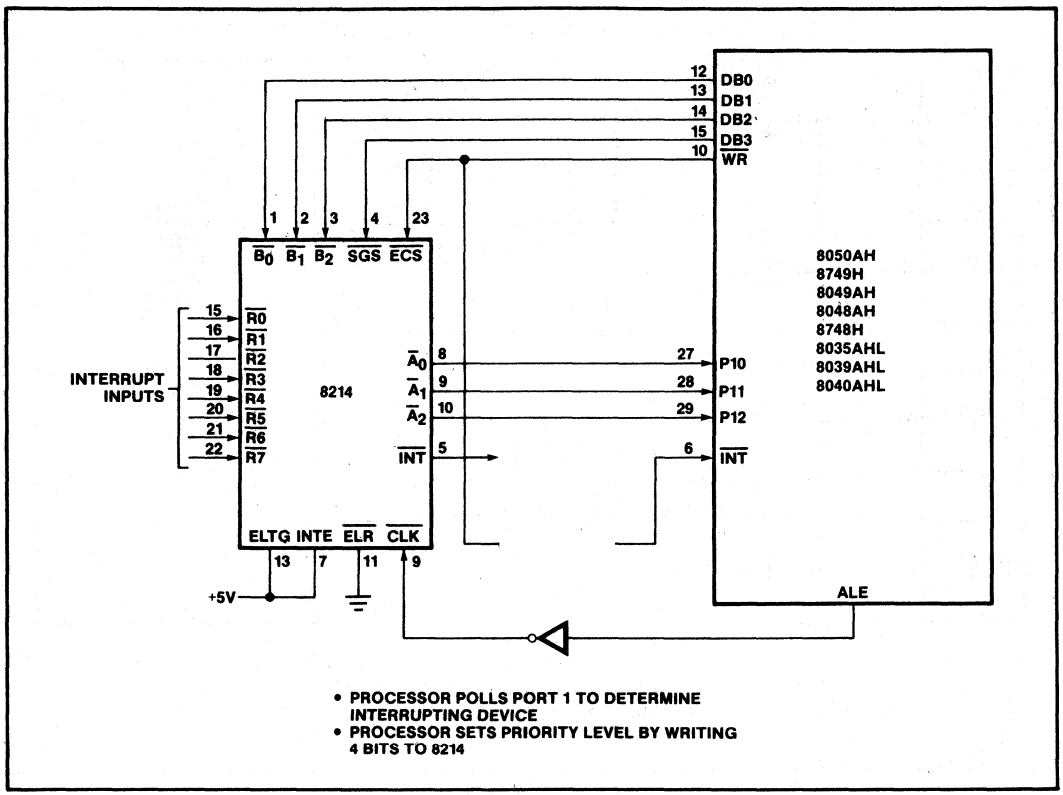


Figure 5-6. Multiple Interrupts with Priority Levels

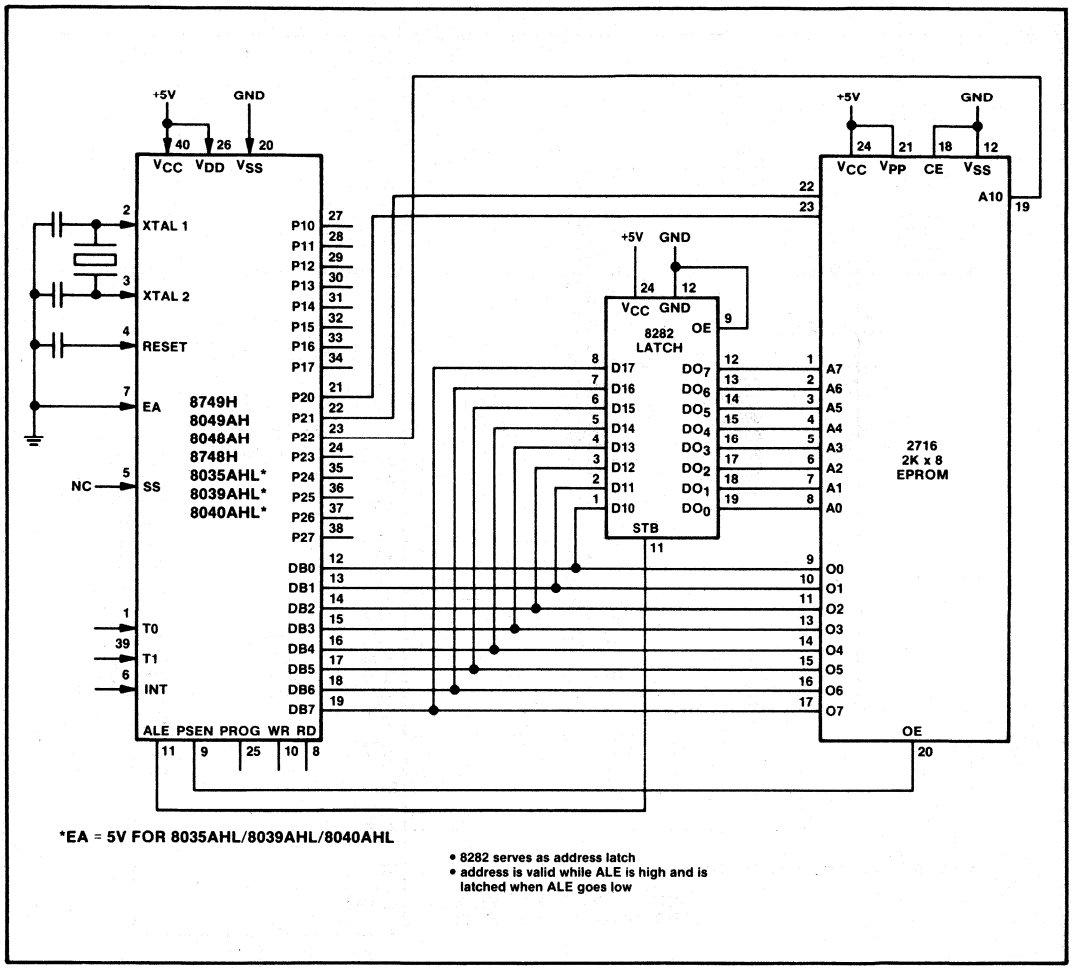


Figure 5-7. External Program Memory

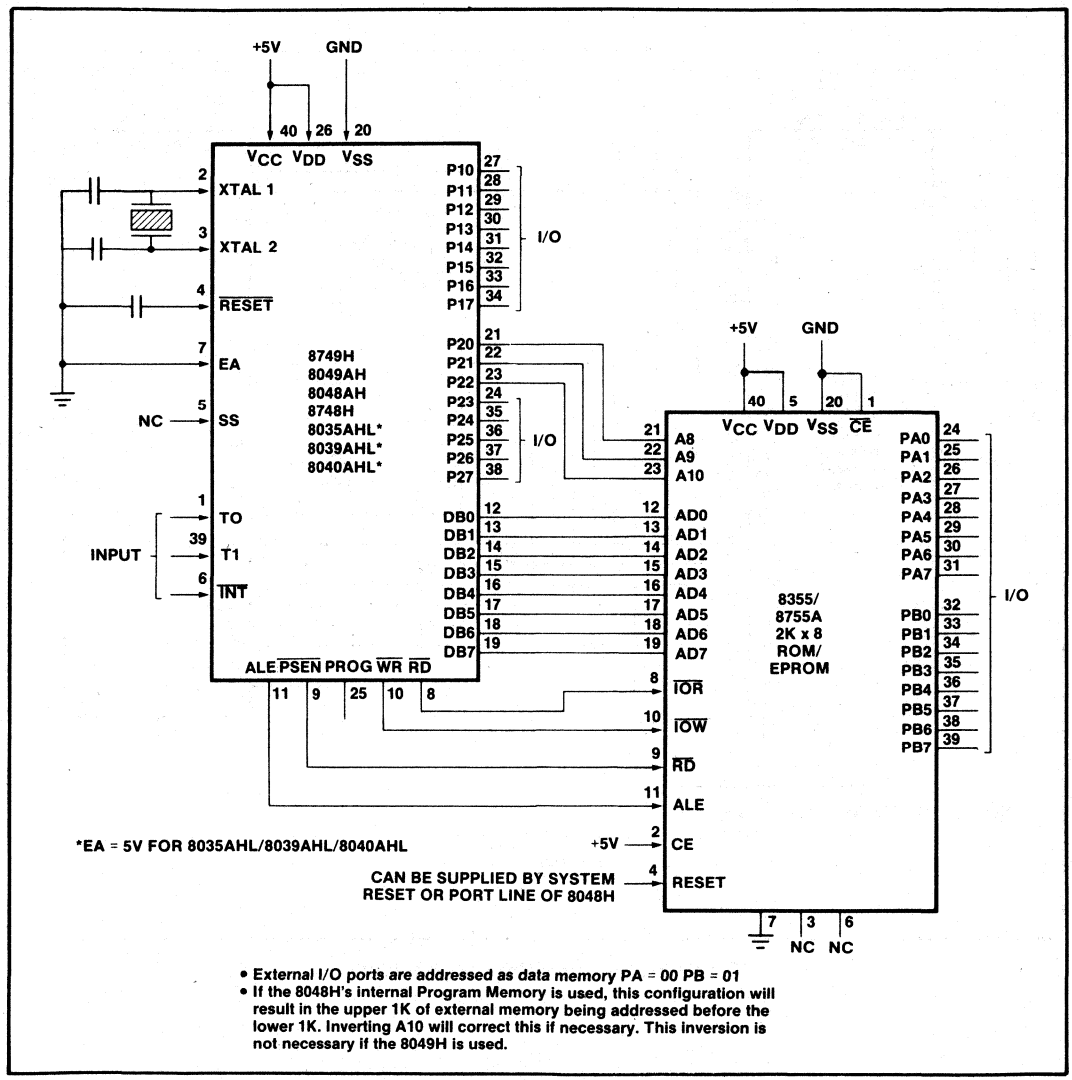


Figure 5-8. Adding a Program Memory and I/O Expander

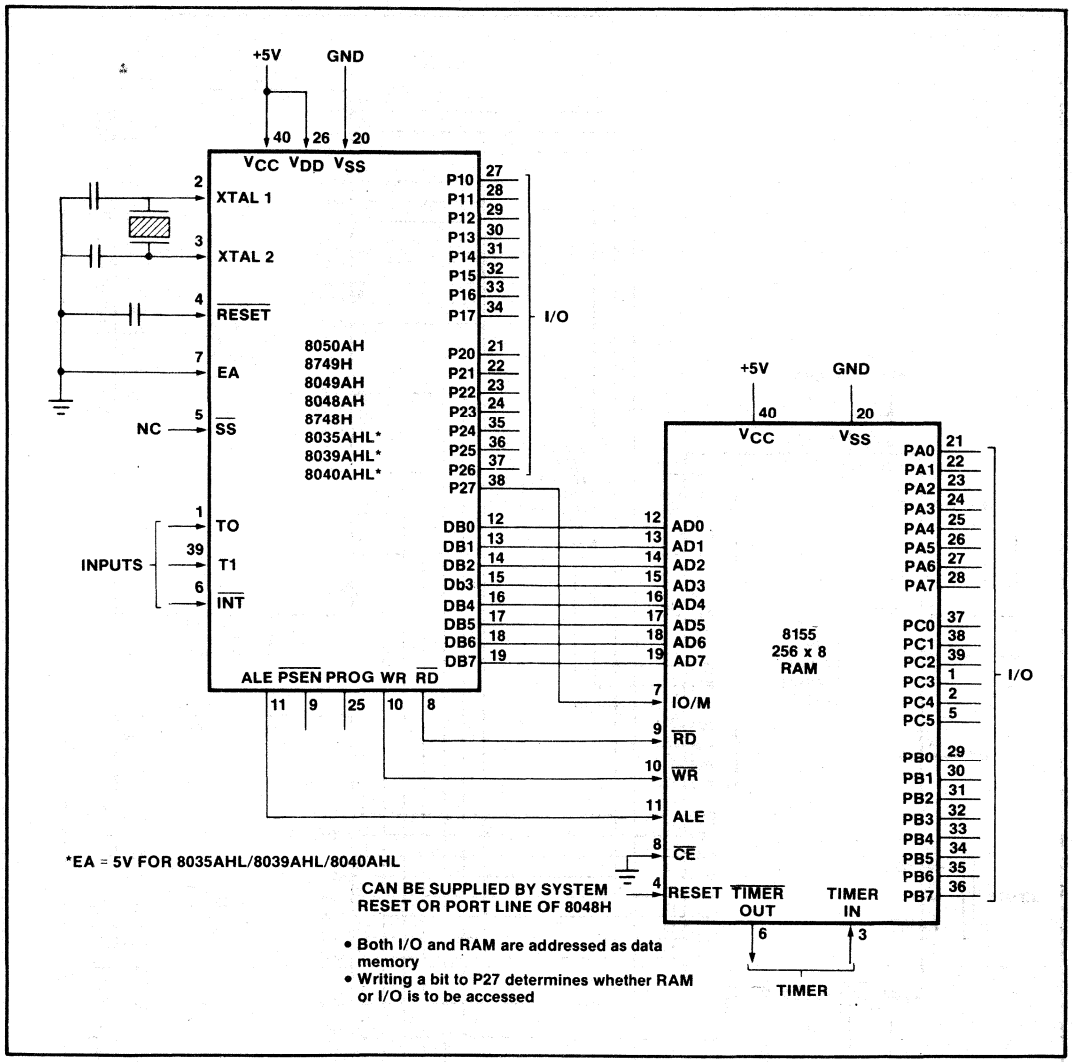


Figure 5-9. Adding a Data Memory and I/O Expander

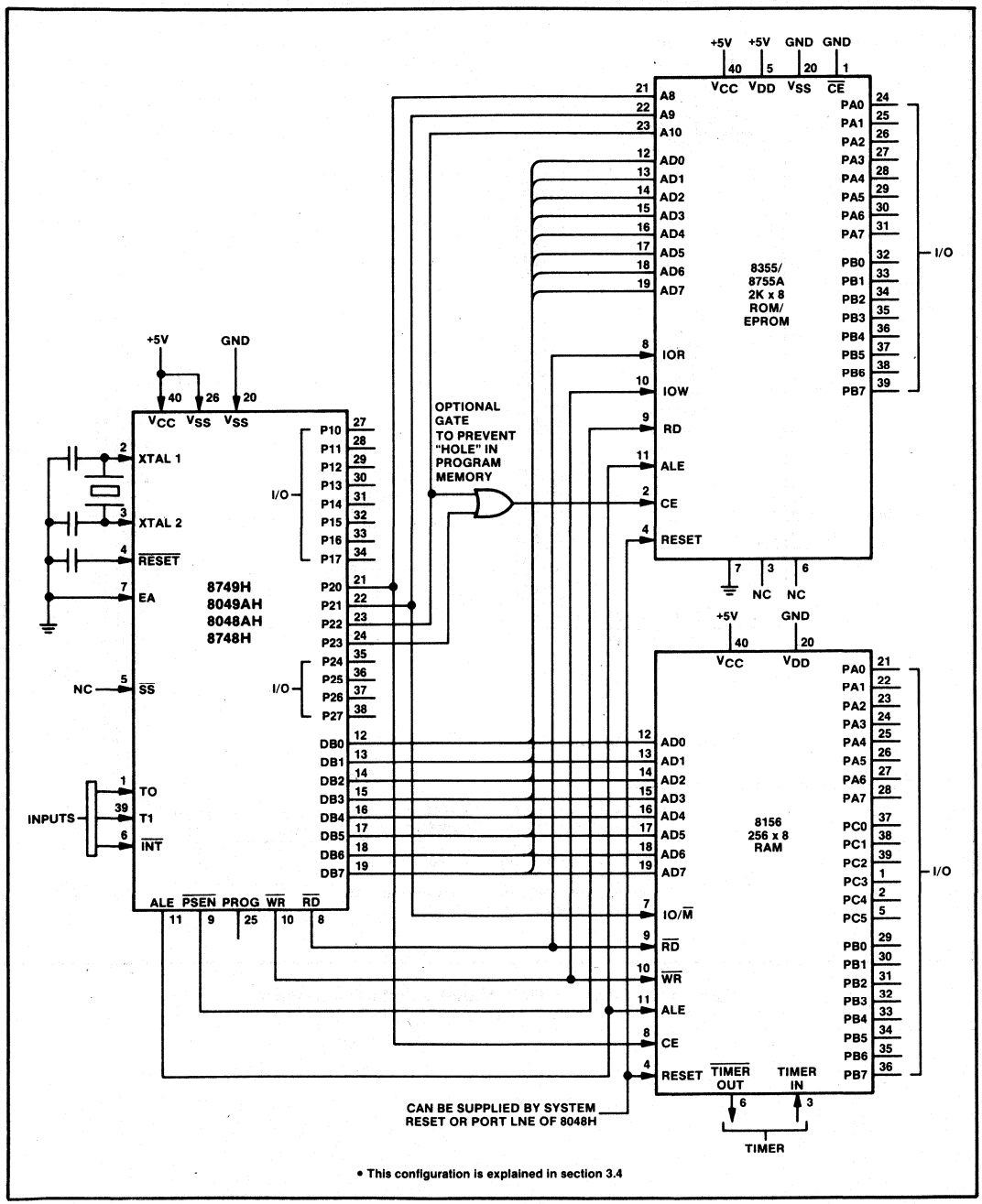


Figure 5-10. The Three-Chip System

5.2 I/O EXPANSION TECHNIQUES

The following are several examples of how the basic I/O capability of the MCS-48 microcomputers can be easily expanded externally using either the 8243 I/O expander device or standard logic circuits. These techniques can be used whenever the combination memory I/O expanders illustrated on the preceding pages are not required.

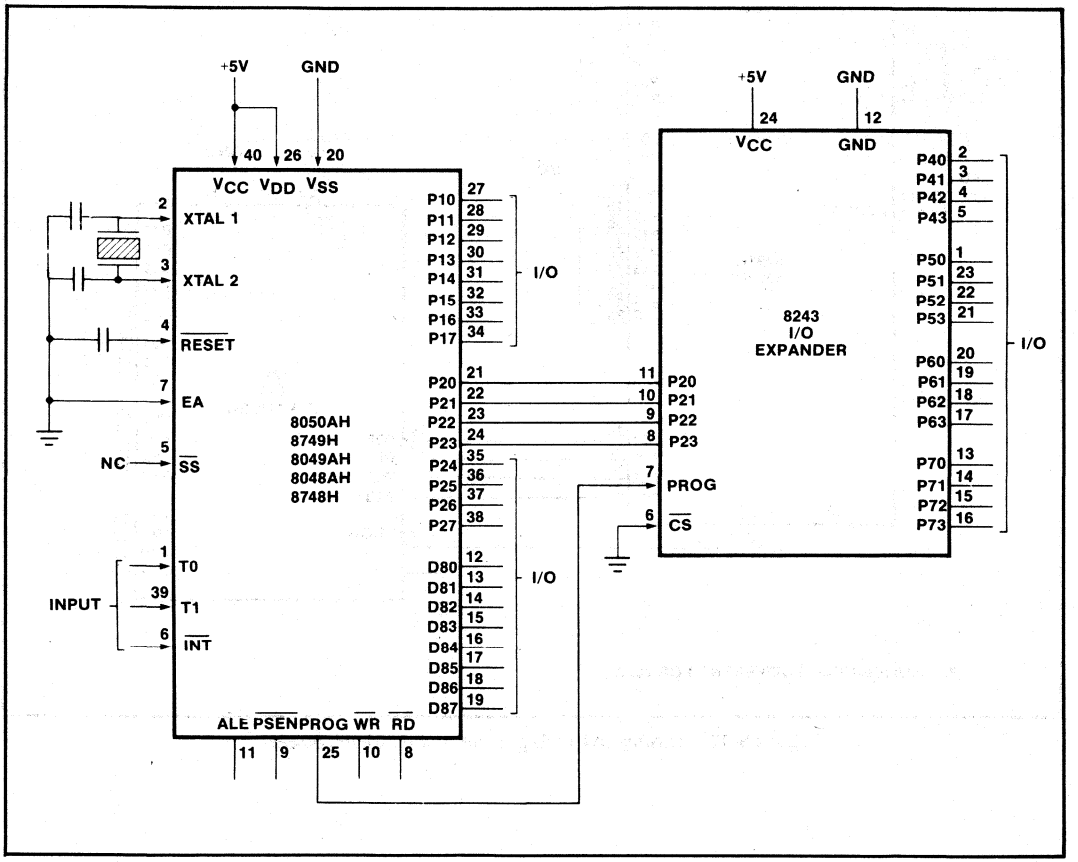


Figure 5-11. Adding an I/O Expander

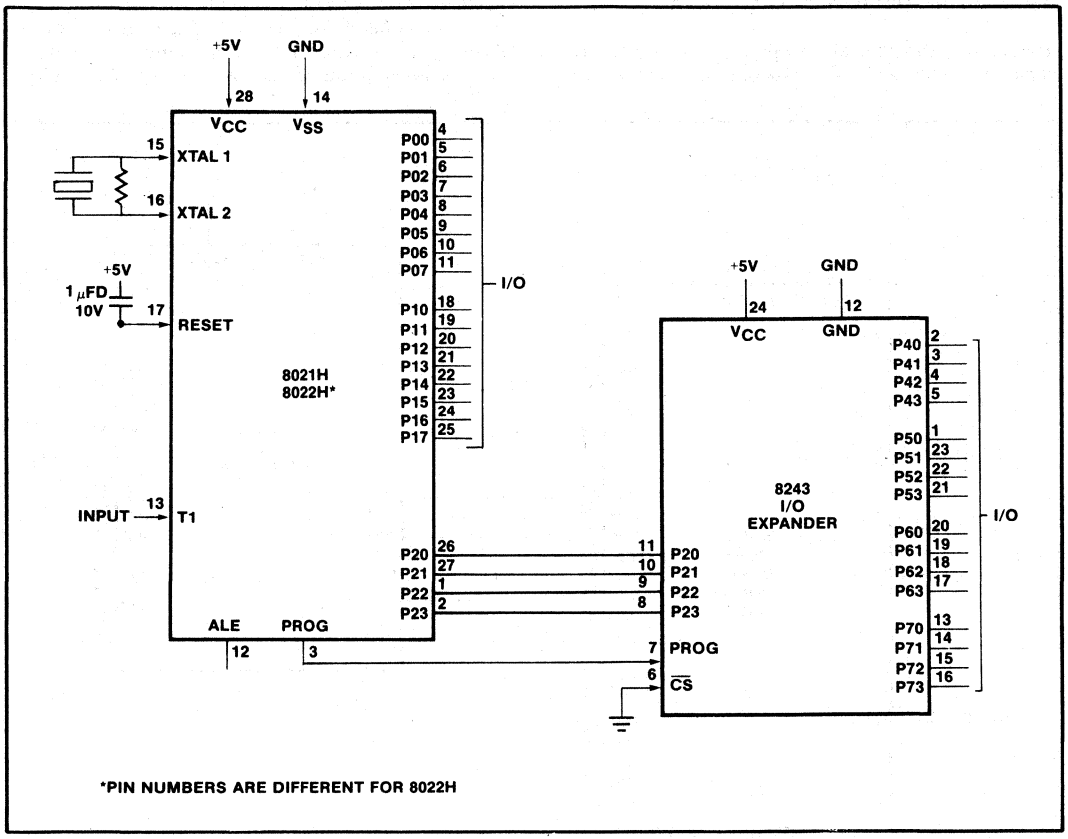


Figure 5-12. Adding an I/O Expander to the 8021H, 8022H

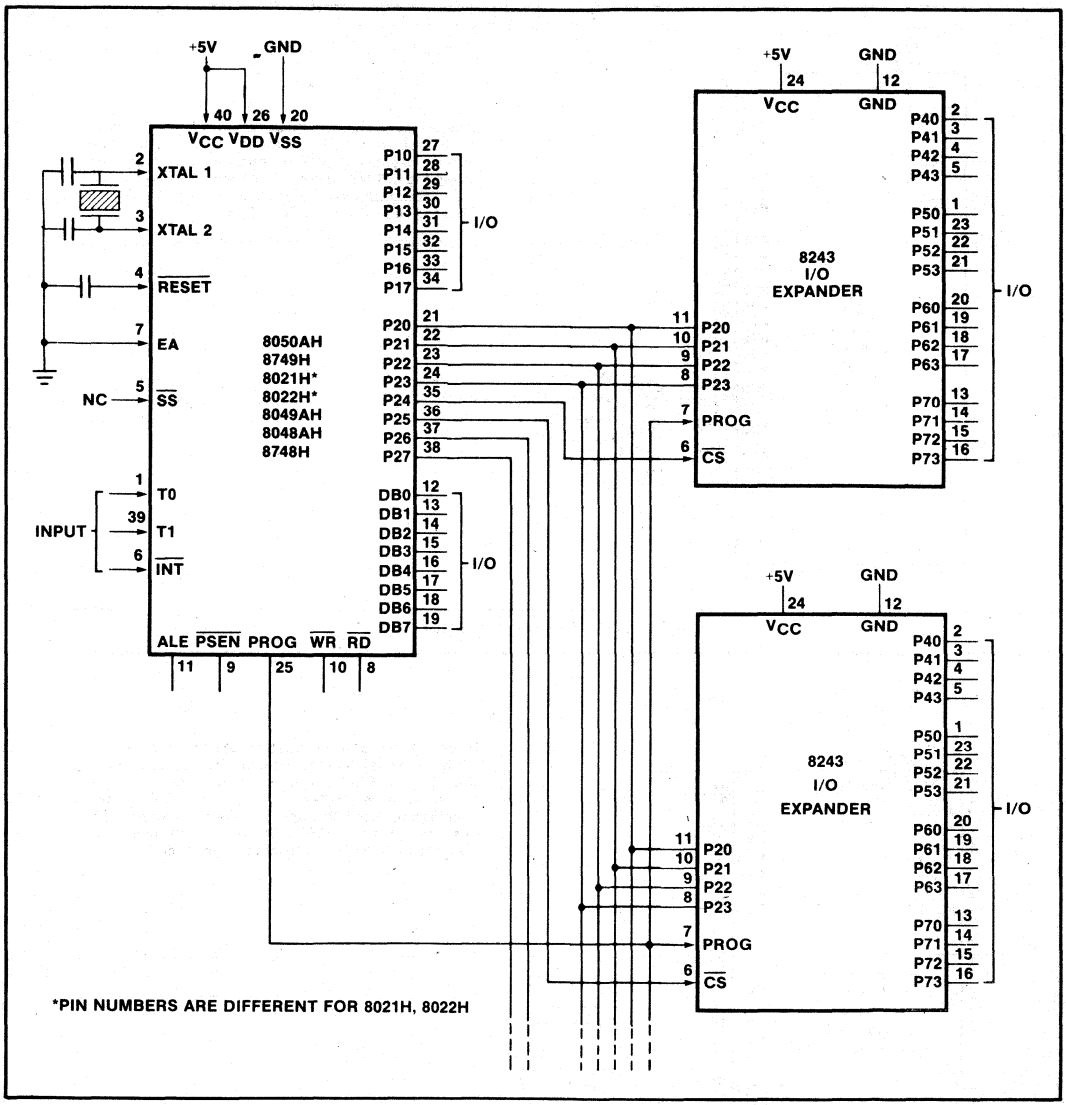
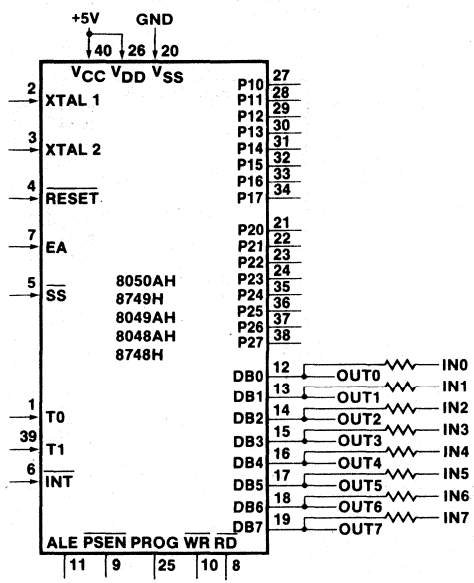
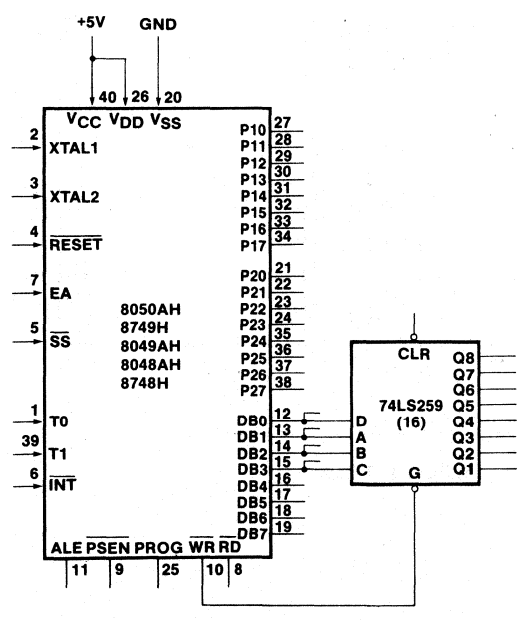


Figure 5-13. Adding Multiple I/O Expanders



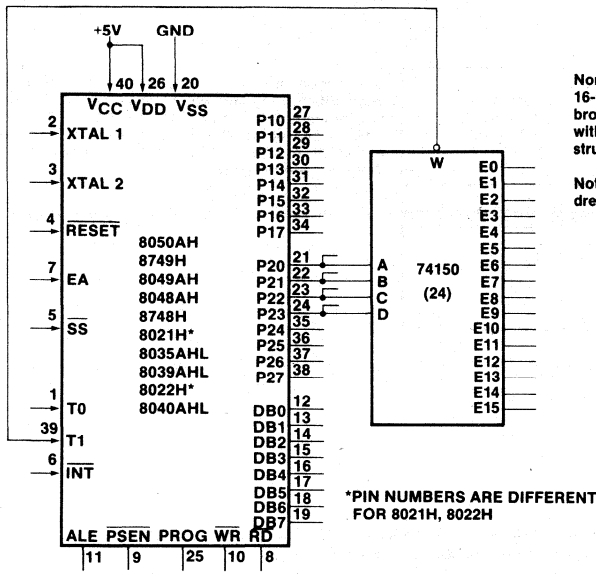
The bus is normally used as an output port. To use it as an input port the bus is put in the high impedance state using the MOVX instruction and then read using the INS instruction. The resistor valve chosen is a function of the output loading and the characteristics of the input signals.

Figure 5-14. Adding 8 Input Lines



Individual bits of the 74LS259 eight-bit addressable latch can be set or reset using the OUTL instruction. During the OUTL operation bit zero of the accumulator is written into the bit of the latch specified by bits 1 through 3 of the accumulator. In this configuration DB^0-DB^7 will be momentarily disturbed while the external latch is loaded.

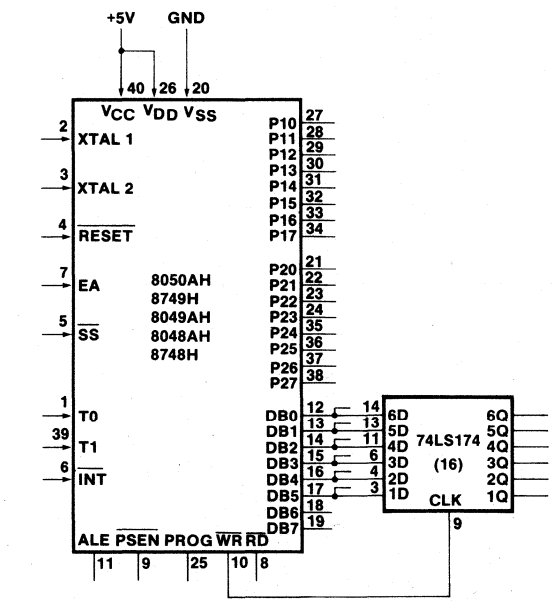
Figure 5-15. Adding 8 Output Lines



Normal I/O port is used to select an address for the 16-to-1 multiplexer. The output of the multiplexer is brought into a test input. Eight inputs could be added with a 74LS151 8-to-1 multiplexer using the same structure.

Note: If external program memory is being addressed, use P24-P27 instead of P20-P23.

Figure 5-16. Adding 16 Input Lines



The latch can be loaded with the OUTL instruction. After the latch is loaded the BUS output state can be modified with the ANL BUS, # DATA and the ORL BUS, # DATA. The OUTL generates a WR strobe; ANL and ORL do not. In this configuration DB0-DB7 will be momentarily disturbed while the external latch is loaded.

Figure 5-17. Adding 6 Output Lines

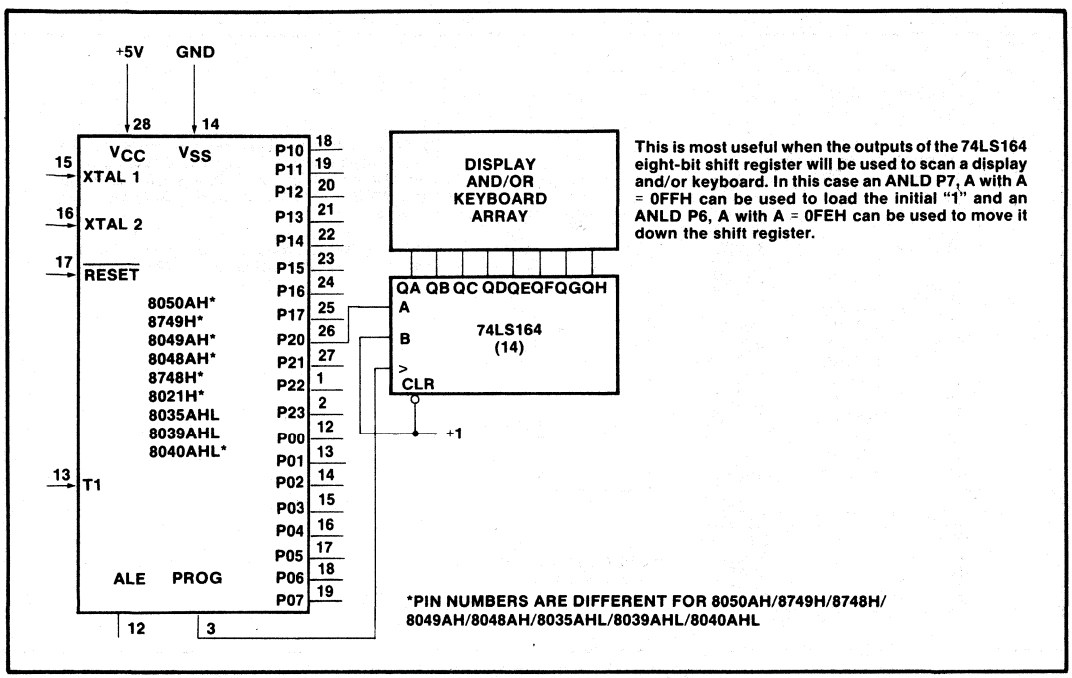


Figure 5-18. Adding Output for Keyboard/Display Scanning

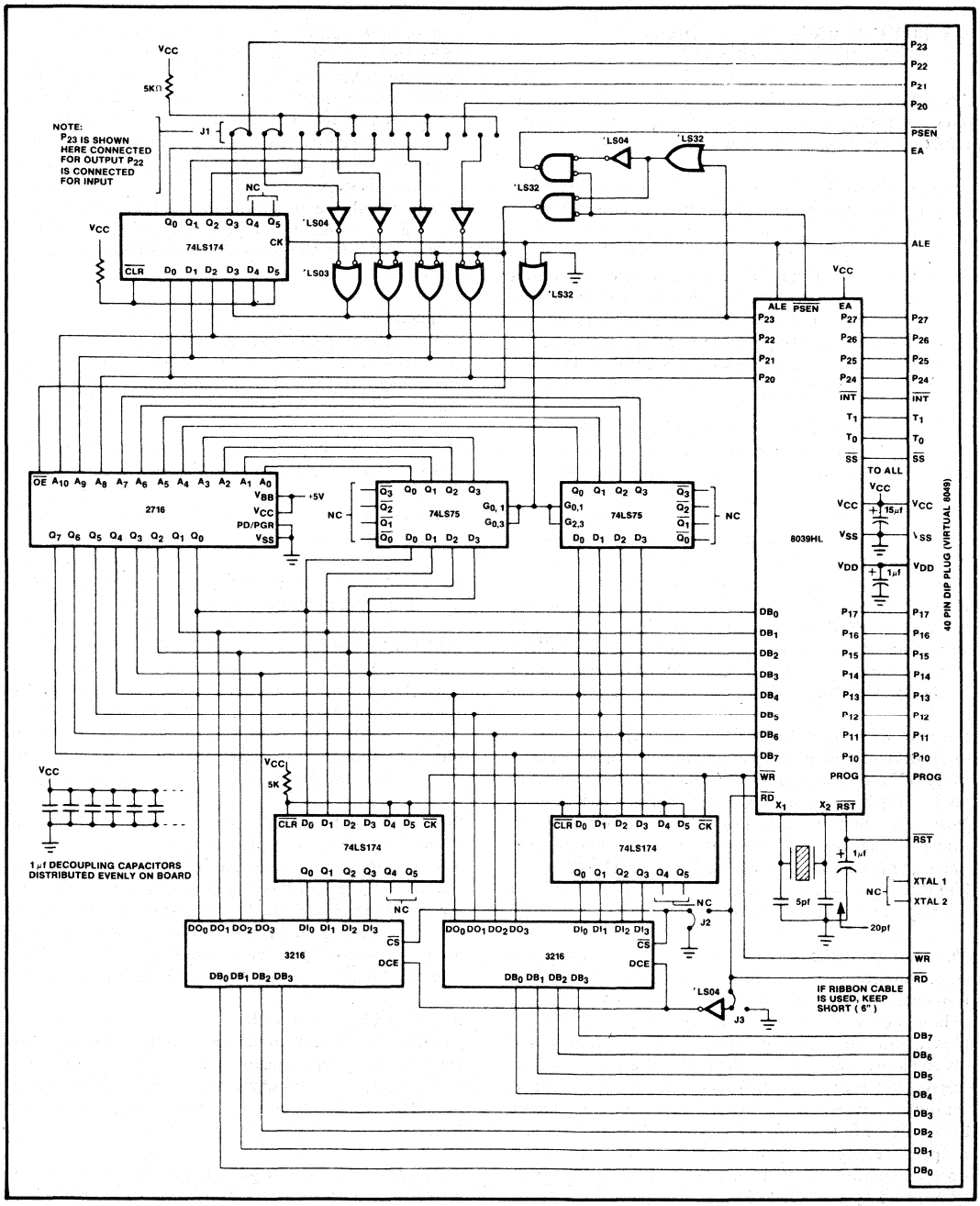


Figure 5-19. 8049AH Emulator Circuit

5.3 8049AH EMULATOR CIRCUIT DESCRIPTION—6 MHZ

The following is an explanation of a circuit which emulates the operation of an Intel 8049AH using a standard EPROM for program storage.

With the 8049AH, software may be developed by running external program memory, but doing so requires the use of the bus and P_{23}–P_{20} to access this memory.

The circuit shown may be used to restore the normal functioning of these twelve I/O pins. The circuit consists of an 8039AHL CPU, 2716 EPROM, two 8216 bidirectional bus drivers, and eight other 7400 Series Low-Power Schottky TTL packages. The whole assembly can be built on a 2¾" x 4" board.

A cable coming off the board can be terminated by a forty-pin plug which may be inserted directly into the CPU socket intended for the 8049AH in a system undergoing design or testing. Alternatively, a pattern of forty pins extending below the board can be used to plug the board directly into the system undergoing testing, "piggy-back" fashion. The emulator board may be configured in various ways so that the 40 pin plug is the logical equivalent of an 8049AH in every legal operating mode. (In the following explanation of the operation of the circuit, an asterisk appearing before a signal or pin number—as in *$\overline{\text{PSEN}}$—refers to that pin on the "virtual 8049AH" represented by the forty-pin plug).

Since the CPU is identical with the 8049AH in all respects other than its lack of program memory, most of the pins of the 8039AHL are simply connected directly to the corresponding pins of the forty-pin plug. These include all of Port 1, the high order bits of Port 2, the test pins, etc. Signals which are emulated with additional logic include the rest of Port 2, DB_7–DB_0, *$\overline{\text{PSEN}}$, etc. $\overline{\text{RD}}$, $\overline{\text{WR}}$, ALE, and $\overline{\text{PSEN}}$ are obtained from the 8039AHL, but are also used by the emulation circuitry.

The EA input of the 8039AHL is hard-wired high so all instruction fetches are made from the 2716. Two 74LS75 four-bit latches gated by the buffered ALE signal are used to hold the lower eight bits of address from the time-multiplexed data bus. Since the Bus is being used for fetching instructions, data latched to the Bus will be lost on the next instruction fetch. Two 74LS174 latches are used to retain the output data when a bus write is executed. These latches are triggered by the trailing edge of the WR pulse, so their outputs are glitch free. Since logical operations to the bus do not generate a WR strobe, the "OUTL BUS, #data" "ANL BUS,#" and "ORL BUS,#" instructions may not be used, though they do function properly with the other ports.

The two 8216 bidirectional bus drivers normally buffer the latched bus contents to the DB pins of the virtual 8049AH. When an "INS A, BUS" instruction is executed, they buffer the input signals on to the emulator data bus. Thus, the circuit is designed to use the Bus for both latched output and strobed input. If DB_7–DB_0 of the 8049AH are to be used solely for input data, J2 and J3 may be changed from what is shown in the Figure, so that DB_7–DB_0 act as high impedance inputs and the 8216s are enabled only when the read operation is performed. If the bus is to be used only for latched output, the 8216s can be omitted entirely.

Bidirectional data transfers which require the transfer of address information as well as data, such as to and from external data memory, require removal of the 8216s and replacement with 16-pin jumper blocks on which the DB_X pins are connected with the respective DO_X pins.

The lower four bits of Port 2 are also used in fetching instructions from the 2716, in addition to their use in input or output pins in the user's system. In configuring the emulator for a particular application, the user must dedicate each of these as either an input or output pin and connect jumper set J1 accordingly. Any mix of input and output pins is allowed. At the beginning of each instruction fetch, the last data written to P2 will be present on P_{23}–P_{20} at the rising edge of ALE and will be latched by a 74LS174. The latched data may be connected through the jumpers to those pins which will be used as outputs on the 8049AH. Emulator pins used as inputs should be pulled above 2.0V for a logic "one". If this is not the case, i.e., if switches to Ground are to be read, 50K pullup resistors should be added to the circuit on each input. They were omitted from the diagram to minimize input loading.

Pins which will be used as inputs may be connected to the input of an OR gate formed of inverters and open-collector NAND gates. The input signals will be relayed directly to the 8039AHL and will be read by an "IN A,P2" instruction. But when $\overline{\text{PSEN}}$ is low, the NAND outputs are forced off, allowing the 8039AHL pins to be used for high-order program addressing. Open-collector outputs are needed to prevent line contention when $\overline{\text{PSEN}}$ is not low.

If 8243s will be used in the final system, the low order pins of Port 2 must be connected directly to the plug. This may be done by replacing the Port 2 latch with four jumpers connecting the inputs to the outputs. The NANDs should be removed or disabled by grounding the common NAND inputs.

The cluster of three OR gates is used to enable the on-board 2716 and generate the *$\overline{\text{PSEN}}$ signal, each of which is a function of $\overline{\text{PSEN}}$, *EA, and the high order bit of the

program counter. Thus *PSEN is generated, forcing an off-board read, only when a jump has been made to Memory Bank 1 or when *EA is brought high. If this feature is to be used to address off-board memory, DB_7–DB_0 may not be used for normal I/O. The 8216s and 74LS174 must be replaced with jumper blocks and the open collector NAND gates disabled, as explained above. The same changes are required to operate the board in single step mode.

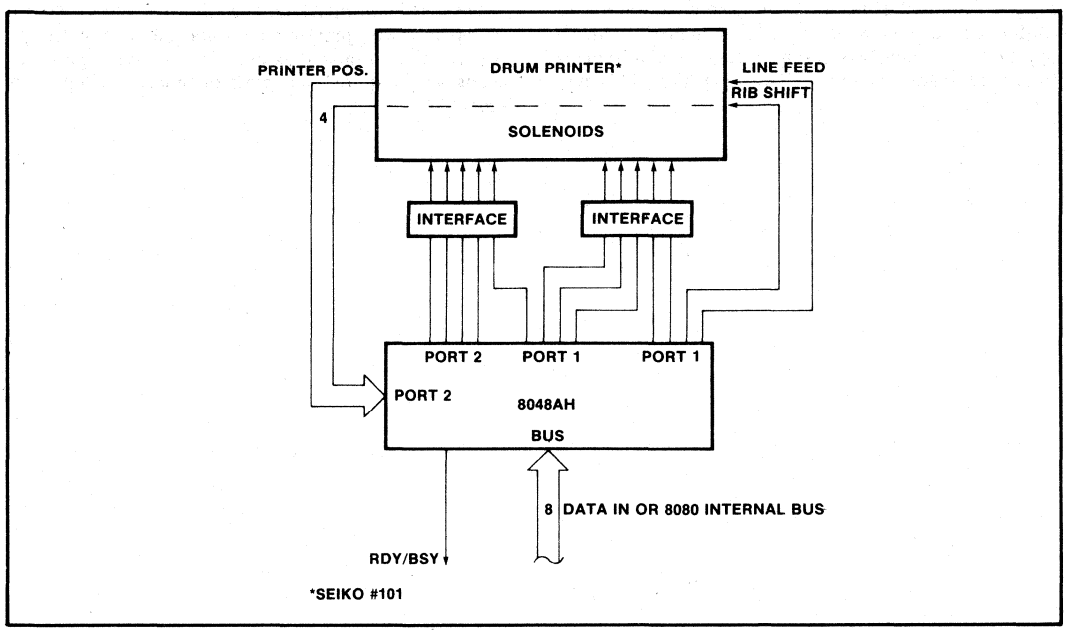


Figure 5-20. 8048AH Interface to Drum Printer

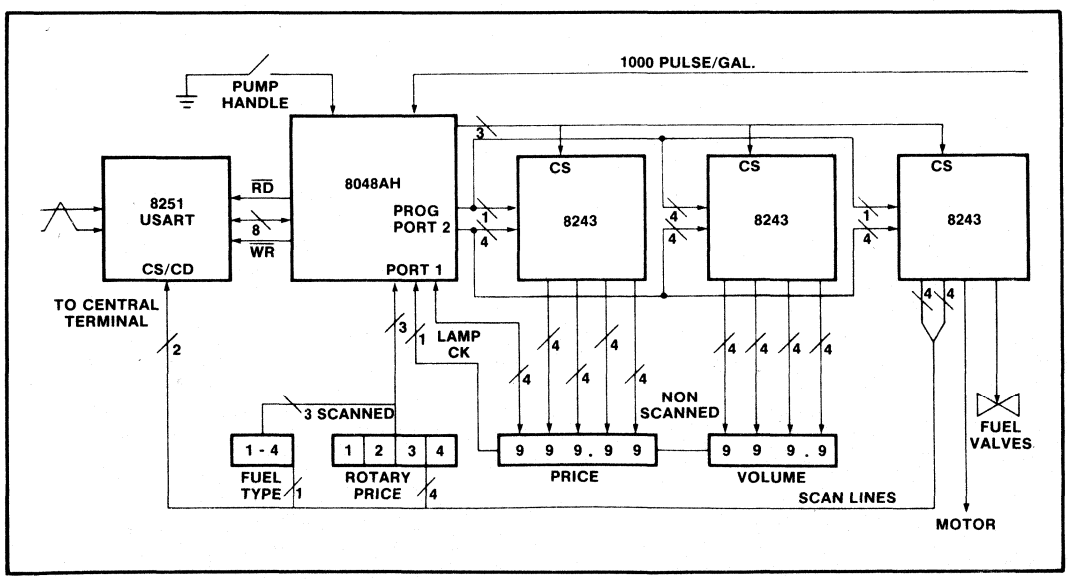


Figure 5-21. MCS-48 Gas Pump

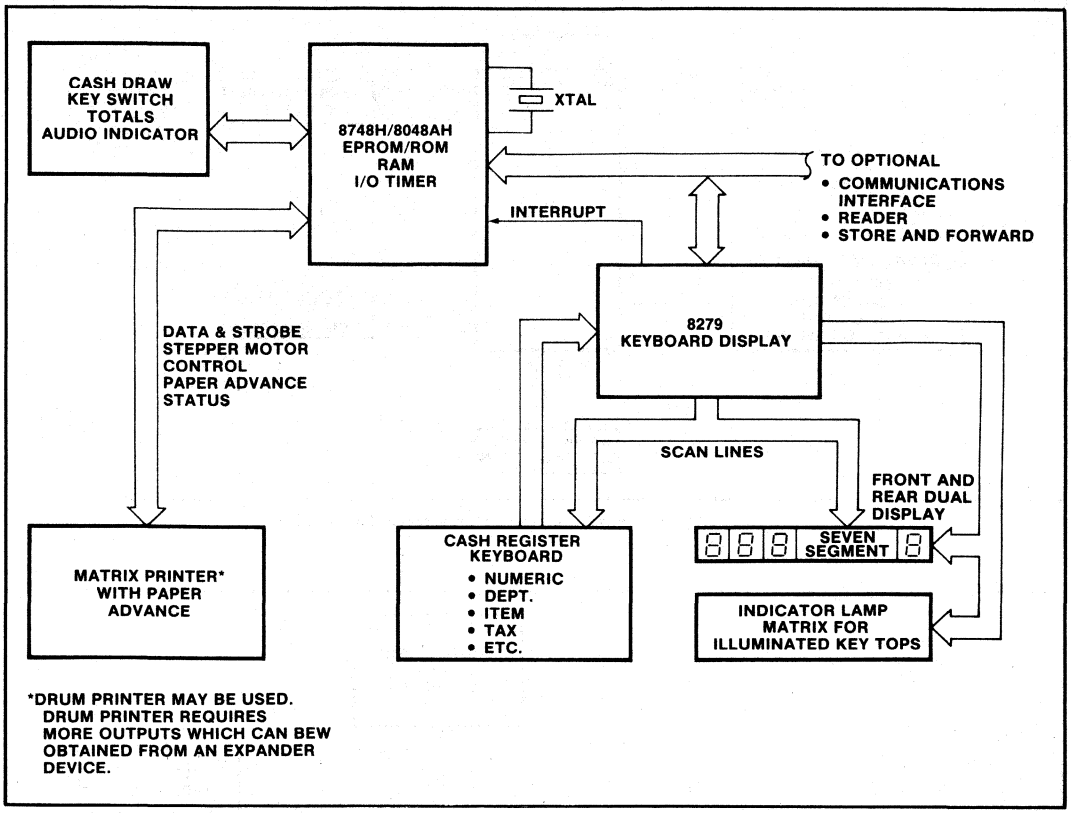


Figure 5-22. Low Cost Point of Sale Terminal

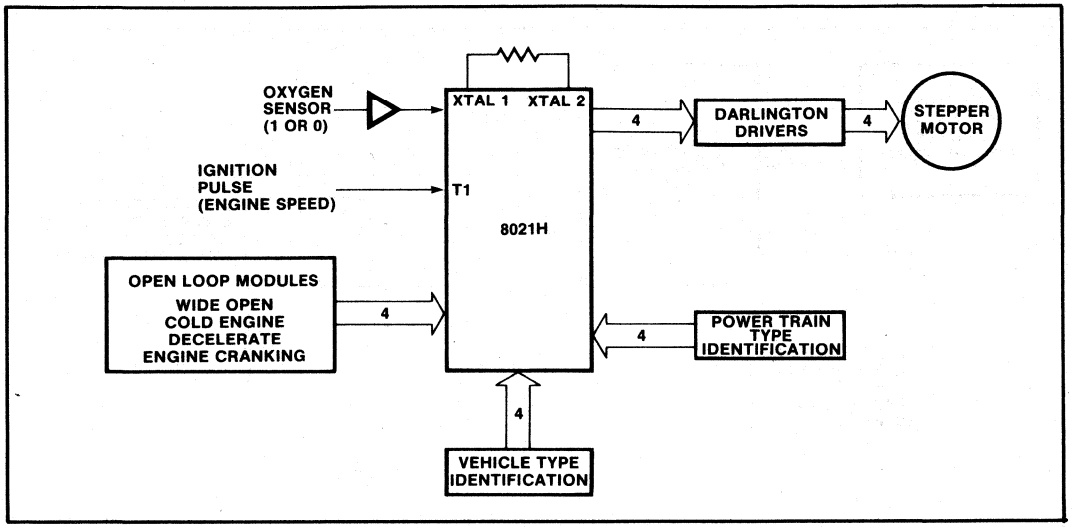


Figure 5-23. Simple Feedback Carburetor Controller

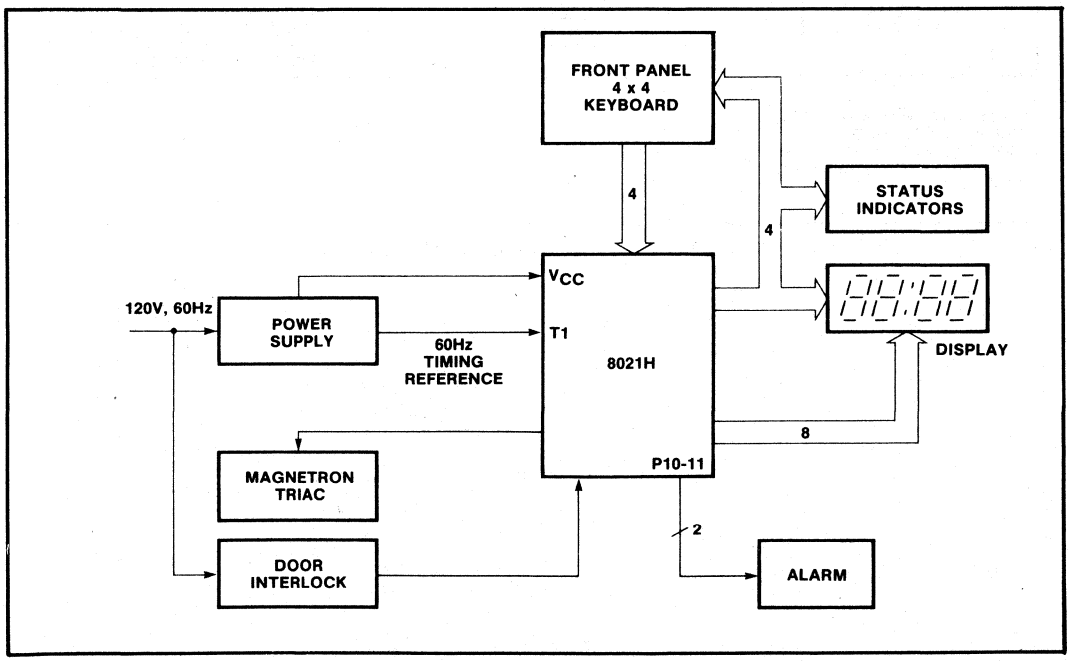


Figure 5-24. Microwave Oven Controller

5.4 SOFTWARE EXAMPLES

The following routines are written as subroutines. R0 and R1 are used as data pointers, R2 is used as an extension of the accumulator and R3 is used as a loop counter.

```
                    RX0 = R0
                    AEX = R2
DOUBLE ADD
DADD:   DEC   RX0       ;GET LOW BYTE AND ADD TO A
        ADD   A,@RX0
        INC   RX0       ;GET HI BYTE AND ADD TO AEX
        XCH   A,AEX
        ADDC  A,@RX0
        XCH   A,AEX
        RET             ;RETURN
DOUBLE SUBTRACT
DMIN:   DEC   RX0       ;GET LOW BYTE AND SUB FROM A
        CPL   A
        ADD   A,@RX0
        CPL   A
        INC   RX0       ;GET HI BYTE AND SUB FROM AEX
        XCH   A,AEX
        CPL   A
        ADDC  A,@RX0
        CPL   A
        XCH   A,AEX
        RET             ;RETURN
DOUBLE LOAD
DLD:    DEC   RX0       ;GET LOW BYTE AND PLACE IN A
        MOV   A,@RX0
        INC   RX0       ;GET HI BYTE AND PLACE IN AEX
        XCH   A,AEX
        MOV   A,@RX0
        XCH   A,AEX
        RET             ;RETURN
DOUBLE STORE
DST:    DEC   RX0       ;MOVE A INTO LOW BYTE
        MOV   @RX0,A
        INC   RX0       ;MOV AEX INTO HIGH BYTE
        XCH   A,AEX
        MOV   @RX0,A
        XCH   A,AEX
        RET             ;RETURN
```

DOUBLE EXCHANGE

```
DEX:    DEC   RX0       ;EXCHANGE A AND LOW BYTE
        XCH   A,@RX0
        INC   RX0       ;EXCHANGE AEX AND HIGH BYTE
        XCH   A,AEX
        XCH   A,@RX0
        XCH   A,AEX
        RET             ;RETURN
```

DOUBLE LEFT LOGICAL SHIFT

```
LLSH:   RLC   A         ;SHIFT A
        XCH   A,AEX     ;SHIFT AEX
        RLC   A
        XCH   A,AEX
        RET             ;RETURN
```

DOUBLE RIGHT LOGICAL SHIFT

```
RLSH:   XCH   A,AEX     ;SHIFT AEX
        RRC   A
        XCH   A,AEX
        RRC   A         ;SHIFT A
        RET             ;RETURN
```

DOUBLE RIGHT ARITHMETIC SHIFT

```
RASH:   CLR   C         ;SET CARRY
        CPL   C
        XCH   A,AEX     ;IF AEX[7]< >1 THEN
        JB7   $+3
        CLR   C         ;CLEAR CARRY
        RRC   A         ;SHIFT C INTO AEX
        XCH   A,AEX
        RRC   A         ;SHIFT A
        RET             ;RETURN
```

5.4.1 Single Precision Binary Multiply

This routine assumes a one-byte multiplier and a one-byte multiplicand. The product, therefore, is two-bytes long.

The algorithm follows these steps:

1) The registers are arranged as follows:
   ACC — 0
   R1 — Multiplier
   R2 — Multiplicand
   R3 — Loop Counter (=8)

   The Accumulator and register R1 are treated as a register pair when they are shifted right (see Step 2)

2) The Accumulator and R1 are shifted right one place, thus the LSB of the multiplier goes into the carry.

3) The multiplicand is added to the accumulator if the carry bit is a 'one'. No action if the carry is a 'zero'.

4) Decrement the loop counter and loop (return to Step 2) until it reaches zero.

5) Shift the result right one last time just before exiting the routine.

*The result will be found in the Accumulator (MS Byte) and R1 (LS Byte).

BINARY MULTIPLY

```
BMPY:   MOV  R3,#08H   ;SET COUNTER TO 8
        CLR  A         ;CLEAR A
        CLR  C         ;CLEAR CARRY BIT
BMPI:   RRC  A         ;DOUBLE SHIFT RIGHT ACC & R1
        XCH  A,R1      ;INTO CARRY
        RRC  A
        XCH  A,R1
        JNC  BMP3      ;IF CARRY = 1 ADD, OTHERWISE DON'T
        ADD  A,R2      ;ADD MULTIPLICAND TO ACCUMULATOR
BMP3:   DJNZ R3,BMPI   ;DECREMENT COUNTER AND LOOP IF 0
        RRC  A         ;DO A FINAL RIGHT SHIFT AT THE
        XCH  A,R1      ;END OF THE ROUTINE
        RRC  A
        XCH  A,R1
```

5.4.2 Interrupt Handling

This interrupt routine assumes single level interrupt. The purpose is to store the status of the machine at the time the interrupt occurs by storing contents of all registers, accumulator, and the status word. At the end of the interrupt the state of the machine is restored and interrupts are enabled again.

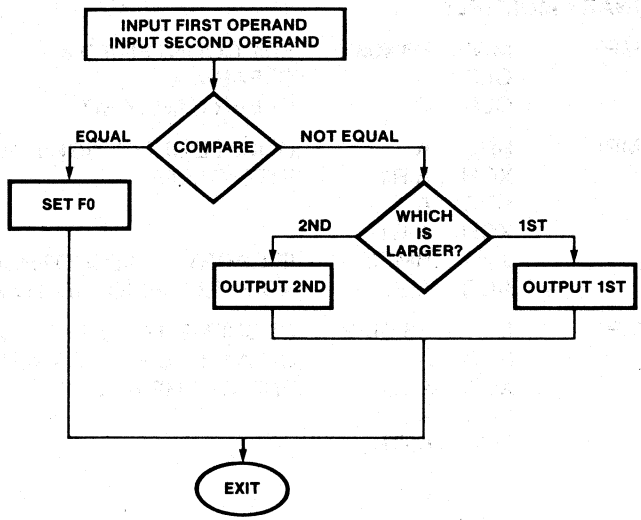
```
INTRPT: SEL  RB1       ;SAVE WORKING REGISTERS
        MOV  @R0,A     ;R0 IN ALTERNATE REGISTER
                       ;BANK CONTAINS SACC
                       ;POINTER FOR SAVING
                       ;ACCUMULATOR
          |    |
          |    |       INTERRUPT SERVICE
          |    |       ROUTINE
          |    |
        MOV  R0,SACC   ;RESTORE SACC
        MOV  A,@R0     ;RESTORE ACCUMULATOR
        RETR           ;RESTORE WORKING REGISTERS
                       ;RESTORE PSW AND
                       ;RE-ENABLE INTERRUPTS
```

5.4.3 Two-Byte Processing System

A suggested model of a processing routine takes two single byte inputs from different ports, compares them, and performs the following, depending on the result of the comparison:

(If Equal) Sets Flag and Exits

(If Not Equal) and Outputs the Larger to a Third Port




```
PROCESS:  CLR   F0       ;CLEAR F0 BIT (INITIALIZE)
          IN    A,P1     ;READ FIRST INPUT, STORE IN R0
          MOV   R0,A
          IN    A,P2     ;READ SECOND INPUT, STORE IN R1
          MOV   R1,A
          CPL   A        ;SUBCONTRACT SECOND FROM FIRST
          INC   A        ;(2's COMPLEMENT AND ADD)
          ADD   A,R0
          JZ    EQUL     ;BRANCH IF THEY ARE EQUAL
          JNC   SECOND   ;IF NEGATIVE, SECOND WAS LARGER
          MOV   A,R0     ;ELSE, OUTPUT FIRST
          OUTL  BUS,A
          JMP   DONE     ;EXIT
SECOND:   MOV   A,R1     ;OUTPUT SECOND
          OUTL  BUS,A
          JMP   DONE     ;EXIT
EQUL:     CPL   F0       ;SET F0
          JMP   DONE     ;EXIT
```

8 × 8 MULTIPLY-ASSEMBLED BY MCS®-48 MACRO ASSEMBLER*

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V3.0

  LOC  OBJ          LINE        SOURCE STATEMENT

                     111 $INCLUDE(:F1:MPY8)
                    1= 112 $INCLUDE(:F1:MPY8.PDL)
                    2= 113 ;*******************************************************************************
                    2= 114 ;*                                                                             *
                    2= 115 ;*       MPY8X8                                                                *
                    2= 116 ;*                                                                             *
                    2= 117 ;*******************************************************************************
                    2= 118 ;*                                                                             *
                    2= 119 ;*       THIS UTILITY PROVIDES AN 8 BY 8 UNSIGNED MULTIPLY                     *
                    2= 120 ;*       AT ENTRY:                                                             *
                    2= 121 ;*         A = LOWER EIGHT BITS OF DESTINATION OPERAND                          *
                    2= 122 ;*         XA= DON'T CARE                                                      *
                    2= 123 ;*         R1= POINTER TO SOURCE OPERAND (MULTIPLIER) IN INTERNAL MEMEORY       *
                    2= 124 ;*                                                                             *
                    2= 125 ;*       AT EXIT:                                                              *
                    2= 126 ;*         A = LOWER EIGHT BITS OF RESULT                                      *
                    2= 127 ;*         XA= UPPER EIGHT BITS OF RESULT                                      *
                    2= 128 ;*         C = SET IF OVERFLOW ELSE CLEARED                                    *
                    2= 129 ;*                                                                             *
                    2= 130 ;*******************************************************************************
                    2= 131 ;
                    2= 132 )
                    2= 133 ;1 MPY8X8:
                    2= 134 ;1 MULTIPLICAND[15-8]:=0
                    2= 135 ;1 COUNT:=8
                    2= 136 ;1 REPEAT
                    2= 137 ;2     IF MULTIPLICAND[0]=0 THEN BEGIN
                    2= 138 ;3         MULTIPLICAND:=MULTIPLICAND/2
                    2= 139 ;2     ELSE
                    2= 140 ;3         MULTIPLICAND[15-8]:=MULTIPLICAND[15-8]+MULTIPLIER
                    2= 141 ;3         MULTIPLICAND:=MULTIPLICAND/2
                    2= 142 ;2     ENDIF
                    2= 143 ;2     COUNT:=COUNT-1
                    2= 144 ;1 UNTIL COUNT=0
                    2= 145 ;1 END MPY8X8
                    1= 146 ;
                    1= 147 ;
                    1= 148 $EJECT
```

*See the application note on "Serial I/O and Math Utilities for the 8049AH Microcomputer" in Intel's *Microcontroller Applications Handbook*.

8 × 8 MULTIPLY-ASSEMBLED BY MCS®-48 MACRO ASSEMBLER*

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V3.0

  LOC  OBJ          LINE         SOURCE STATEMENT

                   1= 149 ;1 MPY8X8:
                   1= 150 MPY8X8:
                   1= 151 ;1 MULTIPLICAND[15-8]:=0
  0035 BA00        1= 152          MOV     XA,#00
                   1= 153 ;1 COUNT:=8
  0037 B808        1= 154          MOV     COUNT,#8
                   1= 155 ;1 REPEAT
                   1= 156 MPY8LP:
                   1= 157 ;2     IF MULTIPLICAND[0]=0 THEN BEGIN
  0039 1243        1= 158          JB0     MPY8A
                   1= 159 ;3       MULTIPLICAND:=MULTIPLICAND/2
  003B 2A          1= 160          XCH     A,XA
  003C 97          1= 161          CLR     C
  003D 67          1= 162          RRC     A
  003E 2A          1= 163          XCH     A,XA
  003F 67          1= 164          RRC     A
  0040 E839        1= 165          DJNZ    COUNT,MPY8LP
  0042 83          1= 166          RET
                   1= 167 ;2     ELSE
                   1= 168 MPY8A:
                   1= 169 ;3       MULTIPLICAND[15-8]:=MULTIPLICAND[15-8]+MULTIPLIER
  0043 2A          1= 170          XCH     A,XA
  0044 61          1= 171          ADD     A,@R1
  0045 67          1= 172          RRC     A
  0046 2A          1= 173          XCH     A,XA
  0047 67          1= 174          RRC     A
  0048 E839        1= 175          DJNZ    COUNT,MPY8LP
  004A 83          1= 176          RET
                   1= 177 ;3       MULTIPLICAND:=MULTIPLICAND/2
                   1= 178 ;2     ENDIF
                   1= 179 ;2     COUNT:=COUNT-1
                   1= 180 ;1 UNTIL COUNT=0
                   1= 181 ;1 END MPY8X8
                   1= 182 $EJECT
```

*See the application note on "Serial I/O and Math Utilities for the 8049AH Microcomputer" in Intel's *Microcontroller Applications Handbook*.

16× 8 DIVIDE—(ASSEMBLED BY MCS®-48 MACRO ASSEMBLER)*

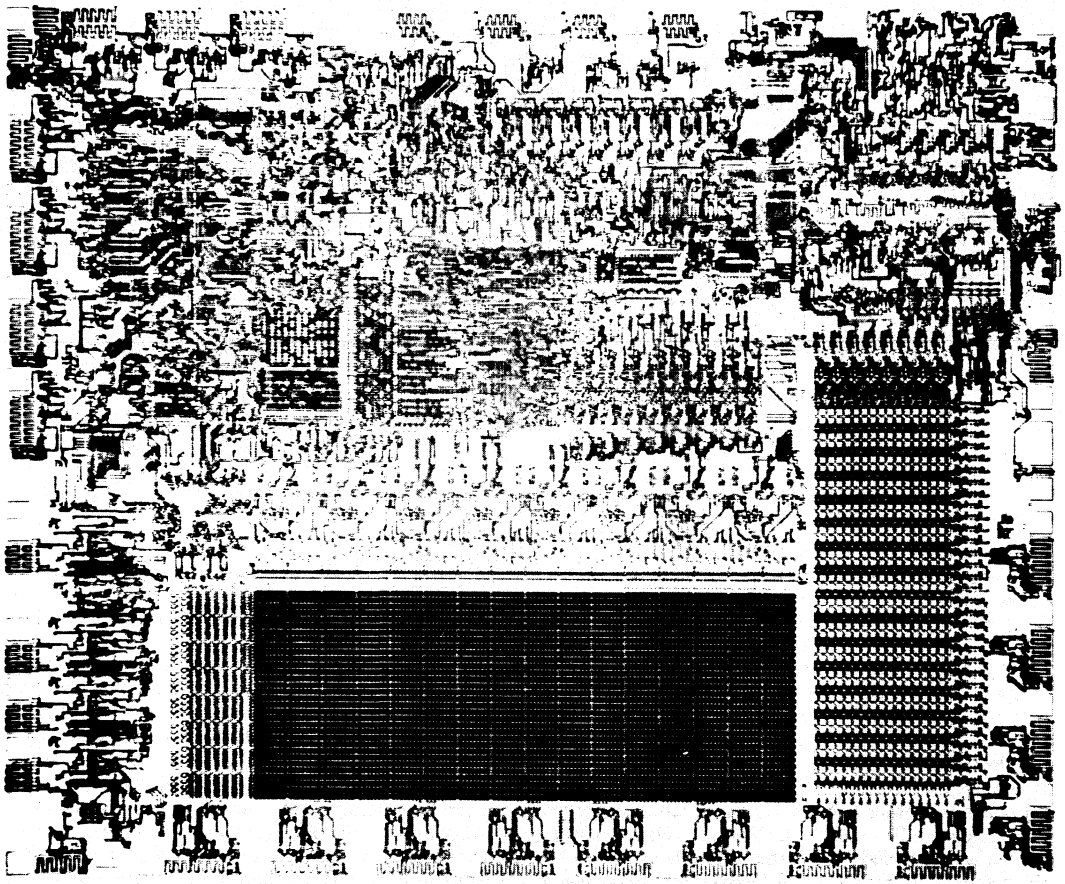
```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V3.0

  LOC  OBJ          LINE        SOURCE STATEMENT

                     = 183 $INCLUDE(:F1:DIV16)
                    1= 184 ;*****************************************************************************
                    1= 185 ;*                                                                           *
                    1= 186 ;*       DIV16                                                               *
                    1= 187 ;*                                                                           *
                    1= 188 ;*===========================================================================*
                    1= 189 ;*                                                                           *
                    1= 190 ;*       THIS UTILITY PROVIDES AN 16 BY 8 UNSIGNED DIVIDE                    *
                    1= 191 ;*       AT ENTRY:                                                           *
                    1= 192 ;*         A = LOWER EIGHT BITS OF DESTINATION OPERAND                        *
                    1= 193 ;*         XA= UPPER EIGHT BITS OF DIVIDEND                                  *
                    1= 194 ;*         R1= POINTER TO DIVISOR IN INTERNAL MEMORY                          *
                    1= 195 ;*                                                                           *
                    1= 196 ;*       AT EXIT:                                                            *
                    1= 197 ;*         A = LOWER EIGHT BITS OF RESULT                                    *
                    1= 198 ;*         XA= REMAINDER                                                     *
                    1= 199 ;*         C = SET IF OVERFLOW ELSE CLEARED                                  *
                    1= 200 ;*                                                                           *
                    1= 201 ;*****************************************************************************
                    1= 202 ;
                    1= 203 ;
                    1= 204 ;1 DIV16:
  004B 2A           1= 205 DIV16:   XCH     A,XA      ; ROUTINE WORKS MOSTLY WITH BITS 15-8
                    1= 206 ;1 COUNT:=8
  004C B808         1= 207          MOV     COUNT,#8
                    1= 208 ;1 DIVIDEND[15-8]:=DIVIDEND[15-8]-DIVISOR
  004E 37           1= 209          CPL     A
  004F 61           1= 210          ADD     A,@R1
  0050 37           1= 211          CPL     A
                    1= 212 ;1 IF BORROW=0 THEN /* IT FITS*/
  0051 F656         1= 213          JC      DIVIA
                    1= 214 ;2     SET OVERFLOW FLAG
  0053 A7           1= 215          CPL     C
  0054 046F         1= 216          JMP     DIVIB
                    1= 217 ;1 ELSE
                    1= 218 DIVIA:
                    1= 219 ;2     RESTORE DIVIDEND
  0056 61           1= 220          ADD     A,@R1
                    1= 221 ;2     REPEAT
                    1= 222 DIVILP:
                    1= 223 ;4           DIVIDEND:=DIVIDEND*2
                    1= 224 ;4           QUOTIENT:=QUOTIENT*2
  0057 97           1= 225          CLR     C
  0058 2A           1= 226          XCH     A,XA
  0059 F7           1= 227          RLC     A
  005A 2A           1= 228          XCH     A,XA
  005B F7           1= 229          RLC     A
  005C E663         1= 230          JNC     DIVIE
  005E 37           1= 231          CPL     A
  005F 61           1= 232          ADD     A,@R1
  0060 37           1= 233          CPL     A
  0061 0468         1= 234          JMP     DIVIC
                    1= 235 ;4           DIVIDEND[15-8]:=DIVIDEND[15-8]-DIVISOR
  0063 37           1= 236 DIVIE:   CPL     A
  0064 61           1= 237          ADD     A,@R1
```

*See the application note on "Serial I/O and Math Utilities for the 8049AH Microcomputer" in Intel's *Microcontroller Applications Handbook*.

16× 8 DIVIDE—(ASSEMBLED BY MCS®-48 MACRO ASSEMBLER)*

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V3.0

 LOC  OBJ          LINE        SOURCE STATEMENT

0065 37         1= 238          CPL     A
                1= 239 ;            IF BORROW=1 THEN
0066 E668       1= 240          JNC     DIVIC
                1= 241 ;                RESTORE DIVIDEND
0068 61         1= 242          ADD     A,@R1
0069 046C       1= 243          JMP     DIVID
                1= 244 ;            ELSE
                1= 245 DIVIC:
                1= 246 ;                QUOTIENT[0]:=1
006B 1A         1= 247          INC     XA
                1= 248 ;            ENDIF
                1= 249 ;            COUNT:=COUNT-1
                1= 250 ;          UNTIL COUNT=0
006C E857       1= 251 DIVID:   DJNZ    COUNT,DIVILP
                1= 252 ;          CLEAR OVERFLOW FLAG
006E 97         1= 253          CLR     C
                1= 254 ;          ENDIF
                1= 255 ;    ENDDIVIDE
006F 2A         1= 256 DIVIB:   XCH     A,XA
0070 83         1= 257          RET
```

*See the application note on "Serial I/O and Math Utilities for the 8049AH Microcomputer" in Intel's *Microcontroller Applications Handbook*.

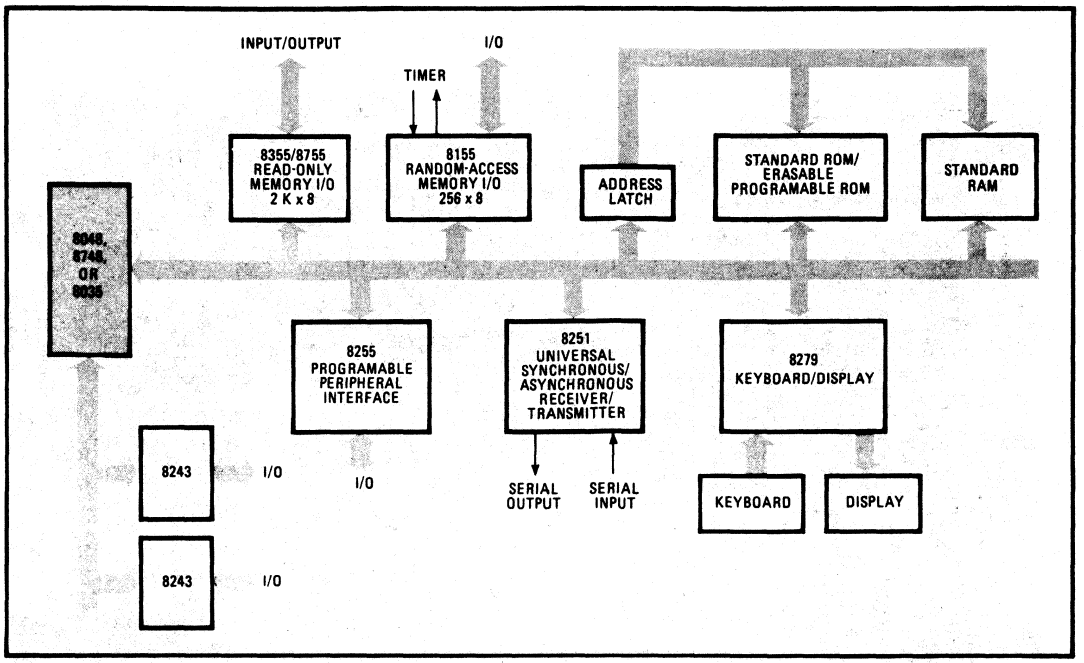


Single-chip 8-bit microcomputer fills gap between calculator types and powerful multichip processors

Capabilities range from stand-alone computing to high-power data processing; ultraviolet light erases programable ROM of one version

by Henry Blume, David Budde, Bill Morgan, Howard Raphael, Phil Salsbury, David Stamm,
Intel Corp., Santa Clara, Calif.

Reprinted from *Electronics*, November 25, 1976.



1. Expandable. Although well able to run a stand-alone controller by itself, the new processor can also work with other family members for larger control systems or with 8080 peripherals to handle complex data processing. This configuration typifies the MCS-48 capability.

☐ Putting an 8-bit microcomputer onto a single chip is achievement enough, but realizing performance nearly equal to multiple-chip devices gives a bonus of added flexibility for the new family. The two devices that are the heart of the family are really high-performance, single-chip microcomputers that fill the gap between 4-bit calculator chips and the 8-bit multichip microprocessors. They can be used for the lowest levels of control, or, by being expanded with other ROM/RAM members of their family or with standard 8080 peripheral memory chips, they can be used in a wide range of high-powered data-processing systems.

The two versions of the microcomputer, the 8748 and the 8048, are like 4-bit calculator devices in that they each contain all the elements needed for stand-alone computing—central processing unit, program read-only memory, data random-access memory, input/output interface, plus clocks and timers. Yet they contain these elements in 8-bit configurations that vastly exceed the power of the calculator types and approach 8080 power.

Two ROM versions

The MCS-48 family is the first to offer a microprocessor with an erasable programable ROM, which will prove handy for low-volume applications and those in which periodic update of the program memory is required. The family also has a CPU-only chip, the 8035, which can be used with external memories.

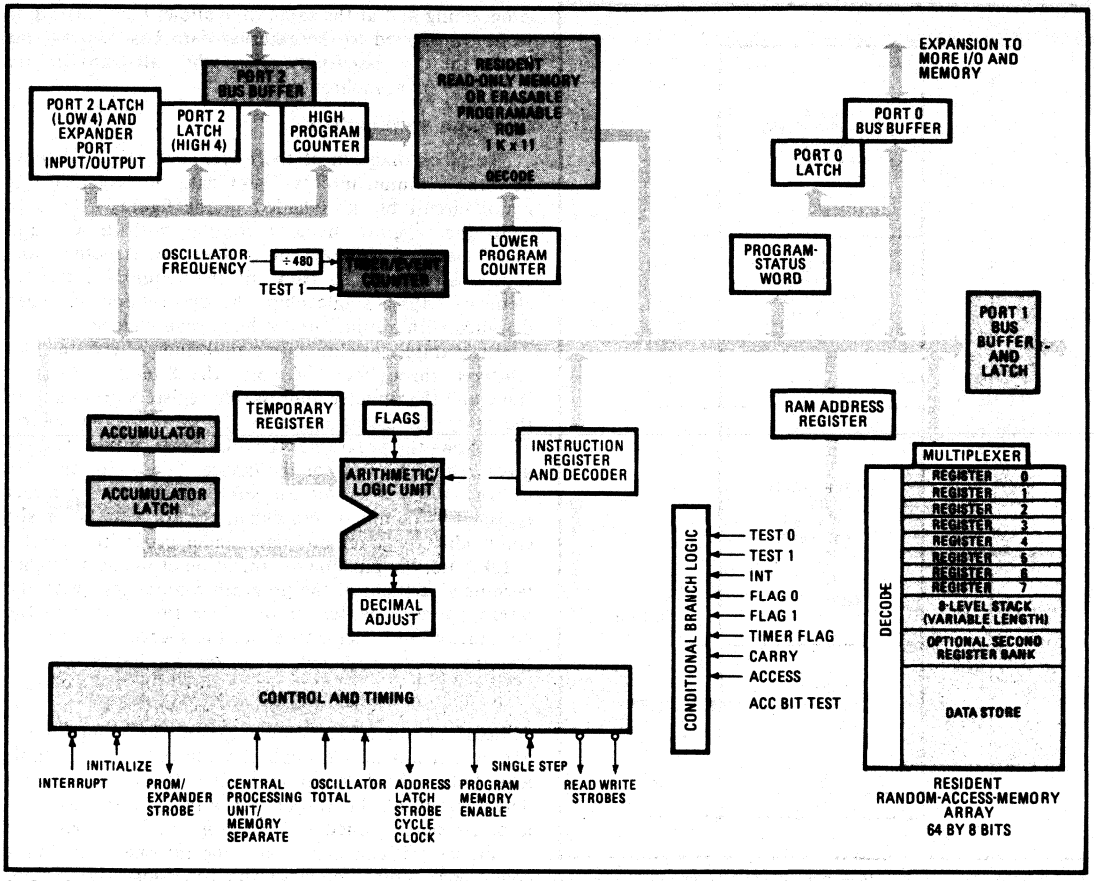
The 8748 has a 2708-type, 8,192-bit EPROM with a program that can be changed by clearing with ultraviolet light and reprograming electrically in the usual way. The 8048 has an 8-k mask-programable ROM. Together they give the user new flexibility: he can develop the program and build the prototypes with the reprogramable chip and switch to mask ROMs for volume production.

The off-the-shelf 8748 also is perfect for quick-turnaround users who require small volumes only, since it can be programed to meet any system specification in any quantity—in contrast to some single-chip controllers requiring mask programing at the factory, which is often available only in large quantities. Equally important, the 8748 can be used in control systems requiring periodic updating in the field, such as point-of-sale price-and-inventory controls. New program data can be fed into the system without a new ROM.

The free-standing operations of the 8048 and 8748 are made possible by the 1,024-by-8-bit ROM or EPROM for program memory, a 64-by-9-bit RAM for scratchpad functions, an 8-bit CPU consisting of an arithmetic/logic unit and accumulator for all the binary and decimal arithmetic functions, and an input/output facility that includes three 8-bit I/O ports plus three test/interrupt ports directly controlled by program instructions.

Memory and input/output of the processors can be expanded to handle large control applications (Fig. 1). There's an inexpensive expander chip, 8243, which allows the processor chips to handle an additional 16 I/O lines. Also included in the family are combination memory and I/O expanders, such as a 2,048-by-8-bit ROM with 16 I/O lines (8355), a 2-k-by-8-bit EPROM with 16 I/O lines (8755), and a 256-by-8-bit RAM with 22 I/O lines (8155).

The MCS-48 components also work directly with all



2. Stacked. The 8748 or 8048 processor chip supplies all the functions needed for a stand-alone microcomputer. It has a CPU complete with arithmetic/logic unit and accumulator, a 256-bit RAM, an 8,192-bit program ROM, a timer/event counter, and plenty of I/O capability.

the 8080 family of standard memory and peripheral parts, soon to number about 30 large-scale-integrated circuits. They include timers, programable I/O controllers, universal synchronous/asynchronous receiver/transmitters, decoders, and keyboard/display controllers.

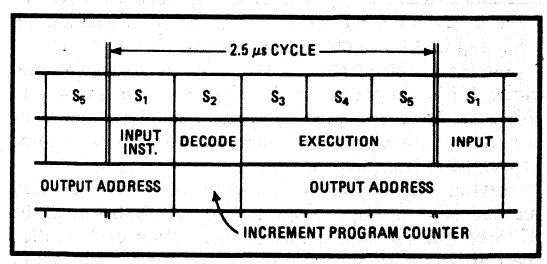
One-chip advantages

The integration of all the basic blocks of a microcomputer system into one circuit brings about some architectural advantages. When the device is used as a stand-alone controller, it need interface only with its I/O peripherals. This means that the execution speed of the processing is limited only by the speed of the chip, because there is no slowdown from transferring data between memory and CPU, as in multiple-chip designs.

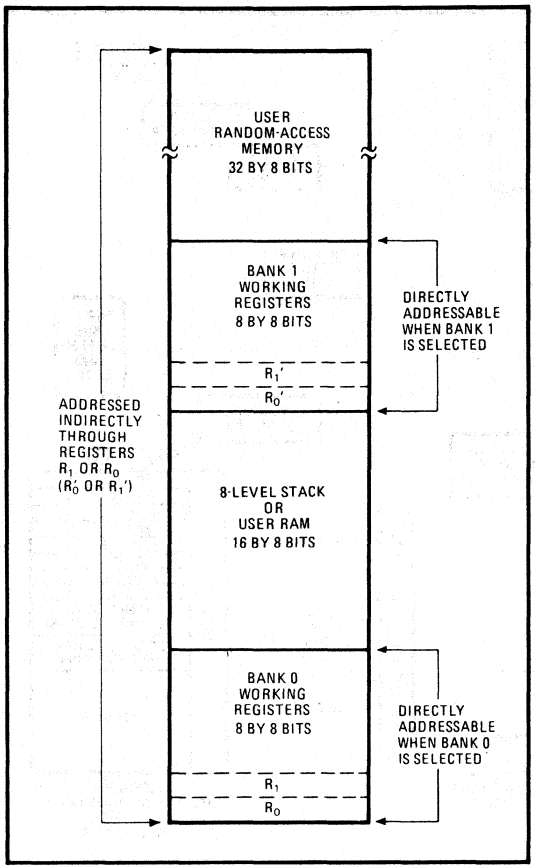
Moreover, technological upgrades can give enhanced performance without waiting for similar upgrades of external components, as is usually the case with multichip families. More immediately, the inclusion of data and program memories, which otherwise would have to be added separately to the system, simplifies the user's interface problems.

Having an active data store on the chip—the quasi-static 64-by-8-bit RAM—also simplifies system implementation, since all scratchpad operations simply became part of the CPU function. There is no need for refresh circuits operate the RAM; yet the device is dynamic in the sense that internal clocks are used for very fast, low-power access to the array.

The major objective was access to a RAM within a



3. Simple. Operating the 8748/8048 is extremely straightforward, with each 2.5-μs cycle consisting of five states. Instruction inputs are made in state 1, decoding and program incrementing in state 2. Program executions begin in state 3 and run through 4 and 5.



4. Powerful. The on-chip RAM, part of which is reserved for one or two banks of 8-bit working registers, also accommodates the stack of subroutine addresses, which can be eight levels deep. Each stack location can handle the program counter and status data.

fraction of an instruction cycle, so that those indirect internal instructions that require multiple addresses could still be executed in one instruction cycle. (Indirect RAM instructions require three separate accesses: one to fetch the address of the memory location to be operated on, one to fetch the contents of the addressed location, and one to store the results of the operation.) Since the RAM is dynamic, its power dissipation, including all decoding and sense circuits, is a mere 75 milliwatts.

Similarly, the EPROM of the 8748 relies on internal clocks for better access and lower power consumption. In this case, however, only one access per instruction cycle is required, since there are no indirect instructions to be processed in program memory.

Having the EPROM on the chip allows for an easy method of verifying a program. To accomplish this, the 8748 can be put into a special instruction cycle (called the third-state mode) for programing and verification of the EPROM. The CPU executes a special double-cycle instruction that allows the address and data information to be transferred to their respective registers during programing and at the same time allows the EPROM data to be transferred to the external data bus. During this mode, the CPU essentially idles, while all transfers are controlled by asynchronous inputs.

Common architecture

The block diagram of the 8748/8048 (Fig. 2) shows how the common internal 8-bit data bus connects the major circuit blocks (shaded in the figure)—the data store, the program memory, the CPU with its ALU and accumulator, a timer/event counter, I/O structure, and control structure. To pack all the required computer elements onto a single chip, the CPU section has been designed with a minimum of logic redundancies.

For example, to eliminate a multitude of register files scattered throughout the chip, the 8-level subroutine stack and the directly addressable registers are found in the same addressing space as the scratchpad memory. This allows the programer maximum use of the RAM, yet gives minimum logic for the device. The programer can utilize unused areas of the subroutine stack or direct registers as common scratchpad memory, or he or she can modify the stack and flags under program control.

Likewise, the pipeline organization of memory fetches permits placement of the program counter (pc) with the internal timer/counter circuit block rather than in the RAM array. Both elements share the source incrementer, resulting in more efficient use of on-chip hardware.

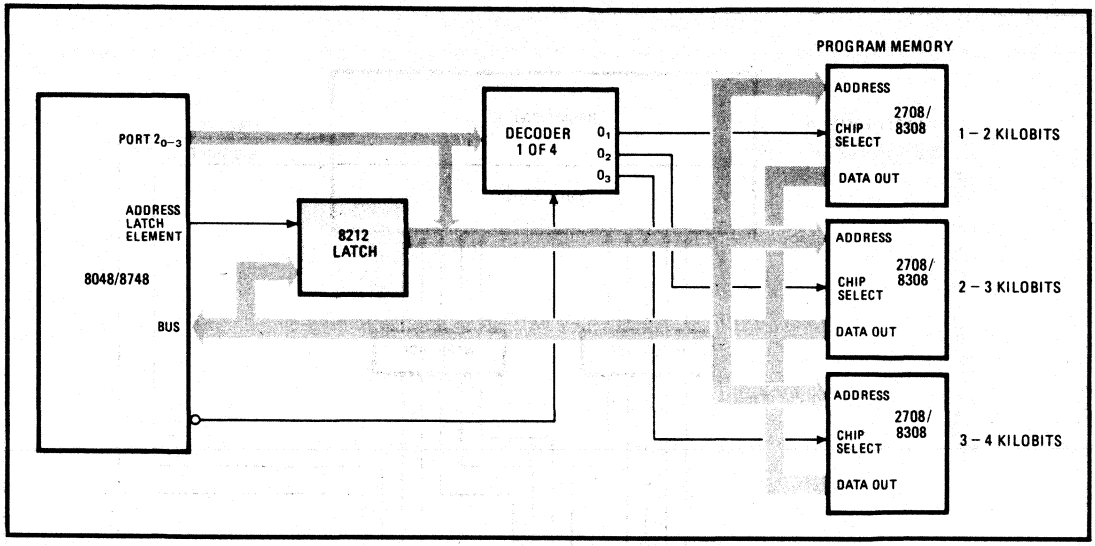
In addition to executing the required functions of ADD, XOR, AND, and OR, the ALU also performs the bit-comparison operations necessary for conditional jump and test facilities. Through the use of a control-table ROM (which holds constant 8-bit values), and a zero-detect circuit on the ALU output, any bit in the accumulator can be examined and the program flow modified.

This setup is also used to test for any one of the many conditional jumps. Each of the conditional-jump flags and inputs is sent to the ALU as an 8-bit conditional word and tested with the same circuitry used to examine individual accumulator bits.

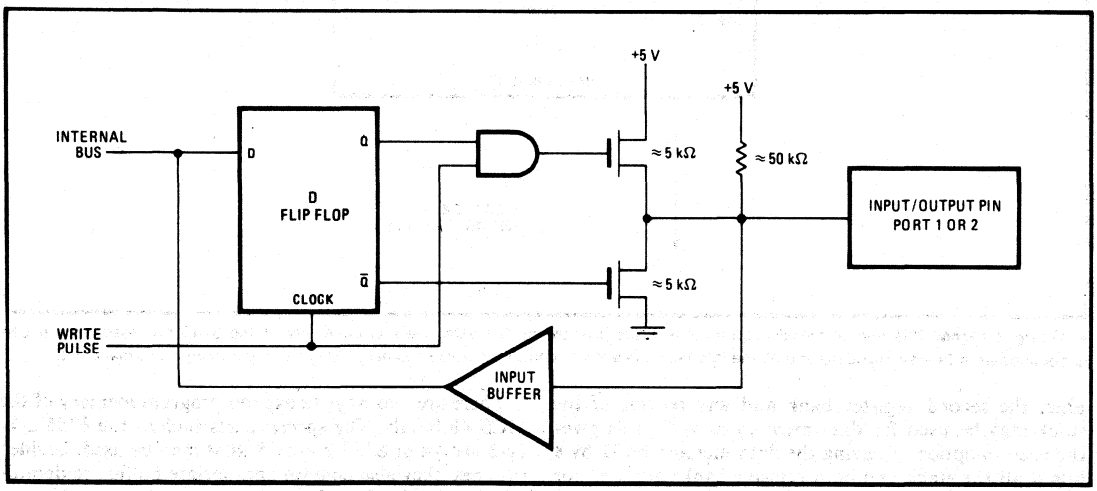
An internal oscillator also gives many system and device savings, such as the elimination of external components (except for a crystal or an RC network for setting the system's operating frequency). It also gives the chip designer maximum freedom in the structure of the internal clocking scheme, because there is no need for high-level, accurate clock inputs.

Through efficient use of internal bus transfers, most instructions can be executed in a single-cycle length. The exceptions are those instructions which require a second memory fetch or an external I/O transfer. In these cases, only a second cycle is required. Moreover, limiting instructions to two lengths reduces the complexity of the internal state generator. Since 70% of all instructions are executed in a single cycle, program-execution times and program-storage size are still minimized.

The multiplexed bus for address and data during external memory references maximizes the number of I/O pins available on a cost-effective 40-pin dual in-line package. For external program-memory references, bits of an additional I/O port are used for address lines, with the input/output data being restored after the memory



5. Latching on. Adding standard memories to the system is quickly done with external latch 8212, which allows standard memory parts to be hooked directly onto the 8748/8048 bus. Operation of the latch is under the control of signals from the processor.



6. Mixing it up. Besides the main system port, 0, the processor chip has two others, 1 and 2, which allow inputs and outputs to be mixed on the same port. Here, writing a 0 causes the pull-down devices to sink the TTL load; writing a 1 calls on the 50-kilohm pull-up resistor.

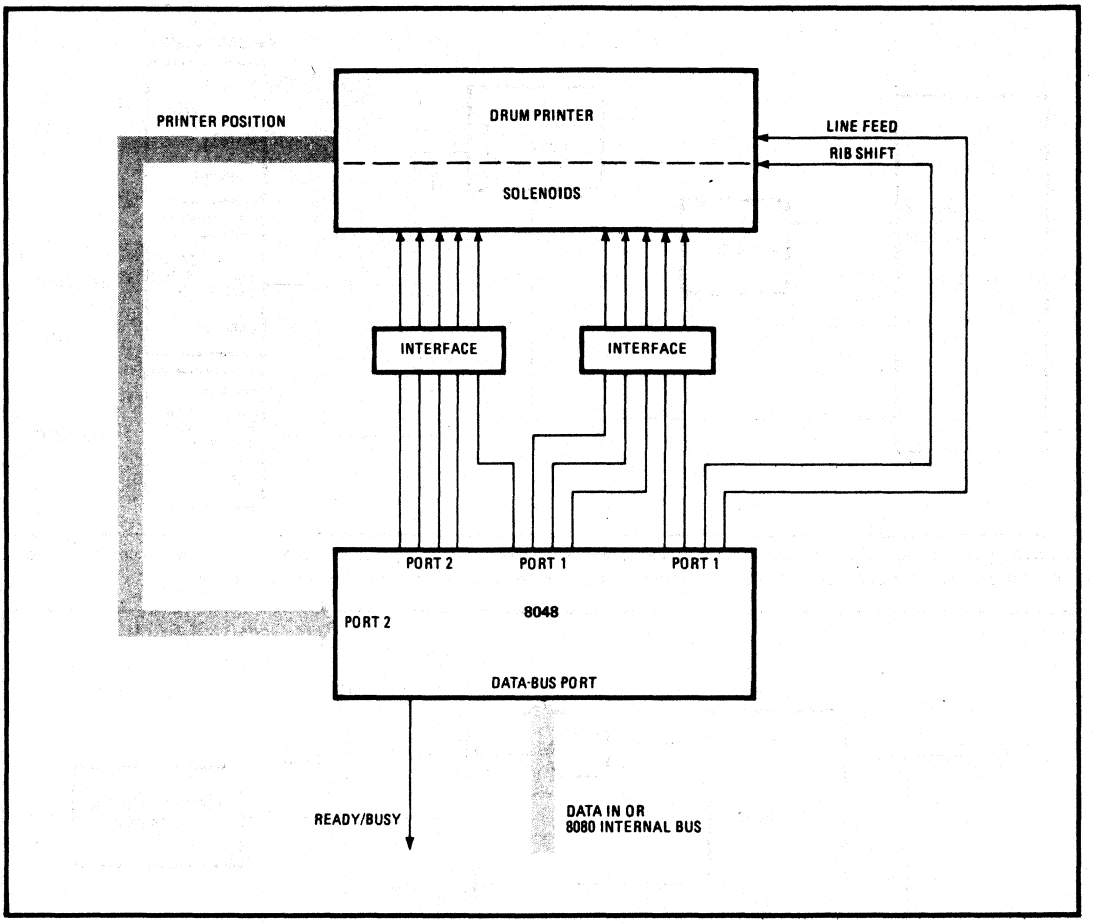
reference is finished with the address.

One key to the simple operation of the 8748/8048 chip is the straightforward program sequences and timing needed for executing an instruction cycle (Fig. 3). Each cycle consists of five states. Instruction input is made in state 1, and decoding and pc incrementing is made in state 2. State 3 starts the beginning of the program execution, which can run through states 4 and 5. Simultaneously, the next cycle's program address is made in state 3, a pipelining (paralleling) of operations that increases device throughput significantly.

Because the chip is built with depletion-load silicon-gate n-channel technology, it operates off a single 5-volt supply with inputs and outputs that are compatible with both transistor-transistor-logic and complementary metal-oxide-semiconductor devices. Instruction cycle time is a modest 2.5 microseconds and power consumption is a low 400 mW. Depletion-load techniques also pay off in practical chip sizes for volume production; the 8048 also is slightly over 200 mils on a side, while the 8748, with its big 8-k EPROM, is 221 by 261 mils.

Storing data in the scratchpad is simple, because part of the RAM can be reserved for one or two banks of 8-bit working registers—eight registers per bank (Fig. 4). The scratchpad also contains the subroutine address stack, which can be eight levels deep. Each location can accommodate the 12-bit pc and 4-bit status data.

Since all locations in the stack are indirectly address-



7. Going it alone. This one-chip scale controller is made possible by the extensive I/O capability of the 8748 processor, which can accommodate a 24-key keyboard and all the interfacing needed to control 14 seven-segment LED arrays, including a decimal point.

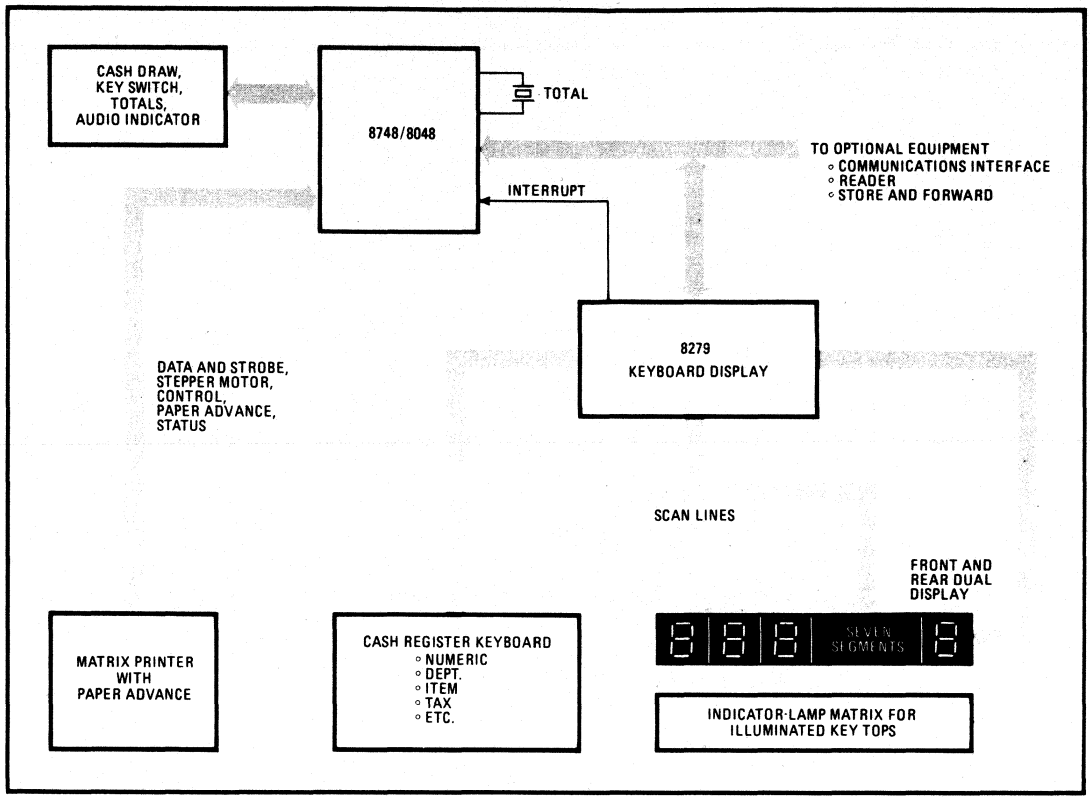
able, the second register bank and any portion of the stack may be used for data memory as well. This gives the user an option of having the data memory be 32 by 8 bits if all the stack and both register banks are used for program counting and status data, or 56 by 8 bits if only one register bank is used.

Program memory

The resident program memories on the 8048/8748 chips are handled so that they can be operated alone for programs of 1,024 bytes or less or combined with external ROM for expanded systems requiring larger programs. The program counter that feeds the memory is split into two parts. The low-order 8 bits can either address the resident 1-k ROM or be routed externally when addressing beyond 1,024 bits. (Since the 8035 contains no internal ROM, all address fetches are external.) The upper 4 bits of the program counter, located near port 2 (see Fig. 2), are gated out on that port for external reference. Two of these most significant 4 bits are then used for internal addressing requirements.

There are two ways to expand program memory of the MCS-48 family. The special parts such as the 8755 2-k-by-8 EPROM or 8355 2-k-by-8 ROM may be used. Besides I/O lines, they also contain appropriate buffers to demultiplex the 8-bit bus from the microcomputer chips to receive address and send back program-memory instructions. Alternately, standard memory parts, such as a 2708 EPROM or 8308 ROM may be used (Fig. 5). An external latch, such as the 8212, would latch up the address from the bus (via a signal from the 8048 or 8748) so that data could be returned on the bus. The high-order 4 bits of the address do not have to be latched, since they are not on the multiplexed bus.

The ALU, in conjunction with the accumulator, provides a full array of binary-and-decimal arithmetic, logic, shift, and increment/decrement functions. For example, the accumulator may be exchanged between registers, data memory, and program memory. Both the timer/counter and the program-status word are also accessible to the accumulator, through a latch that facilitates the accumulator source/destination instruc-



8. Working together. Proof of the MSC-48's ability to handle large systems is this gas-pump controller. The 8243 I/O expander chips allow the processor to interface with 47 lines and a USART communicating with a central control unit inside the service station.

tions. Here, the ALU generates a carry output fully accessible to the programer under program control.

The timer/event counter is an 8-bit register that can operate in one of two modes, selectable under software control. As a timer, the device measures elapsed time. It is fed by the crystal frequency, divided by 280. At maximum frequency, the result is about 80 μs per increment, or about 20 milliseconds over the counter range. As an event counter, a test line is designated to count 0 to 1 transitions of external events. As many as 256 transitions may be accommodated.

Both the timer and the counter indicate overflow by a maskable internal interrupt or by a testable flag bit. The internal interrupt may also be used to provide the system with a second external interrupt.

The input/output facilities of the 8048/8748 have been designed for maximum flexibility and expansion and are fully TTL-compatible. The basic facilities consist of three 8-bit I/O ports plus three test/interrupt inputs.

Port 0, called the bus, provides for system expansion. In essence, the port makes the bus completely compatible with an 8080 bus, so that all 8080 peripherals can be used with the MCS-48 family. In conjunction with four control and strobe lines, the port may be used for bidirectional interfacing to memories and I/O elements. For free-standing operations, it may be statically latched or used as a general input port.

The remaining two I/O ports, 1 and 2, are termed "quasi-bidirectional" (Fig. 6). They allow inputs and outputs to be mixed on the same port. When writing a 0 (low value) to these ports, the pull-down device sinks the TTL load. When writing a 1, a large current is supplied through both pull-up devices to allow a fast transition. After a short time, they shut off and the pull-up of the 50-kilohm resistor sustains the 1 level.

Applying the 8748/8048

Two applications show the range of complexity that can be accommodated with this family. Figure 7 shows a typical minimum-chip MCS-48 system, in this case, a drum printer controller. The three output ports allow the one-chip 8048 to control the printer position, ribbon shift, and line feed. Two interface drivers operate the solenoids.

Figure 8 shows a far more complex system, in which the MCS-48 implements a low-cost point-of-sale terminal. The I/O capability of the 8748/8048 chip can be expanded to control and monitor many cash-register operations. These might include cash in the drawer, key switch, totals, audio indicator, as well as matrix printer, cash-register keyboard, seven-segment display, and a variety of optional equipment. □

Application Techniques for the MCS-48® Family

Contents

INTRODUCTION 5-40

THE MCS-48 FAMILY 5-40

ANALOG I/O 5-42

TABLE LOOKUP TECHNIQUES 5-46

RECEIVING SERIAL CODES — BASIC APPROACHES 5-47

RECEIVING SERIAL CODE — A MORE SOPHISTICATED ALGORITHM 5-51

TRANSMITTING SERIAL CODE 5-61

GENERATING PARITY 5-62

CONCLUSION 5-62

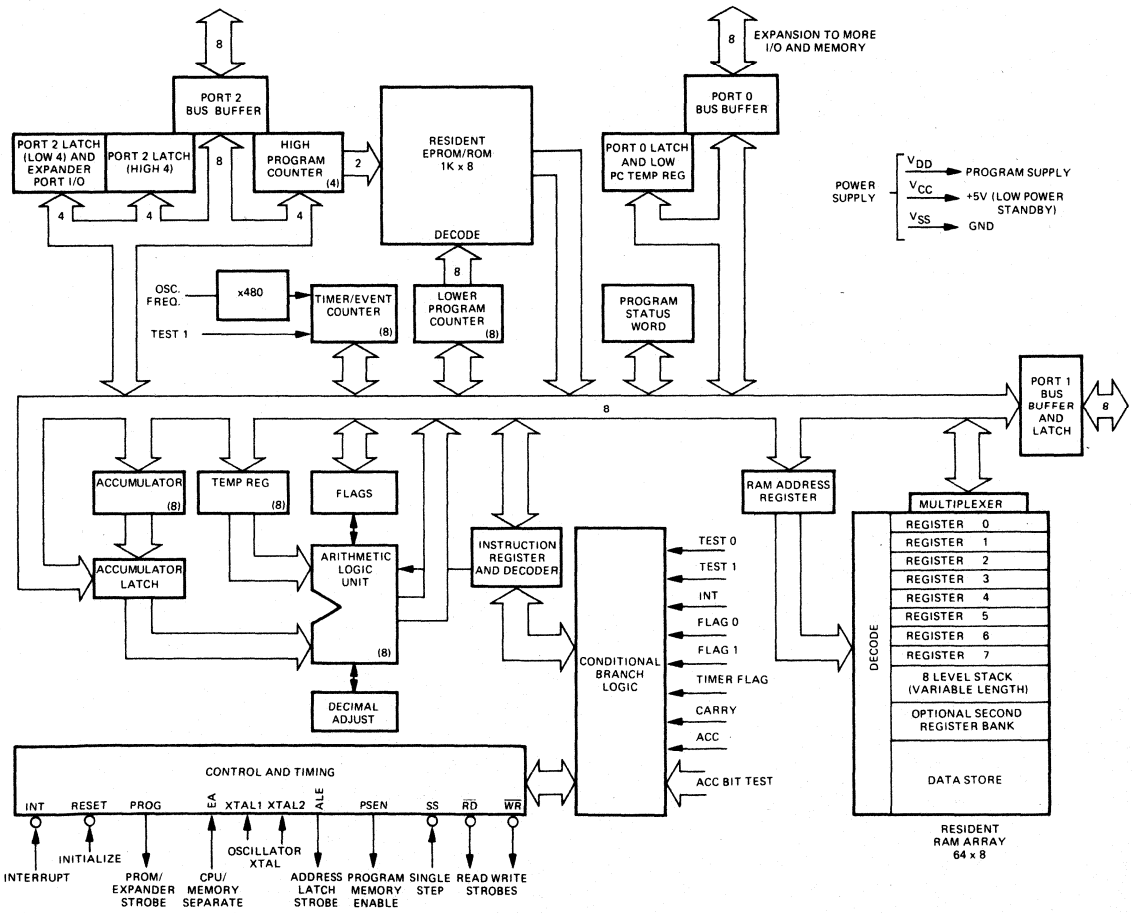
INTRODUCTION

The INTEL® MCS-48™ family consists of a series of seven parts, including three processors, which take advantage of the latest advances in silicon technology to provide the system designer with an effective solution to a wide variety of design problems. The significant contribution of the MCS-48 family is that instead of consisting of integrated microcomputer components it consists of integrated microcomputer systems. A single integrated circuit contains the processor, RAM, ROM (or PROM), a timer, and I/O.

This application note suggests a variety of application techniques which are useful with the MCS-48. Rather than presenting the design of a complete system it describes the implementation of "subsystems" which are common to many microprocessor based systems. The subsystems described are analog input and output, the use of tables for function evaluation, receiving serial code, transmitting serial code, and parity generation. After an overview of the MCS-48 family these areas are discussed in a more or less independent manner.

THE MCS-48™ FAMILY

The processors in the MCS-48 family all share an identical architecture. The only significant difference is the type of on board program storage which is provided. The 8748 (see Figure 1) includes 1024 bytes of erasable, programmable, ROM (EPROM), the 8048 replaces the EPROM with an equivalent amount of mask programmed ROM, and the 8035 provides the CPU function with no on board program storage. All three of these processors



MCS-48™ Internal Structure

INSTRUCTION SET

	Mnemonic	Description	Bytes	Cycle
Accumulator	ADD A,R	Add register to A	1	1
	ADD A, @R	Add data memory to A	1	1
	ADD A, #data	Add immediate to A	2	2
	ADDC A, R	Add register with carry	1	1
	ADDC A, @R	Add data memory with carry	1	1
	ADDC A, #data	Add immediate with carry	2	2
	ANL A, R	And register to A	1	1
	ANL A, @R	And data memory to A	1	1
	ANL A, #data	And immediate to A	2	2
	ORL A, R	Or register to A	1	1
	ORL A, @R	Or data memory to A	1	1
	ORL A, #data	Or immediate to A	2	2
	XRL A, R	Exclusive Or register to A	1	1
	XRL A, @R	Exclusive or data memory to A	1	1
	XRL A, #data	Exclusive or immediate to A	2	2
	INC A	Increment A	1	1
	DEC A	Decrement A	1	1
	CLR A	Clear A	1	1
	CPL A	Complement A	1	1
	DA A	Decimal Adjust A	1	1
	SWAP A	Swap nibbles of A	1	1
	RL A	Rotate A left	1	1
	RLC A	Rotate A left through carry	1	1
	RR A	Rotate A right	1	1
	RRC A	Rotate A right through carry	1	1
Input/Output	IN A, P	Input port to A	1	2
	OUTL P, A	Output A to port	1	2
	ANL P, #data	And immediate to port	2	2
	ORL P, #data	Or immediate to port	2	2
	INS A, BUS	Input BUS to A	1	2
	OUTL BUS, A	Output A to BUS	1	2
	ANL BUS, #data	And immediate to BUS	2	2
	ORL BUS, #data	Or immediate to BUS	2	2
	MOVD A, P	Input Expander port to A	1	2
	MOVD P, A	Output A to Expander port	1	2
	ANLD P, A	And A to Expander port	1	2
	ORLD P, A	Or A to Expander port	1	2
Registers	INC R	Increment register	1	1
	INC @R	Increment data memory	1	1
	DEC R	Decrement register	1	1
Branch	JMP addr	Jump unconditional	2	2
	JMPP @A	Jump indirect	1	2
	DJNZ R, addr	Decrement register and skip	2	2
	JC addr	Jump on Carry = 1	2	2
	JNC addr	Jump on Carry = 0	2	2
	J Z addr	Jump on A Zero	2	2
	JNZ addr	Jump on A not Zero	2	2
	JT0 addr	Jump on T0 = 1	2	2
	JNT0 addr	Jump on T0 = 0	2	2
	JT1 addr	Jump on T1 = 1	2	2
	JNT1 addr	Jump on T1 = 0	2	2
	JF0 addr	Jump on F0 = 1	2	2
	JF1 addr	Jump on F1 = 1	2	2
	JTF addr	Jump on timer flag	2	2
	JNI addr	Jump on $\overline{INT}$ = 0	2	2
	JBb addr	Jump on Accumulator Bit	2	2

	Mnemonic	Description	Bytes	Cycles
Subroutine	CALL	Jump to subroutine	2	2
	RET	Return	1	2
	RETR	Return and restore status	1	2
Flags	CLR C	Clear Carry	1	1
	CPL C	Complement Carry	1	1
	CLR F0	Clear Flag 0	1	1
	CPL F0	Complement Flag 0	1	1
	CLR F1	Clear Flag 1	1	1
	CPL F1	Complement Flag 1	1	1
Data Movers	MOV A, R	Move register to A	1	1
	MOV A, @R	Move data memory to A	1	1
	MOV A, #data	Move immediate to A	2	2
	MOV R, A	Move A to register	1	1
	MOV @R, A	Move A to data memory	1	1
	MOV R, #data	Move immediate to register	2	2
	MOV @R, #data	Move immediate to data memory	2	2
	MOV A, PSW	Move PSW to A	1	1
	MOV PSW, A	Move A to PSW	1	1
	XCH A, R	Exchange A and register	1	1
	XCH A, @R	Exchange A and data memory	1	1
	XCHD A, @R	Exchange nibble of A and register	1	1
	MOVX A, @R	Move external data memory to A	1	2
	MOVX @R, A	Move A to external data memory	1	2
	MOVP A, @A	Move to A from current page	1	2
	MOVP3 A, @A	Move to A from Page 3	1	2
Timer/Counter	MOV A, T	Read Timer/Counter	1	1
	MOV T, A	Load Timer/Counter	1	1
	STRT T	Start Timer	1	1
	STRT CNT	Start Counter	1	1
	STOP TCNT	Stop Timer/Counter	1	1
	EN TCNTI	Enable Timer/Counter Interrupt	1	1
	DIS TCNTI	Disable Timer/Counter Interrupt	1	1
Control	EN I	Enable external interrupt	1	1
	DIS I	Disable external interrupt	1	1
	SEL RB0	Select register bank 0	1	1
	SEL RB1	Select register bank 1	1	1
	SEL MB0	Select memory bank 0	1	1
	SEL MB1	Select memory bank 1	1	1
	ENTO CLK	Enable Clock output on T0	1	1
	NOP	No Operation	1	1

Figure 2. 8048/8748/8035 Instruction Set

operate from a single 5-volt power supply. The 8748 requires an additional 25-volt supply only while the on board EPROM is being programmed. When installed in a system only the 5-volt supply is needed. Aside from program storage, these chips include 64 bytes of data storage (RAM), an eight bit timer which can also be used to count external events, 27 programmable I/O pins and the processor itself. The processor offers a wide range of instruction capability including many designed for bit, nibble, and byte manipulation. The instruction set is summarized in Figure 2.

Aside from the processors, the MCS-48 family includes 4 devices: one pure I/O device and 3 combination memory and I/O devices. The pure I/O device is the 8243, a device which is connected to a special 4 bit bus provided by the MCS-48 processors and which provides 16 I/O pins which can be programmatically controlled.

The combination memory and I/O devices consist of the 8355, the 8755, and the 8155. The 8355 and the 8755 both provide 2,048 bytes of program storage and two eight bit data ports. The only difference between these devices is that the 8355 contains masked program ROM and the 8755 contains EPROM. The 8155 combines 256 bytes of data storage (RAM), two eight bit data ports, a six bit control port, and a 14 bit programmable timer.

Figure 3 shows the various system configurations which can be achieved using the MCS-48 family of parts. It should also be noted that eight of the processors' I/O lines have been configured as a bidirectional bus which can be used to interface to standard Intel peripheral parts such as the 8251 USART (for serial I/O), the 8255A PPI (provides 24 I/O lines) and the complete range of memory components.

More detailed information concerning the MCS-48 family can be obtained from the "MCS-48 Microcomputer User's Manual" which provides a complete description of the MCS-48 family and its members. A general familiarity with this document will make the application techniques which follow easier to understand.

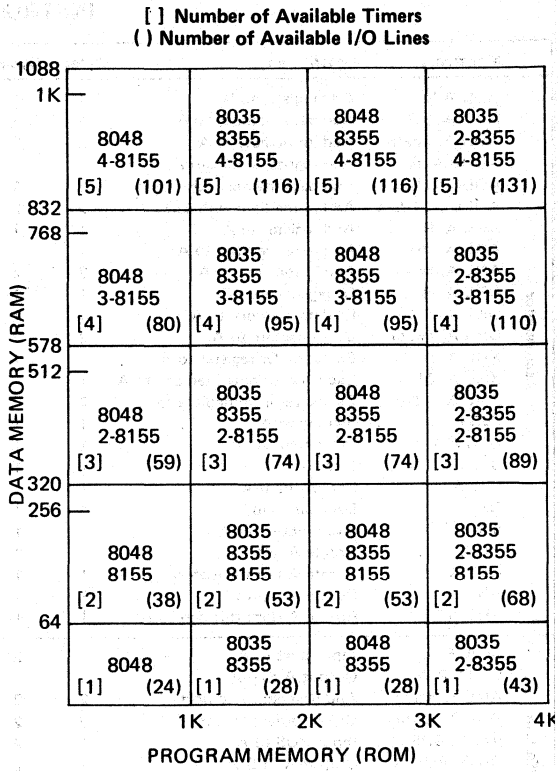


Figure 3. The Expanded MCS-48™ System

ANALOG I/O

If analog I/O is required for a MCS-48™ system there are many alternatives available from the makers of analog I/O modules. By searching through their catalogs it is possible to find almost any combination of features which is technically feasible. Perhaps the best example of such modules are the MP-10 and MP-20 hybrid modules recently introduced by Burr-Brown Research Corporation. The MP-10 provides two analog outputs and the MP-20 provides 16 analog inputs. Both of these units were specifically designed to interface with microprocessors.

A block diagram of the MP-10 is shown in Figure 4. It consists of two eight bit digital to analog converters, two eight bit latches which are loaded from the data bus, and address decoding logic to determine when the latches should be loaded. The D/A converters each generate an analog output in the range of 10 volts with an output impedance of 1Ω. Accuracy is ±0.4% of full scale and the output is stable 25μsec after the eight bit binary data is loaded into the appropriate latch. The latches are loaded by the write pulse ($\overline{WR}$) whenever the proper address is presented to the MP-10. The lower two addresses (A_0 and A_1) are used internally by the device. Addresses A_2 & A_3 are compared with the address determination inputs B_2 and B_3. If their signals are found to be equal, and if addresses A_4-A_{13} are all high, then the device is selected and one of the latches will be loaded. Address bit A_1 selects between output 1 and output 2. If address bit A_0 is set then the initialization channel of the DIA is selected. In order to prepare for operation a data pattern of 80_H must

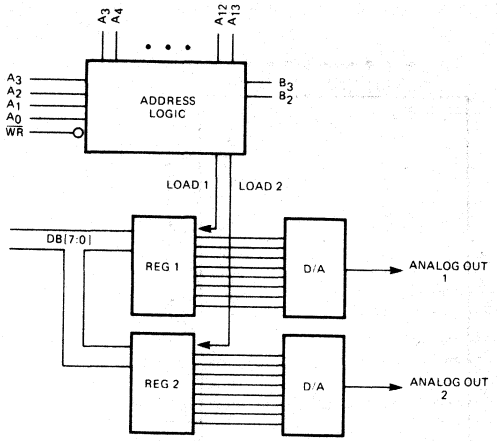


Figure 4. MP-10 Block Diagram

be output to this channel following the reset of the device.

A block diagram of the MP-20 analog to digital converter is shown in figure 5. This unit consists of a 16 input analog multiplexer, an instrumentation amplifier, an eight bit successive approximation analog to digital converter, and control logic. The 16 input multiplexer can be used to input either 16 single ended or 8 differential inputs. The output from the multiplexer is fed into the instrumentation amplifier which is configured so that it can easily be strapped for single ended 0-5 volt inputs, single ended ±5 volt inputs, or differential 0-5 volt signals. Provisions are made for an external gain control resistor on the amplifier. The gain control equation is:

$$G = 2 + \frac{50k\Omega}{R_{ext}}$$

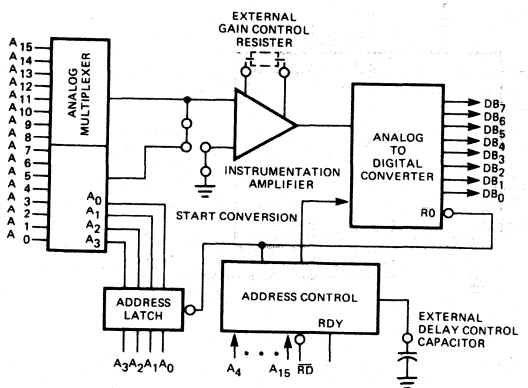


Figure 5. MP-20 Analog Subsystem

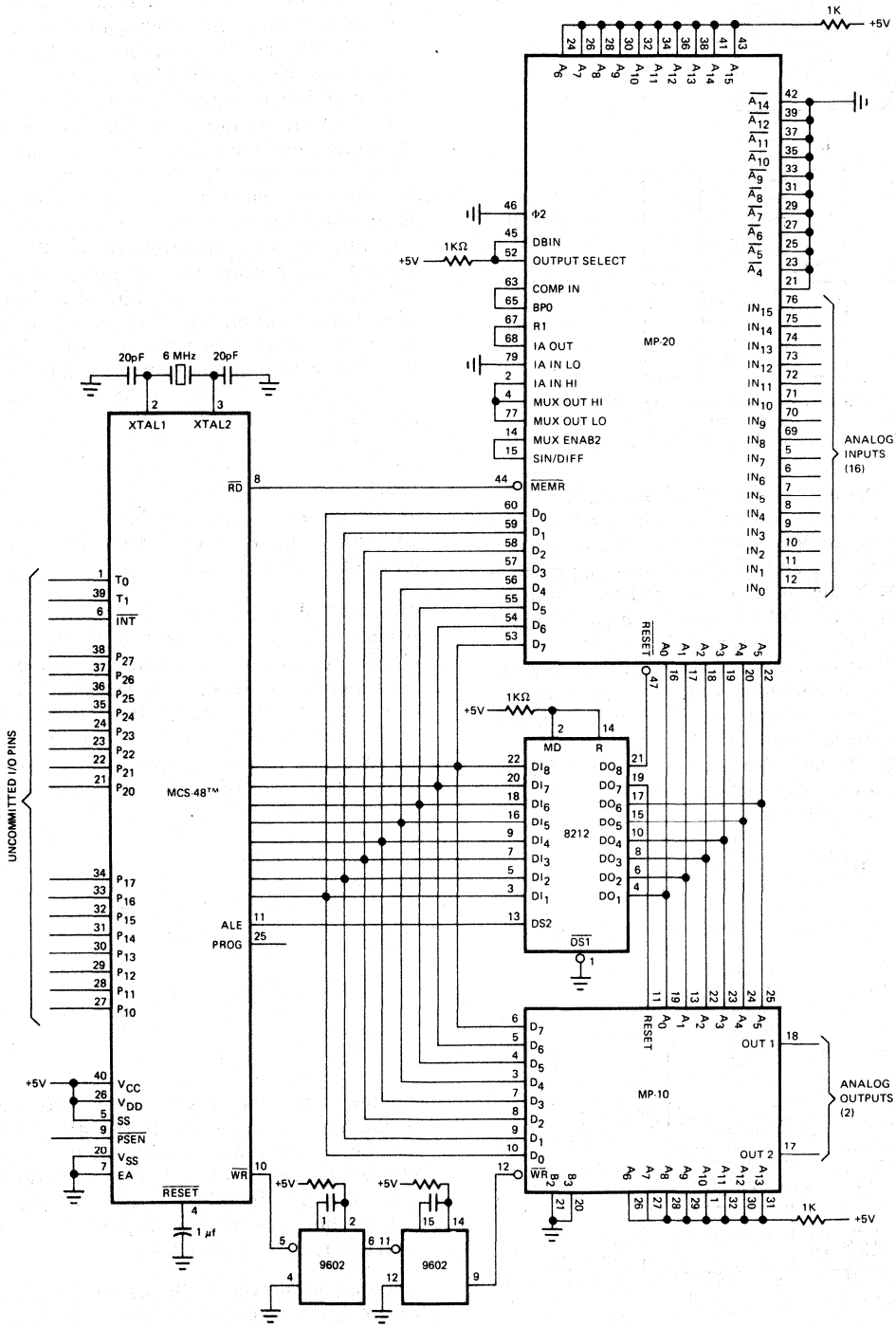
With no R_{ext} ($R_{ext} = \infty$) the gain is two and the input is 0-5 or ±5 volts full scale. Adding an external resistor results in higher gain so that low level (±50mV) signals from thermocouples and strain gauges can be accommodated. The output from the amplifier is applied to the actual A/D converter which provides an eight bit output with guaranteed monotonicity and an accuracy of ±0.4% of full scale. Note that this accuracy is specified for the entire module, not just for the converter itself. The control logic monitors address lines A_{15} through A_4 to determine when the address of the unit has been selected. An address that the unit will respond to is determined by 11 address control pins, labeled $\overline{A4}$ through $\overline{A14}$. If one of these pins is tied to a logic 0 then the corresponding address pin must be high in order for the unit to be selected. If the pin is tied to a logic 1 then the corresponding address pin must be low. If the address of the module is selected when $\overline{MEMR}$ pulse occurs, the lower four addresses (A_3-A_0) are stored in a latch which addresses the multiplexer. The coincidence of the proper address and $\overline{MEMR}$ also initiates a conversion and gates the output of the converter on to the eight bit data bus.

The control logic of the MP-20 was designed to operate directly with an MCS-80™ system. When a $\overline{MEMR}$ occurs and a conversion is initiated the MP-20 generates a READY signal which is used to extend the cycle of the 8080A for the duration of the conversion. READY is brought high after the conversion is complete which allows the 8080A to initiate a conversion and read the resulting data in a single, albeit long, memory or I/O cycle. The conversion time of the MP-20 depends on the gain selected for the amplifier. With no external resistor ($R = \infty$) the gain is two and the conversion time is 35 μsec. For $R = 510\Omega$ the gain is:

$$G = 2 + \frac{50k\Omega}{.51k\Omega} \cong 100$$

and the conversion time becomes 100μsec. These settling times are specified in the MP-20 data sheet and range from 35 to 175 microseconds. The READY timing is controlled by an external capacitor. For a gain of 2 no external capacitor is required but if higher gains are selected a capacitor is needed to extend the timing.

A schematic showing both the MP-10 D/A and the MP-20 A/D connected to the 8748 is shown in Figure 6. This configuration, which consists of only four major components, gives an excellent example of what modern technology can do for



MCS-48™ Based Analog Processor

the system designer. The four components provide:

a. An eight bit microprocessor
b. 64 bytes of RAM
c. 1024 bytes of UV erasable PROM
d. A timer/event counter
e. 16 digital I/O pins
f. 2 testable input pins
g. An interrupt capability
h. 16 eight bit analog inputs
i. 2 eight bit analog outputs

The MCS-48 communicates with the D/A and A/D converters in a memory mapped mode (i.e., it treats the devices as if they were external RAM). By setting an address in either R_0 or R_1 and then executing a MOVX the software can transfer data between the accumulator and the analog I/O. When the MCS-48 executes the MOVX instruction it first sends the eight bit address out on the bus and strobes it into the 8212 latch with the ALE (Address Latch Enable) signal. After the address is latched, the MCS-48 uses the same bus to transfer data to or from the accumulator. If data is being sent out (MOVX ∂Rj, A) the $\overline{WR}$ strobe is used; if the data is being moved into the accumulator (MOVX A, ∂Rj) the $\overline{RD}$ strobe is used. The one shots on the $\overline{WR}$ line are used to delay the write strobe of the MCS-48 to meet the data set up specifications of the MP-10.

In order to provide reset capability for the analog devices without dedicating an I/O pin from the MCS-48, special addresses are used as reset channels. Executing any MOVX with an address of 0XXXXXXX will reset the A/D module; a similar operation with an address of X1XXXXXX will reset the D/A; a MOVX with an address of 01XXXXXX will reset both devices. All data transfers are accomplished with the upper two bits of the address field equal to 10. A summary of the addressing of the analog devices is shown in Table 1. Notice that except for an initialization channel for the D/A (which must be written to following a reset to initialize its internal logic) all channels involve some form of data transfer.

Table 1. Analog Interface Addresses

INPUT OR OUTPUT	
0 X X X X X X X X	Reset A/D
X 1 X X X X X X	Reset D/A
INPUT	
0 0 1 1 n n n n	Read A/D Channel n n n n
OUTPUT	
1 0 1 1 0 0 0 1	Initialize D/A
1 0 1 1 0 0 0 0	Write Channel 1
1 0 1 1 0 0 1 0	Write Channel 2

As was mentioned previously, the MP-20 was designed to use the READY line of the 8080A. Obviously this presents a problem since the MCS-48 does not support a READY line (with its attendant requirement of entering WAIT state). The necessity of a READY input can be overcome by performing a read operation to set the channel address, waiting the required delay (35 μsec for a gain of two) and then performing a second read to actually obtain the data. The second read will read in the data from the channel selected by the first read irrespective of the channel selected for the second read. Thus it is possible to use the second read to set up the channel for the third read. Each read can read in the current channel and select the next channel for conversion.

The MP-20 is shown in Figure 6 strapped to input 16 single ended ±5 volts signals. Programs which were used to test this configuration are shown in Figure 7. The first of these programs uses the D/A converter to generate sawtooth waveforms by outputting an incrementing value to the D/A converters. The second program scans the analog inputs and stores their digital values in a table located in RAM.

```
LOC  OBJ       SEQ     SOURCE STATEMENT

                 0
                 1
                 2 ; -----------------------------
                 3 ; TEST PROGRAM FOR ANALOG OUTPUT
                 4 ;   THIS PROGRAM OUTPUTS A SAW-
                 5 ;   TOOTH WAVEFORM BY OUTPUTING
                 6 ;   AN INCREMENTING PATTERN.
                 7 ; -----------------------------
                 8
                 9 ; -------
                10 ; EQUATES
                11 ; -------
                12
00B3             13 INITCH  EQU    0B3H    ; D/A INITIALIZATION CHANNEL
0000             14 INITDT  EQU    00H     ; D/A INITIALIZATION DATA
00B0             15 DATCH   EQU    0B0H    ; D/A DATA CHANNEL
                 16
                 17 ; -------------
                 18 ; START OF TEST
                 19 ; -------------
0100             20         ORG    100H
                 21                        ; INITIALIZE D/A
0100 2300        22 START:  MOV    A,#INITDT
0102 B0B3        23         MOV    R0,#INITCH
0104 90          24         MOVX   @R0,A
                 25                        ; TEST LOOP-OUTPUT SAWTOOTH
0105 B0B0        26 LOOP:   MOV    R0,#DATCH
0107 17          27         INC    A
0108 90          28         MOVX   @R0,A
0109 2405        29         JMP    LOOP
                 30                        ; END OF PROGRAM
                 31         END
```

Figure 7a. D/A Exercise Program

```
LOC  OBJ        SEQ     SOURCE STATEMENT

                  0
                  1
                  2 ; ----------------------------------------
                  3 ; TEST PROGRAM FOR ANALOG INPUT
                  4 ;   THIS PROGRAM SCANS THE INPUT CHANNELS
                  5 ;   AND STORES THE READINGS IN A TABLE
                  6 ;   STARTING AT BUFF.
                  7 ; ----------------------------------------
                  8
                  9 ; -------
                 10 ; EQUATES
                 11 ; -------
                 12
0020             13 BUFF    EQU     20H     ; START OF BUFFER
000F             14 MAXCH   EQU     15      ; NO OF ANALOG INPUTS
00B0             15 AINCH   EQU     0B0H    ; BASE ADDRESS OF ANALOG INPUTS
0005             16 TICK    EQU     5       ; EXECUTION TIME OF DJNZ
                 17
                 18 ; ------------
                 19 ; START OF TEST
                 20 ; ------------
0100             21         ORG     100H
                 22                         ; SETUP TO SCAN ANALOG INPUTS
0100 B92F        23 START:  MOV     R1,#BUFF+MAXCH
0102 BB0F        24         MOV     R3,#MAXCH
0104 B8BF        25         MOV     R0,#(AINCH+MAXCH)
                 26                         ; SELECT CHANNEL 15
0106 80          27         MOVX    A,@R0
                 28                         ; WAIT >40 MICROSECONDS
0107 BC08        29         MOV     R4,#40/TICK
0109 EC09        30         DJNZ    R4,$
                 31                         ; NOW SCAN ANALOGS
010B C8          32 LOOP:   DEC     R0
                 33                         ; GET DATA
010C 80          34         MOVX    A,@R0
                 35                         ; MOVE INTO BUFFER
010D A1          36         MOV     @R1,A
                 37                         ; DECREMENT BUFFER POINT
010E C9          38         DEC     R1
                 39                         ; PAD 20 MICROSEC
010F BC04        40         MOV     R4,#20/TICK
0111 EC11        41         DJNZ    R4,$
                 42                         ; LOOP UNTIL DONE
0113 EB0B        43         DJNZ    R3,LOOP
                 44                         ; REPEAT TEST FOREVER
0115 2400        45         JMP     START
                 46                         ; END OF PROGRAM
                 47         END
```

Figure 7b. A/D Exercise Program

TABLE LOOKUP TECHNIQUES

In the previous section the interface between analog I/O devices and the MCS-48™ was discussed. In many applications involving analog I/O one quickly finds that nature is inherently nonlinear, and the mathematics involved in 'linearizing it' can tax the computational power of the microprocessor, particularly if it has other tasks to perform. Problems of this nature are good candidates for the use of tables.

As an example of how tables can be used as part of an analog output scheme, consider a system which requires an MCS-48 to output a variable frequency sinusoidal waveform. One method of performing this function would be to use the timer to generate an interrupt at a fixed rate of 256 times the desired output frequency. At each interrupt the appropriate value of the sine function could be calculated from the MacLaurin series:

$$\text{Sin } x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} \cdots \frac{(-1)^k \, x^{2k+1}}{(2K+1)!}$$

Where K is chosen to be large enough to provide the required accuracy.

The above calculation, although conceptually simple, would be time consuming and would severely limit the possible output frequencies which could be obtained. As an alternative to calculating these values in real time, the values could be precalculated off line and stored in a table. Upon each interrupt the MCS-48 would merely have to retrieve the appropriate value from the table and output it to the D/A converter. the MCS-48 provides a special instruction which can be used to access data in a table. If the table is stored in the last 256 bytes of the first kilobyte of MCS-48 memory then the table lookup can be performed by loading the independent variable (time in this case) into the accumulator and executing the instruction.

MOVP3 A, @ A

This instruction uses the initial contents of the accumulator to index into page 3 of program storage. The location pointed to is read and the contents placed in the accumulator. If (as is often the case) a table of fewer than 256 entries is required, then the table can be located in any page of program memory and the instruction:

MOVP A, @ A

can be used to retrieve data from the table. This instruction operates in the same manner as does the previous instruction except that the current page of program storage is assumed to contain the table.

If it is possible to devote slightly more of the microprocessor's time to the table look up process, then a much smaller table can often be utilized by taking advantage of interpolation to determine values of the function between values which are actual entries in the table. As an example of this

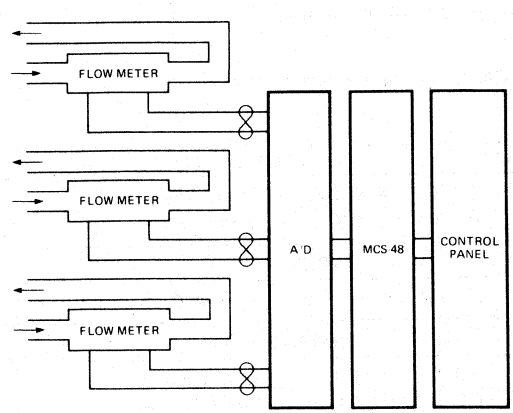


Figure 8. Flow Monitoring System

process consider the hypothetical system shown in Figure 8. The purpose of this system is to measure the flow through the three pipes, add them, and display the total flow on the control panel. The system consists of three flow meters which generate a differential voltage which is some function of flow, an A/D system with at least three differential inputs, an MCS-48, and a control panel. The schematic shown in Figure 6 could easily become part of this system, with the spare digital I/O of the MCS-48 used as an interface to the control panel. The simplicity of this system is clouded by the flow transducers, which are assumed to be not only nonlinear but also to require individual calibration (this is not an unreasonable assumption for a flow transducer). By using a table look up process and an 8748 the flow transducers can be calibrated and the results of the calibration tests stored directly in tables in the 8748. (The 8748 has a PROM in place of the ROM of the 8048 and thus makes such 'one off' programming practical.)

The results which might be obtained from calibrating one of the flow meters is shown in Figure 9. The results are plotted as gals/hour versus the measured voltage generated by the transducer. The voltage is shown in hexadecimal form so that it corresponds directly to the digital output of the analog to digital converter. The flow required to generate seventeen evenly spaced voltages (0H-100H in steps of 10H) has been measured and plotted. This information is shown in tabular form in Figure 10. It is necessary to generate a program which will convert any measured input from 00H to FFH into the flow in units which can be interpreted by a human operator. This can easily be done by simple interpolation.

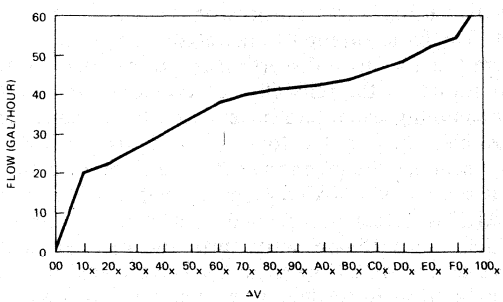


Figure 9. Flow Calibration Curve

TRANSDUCER VOLTAGE (HEX)	00	10	20	30	40	50	60	70	80	90	A0	B0	C0	D0	E0	F0	100
MEASURED FLOW (GAL/HOUR)	0	10	22	26	30	34	38	40	41	42	43	45	48	49	53	56	63

Figure 10. Tabulated Flow Data

The eight bits of independent variable (voltage) can be looked on as two four bit fields. The most significant four bits (7-4) will be used to retrieve one of the table values. The lower four bits (3-0) will be used to interpolate between this value and the value retrieved from the next higher location in the table. If the upper four bits are given the symbol I and the lower four bits the symbol N, then the interpolation can be expressed as:

$$F(x) = F(I) + \frac{N}{16}\,[F(I+1) - F(I)]$$

Where x is the measured voltage and F(x) is the corresponding flow.

If, as an example, the transducer voltage was measured as 48H then the flow (ref. Figure 10) would be:

$$F = 30 + \frac{8}{16}\,(34-30) = 32$$

A subroutine which implements this calculation is shown in Figure 11. Before it is called the independent variable (V) is placed in the accumulator and register R1 is set to point at the first value in the table. Aside from simple additions and subtractions the only arithmetic required is to multiply two values and then divide them by 16. The multiplication is handled via a subroutine which is also shown in Figure 11. The division by 16 can be performed by a four place right shift followed by a rounding operation. The routine shown will handle a monotonic increasing function of a single independent variable. Fairly simple modifications are required for nonmonotonic functions. Functions of two variables can be handled by interpolating on a plane rather than along a straight line. Although this is more time consuming, requiring an interpolation for each of the independent variables and a third to interpolate the final answer, it still provides a simple means of quickly calculating the required function. The use of tables can offer a powerful technique for function evaluation to the designer.

RECEIVING SERIAL CODE–BASIC APPROACHES

Many microprocessor based systems require some form of serial communication. Serial communication is extensively used because it allows two or more pieces of equipment to exchange information with a minimal number of interconnecting wires. The minimization of interconnecting wires results in simpler, cheaper, interconnects because fewer (or smaller) cables and connectors are required. Since the required number of drivers and receivers required is reduced, it can become economically feasible to provide much higher noise immunity

```
LOC  OBJ      SEQ     SOURCE STATEMENT

                 0  ; ****************************************
                 1  ;
                 2  ;       APPROX
                 3  ;         AT ENTRY R1 POINTSAT TABLE
                 4  ;                A HAS INDEPENDENT VARIABLE
                 5  ;
                 6  ; ****************************************
                 7
                 8  ; -------
                 9  ; EQUATES
                10  ; -------
                11
0000            12 RX0     EQU     R0      ; POINTER 0
0001            13 RX1     EQU     R1      ; POINTER1
0002            14 AEX     EQU     R2      ; EXTENSION OF A REGISTER
0003            15 COUNT   EQU     R3      ; COUNTER
0004            16 TEMP    EQU     R4      ; TEMP STORAGE
                17
                18 ; -------------
                19 ; APPROXIMATION
                20 ; -------------
                21
0100            22         ORG     100H
                23                         ; POINT RX0 AT TEMP
0100 B804       24 APPROX: MOV     RX0,#TEMP
                25                         ; TEMP=N AND 0FH
                26                         ; A=P AND 0FH
0102 B000       27         MOV     @RX0,#0
0104 30         28         XCHD    A,@RX0
0105 47         29         SWAP    A
                30                         ; RX1=BASE+A
0106 69         31         ADD     A,RX1
0107 A9         32         MOV     RX1,A
                33                         ; RX1=TABLE(P)
                34                         ; A=TABLE(P+1)
0108 E3         35         MOVP3   A,@A
0109 29         36         XCH     A,RX1
010A 17         37         INC     A
010B E3         38         MOVP3   A,@A
                39                         ; A-TABLE (P+1)-TABLE(P)
010C 37         40         CPL     A
010D 69         41         ADD     A,RX1
010E 37         42         CPL     A
                43                         ; A=N*A/16
010F 341D       44         CALL    MULT
0111 B802       45         MOV     RX0,#AEX
0113 30         46         XCHD    A,@RX0
0114 47         47         SWAP    A
0115 2A         48         XCH     A,AEX
0116 7219       49         JB3     ADJUST
0118 2A         50         XCH     A,AEX
0119 2A         51 ADJUST: XCH     A,AEX
011A 17         52         INC     A
                53                         ; A=A+TABLE(P)
011B 69         54         ADD     A,RX1
                55                         ; RETURN

LOC  OBJ      SEQ     SOURCE STATEMENT

011C 83         56         RET
                57
                58
                59 ; --------
                60 ; MULTIPLY
                61 ; --------
                62                         ; SET UP COUNT AND AEX
011D BB08       63 MULT:   MOV     COUNT,#8
011F BA00       64         MOV     AEX,#0
                65                         ; CLEAR CARRY
0121 97         66 LOOPA:  CLR     C
                67                         ; IF MULTIPLIER (0) <> 1 THEN SHIFT PRODUCT
0122 122B       68 LOOPB:  JB0     SSUM
0124 2A         69         XCH     A,AEX
0125 67         70         RRC     A
0126 2A         71         XCH     A,AEX
0127 67         72         RRC     A
                73                         ; LOOP UNTIL DONE
0128 EB22       74         DJNZ    COUNT,LOOPB
012A 83         75         RET
                76                         ; ELSE ADD MULTIPLIER AND SHIFT PRODUCT
012B 2A         77 SSUM:   XCH     A,AEX
012C 60         78         ADD     A,@RX0
012D 67         79         RRC     A
012E 2A         80         XCH     A,AEX
012F 67         81         RRC     A
                82                         ; LOOP UNTIL DONE
0130 EB21       83         DJNZ    COUNT,LOOPA
0132 83         84         RET
                85
                86
                87 ; --------------------
                88 ; TABLE TO TEST PROGRAM
                89 ; --------------------
                90
0300            91         ORG     300H
                92
0300 00         93 TABLE:  DB      00      ; THIS TABLE IS FROM FIG 10
0301 0A         94         DB      10
0302 16         95         DB      22
0303 1A         96         DB      26
0304 1E         97         DB      30
0305 22         98         DB      34
0306 26         99         DB      38
0307 28        100         DB      40
0308 29        101         DB      41
0309 2A        102         DB      42
030A 2B        103         DB      43
030B 2D        104         DB      45
030C 30        105         DB      48
030D 31        106         DB      49
030E 35        107         DB      53
030F 38        108         DB      56
0310 3F        109         DB      63
               110
               111         END
```

Figure 11. Table Lookup With Interpolation

with more sophisticated (and expensive) line terminators. The final, and usually most persuasive, argument in favor of serial communication is that it may be the only method available to accomplish the job. The obvious example of this is telecommunications where it is necessary to encode parallel information into serial format in order to communicate via the telephone network. The intent of this section is to show how the facilities of the MCS-48™ can be brought to bear on the problem of serial communication.

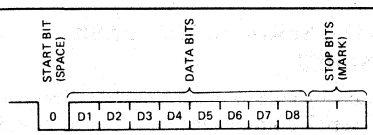


Figure 12. Serial ASCII Code

Probably the most common form of serial communication is that used by the obiquitous Teletype-serial ASCII. This format, shown in Figure 12, consists of a START bit (0 or SPACE) followed by eight data bits which are in turn followed by two STOP bits (1 or MARK). In actual practice the eighth data bit usually consists of even parity on the remaining seven data bits; for the purposes of this discussion the eighth bit will be considered only as data. A minor variation of this format deletes one of the STOP bits. An algorithm which might be used to sample serial data under software control using a microprocessor is shown in Figure 13. The basic intent of this algorithm is to minimize the effects of distortion and transmission rate variations on the reliability of the communication by sampling each data bit as close to its center as possible. Upon entry to this routine the software first samples the incoming data in a tight loop until it is sensed as a MARK (logical one). As soon as a MARK is detected, a second loop is entered during which the software waits until the received data goes to a SPACE (logical zero). The purpose of this construction is to detect as accurately as possible the leading edge of the START bit. This instant of time will be used as a reference point for sampling all of the following bits in the character. After sensing the leading edge of the START bit a wait of one half the expected bit time is implemented. The period of the incoming signal is called P for convenience. At the end of this wait the serial line is tested—if it is MARK then the START bit was

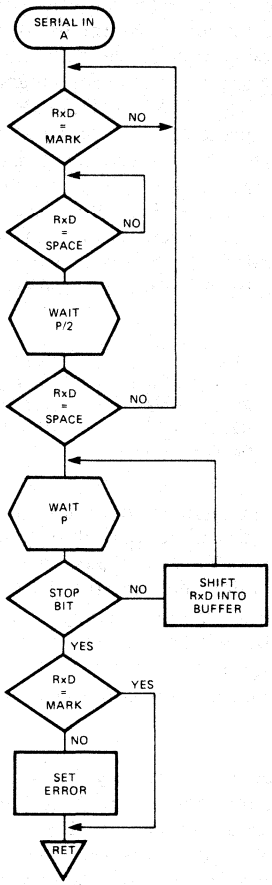


Figure 13. Sample Serial Input Routine

invalid and the process is reinitialized. If the line is still a SPACE, then the START bit is assumed to be valid and a delay of one bit time is started. At the completion of the delay the first data bit is sampled and a new delay of one bit time is initiated. This process is repeated until all eight data bits have been sampled. The last bit sampled is checked to determine if it is a valid STOP bit (a MARK). If it is, the character is assumed to be valid; if it is not, the character has a framing error and is probably invalid. A listing of a program which implements the above procedure is shown in Figure 14.

A disadvantage of the approach outlined in Figure 13 is that while the processor is inputting data serially it must totally dedicate itself to this task. Accurate timing can only be maintained if the program remains in a tight wait loop without allowing itself to be diverted to other functions. During reception of a character from a Teletype the processor will spend only a 100μsecs or so processing data and the rest of the 100 millisecs waiting to do the processing at the right time. This lack of efficiency (approximately 0.1%) in the utilization of processing power is why devices such as the 8251 USART find broad application in microprocessor systems.

```
LOC  OBJ       SEQ        SOURCE STATEMENT

                  0 ; ****************************
                  1 ;
                  2 ;      SIMPLE SERIAL INPUT
                  3 ;      -THIS CODE ASSUMES RXD IS
                  4 ;       CONNECTED TO PIN T0
                  5 ;
                  6 ; ****************************
                  7
                  8 ; -------
                  9 ; EQUATES
                 10 ; -------
                 11
0002             12 COUNT  EQU     R2      ; COUNTER
0008             13 BITNO  EQU     8       ; NO OF BITS TO RECEIVE
0002             14 DLYHI  EQU     2       ; HI DLY COUNT
00A4             15 DLYLO  EQU     0A4H    ; LO DLY COUNT
                 16
0100             17        ORG     100H
                 18                        ; LOOP UNTIL RXD=MARK
0100 2600        19 SERIN: JNT0    $
                 20                        ; NOW LOOP UNTIL RXD=SPACE
0102 3602        21        JT0     $
                 22                        ; WAIT 1/2 BIT TIME
0104 341C        23        CALL    HBIT
                 24                        ; IF FALSE START REINTIALIZE
0106 3600        25        JT0     SERIN
                 26                        ; ELSE SET BIT COUNT
0108 BA09        27        MOV     COUNT,#BITNO+1
                 28                        ; WAIT 1 BIT TIME
010A 341C        29 LOOP:  CALL    HBIT
010C 341C        30        CALL    HBIT
                 31                        ; DECREMENT COUNT
                 32                        ; -IF ZERO EXIT WITH CARRY SET ON
                 33                        ; -FRAMING ERROR
010E EA15        34        DJNZ    COUNT,LOAD
0110 97          35        CLR     C
0111 3614        36        JT0     EXIT
0113 A7          37        CPL     C
0114 83          38 EXIT:  RET
                 39                        ; LOAD DATA
0115 97          40 LOAD:  CLR     C
0116 2619        41        JNT0    LLLA
0118 A7          42        CPL     C
0119 67          43 LLLA:  RRC     A
                 44                        ; AND LOOP
011A 240A        45        JMP     LOOP
                 46
                 47 ; ----------------------
                 48 ; DELAY ONE HALF BIT TIME
                 49 ; ----------------------
                 50
                 51                        ; SET UP LOOP
011C BC02        52 HBIT:  MOV     R4,#DLYHI
                 53                        ; LOOP UNTIL TIME DONE
011E BBA4        54 HLOOP: MOV     R3,#DLYLO
0120 EB20        55        DJNZ    R3,$
0122 EC1E        56        DJNZ    R4,HLOOP
0124 83          57        RET
                 58                        ; END OF PROGRAM
                 59        END
```

Figure 14. Simple Serial Input

The 8251 USART is simple to interface to the MSC-48. Figure 15 shows such an interface. The USART requires a high speed clock (CLK), an initilization signal (RESET), data clocks (TxC and RxC), and data in order to operate. A circuit showing the connection of an 8748 to an 8251 USART is shown in Figure 15. In the circuit shown the high speed clock (which is used for internal sequencing by the USART) is provided by con-

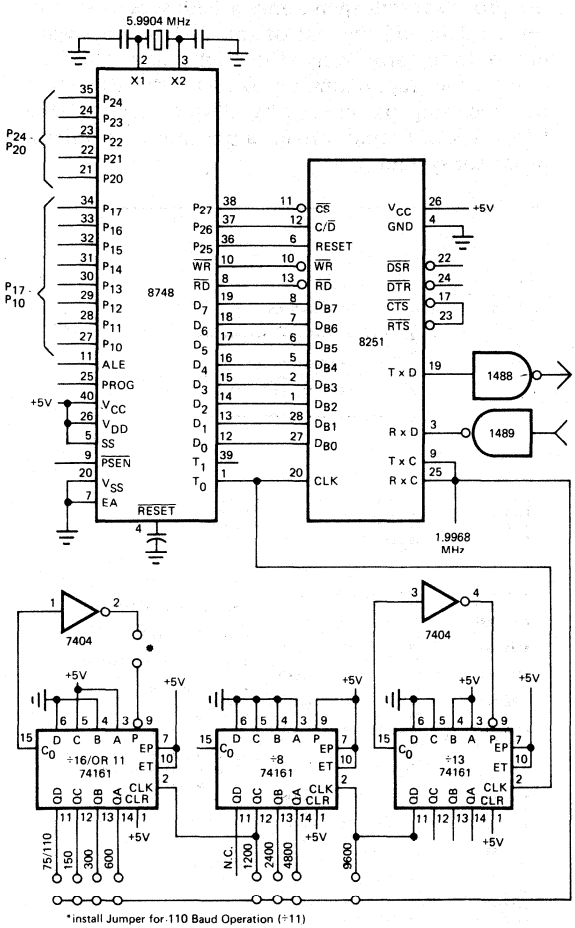


Figure 15. MCS-48™ to 8251 Interface

necting the CLK signal of the USART to the T0 pin of the MCS-48. The T0 pin of the MCS-48 can either be used as a directly testable input pin or it can become, under program control, an output pin which oscillates at one third of the crystal frequency. (Note that once this pin is designated by the software to be an output it will remain so until the system is reset.) In Figure 15 the crystal frequency is 5.9904 MHz so the clock provided to the 8251 is 1.9968 MHz, which conforms to its specifications.

The initialization signal to the USART (RESET) is provided programmatically by manipulation of bit 5 of port 2. It was necessary to place the reset of the 8251 under program control for two reasons. The first reason is that the MCS-48 does not supply a reset signal to other devices. The reason for this is that it was felt to be more useful to provide another pin of I/O function instead of a RESET OUT signal from the MCS-48. Although this situation could have been circumvented by the use of an externally generated reset which drove both the MCS-48 and the 8251, the second reason for program control of the reset to the USART still stands. The USART requires the presence of the CLK signal during reset in order to properly initialize itself. The ENT0 CLK instruction which the MCS-48 must execute before the 8251 will receive the CLK can obviously not be executed until after the system reset has ended. Reset of the USART can be accomplished by the following code segment:

```
      ENT0   CLK          ;TURN ON CLOCK
      ORL    P2, #00100000B ;START RESET
      M0V    R2, #DELAY    ;DELAY USART
LOOP: DJNZ   R2, LOOP      ;RESET TIME
      ANL    P2, #11011111B ;END RESET
```

This code first enables the clock, then asserts the reset signal of a time period determined by the constant DELAY. The delay invoked is (10 + 5*DELAY) microseconds for DELAY >0. The USART requires a reset of approximately 6 CLK periods so DELAY is chosen to be 1 which ensures adequate reset timing. Note that for delays this short, NOP instructions could also be used to time the pulse.

The data clocks required by the USART are provided by the modem if the USART is operated in the synchronous mode. In the more common asynchronous mode, however, these clocks must be provided by circuitry associated with the 8251.

The 5.9904 MHz crystal was chosen because the resulting 1.9968 MHz clock to the USART can be evenly divided to provide transmit and receive clocks to the USART. Assuming the USART is in the x16 mode (i.e. it requires data clocks 16 times the baud rate) the 1.9968 MHz signal can be divided by 13 to generate the proper clock rate for 9600 baud operation. This 9600 baud clock can be further divided to give 4800, 2400, 1200, 600, and 300 baud signals. The 1200 baud signal can be divided by 11 to give a 109.1 baud signal which is within 1% of the 110 baud required by Teletypes.

The MCS-48 communicates with the 8251 in a memory mapped mode (i.e. as if the 8251 were external RAM). The instructions available to do this are MOVX ∂Rj, A which stores the contents of the accumulator at the external RAM location addressed by Rj (j=0 or 1), and its complement, the MOVX A, @ Rj instruction which moves data from the external RAM into the accumulator. Since the MCS-48 multiplexes addresses and data on the same eight bit bus an external latch would be required in order to address the USART with

```
LOC  OBJ        SEQ     SOURCE STATEMENT

                  0 ;-------------------------------
                  1 ; SERIAL TEST
                  2 ;  THIS CODE INTIALIZES THE USART
                  3 ;  AND TRANSMITS AN INCREMENTING
                  4 ;  PATTERN. HARDWARE SHOWN IF FIG 15.
                  5 ;-------------------------------
                  6
                  7 ; -------
                  8 ; EQUATES
                  9 ; -------
                 10
0020              11 MCLR   EQU    20H     ; USART RESET ADDRESS
0001              12 DLY    EQU    01H     ; USART RESET DELAY
007F              13 UCON   EQU    7FH     ; USART CONTROL ADDRESS
00CE              14 MODE   EQU    0CEH    ; USART MODE
0021              15 CMD    EQU    21H     ; USART CMD
007F              16 STAT   EQU    7FH     ; USART STATUS
0001              17 VAL    EQU    R1      ; TEST VALUE
00BF              18 MASK   EQU    0BFH    ; CHANGES CMD TO DATA CHANNEL
                  19
0100              20        ORG    100H
                  21                       ; TURN ON CLOCK
                  22                       ; AND RESET USART
0100 75           23 TEST:  ENT0   CLK
0101 BA20         24        ORL    P2,#MCLR
0103 BA01         25        MOV    R2,#DLY
0105 EA05         26 LOOP:  DJNZ   R2,LOOP
0107 9ADF         27        ANL    P2,#(NOT MCLR)
                  28                       ; SELECT USART CONTROL
0109 237F         29        MOV    A,#UCON
010B 3A           30        OUTL   P2,A
                  31                       ; SEND MODE AND COMMAND
010C 23CE         32        MOV    A,#MODE
010E 90           33        MOVX   @R0,A   ; (CONTENTS OF R0 UNIMPORTANT)
010F 2321         34        MOV    A,#CMD
0111 90           35        MOVX   @R0,A
                  36                       ; DO FOREVER
                  37                       ;    SELECT USART STATUS
                  38                       ;       IF TXRDY=1 THEN
                  39                       ;       DO;
                  40                       ;          OUTPUT VALUE;
                  41                       ;          INCREMENT VALUE;
                  42                       ;       END;
                  43                       ;    END;
0112 237F         44 TLP:   MOV    A,#STAT
0114 3A           45        OUTL   P2,A
0115 80           46        MOVX   A,@R0   ; (CONTENTS OF R0 UNIMPORTANT)
0116 67           47        RRC    A
0117 E612         48        JNC    TLP
0119 F9           49        MOV    A,VAL
011A 9ABF         50        ANL    P2,#MASK
011C 90           51        MOVX   @R0,A
011D 19           52        INC    VAL
011E 2412         53        JMP    TLP
                  54                       ; END OF PROGRAM
                  55        END
```

Figure 16. 8251 Test Program

R0 or R1. In order to minimize the circuitry in Figure 15 an approach utilizing some of the I/O pins of the MCS-48 to address the 8251 was chosen instead. By connecting the chip select ($\overline{CS}$) input of the 8251 to bit 7 of port 2 (P27) and similarly connecting the C/$\overline{D}$ address line of the 8251 to bit 6 of port 2 (P26) it is possible to address the 8251 without using R0 or R1. The instruction sequence to access the 8251 is to first reset P27 and set P26 to the appropriate state, use a MOVX instruction to perform the appropriate operation, and then finally set P27 to deselect the 8251. As a concrete example of this addressing, Figure 16 shows the code necessary to initialize the 8251 and output an incrementing test pattern on a status driven basis.

If more than one 8251 were to be added to the MCS-48, or if other types of peripheral circuitry would be required (e.g. an 8253 timer to generate the data clocks) it would probably become desirable to add the circuitry necessary to use R0 or R1 to address the peripheral devices. The circuitry which has to be added to Figure 15 in order to make use of R0 or R1 to address the USART is shown in Figure 17. Note that only the changes to Figure 15 are shown. The additional component required is the 8212 eight bit latch. This latch is loaded, whenever a valid address is on the bus by the Address Latch Enable (ALE) signal provided by the MCS-48. During an external read or write cycle this address is used to address the 8251 in a linear select mode. In the circuit shown, the 8251 will be selected by any address with bit 1 a logical zero (XXXXXX0X) and the selection of control or data transfer (C/$\overline{D}$) will be based on bit zero of the address obtained from R0 or R1. Figure 18 shows the program of Figure 16 modified to utilize the addressing inherent in the MOVX instructions.

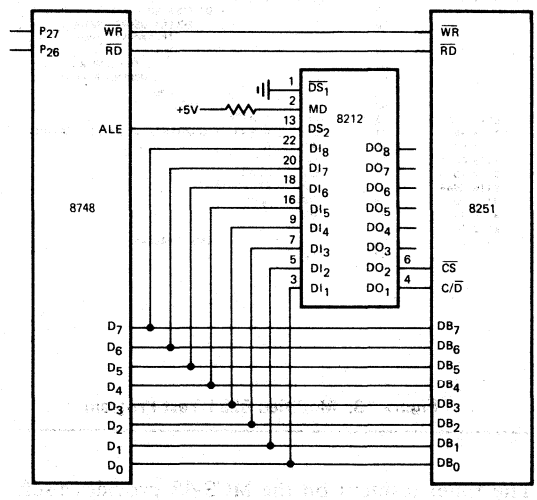


Figure 17. Modified MCS-48 to 8251 Interface

RECEIVING SERIAL CODE–A MORE SOPHISTICATED ALGORITHM

Although the USART does an admirable job of performing the serial I/O function for the MCS-48™, there are some situations where it can not be used. These situations may be caused by economic factors, such as an extremely cost sensitive design, or because the code which must be utilized cannot be accommodated by the USART. An example of of such a code will be discussed later. Recall that the principal objection to the approach to serial input shown in Figure 13 was that it consumes much of the processor's power by merely spinning in loops in order to wait preset time delays.

```
LOC  OBJ        SEQ     SOURCE STATEMENT

                  0 ;-------------------------------------
                  1 ; SERIAL TEST
                  2 ;  THIS CODE INTIALIZES THE USART
                  3 ;  AND TRANSMITS AN INCREMENTING
                  4 ;  PATTERN. HARDWARE SHOWN IF FIG 17.
                  5 ;-------------------------------------
                  6
                  7; -------
                  8; EQUATES
                  9; -------
                 10
0020             11 MCLR   EQU    20H     ; USART RESET ADDRESS
0001             12 DLY    EQU    01H     ; USART RESET DELAY
0003             13 UCON   EQU    03H     ; USART CONTROL ADDRESS
00CE             14 MODE   EQU    0CEH    ; USART MODE
0021             15 CMD    EQU    21H     ; USART CMD
0003             16 STAT   EQU    03H     ; USART STATUS
0001             17 VAL    EQU    R1      ; TEST VALUE
0000             18 DATA   EQU    00      ; USART DATA ADDRESS
                 19
0100             20        ORG    100H
                 21                       ; TURN ON CLOCK
                 22                       ; AND RESET USART
0100 75          23 TEST:  ENT0   CLK
0101 8A20        24        ORL    P2,#MCLR
0103 BA01        25        MOV    R2,#DLY
0105 EA05        26 LOOP   DJNZ   R2,LOOP
0107 9ADF        27        ANL    P2,#(NOT MCLR)
                 28                       ; SELECT USART CONTROL
0109 2303        29        MOV    A,#UCON
                 30                       ; SEND MODE AND COMMAND
010B 23CE        31        MOV    A,#MODE
010D 90          32        MOVX   @R0,A  ; (CONTENTS OF R0 UNIMPORTANT)
010E 2321        33        MOV    A,#CMD
0110 90          34        MOVX   @R0,A
                 35                       ; DO FOREVER
                 36                       ;   SELECT USART STATUS
                 37                       ;     IF TXRDY=1 THEN
                 38                       ;     DO;
                 39                       ;       OUTPUT VALUE;
                 40                       ;       INCREMENT VALUE;
                 41                       ;     END;
                 42                       ;   END;
0111 2303        43 TLP:   MOV    A,#STAT
0113 80          44        MOVX   A,@R0  ; (CONTENTS OF R0 UNIMPORTANT)
0114 67          45        RRC    A
0115 E611        46        JNC    TLP
0117 F9          47        MOV    A,VAL
0118 B800        48        MOV    R0,#DATA
011A 90          49        MOVX   @R0,A
011B 19          50        INC    VAL
011C 2411        51        JMP    TLP
                 52                       ; END OF PROGRAM
                 53        END
```

Figure 18. Modified 8251 Test Program

The timer resident on the MCS-48 provides a solution to this problem. Instead of spinning in a loop the program can set the timer for a given interval, start it, and proceed to other tasks. When the timer overflows, an interrupt will be generated to notify the software that the present time period has elapsed. An extension of the algorithm of Figure 13 which uses the timer in this fashion in shown in Figure 19. This algorithm is identical to the preceding one up until the detection of the leading edge of the start bit. At this point the timer is set to one half of the bit time (P) and a return is made to the calling program which can start additional processing. At the completion of this time interval a timer overflow interrupt is generated. When the first interrupt is detected, the serial line is checked to ensure that it is in a spacing condition (valid START bit). If it is, the timer is set to P (to sample the middle of the first data bit) and a return is made to the program which was running when the interrupt occurred. If the serial line has returned to the MARK state, a status flag is set to indicate an error and a return is made. On subsequent interrupt detection, the data is sampled, the timer is reinitiated, and control is returned to the program which was running when the interrupt occurred. When the last (i.e. STOP) bit is detected a completion flag is set and a return is made to the program running when the timer overflow occurred. By periodically checking the error and completion flags the running program can determine when the interrupt driven receive program has a character assembled for it.

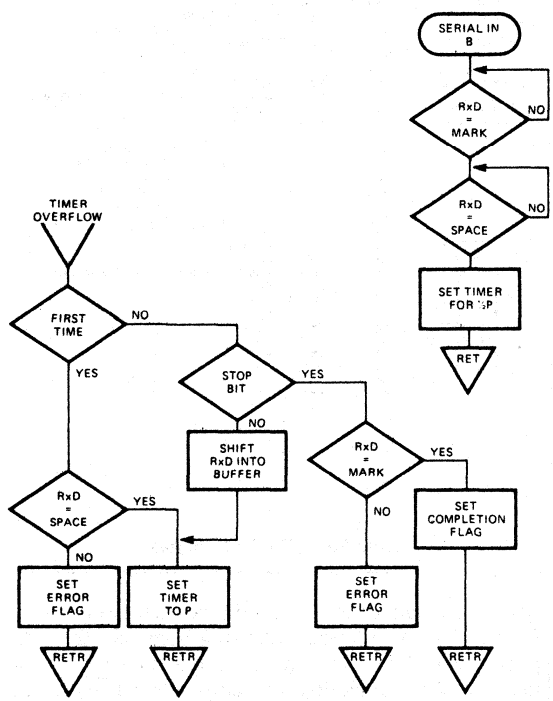


Figure 19. Improved Serial Input Routine

Using the timer to implement time delays as shown in Figure 19 results in considerable savings in processing time; two problems remain, however, which must be solved before an adequate software solution to the problem of receiving serial code can be found. The first problem is that even though the delays between bit samples are implemented via the timer rather than program loops the loop construction is still used to detect the leading edge of

the START bit. Although this results in the waste of processing power, the second problem is even more serious. For longer messages the required accuracy of the clocks becomes more and more stringent. Using the sampling technique discussed a cumulative error of one half a bit time in the time at which a bit sample is taken will result in erroneous reception. The maximum timing error which can be tolerated and yet still allow proper detection of an 11 bit ASCII character is then:

$$\text{Emax} = \frac{0.5*\text{BIT TIME}}{\text{CHARACTER TIME}} - \frac{0.5\text{P}}{11\text{ P}} = 4.5\%$$

where P is the period of single bit. The corresponding calculation for a 32 bit character yields:

$$\text{Emax} = \frac{0.5\text{P}}{32\text{ P}} = 1.6\%$$

Since this calculation does not allow for distortion on the signals, it is obvious that either extremely stable clocks will be required or a more tolerant algorithm must be devised. This problem is particularly serious at relatively high baud rates where the resolution of the counter (80μsecs with a 6 MHz crystal) becomes a significant percentage of the period of the received signal. At the 110 baud rate of the Teletype the 80μsec resolution of the clock allows a maximum accuracy of 0.33%; at 2400 baud this figure is reduced to 3.8%.

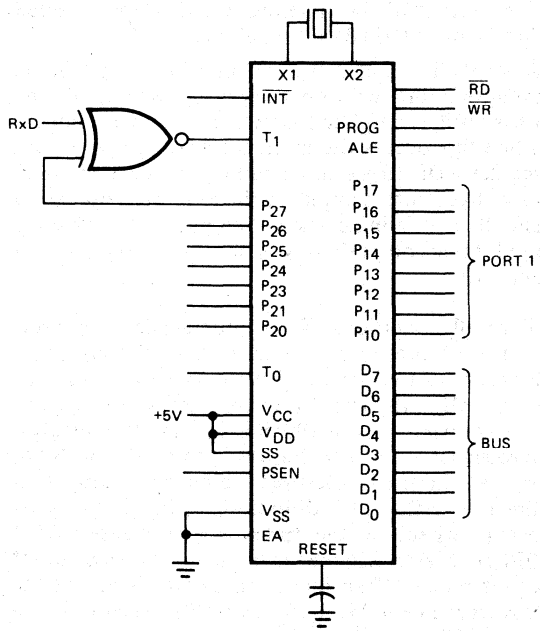


Figure 20. Detecting RxD Edges

Both efficient detection of the start bit and increased timing accuracy can be obtained if the MCS-48 can detect edges on the incoming received data (RxD). A hardware construct which allows this is shown in Figure 20.

The received data (RxD) is Exclusive NORed with bit seven of port two and fed into the TEST (T1) pin of the MCS-48. By manipulating P27 the program can now cause T1 to be either RxD or $\overline{\text{RxD}}$. (If P27 = 1 then T1 = RxD; if P27 = 0 then T1 = $\overline{\text{RxD}}$.) Note that not only can T1 be tested directly by the software but that it is the input which is used when the MCS-48 timer is in the event counter mode. The significance of this will be discussed later. The relationship between T1, P27, and RxD is given by the Boolean expression:

$$\overline{\text{T1}} = \text{P27} \cdot \overline{\text{RxD}} + \overline{\text{P27}} \cdot \text{RxD}$$

Figure 21 flowcharts a means of utilizing this hardware construct to avoid the necessity of wasting time in program loops to detect the leading edge of the start bit. The receive operation is initialized when the program desiring to receive serial data calls the INIT subroutine (Figure 21a). Since INIT is going to manipulate the timer the first action it performs is to disable the timer overflow interrupt. Its next step is to set P27 to a logical 1. Setting P27 in this manner causes the TEST 1 input to the MCS-48 to follow $\overline{\text{RxD}}$. By setting up the receive circuitry in this manner a high to low transition will occur on TEST 1 when the RxD goes from the MARKING to SPACING state (i.e. the START

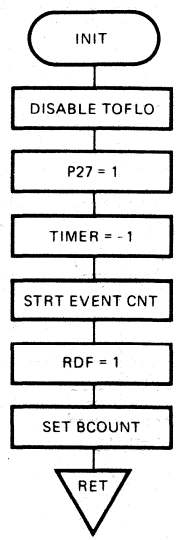


Figure 21a. Interrupt Driven Serial Receive Flowchart

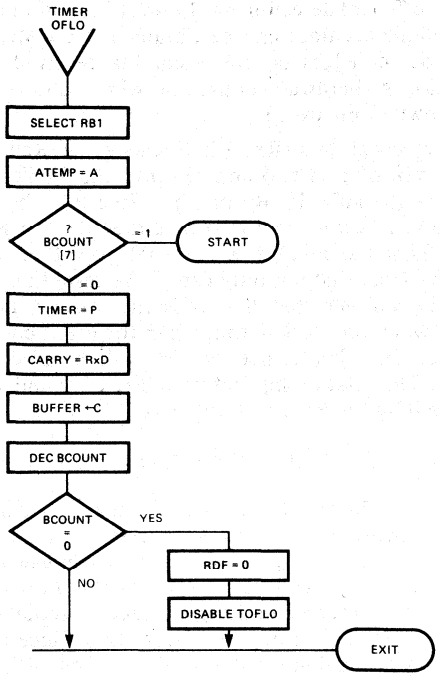


Figure 21b. Interrupt Driven Serial Receive Flowchart

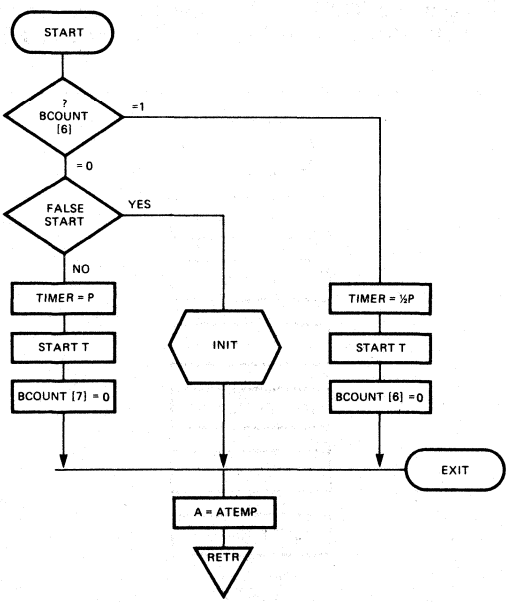
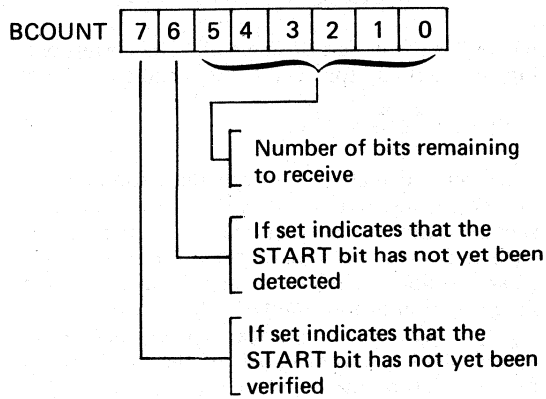


Figure 21c. Interrupt Driven Serial Receive Flowchart

bit occurs). By setting the timer to 0FF$_H$ and enabling it in the event count mode, the INIT routine sets up the MCS-48 to generate a timer overflow interrupt on the next MARK to SPACE transition of RxD (the TEST 1 input doubles as the event counter input). Before returning to the calling program the INIT routine sets a flag (RDF) which will be cleared by the receive program when the requested receive operation is complete. INIT also sets a value into a register called BCOUNT. The receive program interprets BCOUNT as follows:



In order to request the reception of the 11 bit ASCII code INIT would set BCOUNT to 11001011B. The start bit has been neither verified nor detected and 11 bits (1011B) are required.

After INIT is called the reception of the individual serial data bits will proceed on an interrupt driven basis until a complete character has been assembled. When this occurs the interrupt driven program will set the RDF (Receive Done Flag) to a zero to indicate that it has completed the requested operation and then terminate itself. The procedure which is used to accomplish this is shown in Figures 21b and 21c.

Since all operations of this program are the result of the occurence of a timer overflow interrupt, it is necessary to briefly review the interrupt structure of the MCS-48. There are two sources of interrupt; an external interrupt which is the result of a logical zero signal applied to the $\overline{INT}$ pin of the MCS-48, and an internal interrupt which is caused by a timer overflow condition. The timer overflow occurs whenever the timer is incremented from 0FF$_H$ to zero whether it be in the timer or event count mode. When one of these events occurs the hardware in the MCS-48 forces the execution of a CALL. This CALL has a preset address of location 3 if it is due to the external interrupt and location 7 if it is due to a timer overflow. If both of these

events occur simultaneously the external interrupt will take precedence. The CALL automatically saves the contents of the program counter for the running program and its PSW (program status word) on a stack the hardware maintains in RAM locations 8-23. Although the hardware saves the program counter and PSW, it remains the responsibility of any interrupt driven software to make absolutely certain that it does not modify any memory locations or registers which are being used by the main program. The most convenient way of ensuring this in the MCS-48 is to dedicate the second bank of registers (RB1) to the interrupt driven program. One of these registers has to be used to save the accumulator (which is not part of the register bank) but seven registers remain; including two which can be used as pointers to the rest of the RAM (R0 and R1). Note that if this approach is taken then these registers have to be allocated between the program which services the external interrupt and the one which services the timer overflow. This problem is somewhat alleviated by a hardware lockout which prevents the timer overflow interrupt from interrupting the external interrupt service routine and vice versa. This is implemented by locking out new interrupts between the time an interrupt is recognized and the time a RETR instruction is executed. The RETR instruction is like a normal RET (return from subroutine) except that the PSW as well as the program counter is restored. The RETR instruction can be very much thought of as a return from interrupt instruction in the MCS-48.

The receive program under discussion uses register bank 1 in the manner described. Whenever a timer overflow occurs (e.g. on the next MARK to SPACE transition of RxD after INIT is called), control is passed (by the hardware generated CALL) to the point labled TIMER OFLO in Figure 21b. This program segment immediately selects register bank 1 (RB1) and then saves the accumulator (A) in a location called ATEMP which is actually R7 of RB1. The program then tests bit seven of BCOUNT (R6 of RB1) to find out if a START bit has been verified (i.e. the edge of the START bit has first been detected and then verified to still be a SPACE one-half a bit time later. If BCOUNT [7] is a zero the START has been verified and the program proceeds to set the timer to P (the period of the serial bit), get the current serial data into the carry bit, and then shift the carry bit into a buffer. After saving the data the program decrements BCOUNT and tests it for zero. If BCOUNT is zero the receive operation is complete so the program sets RDF to a zero and disables timer overflow interrupts. Whether or not BCOUNT is zero, control is passed to EXIT where A is loaded with ATEMP and a RETR is executed. Note that since the state of the flip flop which selects RB1 is saved as part of the PSW, the execution of RETR automatically selects the register bank which was active when the interrupt occurred.

If BCOUNT [7] is still set when it is tested, control is passed to START (Figure 21c) where bit 6 is tested to determine if the START has been detected yet. If BCOUNT [6] is set it indicates that this is the first occurrence of a timer overflow since the receive process was initialized by the INIT subroutine. If this is so, the program assumes that the START bit has just started and therefore it sets the timer to one-half of a bit time (1/2 P), starts the timer in the timer mode, and clears BCOUNT [6] to indicate that the START bit has been detected. The next overflow will again result in the execution of the program in Figure 21b and again BCOUNT [7] will be found to be set. This time, however, BCOUNT [6] will be reset and the program will know that it should test the START bit to ensure that it is still a SPACE. This test is performed and if successful the timer is set for a bit period P and BCOUNT [7] is reset so that on the next occurrence of a timer overflow the program will know that it should start assembling serial bits into a character. If the test is unsuccessful, the subroutine INIT is used to reinitialize the receive program. In either case control is passed to EXIT where a return from interrupt mode occurs.

This receive program, listings of which appear in Figure 22, allows the reception of serial characters transparently to the main running software. After INIT is called the main program has only to check RDF periodically to find out if there is data in the buffer for it. It would be fairly easy to 'double buffer' this operation by providing a buffer which the receive program uses to deserialize the incoming code and a second buffer to store the assembled character. If the program would reinitialize itself upon completion, the reception of a string of characters could proceed in much the same way as it would if a status driven USART were being used.

Although this program solves the first problem of software controlled reception (lack of efficiency) the second problem—sensitivity to frequency variations—remains. An example of a code which would be susceptible to this problem is the 31,26 BCH code commonly used in supervisory control systems. (A supervisory control system is, in essence, a remote control system which allows a human or computer operator the control of a system via a serial communications link.) The BCH codes are used because of their error detection capabilities and are a class of cyclical redundancy

```
LOC  OBJ       SEQ     SOURCE STATEMENT

                  1 ; ****************************************
                  2 ;
                  3 ;       SERIAL INPUT USING THE MCS-48
                  4 ;        THIS CODE ASSUMES HARDWARE
                  5 ;        SHOWN IN FIG 20. TO USE
                  6 ;        THIS ROUTINE CALL INIT.
                  7 ;        WHEN RDF=0 THE ASSEMBLED
                  8 ;        CHARACTER WILL BE IN SERBUF
                  9 ;
                 10 ; ****************************************
                 11
                 12 ; -------
                 13 ; EQUATES
                 14 ; -------
                 15
0007             16 ATEMP  EQU   R7    ; STORAGE FOR A DURING INTERUPT
0006             17 BCOUNT EQU   R6    ; CONTAINS NUMBER OF BITS IN MSG
0002             18 COUNT  EQU   R2    ; UTILITY COUNTER
0000             19 RX0    EQU   R0    ; POINTER
0008             20 BITNO  EQU   8     ; NUMBER OF BITS
0029             21 P      EQU   41    ; SAMPLE PERIOD
0020             22 SERBUF EQU   20H   ; SERIAL BUFFER
0024             23 RDF    EQU   24H   ; RECEIVE DONE FLAG
                 24
                 25 ; ---------------------------------------
                 26 ; CONTROL PASSED HERE WHEN TIMER OFLO OCCURS
                 27 ; ---------------------------------------
                 28
0007             29        ORG   07H
                 30                    ; /*ENTER INTERRUPT MODE*/
0007 DS          31 IMVEC: SEL   RB1
0008 AF          32        MOV   ATEMP,A
                 33                    ; IF BCOUNT[7]=0 THEN
0009 FE          34        MOV   A,BCOUNT
000A F223        35        JB7   START
                 36                    ; DO;
                 37                    ;    TIMER=P;
000C 23D7        38        MOV   A,#-P
000E 62          39        MOV   T,A
                 40                    ;    START TIMER
000F 55          41 SLLB:  STRT  T
                 42                    ;    /*CARRY=RXD*/
                 43                    ;    CARRY=P27 XNOR TEST1;
0010 0A          44        IN    A,P2
0011 F7          45        RLC   A
0012 5615        46        JT1   TISRD
0014 A7          47        CPL   C
                 48                    ;    /*SHIFT CARRY INTO BUFFER*/
                 49                    ;    RX0=SERBUF:
                 50                    ;    RSHFT MEM(RX0);
0015 B820        51 TISRD: MOV   RX0,#SERBUF
0017 20          52 SLOOP: XCH   A,@RX0
0018 67          53        RRC   A
0019 20          54        XCH   A,@RX0
                 55                    ;    BCOUNT=BCOUNT-1;
                 56                    ;    IF BCOUNT=0 THEN
001A EE3F        57        DJNZ   BCOUNT,SEXIT
                 58                    ;    DO;
                 59                    ;       RDF=0;
                 60                    ;       DISABLE EX INT;
                 61                    ;    END;
001C B824        62        MOV   RX0,#RDF
001E 27          63        CLR   A
001F A0          64        MOV   @RX0,A
0020 35          65        DIS   TCNTI
                 66                    ; END;
0021 043F        67        JMP   SEXIT
                 68                    ; ELSE
                 69                    ; DO;
                 70                    ;    IF BCOUNT[6]=0 THEN

0023 FE          71 START: MOV   A,BCOUNT
0024 D237        72        JB6   SLLC
                 73                    ;    DO;
                 74                    ;       IF TEST1=0 THEN
0026 5635        75        JT1   SLLD
                 76                    ;       DO;
                 77                    ;          TIMER=P;
                 78                    ;          START TIMER;
                 79                    ;          P27=0;
                 80                    ;          EN I
                 81                    ;          BCOUNT[7]=0;
                 82                    ;       END;
0028 23D7        83        MOV   A,#-P
002A 62          84        MOV   T,A
002B 55          85        STRT  T
002C 9A7F        86        ANL   P2,#7FH
002E 05          87        EN    I
002F FE          88        MOV   A,BCOUNT
0030 537F        89        ANL   A,#7FH
0032 AB          90        MOV   BCOUNT,A
0033 043F        91        JMP   SEXIT
                 92                    ;       ELSE
                 93                    ;       DO;
                 94                    ;          CALL INIT;
                 95                    ;       END;
0035 1441        96 SLLD:  CALL INIT
                 97                    ;    ELSE
                 98                    ;    DO;
                 99                    ;       TIMER=P/2;
                100                    ;       START TIMER;
                101                    ;       BCOUNT[6]=0;
                102                    ;    END;
0037 23EC       103 SLLC:  MOV   A,#-(P/2)
0039 62         104        MOV   T,A
003A 55         105        STRT  T
003B FE         106        MOV   A,BCOUNT
003C 53BF       107        ANL   A,#0BFH
003E AE         108        MOV   BCOUNT,A
                109                    ; END;
                110                    ; /*EXIT INTERUPT MODE*/
003F FF         111 SEXIT: MOV   A,ATEMP
0040 93         112        RETR
                113
                114 ; ---------------------------
                115 ; INTIALIZE ROUTINE-
                116 ;       STARTS RECEIVE PROCESS
                117 ; ---------------------------
                118
                119                    ; INIT:
                120                    ;    PROCEDURE;
                121                    ;       DO;
                122                    ;          DISABLE INTERRUPTS;
                123                    ;          P27=1;
                124                    ;          TIMER=-1;
                125                    ;          START EVENT COUNT;
                126                    ;          RDF=1;
                127                    ;          BCOUNT=0C0H OR BITNO
                128                    ;       END;
                129                    ;    END INIT;
0041 35         130 INIT:  DIS   TCNTI
0042 8A80       131        ORL   P2,#80H
0044 23FF       132        MOV   A,#-1
0046 62         133        MOV   T,A
0047 45         134        STRT  CNT
0048 B824       135        MOV   RX0,#RDF
004A B001       136        MOV   @RX0,#01H
004C B81E       137        MOV   RX0,#1EH      ; POINT AT BCOUNT
004E B0C8       138        MOV   @RX0,#(0C0H OR BITNO)
0050 25         139        EN    TCNTI
0051 83         140        RET
                141                    ;END OF PROGRAM
                142
                143        END
```

Figure 22. Interrupt Driven Serial Receive Program

codes such as those used in synchronous data communications (e.g. BISYNC or SDLC). BCH codes, named for their originators Bose, Chaudhuri, and Hocquenghem, are characterized by having a length of $n=2^m-1$. The number of redundant check bits can be mt where t is a positive integer (clearly mt $\leqslant n$). The 31,26 code fits this format with m=5 and and t=1. The length of each message is $n=2^5-1=31$ with 5*1 redundant bits, leaving 26 bits available for data transmission. With an appropriate polynominal BCH codes can detect all errors consisting of 2t error bits and all burst errors of mt or fewer bits. The 31,26 BCH code will therefore detect any erroneous messages with 1 or 2 errors or bursts of errors of less than 5 bits. The 31,26 format (shown in Figure 23) requires the reception of a start bit followed by 31 information bits, clearly beyond the capability of the USART but perhaps within reach of a program controlled approach using the MCS-48 itself.

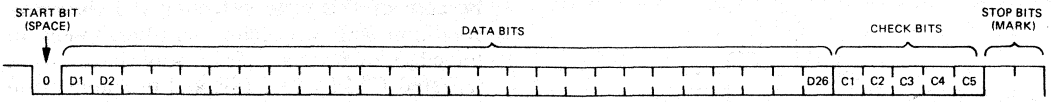


Figure 23. 31,26 BCH Code

A concept which reduces sensitivity to frequency deviations and thus allows the reception of longer codes is shown pictorially in Figure 24. The first line of this timing chart shows an alternative ones and zeros pattern on the RxD with a period of 5 milliseconds. The second line shows that by sampling at a period of exactly 5 milliseconds the data can be properly interpreted. The third and fourth lines show the effects of sampling with a period of six and four milliseconds respectively. In either case, an error occurs at the third sample where both periods result in sampling on an edge of the RxD signal. The third line of Figure 24 shows a hybrid sampling scheme which, based on some additional information, switches sampling periods between the two values. As can be seen in Figure 24, the data is sampled with a 4 millisecond period until the sampling begins to fall behind the data; at this point the sampling period is increased to six milliseconds and the sampling first catches up and then passes the center point of the data. As soon as this happens, the sampling period reverts to the 4 millisecond period and the cycle repeats. It can be seen that this scheme sets up a pattern which repeats indefinitely and the data can be successfully sampled. Note that the sampling pattern established is alternating periods of four and six milliseconds. The average period of this pattern, as might be expected, is 5msec. Line 5 of Figure 24 shows the effect of a change in transmission speed to a period of 5.5 msec with no change in the sampling time. The sampling is again successful but the new sampling pattern is 4-6-6-6; 4-6-6-6, etc. Note that the average sample is again equal to the period of the received data (5.5). While this scheme does seem to work, the question of what additional information is needed remains.

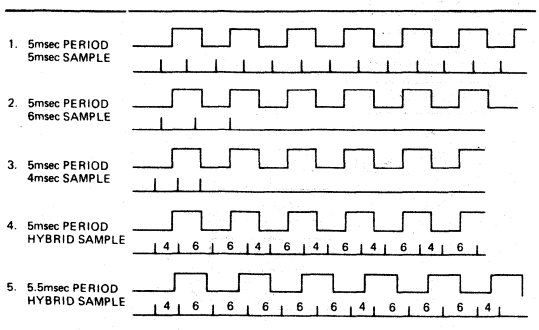


Figure 24. Various Sampling Alternatives

The MSC-48 must somehow decide when it is drifting out of synchronization and take corrective action. By referring back to Figure 24 it can be seen that if the MCS-48 could determine where the edges of RxD occurred with respect to its sampling times then the additional information would be available. As can be seen in the figure the choice of sampling period can be based on the following rule:

> *If an edge on the RxD line occurs during the first half of the current sampling period, then use the short period for the next sample. If an edge occurs during the second half of the period, then use the long sampling period for the next sample.*

If the data on the RxD line does not change, of course, the MCS-48 will drift out of synchronization just as the original algorithum did. As long as edges occur on TxD, however, synchronization can be maintained. To maximize the allowable time between edges, the following addition could be made to the above rule:

> *If no edge occurs on the RxD line during a sample, then change sampling period from short to long or vice versa.*

Note that this addition to the rule will result in using an average of the two sampling periods when no edge occurs for several bit times.

The edges of RxD can be easily detected by the use of the same structure (the Exclusive – NOR gate) which was added to the MCS-48 in Figure 20. This gate, which is used to detect the edge on RxD which begins the START bit, can naturally be used to detect any edge. Since the timer is being used to time the bit period, however, the event count input (T1) is not useful during the receive itself. By connecting the output of this gate, however, to the $\overline{\text{INT}}$ input to the MCS-48 (see Figure 25) it is possible to detect edges on RxD with the event counter when the program is trying to detect the START bit and by the external interrupt when the program is using the timer to control the sampling times.

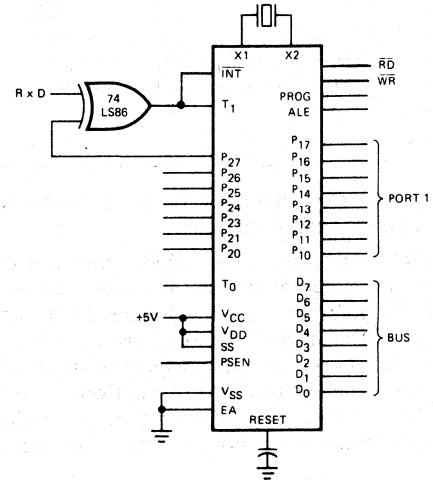
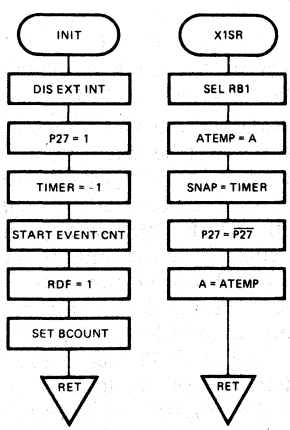


Figure 25. Modified Edge Detection

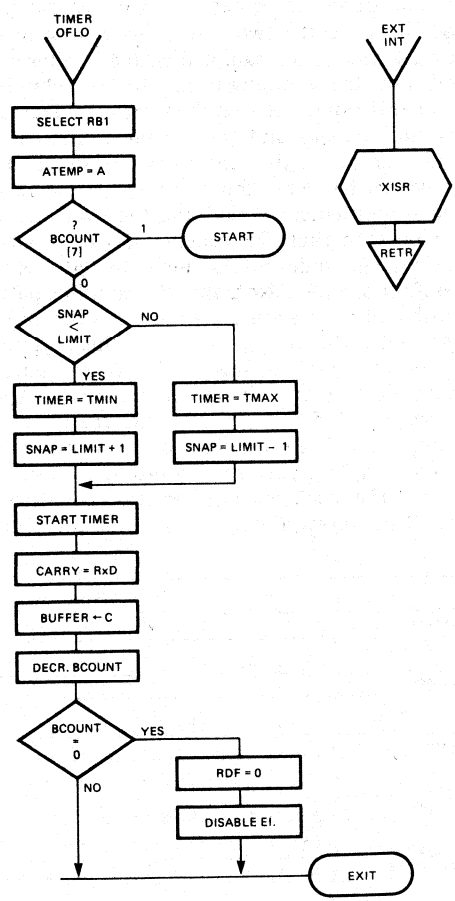
A modification to the program of Figure 21 which implements this new sampling algorithm is shown in Figure 26. The first deviation from the original program is the addition of a routine (XISR, Figure 26a which is called when an external interrupt occurs (i.e. when an edge occurs on RxD). This routine saves the status of the running program and then stores the current value of the timer register in a location called SNAP (R5 of RB1). After doing these operations the program complements bit 7 of port 2. Manipulating P27 in this manner will cause the Exclusive NOR gate to turn off the external interrupt and will set it up to generate another interrupt when the RxD line changes again (has another edge).



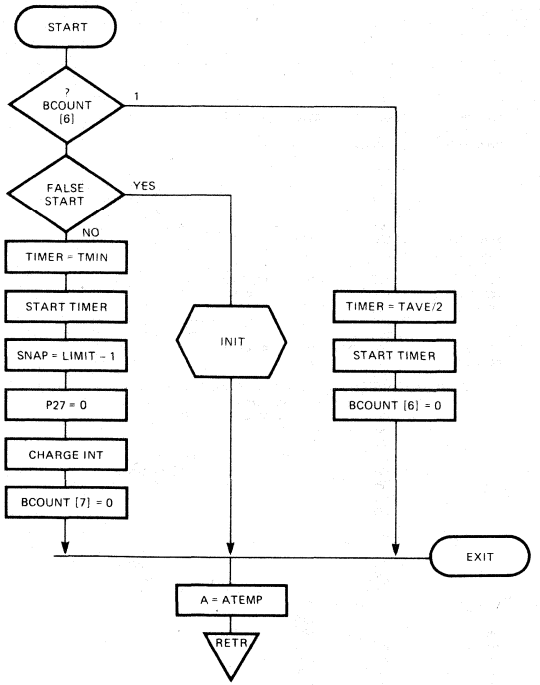
Hybrid Sampling Flowchart

Because of this edge detection it is important to condition RxD with hardware filters to ensure that the edges of RxD are clean. Any ringing will cause repeated CALLs to XISR and probable erroneous operation. The changes to the START process (Figure 26c) are two-fold; first the TIMER is set to one half the average of the two sample periods when the START bit is first detected (BCOUNT [6] = 1), and second the processing of the edge information is initialized by presetting SNAP and clearing P27.

SNAP is preset so that when the reception of data actually begins (Figure 26b BCOUNT [7] = 0), the decision block which tests SNAP against LIMIT will be initialized. This block actually compares the value in SNAP with a LIMIT value which is used to determine if the sampling point is ahead or behind the actual midpoint of the serial data. If the sampling is ahead then the timer is set for TMIN; if the sampling is behind then the timer is set for



Hybrid Sampling Flowchart



Hybrid Sampling Flowchart

TMAX. By presetting SNAP in the manner shown in the flowcharts the second rule of the algorithm, (*if no edge appears on the RxD line during a sample, then change the sampling periods short to long or vice versa*) is automatically met. If an edge occurs then XISR will modify SNAP, if XISR is not invoked between two samples then the choice of timer periods will alternate. The only other significant change to the algorithm is that the INIT routine must now lock out all interrupts, not just the timer overflow interrupt, while it is operating. A program which uses this algorithm to receive a 32 bit message is shown in Figure 27.

```
LOC  OBJ        SEQ       SOURCE STATEMENT

                    0
                    1 ; ************************************
                    2 ;
                    3 ;       SERIAL INPUT USING MCS-48
                    4 ;         THIS CODE ASSUMES HARDWARE
                    5 ;         SHOWN IN FIG 25. PROGRAM
                    6 ;         IS SIMILAR TO PREVIOUS
                    7 ;         ONE. A MORE SOPHISTICATED
                    8 ;         SAMPLING ALGORITHM IS USED
                    9 ;
                   10 ; NOTE:  A PL/M LIKE LANGUAGE WAS USED
                   11 ;        TO COMMENT THIS LISTING AND
                   12 ;        SEVERAL OTHERS IN THIS NOTE. NO
                   13 ;        COMPILER EXISTS FOR THE MCS-48.
                   14 ;        THE COMMENTS WERE 'HAND
                   15 ;        COMPILED' INTO ASSEMBLY CODE
                   16 ;;
                   17 ; ************************************
                   18
                   19 ; -------
                   20 ; EQUATES
                   21 ; -------
                   22
0007               23 ATEMP   EQU     R7      ; STORAGE FOR A DURING INTERUPT
0006               24 BCOUNT  EQU     R6      ; CONTAINS NUMBER OF BITS IN MSG
0005               25 SNAP    EQU     R5      ; TAKES TIMER SNAP SHOT ON RXD EDGE
0002               26 COUNT   EQU     R2      ; UTILITY COUNTER
0000               27 RXD     EQU     R0      ; POINTER
0020               28 BITNO   EQU     32      ; NUMBER OF BITS
0014               29 LIMIT   EQU     20      ; TEST VALUE FOR MIN/MAX SAMPLING
FFD5               30 TMAX    EQU     -43     ; MAX SAMPLE PERIOD
FFD9               31 TMIN    EQU     -39     ; MINIMUM SAMPLE PERIOD
FFEC               32 HALF    EQU     -20     ; HALF NOMINAL PERIOD
0020               33 SERBUF  EQU     20H     ; START OF SERIAL BUFFER
0024               34 RDF     EQU     24H     ; RECEIVE DONE FLAG
                   35
                   36 ; ---------------------------------
                   37 ; CONTROL PASSED HERE ON EXT. INT.
                   38 ; ---------------------------------
                   39
0003               40         ORG     03H
                   41                         ; CALL SERVICE ROUTINE
0003 1466          42 EIVEC:  CALL    XISR
0005 93            43         RETR
                   44
                   45 ; -----------------------------------------
                   46 ; CONTROL PASSED HERE WHEN TIMER OFLO OCCURS
                   47 ; -----------------------------------------
                   48
                   49                         ; /*ENTER INTERUPT MODE*/
0006 D5            50 TMVEC:  SEL RB1
0007 AF            51         MOV     ATEMP,A
                   52                         ; IF BCOUNT[7]=0 THEN
0008 FE            53         MOV     A,BCOUNT
0009 F236          54         JB7     START
                   55                         ; DO;
                   56                         ;    IF SNAP<LIMIT THEN
000B FD            57         MOV     A,SNAP
000C 0314          58         ADD     A,#LIMIT
000E F217          59         JB7     SLLA
                   60                         ;       DO;
                   61                         ;          TIMER=TMIN;
                   62                         ;          SNAP=LIMIT+1;
                   63                         ;       END;
0010 23D9          64         MOV     A,#TMIN
0012 62            65         MOV     T,A
0013 BD13          66         MOV     SNAP,#LIMIT-1
0015 041C          67         JMP     SLLB
                   68                         ;    ELSE
                   69                         ;    DO;
                   70                         ;       TIMER=TMAX;
                   71                         ;       SNAP=LIMIT-1;
                   72                         ;    END;
0017 23D5          73 SLLA:   MOV     A,#TMAX
```

Figure 27. Hybrid Sampling Program

```
LOC  OBJ       SEQ       SOURCE STATEMENT

0019 62          74        MOV     T,A
001A BD13        75        MOV     SNAP,#LIMIT-1
                 76                        ;   START TIMER;
001C 55          77 SLLB:  STRT    T
                 78                        ;   /*CARRY=RXD*/
                 79                        ;   CARRY=P27 XOR TEST1;
001D 0A          80        IN      A,P2
001E F7          81        RLC     A
001F 4622        82        JNT1    TISRD
0021 A7          83        CPL     C
                 84                        ;   /*SHIFT CARRY INTO BUFFER*/
                 85                        ;   RX0=SERBUF;
                 86                        ;   COUNT=4;
                 87                        ;   DO WHILE COUNT<>0;
                 88                        ;      RSHFT MEM(RX0);
                 89                        ;      RX0=RX0+1;
                 90                        ;      COUNT=COUNT-1;
                 91                        ;   END;
0022 B820        92 TISRD: MOV     RX0,#SERBUF
0024 BA04        93        MOV     COUNT,#4
0026 28          94 SLOOP  XCH     A,RX0
0027 67          95        RRC     A
0028 28          96        XCH     A,@RX0
0029 18          97        INC     RX0
002A EA26        98        DJNZ    COUNT,SLOOP
                 99                        ;   BCOUNT=BCOUNT-1;
                100                        ;   IF BCOUNT=0 THEN
002C EE54       101        DJNZ    BCOUNT,SEXIT
                102                        ;   DO;
                103                        ;      RDF=0;
                104                        ;      DISABLE EX INT;
                105                        ;   END;
002E B824       106        MOV     RX0,#RDF
0030 27         107        CLR     A
0031 A0         108        MOV     @RX0,A
0032 35         109        DIS     TCNTI
0033 15         110        DIS     I
                111                        ; END;
0034 0454       112        JMP     SEXIT
                113                        ; ELSE
                114                        ; DO;
                115                        ;   IF BCOUNT[6]=0 THEN
0036 FE         116 START: MOV     A,BCOUNT
0037 D24C       117        JB6     SLLC
                118                        ;   DO;
                119                        ;      IF TEST1=0 THEN
0039 564A       120        JT1     SLLD
                121                        ;      DO;
                122                        ;         TIMER=TMIN;
                123                        ;         START TIMER;
                124                        ;         SNAP=LIMIT+1;
                125                        ;         P27=0;
                126                        ;         EN I
                127                        ;         BCOUNT[7]=0;
                128                        ;      END;
003B 23D9       129        MOV     A,#TMIN
003D 62         130        MOV     T,A
003E 55         131        STRT    T
003F BD15       132        MOV     SNAP,#LIMIT+1
0041 9A7F       133        ANL     P2,#7FH
0043 05         134        EN      I
0044 FE         135        MOV     A,BCOUNT
0045 537F       136        ANL     A,#7FH
0047 AE         137        MOV     BCOUNT,A
0048 0454       138        JMP     SEXIT
                139                        ;      ELSE
                140                        ;      DO;
                141                        ;         CALL INIT;
                142                        ;      END;

LOC  OBJ       SEQ       SOURCE STATEMENT

004A 1456       143 SLLD:  CALL INIT
                144                        ;   ELSE
                145                        ;   DO;
                146                        ;      TIMER=(TMIN+TMAX)/2;
                147                        ;      START TIMER;
                148                        ;      BCOUNT[6]=0;
                149                        ;   END;
004C 23EC       150 SLLC:  MOV     A,#HALF
004E 62         151        MOV     T,A
004F 55         152        STRT    T
0050 FE         153        MOV     A,BCOUNT
0051 53BF       154        ANL     A,#0BFH
0053 AE         155        MOV     BCOUNT,A
                156                        ; END;
                157                        ; /*EXIT INTERUPT MODE*/
0054 FF         158 SEXIT: MOV     A,ATEMP
0055 93         159        RETR
                160
                161 ; ----------------------------
                162 ; INTIALIZE ROUTINE-
                163 ;      STARTS RECEIVE PROCESS
                164 ; ----------------------------
                165
                166                        ; INIT:
                167                        ;   PROCEDURE;
                168                        ;      DO;
                169                        ;         DISABLE INTERUPTS;
                170                        ;         P27=1;
                171                        ;         TIMER=-1;
                172                        ;         START EVENT COUNT;
                173                        ;         RDF=1;
                174                        ;         BCOUNT=0C0H OR BITNO
                175                        ;      END;
                176                        ;   END INIT;
0056 15         177 INIT:  DIS     I
0057 35         178        DIS     TCNTI
0058 8A86       179        ORL     P2,#86H
005A 23FF       180        MOV     A,#-1
005C 62         181        MOV     T,A
005D 45         182        STRT    CNT
005E B824       183        MOV     RX0,#RDF
0060 F9         184        MOV     A,@1
0061 A0         185        MOV     @RX0,A
0062 25         186        EN      TCNTI
0063 BEC0       187        MOV     BCOUNT,#0C0H OR BITNO
0065 83         188        RET
                189
                190
                191 ; -------------------------
                192 ; INTERUPT SERVICE ROUTINE
                193 ; -------------------------
                194                        ; XISR:
                195                        ;   PROCEDURE;
                196                        ;   DO;
                197                        ;      /*ENTER INTERUPT MODE*/
                198                        ;      SNAP=TIMER;
                199                        ;      P27=NOT P27;
                200                        ;   END XISR;
0066 D5         201 XISR:  SEL     RB1
0067 AF         202        MOV     ATEMP,A
0068 42         203        MOV     A,T
0069 AD         204        MOV     SNAP,A
006A 0A         205        IN      A,P2
006B D380       206        XRL     A,#80H
006D 3A         207        OUTL    P2,A
006E FF         208        MOV     A,ATEMP
006F 83         209        RET
                210                        ; END OF PROGRAM
                211        END
```

Figure 27. Hybrid Sampling Program

TRANSMITTING SERIAL CODE

Serial transmission is conceptually far simpler than serial reception since no synchronization is required. All that is required is to use the timer to generate interrupts at the bit rate and present the character to be transmitted serially at an I/O pin. A program which does this is shown in Figure 28. The transmission of serial data becomes much more complicated if it must occur simultaneously with reception.

If both reception and transmission are to occur simultaneously then obviously contention will exist for the use of the timer. It is possible to allow the simultaneous reception and transmission of serial data using the timer as a general clock which controls software maintained timers. The attainable baud rates using such techniques are, however, limited and the use of a 8251 USART is probably indicated in all but the most cost sensitive applications. An exception to this rule occurs when the system, although full duplex in nature, actually transmits the same data as it receives. An example of this is a microprocessor driving a terminal such as a Teletype. Although the circuit to the terminal is full duplex, the data that is transmitted is generally the same as that received. A minor modification to the program shown in Figure 26 would implement this mode of operation. The modification would be to the XISR routine and it would add the code necessary to place the TxD I/O pin in the same state as the RxD line. Since any change in RxD results in a call to XISR, this modification would cause the retransmission of any received data. Whenever it becomes necessary to transmit data which is not being received, the program of Figure 28 could be used in a half duplex manner.

```
LOC  OBJ       SEQ       SOURCE STATEMENT

                  0
                  1 ; ---------- -------------------------
                  2 ; SERIAL TRANSMIT ON THE MCS48
                  3 ;   TO USE PUT A CHAR IN BUFF AND
                  4 ;   SET CHARAV TO 0FFH. WHEN THE
                  5 ;   TRANSMITTER IS READY FOR ANOTHER
                  6 ;   CHAR IT WILL CLEAR CHARAV. THE
                  7 ;   TRANSMISSION IS DOUBLE BUFFERED.
                  8 ;-----------------------------------
                  9
                 10 ; -------
                 11 ; EQUATES
                 12 ; -------
                 13
0007             14 ATEMP   EQU     R7      ; STORAGE FOR A DURING INT.
0006             15 PTOS    EQU     R6      ; PARALLEL TO SERIAL CONVERTER
0005             16 BUFF    EQU     R5      ; CHARACTER BUFFER
0004             17 CHARAV  EQU     R4      ; CHARACTER AVAILABLE FLAG
0003             18 COUNT   EQU     R3      ; BIT COUNTER
00EF             19 CBIT    EQU     0EFH    ; MASK TO CLEAR TXD IN P24
0010             20 SBIT    EQU     010H    ; MASK TO SET TXD IN P24
FFD7             21 P       EQU     -41     ; PERIOD OF TXD
                 22
                 23 ; ---------------------------------
                 24 ; CONTROL PASSED HERE ON TIMER OVERFLOW
                 25 ; ---------------------------------
0007             26         ORG     07H
                 27                         ; ENTER INTERUPT MODE
0007 D5          28 TOFLO:  SEL     RB1
0008 AF          29         MOV     ATEMP,A
                 30                         ; SET TIMER FOR P
0009 23D7        31         MOV     A,#P
000B 62          32         MOV     T,A
000C 55          33         STRT    T
                 34                         ; GET BIT INTO CARRY
000D 141D        35         CALL    BIT
                 36                         ; SET TXD TO CARRY
```

```
LOC  OBJ       SEQ       SOURCE STATEMENT

000F 0A          37         IN      A,P2
0010 D300        38         XRL     A,#00H
0012 3A          39         OUTL    P2,A
0013 F619        40         JC      BITON
0015 9AEF        41         ANL     P2,#CBIT
0017 041B        42         JMP     EXIT
0019 8A10        43 BITON:  ORL     P2,#SBIT
001B FF          44 EXIT:   MOV     A,ATEMP
001C 93          45         RETR
                 46
                 47 ; --------------------------------
                 48 ; BIT ROUTINE
                 49 ; -PICKS THE NEXT BIT TO TRANSMIT
                 50 ; --------------------------------
                 51
001D FB          52 BIT:    MOV     A,COUNT
001E C627        53         JZ      IDLE
0020 FE          54         MOV     A,PTOS
0021 67          55         RRC     A
0022 4300        56         ORL     A,#00H
0024 AE          57         MOV     PTOS,A
0025 CB          58         DEC     COUNT
0026 83          59         RET
                 60
0027 97          61 IDLE:   CLR     C
0028 FC          62         MOV     A,CHARAV
0029 962D        63         JNZ     GOTONE
002B A7          64         CPL     C
002C 83          65         RET
                 66
002D FD          67 GOTONE: MOV     A,BUFF
002E AE          68         MOV     PTOS,A
002F BB0A        69         MOV     COUNT,#10
0031 BC00        70         MOV     CHARAV,#0
0033 83          71         RET
                 72                         ; END OF PROGRAM
                 73         END
```

Figure 28. Serial Transmission

GENERATING PARITY

Many communications schemes require the generation and checking of parity. If a USART is used it can be programmed to automatically generate and check parity. If the communications is handled by software within the MCS-48™ then the program must perform parity calculations. Calculating parity is easy if one remembers what parity really means. A character has even parity if the number of one bits in it is even. A character has odd parity if it has an odd number of ones. The program segment shown in Figure 29 can be caused to calculate parity. It starts by setting a loop count to eight and

```
LOC  OBJ       SEQ       SOURCE STATEMENT

                  0
                  1
                  2 ; *************
                  3 ;
                  4 ; PARITY
                  5 ;   THIS PROGRAM GENERATES PARITY
                  6 ;   ON THE ACCUMULATOR
                  7 ;    CARRY WILL BE SET IF A HAS ODD PARITY
                  8 ;
                  9 ; *************
                 10
                 11
                 12 ; -------
                 13 ; EQUATES
                 14 ; -------
                 15
0002             16 COUNT   EQU     R2
                 17
0100             18 PAR:    ORG     100H
0100 BA08        19         MOV     COUNT,#8      ; SET LOOP COUNT
0102 97          20         CLR     C             ; INITIALIZE CARRY
                 21                               ; FOR EACH ZERO BIT IN A
                 22                               ; COMPLEMENT THE CARRY FLAG
0103 77          23 LOOP:   RR      A
0104 1207        24         JB0     OVER
0106 A7          25         CPL     C
                 27                               ; END OF PROGRAM
                 28         END
```

Figure 29. Parity Generation

clearing the CARRY flag. After this initialization a loop is executed eight times. During each execution the accumulator is rotated and the least significant bit is tested. If the bit is a zero the CARRY flag is complemented, if the bit is a one no further action is taken. Since an even number of zeros implies an even number of ones for an eight bit character, after all eight loops have been accomplished the CARRY bit will be set if an odd number of ones were encountered; it will be reset if the number were even. Since the RR instruction does not involve CARRY the net result of executing this program loop is to set CARRY if parity is odd without effecting the character in the accumulator.

CONCLUSION

This Application Note has presented a very small sampling of the application techniques possible with the MCS-48™ family. The application of this new single chip computer system to tasks which have not yet yielded to the power of the microprocessor will present a fascinating challenge to the system designer.

Keyboard/Display Scanning with Intel's MCS-48® Microcomputers

Contents

INTRODUCTION 5-64

HARDWARE SCHEMATIC 5-67

SOFTWARE LISTING 5-68

CONFIGURATION EQUATES 5-72

UTILITY SUBROUTINES 5-79

ASSEMBLY CROSS-REFERENCE 5-86

INTRODUCTION

This application notes presents a software package for interfacing members of Intel's MCS-48™ family of single-chip microcomputers with keyboards and displays using a minimum of external components. Because of the similarity of the architectures of the various members of the family (the 8035, 8048, 8748, 8039, 8049, 8021, and 8022 microcomputers; also the 8041 and 8741 universal peripheral interfaces in the UPI-41® family), the code included here could run with minor modifications on any member of the family.

Since keyboard and display logic can be just one of several functions handled by a microprocessor, the added cost of including these functions in a system is minimal. In fact, considering the extremely low cost of standard X-Y matrix keyboards and integrated displays, their use is often more cost effective than even a handful of discrete switches and indicators. Thus, the additional flexibility of keyboard input and display output can be added to inexpensive consumer products (such as games, clocks, thermostats, tape recorders, etc.), while producing a net savings in system cost.

Since each potential application will have its own unique combination of keys and display characters, the program is written so that very little modification is needed to interface it with a wide variety of hadware configurations. In general, the only changes required are within the set of initial EQUates at the beginning of the program.

Along with the basic software for driving a multiplexed display and/or scanning and debouncing an X-Y matrix of key switches, a collection of utility subroutines is also included for implementing the most commonly used keyboard and display utility functions, such as copying simple messages onto the display or determining the encoded value of each key in the key matrix. As a result of the versatile architecture and applications-oriented instruction set of the MCS-48 family, the entire package fits into about 250 bytes of internal program ROM or EPROM, leaving the rest of the ROM space for the program to cook the perfect piece of toast, or whatever. By tailoring the software to match a known hardware configuration, or by selecting only those functions needed for a given application, the program size could be even further reduced.

Since what is being presented in this application note is a software package, rather than the usual hardware/software system design, the format of this note is somewhat different from most — it consists primarily of a long program listing reproduced in the following pages. For the most part, the listing is self-explanatory, with comments introducing each subroutine and major code segment. Some parts of this introduction are reproduced in the program listing itself, explaining the configuration of the prototype system. However, an additional bit of explanation would make the listing easier to understand, especially for those readers unfamiliar with the concept of multiplexed displays and keyboards.

In traditional digital system design, various hardware registers or counters were used to hold binary or BCD values which had to be conveyed to the user. The standard way of presenting this information was by connecting each register to a seven-segment encoder (such as the 7447) driving a single display character, as represented by Figure 1. Thus, two ICs, seven current limiting resistors, and about 45 solder joints were required for each digit of output. Consider how traditional techniques might be (mis-)applied in designing a microprocessor system: the designer could add a latch, encoder, and resistors for each digit of the display. Still another latch and decoder could be used to turn on one of the decimal points (if used). The characters displayed could only be a sequence of decimal digits. In the same vein, a large matrix of key switches could be read by installing an MSI TTL priority encoder read by an additional input port. Not only would all this use a lot of extra I/O ports and increase the system price and part count drastically, but the flexibility and reliability of the system would be greatly reduced.

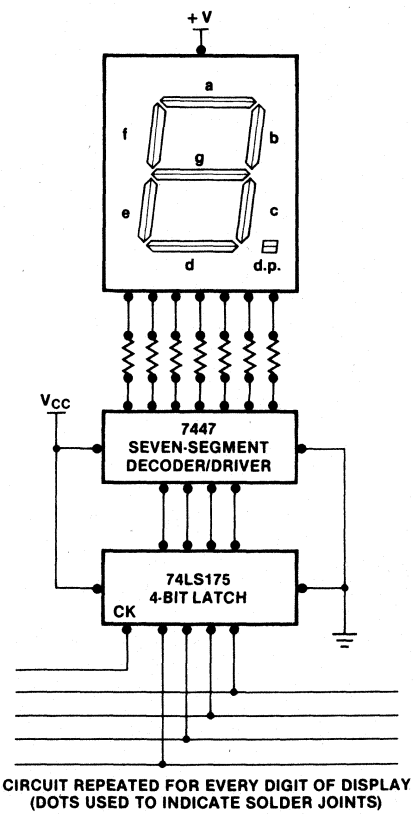


Figure 1. Wrong Way to Design Multiple Digit Displays for Microcomputer Systems

Instead, a scheme of time-multiplexing the display can be used to decrease costs, part count, and interconnections, while allowing a wider range of character types to be used on the display. The techniques used here are fairly typical of today's integrated subsystems designed especially for controlling keyboards and displays (such as in calculators or the Intel® 4269, 8278, and 8279 Keyboard/Display Controller Devices).

In a multiplexed display, all the segments of all the characters are interconnected in a regular two-dimensional array. One terminal of each segment is in common with the other segments of the same character; the other terminal is connected with the same segments of the other characters. This is represented schematically in Figure 2. A digit driver or segment driver is needed for each of these common lines.

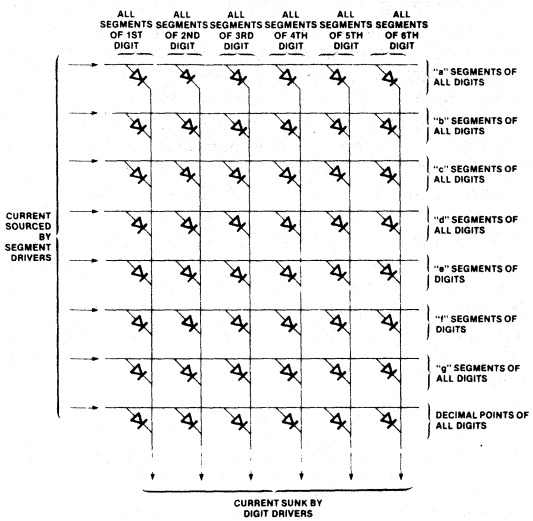


Figure 2. Schematic Representation of 6-Digit, 7-Segment Common-Cathod LED Multiplexed Display

The various characters of the display are not all on at once; rather, only one character at a time is energized. As each character is enabled, some combination of segment drivers is turned on, with the result that a digit appears on the enabled character. (For example, in Figure 3, if segment drivers 'a', 'b', and 'c' were on when character position #6 was enabled, the digit '7' would appear in the left-most place.) Each character is enabled in this way, in sequence, at a rate fast enough to ensure that the display characters seem to be on constantly, with no appearance of flashing or flickering.

In the system presented here, these rapid modifications to the display are all made under the control of the MCS-48™ microcomputer. At periodic intervals the computer quickly turns off all display segments, disables the character now being displayed and enables the next, looks up the pattern of segments for the next character to be displayed, and turns on the appropriate segments. With the next character now turned on, the processor may now resume whatever it had been doing before. The whole display updating task consumes only a small fraction of the processor's time.

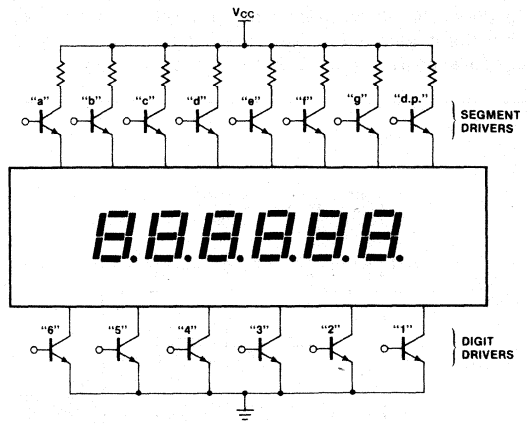


Figure 3. Segment and Digit Drivers used with 6-Position, 7-Segment LED Display

Moreover, since the computer rather than a standard decoder circuit is used to turn the segments off and on, patterns for characters other than decimal digits may be included in the display. Hexadecimal characters, special symbols, and many letters of the alphabet are possible. With sufficient imagination this feature can be exploited for some applications, as suggested by the examples in Figure 4.

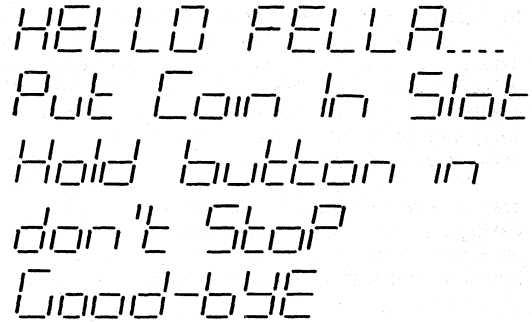


Figure 4. Examples of Typical Messages Possible with Simple 7-Segment Displays

As each character of the display is turned on, the same signal may be used to enable one row of the key matrix. Any keys in that row which are being pressed at the time will then pass the signal on to one of several "return lines", one corresponding to each column of the matrix. (See Figure 5.) By reading the state of these control lines, and knowing which row is enabled, it is possible to compute which (if any) of the keys are down. Note that the keys need not be physically arranged in a rectangular array; Figure 5 is merely a schematic.

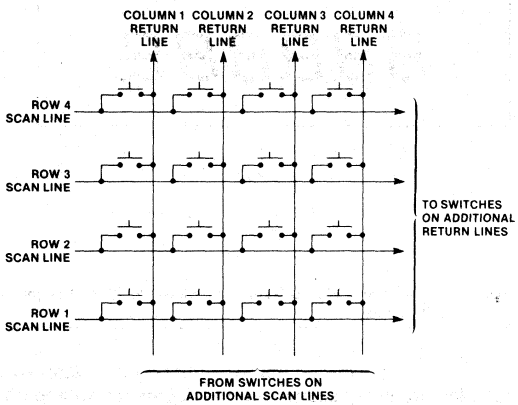


Figure 5. Schematic of X-Y Matrix Multiplexed Keyboard

Since each character is on for only a small fraction of the total display cycle, its segments must be driven with a proportionately higher current so that their brightness averages out over time. This requires character and segment drivers which can handle higher than normal levels of current. Various types of drivers can be used, ranging from specially designed circuits to integrated or discrete transistor arrays. The selection depends on several factors, including the type of display being used (LED, vacuum flourescent, neon, etc.), its size, the number of characters, and the polarity of the individual segments. Some drivers have active high inputs, some active low. Some invert their input logic levels, some do not. Some require insignificant input currents, some present a considerable load. Some systems use external logic to enable one of N characters or to produce the appropriate segment pattern for a given digit, some systems implement these functions through software.

Because of these and the other variables which make each application unique, provisions are made in the first page of symbol EQUates to allow the user to specify such things as the number of characters in the display or the polarity of the drivers used, and the program will be assembled accordingly. The display is refreshed on each timer interrupt, which occurs every 32× (TICK) machine cycles. (One machine cycle occurs every 30 crystal oscillations for the 8021 and 8022, or every 15 oscillations for all other members of the family.) A more detailed explanation of these variables is included in the listing.

Port assignment is also at the discretion of the user — all port references in the listing are "logical" rather than physical port names. The port used to specify which character is enabled is referred to as "PDIGIT". The output segment pattern is written to "PSGMNT" and the keyboard return lines are read by "PINPUT". These logical port names may be assigned to whichever ports the user pleases.

By way of example, the breadboard used to develop and debug this software used a matrix of 16 single-pole pushbuttons and an 8-character common-cathode LED display with right-hand decimal point. No decoders external to the 8748 microcomputer were used; all logic was handled through software. PDIGIT was the 8-bit bus, PSGMNT was port 1, and PINPUT was port 2. The drivers used were 75491 and 75492 logically non-inverting buffers: high level inputs were used to turn a segment or character on. Pull-up resistors were used on the 8748 output lines to source the current levels needed by the buffers. The 8748 was socketed on the breadboard, and was driven with an inexpensive 3.59 MHz television crystal. The short test program included in this listing was used to echo key depressions as they were detected, and to invoke four demonstration subroutines. A summary of the subroutines included in this listing with a short explanation of the function of each is included in Figure 6; Figure 7 shows how the various utilities interact.

KBDIN	**Keyboard Input.** Waits until one keystroke input has been received from the keyboard; determines the meaning or legend of that key, and returns with the encoded value in the accumulator.
CLEAR	Blank out the display.
ENCACC	Encode accumulator with bit pattern corresponding to the segment pattern needed by the display to represent that symbol or character. Uses the value of the accumulator when called to access a table containing the patterns for all legal input values.
WDISP	**Write into Display.** Writes the bit pattern in the accumulator into the next character position of the display. Maintains a character position counter so that repeated calls will automatically write characters into sequential positions.
RENTRY	**Right-hand Entry.** Stores the accumulator segment pattern in the display in the right-most character position. Shifts all other characters to the left one place.
PRINT	Print a string of arbitrary characters onto the display. Useful for prompting messages, warnings, etc. Uses a table of segment patterns in ROM, so that messages will not be restricted to numbers, letters, etc.
FILL	Fill the display with the character pattern in the accumulator. Useful for writing dashes, segment test patterns, etc., into all character positions.
ECHO	Wait for a key to be pressed by the operator and write that key onto the display. Used for providing feedback to the operator when entering numeric data, etc.
RDPADD	Adds or deletes a decimal point to the character at the right-hand side of the display, for entering floating point numbers.
HOLD	Called when a key is known to be down. Does not return until all keys have been released. Used for organ-type keyboards, or when some action should not be initiated until the key invoking that action has been released.
DELAY	Provides a crude real-time delay corresponding to the value of the accumulator when called. Can be used to cause display characters to blink, to momentarily flash information, to enable a buzzer, etc. Could also be used by the program when delays are needed, such as to slow down the computer reaction rate while playing a game against the human operator.

Figure 6. Utility Subroutine Definitions

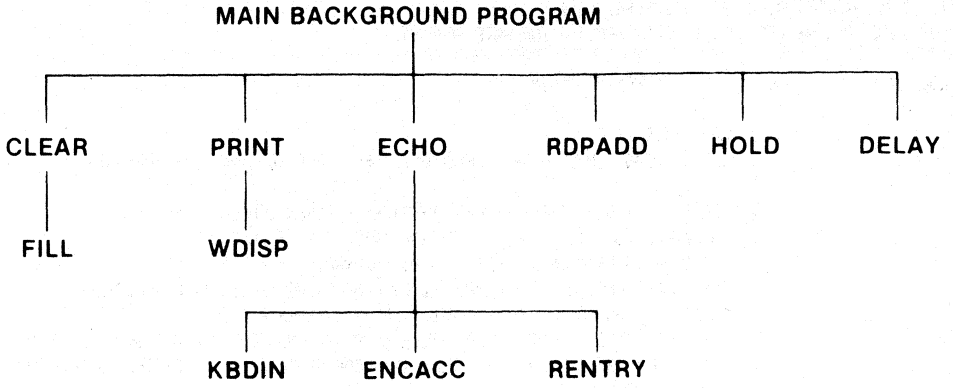


Figure 7. Subroutine Interrelationships

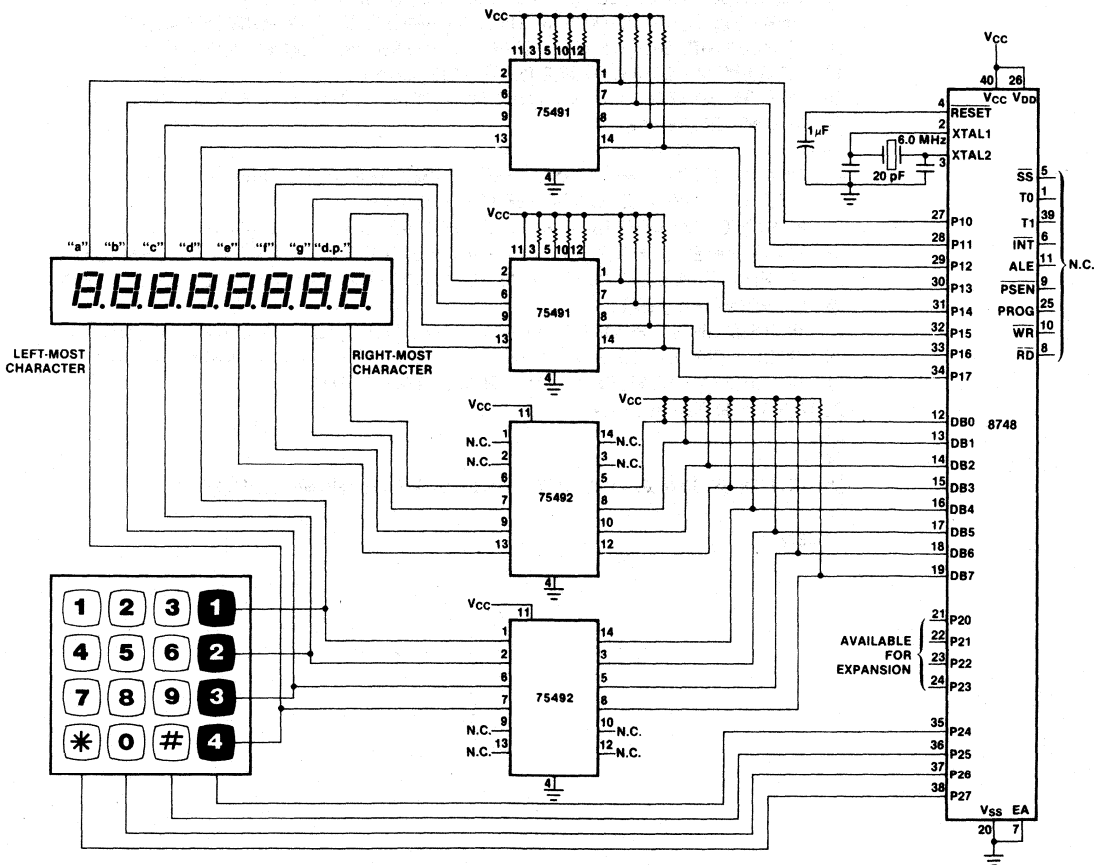


Figure 8 Prototype System Schematic

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                    1 $MACROFILE XREF
                    2 $TITLE('AP40: INTEL MCS-48 KEYBOARD/DISPLAY APPLICATION NOTE APPENDIX')
                    3 ;
                    4 ;THE FOLLOWING SOFTWARE PACKAGE PROVIDES A SEVEN SEGMENT DISPLAY
                    5 ;INTERFACE FOR MICROCOMPUTERS IN THE INTEL MCS-48 FAMILY.
                    6 ;THE CODE IS WRITTEN SO THAT VARIOUS HARDWARE
                    7 ;CONFIGURATIONS CAN BE ACCOMODATED BY REDEFINING THE INITIAL VARIABLES.
                    8 ;IN MOST SITUATIONS, THE KEYBOARD/DISPLAY INTERFACE WILL BE REQUIRED TO
                    9 ;IMPLEMENT MORE SOPHISTICATED SINGLE-CHIP SYSTEMS (CALCULATORS, SCALES, CLOCKS,
                   10 ;ETC.), WITH SECTIONS OF THE FOLLOWING CODE SELECTED AND MODIFIED AS NECESSARY
                   11 ;FOR EACH APPLICATION.
                   12 ;
                   13 ;A SINGLE SUBROUTINE (CALLED REFRSH) IS USED TO IMPLEMENT BOTH THE DISPLAY
                   14 ;MULTIPLEXING AND KEYBOARD SCANNING, USING THE SAME SIGNAL BOTH TO ENABLE
                   15 ;ONE CHARACTER OF THE DISPLAY AND TO STROBE ONE ROW OF THE X-Y KEY MATRIX.
                   16 ;THE SUBROUTINE MUST BE CALLED SUFFICIENTLY OFTEN TO ENSURE THE DISPLAY
                   17 ;CHARACTERS DO NOT FLICKER- AT LEAST 50 COMPLETE DISPLAY SCANS PER SECOND.
                   18 ;TO ACCOMODATE SWITCHES OF ARBITRARY CHEAPNESS, THE DEBOUNCE TIME CAN BE
                   19 ;SET TO BE ANY DESIRED NUMBER OF COMPLETE SCANS.
                   20 ;THUS THE DEBOUNCE TIME IS A FUNCTION OF BOTH THE SCAN RATE AND THE VALUE
                   21 ;OF CONSTANT 'DEBNCE'.
                   22 ;
                   23 ;IN THIS LISTING, THE INTERNAL TIMER IS USED TO GENERATE INTERRUPTS THAT
                   24 ;SERVE AS A TIME BASE FOR THE REFRESH SUBROUTINE.
                   25 ;ALTERNATE TIME BASES MIGHT BE AN EXTERNAL OSCILLATOR (DRIVING THE INTERRUPT
                   26 ;PIN OR POLLED BY A TEST OR INPUT PIN), A SOFTWARE DELAY LOOP IN THE BACKGROUND
                   27 ;PROGRAM, OR PERIODIC CALLS TO THE SUBROUTINE FROM THROUGHOUT THE USER'S PROGRAM
                   28 ;AT APPROPRIATE PLACES.
                   29 ;IN THESE CASES, THE CODE STARTING AT LABEL TIINT (TIMER INTERRUPT) AND TIRET
                   30 ;(TIINT RETURN) COULD STILL BE USED TO SAVE AND RESTORE ACCUMULATOR CONTENTS.
                   31 ;THE INTERRUPT SERVICING ROUTINE SELECTS REGISTER BANK 1
                   32 ;FOR THE NEEDED REGISTERS.
                   33 ;
                   34 ;
                   35 ;WRITTEN BY JOHN WHARTON, INTEL SINGLE-CHIP COMPUTER APPLICATIONS
                   36 ;
                   37 $EJECT
```

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0
AP40: INTEL MCS-48 KEYBOARD/DISPLAY APPLICATION NOTE APPENDIX

 LOC  OBJ          SEQ          SOURCE STATEMENT

                     38 ;IN THIS IMPLEMENTATION OF THE DISPLAY SCAN, IT IS ASSUMED THAT THERE WILL
                     39 ;BE RELATIVELY LITTLE I/O OTHER THAN FOR THE KEYBOARD/DISPLAY.
                     40 ;IF THIS IS THE CASE, THEN THERE IS NO NEED FOR FOR ANY ADDITIONAL EXTERNAL
                     41 ;LOGIC (SUCH AS ONE-OF-EIGHT DECODERS OR SEVEN-SEGMENT ENCODERS), THOUGH
                     42 ;THERE WILL STILL BE A NEED FOR CURRENT OR VOLTAGE DRIVERS, ACCORDING TO
                     43 ;THE TYPE OF DISPLAY BEING USED.
                     44 ;
                     45 ;IN THIS LISTING, THE PROCESSOR I/O PORTS ARE LOGICALLY DIVIDED AS FOLLOWS:
                     46 ;
                     47 ;PDIGIT-EIGHT BIT PORT USED TO ENABLE, ONE AT A TIME, THE INDIVIDUAL
                     48 ;       CHARACTERS OF AN EIGHT DIGIT SEVEN-SEGMENT DISPLAY, WHILE ALSO
                     49 ;       STROBING THE ROWS OF AN X-Y MATRIX KEYBOARD.
                     50 ;       BIT7 ENABLES THE LEFTMOST CHARACTER AND THE BOTTOM ROW OF THE KBD,
                     51 ;       BIT4 ENABLES THE TOP ROW OF THE 4X4 KBD AND THE FOURTH CHARACTER,
                     52 ;       BIT0 ENABLES THE RIGHTMOST CHARACTER.
                     53 ;       (A 4X8 KEYBOARD COULD BE STROBED BY ALSO USING BIT3-BIT0
                     54 ;       AND EXTENDING OR ELIMINATING THE TABLE, "LEGNDS".)
                     55 ;       THE ENABLING OF ONE BIT (ACTIVE HIGH OR LOW) IS ACCOMODATED BY
                     56 ;       ACCESSING A LOOK-UP TABLE CALLED CHRSTB.
                     57 ;       THIS TECHNIQUE TAKES ABOUT FOUR BYTES MORE ROM THAN A TECHNIQUE
                     58 ;       OF ROTATING A 'ONE' THROUGH A FIELD OF 'ZEROES' IN THE ACC
                     59 ;       AN APPROPRIATE NUMBER OF TIMES, BUT IT ALLOWS SOME ADDITIONAL
                     60 ;       FLEXABILITY: IF THE DRIVERS BEING USED HAVE A COMBINATORIAL INPUT
                     61 ;       (AS IN THE 7545X FAMILY OF HIGH-CURRENT, HIGH-VOLTAGE DRIVERS),
                     62 ;       THE CHRSTB TABLE COULD PROVIDE ENCODED OUTPUTS.  NINE DIGITS, FOR
                     63 ;       EXAMPLE, COULD BE ENABLED WITH SIX BITS OF (BUFFERED) OUTPUT:
                     64 ;       (001001,001010,001100,010001,010010,010100,100001,100010,100100)
                     65 ;       IF I/O LINES NEED TO BE CONSERVED, OR IF MANY DIGITS
                     66 ;       MUST BE DISPLAYED, AN EXTERNAL DECODER COULD BE ADDED TO THE SYSTEM.
                     67 ;       DURING CHARACTER TRANSITIONS A 'BLANK' CHARACTER IS
                     68 ;       EXPLICITLY WRITTEN TO THE DISPLAY.  THUS,
                     69 ;       THERE WILL BE NO CHARACTER 'SHADOWING' CAUSED BY THE
                     70 ;       FACT THAT THE HARDWARE OR SOFTWARE DECODER KEEPS ONE
                     71 ;       OUTPUT, AND THUS ONE CHARACTER, ACTIVE AT ALL TIMES.
                     72 ;
                     73 ;PSGMNT-EIGHT BIT PORT TO ENABLE THE SEVEN SEGMENTS & D.P. OF A STANDARD
                     74 ;       DISPLAY.
                     75 ;       BIT7-BIT0 CORRESPOND TO THE DP AND SEGMENTS G THROUGH A, RESPECTIVELY.
                     76 ;       IT IS POSSIBLE TO ACCOMODATE
                     77 ;       DRIVERS WHICH ARE EITHER LOGICALLY INVERTING OR NON-INVERTING BY
                     78 ;       SETTING VARIABLE 'SEGPOL' (SEGMENT POLARITY).
                     79 ;       NOTE THAT BY HAVING ARBITRARY CONTROL OVER EACH SEGMENT, NON-NUMERIC
                     80 ;       CHARACTERS CAN BE REPRESENTED ON A SEVEN SEGMENT DISPLAY,
                     81 ;       AS SHOWN IN EXAMPLE SUBROUTINE 'TEST2'.
                     82 ;
                     83 $EJECT
```

```
LOC  OBJ          SEQ          SOURCE STATEMENT

                      84 ;PINPUT-FOUR HIGH-ORDER BITS USED AS INPUTS FROM THE KEYBOARD RETURN LINES.
                      85 ;       ASSUMES THAT A KEY DOWN IN THE CURRENTLY ENABLED ROW WOULD RETURN
                      86 ;       A LOW LEVEL.
                      87 ;       IN THIS CASE, BIT7 RETURNS THE LEFTMOST COLUMN, BIT4 THE RIGHTMOST.
                      88 ;       THE HIGH-ORDER BITS ARE USED SO THAT IF AN OFF-CHIP DECODER IS USED
                      89 ;       TO ENABLE UP TO 16 CHARACTERS, FOR EXAMPLE, IT COULD BE DRIVEN BY
                      90 ;       THE LOW ORDER BITS OF THE SAME PORT.
                      91 ;       NOTE ALSO THAT IF A SIXTEEN KEY MATRIX WERE ELECTRICALLY ORGANIZED
                      92 ;       IN A 2X8 ARRAY, ONLY TWO RETURN LINES WOULD BE NEEDED.
                      93 ;       (IN THIS CASE, PERHAPS T0 AND T1 COULD BE USED FOR INPUT BITS.)
                      94 ;
                      95 ;PULL-UP RESISTORS ON THE RETURN LINES MIGHT BE IN ORDER IF THERE IS ANY
                      96 ;POSSIBILITY OF A HIGH-IMPEDENCE CONDUCTIVE PATH THROUGH THE SWITCH WHEN
                      97 ;IT IS SUPPOSED TO BE 'OPEN'.
                      98 ;(THIS PHENOMENON HAS ACTUALLY BEEN OBSERVED.)
                      99 ;
                     100 ;THE DRIVERS USED IN THE PROTOTYPE WERE ALL NON-INVERTING IN THAT
                     101 ;A HIGH LEVEL ON AN OUTPUT LINE IS USED TO TURN A CHARACTER OR SEGMENT ON.
                     102 ;THERE ARE A TOTAL OF SEVEN I/O LINES LEFT OVER.
                     103 ;
                     104 ;THE ALGORITHM FOR DRIVING THE DISPLAY USES A BLOCK OF INTERNAL RAM
                     105 ;AS DISPLAY REGISTERS, WITH ONE BYTE CORRESPONDING TO EACH CHARACTER OF THE
                     106 ;DISPLAY.  THE EIGHT BITS OF EACH BYTE CORRESPOND TO THE SEVEN SEGMENTS & DP
                     107 ;OF EACH CHARACTER.  IF AN EXTERNAL ENCODER IS USED (SUCH AS A FOUR-BIT TO
                     108 ;SEVEN-SEGMENT ENCODER OR A ROM FOR TRANSLATING ASCII TO
                     109 ;SIXTEEN-SEGMENT "STARBURST" DISPLAY PATTERNS), THE TABLE ENTRIES WOULD HOLD
                     110 ;THE CHARACTER CODES.  (IN THE FORMER CASE, AN UNUSED BIT COULD BE USED TO
                     111 ;ENABLE THE D.P.)
                     112 ;THUS, WRITING CHARACTERS TO THE DISPLAY FROM THE BACKGROUND PROGRAM
                     113 ;REALLY ENTAILS WRITING THE APPROPRIATE SEGMENT
                     114 ;PATTERNS TO A DISPLAY REGISTER- THE ACTUAL OUTPUTTING IS AUTOMATIC.
                     115 ;THE LEFTMOST CHARACTER CORRESPONDS TO THE LAST BYTE OF THE DISPLAY
                     116 ;REGISTERS, AND IS ACCESSED BY NEXTPL=8 (SEE SOURCE); THE RIGHTMOST
                     117 ;CHARACTER IS THE FIRST DISPLAY BYTE, WHEN NEXTPL=1.
                     118 ;UTILITY SUBROUTINES ARE INCLUDED HERE TO TRANSLATE FOUR BIT NUMBERS TO HEX
                     119 ;DIGIT PATTERNS, AND WRITE THEM INTO THE DISPLAY REGISTERS SEQUENTIALLY
                     120 ;(EITHER FILLING FROM THE LEFT- H.P. CALCULATOR STYLE OR FROM THE
                     121 ;RIGHT- T.I. STYLE; SUBROUTINES WDISP AND RENTRY, RESPECTIVELY).
                     122 ;
                     123 ;THE KEYBOARD SCANNING ALGORITHM SHOWN HERE REQUIRES A KEY BE DOWN FOR
                     124 ;SOME NUMBER OF COMPLETE DISPLAY SCANS TO BE ACKNOWLEGED.  SINCE IT IS
                     125 ;INTENDED FOR 'ONE-FINGER' OPERATION, TWO-KEY ROLLOVER/N-KEY LOCKOUT HAS
                     126 ;BEEN IMPLEMENTED.  HOWEVER, MODIFICATIONS WOULD BE POSSIBLE TO ALLOW, FOR
                     127 ;EXAMPLE, ONE KEY IN THE MATRIX TO BE USED AS A SHIFT KEY OR CONTROL KEY
                     128 ;TO BE HELD DOWN WHILE ANOTHER KEY IN THE MATRIX IS PRESSED.  (SEE NOTE WITHIN
                     129 ;THE BODY OF THE LISTING.)
                     130 ;
                     131 $EJECT
```

```
 LOC  OBJ        SEQ        SOURCE STATEMENT

                  132 ;(BE AWARE THAT NO MORE THAN TWO KEYS CAN EVER BE DOWN UNLESS DIODES
                  133 ;ARE PLACED IN SERIES WITH ALL OF THE SWITCHES- CERTAINLY NOT THE CASE FOR EL
                  134 ;CHEAPO KEYBOARDS- BECAUSE SOME COMBINATIONS OF THREE KEYS DOWN WILL RESULT
                  135 ;IN A 'PHANTOM' FOURTH KEY BEING PERCEIVED.
                  136 ;THE PHANTOM KEY WOULD BE THE FOURTH 'CORNER' WHEN THREE KEYS FORMING
                  137 ;A RECTANGULAR PATTERN (IN THE X-Y KEY MATRIX) ARE DOWN.)
                  138 ;IF DIODES ARE PLACED IN THE SCANNING ARRAY, CONSIDERATIONS MUST BE MADE
                  139 ;ABOUT HOW THE DIODE VOLTAGE DROP WILL AFFECT INPUT LOGIC LEVELS.
                  140 ;
                  141 ;WHEN A DEBOUNCED KEY IS DETECTED, THE NUMBER OF ITS POSITION IN THE KEY
                  142 ;MATRIX (LEFT-TO-RIGHT, BOTTOM-TO-TOP, STARTING FROM 00) IS PLACED INTO
                  143 ;RAM LOCATION 'KBDBUF'.  AN INPUT SUBROUTINE THEN NEED ONLY READ THIS LOCATION
                  144 ;REPEATEDLY TO DETERMINE WHEN A KEY HAS BEEN PRESSED.  WHEN A KEY IS DETECTED,
                  145 ;A SPECIAL CODE BYTE SHOULD BE WRITTEN BACK TO INTO 'KBDBUF' TO PREVENT
                  146 ;REPEATED DETECTIONS OF THE SAME KEY.
                  147 ;THE ROUTINE 'KBDIN' DEMONSTRATES A TYPICAL INPUT PROTOCOL, ALONG WITH A METHOD
                  148 ;FOR TRANSLATING A KEY POSITION TO ITS ASSOCIATED SIGNIFICANCE BY ACCESSING
                  149 ;TABLE 'LEGNDS' IN ROM.
                  150 ;
                  151 $EJECT
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                152 ;*********************************************************
                153 ;
                154 ;       INITIAL EQUATES TO DEFINE SYSTEM CONFIGURATION
                155 ;
                156 ;*********************************************************
                157 ;
0010            158 PDIGIT  EQU     BUS     ;USED TO ENABLE CHARACTERS AND STROBE ROWS OF KEYBOARD
0008            159 PSGMNT  EQU     P1      ;USED TO TURN ON SEGMENTS OF CURRENTLY ENABLED DIGIT
0009            160 PINPUT  EQU     P2      ;PORT USED TO SCAN FOR KEY CLOSURES
                161                         ;(NOTE THAT THIS PORT ALLOCATION USES THE HIGHER
                162                         ;CURRENT SOURCING ABILITY OF THE BUS TO SWITCH ON THE
                163                         ;DIGIT DRIVERS, AND LEAVES P23-P20 FREE FOR USING
                164                         ;AN 8243 PORT EXPANDER IN THE SYSTEM.)
                165 ;
0000            166 POSLOG  EQU     00H
00FF            167 NEGLOG  EQU     0FFH
                168 ;
0000            169 CHRPOL  EQU     POSLOG  ;DEFINES WHETHER OUTPUT LINES ARE ACTIVE HI OR LOW
0000            170 SEGPOL  EQU     POSLOG  ;\FOR DRIVING CHARACTERS AND SEGMENT PATTERNS
00F0            171 INPMSK  EQU     0F0H    ;DEFINES BITS USED AS INPUT
                172 ;
0008            173 CHARNO  EQU     8       ;NUMBER OF DIGITS IN DISPLAY
0004            174 NROWS   EQU     4       ;ROWS OF KEYS (LESS THAN OR EQUAL TO CHARNO)
0004            175 NCOLS   EQU     4       ;LESSER DIMENSION OF KEYBOARD MATRIX
                176 ;
FFF0            177 TICK    EQU     -10H    ;DETERMINES INTERRUPT INTERVAL
0004            178 DEBNCE  EQU     4       ;NUMBER OF SUCESSIVE SCANS BEFORE KEY CLOSURE ACCEPTED
0000            179 BLANK   EQU     00H     ;CODE TO BLANK DISPLAY CHARACTERS.
                180                         ;(WOULD BE 20H IF ASCII DECODING ROM USED OR 0FH IF
                181                         ;7447-TYPE SEVEN-SEGMENT DECODER EXTERNAL TO 8748)
                182 ;
000F            183 ENCMSK  EQU     0FH     ;SELECTS WHICH BITS ARE RELEVANT TO ENCACC SUBROUTINE
                184 ;
                185 $EJECT
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                186 ;*********************************************************
                187 ;
                188 ;       BANK 0 REGISTERS USED
                189 ;
                190 ;POINTERS USED FOR INDIRECT RAM ACCESSING:
0000            191 PNTR0   EQU     R0
0001            192 PNTR1   EQU     R1
0007            193 NEXTPL  EQU     R7      ;USED TO KEEP TRACK OF CHARACTER POSITION BEING
                194                         ;WRITTEN INTO
                195 ;
                196 ;*********************************************************
                197 ;
                198 ;       BANK 1 REGISTER ALLOCATION
                199 ;
                200 ;PNTR0  EQU     R0      (ALREADY DEFINED)
                201 ;PNTR1  EQU     R1
0002            202 ASAVE   EQU     R2      ;HOLDS ACCUMULATOR VALUE DURING SERVICE ROUTINE
0004            203 ROTPAT  EQU     R4      ;USED TO HOLD INPUT PATTERN BEING ROTATED THROUGH CY
0005            204 ROTCNT  EQU     R5      ;COUNTS NUMBER OF BITS ROTATED THROUGH CY
0006            205 LASTKY  EQU     R6      ;HOLDS KEY POSITION OF LAST KEY DEPRESSION DETECTED
0007            206 CURDIG  EQU     R7      ;HOLDS POSITION OF NEXT CHARACTER TO BE DISPLAYED
                207 ;
                208 ;*********************************************************
                209 ;
                210 ;       DATA RAM ALLOCATION
                211 ;
0020            212 NREPTS  EQU     32      ;KEEPS TRACK OF SUCCESSIVE READS OF SAME KEYSTROKE
0021            213 KEYLOC  EQU     33      ;INCREMENTED AS SUCCESSIVE KEY LOCATIONS SCANNED
0022            214 KBDBUF  EQU     34      ;CARRIES POSITION OF DEBOUNCED KEY FROM REFRSH ROUTINE
                215                         ;\ BACK TO BACKGROUND PROGRAM
0023            216 RDELAY  EQU     35      ;NON-ZERO WHEN DISPLAY IN PROGRESS
                217 ;
                218 ;       THE LAST <CHARNO> REGISTERS HOLD THE DISPLAY SEGMENT PATTERNS
                219 ;
0037            220 SEGMAP  EQU     (63-CHARNO)     ;BASE OF REGISTER ARRAY FOR DISPLAY PATTERNS
                221                                 ;\ (COULD BE ANYWHERE IN INTERNAL RAM)
                222 ;
                223 ;*********************************************************
                224 ;
                225 ;       NOTE THAT LASTKY, CURDIG, AND F1 RETAIN STATUS INFORMATION FROM
                226 ;       ONE INTERRUPT TO THE NEXT.  ALL OTHER REGISTERS MAY BE USED IN
                227 ;       THE USER'S OWN INTERRUPT SERVICING ROUTINE
                228 ;
                229 ;*********************************************************
                230 ;
                231 $EJECT
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                232 ;
                233 ;*************************************************************
                234 ;
0000            235 ORG     000H
0000 0460       236         JMP     INIT
                237 ;
                238 ;
                239 ;*************************************************************
                240 ;
0007            241 ORG     007H
                242 ;
                243 ;TIINT  TIMER INTERRUPT SUBROUTINE.
                244 ;       CALL MADE TO LOC 007H WHEN TIMER TIMES OUT.
                245 ;       TIMER CAN BE RE-INITIALLIZED AT THIS POINT IF DESIRED.
                246 ;       USED HERE TO CAUSE THE DISPLAY REFRESH AND KEY SCAN ROUTINES TO
                247 ;       BE CALLED PERIODICALLY.
0007 D5         248 TIINT:  SEL     RB1
0008 AA         249         MOV     ASAVE,A
0009 23F0       250         MOV     A,#TICK
000B 62         251         MOV     T,A             ;RELOAD TIMER INTERVAL
                252 ;
                253 ;*************************************************************
                254 ;
                255 ;       THE USER'S OWN TIMER INTERRUPT ROUTINE (IF IT EXISTS) COULD
                256 ;       BE PLACED AT THIS POINT
                257 ;
                258 ;*************************************************************
                259 ;
000C 1410       260         CALL    REFRSH          ;CAUSE DISPLAY TO BE UPDATED
                261 ;
                262 ;       THE COMPLETE INTERRUPT ROUTINE SHOULD BE COPIED HERE
                263 ;       TO SAVE A FULL LEVEL OF SUBROUTINE NESTING.
                264 ;       IT WAS WRITTEN AS A SUBROUTINE HERE FOR THE SAKE OF CLARITY.
                265 ;
                266 ;*************************************************************
                267 ;
                268 ;TIRET  TIMER INTERRUPT RETURN CODE- RESTORES ACC VALUE
000E FA         269 TIRET:  MOV     A,ASAVE
000F 93         270         RETR
                271 ;
                272 $EJECT
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                273 ;*********************************************************
                274 ;REFRSH SUBROUTINE TO MULTIPLEX SEVEN-SEGMENT DISPLAYS.
                275 ;       EACH CALL CAUSES THE NEXT CHARACTER TO BE DISPLAYED,
                276 ;       ACCORDING TO THE CONTENTS OF THE SEGMAP REGISTER ARRAY.
                277 ;       REFRSH SHOULD BE CALLED AT LEAST EVERY MSEC OR SO.
                278 ;*********************************************************
                279 ;
0010 2300       280 REFRSH: MOV     A,#BLANK XOR SEGPOL
0012 39         281         OUTL    PSGMNT,A        ;WRITE BLANK PATTERN TO SEG DRIVERS
0013 2357       282 REFR1:  MOV     A,#CHRSTB       ;LOOK UP DIGIT ENABLE PATTERN
0015 6F         283         ADD     A,CURDIG        ;ADD CURDIG DISPLACEMENT
0016 A3         284         MOVP    A,@A            ;ENABLE ONE BIT OF ACCUMULATOR
0017 02         285         OUTL    PDIGIT,A        ;ENERGIZE CHARACTER
                286 ;
                287                                 ;WRITE NEXT SEGMENT PATTERN
0018 2337       288         MOV     A,#SEGMAP       ;LOAD BASE OF REGISTER ARRAY
001A 6F         289         ADD     A,CURDIG        ;ADD CURDIG DISPLACMENT
001B A9         290         MOV     PNTR1,A
001C F1         291         MOV     A,@PNTR1        ;LOAD ACC W/ NEXT SEGMENT PATTERN
001D 39         292         OUTL    PSGMNT,A        ;ENABLE APPROPRIATE SEGMENTS
                293 ;
                294 ;*********************************************************
                295 ;       THE NEXT CHARACTER IS NOW BEING DISPLAYED.
                296 ;       THE KEYBOARD SCAN ROUTINE IS INTEGRATED INTO THE DISPLAY SCAN.
                297 ;       WITH THE CURRENT ROW ENERGIZED, CHECK IF THERE ARE ANY INPUTS.
                298 ;*********************************************************
                299 ;
001E B821       300 SCAN:   MOV     PNTR0,#KEYLOC   ;SET POINTER FOR SEVERAL KEYLOC REFERENCES
0020 0A         301         IN      A,PINPUT        ;LOAD ANY SWITCH CLOSURES
                302 ;
                303 ;#########################################################################
                304 ;##     THIS BLOCK OF CODE IS NOT NEEDED BY THE KEYBOARD SCAN LOGIC.     ###
                305 ;##     HOWEVER, ITS INCLUSION WOULD SPEED THINGS UP A BIT BY           ###
                306 ;##     SKIPPING OVER ROWS IN WHICH NO KEYS ARE DOWN.                   ###
                307 ;##     IT WAS OMITTED HERE TO CONSERVE ROM SPACE, BUT MIGHT BE         ###
                308 ;##     RESTORED IF VERY LARGE KEYBOARDS (ESPECIALLY THOSE WITH EIGHT   ###
                309 ;##     KEYS PER ROW) ARE TO BE USED WITH THIS ALGORITHM.               ###
                310 ;#########################################################################
                311 ;##     CPL     A               ;ANY CLOSURES DETECTED ARE NOW ONE BITS ###
                312 ;##     ANL     A,#INPMSK                                               ###
                313 ;##     JNZ     SCAN1   ;-IF A KEY IN THE CURRENTLY ENABLED ROW IS DOWN ###
                314 ;##;    NO KEY IS NOW DOWN SO THE KEYLOC COUNT MAY BE UPDATED DIRECTLY  ###
                315 ;##     MOV     A,@PNTR0                                                ###
                316 ;##     ADD     A,#NCOLS                                                ###
                317 ;##     MOV     @PNTR0,A                                                ###
                318 ;##     JMP     SCAN6                                                   ###
                319 ;#########################################################################
                320 ;##     IF THIS CODE IS USED, SUBSTITUTE THE 'JC  SCAN5' FOUR LINES     ###
                321 ;##     HENCE WITH 'JNC  SCAN5' TO ACCOMODATE THE INVERTED POLARITY     ###
                322 ;#########################################################################
                323 $EJECT
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                324 ;*********************************************************
                325 ;       ROTATE BITS THROUGH THE CY WHILE INCREMENTING KEYLOC.
                326 ;*********************************************************
                327 ;
0021 BD04       328 SCAN1:  MOV     ROTCNT,#NCOLS   ;SET UP FOR <NCOLS> LOOPS THROUGH 'NXTLOC'
0023 F7         329 NXTLOC: RLC     A
0024 AC         330         MOV     ROTPAT,A        ;SAVE SHIFTED BIT PATTERN
0025 F63F       331         JC      SCAN5           ;ONE BIT IN CY INDICATES KEY NOT DOWN
                332 ;
                333 ;*********************************************************
                334 ;
                335 ;       AT THIS POINT IT HAS JUST BEEN DETERMINED THAT THE VALUE
                336 ;       OF KEYLOC IS THE POSITION OF A KEY WHICH IS NOW DOWN.
                337 ;       THE FOLLOWING CODE DEBOUNCES THE KEY, ETC.
                338 ;       IF MODIFICATIONS TO THE KEYBOARD LOGIC, I.E. THE INCLUSION
                339 ;       OF A SHIFT, CONTROL, OR MODE KEY IN THE KEY MATRIX ITSELF)
                340 ;       ARE DESIRED, THEY SHOULD BE MADE AT THIS POINT, BEFORE
                341 ;       THE DEBOUNCE LOGIC BEGINS.  FOR EXAMPLE, AT THIS POINT
                342 ;       KEYLOC COULD BE COMPARED AGAINST THE POSITION OF THE MODE
                343 ;       KEY, AND IF THEY MATCH SET SOME FLAG BIT AND JUMP TO
                344 ;       LABEL 'SCAN5'.  OR, BY COMPARING KEYLOC AGAINST THE LAST
                345 ;       KEY DEBOUNCED, IMMEDIATE TWO-KEY ROLLOVER COULD BE
                346 ;       IMPLEMENTED.
                347 ;
                348 ;*********************************************************
                349 ;
0027 A5         350         CLR     F1              ;MARK THAT AT LEAST ONE KEY WAS DETECTED
0028 B5         351         CPL     F1              ;\ IN THE CURRENT SCAN
                352 ;
                353 ;*********************************************************
                354 ;       A KEYSTROKE WAS DETECTED FOR THE CURRENT COLUMN.  ITS
                355 ;       POSITION IS IN REGISTER KEYLOC.  SEE IF SAME KEY SENSED LAST CYCLE.
                356 ;*********************************************************
                357 ;
0029 F0         358         MOV     A,@PNTR0        ;PNTR0 STILL HOLDS #KEYLOC
002A 2E         359         XCH     A,LASTKY
002B DE         360         XRL     A,LASTKY
002C B820       361         MOV     PNTR0,#NREPTS   ;PREPARE TO CHECK AND/OR MODIFY REPEAT COUNT
002E C634       362         JZ      SCAN3
                363 ;
                364 $EJECT
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                365 ;*********************************************************
                366 ;       A DIFFERENT KEY WAS READ ON THIS CYCLE THAN ON THE PREVIOUS CYCLE.
                367 ;       SET NREPTS TO THE DEBOUNCE PARAMETER FOR A NEW COUNTDOWN.
                368 ;*********************************************************
                369 ;
0030 B004       370         MOV     @PNTR0,#DEBNCE
0032 043F       371         JMP     SCAN5
                372 ;
                373 ;*********************************************************
                374 ;       SAME KEY WAS DETECTED AS ON PREVIOUS CYCLE
                375 ;       LOOK AT NREPTS: IF ALREADY ZERO, DO NOTHING.
                376 ;       ELSE DECREMENT NREPTS.
                377 ;       IF THIS RESULTS IN ZERO, MOVE LASTKY INTO KBDBUF.
                378 ;*********************************************************
                379 ;
0034 F0         380 SCAN3:  MOV     A,@PNTR0
0035 C63F       381         JZ      SCAN5           ;IF ALREADY ZERO
0037 07         382         DEC     A               ;INDICATE ONE MORE SUCCESIVE KEY DETECTION
0038 A0         383         MOV     @PNTR0,A
0039 963F       384         JNZ     SCAN5           ;IF DECREMENT DOES NOT RESULT IN ZERO
003B FE         385         MOV     A,LASTKY
003C B822       386         MOV     PNTR0,#KBDBUF
003E A0         387         MOV     @PNTR0,A        ;TO MARK NEW KEY CLOSURE
                388 ;
003F B821       389 SCAN5:  MOV     PNTR0,#KEYLOC
0041 10         390         INC     @PNTR0
0042 FC         391         MOV     A,ROTPAT
0043 ED23       392         DJNZ    ROTCNT,NXTLOC
                393 ;
                394 ;
0045 EF57       395 SCAN6:  DJNZ    CURDIG,SCAN9
                396 ;
                397 $EJECT
```

```
 LOC  OBJ        SEQ        SOURCE STATEMENT

                  398 ;
                  399 ;*************************************************************
                  400 ;       THE FOLLOWING CODE SEGMENT IS USED BY THE KEYBOARD SCANNING ROUTINE.
                  401 ;       IT IS EXECUTED ONLY AFTER A REFRESH SEQUENCE OF ALL
                  402 ;       THE CHARACTERS IN THE DISPLAY IS COMPLETED
                  403 ;*************************************************************
                  404 ;
 0047 BF08        405         MOV     CURDIG,#CHARNO
 0049 B000        406         MOV     @PNTR0,#0       ;PNTR0 STILL CONTAINS #KEYLOC
 004B 764F        407         JF1     SCAN8           ;JUMP IF ANY KEYS WERE DETECTED
 004D BEFF        408         MOV     LASTKY,#0FFH    ;CHANGE <LASTKY> WHEN NO KEYS ARE DOWN
 004F A5          409 SCAN8:  CLR     F1
                  410 ;
                  411 ;*************************************************************
                  412 ;       THE NEXT CODE SEGMENT IS THE INTERRUPT-DRIVEN PORTION OF THE 'DELAY'
                  413 ;       UTILITY.  IT DECREMENTS RAM LOCATION 'RDELAY' ONCE PER DISPLAY SCAN
                  414 ;       IF 'RDELAY' IS NOT ALREADY ZERO.
                  415 ;*************************************************************
                  416 ;
 0050 B923        417         MOV     PNTR1,#RDELAY
 0052 F1          418         MOV     A,@PNTR1
 0053 C657        419         JZ      SCAN9
 0055 07          420         DEC     A
 0056 A1          421         MOV     @PNTR1,A
                  422 ;
 0057 83          423 SCAN9:  RET
                  424 ;
                  425 ;*************************************************************
                  426 ;
                  427 ;CHRSTB IS THE BASE FOR THE PATTERNS TO ENABLE ONE-OF-CHARNO CHARACTERS.
 0057             428 CHRSTB  EQU     ($-1) AND 0FFH
 0058 01          429         DB      (00000001B XOR CHRPOL)
 0059 02          430         DB      (00000010B XOR CHRPOL)
 005A 04          431         DB      (00000100B XOR CHRPOL)
 005B 08          432         DB      (00001000B XOR CHRPOL)
 005C 10          433         DB      (00010000B XOR CHRPOL)
 005D 20          434         DB      (00100000B XOR CHRPOL)
 005E 40          435         DB      (01000000B XOR CHRPOL)
 005F 80          436         DB      (10000000B XOR CHRPOL)
                  437
                  438 $EJECT
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                439 ;INIT   INITIALIZES PROCESSOR REGISTERS
0060 D5         440 INIT:   SEL     RB1
0061 BF08       441         MOV     CURDIG,#CHARNO
0063 B822       442         MOV     PNTR0,#KBDBUF
0065 B0FF       443         MOV     @PNTR0,#0FFH
0067 B821       444         MOV     PNTR0,#KEYLOC
0069 B000       445         MOV     @PNTR0,#0
006B 23F0       446         MOV     A,#INPMSK
006D 3A         447         OUTL    PINPUT,A        ;SET BIDIRECTIONAL INPUT LINES
006E C5         448         SEL     RB0
006F 149E       449         CALL    CLEAR           ;UTILITY FOR SETTING INITIAL DISPLAY REGISTERS.
0071 A5         450         CLR     F1
0072 23F0       451         MOV     A,#TICK         ;LOAD INTERRUPT RATE VALUE
0074 62         452         MOV     T,A
0075 55         453         STRT    T
0076 25         454         EN      TCNTI           ;ENABLE TIMER INTERRUPTS
                455 ;
                456 ;
                457 ;*************************************************************
                458 ;
                459 ;ECHO   CHECK FOR ANY NEW KEYSTROKES DETECTED.
                460 ;       TRANSLATE EACH KEYSTROKE INTO A SEGMENT PATTERN
                461 ;       AND WRITE IT INTO THE APPROPRIATE DISPLAY REGISTER.
                462 ;
                463 ;*************************************************************
                464 ;
0077 1483       465 ECHO:   CALL    KBDIN           ;GET NEXT KEYSTROKE
0079 B281       466         JB5     FKEY            ;JUMP IF KEY IN RIGHTHAND COLUMN
                467 ;       SINCE THE ACC IS USED BY ENCACC AND RENTRY, ITS CONTENTS MUST
                468 ;       BE PROCESSED OR SAVED BEFORE ENCACC IS CALLED
007B 14BA       469         CALL    ENCACC          ;FORM APPROPRIATE SEGMENT PATTERN
007D 14DB       470         CALL    RENTRY          ;WRITE PATTERN INTO DISPLAY REGISTERS
007F 0477       471         JMP     ECHO            ;LOOP INDEFINITELY
                472 ;
0081 2400       473 FKEY:   JMP     FUNCTN          ;JUMP TO OFF-PAGE CODE TO CALL DEMO ROUTINE
                474 ;
                475 $EJECT
```

```
LOC  OBJ        SEQ     SOURCE STATEMENT

                476 ;*******************************************************
                477 ;
                478 ;       THE FOLLOWING SUBROUTINES IMPLEMENT THE UTILITIES COMMONLY USED FOR
                479 ;       MOST KEYBOARD/DISPLAY APPLICATIONS.
                480 ;       THEY COULD BE USED EXACTLY AS SHOWN HERE OR ADAPTED FOR SPECIAL CASES.
                481 ;
                482 ;*******************************************************
                483 ;
                484 ;KBDIN  KEYBOARD INPUT SUBROUTINE.
                485 ;       COULD BE USED TO INTERFACE THE USER'S BACKGROUND PROGRAM WITH
                486 ;       THE INTERRUPT DRIVEN KEYBOARD SCANNER.
                487 ;       RETURNS ONLY AFTER A NEW KEYSTROKE HAS BEEN DETECTED AND DEBOUNCED.
                488 ;       ENCODED VALUE OF KEY (RATHER THAN ITS POSITION IN SWITCH MATRIX) IS
                489 ;       RETURNED IN THE ACCUMULATOR.
0083 B922       490 KBDIN:  MOV     PNTR1,#KBDBUF
0085 2380       491         MOV     A,#80H          ;KBDBUF WILL BE MARKED AS CLEAR
0087 21         492         XCH     A,@PNTR1        ;LOAD BUFFER VALUE
0088 F283       493         JB7     KBDIN
008A 038E       494         ADD     A,#LEGNDS       ;ADD BASE OF KEY ENCODING TABLE
008C A3         495         MOVP    A,@A            ;OBTAIN BYTE REPRESENTING KEY SIGNIFICANCE
008D 83         496         RET
                497 ;
                498 ;
                499 ;LEGNDS IS THE BASE FOR TABLE SHOWING KEY MATRIX SIGNIFICANCE
                500 ;       FOR THE KEYBOARD USED IN THE PROTOTYPE.
                501 ;       KEY LAYOUT IS AS SHOWN TO THE RIGHT.
                502 ;
                503 ;       NOTE THAT BIT6-BIT4 MAY BE USED TO ENCODE KEY TYPE.  IN THIS CASE:
                504 ;               BIT4 INDICATES REGULAR DECIMAL DIGITS,
                505 ;               BIT5 INDICATES RIGHT-COLUMN FUNCTION KEYS,
                506 ;               BIT6 INDICATES PUNCTUATION MARKS ( * AND # ).
                507 ;
008E            508 LEGNDS  EQU     ($ AND 0FFH)    ;USE LOW ORDER BITS AS TABLE INDEX
008E 4F         509         DB      4FH
008F 10         510         DB      10H
0090 4E         511         DB      4EH
0091 28         512         DB      28H     ; PDIGIT4==>  1       2       3       <1>
0092 17         513         DB      17H
0093 18         514         DB      18H     ; PDIGIT5==>  4       5       6       <2>
0094 19         515         DB      19H
0095 24         516         DB      24H     ; PDIGIT6==>  7       8       9       <3>
0096 14         517         DB      14H
0097 15         518         DB      15H     ; PDIGIT7==>  *       0       #       <4>
0098 16         519         DB      16H
0099 22         520         DB      22H     ;             !       !       !       !
009A 11         521         DB      11H     ;             !       !       !       !
009B 12         522         DB      12H     ;             V       V       V       V
009C 13         523         DB      13H     ;           PINPUT7 PINPUT6 PINPUT5  PINPUT4
009D 21         524         DB      21H
                525 $EJECT
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                526 ;*********************************************************
                527 ;
                528 ;CLEAR  WRITES 'BLANK' CHARACTERS INTO ALL DISPLAY REGISTERS.
                529 ;       RETURNS WITH NEXTPL SET TO LEFTMOST CHARACTER POSITION
                530 ;FILL   WRITES SEGMENT PATTERN NOW IN ACC INTO ALL DISPLAY REGISTERS
009E 2300       531 CLEAR:  MOV     A,#BLANK XOR SEGPOL
00A0 B938       532 FILL:   MOV     PNTR1,#SEGMAP+1
00A2 BF08       533         MOV     NEXTPL,#CHARNO
00A4 A1         534 CLR1:   MOV     @PNTR1,A        ;STORE THE BLANK CODE
00A5 19         535         INC     PNTR1           ;POINT TO NEXT CHARACTER TO THE LEFT
00A6 EFA4       536         DJNZ    NEXTPL,CLR1
00A8 BF08       537         MOV     NEXTPL,#CHARNO
00AA 83         538         RET
                539 ;
                540 ;*********************************************************
                541 ;
                542 ;PRINT  SUBROUTINE TO COPY A STRING OF BIT PATTERNS FROM ROM TO THE
                543 ;       DISPLAY REGISTERS.  STRING STARTS AT LOCATION POINTED TO BY PNTR0.
                544 ;       CONTINUES UNTIL AN ESCAPE CODE (0FFH) IS REACHED.
                545 ;       NOTE THAT THE CHARACTER STRING PUT OUT MUST BE LOCATED ON THE SAME
                546 ;       PAGE AS THIS SUBROUTINE, SINCE SAME-PAGE MOVES ARE USED.
                547 ;       PRINT IN TURN CALLS EITHER SUBROUTINE 'WDISP' OR 'RENTRY'
                548 ;       TO ACTUALLY EFFECT WRITING INTO THE DISPLAY REGISTERS.
00AB F8         549 PRINT:  MOV     A,PNTR0         ;LOAD NEXT CHARACTER LOCATION
00AC A3         550         MOVP    A,@A            ;LOAD BIT PATTERN INDIRECT
00AD C6B4       551         JZ      PRNT1           ;ESCAPE PATTERN
00AF 14D0       552         CALL    WDISP           ;OUTPUT TO NEXT CHARACTER POSITION
                553 ;##    (CALL    RENTRY          INSTEAD IF MESSAGE IS TO BE RIGHT JUSTIFIED)
00B1 18         554         INC     PNTR0           ;INDEX POINTER
00B2 04AB       555         JMP     PRINT
00B4 83         556 PRNT1:  RET                     ;DONE
                557 ;
                558 ;*********************************************************
                559 ;
                560 ;JOHN   ARRAY HOLDS THE BIT PATTERNS FOR THE LETTERS 'JOHN' (SEE 'TEST2')
                561 ;       (NOTE THAT 'OHN' IS WRITTEN IN LOWER CASE LETTERS)
00B5            562 JOHN    EQU     $ AND 0FFH
00B5 1E         563         DB      00011110B XOR SEGPOL
00B6 5C         564         DB      01011100B XOR SEGPOL
00B7 74         565         DB      01110100B XOR SEGPOL
00B8 54         566         DB      01010100B XOR SEGPOL
00B9 00         567         DB      00
                568 ;
                569 $EJECT
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                570 ;*********************************************************
                571 ;
                572 ;ENCACC ENCODES LSNIBBLE OF ACC INTO HEX BIT PATTERN INTO ACC
00BA 530F       573 ENCACC: ANL     A,#ENCMSK
00BC 03C0       574         ADD     A,#DGPATS
00BE A3         575         MOVP    A,@A
00BF 83         576         RET
                577 ;DGPATS IS THE BASE FOR THE TABLE OF SEGMENT PATTERNS FOR THE BASIC
                578 ;DIGITS.  HERE THE FULL HEX SET (0-F) IS INCLUDED.
                579 ;FOR MANY USER APPLICATIONS, THE CHARACTER SET MAY BE AMENDED OR AUGMENTED
                580 ;TO INCLUDE ADDITIONAL SPECIAL PURPOSE PATTERNS.
                581 ;FORMAT IS      PGFEDCBA        IN STANDARD SEVEN-SEGMENT ENCODING CONVENTION
                582 ;                               WHERE P REPRESENTS THE DECIMAL POINT
00C0            583 DGPATS  EQU     $ AND 0FFH
00C0 3F         584         DB      00111111B XOR SEGPOL
00C1 06         585         DB      00000110B XOR SEGPOL
00C2 5B         586         DB      01011011B XOR SEGPOL
00C3 4F         587         DB      01001111B XOR SEGPOL
00C4 66         588         DB      01100110B XOR SEGPOL
00C5 6D         589         DB      01101101B XOR SEGPOL
00C6 7D         590         DB      01111101B XOR SEGPOL
00C7 07         591         DB      00000111B XOR SEGPOL
00C8 7F         592         DB      01111111B XOR SEGPOL
00C9 67         593         DB      01100111B XOR SEGPOL
00CA 77         594         DB      01110111B XOR SEGPOL
00CB 7C         595         DB      01111100B XOR SEGPOL
00CC 39         596         DB      00111001B XOR SEGPOL
00CD 5E         597         DB      01011110B XOR SEGPOL
00CE 79         598         DB      01111001B XOR SEGPOL
00CF 71         599         DB      01110001B XOR SEGPOL
                600 ;
                601 ;*********************************************************
                602 ;
                603 ;WDISP  WRITES BIT PATTERN NOW IN ACC INTO NEXT CHARACTER POSITION
                604 ;      OF THE DISPLAY (NEXTPL).  ADJUSTS NEXTPL POINTER VALUE.
                605 ;      RESULTS IN DISPLAY BEING FILLED LEFT TO RIGHT, THEN RESTARTING
00D0 A9         606 WDISP:  MOV     PNTR1,A
00D1 FF         607         MOV     A,NEXTPL
00D2 0337       608         ADD     A,#SEGMAP
00D4 29         609         XCH     A,PNTR1
00D5 A1         610         MOV     @PNTR1,A
00D6 EFDA       611         DJNZ    NEXTPL,WDISP1
00D8 BF08       612         MOV     NEXTPL,#CHARNO
00DA 83         613 WDISP1: RET
                614 ;
                615 $EJECT
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                616 ;*********************************************************
                617 ;
                618 ;RENTRY SUBROUTINE TO ENTER ACC CONTENTS INTO THE RIGHTMOST DIGIT
                619 ;       AND SHIFT EVERYTHING ELSE ONE PLACE TO THE LEFT
00DB B938       620 RENTRY: MOV     PNTR1,#SEGMAP+1
00DD BF08       621         MOV     NEXTPL,#CHARNO
00DF 21         622 RENTR1: XCH     A,@PNTR1
00E0 19         623         INC     PNTR1
00E1 EFDF       624         DJNZ    NEXTPL,RENTR1
00E3 BF08       625         MOV     NEXTPL,#CHARNO  ;POINT TO LEFTMOST CHARACTER
00E5 83         626         RET
                627 ;
                628 ;*********************************************************
                629 ;
                630 ;RDPADD TOGGLE DECIMAL POINT IN LAST CHARACTER DISPLAY CHARACTER
                631 ;DPADD  TOGGLES DECIMAL POINT IN THE CHARACTER POINTED TO BY THE ACC
                632 ;
00E6 2301       633 RDPADD: MOV     A,#01H          ;SET INDEX TO RIGHTMOST POSITION
00E8 0337       634 DPADD:  ADD     A,#SEGMAP       ;ACCESS DISPLAY REGISTER FOR DESIRED PLACE
00EA A9         635         MOV     PNTR1,A
00EB F1         636         MOV     A,@PNTR1
00EC D380       637         XRL     A,#80H
00EE A1         638         MOV     @PNTR1,A
00EF 83         639         RET
                640 ,
                641 ;*********************************************************
                642 ;
                643 ;HOLD   SUBROUTINE CALLED WHEN KEY IS KNOWN TO BE DOWN.
                644 ;       WILL NOT RETURN UNTIL KEY IS RELEASED.
00F0 D5         645 HOLD:   SEL     RB1
00F1 FE         646         MOV     A,LASTKY        ;<LASTKY>=0FFH IFF NO KEYS DOWN
00F2 C5         647         SEL     RB0
00F3 37         648         CPL     A
00F4 96F0       649         JNZ     HOLD
00F6 83         650         RET
                651 ;
                652 ;*********************************************************
                653 ;
                654 ;DELAY  SUBROUTINE HANGS UP FOR THE NUMBER OF COMPLETE DISPLAY SCANS EQUAL
                655 ;       TO THE CONTENTS OF THE ACCUMULATOR WHEN CALLED.
00F7 B923       656 DELAY:  MOV     PNTR1,#RDELAY
00F9 A1         657         MOV     @PNTR1,A
00FA F1         658 DELAY1: MOV     A,@PNTR1
00FB 96FA       659         JNZ     DELAY1
00FD 83         660         RET
                661 $EJECT
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

0100             662 ORG    100H
                 663 ;
                 664 ;*************************************************************
                 665 ;
                 666 ;THE CODE ON THIS PAGE IS FOR DEMONSTRATION PURPOSES ONLY-
                 667 ;I TRUELY DOUBT WHETHER ANY END USERS WOULD LIKE TO SEE A NAME
                 668 ;POPPING UP ON THEIR CALCULATOR SCREENS.
                 669 ;HOWEVER, THE CODE SHOWN HERE DOES INDICATE HOW THE UTILITY SUBROUTINES
                 670 ;INCLUDED HERE COULD BE ACCESSED.
                 671 ;THE ROUTINES THEMSELVES ARE CALLED WHEN ONE OF THE FOUR BUTTONS
                 672 ;ON THE RIGHT-HAND SIDE OF THE PROTOTYPE KEYBOARD IS PRESSED.
                 673 ;
                 674 ;*************************************************************
                 675 ;
                 676 ;FUNCTN ROUTINE TO IMPLEMENT ONE OF FOUR DEMO UTILITIES, ACCORDING
                 677 ;       TO WHICH OF THE FOUR FUNCTION KEYS WAS PRESSED
0100 1212        678 FUNCTN: JB0     FUNCT1
0102 320E        679         JB1     FUNCT2
0104 520A        680         JB2     FUNCT3
                 681 ;
0106 14E6        682 FUNCT4: CALL    RDPADD
0108 0477        683         JMP     ECHO
                 684 ;
010A 342E        685 FUNCT3: CALL    TEST3
010C 0477        686         JMP     ECHO
                 687 ;
010E 3424        688 FUNCT2: CALL    TEST2
0110 0477        689         JMP     ECHO
                 690 ;
0112 3416        691 FUNCT1: CALL    TEST1
0114 0477        692         JMP     ECHO
                 693 ;
                 694 ;*************************************************************
                 695 ;
                 696 ;TEST1  CODE SEGMENT TO FILL DISPLAY REGISTERS WITH DIGITS DOWN TO '1'
0116 BF08        697 TEST1:  MOV     NEXTPL,#CHARNO
0118 B808        698         MOV     PNTR0,#CHARNO   ;SET FOR EIGHT LOOP REPETITIONS
011A FF          699 TST11:  MOV     A,NEXTPL
011B 14BA        700         CALL    ENCACC
011D 14D0        701         CALL    WDISP
011F E81A        702         DJNZ    PNTR0,TST11     ;COPY NEXT DIGIT INTO DISPLAY REGISTERS
0121 BF08        703         MOV     NEXTPL,#CHARNO
0123 83          704         RET
                 705 ;
                 706 $EJECT
```

```
 LOC  OBJ        SEQ        SOURCE STATEMENT

                  707 ;*********************************************************
                  708 ;
                  709 ;TEST2  WRITES THE SEGMENT PATTERN FOR 'JOHN' ONTO THE DISPLAY,
                  710 ;       WAITS FOR A WHILE, AND THEN CLEARS THE DISPLAY
 0124 B8B5        711 TEST2:  MOV     PNTR0,#JOHN
 0126 14AB        712         CALL    PRINT
 0128 2364        713         MOV     A,#100  ;SCAN DISPLAY FOR 100 CYCLES
 012A 14F7        714         CALL    DELAY
 012C 049E        715         JMP     CLEAR
                  716 ;
                  717 ;*********************************************************
                  718 ;
                  719 ;TEST3  SUBROUTINE TO FILL DISPLAY WITH DASHES
                  720 ;       JUMPS INTO SUBROUTINE 'CLEAR'
                  721 ;       AS SOON AS THE KEY IS RELEASED.
 012E 2340        722 TEST3:  MOV     A,#01000000B XOR SEGPOL ;PATTERN FOR '-'
 0130 14A0        723         CALL    FILL
 0132 14F0        724         CALL    HOLD
 0134 049E        725         JMP     CLEAR
                  726 ;
                  727 ;*********************************************************
                  728 ;
                  729 END

USER SYMBOLS
ASAVE  0002   BLANK  0000   CHARNO 0008   CHRPOL 0000   CHRSTB 0057   CLEAR  009E   CLR1   00A4   CURDIG 0007
DEBNCE 0004   DELAY  00F7   DELAY1 00FA   DGPATS 00C0   DPADD  00E8   ECHO   0077   ENCACC 00BA   ENCMSK 000F
FILL   00A0   FKEY   0081   FUNCT1 0112   FUNCT2 010E   FUNCT3 010A   FUNCT4 0106   FUNCTN 0100   HOLD   00F0
INIT   0060   INPMSK 00F0   JOHN   00B5   KBDBUF 0022   KBDIN  0083   KEYLOC 0021   LASTKY 0006   LEGNDS 008E
NCOLS  0004   NEGLOG 00FF   NEXTPL 0007   NREPTS 0020   NROWS  0004   NXTLOC 0023   PDIGIT 0010   PINPUT 0009
PNTR0  0000   PNTR1  0001   POSLOG 0000   PRINT  00AB   PRNT1  00B4   PSGMNT 0008   RDELAY 0023   RDPADD 00E6
REFR1  0013   REFRSH 0010   RENTR1 00DF   RENTRY 00DB   ROTCNT 0005   ROTPAT 0004   SCAN   001E   SCAN1  0021
SCAN3  0034   SCAN5  003F   SCAN6  0045   SCAN8  004F   SCAN9  0057   SEGMAP 0037   SEGPOL 0000   TEST1  0116
TEST2  0124   TEST3  012E   TICK   FFF0   TIINT  0007   TIRET  000E   TST11  011A   WDISP  00D0   WDISP1 00DA

ASSEMBLY COMPLETE,  NO ERRORS
```

ASAVE 202# 249 269
BLANK 179# 280 531
CHARNO 173# 220 405 441 533 537 612 621 625 697 698 703
CHRPOL 169# 429 430 431 432 433 434 435 436
CHRSTB 282 428#
CLEAR 449 531# 715 725
CLR1 534# 536
CURDIG 206# 283 289 395 405 441
DEBNCE 178# 370
DELAY 656# 714
DELAY1 658# 659
DGPATS 574 583#
DPADD 634#
ECHO 465# 471 683 686 689 692
ENCACC 469 573# 700
ENCMSK 183# 573
FILL 532# 723
FKEY 466 473#
FUNCT1 678 691#
FUNCT2 679 688#
FUNCT3 680 685#
FUNCT4 682#
FUNCTN 473 678#
HOLD 645# 649 724
INIT 236 440#
INPMSK 171# 446
JOHN 562# 711
KBDBUF 214# 386 442 490
KBDIN 465 490# 493
KEYLOC 213# 300 389 444
LASTKY 205# 359 360 385 408 646
LEGNDS 494 508#
NCOLS 175# 328
NEGLOG 167#
NEXTPL 193# 533 536 537 607 611 612 621 624 625 697 699 703
NREPTS 212# 361
NROWS 174#
NXTLOC 329# 392
PDIGIT 158# 285
PINPUT 160# 301 447
PNTR0 191# 300 358 361 370 380 383 386 387 389 390 406 442 443 444 445
549 554 698 702 711
PNTR1 192# 290 291 417 418 421 490 492 532 534 535 606 609 610 620 622
623 635 636 638 656 657 658
POSLOG 166# 169 170
PRINT 549# 555 712
PRNT1 551 556#
PSGMNT 159# 281 292
RDELAY 216# 417 656
RDPADD 633# 682
REFR1 282#
REFRSH 260 280#
RENTR1 622# 624
RENTRY 470 620#

```
ROTCNT  204#  328   392
ROTPAT  203#  330   391
SCAN    300#
SCAN1   328#
SCAN3   362   380#
SCAN5   331   371   381   384   389#
SCAN6   395#
SCAN8   407   409#
SCAN9   395   419   423#
SEGMAP  220#  288   532   608   620   634
SEGPOL  170#  280   531   563   564   565   566   584   585   586   587   588   589   590   591   592
        593   594   595   596   597   598   599   722
TEST1   691   697#
TEST2   688   711#
TEST3   685   722#
TICK    177#  250   451
TIINT   248#
TIRET   269#
TST11   699#  702
WDISP   552   606#  701
WDISP1  611   613#

CROSS REFERENCE COMPLETE
```

Serial I/O and Math Utilities for the 8049 Microcomputer

Contents

INTRODUCTION 5-90

FULL DUPLEX SERIAL COMMUNICATIONS 5-90

MULTIPLY ALGORITHMS 5-99

DIVIDE ALGORITHMS 5-102

BINARY AND BCD CONVERSIONS 5-105

CONCLUSION 5-111

INTRODUCTION

The Intel® MCS-48 family of microcomputers marked the first time an eight bit computer with program storage, data storage, and I/O facilities was available on a single LSI chip. The performance of the initial processors in the family (the 8748 and the 8048) has been shown to meet or exceed the requirements of most current applications of microcomputers. A new member of the family, however, has been recently introduced which promises to allow the use of the single chip microcomputer in many application areas which have previously required a multichip solution. The Intel® 8049 virtually doubles processing power available to the systems designer. Program storage has been increased from 1K bytes to 2K bytes, data storage has been increased from 64 bytes to 128 bytes, and processing speed has been increased by over 80%. (The 2.5 microsecond instruction cycle of the first members of the family has been reduced to 1.36 microseconds.)

It is obvious that this increase in performance is going to result in far more ambitious programs being written for execution in a single chip microcomputer. This article will show how several program modules can be designed using the 8049. These modules were chosen to illustrate the capability of the 8049 in frequently encountered design situations. The modules included are full duplex serial I/O, binary multiply and divide routines, binary to BCD conversions, and BCD to binary conversion. It should be noted that since the 8049 is totally software compatible with the 8748 and 8048 these routines will also be useful directly on these processors. In addition the algorithms for these programs are expressed in a program design language format which should allow them to be easily understood and extended to suit individual applications with minimal problems.

FULL DUPLEX SERIAL COMMUNICATIONS

Serial communications have always been an important facet in the application of microprocessors. Although this has been partially due to the necessity of connecting a terminal to the microprocessor based system for program generation and debug, the main impetus has been the simple fact that a large share of microprocessors find their way into end products (such as intelligent terminals) which themselves depend on serial communication. When it is necessary to add a serial link to a microprocessor such as the Intel® MCS-85 or 86 the solution is easy; the Intel® 8251A USART or 8273 SDLC chip can easily be added to provide the necessary protocol. When it is necessary to do the same thing to a single chip microcomputer, however, the situation becomes more difficult.

Some microcomputers, such as the Intel 8048 and 8049 have a complete bus interface built into them which allows the simple connection of a USART to the processor chip. Most other single chip microcomputers, although lacking such a bus, can be connected to a USART with various artificial hardware and software constructs. The difficulty with using these chips, however, is more economic than technical; these same peripheral chips which are such a bargain when coupled to a microprocessor such as the MCS-85 or 86, have a significant cost impact on a single chip microcomputer based system. The high speed of the 8049, however, makes it feasible to implement a serial link under software control with no hardware requirements beyond two of the I/O pins already resident on the microcomputer.

There are many techniques for implementing serial I/O under software control. The application note "Application Techniques for the MCS-48 Family" describes several alternatives suitable for half duplex operation. Full duplex operation is more difficult, however, since it requires the receive and transmit processes to operate concurrently. This difficulty is made more severe if it is necessary for some other process to also operate while serial communication is occurring. Scanning a keyboard and display, for example, is a common operation of single chip microcomputer based system which might have to occur concurrently with the serial receive/transmit process. The next section will describe an algorithm which implements full duplex serial communication to occur concurrently with other tasks. The design goal was to allow 2400 baud, full duplex, serial communication while utilizing no more than 50% of the available processing power of the high speed 8049 microcomputer.

The format used for most asynchronous communication is shown in Figure 1. It consists of eight data bits with a leading 'START' bit and one or more trailing 'STOP' bits. The START bit is used to establish synchronization between the receiver and transmitter. The STOP bits ensure that the receiver will be ready to synchronize itself when the next start bit occurs. Two stop bits are normally used for 110 baud communication and one stop bit for higher rates.

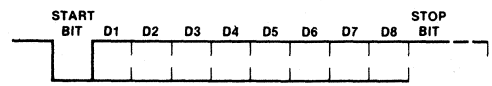


Figure 1.

The algorithm used for reception of the serial data is shown in Figure 2. It uses the on board timer of the 8049 to establish a sampling period of four times the desired baud rates. For 2400 baud operation a crystal frequency of 9.216 MHz was chosen after the following calculation:

$f = 480N(2400)(4)$

where
- 480 is the factor by which the crystal frequency is divided within the processor to get the basic interrupt rate
- 2400 is the desired baud rate
- 4 is the required number of samples per bit time
- N is the value loaded into the MCS-48 timer when it overflows

The value N was chosen to be two (resulting in f = 9.216 MHz) so that the operating frequency of the 8049 could be as high as possible without exceeding the maximum frequency specification of the 8049 (11 MHz).

```
;
;
;START OF RECEIVE ROUTINE
;======================
;
;1 IF RECEIVE FLAG=0 THEN
;2    IF SERIAL INPUT=SPACE THEN
;3       RECEIVE FLAG:=1
;3       BYTE FINISHED FLAG:=0
;2    ENDIF
;1 ELSE    SINCE RECEIVE FLAG=1 THEN
;2    IF SYNC FLAG=0 THEN
;3       IF SERIAL INPUT=SPACE THEN
;4          SYNC FLAG:=1
;4          DATA:=80H
;4          SAMPLE CNTR:=4
;3       ELSE    SINCE SERIAL INPUT=MARK THEN
;4          RECEIVE FLAG:=0
;3       ENDIF
;2    ELSE    SINCE SYNC FLAG=1 THEN
;3       SAMPLE COUNTER:=SAMPLE COUNTER-1
;3       IF SAMPLE COUNTER=0 THEN
;4          SAMPLE COUNTER:=4
;4          IF BYTE FINISHED FLAG=0 THEN
;5             CARRY:=SERIAL INPUT
;5             SHIFT DATA RIGHT WITH CARRY
;5             IF CARRY=1 THEN
;6                OKDATA:=DATA
;6                IF DATA READY FLAG=0 THEN
;7                   BYTE FINISHED FLAG=1
;6                ELSE
;7                   BYTE FINISHED FLAG:=1
;7                   OVERRUN FLAG:=1
;6                ENDIF
;5             ENDIF
;4          ELSE    SINCE BYTE FINISHED FLAG=1 THEN
;5             IF SERIAL INPUT=MARK THEN
;6                DATA READY FLAG:=1
;5             ELSE    SINCE SERIAL INPUT=SPACE THEN
;6                ERROR FLAG:=1
;5             ENDIF
;5             RECEIVE FLAG:=0
;5             SYNC FLAG:=0
;4          ENDIF
;3       ENDIF
;2    ENDIF
;1 ENDIF
```

Figure 2

The timer interrupt service routine always loads the timer with a constant value. In effect the timer is used to generate an independent time base of four times the required baud rate. This time base is free running and is never modified by either the receive or transmit programs, thus allowing both of them to use the same timer. Routines which do other time dependent tasks (such as scanning keyboards) can also be called periodically at some fixed multiple of this basic time unit.

The algorithm shown in Figure 2 uses this basic clock plus a handful of flags to process the serial input data. Once the meaning of these flags are understood the operation of the algorithm should be clear. The ***Receive Flag*** is set whenever the program is in the process of receiving a character. The ***Synch Flag*** is set when the center of the start bit has been checked and found to be a SPACE (if a MARK is detected at this point the receiver process has been triggered by a noise pulse so the program clears the ***Receive Flag*** and returns to the idle state). When the program detects synchronization it loads the variable ***DATA*** with 80H and starts sampling the serial line every four counts. As the data is received it is right shifted into variable ***DATA***; after eight bits have been received the initial one set into ***DATA*** will result in a carry out and the program knows that it has received all eight bits. At this point it will transfer all eight bits to the variable ***OKDATA*** and set the ***Byte Finished Flag*** so that on the next sample it will test for a valid stop bit instead of shifting in data. If this test is successful the ***Data Ready Flag*** will be set to indicate that the data is available to the main process. If the test is unsuccessful the ***Error Flag*** will be set.

The transmit algorithm is shown in Figure 3. It is executed immediately following the receive process. It is a simple program which divides the free running clock down and transmits a bit every fourth clock. The variable ***TICK COUNTER*** is used to do the division. The ***Transmitting Flag*** indicates when a character transmission is in progress and is also used to determine when the START bit should be sent. The ***TICK COUNTER*** is used to determine when to send the next bit (TICK COUNTER MODULO 4 = 0) and also when the STOP bits should be sent (TICK COUNTER = 9 4). After the transmit routine completes any other timer based routines, such as a keyboard/display scanner or a real time clock, can be executed.

```
;
;START OF TRANSMIT ROUTINE
;=======================
;
;1
;1 TICK COUNTER:=TICK COUNTER+1
;1 IF TICK COUNTER MOD 4=0 THEN
;2    IF TRANSMITTING FLAG=1 THEN
;3       IF TICK COUNTER=00 1010 00 BINARY THEN
;4          TRANSMITTING FLAG:=0
;3       ELSE    IF TICK COUNTER=00 1001 00 BINARY THEN
;4          SEND END MARK
;4          TRANSMITTING FLAG:=0
;3       ELSE    SINCE TICK COUNTER<>THE ABOVE COUNT THEN
;4          SEND NEXT BIT
;3       ENDIF
;2    ELSE    SINCE TRANSMITTING FLAG=0 THEN
;3       IF TRANSMIT REQUEST FLAG=1 THEN
;4          XMTBYT:=NXTBYT
;4          TRANSMIT REQUEST FLAG:=0
;4          TRANSMITTING FLAG:=1
;4          TICK COUNTER:=0
;4          SEND SYNC BIT (SPACE)
;3       ENDIF
;2    ENDIF
;1 ENDIF
```

Figure 3

Figure 4 shows the complete receive and transmit programs as they are implemented in the instruction set of the 8049. Also included in Fig. 4 is a short routine which was used to test the algorithm.

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0


 LOC  OBJ        SEQ        SOURCE STATEMENT

                    1 ;*****************************************************************************
                    2 ;*                                                                           *
                    3 ;*      THIS PROGRAM TESTS THE FULL DUPLEX COMMUNICATION SOFTWARE            *
                    4 ;*                                                                           *
                    5 ;*****************************************************************************
                    6 ;
                    7 $INCLUDE(:F1:URTEST.PDL)
                =   8 ;
                =   9 ;       START OF TEST ROUTINE
                =  10 ;       ====================
                =  11 ;
                =  12 ;
                =  13 ;
                =  14 ;
                =  15 ;
                =  16 ;1 ERROR COUNT:=0
                =  17 ;1 REPEAT
                =  18 ;2    PATTERN:=0
                =  19 ;2    INITIALIZE TIMER
                =  20 ;2    CLEAR FLAGBYTE
                =  21 ;2    FLAG1=MARK
                =  22 ;2    REPEAT
                =  23 ;3       IF TRANSMIT REQUEST FLAG=0 THEN
                =  24 ;4          NXTBYTE:=PATTERN
                =  25 ;4          TRANSMIT REQUEST FLAG=1
                =  26 ;3       ENDIF
                =  27 ;3       IF DATA READY FLAG=1 THEN
                =  28 ;4          PATTERN:=OKDATA
                =  29 ;4          DATA READY FLAG:=0
                =  30 ;3       ENDIF
                =  31 ;2    UNTIL ERROR FLAG OR OVERRUN FLAG
                =  32 ;2    INCREMENT ERROR COUNT
                =  33 ;1 UNTIL FOREVER
                =  34 ;EOF
                   35 $EJECT
0000               36          ORG     0
                   37 ;1 SELECT REGISTER BANK 0
0000 C5            38          SEL     RB0
                   39 ;1 GOTO TEST
0001 2400          40          JMP     TEST
                   41 $        INCLUDE(:F1:UART)
                =  42 ;
                =  43 ;
                =  44 ;       ASYNCHRONOUS RECEIVE/TRANSMIT ROUTINE
                =  45 ;       ====================================
                =  46 ;       THIS ROUTINE RECEIVES SERIAL CODE USING PIN T0 AS RXD
                =  47 ;       AND CONCURRENTLY TRANSMITS USING PIN P27
                =  48 ;       NOTE:
                =  49 ;          THIS ROUTINE USES FLAG 1 TO BUFFER THE TRANSMITTED
```

Figure 4

```
LOC  OBJ          SEQ          SOURCE STATEMENT

                  =  50 ;1 DATA LINE.  THIS ELIMINATES THE JITTER THAT
                  =  51 ;1 WOULD BE CAUSED BY VARIATIONS IN THE RECEIVE
                  =  52 ;1 TIMING.  NO OTHER PROGRAM MAY USE FLAG 1 WHILE
                  =  53 ;1 THE TIMER INTERRUPT IS ENABLED
                  =  54 ;
                  =  55 ;
                  =  56 ;
                  =  57 ;
                  =  58 ;
                  =  59 ;         REGISTER ASSIGNMENTS-BANK1
                  =  60 ;         ========================
                  =  61 ;
                  =  62 ;
0007              =  63 ATEMP     EQU     R7      ; USED TO SAVE ACCUMULATOR CONTENTS DURING INTERRUPT
0006              =  64 FLGBYT    EQU     R6      ; CONTAINS VARIOUS FLAGS USED TO CONTROL THE RECEIVE
                  =  65                           ; AND TRANSMIT PROCESS.  SEE CONSTANT DEFINITIONS FOR
                  =  66                           ; THE MEANING OF EACH BIT
0005              =  67 SAMCTR    EQU     R5      ; SAMPLE COUNTER FOR THE RECIEVE PROCESS
0004              =  68 TCKCTR    EQU     R4      ; SAMPLE COUNTER FOR THE TRANSMIT PROCESS
0000              =  69 REG0      EQU     R0      ; USED AS POINTER REGISTER
                  =  70 ;
                  =  71 ;         RAM ASSIGNMENTS
                  =  72 ;         ===============
                  =  73 ;
0020              =  74 MOKDAT    EQU     20H     ; RECEIVE RETURNS VALID DATA IN THIS BYTE
0021              =  75 MDATA     EQU     21H     ; RECEIVE ACCUMULATES DATA IN THIS BYTE
0022              =  76 MXMTBY    EQU     22H     ; CONTAINS BYTE BEING TRANSMITTED
0023              =  77 MNXTBY    EQU     23H     ; CONTAINS THE NEXT BYTE TO BE TRANSMITTED
                  =  78 $EJECT
                  =  79 ;
                  =  80 ;
                  =  81 ;         CONSTANTS
                  =  82 ;         =========
                  =  83 ;
                  =  84 ;         THE FOLLOWING CONSTANTS ARE USED TO ACCESS THE FLAG BITS CONTAINED
                  =  85 ;         IN REGISTER FLGBYT
                  =  86 ;
0001              =  87 RCVFLG    EQU     01H     ; SET WHEN START BIT IS FIRST DETECTED
                  =  88                           ; RESET WHEN RECEIVE PROCESS IS COMPLETE
0002              =  89 SYNFLG    EQU     02H     ; SET WHEN START BIT IS VERIFIED
                  =  90                           ; RESET WHEN RECEIVE PROCESS IS COMPLETE
0004              =  91 BYFNFL    EQU     04H     ; RESET WHEN START BIT IS FIRST DETECTED
                  =  92                           ; SET WHEN THE EIGHT DATA BITS HAVE ALL BEEN RECEIVED
0008              =  93 DRDYFL    EQU     08H     ; SHOULD BE RESET BY MAIN PROGRAM WHEN DATA IS ACCEPTED
                  =  94                           ; SET BY RECEIVE PROCESS WHEN STOP BIT(S) ARE VERIFIED
0010              =  95 ERRFLG    EQU     10H     ; SHOULD BE RESET BY MAIN PROGRAM WHEN SAMPLED
                  =  96                           ; SET BY RECEIVE PROCESS IF A FRAMING ERROR IS DETECTED
0020              =  97 TRRQFL    EQU     20H     ; TESTED BY MAIN PROGRAM TO DETERMINE IF READY TO
                  =  98                           ; TRANSMIT A NEW BYTE-SET TO INDICATE THAT NXTBYT
                  =  99                           ; HAS BEEN LOADED
                  = 100                           ; RESET BY TRANSMIT PROCESS WHEN BYTE IS ACCEPTED
0040              = 101 TRNGFL    EQU     40H     ; SET WHEN TRANSMISSION OF A BYTE STARTS
                  = 102                           ; RESET WHEN STOP BIT IS TRANSMITTED
0080              = 103 OVRUN     EQU     80H     ; SET BY RECEIVE PROCESS WHEN OVERUN OCCURRS
                  = 104                           ; SHOULD BE RESET BY MAIN PROGRAM WHEN SAMPLED
```

Figure 4 (continued)

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                = 105 ;
                = 106 ;       GENERAL CONSTANTS
                = 107 ;       =================
                = 108 ;
0080            = 109 MARK    EQU     80H     ; USED TO GENERATED A MARK
FF7F            = 110 SPACE   EQU     NOT 80H ; USED TO GENERATE A SPACE
0000            = 111 STPBTS  EQU     0       ; CONTROLS THE NUMBER OF STOP BITS
                = 112                         ;   0 GENERATES ONE STOP BIT
                = 113                         ;   1 GENERATES TWO STOP BITS
                = 114 ;
                = 115 $EJECT
                = 116 ;
                = 117 ;       START OF RECEIVE/TRANSMIT INTERRUPT SERVICE ROUTINE
                = 118 ;       ================================================
                = 119 ;
0007            = 120         ORG     0007H
                = 121
                = 122 ;1 ENTER INTERRUPT MODE
0007 160A       = 123 TISR:   JTF     UART
0009 93         = 124         RETR
000A D5         = 125 UART:   SEL     RB1
                = 126 ;1 SAVE ACCUMULATOR CONTENTS
000B AF         = 127         MOV     ATEMP,A
                = 128 ;1 RELOAD TIMER
000C 23FE       = 129         MOV     A,#TIMCNT
000E 62         = 130         MOV     T,A
                = 131 ;
                = 132 ;       OUTPUT TXD BUFFER (F1) TO TXD I/O LINE (P27)
                = 133 ;       ============================================
                = 134 ;
000F 7615       = 135         JF1     OMARK
0011 9A7F       = 136 OSPACE: ANL     P2,#SPACE
0013 0417       = 137         JMP     RCV000
0015 8A80       = 138 OMARK:  ORL     P2,#MARK
                = 139 ;
                = 140 ;       START OF RECEIVE ROUTINE
                = 141 ;       ========================
                = 142 ;
                = 143 ;1 IF RECEIVE FLAG=0 THEN
0017 FE         = 144 RCV000: MOV     A,FLGBYT
0018 1224       = 145         JB0     RCV010
                = 146 ;2    IF SERIAL INPUT=SPACE THEN
001A 3664       = 147         JT0     XMIT
                = 148 ;3      RECEIVE FLAG:=1
001C FE         = 149         MOV     A,FLGBYT
001D 4301       = 150         ORL     A,#RCVFLG
                = 151 ;3      BYTE FINISHED FLAG:=0
001F 53FB       = 152         ANL     A,#NOT BYFNFL
                = 153 ;2    ENDIF
0021 AE         = 154         MOV     FLGBYT,A
0022 0464       = 155         JMP     XMIT
                = 156 ;1 ELSE    SINCE RECEIVE FLAG=1 THEN
                = 157 ;2    IF SYNC FLAG=0 THEN
0024 3238       = 158 RCV010: JB1     RCV030
                = 159 ;3      IF SERIAL INPUT=SPACE THEN
```

Figure 4 (continued)

```
LOC  OBJ        SEQ         SOURCE STATEMENT

0026 3633       = 160          JT0     RCV020
                = 161 ;4           SYNC FLAG:=1
0028 4302       = 162          ORL     A,#SYNFLG
002A AE         = 163          MOV     FLGBYT,A
                = 164 ;4           DATA:=80H
002B B821       = 165          MOV     R0,#MDATA
002D B080       = 166          MOV     @R0,#80H
                = 167 ;4           SAMPLE CNTR:=4
002F BD04       = 168          MOV     SAMCTR,#4
0031 0464       = 169          JMP     XMIT
                = 170 ;3        ELSE    SINCE SERIAL INPUT=MARK THEN
                = 171 ;4           RECEIVE FLAG:=0
0033 53FE       = 172 RCV020: ANL      A,#NOT RCVFLG
                = 173 ;3        ENDIF
0035 AE         = 174          MOV     FLGBYT,A
0036 0464       = 175          JMP     XMIT
                = 176 ;2     ELSE    SINCE SYNC FLAG=1 THEN
                = 177 ;3        SAMPLE COUNTER:=SAMPLE COUNTER-1
0038 ED64       = 178 RCV030: DJNZ     SAMCTR,XMIT
                = 179 ;3        IF SAMPLE COUNTER=0 THEN
                = 180 ;4           SAMPLE COUNTER:=4
003A BD04       = 181          MOV     SAMCTR,#4
                = 182 ;4           IF BYTE FINISHED FLAG=0 THEN
003C 5259       = 183          JB2     RCV050
003E 97         = 184          CLR     C
                = 185 ;5              CARRY:=SERIAL INPUT
003F 2642       = 186          JNT0    RCV040
0041 A7         = 187          CPL     C
0042 B821       = 188 RCV040: MOV      R0,#MDATA
0044 F0         = 189          MOV     A,@R0
                = 190 ;5              SHIFT DATA RIGHT WITH CARRY
0045 67         = 191          RRC     A
0046 A0         = 192          MOV     @R0,A
                = 193 ;5              IF CARRY=1 THEN
0047 E664       = 194          JNC     XMIT
                = 195 ;6                 OKDATA:=DATA
0049 B820       = 196          MOV     R0,#MOKDAT
004B A0         = 197          MOV     @R0,A
                = 198 ;6                 IF DATA READY FLAG=0 THEN
004C FE         = 199          MOV     A,FLGBYT
004D 7254       = 200          JB3     RCV045
                = 201 ;7                    BYTE FINISHED FLAG=1
004F 4304       = 202          ORL     A,#BYFNFL
0051 AE         = 203          MOV     FLGBYT,A
0052 0464       = 204          JMP     XMIT
                = 205 ;6                 ELSE
                = 206 ;7                    BYTE FINISHED FLAG:=1
                = 207 ;7                    OVERRUN FLAG:=1
                = 208 RCV045:
                = 209          ;MOV     A,FLGBYT
0054 4384       = 210          ORL     A,#(BYFNFL OR OVRUN)
0056 AE         = 211          MOV     FLGBYT,A
                = 212 ;6                 ENDIF
                = 213 ;5              ENDIF
0057 0464       = 214          JMP     XMIT
```

Figure 4 (continued)

```
LOC  OBJ        SEQ         SOURCE STATEMENT

                = 215 ;4          ELSE    SINCE BYTE FINISHED FLAG=1 THEN
                = 216 ;5            IF SERIAL INPUT=MARK THEN
0059 265F       = 217 RCV050: JNT0    RCV060
                = 218 ;6                DATA READY FLAG:=1
005B 4308       = 219         ORL     A,#DRDYFL
005D 0461       = 220         JMP     RCV070
                = 221 ;5            ELSE    SINCE SERIAL INPUT=SPACE THEN
                = 222 ;6                ERROR FLAG:=1
005F 4310       = 223 RCV060: ORL     A,#ERRFLG
                = 224 ;5            ENDIF
                = 225 ;5            RECEIVE FLAG:=0
                = 226 ;5            SYNC FLAG:=0
0061 53FC       = 227 RCV070: ANL     A,#NOT(SYNFLG OR RCVFLG)
0063 AE         = 228         MOV     FLGBYT,A
                = 229 ;4          ENDIF
                = 230 ;3        ENDIF
                = 231 ;2      ENDIF
                = 232 ;1 ENDIF
                = 233 $EJECT
                = 234 ;
                = 235 ;       START OF TRANSMIT ROUTINE
                = 236 ;       ========================
                = 237 ;
                = 238 ;1
                = 239         ;TRANSMITTER OUTPUT BIT IS P2-7
                = 240 ;1 TICK COUNTER:=TICK COUNTER+1
0064 1C         = 241 XMIT:   INC     TCKCTR
                = 242 ;1 IF TICK COUNTER MOD 4=0 THEN
0065 2303       = 243         MOV     A,#03H
0067 5C         = 244         ANL     A,TCKCTR
0068 9697       = 245         JNZ     RETURN
                = 246 ;2    IF TRANSMITTING FLAG=1 THEN
006A FE         = 247         MOV     A,FLGBYT
006B 37         = 248         CPL     A
006C D286       = 249         JB6     XMT040
                = 250         IF STPBTS EQ 1
                = 251 ;3       IF TICK COUNTER=00 1010 00 BINARY THEN
                = 252         MOV     A,#28H                  ; CONDITIONAL ASSEMBLY
                = 253         XRL     A,TCKCTR                ;
                = 254         JNZ     XMT010                  ;
                = 255 ;4          TRANSMITTING FLAG:=0
                = 256         MOV     A,FLGBYT                ;
                = 257         ANL     A,#NOT TRNGFL           ;
                = 258         MOV     FLGBYT,A                ;
                = 259         JMP     RETURN                  ;
                = 260         ENDIF
                = 261 ;3       ELSE    IF TICK COUNTER=00 1001 00 BINARY THEN
006E 2324       = 262 XMT010: MOV     A,#24H
0070 DC         = 263         XRL     A,TCKCTR
0071 967B       = 264         JNZ     XMT020
                = 265 ;4          SEND END MARK
0073 A5         = 266         CLR     F1      ; SET FLAG1 TO MARK
0074 B5         = 267         CPL     F1
                = 268         IF STPBTS EQ 0
                = 269 ;4          TRANSMITTING FLAG:=0
```

Figure 4 (continued)

```
LOC  OBJ        SEQ         SOURCE STATEMENT

0075 FE         = 270         MOV     A,FLGBYT                ; CONDITIONAL ASSEMBLY
0076 53BF       = 271         ANL     A,#NOT TRNGFL           ;
0078 AE         = 272         MOV     FLGBYT,A                ;
0079 0497       = 273         JMP     RETURN                  ;
                = 274         ENDIF
                = 275 ;3       ELSE    SINCE TICK COUNTER<>THE ABOVE COUNT THEN
                = 276 ;4          SEND NEXT BIT
007B B822       = 277 XMT020: MOV     R0,#MXMTBY
007D F0         = 278         MOV     A,@R0
007E 67         = 279         RRC     A
007F A0         = 280         MOV     @R0,A
0080 A5         = 281         CLR     F1      ; FLAG 1 WILL BE USED TO BUFFER TXD
0081 E697       = 282         JNC     RETURN  ; GO TO RETURN POINT IF TXD=SPACE (0)
0083 B5         = 283         CPL     F1      ; ELSE COMPLEMENT FLAG 1 TO A MARK
0084 0497       = 284         JMP     RETURN
                = 285 ;3       ENDIF
                = 286 ;2    ELSE    SINCE TRANSMITTING FLAG=0 THEN
                = 287 ;3       IF TRANSMIT REQUEST FLAG=1 THEN
0086 B297       = 288 XMT040: JB5     RETURN                  ;FLAG BYTE THERE
                = 289 ;4          XMTBYT:=NXTBYT
0088 B823       = 290         MOV     R0,#MNXTBY
008A F0         = 291         MOV     A,@R0
008B B822       = 292         MOV     R0,#MXMTBY
008D A0         = 293         MOV     @R0,A
                = 294 ;4          TRANSMIT REQUEST FLAG:=0
008E FE         = 295         MOV     A,FLGBYT
008F 53DF       = 296         ANL     A,#NOT TRRQFL
                = 297 ;4          TRANSMITTING FLAG:=1
0091 4340       = 298         ORL     A,#TRNGFL
0093 AE         = 299         MOV     FLGBYT,A
                = 300 ;4          TICK COUNTER:=0
0094 BC00       = 301         MOV     TCKCTR,#0
                = 302 ;4          SEND SYNC BIT (SPACE)
0096 A5         = 303         CLR     F1      ; SET FLAG 1 TO CAUSE A SPACE
                = 304 ;3       ENDIF
                = 305 ;2    ENDIF
                = 306 ;1 ENDIF
                = 307 RETURN:
                = 308 ;1 RESTORE ACCUMULATOR
0097 FF         = 309         MOV     A,ATEMP
0098 93         = 310         RETR
                  311 $EJECT
                  312 ;
                  313 ;       START OF TEST ROUTINE
                  314 ;       ====================
                  315 ;
0100              316         ORG     0100H
FFFE              317 TIMCNT  EQU     -2
001E              318 MFLGBY  EQU     1EH
001D              319 MSAMCT  EQU     1DH
001C              320 MTCKCT  EQU     1CH
                  321 ;
0007              322 ERRCNT  EQU     R7
0006              323 PATT    EQU     R6
                  324 ;
```

Figure 4 (continued)

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                325 ;
                326 ;
                327 ;1 ERROR COUNT:=0
0100 BF00       328 TEST:   MOV     ERRCNT,#0
                329 ;1 REPEAT
                330 TLOP:
                331 ;2    PATTERN:=0
0102 BE00       332         MOV     PATT,#00
                333 ;2    INITIALIZE TIMER
0104 23FE       334         MOV     A,#TIMCNT
0106 62         335         MOV     T,A
0107 55         336         STRT    T
0108 25         337         EN      TCNTI
                338 ;2    CLEAR FLAGBYTE
0109 B81E       339         MOV     R0,#MFLGBY
010B B000       340         MOV     @R0,#0
                341 ;2    FLAG1=MARK
010D A5         342         CLR     F1
010E B5         343         CPL     F1
                344 ;2    REPEAT
                345 TILOP:
                346 ;3      IF TRANSMIT REQUEST FLAG=0 THEN
010F B81E       347         MOV     R0,#MFLGBY
0111 F0         348         MOV     A,@R0
0112 B224       349         JB5     TREC
                350 ;4          NXTBYTE:=PATTERN
0114 B923       351         MOV     R1,#MNXTBY
0116 FE         352         MOV     A,PATT
0117 A1         353         MOV     @R1,A
                354 ;4          TRANSMIT REQUEST FLAG=1
0118 35         355         DIS     TCNTI   ; LOCK OUT TIMER INTERRUPT
                356                         ; SO THAT MUTUAL EXCLUSION IS MAINTAINED WHILE
                357                         ; THE FLAG BYTE IS BEING MODIFIED
0119 F0         358         MOV     A,@R0
011A 4320       359         ORL     A,#TRRQFL
011C A0         360         MOV     @R0,A
011D 25         361         EN      TCNTI
011E 1622       362         JTF     TESTA
0120 2424       363         JMP     TREC
0122 140A       364 TESTA:  CALL    UART    ; CALL UART BECAUSE TIMER OVERFLOWED DURING LOCKOUT
                365 ;3      ENDIF
                366 ;3      IF DATA READY FLAG=1 THEN
                367 TREC:
0124 F0         368         MOV     A,@R0
0125 37         369         CPL     A
0126 7238       370         JB3     TRECE
                371 ;4          PATTERN:=OKDATA
0128 B920       372         MOV     R1,#MOKDAT
012A F1         373         MOV     A,@R1
012B AE         374         MOV     PATT,A
                375 ;4          DATA READY FLAG:=0
012C 35         376         DIS     TCNTI   ; LOCK OUT TIMER INTERRUPT
                377                         ; SO THAT MUTUAL EXCLUSION IS MAINTIANED WHILE
                378                         ; THE FLAG BYTE IS BEING MODIFIED
012D F0         379         MOV     A,@R0
```

Figure 4 (continued)

```
LOC  OBJ        SEQ        SOURCE STATEMENT

012E 53F7          380          ANL     A,#NOT DRDYFL
0130 A0            381          MOV     @R0,A
0131 25            382          EN      TCNTI
0132 1636          383          JTF     TESTB
0134 2438          384          JMP     TRECE
0136 140A          385 TESTB:   CALL    UART    ; CALL UART IF TIMER OVERFLOWED DURING LOCKOUT
                   386 TRECE:
                   387 ;3         ENDIF
                   388 ;2     UNTIL ERROR FLAG OR OVERRUN FLAG
0138 F0            389          MOV     A,@R0
0139 5390          390          ANL     A,#(OVRUN OR ERRFLG)
013B C60F          391          JZ      TILOP
                   392 ;2     INCREMENT ERROR COUNT
013D 1F            393          INC     ERRCNT
                   394 ;1 UNTIL FOREVER
013E 2402          395          JMP     TLOP
                   396 ;EOF
                   397          END


USER SYMBOLS
ATEMP  0007   BYFNFL 0004   DRDYFL 0008   ERRCNT 0007   ERRFLG 0010   FLGBYT 0006   MARK   0080   MDATA  0021
MFLGBY 001E   MNXTBY 0023   MOKDAT 0020   MSAMCT 001D   MTCKCT 001C   MXMTBY 0022   OMARK  0015   OSPACE 0011
OVRUN  0080   PATT   0006   RCV000 0017   RCV010 0024   RCV020 0033   RCV030 0038   RCV040 0042   RCV045 0054
RCV050 0059   RCV060 005F   RCV070 0061   RCVFLG 0001   REG0   0000   RETURN 0097   SAMCTR 0005   SPACE  FF7F
STPBTS 0000   SYNFLG 0002   TCKCTR 0004   TEST   0100   TESTA  0122   TESTB  0136   TILOP  010F   TIMCNT FFFE
TISR   0007   TLOP   0102   TREC   0124   TRECE  0138   TRNGFL 0040   TRRQFL 0020   UART   000A   XMIT   0064
XMT010 006E   XMT020 007B   XMT040 0086


ASSEMBLY COMPLETE,  NO ERRORS
```

Figure 4 (continued)

MULTIPLY ALGORITHMS

Most microcomputer programmers have at one time or another implemented a multiply routine as part of a larger program. The usual procedure is to find an algorithm that works and modify it to work on the machine being used. There is nothing wrong with this approach. If engineers felt that they had to reinvent the wheel every time a new design is undertaken, that's probably what most of us would be doing—designing wheels. If the efficiency of the multiply algorithm, either in terms of code size or execution time is important, however, it is necessary to be reasonably familiar with the multiplication process so that appropriate optimizations for the machine being used can be made.

To understand how multiplication operates in the binary number system, consider the multiplication of two four bit operands A and B. The "ones and zeros" in A and B represent the coefficients of two polynomials. The operation A × B can be represented as the following multiplication of polynomials:

$$\begin{array}{rl} & A3*2^3 + A2*2^2 + A1*2^1 + A0*2^0 \\ X & B3*2^3 + B2*2^2 + B1*2^1 + B0*2^0 \end{array}$$

$$\begin{array}{l} + B0A3*2^3 + B0A2*2^2 + B0A1*2^1 + B0A0*2^0 \\ + B1A3*2^4 + B1A2*2^3 + B1A1*2^2 + B1A0*2^1 \\ + B2A3*2^5 + B2A2*2^4 + B2A1*2^3 + B2A0*2^2 \\ + B3A3*2^6 + B3A2*2^5 + B3A1*2^4 + B3A0*2^3 \end{array}$$

The sum of all these terms represents the product of A and B. The simplest multiply algorithm factors the above terms as follows:

$$A*B = B0*(A)*2^0 + B1*(A)*2^1 + B2*(A)*2^2 + B3*(A)*2^3$$

Since the coefficients of B (i.e., B0, B1, B2, and B3) can only take on the binary values of 1 or 0, the sum of the products can be formed by a series of simple adds and multiplications by two. The simplest implementation of this would be:

```
MULTIPLY:
  PRODUCT = 0
  IF B0 = 1 THEN PRODUCT: = PRODUCT + A
  IF B1 = 1 THEN PRODUCT: = PRODUCT + 2*A
  IF B2 = 1 THEN PRODUCT: = PRODUCT + 4*A
  IF B3 = 1 THEN PRODUCT: = PRODUCT + 8*A
END MULTIPLY
```

In order to conserve memory, the above straight line code is normally converted to the following loop:

```
MULTIPLY:
 PRODUCT: = 0
 COUNT: = 4
 REPEAT
  IF B[0] = 1 THEN PRODUCT: = PRODUCT + A ENDIF
  A: = 2*A
  B: = B/2
  COUNT: = COUNT - 1
 UNTIL COUNT: = 0
END MULTIPLY
```

The repeated multiplication of A by two (which can be performed by a simple left shift) forms the terms 2*A, 4*A, and 8*A. The variable B is divided by two (performed by a simple right shift) so that the least significant bit can always be used to determine whether the addition should be executed during each pass through the loop. It is from these shifting and addition operations that the "shift and add" algorithm takes its common name.

The "shift and add" algorithm shown above has two areas where efficiency will be lost if implemented in the manner shown. The first problem is that the addition to the partial product is double precision relative to the two operands. The other problem, which is also related to double precision operations, is that the A operand is double precision and that it must be left shifted and then the B operand must be right shifted. An examination of the "longhand" polynomial multiplication will reveal that, although the partial product is indeed double precision, each addition performed is only single precision. It would be desirable to be able to shift the partial product as it is formed so that only single precision additions are performed. This would be especially true if the partial product could be shifted into the "B" operand since one bit of the partial product is formed during each pass through the loop and (happily) one bit of the "B" operand is vacated. To do this, however, it is necessary to modify the algorithm so that both of the shifts that occur are of the same type.

To see how this can be done one can take the basic multiplication equation already presented:

$$A*B = B0*(A*2^0) + B1*(A*2^1) + B2*(A*2^2) + B3*(A*2^3)$$

and factoring 2^4 from the right side:

$$A*B = 2^4[B0*(A*2^{-4}) + B1*(A*2^{-3}) + B2*(A*2^{-2}) + B3*(A*2^{-1})]$$

This operation has resulted in a term (within the brackets) which can be formed by right shifts and adds and then multiplied by 2^4 to get the final result. The resulting algorithm, expanded to form an eight by eight multiplication, is shown in figure 5. Note that although the result is a full sixteen bits, the algorithm only performs eight bit additions and that only a single sixteen bit shift operation is involved. This has the effect of reducing both the code space and the execution time for the routine.

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0

 LOC  OBJ        SEQ        SOURCE STATEMENT

                    1 $MACROFILE
                    2 $INCLUDE(:F1:MPY8.HED)
                  = 3 ;*****************************************************************************
                  = 4 ;*                                                                           *
                  = 5 ;*      MPY8X8                                                               *
                  = 6 ;*                                                                           *
                  = 7 ;*===========================================================================*
                  = 8 ;*                                                                           *
                  = 9 ;*      THIS UTILITY PROVIDES AN 8 BY 8 UNSIGNED MULTIPLY                    *
                  = 10 ;*     AT ENTRY:                                                            *
                  = 11 ;*       A = LOWER EIGHT BITS OF DESTINATION OPERAND                        *
                  = 12 ;*       XA= DON'T CARE                                                     *
                  = 13 ;*       R1= POINTER TO SOURCE OPERAND (MULTIPLIER) IN INTERNAL MEMEORY     *
```

Figure 5

```
LOC  OBJ        SEQ        SOURCE STATEMENT


                =  14 ;*                                                              *
                =  15 ;*      AT EXIT:                                                *
                =  16 ;*        A = LOWER EIGHT BITS OF RESULT                        *
                =  17 ;*        XA= UPPER EIGHT BITS OF RESULT                        *
                =  18 ;*        C = SET IF OVERFLOW ELSE CLEARED                      *
                =  19 ;*                                                              *
                =  20 ;*******************************************************************************
                   21 ;
                   22 ;
                   23 $INCLUDE(:F1:MPY8.PDL)
                =  24 ;1 MPY8X8:
                =  25 ;1 MULTIPLICAND[15-8]:=0
                =  26 ;1 COUNT:=8
                =  27 ;1 REPEAT
                =  28 ;2    IF MULTIPLICAND[0]=0 THEN BEGIN
                =  29 ;3       MULTIPLICAND:=MULTIPLICAND/2
                =  30 ;2    ELSE
                =  31 ;3       MULTIPLICAND[15-8]:=MULTIPLICAND[15-8]+MULTIPLIER
                =  32 ;3       MULTIPLICAND:=MULTIPLICAND/2
                =  33 ;2    ENDIF
                =  34 ;2    COUNT:=COUNT-1
                =  35 ;1 UNTIL COUNT=0
                =  36 ;1 END MPY8X8
                   37 ;
                   38 ; EQUATES
                   39 ; =======
                   40 ;
0002               41 XA      EQU     R2
0003               42 COUNT   EQU     R3
0004               43 ICNT    EQU     R4
                   44 ;
0003               45 DIGPR   EQU     3
                   46 ;
                   47 $EJECT
                   48 $INCLUDE(:F1:MPY8)
                =  49 ;1 MPY8X8:
                =  50 MPY8X8:
                =  51 ;1 MULTIPLICAND[15-8]:=0
0000 BA00       =  52          MOV     XA,#00
                =  53 ;1 COUNT:=8
0002 BB08       =  54          MOV     COUNT,#8
                =  55 ;1 REPEAT
                =  56 MPY8LP:
                =  57 ;2    IF MULTIPLICAND[0]=0 THEN BEGIN
0004 120E       =  58          JB0     MPY8A
                =  59 ;3       MULTIPLICAND:=MULTIPLICAND/2
0006 2A         =  60          XCH     A,XA
0007 97         =  61          CLR     C
0008 67         =  62          RRC     A
0009 2A         =  63          XCH     A,XA
000A 67         =  64          RRC     A
000B EB04       =  65          DJNZ    COUNT,MPY8LP
000D 83         =  66          RET
                =  67 ;2    ELSE
```

Figure 5 (continued)

```
LOC  OBJ          SEQ          SOURCE STATEMENT

                  =  68 MPY8A:
                  =  69 ;3       MULTIPLICAND[15-8]:=MULTIPLICAND[15-8]+MULTIPLIER
000E 2A           =  70          XCH     A,XA
000F 61           =  71          ADD     A,@R1
0010 67           =  72          RRC     A
0011 2A           =  73          XCH     A,XA
0012 67           =  74          RRC     A
0013 EB04         =  75          DJNZ    COUNT,MPY8LP
0015 83           =  76          RET
                  =  77 ;3       MULTIPLICAND:=MULTIPLICAND/2
                  =  78 ;2     ENDIF
                  =  79 ;2     COUNT:=COUNT-1
                  =  80 ;1 UNTIL COUNT=0
                  =  81 ;1 END MPY8X8
                     82  END

USER SYMBOLS
COUNT  0003    DIGPR  0003    ICNT   0004    MPY8A  000E    MPY8LP 0004    MPY8X8 0000    XA     0002

ASSEMBLY COMPLETE,  NO ERRORS
```


DIVIDE ALGORITHMS

In order to understand binary division a four bit operation will again be used as an example. The following algorithm will perform a four by four division:

```
DIVIDE:
  IF 16*DIVISOR> = DIVIDEND THEN
    SET OVERFLOW ERROR FLAG
  ELSE
    IF 8*DIVISOR> = DIVIDEND THEN
      QUOTIENT[3]: = 1
      DIVIDEND: = DIVIDEND – 8*DIVISOR
    ELSE
      QUOTIENT[3]: = 0
    ENDIF
    IF 4*DIVISOR> = DIVIDEND THEN
      QUOTIENT[2]: = 1
      DIVIDEND: = DIVIDEND – 4*DIVISOR
    ELSE
      QUOTIENT[2]: = 0
    ENDIF
    IF 2*DIVISOR> = DIVIDEND THEN
      QUOTIENT[1]: = 1
      DIVIDEND: = DIVIDEND – 2*DIVISOR
    ELSE
      QUOTIENT[1]: = 0
    ENDIF
    IF 1*DIVISOR> = DIVIDEND THEN
      QUOTIENT[0]: = 1
      DIVIDEND: = DIVIDEND – 1*DIVISOR
    ELSE
      QUOTIENT[0]: = 0
    ENDIF
  ENDIF
END DIVIDE
```

The algorithm is easy to understand. The first test asks if the division will fit into the dividend sixteen times. If it will, the quotient cannot be expressed in only four bits so an overflow error flag is set and the divide algorithm ends. The algorithm then proceeds to determine if eight times the divisor fits, four times, etc. After each test it either sets or clears the appropriate quotient bit and modifies the dividend. To see this algorithm in action, consider the division of 15 by 5:

```
  00001111     (15)
– 01010000     (16*5)
--------
               Doesn't fit—no overflow

  00001111     (15)
– 00101000     (8*5)
--------
               Doesn't fit—Q[3] = 0

  00001111     (15)
– 00010100     (4*5)
--------
               Doesn't fit—Q[2] = 0

  00001111     (15)
– 00001010     (2*5)
--------
  00000101     Fits—Q[1] = 1

  00000101     (15-2*5)
– 00000101     (1*5)
--------
  00000000     Fits—Q[0] = 1
```

The result is Q = 0011 which is the binary equivalent of 3—the correct answer. Clearly this algorithm can (and has been) converted to a loop and used to perform divisions. An examination of the procedure, however, will show that it has the same problems as the original multiply algorithm.

The first problem is that double precision operations are involved with both the comparison of the division with the dividend and the conditional subtraction. The second problem is that as the quotient bits are derived they must be shifted into a register. In order to reduce the register requirements, it would be desirable to shift them into the divisor register as they are generated since the divisor register gets shifted anyway. Unfortunately the quotient bits are derived most significant bits first so doing this will form a mirror image of the quotient—not very useful.

Both of these problems can be solved by observing that the algorithm presented for divide will still work if both sides of all the "equations" involving the dividend are divided by sixteen. The looping algorithm then would proceed as follows:

```
DIVIDE:
 QUOTIENT:=0
 COUNT:=4
 DIVIDEND:=DIVIDEND/16
 IF DIVISOR>=DIVIDEND THEN
  OVERFLOW FLAG:=1
 ELSE
  REPEAT
   DIVIDEND:=DIVIDEND*2
   QUOTIENT:=QUOTIENT*2
   IF DIVISOR>=DIVIDEND THEN
    QUOTIENT:=QUOTIENT+1/*SET QUOTIENT[0]*/
    DIVIDEND:=DIVIDEND-DIVISOR
   ENDIF
   COUNT:=COUNT-1
  UNTIL COUNT=0
 ENDIF
END DIVIDE
```

When this algorithm is implemented on a computer which does not have a direct compare instruction the comparison is done by subtraction and the inner loop of the algorithm is modified as follows:

```
*
*
REPEAT
  DIVIDEND:=DIVIDEND*2
  QUOTIENT:=QUOTIENT*2
  DIVIDEND:=DIVIDEND-DIVISOR
  IF BORROW=0 THEN
    QUOTIENT:=QUOTIENT+1
  ELSE
    DIVIDEND:=DIVIDEND+DIVISOR
  ENDIF
  COUNT:=COUNT-1
UNTIL COUNT=0
*
*
```

An implementation of this algorithm using the 8049 instruction set is shown in figure 6. This routine does an unsigned divide of a 16 bit quantity by an eight bit quantity. Since the multiply algorithm of figure 5 generates a 16 bit result from the multiplication of two eight bit operands, these two routines complement each other and can be used as part of more complex computations.

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0

 LOC  OBJ        SEQ        SOURCE STATEMENT

                    1 $MACROFILE
                    2 $INCLUDE(:F1:DIV16.HED)
                =   3 ;*******************************************************************************
                =   4 ;*                                                                             *
                =   5 ;*      DIV16                                                                  *
                =   6 ;*                                                                             *
                =   7 ;*=============================================================================*
                =   8 ;*                                                                             *
                =   9 ;*      THIS UTILITY PROVIDES AN 16 BY 8 UNSIGNED DIVIDE                       *
                =  10 ;*      AT ENTRY:                                                              *
                =  11 ;*        A = LOWER EIGHT BITS OF DESTINATION OPERAND                          *
                =  12 ;*        XA= UPPER EIGHT BITS OF DIVIDEND                                     *
                =  13 ;*        R1= POINTER TO DIVISOR IN INTERNAL MEMORY                            *
                =  14 ;*                                                                             *
                =  15 ;*      AT EXIT:                                                               *
                =  16 ;*        A = LOWER EIGHT BITS OF RESULT                                       *
                =  17 ;*        XA= REMAINDER                                                        *
```

Figure 6

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                =  18 ;*       C = SET IF OVERFLOW ELSE CLEARED                          *
                =  19 ;*                                                                 *
                =  20 ;*************************************************************************
                   21 ;
                   22 ;
                   23 $INCLUDE(:F1:DIV16.PDL)
                =  24 ;1 DIV16:
                =  25 ;1 COUNT:=8
                =  26 ;1 DIVIDEND[15-8]:=DIVIDEND[15-8]-DIVISOR
                =  27 ;1 IF BORROW=0 THEN /* IT FITS*/
                =  28 ;2    SET OVERFLOW FLAG
                =  29 ;1 ELSE
                =  30 ;2    RESTORE DIVIDEND
                =  31 ;2    REPEAT
                =  32 ;3       DIVIDEND:=DIVIDEND*2
                =  33 ;3       QUOTIENT.=QUOTIENT*2
                =  34 ;3       DIVIDEND[15-8]:=DIVIDEND[15-8]-DIVISOR
                =  35 ;3       IF BORROW=1 THEN
                =  36 ;4          RESTORE DIVIDEND
                =  37 ;3       ELSE
                =  38 ;4          QUOTIENT[0]:=1
                =  39 ;3       ENDIF
                =  40 ;3       COUNT:=COUNT-1
                =  41 ;2    UNTIL COUNT=0
                =  42 ;2    CLEAR OVERFLOW FLAG
                =  43 ;1 ENDIF
                =  44 ;1 ENDDIVIDE
                   45 ;
                   46 ; EQUATES
                   47 ; =======
                   48 ;
0002               49 XA      EQU     R2
0003               50 COUNT   EQU     R3
                   51 ;
                   52 $EJECT
                   53 $INCLUDE(:F1:DIV16)
                =  54 ;1 DIV16:
0000 2A         =  55 DIV16:  XCH     A,XA    ; ROUTINE WORKS MOSTLY WITH BITS 15-8
                =  56 ;1 COUNT:=8
0001 BB08       =  57         MOV     COUNT,#8
                =  58 ;1 DIVIDEND[15-8]:=DIVIDEND[15-8]-DIVISOR
0003 37         =  59         CPL     A
0004 61         =  60         ADD     A,@R1
0005 37         =  61         CPL     A
                =  62 ;1 IF BORROW=0 THEN /* IT FITS*/
0006 F60B       =  63         JC      DIVIA
                =  64 ;2    SET OVERFLOW FLAG
0008 A7         =  65         CPL     C
0009 0424       =  66         JMP     DIVIB
                =  67 ;1 ELSE
                =  68 DIVIA:
                =  69 ;2    RESTORE DIVIDEND
000B 61         =  70         ADD     A,@R1
                =  71 ;2    REPEAT
                =  72 DIVILP:
                =  73 ;3       DIVIDEND:=DIVIDEND*2
```

Figure 6 (continued)

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                =  74 ;3       QUOTIENT:=QUOTIENT*2
000C 97         =  75          CLR     C
000D 2A         =  76          XCH     A,XA
000E F7         =  77          RLC     A
000F 2A         =  78          XCH     A,XA
0010 F7         =  79          RLC     A
0011 E618       =  80          JNC     DIVIE
0013 37         =  81          CPL     A
0014 61         =  82          ADD     A,@R1
0015 37         =  83          CPL     A
0016 0420       =  84          JMP     DIVIC
                =  85 ;3       DIVIDEND[15-8]:=DIVIDEND[15-8]-DIVISOR
0018 37         =  86 DIVIE:   CPL     A
0019 61         =  87          ADD     A,@R1
001A 37         =  88          CPL     A
                =  89 ;3       IF BORROW=1 THEN
001B E620       =  90          JNC     DIVIC
                =  91 ;4           RESTORE DIVIDEND
001D 61         =  92          ADD     A,@R1
001E 0421       =  93          JMP     DIVID
                =  94 ;3       ELSE
                =  95 DIVIC:
                =  96 ;4           QUOTIENT[0]:=1
0020 1A         =  97          INC     XA
                =  98 ;3       ENDIF
                =  99 ;3       COUNT:=COUNT-1
                = 100 ;2   UNTIL COUNT=0
0021 EB0C       = 101 DIVID:   DJNZ    COUNT,DIVILP
                = 102 ;2   CLEAR OVERFLOW FLAG
0023 97         = 103          CLR     C
                = 104 ;1 ENDIF
                = 105 ;1 ENDDIVIDE
0024 2A         = 106 DIVIB:   XCH     A,XA
0025 83         = 107          RET
                  108  END

USER SYMBOLS
COUNT  0003    DIV16  0000    DIVIA  0008    DIVIB  0024    DIVIC  0020    DIVID  0021    DIVIE  0018    DIVILP 000C
XA     0002

ASSEMBLY COMPLETE,  NO ERRORS
```

Figure 6 (continued)

BINARY AND BCD CONVERSIONS

The conversion of a binary value to a BCD (binary coded decimal) number can be done with a very straightforward algorithm:

```
CONVERT_TO_BCD:
  BCDACCUM:= 0
  COUNT:= PRECISION
  REPEAT
    BIN:= BIN * 2
    BCD:= BCD * 2 + CARRY
    COUNT:= COUNT - 1
  UNTIL COUNT = 0
END CONVERT_TO_BCD
```

The variable ***BCDACCUM*** is a BCD string used to accumulate the result; the variable ***BIN*** is the binary number to be converted. ***PRECISION*** is a constant which gives the length, in binary bits of BIN. To see how this works, assume that BIN is a sixteen bit value with the most significant bit set. On the first pass through the loop the multiplication of ***BIN*** will result in a carry and this carry will be added to BCD. On the remaining passes through the loop BCD will be multiplied by two 15 times. The initial carry into BCD will be multiplied by 2^{15} or 32678, which is the "value" of the most significant bit of ***BIN***. The process repeats with each bit of ***BIN*** being introduced to ***BCDACCUM*** and then being scaled up on successive passes through the loop. Figure 7 shows the implementation of this algorithm for the 8049.

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0

 LOC  OBJ        SEQ        SOURCE STATEMENT

                    1 $MACROFILE
                    2 $INCLUDE(:F1:CONBCD.HED)
                =   3 ;*******************************************************************************
                =   4 ;*                                                                             *
                =   5 ;*      CONBCD                                                                 *
                =   6 ;*                                                                             *
                =   7 ;*=============================================================================*
                =   8 ;*                                                                             *
                =   9 ;*      THIS UTILITY CONVERTS A 16 BIT BINARY VALUE TO BCD                      *
                =  10 ;*      AT ENTRY:                                                              *
                =  11 ;*        A = LOWER EIGHT BITS OF BINARY VALUE                                 *
                =  12 ;*        XA= UPPER EIGHT BITS OF BINARY VALUE                                 *
                =  13 ;*        R0= POINTER TO A PACKED BCD STRING                                   *
                =  14 ;*                                                                             *
                =  15 ;*      AT EXIT:                                                               *
                =  16 ;*        A = UNDEFINED                                                        *
                =  17 ;*        XA= UNDEFINED                                                        *
                =  18 ;*        C = SET IF OVERFLOW ELSE CLEARED                                     *
                =  19 ;*                                                                             *
                =  20 ;*******************************************************************************
                   21 ;
                   22 ;
                   23 $INCLUDE(:F1:CONBCD.PDL)
                =  24 ;1 CONVERT_TO_BCD
                =  25 ;1 BCDACC:=0
                =  26 ;1 COUNT:=16
                =  27 ;1 REPEAT
                =  28 ;2    BIN:=BIN*2
                =  29 ;2    BCD:=BCD*2+CARRY
                =  30 ;2    IF CARRY FROM BCDACC GOTO ERROR EXIT
                =  31 ;2    COUNT:=COUNT-1
                =  32 ;1 UNTIL COUNT=0
                =  33 ;1 END CONVERT_TO_BCD
                   34 ;
                   35 ; EQUATES
                   36 ; =======
                   37 ;
0002               38 XA      EQU     R2
0003               39 COUNT   EQU     R3
0004               40 ICNT    EQU     R4
                   41 ;
0003               42 DIGPR   EQU     3
                   43 ;
                   44 $EJECT
                   45 $INCLUDE(:F1:CONBCD)
                =  46 ;
0005            =  47 TEMP1   SET     R5
                =  48 ;
                =  49 ;1 CONVERT_TO_BCD
                =  50 CNBCD:
                =  51 ;1 BCDACC:=0
0000 28         =  52         XCH     A,R0
```

Figure 7

```
LOC  OBJ          SEQ          SOURCE STATEMENT

0001 A9         =  53           MOV     R1,A
0002 28         =  54           XCH     A,R0
0003 BC03       =  55           MOV     ICNT,#DIGPR
0005 B100       =  56 BCDCOA:   MOV     @R1,#00
0007 19         =  57           INC     R1
0008 EC05       =  58           DJNZ    ICNT,BCDCOA
                =  59 ;1 COUNT:=16
000A BB10       =  60           MOV     COUNT,#16
                =  61 ;1 REPEAT
                =  62 BCDCOB:
                =  63 ;2    BIN:=BIN*2
000C 97         =  64           CLR     C
000D F7         =  65           RLC     A
000E 2A         =  66           XCH     A,XA
000F F7         =  67           RLC     A
0010 2A         =  68           XCH     A,XA
                =  69 ;2    BCD:=BCD*2+CARRY
0011 28         =  70           XCH     A,R0
0012 A9         =  71           MOV     R1,A
0013 28         =  72           XCH     A,R0
0014 BC03       =  73           MOV     ICNT,#DIGPR
0016 AD         =  74           MOV     TEMP1,A
0017 F1         =  75 BCDOC:    MOV     A,@R1
0018 71         =  76           ADDC    A,@R1
0019 57         =  77           DA      A
001A A1         =  78           MOV     @R1,A
001B 19         =  79           INC     R1
001C EC17       =  80           DJNZ    ICNT,BCDOC
001E FD         =  81           MOV     A,TEMP1
                =  82 ;2    IF CARRY FROM BCDACC GOTO ERROR EXIT
001F F624       =  83           JC      BCDCOD
                =  84 ;2    COUNT:=COUNT-1
                =  85 ;1 UNTIL COUNT=0
0021 EB0C       =  86           DJNZ    COUNT,BCDCOB
0023 97         =  87           CLR     C       ; CLEAR CARRY TO INDICATE NORMAL TERMINATION
                =  88 ;1 END CONVERT_TO_BCD
0024 83         =  89 BCDCOD:   RET
                   90  END

USER SYMBOLS
BCDCOA 0005   BCDCOB 000C   BCDCOD 0024   BCDOC  0017   CNBCD  0000   COUNT  0003   DIGPR  0003   ICNT   0004
TEMP1  0005   XA     0002

ASSEMBLY COMPLETE,  NO ERRORS
```

Figure 7 (continued)

The conversion of a BCD value to binary is essentially the same process as converting a binary value to BCD.

```
CONVERT_TO_BINARY
  BIN:=0
  COUNT:=DIGNO
  REPEAT
    BCDACCUM:=BCDACCUM * 10
    BIN:=10 * BIN+CARRY DIGIT
    COUNT:=COUNT-1
  UNTIL COUNT=0
END CONVERT_TO_BINARY
```

The only complexity is the two multiplications by ten. The BCDACCUM can be multiplied by ten by shifting it left four places (one digit). The variable BIN could be multiplied using the multiply algorithm already discussed, but it is usually more efficient to do this by making the following substitution:

$$BIN = 10 * BIN = (2) * (5) * (BIN) = 2 * (2 * 2 + 1) * BIN$$

This implies that the value 10 * BIN can be generated by saving the value of BIN and then shifting BIN two places left. After this the original value of BIN can be added to the new value of BIN (forming 5 * BIN) and then BIN can be multiplied by two. It is often possible to implement the multiplication of a value by a constant by using such techniques. Figure 8 shows an 8049 routine which converts BCD values to binary. This routine differs slightly from the algorithm above in that the BCD digits are read, and converted to binary, two digits at a time. Protection has also been added to detect BCD operands which, if converted, would yield binary values beyond the range of the result.

```
ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0


 LOC  OBJ        SEQ        SOURCE STATEMENT

                    1 $MACROFILE
                    2 $INCLUDE(:F1:CONBIN.HED)
                =   3 ;*******************************************************************************
                =   4 ;*                                                                             *
                =   5 ;*      CONBIN                                                                 *
                =   6 ;*                                                                             *
                =   7 ;*=============================================================================*
                =   8 ;*                                                                             *
                =   9 ;*      THIS UTILITY CONVERTS A 6 DIGIT BCD VALUE TO BINARY                     *
                =  10 ;*      AT ENTRY:                                                              *
                =  11 ;*        R0= POINTER TO A PACKED BCD STRING                                   *
                =  12 ;*                                                                             *
                =  13 ;*      AT EXIT:                                                               *
                =  14 ;*        A = LOWER EIGHT BITS OF THE BINARY RESULT                            *
                =  15 ;*        XR= UPPER EIGHT BITS OF THE BINARY RESULT                            *
                =  16 ;*        C = SET IF OVERFLOW ELSE CLEARED                                     *
                =  17 ;*                                                                             *
                =  18 ;*******************************************************************************
                   19 ;
                   20 ;
                   21 $INCLUDE(:F1:CONBIN.PDL)
                =  22 ;
                =  23 ;
                =  24 ;1 CONVERT_TO_BINARY
                =  25 ;1 POINTER0:=POINTER0+DIGITPAIR-1
                =  26 ;1 COUNT:=DIGITPAIR
                =  27 ;1 BIN:=0
                =  28 ;1 REPEAT
                =  29 ;2    BIN:=BIN*10
                =  30 ;2    BIN:=BIN+MEM(R0)[7-4]
                =  31 ;2    BIN:=BIN*10
                =  32 ;2    BIN:=BIN+MEM(R0)[3-0]
```

```
LOC  OBJ        SEQ        SOURCE STATEMENT

                =  33 ;2    POINTER0:=POINTER0-1
                =  34 ;2    COUNT:=COUNT-1
                =  35 ;1 UNTIL COUNT=0
                =  36 ;1 END CONVERT_TO_BINARY
                   37 ;
                   38 ; EQUATES
                   39 ; =======
                   40 ;
0002               41 XA      EQU     R2
0003               42 COUNT   EQU     R3
0004               43 ICNT    EQU     R4
                   44 ;
0003               45 DIGPR   EQU     3
                   46 ;
                   47 $EJECT
                   48 $INCLUDE(:F1:CONBIN)
                =  49 ;
0005            =  50 TEMP1   SET     R5
0006            =  51 TEMP2   SET     R6
                =  52 ;
                =  53 ;1 CONVERT_TO_BINARY
                =  54 CONBIN:
                =  55 ;1 POINTER0:=POINTER0+DIGITPAIR-1
0000 F8         =  56          MOV     A,R0
0001 0302       =  57          ADD     A,#DIGPR-1
0003 A8         =  58          MOV     R0,A
                =  59 ;1 COUNT:=DIGITPAIR
0004 BB03       =  60          MOV     COUNT,#DIGPR
                =  61 ;1 BIN:=0
0006 27         =  62          CLR     A
0007 AA         =  63          MOV     XA,A
                =  64 ;1 REPEAT
                =  65 CONBLP:
                =  66 ;2    BIN:=BIN*10
0008 142B       =  67            CALL    CONB10
000A F62A       =  68            JC      CONBER
                =  69 ;2    BIN:=BIN+MEM(R0)[7-4]
000C AD         =  70            MOV     TEMP1,A
000D F0         =  71            MOV     A,@R0
000E 47         =  72            SWAP    A
000F 530F       =  73            ANL     A,#0FH
0011 6D         =  74            ADD     A,TEMP1
0012 2A         =  75            XCH     A,XA
0013 1300       =  76            ADDC    A,#00
0015 2A         =  77            XCH     A,XA
0016 F62A       =  78            JC      CONBER
                =  79 ;2    BIN:=BIN*10
0018 142B       =  80            CALL    CONB10
001A F62A       =  81            JC      CONBER
                =  82 ;2    BIN:=BIN+MEM(R0)[3-0]
001C AD         =  83            MOV     TEMP1,A
001D F0         =  84            MOV     A,@R0
001E 530F       =  85            ANL     A,#0FH
0020 6D         =  86            ADD     A,TEMP1
0021 2A         =  87            XCH     A,XA
```

```
LOC  OBJ        SEQ         SOURCE STATEMENT

 0022 1300      =  88          ADDC    A,#00
 0024 2A        =  89          XCH     A,XA
 0025 F62A      =  90          JC      CONBER
                =  91 ;2     POINTER0:=POINTER0-1
 0027 C8        =  92          DEC     R0
                =  93 ;2     COUNT:=COUNT-1
                =  94 ;1 UNTIL COUNT=0
 0028 EB08      =  95          DJNZ    COUNT,CONBLP
                =  96 ;1 END CONVERT_TO_BINARY
 002A 83        =  97 CONBER:  RET
                =  98 $EJECT
                =  99  ;
                = 100  ;
                = 101  ;       UTILITY TO MULTIPLY BIN BY 10
                = 102  ;       CARRY WILL BE SET IF OVERFLOW OCCURS
                = 103  ;
 002B AD        = 104 CONB10:  MOV     TEMP1,A ; SAVE A
 002C 2A        = 105          XCH     A,XA    ; SAVE XA
 002D AE        = 106          MOV     TEMP2,A
 002E 2A        = 107          XCH     A,XA
                = 108  ;
 002F 97        = 109          CLR     C
 0030 F7        = 110          RLC     A       ; BIN:=BIN*2
 0031 2A        = 111          XCH     A,XA
 0032 F7        = 112          RLC     A
 0033 2A        = 113          XCH     A,XA
 0034 F646      = 114          JC      CONB1E  ; ERROR ON OVERFLOW
                = 115  ;
 0036 F7        = 116          RLC     A       ; BIN:=BIN*4
 0037 2A        = 117          XCH     A,XA
 0038 F7        = 118          RLC     A
 0039 2A        = 119          XCH     A,XA
 003A F646      = 120          JC      CONB1E  ; ERROR ON OVERFLOW
                = 121  ;
 003C 6D        = 122          ADD     A,TEMP1 ; BIN:=BIN*5
 003D 2A        = 123          XCH     A,XA
 003E 7E        = 124          ADDC    A,TEMP2
 003F 2A        = 125          XCH     A,XA
 0040 F646      = 126          JC      CONB1E  ; ERROR ON OVERFLOW
                = 127  ;
 0042 F7        = 128          RLC     A       ; BIN:=BIN*10
 0043 2A        = 129          XCH     A,XA
 0044 F7        = 130          RLC     A
 0045 2A        = 131          XCH     A,XA
                = 132  ;
 0046 83        = 133 CONB1E:  RET
                = 134
                = 135  ;
                  136  END

USER SYMBOLS
CONB10 002B    CONB1E 0046    CONBER 002A    CONBIN 0000    CONBLP 0008    COUNT  0003    DIGPR  0003    ICNT   0004
TEMP1  0005    TEMP2  0006    XA     0002

ASSEMBLY COMPLETE,  NO ERRORS
```

CONCLUSION

The design goals of the full duplex serial communications software were realized; if transmission and reception are occurring concurrently, only 42 percent of the real time available to the 8049 will be consumed by the serial link. This implies that an 8049 running full duplex serial I/O will still outperform earlier members of the family running without the serial I/O requirement. It is also possible to run this program in an 8048 or 8748 at 1200 baud with the same 42 percent CPU utilization.

The execution times for the other routines that have been discussed have been summarized in Table 1. All of these routines were written to maintain maximum useability rather than minimum code size or execution time. The resulting execution times and code size are therefore what the user can expect to see in a real application. The results that were obtained clearly show the efficiency and speed of the 8049. The equivalent times for the 8048 are also shown. It is clear that the 8049 represents a substantial performance advantage over the 8048. Considering, in most applications, that the 8048 is the highest performance microcomputer available to date, the performance advantage of the 8049 should allow the cost benefits of a single chip microcomputer to be realized in many applications which up until now have required too much "computer power" for a single chip approach.

	BYTES	EXECUTION TIME (MICROSECONDS) 8049	8048
MPY8	21	109	200
DIV 16	37	183 MIN 204 MAX	335 MIN 375 MAX
CONBCD	36	733	1348
CONBIN	70	388	713

Table 1. Program Performance

A High-Speed Emulator for Intel MCS-48® Microcomputers

Contents

I. PURPOSE AND SCOPE 5-114

II. THE HSE-49™ DEVELOPMENT TOOL 5-114

III. GENERAL HARDWARE OVERVIEW 5-114

IV. INTERPROCESSOR COMMUNICATION 5-116

V. HSE-49 COMMAND DESCRIPTION 5-117

VI. SYSTEM LIMITATIONS 5-120

VII. HARDWARE CONFIGURATIONS 5-121

APPENDIX A. SCHEMATIC DIAGRAMS 5-123

APPENDIX B. MONITOR LISTINGS 5-127

APPENDIX C. COMMAND SUMMARY 5-214

APPENDIX D. ERROR MESSAGES 5-214

I. PURPOSE AND SCOPE

This Application Note presents a description of the design and operation of a high-speed emulator for the Intel® MCS-48™ family of single chip microcomputers. The HSE-49™ emulator provides a simple and inexpensive means for executing and debugging 8049 programs which require the full 11-MHz operating speed of the part.

Section II of this Application Note describes some of the features of this development tool and how it may be used. Section III briefly discusses the hardware used to implement these features, while Section IV describes the manner in which program execution status is made available to the operator.

A detailed description of all of the operator commands is presented in Section V of this note, along with the modifiers and options which may be specified for each command. Known restrictions and limitations of the HSE-49 system are listed and explained in Section VI. Section VII shows how the basic circuit may be modified to provide options on memory organization, I/O configurations, etc.

Full schematics of the system hardware, as well as monitor software listings, are presented in Appendices A and B, respectively. A short summary of the command syntax is presented in Appendix C,. Appendix D explains the error message codes which may appear during use.

It is assumed that the reader is already familiar with the operation of the 8048 or 8049 microcomputers. Some knowledge of the 8048 architecture is needed to understand sections of the command and modifier descriptions. Most users will already have this background. Other readers are referred to the *MCS-48 Microcomputer User's Manual,* Intel publication number 9800270.

II. THE HSE-49 DEVELOPMENT TOOL

In essence, the HSE-49 emulator provides the user a means for executing an MCS-48 program located in external RAM rather than internal ROM or EPROM. This allows programs being debugged to be modified easily and quickly during the debug cycle. A user's program may be entered into system RAM either manually or via a serial link from a host computer such as an Intellec® Microcomputer Development System. Once loaded, the program can be modified using an on-board keyboard and display, and executed in real-time in a number of breakpoint modes. The internal state of the processor, including RAM, accumulator, timer/counter, and status register contents, can also be read and modified through the keyboard.

Breakpoint and debug facilities are extremely flexible. The following execution modes are provided.

- Programs may be run in full (11 MHz) real time;
- Programs may be single-stepped;
- In break mode, programs run in full real time until break occurs;
- Breaks may be triggered by either program or external data RAM accesses;
- Any number of breakpoints may be used in any combination;
- "Auto-Step" operation causes the current program counter and Accumulator contents to be printed on the display for a short time on every instruction cycle;
- "Auto-Break" provides the above display only when a break flag is encountered, with real time operation otherwise;
- While running in non-break mode, a TTL-level pulse is generated whenever a break flag is encountered. This signal may be used to trigger an oscilloscope or Logic Analyzer to assist in hardware and software debug.
- While running in any mode, the keyboard and display are "alive". Execution may be suspended or terminated by commands from the keyboard.

Intent of this Note

While the HSE-49 emulator can assist a new microcomputer user in becoming familiar with the 8048 and 8049 microcomputers, its inherent debug capabilities will also prove helpful to design engineers. The design could be used for new system development and verification or adapted for prototype production.

The main concern in designing the HSE-49 emulator was to keep the basic design simple, while maximizing the system's flexibility. The design allows the use of jumpers, hardware and software switches, etc. to allow the user to reconfigure the system according to the way he dedicates chip-select pins, I/O, etc. The emulator can be changed to fit each user's unique needs, rather than forcing the user to alter his needs to what is provided.

The primary intent of note is to provide the reader with the information needed to reconstruct and make full use of the HSE-49 emulator. Less emphasis is placed on describing how the hardware operates or how the commands are implemented. This information may be found in the schematic diagrams and software listings included in the Appendices.

III. GENERAL HARDWARE OVERVIEW

User Program Emulation

The actual emulation of the user's program is done using an 8039 microcomputer (IC29 on the schematics in Appendix A) executing a program stored in external RAM. The basic minimum configuration includes the 8039 microcomputer, an 8282 address latch (IC19), and 2K bytes of 2114 RAM to use for program development and real-time execution (ICs B1, C1, B2, and C2). Additional RAM may be added to allow the user to expand his program and data memory to 4K each. (If an 11-MHz crystal is used with the microcomputer, type 2114-3 RAMs must be used.)

System Supervision

A second microcomputer — another 8039 (IC25) with an 8282 address latch (IC16) and off-chip program memory in a 2716 EPROM (IC15) — is used to scan the on-board keyboard and display, interpret and implement commands, drive serial interfaces, etc. In general, the master processor is used to interface the execution processor's memory spaces with the outside world and control the operation of the execution processor. In this note the two processors will be abbreviated "MP" and "EP", respectively. Figure 1 shows how the two processors interrelate with the rest of the system.
system.

Keyboard/Display

The 33-key keyboard shown in Figure 2 includes a 16-key hexidecimal keypad and 17 special function keys for specifying commands and modifiers. Readers already familiar with the PROMPT-48™ debug tool for the 8048 will find that 25 of the HSE-49 emulator keys are identical in function and layout to the PROMPT-48 keyboard, and use the PROMPT-48 command syntax. The eight additional keys are used to generalize and augment the PROMPT-48 capabilities, as described in Section V.

The eight-character seven-segment display (DS1-DS8) is used for displaying addresses, data, and pseudo-alphanumeric messages. The display responses printed in Section V and throughout this note use a mix of upper and lower case letters to indicate what seven-segment patterns appear. An 8243 (IC9) and eight DIP packages (resistor packs, current buffers, etc.) are used for multiplexing the display and scanning the keyboard.

Breakpoint Detection

Breakpoints are specified and detected using a 2102A 1K × 8 RAM corresponding to each pair of 2114s (ICs A1

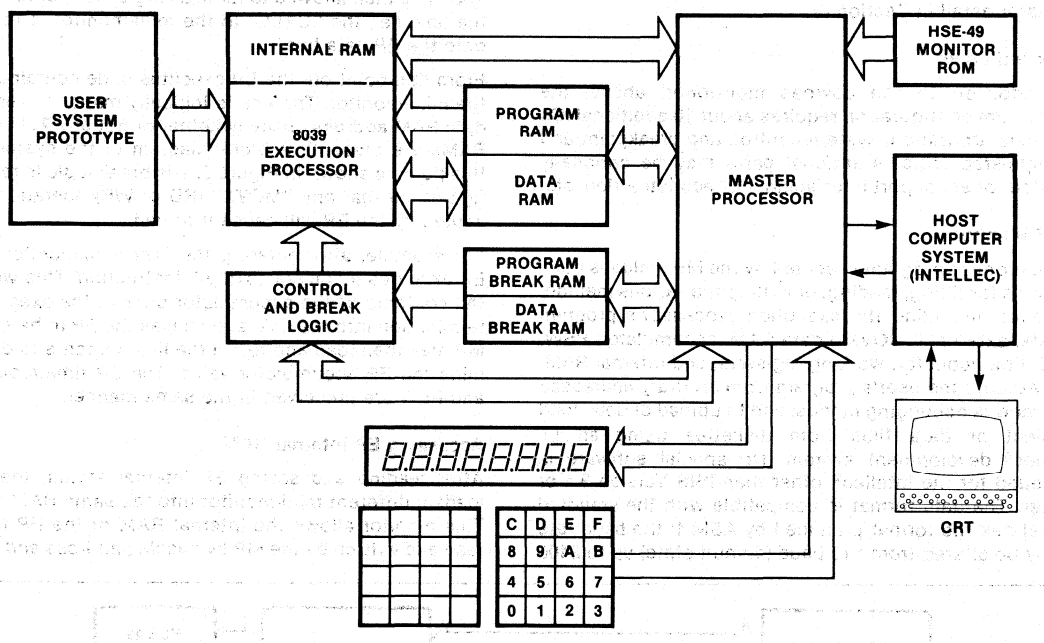


Figure 1. HSE-49™ Emulator Signal Flow Diagram

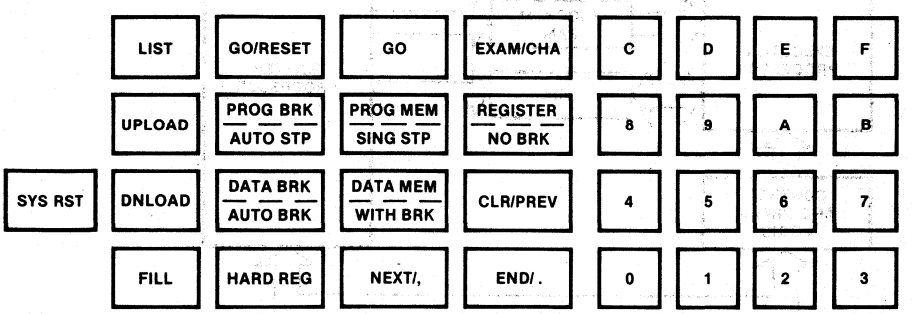


Figure 2. HSE-49™ Emulator Command Keyboard Organization

and A2). In effect, each program or data address accesses a 9-bit word. Eight bits are used normally for code or data storage. The ninth bit, accessed in parallel with the other eight, is used to indicate if a breakpoint has been set for that address. This output, when asserted, is latched (IC27 and IC36) and used to halt the execution processor via the single-step input. (In other modes, the break logic can be reconfigured to set the break requested flip-flop on any EP machine cycle or any EP "MOVX" instruction.)

Link Register

An 8212 8-bit latch (IC18) is used to communicate data and commands between the master and control processors. Under control of the MP, this register, called the "Link" register, may be logically mapped into either the program or data RAM address spaces. When this is done, the 2114s in the respective memory space are disabled and the link responds to all accesses, regardless of address. The link will be discussed in greater detail in Section IV.

Control Logic

In addition to the devices mentioned above, the minimum configuration requires about 10 additional ICs for bus arbitration, system control, and breakpoint and single-step logic. Additional parts may be optionally added for serial port interfacing, I/O reconstruction, etc.

MP Monitor

The monitor program executed by the MP includes commands for filling, reading, or writing the various memory spaces, including the execution processor's program RAM, external ("MOVX") data RAM, accumulator, PSW, PC, timer/counter, working registers, and internal RAM; to execute the user's program from arbitrary addresses in various debugging modes; and to upload or download object or data files from diskettes using an Intellec® development system. No special software is needed for the Intellec® other than ISIS Version 3.4 or later. The data format is compatible with the standard Intel hex file format produced by ASM-4; the baud rate may be altered from 110 baud (default state) up to 2400 baud from the on-board keybad. Blocks of data may be transmitted to a CRT or printer and displayed in a tabular format.

IV. INTERPROCESSOR COMMUNICATION

Program Break Sequence

When the MP detects that the EP has been halted by the breakpoint hardware, or when the operator presses a key while the program is executing, the program break sequence is initiated. The low-order 23 bytes of user program memory is read into a buffer within the internal RAM of the MP. A short program for reading and transmitting internal EP status is written over the low-order program memory. (This is one of several "mini-monitors" overlayed over the user program area.) The link register is mapped logically over the user program memory, and loaded with the 8049 machine code for a "CALL" instruction to the mini-monitor program area. The EP is then allowed to fetch a single instruction from the link, i.e., the "CALL" to the mini-monitor is forced onto the EP data bus.

From this point on, the EP executes code contained in the mini-monitor. The link is logically mapped over the data RAM address space (whether or not any 2114 data RAMs are present). A block diagram of the system at this point is shown in Figure 3. The break logic is reconfigured so that any "MOVX" (RD or WR) operation executed by the EP will cause it to halt.

For example, after entering the first mini-monitor, the EP executes a "MOVX @R0,A" instruction. This writes the contents of the accumulator prior to the execution termination into the link, and causes the EP to halt. The MP may then read and retain the link contents to determine the EP accumulator value. The EP timer/counter and PSW are preserved in the same manner.

Accessing EP Internal RAM

After reading and saving EP internal status, the MP loads a different mini-monitor into the same RAM area. This monitor allows the internal RAM of the EP to be read and written by the MP by passing address and data

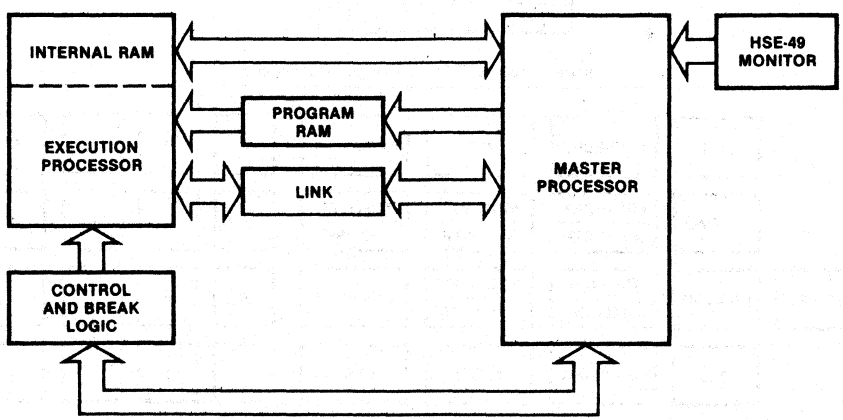


Figure 3. Communication between EP & MP

values between the two processors using the link register.

This is needed for two reasons. First, the EP program counter prior to the forced "CALL" instruction may be derived from the EP stack contents, and may be modified to cause the EP to resume execution at any desired address. Secondly, the internal RAM of the EP may then be accessed and modified in the process of executing a number of the monitor commands.

Resuming User Program Execution

In order to resume user program execution, a status-restoration mini-monitor is overlayed. This restores the EP internal status using a scheme analogous to the one in which the status was originally saved. The final step of the last mini-monitor is an "RETR" instruction, after which the EP is again halted. The low-order program memory saved earlier is rewritten into the appropriate area, the break logic is reconfigured for the desired execution mode, and the EP is released to run at full speed until the next break situation is encountered.

Note that all commands are implemented using "logical" rather than "physical" addressing. Thus the operator need not be concerned with the intricacies of the system design. For example, when any monitor command refers to low-order user program memory, the appropriate byte of storage within the MP internal RAM is accessed instead. If the location is altered, the internal RAM is modified appropriately. When program memory is reloaded prior to resuming user program execution, the modified version of the user program will be the one loaded.

Baud	HR06	HR07
110	93H	04H
150	96H	03H
300	45H	02H
600	9DH	01H
1200	44H	01H
2400	1AH	01H

Table 1. Serial Interface Data Rate Parameters

V. HSE-49 COMMAND DESCRIPTION

Whenever the characters "HSE-49" are present on the system display, a command string may be entered by the operator. In general, all command strings consist of a basic command initiator, an optional command modifier or type-designator, and a number of parameters or delimiters entered as hexidecimal digits. A command is executed, or a command in progress terminated, by pressing the [END/.] key. Logical default values are assumed for the modifier and parameters if either (or both) are omitted. A defualt parameter assumed for the command modifier will be presented on the display when the first parameter is entered.

Each parameter is a string of up to three hexidecimal digits. If more than three digits are entered, only the most recent three are considered. This allows an erroneous digit to be corrected without respecifying the entire command. A parameter is completed by pressing the [NEXT/,] key. Some commands may only need the low order part of a parameter; i.e., a command incorporating a data byte (such as [FILL]) will use only the low-order 8 bits of the corresponding parameter; Internal RAM and hardware register addressing uses only seven. In each case, higher order bits are ignored.

A command string is terminated and the command invoked by pressing the [END/.] key. The command will also be invoked by pressing the [NEXT/,] key when no additional parameters are allowed. A command string may be aborted at any point before the command is invoked by pressing the [CLEAR/PREV] key, and the sign-on message will appear.

Errors

An illegal command string, command terminator, or hardware failure will cause an error message and error code number to appear on the display (e.g., "Error-.3"). When this occurs, the monitor can be returned to command mode by pressing the [CLEAR] or [END/.] keys. An explanation of the various error codes is given in Appendix D.

Command Classes

Commands for the HSE-49 emulator are divided into general classes, where all commands in each class have the same choice of options or modifiers. A brief description of each command, followed by a description of the allowed options, is presented below by class.

Data Manipulation/Control Command Group

Commands:

[EXAM/CHA]

Display Response — "ECh."

Function — Examine/change memory location.

Causes the memory address specified to be read and presented on the display. New data may be entered (if desired) from the hexidecimal keypad. New data is verified before appearing on the display. Subsequent or previous locations may be read by pressing the [NEXT/,] or [PREV] keys, respectively. Command terminated with [END/.] key.

[FILL]

Display Response — "FIL."

Function — Fill range of memory addresses with a single data value.

Fill the appropriate memory space between the addresses specified by the first two parameters with the low-order byte of the third parameter. If second parameter less than first, only the location specified by the first is affected. If third parameter omitted, zero is assumed. If second and third parameters omitted, individual address specified is cleared. Command is useful for setting a large range of breakpoints; e.g., all of page 3 may be enabled for break with the command:

[FILL][PROG BRK]<300>[,]<3FF>[,]<1>[.]

[LIST]

Display Response — "LSt."

Function — List memory to output device through HSE-49 serial port.

Display the contents of a range of addresses given by two parameters to a teletype or CRT screen. Data is formatted, 16 separated bytes per line, with the starting address of each line printed. If used with an Intellec® system, the operator first uses ISIS-II to transfer the TTY input to the CRT output ("COPY :TI: TO :CO:") then invokes this command from the keypad. Alternatively, any ISIS device or disk file name(:TO:, :LP:, :F1:HRDREG.SAV, etc.) may be used as the destination.

[DNLOAD]

Display Response — "dnL."

Function — Download memory through HSE-49 serial port

Load data in hex file format through the serial input port. If used with Intellec® system, the operator first invokes this command from the keypad, then uses ISIS-II to transfer a disk file to the teletype port ("COPY : Fn:file.HEX TO :TO:").

The use of the checksum field for the download command is expanded slightly over the Intel hex file format standard. If the first character of the checksum field is a question mark ("?"), the checksum for that record will not be verified. This allows large object files produced by the assembler to be patched using the ISIS text editor without the necessity of manually recomputing the checksum value.

[UPLOAD]

Display Response — "UPL."

Function — Upload memory through HSE-49 serial port.

Output the contents of a range of addresses specified by the two parameters through the HSE-49 serial port in standard Intel hex file format. If used with Intellec® system, the operator first uses ISIS-II to transfer the TTY input to a disk file ("COPY :TI: TO :Fn:file.HEX"), then invokes this command from the keypad.

Data types allowed:

[PROG MEM]

Display Response — "Pr."

Function — User program memory.

Memory used to develop and execute user program. Addresses 000 through 7FF are the execution processor's memory bank 0; 800 through FFF are memory bank 1.

[REGISTER]

Display Response — "rG."

Function — Register memory and RAM.

Internal RAM of execution processor. Locations 0-7 are working register bank 0; 18-1F are working register bank 1. Only the low-order 7 bits of an address are considered.

[DATA MEM]

Display Response — "dA."

Function — External data memory (if installed).

Memory accessed by execution processor "MOVX A,@Rr" or "MOVX @Rr,A" instructions. High-order 4 bits may or may not be relevant, depending on jumpering option selected (explained in Section VII of this note).

[HARD REG]

Display Response — "Hr."

Function — Hardware registers.

The execution processor (EP) hardware registers (accumulator, timer/counter, etc.), as well as several parameters for controlling HSE-49 system status, are accessible through this catch-all memory space. Addresses are as follows:

00 — EP accumulator.

01 — EP PSW.
Bits correspond to 8049 PSW except that bit 3 (unused in the 8049) is used to monitor and alter the state of F1. Bits 2-0 correspond to the stack pointer value after the EP executes a CALL to the mini-monitor; i.e., one greater than when EP was running the user's program.

02 — EP timer/counter.

03 — EP internal RAM location 00.
(This value is also accessible through [REGISTER] space.)

04 — EP program counter (low byte).

05 — EP program counter (high nibble).

06-07 — HSE-49 serial interface baud rate parameters. Defaults to 110 baud; other rates may be selected by loading the values listed in Table 1.

08 — HSE-49 automatic sequencing rate parameter. Used in [GO][AUTO STP] and [GO][AUTO BRK] execution commands. 00 → fastest; FF → slowest. Defaults to 20H; approximately two steps per second.

09 — Monitor version/release number (packed BCD).

0A-0F — Currently unused by the monitor program.

10-7F — Variables used by master processor (MP) monitor. Should not be altered by operator.

[PROG BRK]

Display Response — "Pb."

Function — User program breakpoint memory.

Memory space used to indicate points where program execution should halt when running in a mode with breakpoints enabled ([GO][W/ BRK] and [GO][AUTOBRK]). Break will occur if enabled byte is read as the first or last byte of a 2-byte instruction, or read in executing a MOVP, MOVP3, or JMPP instruction. Memory is only 1 bit per location; 00 indicates continue, 01 causes a halt. Addresses 000 through 7FF are the execution processor's memory bank 0; 800 through FFF are memory bank 1.

[DATA BRK]

Display Response — "db."

Function — External data RAM breakpoint memory.

Memory space used to indicate points where data accesses should halt when running in a mode with breakpoints enabled ([GO][W/ BRK] and [GO][AUTOBRK]). Memory is only 1 bit per location; 00 indicates continue, 01 causes a halt. High-order 4 bits of breakpoint address may or may not be relevant, dependent on jumpering option selected for the corresponding data RAM (explained in Section VII of this note).

User Program Execution Control Group

Commands:

[GO]

Display Response — "Go."

Function — Begin execution.

If a parameter is given as part of the command string, execution will begin at that address. Otherwise, the EP program counter (hardware registers 04 and 05) will be used. These will contain the program counter from an earlier program execution break unless they have since been explicitly modified by the operator.

If command is terminated by [END/.], the EP's F1, PSW and stack pointer will be cleared. If command string is terminated by [NEXT/,], PSW will be taken from the EP PSW contents (hardware register 01).

While running the user's program, the characters "-run-." are written on the display. Execution may be halted and another command initiated by pressing the appropriate command key. Execution may be suspended at any time in any mode by pressing the [END/.] key. This will cause the current value of the execution processor program counter and accumulator to appear on the display in the form "PC.234-56". System status is saved in the appropriate hardware registers. At this point, or when an enabled breakpoint is encountered, pressing the [NEXT/,] key will cause the program to continue in the same mode as before. Any other command may be invoked by pressing the appropriate command string.

[GO/RESET]

Display Response — "Gr."

Function — Go from reset state.

EP is hardware-reset and released to execute the user's program from location 000H. No parameters are allowed. F0, F1, PSW, stack printer, memory bank flip-flop, etc., are cleared.

Note that this command does not require the use of mini-monitors to initiate program execution. As the last phase of the program development cycle, the 2114 program RAMs and address decoder may be removed and replaced by a ROM or EPROM part (not shown in schematics). This command may be used to start execution when the program RAM has been removed. No interrogation of EP status or internal RAM may be done, nor are break or single-step modes allowed in this case, though the 2102A breakpoint RAM outputs may still be used to trigger a logic analyzer.

Execution modes allowed:

[NO BRK]

Display Response — "nb."

Function — Without breakpoints.

Full-speed execution without breakpoints enabled. Does not affect the state of the breakpoint memories.

[SING STP]

Display Response — "SSt."

Function — Single Step.

Step through program one instruction at a time. After each instruction is executed, execution halts with the current value of the Execution Processor Program Counter and Accumulator appearing on the display in the form "PC.234-56". System status is saved in the appropriate Hardware Registers. At the point, [NEXT/,] will cause the program to execute one more instruction, or any other command may be invoked by pressing the appropriate command string. Does not affect the state of the Breakpoint Memories.

[W/ BRK]

Display Response — "br."

Function — With breakpoints.

Full-speed execution with breakpoints enabled. When a breakpoint is encountered, execution halts with the current value of the execution processor program counter and accumulator appearing on the display in the form "PC.234-56". System status is saved in the appropriate hardware registers. At this point, [NEXT.,] will cause the program to continue until the next breakpoint is reached, or any other command may be invoked by pressing the appropriate command string.

[AUTO STP]

Display Response — "ASt."

Function — Automatically sequence through a series of instructions.

Step through program one instruction at a time. After each instruction is executed, execution halts with the current value of the execution processor program counter and accumulator appearing on the display in the form "PC.234-56". System status is saved in the appropriate hardware registers. Execution resumes after a time determined by contents of hardware register 08. Does not affect the state of the breakpoint memories.

[AUTO BRK]

Display Response — "Abr."

Function — Automatically sequence between breakpoints.

Execute a series of instructions in real time between breakpoints. When breakpoint is encountered, halt EP temporarily while program counter and accumulator contents are displayed, then continue. Display is sustained after execution resumes. Does not affect the state of the breakpoint memories.

Breakpoint Control Command Group

Commands:

[B]

Display Response — "Stb."

Function — Breakpoint set.

Set breakpoint for the address given. Multiple breakpoints may be set by entering additional addresses, separated by the [NEXT/,] key. Command terminated by pressing [END/.]. Action taken is to fill the appropriate breakpoint memory locations with logical ones.

[C]

Display Response — "CLb."

Function — Clear breakpoint.

Clear breakpoint for the address given. Multiple breakpoints may be cleared by entering additional addresses, separated by the [NEXT/,] key. Command terminated by pressing [END/.]. Action taken is to fill the appropriate breakpoint memory locations with logical zeroes.

Data types allowed:

[PROG MEM]

Display Response — "Pr."

Function — Break on program memory fetch.

Applies command to the program breakpoint memory space.

[DATA MEM]

Display Response — "dA."

Function — Break on data memory access.

Applies command to the external data breakpoint memory space.

System Control Command Group

Command:

[SYS RST]

Display Response — "HSE-49."

Function — System reset.

Reset both the MP and EP and clear all breakpoints (requires approximately one second). CAUTION — If reset while EP is executing the user's program, the low order section of program memory (about 23 bytes) will be altered.

VI. SYSTEM LIMITATIONS

In designing the HSE-49 emulator, certain compromises were made in an attempt to maximize the usefulness of the emulator while keeping the circuitry simple and inexpensive. As a result, the following limitations exist and must be taken into account when using the system.

1. As explained in Section IV, user program execution is terminated (by single-stepping, breakpoints, pressing the [END/.] key, etc.) by forcing the execution processor to execute a "CALL" instruction to the mini-monitor. This uses one level of the EP subroutine stack. The EP PSW reflects the value of the stack pointer *after* processing this CALL. As a result, the value indicated for stack depth by examining the EP PSW (hardware register 01) is one greater than the depth when the break was initiated. The user program must not be using all eight levels of stack when a break is initiated or the bottom level will be destroyed.
2. User program is initiated (by the [GO] command or when resuming execution after a breakpoint, single-stepping, etc.) by forcing the EP to execute an "RETR" instruction. This will clear the EP interrupt-in-progress flip-flop. If the user program allows both external and timer interrupts to be enabled at the same time, care must be taken to avoid causing a break while the EP is within an interrupt servicing routine. No limitation is placed on breakpoints or single-stepping in the background program because of this.
3. When the user program execution is terminated (by a break, single-stepping, etc.) and later resumed, the EP timer/counter is restored to its value when the break occurred (unless modified by the user). The prescaler, however, will have changed. Thus, up to 31 machine cycles may be "lost" or "gained" if a break occurs while the timer is running.
4. Timer interrupts occurring at the same time as an EP break may be ignored if the timer overflow occurs after breaking user program execution before the timer value is saved.
5. The 8049 "RET" and "RETR" instructions are each 1-byte, 2-cycle instructions. During the second cycle the byte following the return instruction is fetched and ignored. If a program breakpoint is set for a location following a "RET" or "RETR" instruction, a break will be initiated when the return is executed.

6. Breakpoints should not be placed in the last 3 bytes of an EP memory bank (locations 7FDH-7FFH and 0FFDH-0FFFH). User program should not be single-stepped or auto-stepped through these locations.
7. Since I/O configuration is determined by external hardware rather than software, I/O modes may not be altered while a program is executing. (See Section VII for further details.)
8. The "ANL BUS,#nn" and "ORL BUS,#nn" instructions may not be used in the user program, as external hardware cannot properly restore these functions.
9. The memory bank select flag is not affected by the user program break sequence. Upon resuming execution with the [GO] command this flag will remain in the same state as before the preceding break. The flag may be cleared only by executing the [GO/RESET] or [SYS RST] commands.

VII. HARDWARE CONFIGURATIONS

A number of control and status lines are available to the user. All are low-power Schottky TTL-compatible signals.

TP1 — Unused MP input.

TP2 — Unused MP output.

TP3 — User program suspended. Low when EP running user code. High when halted or running mini-monitors.

TP4 — Breakpoint encountered. Normally low. High-level pulse generated when breakpoint passed. Useful for triggering logic analyzers, oscilloscopes, etc.

TP5 & TP6 — Memory matrix mode control. Select program vs. data RAM, link mapping configuration, etc. (See Appendix B for details.)

TP7 — Bus control. Low when MP controls common memory buses. High when EP controls memory buses.

The HSE-49 emulator hardware is designed to allow the user to reconfigure the system for a wide variety of different applications by installing or removing jumper wires or additional components. The schematics in Appendix A show the components needed for a variety of different configurations. In general, not all of the devices are required (or allowed) for any one configuration. The devices which are required are included in the following description.

The types of options allowed are divided below into several general classes and subdivided into mutually-independent features. Within some of these features there are numbered, mutually exclusive configurations; i.e., the serial interface (if desired) may use either current-loop or RS-232C current buffers, but not both at one time.

Standard Operating Configuration

(Minimum system configurations — up to 4K program RAM; no data RAM; no serial interfaces; no execution processor I/O reconstruction.)

A. Basic 2K monitor from Appendix B:

Install resistors R4-R6
Install transistor Q1
Install crystals Y1-Y2
Install capacitors C5-C38
Install switches S1-S33
Install displays DS1-DS8
Install IC1-IC2
Install RP3-RP5
Install IC6-IC7
Install RP8
Install IC9
Install IC15-IC20
Install IC25-IC30
Install IC34
Install IC36-IC38
Install A1-A2
Install B1-B2
Install C1-C3
Install jumpers 13-15
Install jumpers 17-18
Install jumper 20

B. Expansion 2K monitor:

Install IC14
Remove jumper 17

Serial Interface Buffer Selection

A. Current loop serial interfaces (4N46s) installed for use with full Intellec® Model 800 development system TTY port.

Install IC21-IC22
Install resistor R1-R3
Install jumpers 4-9
(Remove RS-232 jumpers)

B. RS-232C serial interfaces (MC1488 and MC1489) installed for use with CRT as output device for data dumps:

Install IC23-IC24
Install jumpers 1-3
Install jumpers 10-11
(Remove current-loop jumpers)

External Data RAM Address Decoding Scheme for Execution Processor

A. Up to 16 pages of on-board external data RAM installed for execution processor (addresses 0 through

0FFFH = 4K bytes); port 2 used for addressing pages 0 through 15:

Install jumpers 21-25
Install jumper 27
Install A5-A8
Install B5-B8
Install C5-C8

B. One page of on-board external data RAM installed for execution processor (addresses 0 through 0FFH); port 2 not used for data addressing:

Install jumper 26
Install jumper 28
Install A5
Install B5
Install C5

Connect the outputs of IC20, pins 7, 9, 10, & 11 to the inputs of a 74LS21 AND gate (not shown). Connect the output to CE and CS inputs of A5-C5. (Note: these signals are all present at jumpers 21-24 on the schematics.)

Reconstructing I/O for Execution Processor

A. Application of port 2, pins P23-P20:

(1) Using P23-P20 for latched output data (used with "OUTL P2,A", "ANL P2,#data", and "ORL P2,#data" instructions):

Install IC31

(2) Using P23-P20 for interfacing to an 8243 in user's prototype:

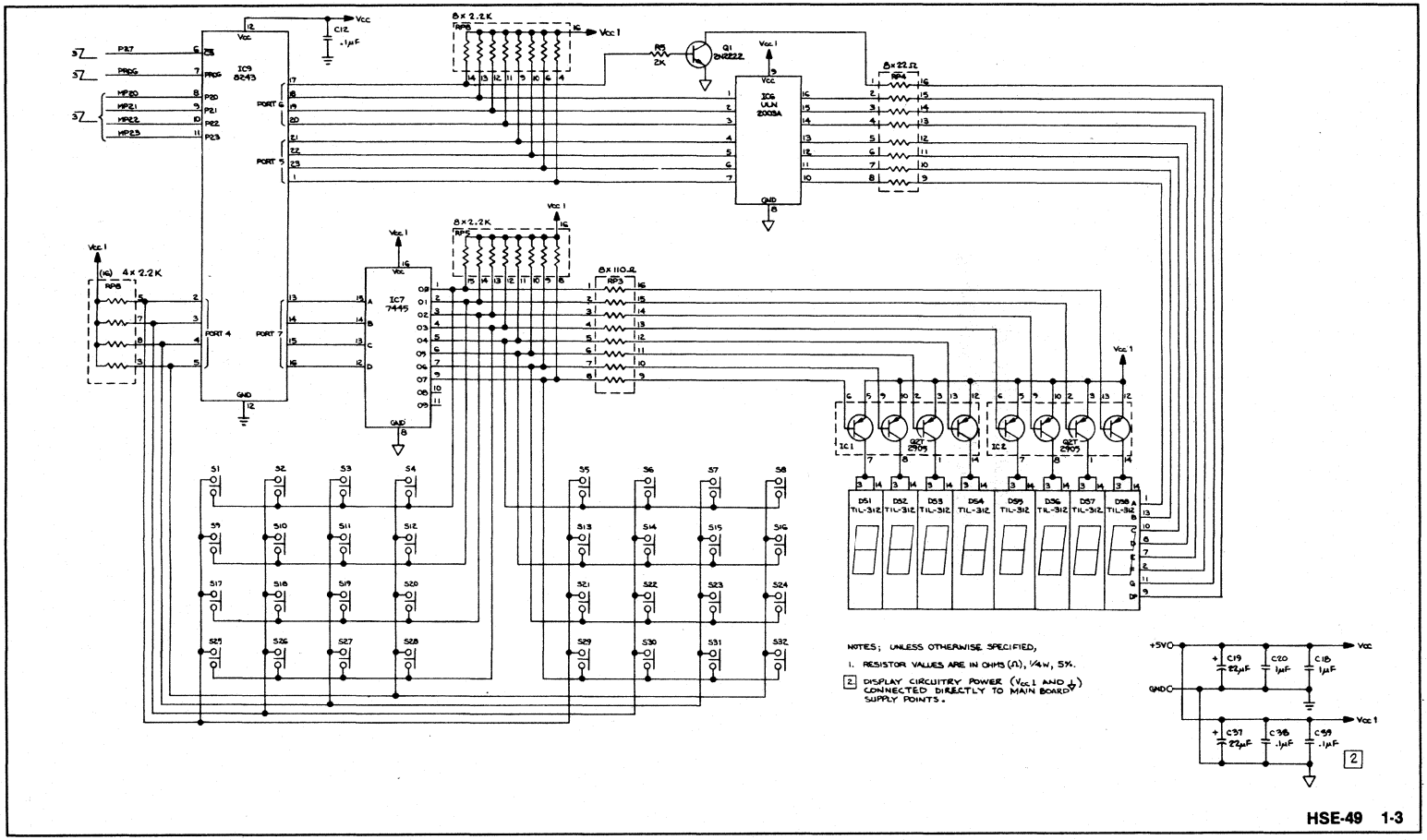
Connect D3-D0 pins on IC31 socket to corresponding Q3-Q0 pins.

B. Application of execution processor BUS:

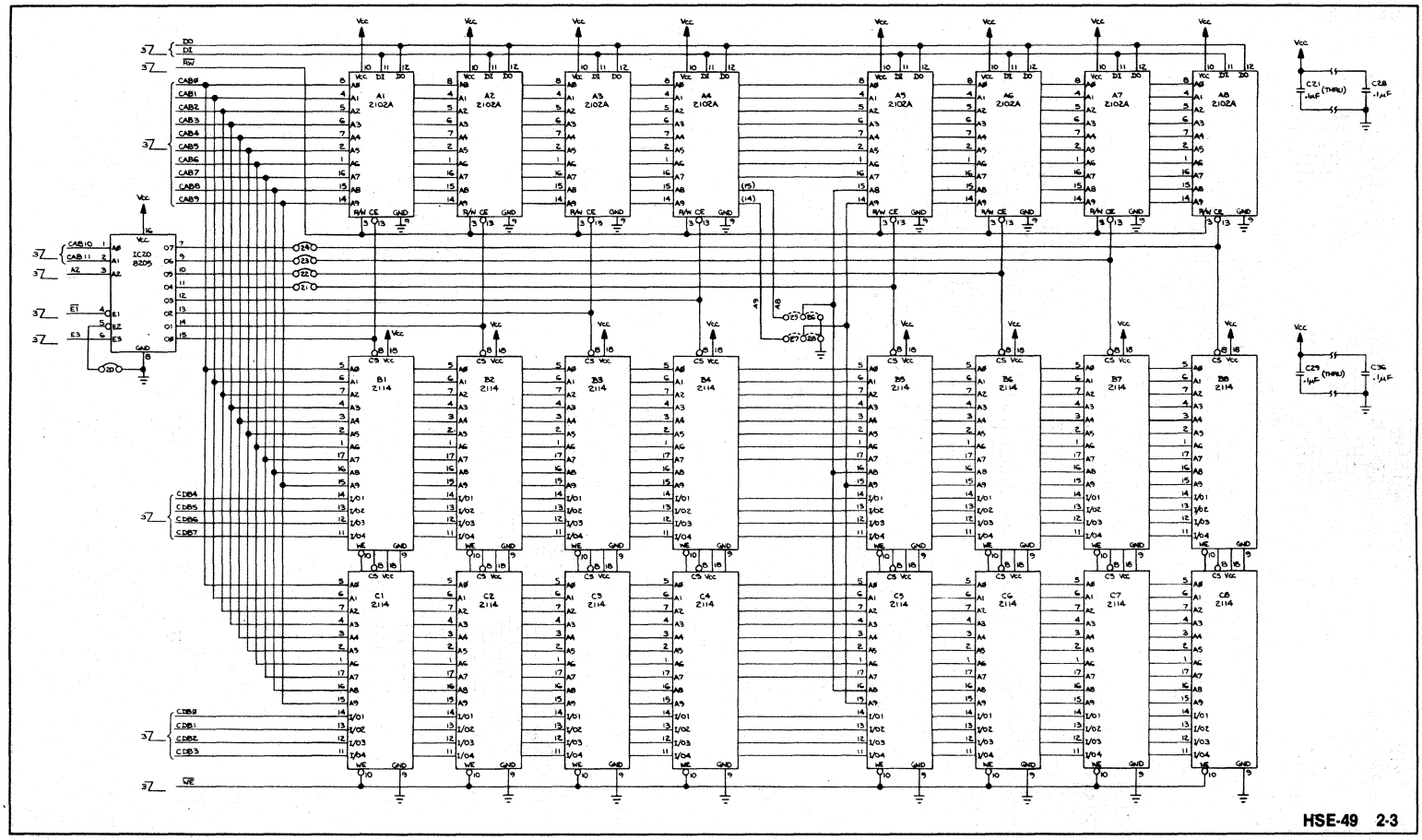
(1) Use of BUS as latched output port ("OUTL BUS,A"):

Install IC32

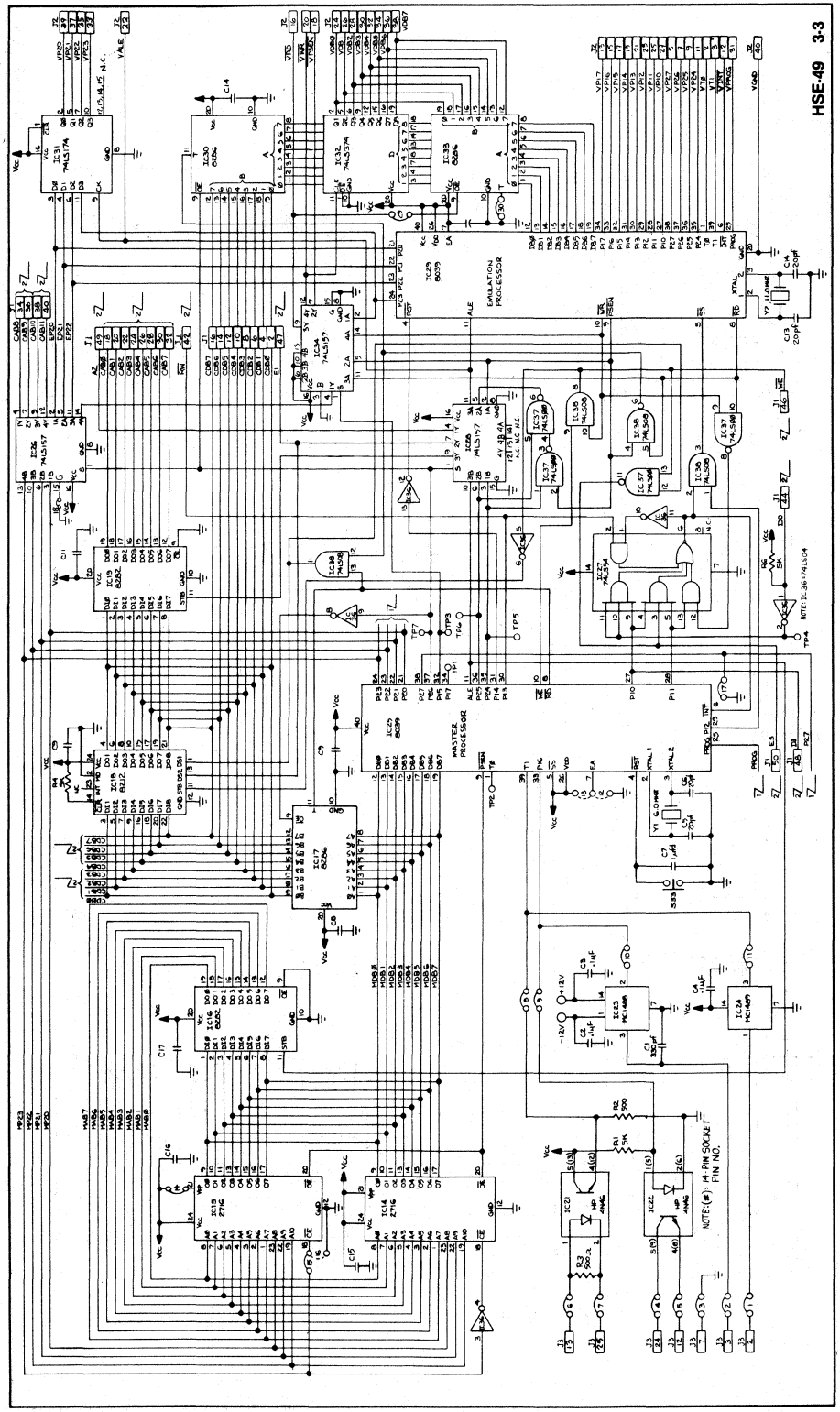
Appendix A
Schematic Diagrams



Key Board Display



Ram Memory



Central Processor

Appendix B
Monitor Listings

```
ASM48 HSE49.LNK PRINT(:LP:)


ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V3.0            PAGE    1
HSE-49(TM) EMULATOR MONITOR VERSION 2.5

  LOC  OBJ         LINE        SOURCE STATEMENT

                     1 $MACROFILE NOGEN NOCOND XREF
                     2 $TITLE('HSE-49(TM) EMULATOR MONITOR VERSION 2.5')
                     3 ;
                     4 ;*******************************************************************************
                     5 ;
                     6 ;                       PROGRAM: HSE-49(TM) EMULATOR MONITOR
                     7 ;                               VERS 2.5/709
                     8 ;
                     9 ;                       COPYRIGHT (C) 1979
                    10 ;                       INTEL CORPORATION
                    11 ;                       3065 BOWERS AVENUE
                    12 ;                       SANTA CLARA, CALIFORNIA 95051
                    13 ;
                    14 ;*******************************************************************************
                    15 ;
                    16 ;  ABSTRACT
                    17 ;  ========
                    18 ;
                    19 ; THIS PROGRAM CONTAINS THE SOFTWARE NECESSARY TO RUN THE HSE-49(TM)
                    20 ; HIGH-SPEED EMULATOR FOR INTEL'S MCS-48(TM) FAMILY FAMILY OF MICROCOMPUTERS.
                    21 ; THE EMULATOR PROVIDES AN ASSORTMENT OF UTILITY FUNCTIONS FOR
                    22 ; DEVELOPING AND DEBUGGING 8049-BASED APPLICATIONS, INCLUDING THE
                    23 ; ABILITY TO ENTER AND MODIFY PROGRAMS IN PROGRAM RAM,
                    24 ; ALTER DATA, SINGLE-STEP SECTIONS OF A PROGRAM, AND EXECUTE PROGRAMS
                    25 ; AT SPEEDS OF UP TO 11 MHZ. WITH OR WITHOUT BREAKPOINTS ENABLED.
                    26 ; THE EMULATOR IS DESCRIBED IN GREATER DEPTH IN INTEL'S APPLICATION NOTE
                    27 ; AP-55, "A HIGH-SPEED EMULATOR FOR INTEL MCS-48(TM) MICROCOMPUTERS."
                    28 ;
                    29 ;  PROGRAM ORGANIZATION
                    30 ;  ======= ============
                    31 ;
                    32 ; THIS LISTING IS ORGANIZED AS FOLLOWS:
                    33 ;
                    34 ;       INTRODUCTION AND HARDWARE OVERVIEW;
                    35 ;       VARIABLE DECLARATION AND DEFINITION;
                    36 ;       POWER-ON SYSTEM INITIALIZATION;
                    37 ;       KEYBOARD COMMAND PARSER AND ASSOCIATED TABLES;
                    38 ;       IMPLEMENTATIONS OF THE PRIMARY COMMANDS;
                    39 ;       DATA ACCESSING UTILITY SUBROUTINES USED THROUGHOUT;
                    40 ;       KEYBOARD SCANNING AND DISPLAY DRIVING SUBROUTINE;
                    41 ;       KEYBOARD AND DISPLAY INTERFACING UTILITIES;
                    42 ;       ROUTINES AND UTILITY SUBROUTINES WHICH INTERACT BETWEEN MP AND EP.
                    43 ;
                    44 ;
                    45 $EJECT
```

```
LOC  OBJ        LINE       SOURCE STATEMENT

                  46 ;
                  47 ;  INTRODUCTION AND HARDWARE OVERVIEW
                  48 ;  ============ === ======== ========
                  49 ;
                  50 ; THE EMULATOR DESIGN USES TWO MICROPROCESSORS.  ONE PROCESSOR CONTROLS
                  51 ; SYSTEM STATUS, INTERPRETS MONITOR COMMANDS, AND COMMUNICATES
                  52 ; WITH THE OUTSIDE WORLD THROUGH THE ON-BOARD KEYBOARD, DISPLAY, SERIAL
                  53 ; INTERFACES, CONTROL SIGNALS, ETC.
                  54 ; A SECOND PROCESSOR IS USED TO ACTUALLY
                  55 ; EXECUTE THE USER'S PROGRAM UNDER THE CONTROL OF THE FIRST.
                  56 ; THESE PROCESSORS ARE REFERRED TO
                  57 ; THROUGHOUT THIS PROGRAM AS THE MASTER PROCESSOR (MP) AND EXECUTION
                  58 ; PROCESSOR (EP) RESPECTIVELY.
                  59 ;
                  60 ; THE PROGRAM IN THIS LISTING IS EXECUTED BY THE MASTER PROCESSOR.
                  61 ; AT THE END OF THIS LISTING ARE SEVERAL SHORT "MINI-MONITOR OVERLAYS"
                  62 ; WHICH THE EXECUTION PROCESSOR EXECUTES WHEN INTERACTION BETWEEN THE
                  63 ; TWO PROCESSORS IS NECESSARY.
                  64 ;
                  65 ; THIS PROGRAM WAS WRITTEN USING A NUMBER OF MACROS TO HANDLE THE ALLOCATION
                  66 ; OF MPU RESOURCES (WORKING REGISTERS, INTERNAL RAM, AND MP MONITOR ROM
                  67 ; FOR CODE AND DATA STORAGE).  THESE MACRO DEFINITIONS ARE INCLUDED IN A FILE
                  68 ; NAMED "ALLOC.MAC," AND ARE PRINTED IN THIS LISTING FOR REFERENCE.
                  69 ; ANOTHER SET OF MACROS IS USED TO SIMPLIFY THE ACCESSING OF VARIABLES
                  70 ; STORED IN INTERNAL RAM (AS OPPOSED TO WORKING REGISTERS) BY USING R1 TO
                  71 ; INDIRECTLY ADDRESS THE APPROPRIATE RAM LOCATION WHEN NECESSARY.
                  72 ; THESE MACROS ARE INCLUDED IN "MOPCOD.MAC", AND ARE ALSO PRINTED HERE.
                  73 ; COMPLETE UNDERSTANDING OF THESE MACROS IS NOT REQUIRED TO UNDERSTAND THE
                  74 ; MONITOR PROPER:  ALL LINES WHICH ACTUALLY PRODUCE OBJECT CODE APPEAR IN
                  75 ; THE LISTING ITSELF, INDENTED TWO SPACES FROM THE NORMAL TABULATION COLUMNS.
                  76 ; THE ACTUAL MONITOR PROGRAM FOR THE EMULATOR BEGINS AT APPROXIMATELY
                  77 ; SOURCE LINE NUMBER 500.
                  78 ;
                  79 ; LINES GENERATED BY MACRO EXPANSION ARE FLAGGED BY A PLUS SIGN ("+")
                  80 ; IMMEDIATELY FOLLOWING THE SOURCE LINE NUMBER.
                  81 ; A NUMBER OF LINES FROM THE VARIOUS MACRO DEFINITIONS WHICH DO NOT
                  82 ; PRODUCE ANY OBJECT CODE ARE PROCESSED BY THE ASSEMBLER
                  83 ; AS THESE MACROS ARE EXPANDED.  WHEN THIS IS THE CASE, THESE LINES ARE
                  84 ; SUPPRESSED FROM THE LIST FILE.  AS A RESULT, THE LINE NUMBERS ARE
                  85 ; NOT ALWAYS CONSECUTIVE WHERE A MACRO IS BEING INVOKED.
                  86 ;
                  87 ; NOTE:
                  88 ; ====
                  89 ; "SOURCE-LINE" REFERS TO THE DECIMAL NUMBERS LEFT OF EACH INSTRUCTION.
                  90 ; AT THE END OF THE LISTING IS AN ASSEMBLY CROSS-REFERENCE TABLE INDICATING
                  91 ; THE SEQUENTIAL SOURCE-LINE NUMBER OF ALL INSTANCES WHERE ANY VARIABLE
                  92 ; IS DEFINED OR REFERENCED.  THIS WILL BE OF GREAT ASSISTANCE IN
                  93 ; LOCATING SPECIFIC SUBROUTINES, ETC.  IN THE LISTING.
                  94 ;
                  95 ; MNEMONICS COPYRIGHT (C) 1976 INTEL CORPORATION
                  96 ;
                  97 $EJECT
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

                  98 $       INCLUDE(:F0:ALLOC.MAC)
0000            =  99 ?R1     SET     0
                = 100 ;
0000            = 101 ?RB0    EQU     0
0001            = 102 ?RB1    EQU     1
0002            = 103 ?RAM    EQU     2
0003            = 104 ?CONST  EQU     3
0004            = 105 ?R      EQU     4       ;ACCUMULATOR VARIABLE TYPE
                = 106 ;
                = 107 ;THE FOLLOWING INITIALIZES THE LINKED LIST POINTERS FOR
                = 108 ;THE REGISTER ALLOCATION AND DEALLOCATION ROUTINES.
                = 109 ;
0003            = 110 ?B0R2   SET     3
0004            = 111 ?B0R3   SET     4
0005            = 112 ?B0R4   SET     5
0006            = 113 ?B0R5   SET     6
0007            = 114 ?B0R6   SET     7
0008            = 115 ?B0R7   SET     8
                = 116 ;
0002            = 117 ?B0PNT  SET     2
                = 118 ;
0003            = 119 ?B1R2   SET     3
0004            = 120 ?B1R3   SET     4
0005            = 121 ?B1R4   SET     5
0006            = 122 ?B1R5   SET     6
0007            = 123 ?B1R6   SET     7
0008            = 124 ?B1R7   SET     8
                = 125 ;
0002            = 126 ?B1PNT  SET     2
                = 127 ;
0000            = 128 ORGPG0  SET     000H
0100            = 129 ORGPG1  SET     100H
0200            = 130 ORGPG2  SET     200H
0300            = 131 ORGPG3  SET     300H
0400            = 132 ORGPG4  SET     400H
0500            = 133 ORGPG5  SET     500H
0600            = 134 ORGPG6  SET     600H
0700            = 135 ORGPG7  SET     700H
                = 136 ;
                = 137 $EJECT
```

```
LOC  OBJ          LINE        SOURCE STATEMENT

                  = 138 ;*****************************************************************
                  = 139 ;
                  = 140 ;       START OF ALLOCATION MACROS
                  = 141 ;
                  = 142 ;*****************************************************************
                  = 143 ;
                  = 144 ?RSAVE  MACRO   SYMBOL,BANK,PNTVAL
-                 = 145 IF      PNTVAL EQ 8
--                = 146         ERROR   2
-                 = 147         EXITM
-                 = 148 ENDIF
--                = 149 $       SAVE GEN
-                 = 150         SYMBOL  SET  R&PNTVAL
--                = 151 $       RESTORE
-                 = 152 ?B&BANK&PNT     SET     ?B&BANK&R&PNTVAL
                  = 153         ENDM
                  = 154 ;
                  = 155 ;
0020              = 156 ?MINDX  SET     20H
                  = 157 ;
                  = 158 ?MSAVE  MACRO   SYMBOL,LENGTH,ADDR
                  = 159 $       SAVE GEN
--                = 160         SYMBOL  EQU  ADDR
-                 = 161 $       RESTORE
-                 = 162 ?MINDX  SET     ?MINDX+LENGTH
                  = 163 ENDM
                  = 164 ;
                  = 165 MBLOCK  MACRO   SYMBOL,LENGTH
-                 = 166 ?&SYMBOL        EQU   3
--                = 167         ?MSAVE  SYMBOL,LENGTH,%?MINDX
                  = 168 ENDM
                  = 169 ;
                  = 170 DECLARE MACRO   SYMBOL,TYPE
-                 = 171 ?&SYMBOL        SET     ?&TYPE
-                 = 172 IF      ?&TYPE EQ 2
-                 = 173         ?MSAVE  SYMBOL,1,%?MINDX
--                = 174         EXITM
-                 = 175 ENDIF
--                = 176 IF      ?&TYPE EQ 0
-                 = 177         ?RSAVE  SYMBOL,0,%?B0PNT
-                 = 178         EXITM
-                 = 179 ENDIF
-                 = 180 IF      ?&TYPE EQ 1
--                = 181         ?RSAVE  SYMBOL,1,%?B1PNT
-                 = 182         EXITM
--                = 183 ENDIF
                  = 184         ENDM
                  = 185 ;
                  = 186 $       EJECT
```

```
LOC  OBJ         LINE        SOURCE STATEMENT

                 = 187 ;
                 = 188 ;REORG  MACRO TO RESET THE INSTRUCTION LOCATION COUNTER
                 = 189 ;       TO THE FIRST FREE LOCATION ON THE FIRST PAGE MODULE WILL
                 = 190 ;       FIT WITHIN.
                 = 191 REORG   MACRO LOCATION
                 = 192 $SAVE GEN
-                = 193           ORG     LOCATION
-                = 194 $RESTORE
                 = 195         ENDM
                 = 196 ;
                 = 197 ;CODEBLK       MACRO TO FIND A PAGE OF ROM
                 = 198 ;       WHICH THIS BLOCK OF CODE WILL FIT WITHIN
                 = 199 CODEBLK MACRO   LENGTH
-                = 200 ?LENGTH SET     LENGTH
--               = 201 IF      HIGH(ORGPG0+LENGTH-1) EQ 0
--               = 202         REORG   %ORGPG0
-                = 203 ?START  SET     $
--               = 204 EXITM
--               = 205 ENDIF
--               = 206 IF      HIGH(ORGPG1+LENGTH-1) EQ 1
--               = 207         REORG   %ORGPG1
--               = 208 ?START  SET     $
--               = 209 EXITM
-                = 210 ENDIF
-                = 211 IF      HIGH(ORGPG2+LENGTH-1) EQ 2
-                = 212         REORG   %ORGPG2
--               = 213 ?START  SET     $
-                = 214 EXITM
-                = 215 ENDIF
--               = 216 IF      HIGH(ORGPG4+LENGTH-1) EQ 4
-                = 217         REORG   %ORGPG4
-                = 218 ?START  SET     $
--               = 219 EXITM
--               = 220 ENDIF
--               = 221 IF      HIGH(ORGPG5+LENGTH-1) EQ 5
--               = 222         REORG   %ORGPG5
-                = 223 ?START  SET     $
--               = 224 EXITM
-                = 225 ENDIF
-                = 226 IF      HIGH(ORGPG6+LENGTH-1) EQ 6
-                = 227         REORG   %ORGPG6
--               = 228 ?START  SET     $
--               = 229 EXITM
-                = 230 ENDIF
-                = 231 IF      HIGH(ORGPG7+LENGTH-1) EQ 7
-                = 232         REORG   %ORGPG7
-                = 233 ?START  SET     $
--               = 234 EXITM
--               = 235 ENDIF
-                = 236 IF      HIGH(ORGPG3+LENGTH-1) EQ 3
--               = 237         REORG   %ORGPG3
-                = 238 ?START  SET     $
--               = 239 EXITM
--               = 240 ENDIF
-                = 241         ERROR   0       ;*** INSUFFICIENT SPACE FOR CODE ON ANY PAGE ***
```

```
LOC  OBJ          LINE        SOURCE STATEMENT

                  = 242         ENDM
                  = 243 ;DATABLK        INSERTS ONTO PAGE 3
                  = 244 DATABLK MACRO   LENGTH
-                 = 245 ?LENGTH SET     LENGTH
--                = 246 IF      HIGH(ORGPG3+LENGTH-1) EQ 3
--                = 247         REORG   %ORGPG3
-                 = 248 ?START  SET     $
-                 = 249 EXITM
-                 = 250 ENDIF
-                 = 251         ERROR   0       ;*** INSUFFICIENT SPACE FOR DATA BLOCK ON PAGE 3 ***
                  = 252         ENDM
                  = 253 ;?SIZE  PRINTS A LINE TO THE SOURCE FILE GIVING BLOCK SIZE.
                  = 254 ;       AND UPDATES APPROPRIATE ORGPG#
                  = 255 ?SIZE   MACRO   BLK,PGE
                  = 256 $SAVE GEN
-                 = 257   SIZE  SET   BLK
-                 = 258 ;
-                 = 259 ;*******************************************************
--                = 260 IF ?LENGTH LT SIZE
-                 = 261         ERROR   0       ;*** SIZE EXCEEDS SPACE CHECKED FOR BY CODEBLK MACRO
-                 = 262 ENDIF
-                 = 263 IF      HIGH($-1) NE HIGH(?START)
-                 = 264         ERROR   0       ;*** CODE OR DATA BLOCK ROLLED OVER PAGE BOUNDARY ***
--                = 265 ENDIF
-                 = 266 $RESTORE
--                = 267 ORGPG&PGE       SET     $
                  = 268         ENDM
                  = 269 ;SIZECHK        CHECKS SIZE OF PRECEDING BLOCK, PRINTS SIZE TO .LST FILE.
                  = 270 SIZECHK MACRO
--                = 271         ?SIZE   %($-?START),%HIGH(?START)
                  = 272         ENDM
                  = 273 ;
                  = 274 ;
                  = 275 ;RSOURCE        CODE SPACE  ALLOCATION SUMMARY STATEMENT
                  = 276 RSOURCE MACRO
                  = 277 $SAVE LIST GEN
-                 = 278           PGSIZE  SET   ORGPG0-000H     ;BYTES USED ON PAGE 0
-                 = 279           PGSIZE  SET   ORGPG1-100H     ;BYTES USED ON PAGE 1
-                 = 280           PGSIZE  SET   ORGPG2-200H     ;BYTES USED ON PAGE 2
-                 = 281           PGSIZE  SET   ORGPG3-300H     ;BYTES USED ON PAGE 3
-                 = 282           PGSIZE  SET   ORGPG4-400H     ;BYTES USED ON PAGE 4
-                 = 283           PGSIZE  SET   ORGPG5-500H     ;BYTES USED ON PAGE 5
-                 = 284           PGSIZE  SET   ORGPG6-600H     ;BYTES USED ON PAGE 6
-                 = 285           PGSIZE  SET   ORGPG7-700H     ;BYTES USED ON PAGE 7
-                 = 286 $EJECT
-                 = 287 $RESTORE
                  = 288         ENDM
                  = 289 $EJECT
```

```
LOC  OBJ          LINE        SOURCE STATEMENT

                    290 ;
                    291 $       INCLUDE(:F0:MOPCOD.MAC)
                  = 292 ;
                  = 293 ;?FORM1 MACRO   FOR GENERALIZING OPCODE INSTRUCTION
                  = 294 ;
                  = 295 ?FORM1  MACRO   OPCODE,SRC
-                 = 296 IF      ?&SRC EQ 2
--                = 297 $       SAVE GEN
-                 = 298           MOV     R1,#SRC
-                 = 299           OPCODE          A,@R1
-                 = 300 $       RESTORE
-                 = 301         EXITM
-                 = 302 ENDIF
--                = 303 IF      ?&SRC EQ 0 OR ?&SRC EQ 1
-                 = 304 $       SAVE GEN
--                = 305           OPCODE          A,SRC
-                 = 306 $       RESTORE
-                 = 307         EXITM
-                 = 308 ENDIF
--                = 309 IF      ?&SRC EQ 3
--                = 310 $       SAVE GEN
-                 = 311           OPCODE          A,#SRC
--                = 312 $       RESTORE
--                = 313         EXITM
--                = 314 ENDIF
--                = 315         ERROR   1
                  = 316 ENDM
                  = 317 ;
                  = 318 ;?FORM2 MACRO   FOR GENERALIZING MOVES FROM THE ACC TO A VARIABLE
                  = 319 ?FORM2  MACRO   DEST
-                 = 320 IF      ?&DEST EQ 2
-                 = 321 $       SAVE GEN
-                 = 322           MOV     R1,#DEST
-                 = 323           MOV     @R1,A
-                 = 324 $       RESTORE
--                = 325         EXITM
-                 = 326 ENDIF
--                = 327 IF      ?&DEST EQ 0 OR ?&DEST EQ 1
--                = 328 $       SAVE GEN
-                 = 329           MOV     DEST,A
-                 = 330 $       RESTORE
-                 = 331         EXITM
--                = 332 ENDIF
-                 = 333         ERROR   1
                  = 334 ENDM
                  = 335 ;
                  = 336 ;?FORM3 MACRO   FOR GENERALIZING MOVES FROM THE ACC TO A VARIABLE
                  = 337 ;               WHEN IT IS KNOWN THAT R1 (IF NEEDED FOR INDIRECT ADDRESSING)
                  = 338 ;               IS ALREADY PRESET.
                  = 339 ?FORM3  MACRO   DEST
--                = 340 IF      ?&DEST EQ 2
-                 = 341 $       SAVE GEN
--                = 342           MOV     @R1,A
--                = 343 $       RESTORE
-                 = 344         EXITM
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

                = 345 ENDIF
                = 346 IF      ?&DEST EQ 0 OR ?&DEST EQ 1
                = 347 $       SAVE GEN
                = 348           MOV     DEST,A
                = 349 $       RESTORE
                = 350         EXITM
                = 351 ENDIF
                = 352         ERROR   1
                = 353 ENDM
                = 354 ;
                = 355 ;?FORM4 MACRO   FOR GENERALIZING 'MOV   A,SRC' INSTRUCTION
                = 356 ?FORM4  MACRO   SRC
                = 357 IF      ?&SRC EQ 2
                = 358 $       SAVE GEN
                = 359           MOV     R1,#SRC
                = 360           MOV     A,@R1
                = 361 $       RESTORE
                = 362         EXITM
                = 363 ENDIF
                = 364 IF      ?&SRC EQ 0 OR ?&SRC EQ 1
                = 365 $       SAVE GEN
                = 366           MOV     A,SRC
                = 367 $       RESTORE
                = 368         EXITM
                = 369 ENDIF
                = 370 IF      ?&SRC EQ 3
                = 371 $       SAVE GEN
                = 372           MOV     A,#SRC
                = 373 $       RESTORE
                = 374         EXITM
                = 375 ENDIF
                = 376         ERROR   1
                = 377 ENDM
                = 378 ;
                = 379 ;?FORM5 MACRO   FOR GENERALIZING MOVING A CONSTANT INTO A VARIABLE
                = 380 ?FORM5  MACRO   DEST,CONST
                = 381 IF      ?&DEST EQ 0 OR ?&DEST EQ 1 OR ?&DEST EQ 4
                = 382 $       SAVE GEN
                = 383           MOV     DEST,#CONST
                = 384 $       RESTORE
                = 385         EXITM
                = 386 ENDIF
                = 387 IF      ?&DEST EQ 2
                = 388 $       SAVE GEN
                = 389           MOV     R1,#DEST
                = 390           MOV     @R1,#CONST
                = 391 $       RESTORE
                = 392         EXITM
                = 393 ENDIF
                = 394         ERROR   1
                = 395 ENDM
                = 396 ;
                = 397 ;MMOV   MACRO   GENERALIZED MOVE FROM SRC TO DEST
                = 398 MMOV    MACRO   DEST,SRC
                = 399 IF      ?&SRC EQ 3
```

```
LOC  OBJ          LINE        SOURCE STATEMENT

                  = 400         ?FORM5  DEST,SRC
                  = 401         EXITM
                  = 402 ENDIF
                  = 403 IF      ?&DEST EQ 4
                  = 404         ?FORM1  MOV,SRC
                  = 405         EXITM
                  = 406 ENDIF
                  = 407 IF      ?&SRC EQ 4
                  = 408         ?FORM2  DEST
                  = 409         EXITM
                  = 410 ENDIF
                  = 411         ?FORM1  MOV,SRC
                  = 412         ?FORM2  DEST
                  = 413 ENDM
                  = 414 ;?BINOP MACRO   GENERALIZES ARITHMETIC AND LOGICAL OPERATIONS
                  = 415 ?BINOP  MACRO   OPCODE,DEST,SRC
                  = 416 IF      ?&DEST EQ 4
                  = 417         ?FORM1  OPCODE,SRC
                  = 418         EXITM
                  = 419 ENDIF
                  = 420 IF      ?&SRC EQ 4
                  = 421         ?FORM1  OPCODE,DEST
                  = 422         ?FORM3  DEST
                  = 423         EXITM
                  = 424 ENDIF
                  = 425         ?FORM1  MOV,SRC
                  = 426         ?FORM1  OPCODE,DEST
                  = 427         ?FORM3  DEST
                  = 428 ENDM
                  = 429 ;MADD   MACRO   FOR GENERALIZING ADD INSTRUCTION
                  = 430 MADD    MACRO   DEST,SRC
                  = 431         ?BINOP  ADD,DEST,SRC
                  = 432         ENDM
                  = 433 ;
                  = 434 ;MADDC  MACRO   FOR GENERALIZING ADDC INSTRUCTION
                  = 435 MADDC   MACRO   DEST,SRC
                  = 436         ?BINOP  ADDC,DEST,SRC
                  = 437         ENDM
                  = 438 ;
                  = 439 ;MANL   MACRO   FOR GENERALIZING ANL INSTRUCTION
                  = 440 MANL    MACRO   DEST,SRC
                  = 441         ?BINOP  ANL,DEST,SRC
                  = 442         ENDM
                  = 443 ;
                  = 444 ;MORL   MACRO   FOR GENERALIZING ORL INSTRUCTION
                  = 445 MORL    MACRO   DEST,SRC
                  = 446         ?BINOP  ORL,DEST,SRC
                  = 447         ENDM
                  = 448 ;
                  = 449 ;MXRL   MACRO   FOR GENERALIZING XRL INSTRUCTION
                  = 450 MXRL    MACRO   DEST,SRC
                  = 451         ?BINOP  XRL,DEST,SRC
                  = 452         ENDM
                  = 453 ;
                  = 454 ;MXCH   MACRO   FOR GENERALIZING XCH INSTRUCTION
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

                = 455 MXCH    MACRO   DEST,SRC
-               = 456         ?BINOP  XCH,DEST,SRC
                = 457         ENDM
                = 458 ;
                = 459 ?UNARY  MACRO   OPCODE,DEST
-               = 460         ?FORM1  MOV,DEST
-               = 461 $SAVE GEN
--              = 462           OPCODE          A
--              = 463 $RESTORE
--              = 464         ?FORM3  DEST
                = 465         ENDM
                = 466 ;
                = 467 MINC    MACRO   DEST
-               = 468         ?UNARY  INC,DEST
                = 469         ENDM
                = 470 ;
                = 471 MDEC    MACRO   DEST
-               = 472         ?UNARY  DEC,DEST
                = 473         ENDM
                = 474 ;
                = 475 MDJNZ   MACRO   DEST,ADDR
-               = 476         ?UNARY  DEC,DEST
--              = 477 $SAVE GEN
---             = 478           JNZ     ADDR
--              = 479 $RESTORE
                = 480         ENDM
                = 481 ;
                = 482 MRL     MACRO   DEST
-               = 483         ?UNARY  RL,DEST
                = 484         ENDM
                = 485 ;
                = 486 MRR     MACRO   DEST
--              = 487         ?UNARY  RR,DEST
                = 488         ENDM
                = 489 ;
                = 490 MRRC    MACRO   DEST
-               = 491         ?UNARY  RRC,DEST
                = 492         ENDM
                = 493 ;
                = 494 MRLC    MACRO   DEST
-               = 495         ?UNARY  RLC,DEST
                = 496         ENDM
                = 497 ;
                = 498 $EJECT
```

```
LOC  OBJ        LINE       SOURCE STATEMENT

                  499 ;
                  500 ;=======================================================
                  501 ;=======================================================
                  502 ;                BEGINNING OF PROGRAM PROPER
                  503 ;=======================================================
                  504 ;=======================================================
                  505 ;
                  506 ;
                  507 ;*********************************************************
                  508 ;
                  509 ;       ALLOCATION OF MP I/O PORTS:
                  510 ;
                  511 ;*********************************************************
                  512 ;
                  513 ;       BUS             ;USED FOR BIDIRECTIONAL ADDRESS AND DATA TRANSFERS
                  514 ;       P1              ;USED AS INDIVIDUAL CONTROL OUTPUTS AND BREAK LOGIC
                  515 ;       P2              ;HIGH-ORDER ADDRESS AND ADDRESS SPACE SELECTION
                  516 ;
000E              517 PDIGIT  EQU     P7      ;USED TO ENABLE CHARACTERS AND STROBE ROWS OF KEYBOARD
000D              518 PSEGHI  EQU     P6      ;USED TO TURN ON HI SEGMENTS OF CURRENTLY ENABLED DIGIT
000C              519 PSEGLO  EQU     P5      ;PORT FOR LOWER FOUR SEGMENTS
000B              520 PINPUT  EQU     P4      ;PORT USED TO SCAN FOR KEY CLOSURES
                  521 ;
                  522 ;*********************************************************
                  523 ;
                  524 ;       INDIVIDUAL PINS OF PORT 1 USED AS FOLLOWS:
                  525 ;
                  526 ;*********************************************************
                  527 ;
0001              528 ENBRAM  EQU     00000001B   ;P10 - HI ENABLES BREAK ON BREAK RAM OUTPUT SIGNAL
0002              529 ENBLNK  EQU     00000010B   ;P11 - HI ENABLES BREAK ON RD OR WR TO LINK BY EP
                  530                             ;      (NOTE: P11 & P10 BOTH HI ENABLES
                  531                             ;      BREAK ON ANY EP INSTRUCTION CYCLE)
0004              532 EPSSTP  EQU     00000100B   ;P12 - LO FORCES EP SS INPUT LOW,
                  533                             ;      HI GATES BREAKPOINT FLIP-FLOP TO EP SS INPUT.
0008              534 CLRBFF  EQU     00001000B   ;P13 - LO CLEARS BREAK FLIP-FLOP
                  535                             ;      AND ENABLES WR CONTROL TO BREAKPOINT RAM.
0010              536 EPRSET  EQU     00010000B   ;P14 - HI RESETS EP
0020              537 MODOUT  EQU     00100000B   ;P15 - LO WHEN EP IS EXECUTING USER PROGRAM,
                  538                             ;      HI WHEN EP FROZEN OR RUNNING OVERLAYS.
0040              539 TTYOUT  EQU     01000000B   ;P16 - SERIAL OUTPUT TO TTY OR CRT
                  540                             ;P17 - UNUSED
                  541 ;
                  542 $EJECT
```

```
LOC  OBJ        LINE       SOURCE STATEMENT

                543 ;*********************************************************
                544 ;
                545 ;       INDIVIDUAL PINS OF PORT 2 USED AS FOLLOWS:
                546 ;
                547 ;*********************************************************
                548 ;
                549 ;       P23-P20             ;ADR11-ADR8 FOR ACCESSING PROGRAM OR DATA RAM ARRAY
                550 ;
0010            551 M0      EQU     00010000B   ;P24 - MEMORY MATRIX CONTROL PIN 0
0020            552 M1      EQU     00100000B   ;P25 - MEMORY MATRIX CONTROL PIN 1
0040            553 MPUSEL  EQU     01000000B   ;P26 - HIGH WHEN MP IN CONTROL OF COMMON MEM ARRAY,
                554                             ;      LOW WHEN EP IN CONTROL.
0080            555 EXPMON  EQU     10000000B   ;P27 - JUMPERED TO GROUND FOR STANDARD MONITOR,
                556                             ;      FLOATING WHEN EXPANSION MONITOR PRESENT.
                557 ;
                558 ;
                559 ;WHEN MP IN CONTROL OF MEMORY MATRIX M1-M0 USED AS FOLLOWS:
                560 ;
                561 ;       M1 M0   MODE
                562 ;        0  0   PROGRAM RAM ARRAY ENABLED FOR READ & WRITE
                563 ;        0  1   DATA RAM ARRAY ENABLED FOR READ & WRITE
                564 ;        1  X   LINK REGISTER ENABLED FOR READ, RAM ARRAYS DISABLED.
                565 ;               (NOTE: LINK REGISTER ALWAYS ENABLED FOR MP WRITES)
                566 ;
                567 ;WHEN EP IN CONTROL OF MATRIX M1-M0 USED AS FOLLOWS:
                568 ;
                569 ;       M1 M0   MODE
                570 ;        0  X   EP PSEN FETCHES FROM LINK REGISTER (USED TO FORCE OPCODES)
                571 ;        1  0   EP PSEN FETCHES FROM PROGRAM RAM ARRAY,
                572 ;               EP RD & WR CONTROL DATA RAM ARRAY.
                573 ;        1  1   EP PSEN FETCHES FROM PROGRAM RAM ARRAY,
                574 ;               RD & WR CONTROL LINK REGISTER.
                575 ;
                576 $EJECT
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

                577 ;
                578 ;*******************************************************
                579 ;
                580 ;       SYSTEM CONSTANT DEFINITIONS:
                581 ;
                582 ;*******************************************************
                583 ;
                584 DECLARE CHARNO,CONST    ;NUMBER OF DIGITS IN DISPLAY AND ROWS OF KEYS
0008            598         CHARNO  EQU     8
                599 ;
                600 DECLARE NCOLS,CONST     ;LESSER DIMENSION OF KEYBOARD MATRIX
0004            614         NCOLS   EQU     4
                615 ;
                616 DECLARE DEBNCE,CONST    ;NUMBER OF SUCESSIVE SCANS BEFORE KEY CLOSURE ACCEPTED
0008            630         DEBNCE  EQU     8
                631 ;
                632 DECLARE OVSIZE,CONST    ;SIZE OF LARGEST MINI-MONITOR OVERLAY FOR EP
0017            646         OVSIZE  EQU     23
                647 ;
                648 DECLARE BUFLEN,CONST    ;LENGTH OF HEX FORMAT XMIT BUFFER (MAX RECORD LENGTH)
0010            662         BUFLEN  EQU     16
                663 ;
                664 ;*******************************************************
                665 ;
                666 ;       UTILITY CONSTANT DECLARATIONS
                667 ;
                668 ;*******************************************************
                669 ;
                670 DECLARE ZERO,CONST
0000            684 ZERO    EQU     0
                685 DECLARE PLUS1,CONST
0001            699 PLUS1   EQU     1
                700 DECLARE PLUS3,CONST
0003            714 PLUS3   EQU     3
                715 DECLARE NEG1,CONST
FFFF            729 NEG1    EQU     -1
                730 ;
                731 $EJECT
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

                732 ;
                733 ;*******************************************************
                734 ;
                735 ;       BANK 0 REGISTER ALLOCATION:
                736 ;
                737 ;*******************************************************
                738 ;
                739 DECLARE LDATA,RB0       ;DATA USED BY LOGICAL ADDRESSING READ/WRITE UTILITIES
0002            752+        LDATA   SET  R2
                756 DECLARE KEY,RB0         ;HOLDS KEYCODE RETURNED FROM KBD INPUT ROUTINE.
0003            769+        KEY     SET  R3
                773 DECLARE ITMP,RB0        ;COUNTER USED AS AN INDEX IN PARSER ROUTINE
0004            786+        ITMP    SET  R4
                790 DECLARE CHKSUM,RB0      ;CHECKSUM OF DATA BYTES TRANSMITTED IN HEX FILE FORMAT
0005            803+        CHKSUM  SET  R5
                807 DECLARE DSPTMP,RB0      ;TEMPORARY STORAGE FOR DISPLAY PATTERNS IN 'DSPACC'
0006            820+        DSPTMP  SET  R6
                824 DECLARE XPCODE,RB0      ;EXPANSION MONITOR ROUTINE CODE NUMBER
0007            837+        XPCODE  SET  R7
                841 ;
                842 ;*******************************************************
                843 ;
                844 ;       BANK 1 REGISTER ALLOCATION
                845 ;
                846 ;*******************************************************
                847 ;
                848 DECLARE ROTPAT,RB1      ;USED TO HOLD INPUT PATTERN BEING ROTATED THROUGH CY
0002            865+        ROTPAT  SET  R2
                869 DECLARE ROTCNT,RB1      ;COUNTS NUMBER OF BITS ROTATED THROUGH CY
0003            886+        ROTCNT  SET  R3
                890 DECLARE LASTKY,RB1      ;HOLDS KEY POSITION OF LAST KEY DEPRESSION DETECTED
0004            907+        LASTKY  SET  R4
                911 DECLARE CURDIG,RB1      ;HOLDS POSITION OF NEXT CHARACTER TO BE DISPLAYED
0005            928+        CURDIG  SET  R5
                932 DECLARE KEYFLG,RB1      ;FLAG TO DETECT WHEN ALL KEYS ARE RELEASED
0006            949+        KEYFLG  SET  R6
                953                         ;(REGISTER 7 NOT USED FOR PRIMARY MONITOR)
                954 ;
                955 ;*******************************************************
                956 $EJECT
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

                 957 ;
                 958 ;*******************************************************
                 959 ;
                 960 ;       DATA RAM ALLOCATION
                 961 ;
                 962 ;*******************************************************
                 963 ;
                 964 DECLARE EPACC,RAM       ;STORAGE IN MP FOR EP ACCUMULATOR
0020             969+        EPACC   EQU  32
                 973 DECLARE EPPSW,RAM       ;STORAGE IN MP FOR EP PROGRAM STATUS WORD
0021             978+        EPPSW   EQU  33
                 982 DECLARE EPTIMR,RAM      ;STORAGE IN MP FOR EP TIMER/COUNTER REGISTER
0022             987+        EPTIMR  EQU  34
                 991 DECLARE EPR0,RAM        ;STORAGE IN MP FOR EP REGISTER 0 OF BANK 0
0023             996+        EPR0    EQU  35
                1000 DECLARE EPPCLO,RAM      ;STORAGE IN MP FOR LOW BYTE OF EP PROGRAM COUNTER
0024            1005+        EPPCLO  EQU  36
                1009 DECLARE EPPCHI,RAM      ;STORAGE IN MP FOR HIGH NIBBLE OF EP PROGRAM COUNTER
0025            1014+        EPPCHI  EQU  37
                1018 DECLARE HBITLO,RAM      ;PARAMETER 1 FOR SERIAL LINK DATA RATE GENERATOR
0026            1023+        HBITLO  EQU  38
                1027 DECLARE HBITHI,RAM      ;PARAMETER 2 FOR SERIAL LINK DATA RATE GENERATOR
0027            1032+        HBITHI  EQU  39
                1036 DECLARE DSPTIM,RAM      ;PARAMETER FOR AUTO-STEP AND AUTO-BREAK SEQUENCING RATE
0028            1041+        DSPTIM  EQU  40
                1045 DECLARE VERSNO,RAM      ;MONITOR VERSION NUMBER
0029            1050+        VERSNO  EQU  41
                1054 DECLARE HREGA,RAM       ;  (UNUSED)
002A            1059+        HREGA   EQU  42
                1063 DECLARE HREGB,RAM       ;  (UNUSED)
002B            1068+        HREGB   EQU  43
                1072 DECLARE HREGC,RAM       ;  (UNUSED)
002C            1077+        HREGC   EQU  44
                1081 DECLARE HREGD,RAM       ;  (UNUSED)
002D            1086+        HREGD   EQU  45
                1090 DECLARE HREGE,RAM       ;  (UNUSED)
002E            1095+        HREGE   EQU  46
                1099 DECLARE HREGF,RAM       ;  (UNUSED)
002F            1104+        HREGF   EQU  47
                1108 DECLARE SMALO,RAM       ;PRIMARY COMMAND STARTING MEMORY ADDRESS (LOW BYTE)
0030            1113+        SMALO   EQU  48
                1117 DECLARE SMAHI,RAM       ;PRIMARY COMMAND STARTING MEMORY ADDRESS (HIGH BYTE)
0031            1122+        SMAHI   EQU  49
                1126 DECLARE EMALO,RAM       ;PRIMARY COMMAND ENDING MEMORY ADDRESS (LOW BYTE)
0032            1131+        EMALO   EQU  50
                1135 DECLARE EMAHI,RAM       ;PRIMARY COMMAND ENDING MEMORY ADDRESS (HIGH BYTE)
0033            1140+        EMAHI   EQU  51
                1144 DECLARE MEMLO,RAM       ;THIRD PARSER PARAMETER & HEX RECORD ADDRESS (LOW)
0034            1149+        MEMLO   EQU  52
                1153 DECLARE MEMHI,RAM       ;THIRD PARSER PARAMETER & HEX RECORD ADDRESS (HIGH)
0035            1158+        MEMHI   EQU  53
                1162 DECLARE BCODE,RAM       ;PRIMARY COMMAND NUMBER FROM PARSER TABLES (0-8)
0036            1167+        BCODE   EQU  54
                1171 DECLARE TYPE,RAM        ;PRIMARY COMMAND MODIFIER/OPTION (0-5)
0037            1176+        TYPE    EQU  55
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

                1180 DECLARE NUMCON,RAM     ;MAX. NUMBER OF PARAMETERS ALLOWED FOR SELECTED COMMAND
0038            1185+        NUMCON  EQU  56
                1189 DECLARE OPTION,RAM     ;INDEX POINTER USED IN SEARCHING PARSER TABLES
0039            1194+        OPTION  EQU  57
                1198 DECLARE NEXTPL,RAM     ;CHARACTER POSITION FOR DISPLAY UTILITIES TO WRITE NEXT
003A            1203+        NEXTPL  EQU  58
                1207 DECLARE KBDBUF,RAM     ;POSITION OF KEY DEBOUNCED BY SCANNING SUBROUTINE
003B            1212+        KBDBUF  EQU  59
                1216 DECLARE KEYLOC,RAM     ;INCREMENTED AS SUCCESSIVE KEY LOCATIONS SCANNED
003C            1221+        KEYLOC  EQU  60
                1225 DECLARE NREPTS,RAM     ;KEEPS TRACK OF SUCCESSIVE READS OF SAME KEYSTROKE
003D            1230+        NREPTS  EQU  61
                1234 DECLARE ASAVE,RAM      ;HOLDS ACCUMULATOR VALUE DURING SERVICE ROUTINE
003E            1239+        ASAVE   EQU  62
                1243 DECLARE RDELAY,RAM     ;COUNTER DECREMENTED WHEN AUTO-STEP DELAY IN PROGRESS
003F            1248+        RDELAY  EQU  63
                1252 DECLARE STRTMP,RAM     ;INDEX POINTER FOR DISPLAY CHARACTER STRING ACCESSING
0040            1257+        STRTMP  EQU  64
                1261 DECLARE BUFCNT,RAM     ;COUNT OF DATA BYTES IN HEX FORMAT RECORD BUFFER
0041            1266+        BUFCNT  EQU  65
                1270 DECLARE RECTYP,RAM     ;TYPE OF HEX FORMAT RECORD (0 OR 1)
0042            1275+        RECTYP  EQU  66
                1279 DECLARE B,RAM          ;BIT COUNTER FOR ASCII SERIAL I/O UTILITY SUBROUTINES
0043            1284+        B       EQU  67
                1288 DECLARE REGC,RAM       ;CHARACTER BEING SHIFTED DURING SERIAL I/O PROCESS
0044            1293+        REGC    EQU  68
                1297 DECLARE H,RAM          ;COUNTER IN SOFTWARE DELAY DATA RATE GENERATOR
0045            1302+        H       EQU  69
                1306 ;
                1307 MBLOCK  SEGMAP,CHARNO  ;REGISTER ARRAY FOR DISPLAY PATTERNS
0046            1311+        SEGMAP  EQU  70
                1314 ;
                1315 MBLOCK  OVBUF,OVSIZE   ;LOW-ORDER USER PROGRAM DURING MINI-MONITOR OVERLAYS
004E            1319+        OVBUF   EQU  78
                1322 ;
                1323 MBLOCK  HEXBUF,BUFLEN  ;ALLOCATE BLOCK OF RAM FOR USE AS HEX RECORD BUFFER
0065            1327+        HEXBUF  EQU  101
                1330 ;
                1331 $EJECT
```

```
LOC  OBJ        LINE     SOURCE STATEMENT

                1332        DATABLK 40
0300            1337+         ORG    768
                1341 ;INVALS TABLE OF CONSTANTS TO BE LOADED INTO MP INTERNAL RAM VARIABLES
                1342 ;       AS PART OF SYSTEM INITIALIZATION PROCEDURE:
                1343 ;
                1344 ;       INITIAL VALUE   VARIABLE       TYPE
                1345 ;       ======= =====   ========       ====
0300 00         1346 INVALS: DB      00H     ;ROTPAT        RB1
0301 00         1347         DB      00H     ;ROTCNT        RB1
0302 00         1348         DB      00H     ;LASTKY        RB1
0303 08         1349         DB      CHARNO  ;CURDIG        RB1
0304 00         1350         DB      00H     ;KEYFLG        RB1
0305 00         1351         DB      00H     ;<REG7>        RB1
0306 00         1352         DB      00H     ;EPACC         RAM
0307 01         1353         DB      01H     ;EPPSW         RAM
0308 00         1354         DB      00H     ;EPTIMR        RAM
0309 00         1355         DB      00H     ;EPR0          RAM
030A 00         1356         DB      00H     ;EPPCLO        RAM
030B 00         1357         DB      00H     ;EPPCHI        RAM
030C 93         1358         DB      93H     ;HBITLO        RAM
030D 04         1359         DB      04H     ;HBITHI        RAM
030E 20         1360         DB      20H     ;DSPTIM        RAM
030F 25         1361         DB      25H     ;VERSNO        RAM
0310 00         1362         DB      00H     ;HREGA         RAM
0311 00         1363         DB      00H     ;HREGB         RAM
0312 00         1364         DB      00H     ;HREGC         RAM
0313 00         1365         DB      00H     ;HREGD         RAM
0314 00         1366         DB      00H     ;HREGE         RAM
0315 00         1367         DB      00H     ;HREGF         RAM
0316 00         1368         DB      00H     ;SMALO         RAM
0317 00         1369         DB      00H     ;SMAHI         RAM
0318 FF         1370         DB      0FFH    ;EMALO         RAM
0319 0F         1371         DB      0FH     ;EMAHI         RAM
031A 00         1372         DB      00H     ;MEMLO         RAM
031B 00         1373         DB      00H     ;MEMHI         RAM
031C 00         1374         DB      00H     ;BCODE         RAM
031D 04         1375         DB      04H     ;TYPE          RAM
031E 01         1376         DB      01H     ;NUMCON        RAM
031F 00         1377         DB      00H     ;OPTION        RAM
0320 08         1378         DB      CHARNO  ;NEXTPL        RAM
0321 FF         1379         DB      0FFH    ;KBDBUF        RAM
0322 00         1380         DB      00H     ;KEYLOC        RAM
0023            1381 NOVALS  EQU     $-INVALS
                1382         SIZECHK
0023            1385+  SIZE  SET   35
                1386+;
                1387+;**************************************************************
                1396 $EJECT
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

                 1397 $       INCLUDE(:F0:PARSER.MOD)
                =1398         CODEBLK 45
0000            =1403+          ORG     0
                =1407 ;INIT   INITIALIZES PROCESSOR REGISTERS
                =1408 ;       AND RAM LOCATIONS TO DEFINED VALUES.
0000 C5         =1409 INIT:   SEL     RB0
0001 BF00       =1410         MOV     XPCODE,#0
0003 74D1       =1411         CALL    XPTEST
0005 27         =1412         CLR     A
0006 3D         =1413         MOVD    PSEGLO,A
0007 3E         =1414         MOVD    PSEGHI,A
0008 B81A       =1415         MOV     R0,#1AH ;START AT RB1 (REG2) = RAM LOC 1AH
000A B923       =1416         MOV     R1,#LOW NOVALS
000C BA00       =1417         MOV     R2,#LOW INVALS
000E FA         =1418 INITLP: MOV     A,R2
000F E3         =1419         MOVP3   A,@A
0010 A0         =1420         MOV     @R0,A
0011 18         =1421         INC     R0
0012 1A         =1422         INC     R2
0013 E90E       =1423         DJNZ    R1,INITLP
0015 55         =1424         STRT    T
0016 744F       =1425         CALL    EPBRK
0018 B8BB       =1426         MOV     R0,#LOW(OV1BAS+OVSIZE)
001A 746A       =1427         CALL    OVLORD
001C 54E5       =1428         CALL    COMFIL
001E B937       =1429         MOV     R1,#TYPE
0020 11         =1430         INC     @R1
0021 34F2       =1431         CALL    INCSMA
0023 54E5       =1432         CALL    COMFIL
0025 99EF       =1433         ANL     P1,#(NOT EPRSET)        ;REMOVE EP RESET SIGNAL.
0027 0429       =1434         JMP     MAIN
                =1435 ;
                =1436         SIZECHK
0029            =1439+  SIZE  SET   41
                =1440+;
                =1441+;*******************************************************
                =1450 $EJECT
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

                =1451 ;
                =1452 ;       KEYBOARD LAYOUT:
                =1453 ;       ======== ======
                =1454 ;
                =1455 ;    ---------- ---------- ---------- ---------- ------ ------ ------ ------
                =1456 ;    !         !!         !!         !!         ! !    !!     !!     !!     !
                =1457 ;    ! LIST    !!GO/RESET!!   GO    !!EXAM/CHG! ! C  !! D  !! E  !! F  !
                =1458 ;    !         !!         !!         !!         ! !    !!     !!     !!     !
                =1459 ;    ---------- ---------- ---------- ---------- ------ ------ ------ ------
                =1460 ;    ---------- ---------- ---------- ---------- ------ ------ ------ ------
                =1461 ;    !         !!PROG BRK!!PROG MEM!!REGISTER! !    !!     !!     !!     !
                =1462 ;    ! UPLOAD  !! ----  !! ----  !! ----  ! ! 8  !! 9  !! A  !! B  !
                =1463 ;    !         !!AUTO STP!!SING STP!! NO BRK ! !    !!     !!     !!     !
                =1464 ;    ---------- ---------- ---------- ---------- ------ ------ ------ ------
                =1465 ;    ---------- ---------- ---------- ---------- ------ ------ ------ ------
                =1466 ;    !         !!DATA BRK!!DATA MEM!!         ! !    !!     !!     !!     !
                =1467 ;    ! DNLOAD  !! ----  !! ----  !!CLR/PREV! ! 4  !! 5  !! 6  !! 7  !
                =1468 ;    !         !!AUTO BRK!!WITH BRK!!         ! !    !!     !!     !!     !
                =1469 ;    ---------- ---------- ---------- ---------- ------ ------ ------ ------
                =1470 ;    ---------- ---------- ---------- ---------- ------ ------ ------ ------
                =1471 ;    !         !!         !!         !!         ! !    !!     !!     !!     !
                =1472 ;    ! FILL    !!HARD REG!! NEXT/  !! END/    ! ! 0  !! 1  !! 2  !! 3  !
                =1473 ;    !         !!         !!         !!         ! !    !!     !!     !!     !
                =1474 ;    ---------- ---------- ---------- ---------- ------ ------ ------ ------
                =1475 ;
                =1476 $EJECT
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

                =1477 ;
                =1478 ;         THE FOLLOWING EQUATES DETERMINES HOW THE PARSER INTERPRETS
                =1479 ;         VALUES RETURNED BY THE KEYBOARD SCANNING INPUT ROUTINE
                =1480 ;         WHEN THE VARIOUS KEYS OF THE KEYBOARD ARE PRESSED.
                =1481 ;
                =1482 ;
                =1483 ;KEY0    EQU     00H     VALUE RETURNED FOR EACH KEY OF KEYBOARD MATRIX
                =1484 ;KEY1    EQU     01H     BY KEYBOARD SCANNING SUBROUTINE "KBDIN".
                =1485 ;KEY2    EQU     02H
                =1486 ;KEY3    EQU     03H     +----+----+----+----+  +----+----+----+----+
                =1487 ;KEY4    EQU     04H     ! 1C ! 1D ! 1E ! 1F !  ! 0C ! 0D ! 0E ! 0F !
                =1488 ;KEY5    EQU     05H     +----+----+----+----+  +----+----+----+----+
                =1489 ;KEY6    EQU     06H     ! 18 ! 19 ! 1A ! 1B !  ! 08 ! 09 ! 0A ! 0B !
                =1490 ;KEY7    EQU     07H     +----+----+----+----+  +----+----+----+----+
                =1491 ;KEY8    EQU     08H     ! 14 ! 15 ! 16 ! 17 !  ! 04 ! 05 ! 06 ! 07 !
                =1492 ;KEY9    EQU     09H     +----+----+----+----+  +----+----+----+----+
                =1493 ;KEYA    EQU     0AH     ! 10 ! 11 ! 12 ! 13 !  ! 00 ! 01 ! 02 ! 03 !
                =1494 ;KEYB    EQU     0BH     +----+----+----+----+  +----+----+----+----+
                =1495 ;KEYC    EQU     0CH
                =1496 ;KEYD    EQU     0DH
                =1497 ;KEYE    EQU     0EH
                =1498 ;KEYF    EQU     0FH
0010            =1499 KEYFIL   EQU     10H     ;[FILL COMMAND]
0012            =1500 KEYNXT   EQU     12H     ;[NEXT/.]
0013            =1501 KEYEND   EQU     13H     ;[END/.]
0014            =1502 KEYREL   EQU     14H     ;[DOWNLOAD COMMAND]
0015            =1503 KEYPAT   EQU     15H     ;[AUTOBREAK MODIFIER]
0016            =1504 KEYDM    EQU     16H     ;[DATA MEMORY MODIFIER]
0017            =1505 KEYCLR   EQU     17H     ;[CLEAR/PREVIOUS]
0018            =1506 KEYREC   EQU     18H     ;[UPLOAD COMMAND]
0019            =1507 KEYTRA   EQU     19H     ;[AUTOSTEP MODIFIER]
001A            =1508 KEYPM    EQU     1AH     ;[PROGRAM MEMORY MODIFIER]
001B            =1509 KEYREG   EQU     1BH     ;[REGISTER MEMORY MODIFIER]
001C            =1510 KEYLST   EQU     1CH     ;[FORMATTED DATA OUTPUT COMMAND]
001D            =1511 KGORES   EQU     1DH     ;[GO FROM RESET STATE COMMAND]
001E            =1512 KEYGO    EQU     1EH     ;[GO COMMAND]
001F            =1513 KEYMOD   EQU     1FH     ;[EXAMINE/MODIFY COMMAND]
000B            =1514 KSETB    EQU     0BH     ;[SET BREAKPOINT COMMAND]
000C            =1515 KCLRB    EQU     0CH     ;[CLEAR BREAKPOINT COMMAND]
                =1516 ;
                =1517 ;
0019            =1518 PBRK     EQU     19H     ;[PROGRAM BREAKPOINT MEMORY MODIFIER]
0015            =1519 DBRK     EQU     15H     ;[DATA BREAKPOINT MEMORY MODIFIER]
0011            =1520 RINT     EQU     11H     ;[HARDWARE REGISTER MEMORY MODIFIER]
001B            =1521 NOBRK    EQU     1BH     ;[WITHOUT BREAKPOINTS MODIFIER]
0016            =1522 WBRK     EQU     16H     ;[WITH BREAKPOINTS ENABLED MODIFIER]
001A            =1523 SING     EQU     1AH     ;[SINGLE STEP MODIFIER]
                =1524 ;
                =1525 $EJECT
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

                =1526          CODEBLK 160
0029            =1531+           ORG     41
                =1535 ;MAIN    OUTPUT_MESSAGE(COMMAND_PROMPT)
                =1536 ;        CALL INPUT_BYTE(KEY)
                =1537 ;MAIN2   IF THE KEY=END GO TO MAIN
                =1538 ;
0029 BF01       =1539 MAIN:    MOV     XPCODE,#1
002B 74D1       =1540          CALL    XPTEST
002D 2301       =1541          MOV     A,#1
002F 3400       =1542          CALL    OUTUTL
0031 14EC       =1543          CALL    INPKEY
0033 FB         =1544 MAIN2:   MOV     A,KEY
0034 D313       =1545          XRL     A,#KEYEND
0036 C629       =1546          JZ      MAIN
                =1547 ;
                =1548 ;FINDOP FIND OUT IF THE KEY PRESSED IS A LEGITIMATE COMMAND INITIATOR:
                =1549 ;        ITMP:=CTAB
                =1550 ;        BCODE:=TYPE:=0
                =1551 ;        WHILE CTAB(ITMP)<>0             /CTAB EXHAUSTED/
                =1552 ;          IF CTAB(ITMP)=KEY GOTO MAINA  /COMMAND ENTRY FOUND IN CTAB/
                =1553 ;          ELSE     ITMP:=ITMP+COMMAND_ENTRY_SIZE
                =1554 ;                   BCODE:=BCODE+1
                =1555 ;        ENDWHILE
                =1556 ;        GOTO ERROR
0038 BC23       =1557          MOV     ITMP,#CTAB
                =1558          MMOV    BCODE,ZERO
003A B936       =1569+           MOV     R1,#BCODE
003C B100       =1570+           MOV     @R1,#ZERO
                =1574          MMOV    TYPE,ZERO
003E B937       =1585+           MOV     R1,#TYPE
0040 B100       =1586+           MOV     @R1,#ZERO
0042 FC         =1590 FINDOP:  MOV     A,ITMP
0043 E3         =1591          MOVP3   A,@A
0044 B2BC       =1592          JB5     MERROR
0046 DB         =1593          XRL     A,KEY
0047 C652       =1594          JZ      MAINA
0049 FC         =1595          MOV     A,ITMP
004A 0303       =1596          ADD     A,#COMSIZ
004C AC         =1597          MOV     ITMP,A
004D B936       =1598          MOV     R1,#BCODE
004F 11         =1599          INC     @R1
0050 0442       =1600          JMP     FINDOP
                =1601 ;
                =1602 ;        OUTPUT_MESSAGE(STRCOM(BCODE))  /*PROMPT FOR THE CURRENT COMMAND*/
                =1603 ;        I:=1+1
                =1604 ;        OPTION:=MEM(I)
                =1605 ;        I:=I+1
                =1606 ;        NO_OF_PARAMETERS:=MEM(I)
                =1607 ;        I:=3
                =1608 ;
                =1609 MAINA:   MMOV    A,BCODE
0052 B936       =1618+           MOV     R1,#BCODE
0054 F1         =1619+           MOV     A,@R1
0055 031D       =1623          ADD     A,#STRCOM
0057 3402       =1624          CALL    OUTCLR
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

0059 1C         =1625         INC     ITMP
005A FC         =1626         MOV     A,ITMP
005B E3         =1627         MOVP3   A,@A            ; GET OPTION POINTER
                =1628         MMOV    OPTION,A
005C B939       =1641+          MOV     R1,#OPTION
005E A1         =1642+          MOV     @R1,A
005F 1C         =1646         INC     ITMP
0060 FC         =1647         MOV     A,ITMP
0061 E3         =1648         MOVP3   A,@A            ; GET NO OF PARAMETERS
                =1649         MMOV    NUMCON,A
0062 B938       =1662+          MOV     R1,#NUMCON
0064 A1         =1663+          MOV     @R1,A
                =1667 ;
                =1668 ;       PARAMETER_BUFFER(0=>5):=0
                =1669 ;
0065 B906       =1670         MOV     R1,#6           ; EACH PARAM IS 2 BYTES
0067 B830       =1671         MOV     R0,#SMALO       ; START OF PARAM BUFFERS
0069 B000       =1672 MAINB:  MOV     @R0,#00H
006B 18         =1673         INC     R0
006C E969       =1674         DJNZ    R1,MAINB
006E 14EC       =1675         CALL    INPKEY
                =1676 ;
                =1677 ;       WHILE KEY<>MEM(OPTION+TYPE)[6-0] DO
                =1678 ;         IF MEM(OPTION+TYPE)[7]=1 GOTO MAIND1
                =1679 ;         TYPE:=TYPE+1
                =1680 ;       ENDWHILE
                =1681 ;
                =1682         MMOV    ITMP,OPTION
0070 B939       =1698+          MOV     R1,#OPTION
0072 F1         =1699+          MOV     A,@R1
0073 AC         =1712+          MOV     ITMP,A
0074 1C         =1715         INC     ITMP
                =1716 MAINC1: MMOV    A,ITMP
0075 FC         =1732+          MOV     A,ITMP
0076 E3         =1736         MOVP3   A,@A
0077 97         =1737         CLR     C
0078 F7         =1738         RLC     A
0079 77         =1739         RR      A       ; STRIP BIT SEVEN INTO CARRY
007A DB         =1740         XRL     A,KEY
007B C693       =1741         JZ      MAIND
007D F687       =1742         JC      MAIND1
                =1743         MINC    TYPE
007F B937       =1748+          MOV     R1,#TYPE
0081 F1         =1749+          MOV     A,@R1
0082 17         =1753+          INC     A
0083 A1         =1758+          MOV     @R1,A
0084 1C         =1761         INC     ITMP
0085 0475       =1762         JMP     MAINC1
                =1763 ;
                =1764 ;       MODIFIER NOT FOUND SO RESET TYPE INDEX TO DEFAULT CASE (ZERO).
                =1765 ;
                =1766 MAIND1: MMOV    TYPE,ZERO
0087 B937       =1777+          MOV     R1,#TYPE
0089 B100       =1778+          MOV     @R1,#ZERO
                =1782         MMOV    A,OPTION
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

008B B939       =1791+          MOV     R1,#OPTION
008D F1         =1792+          MOV     A,@R1
008E E3         =1796         MOVP3   A,@A
008F 3404       =1797         CALL    OUTMSG
0091 049E       =1798         JMP     MAINB0
                =1799 ;
                =1800 ;       CALL OUTPUT_MESSAGE(MODIFIER)
                =1801 MAIND:  MMOV    A,OPTION
0093 B939       =1810+          MOV     R1,#OPTION
0095 F1         =1811+          MOV     A,@R1
0096 E3         =1815         MOVP3   A,@A
                =1816         MADD    A,TYPE
0097 B937       =1822+          MOV     R1,#TYPE
0099 61         =1823+          ADD     A,@R1
009A 3404       =1827         CALL    OUTMSG
009C 14EC       =1828         CALL    INPKEY
                =1829 ;
009E BC00       =1830 MAINB0: MOV     ITMP,#0
00A0 2330       =1831 MAINB1: MOV     A,#SMALO
00A2 6C         =1832         ADD     A,ITMP
00A3 6C         =1833         ADD     A,ITMP
00A4 A8         =1834         MOV     R0,A
00A5 14C0       =1835         CALL    INPADR
00A7 FGBA       =1836         JC      CMDINT
00A9 1C         =1837         INC     ITMP
00AA B938       =1838         MOV     R1,#NUMCON
00AC F1         =1839         MOV     A,@R1
00AD 07         =1840         DEC     A
00AE A1         =1841         MOV     @R1,A
00AF CGBA       =1842         JZ      CMDINT
00B1 FB         =1843         MOV     A,KEY
00B2 D313       =1844         XRL     A,#KEYEND
00B4 CGBA       =1845         JZ      CMDINT
00B6 14EC       =1846         CALL    INPKEY
00B8 04A0       =1847         JMP     MAINB1
                =1848 ;
                =1849 ;CMDINT ENTER THE COMMAND PROCESSOR WITH:
                =1850 ;       BASE_CODE=THE MAIN COMMAND TYPE
                =1851 ;       TYPE=SUBCOMMAND TYPE
                =1852 ;       PARAMETER(1)=FIRST ADDRESS
                =1853 ;       PARAMETER(2)=SECOND ADDRESS
                =1854 ;       PARAMETER(3)=DATA
00DA 4400       =1855 CMDINT: JMP     IMPLEM
                =1856 ;
                =1857 ;MERROR ERROR ENCOUNTERED IN MAIN PARSING ROUTINE.
00BC BA01       =1858 MERROR: MOV     LDATA,#1
00BE 249A       =1859         JMP     PERROR
                =1860         SIZECHK
0097            =1863+  SIZE  SET   151
                =1864+;
                =1865+;*************************************************************
                =1874 $EJECT
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

                =1875         DATABLK 50
0323            =1880+          ORG     803
                =1884 ;
                =1885 ;*********************************************************
                =1886 ;
                =1887 ;       TABLES FOR PARSER
                =1888 ;
                =1889 ;*********************************************************
                =1890 ;
                =1891 ;       THE CTAB TABLE CONTAINS <COMSIZ> ENTRIES FOR EACH COMMAND.  THE MEANING
                =1892 ;       OF THE ENTRIES IS AS FOLLOWS:
                =1893 ;
                =1894 ;       ENTRY 0.  COMMAND KEY TO INITIATE
                =1895 ;       ENTRY 1.  POINTER TO THE LIST OF OPTIONS APPLICABLE TO THIS COMMAND
                =1896 ;       ENTRY 2.  NUMBER OF NUMERIC PARAMETERS REQUIRED BY THE COMMAND
                =1897 ;
0023            =1898 CTAB    EQU     $ AND 0FFH
0003            =1899 COMSIZ  EQU     3
                =1900 ;
0323 1F         =1901         DB      KEYMOD,LOW OPTAB1,1     ;EXAM
0324 3F         =
0325 01         =
0326 1E         =1902         DB      KEYGO,LOW OPTAB3,1      ;GO
0327 49         =
0328 01         =
0329 10         =1903         DB      KEYFIL,LOW OPTAB1,3     ;FILL
032A 3F         =
032B 03         =
032C 1C         =1904         DB      KEYLST,LOW OPTAB1,2     ;DUMP
032D 3F         =
032E 02         =
032F 18         =1905         DB      KEYREC,LOW OPTAB1,2     ;RECORD
0330 3F         =
0331 02         =
0332 14         =1906         DB      KEYREL,LOW OPTAB1,0     ;RELOAD
0333 3F         =
0334 00         =
0335 0C         =1907         DB      KSETB,LOW OPTAB2,1      ;SETBRK
0336 46         =
0337 01         =
0338 0C         =1908         DB      KCLRB,LOW OPTAB2,1      ;CLRBRK
0339 46         =
033A 01         =
033B 1D         =1909         DB      KGORES,LOW OPTAB3,0     ;GO FROM RESET STATE
033C 49         =
033D 00         =
033E FF         =1910         DB      0FFH                    ;ESCOP
                =1911 ;
                =1912 $EJECT
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

                =1913 ;
                =1914 ;       THE OPTION TABLE GIVES THE VARIOUS OPTIONS ALLOWED FOR EACH
                =1915 ;       BASIC COMMAND, AS FOLLOWS:
                =1916 ;
                =1917 ;       ENTRY 0.  START OF TABLE OF MODIFIER RESPONSES.
                =1918 ;       ENTRY 1+.  ALLOWED MODIFIER KEYSTROKES CORRESPONDING TO OPTIONS 0-5.
                =1919 ;              NOTE THAT THE LAST BYTE IN EACH OPTION GROUP HAS BIT
                =1920 ;              SEVEN SET TO INDICATE THE END.
                =1921 ;
033F 26         =1922 OPTAB1: DB     STRMEM
0340 1A         =1923         DB     KEYPM,KEYDM,KEYREG,RINT
0341 16         =
0342 1B         =
0343 11         =
0344 19         =1924         DB     PBRK,DBRK OR 80H
0345 95         =
0346 26         =1925 OPTAB2: DB     STRMEM
0347 1A         =1926         DB     KEYPM,KEYDM OR 80H
0348 96         =
0349 2C         =1927 OPTAB3: DB     STRGOC
034A 1B         =1928         DB     NOBRK,WBRK,SING
034B 16         =
034C 1A         =
034D 15         =1929         DB     KEYPAT,KEYTRA OR 80H
034E 99         =
                =1930         SIZECHK
002C            =1933+  SIZE  SET   44
                =1934+;
                =1935+;*******************************************************
                =1944 $EJECT
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

                =1945         CODEBLK 130
0100            =1955+          ORG     256
                =1959 ;OUTUTL OUTPUT ONE OF FOUR UTILITY DISPLAY PROMPTS (LEFT JUSTIFIED)
                =1960 ;       ACCORDING TO ACC CONTENTS (0-3).
                =1961 ;OUTCLR CLEAR DISPLAY AND OUTPUT CHARACTER STRING STARTING
                =1962 ;       AT THE ADDRESS POINTED TO BY BYTE AT ADDRESS IN ACCUMULATOR.
                =1963 ;OUTMSG SUBROUTINE TO COPY A STRING OF BIT PATTERNS FROM ROM TO THE
                =1964 ;       DISPLAY REGISTERS.
                =1965 ;       STRING SELECTED IS DETERMINED BY ACC WHEN CALLED.
                =1966 ;       ON ENTERING OUTMSG, ACC CONTENTS ARE USED TO ADDRESS A BYTE IN A
                =1967 ;       LOOKUP TABLE ON THE CURRENT PAGE WHICH CONTAINS THE ADDRESS OF
                =1968 ;       A STRING OF SEGMENT PATTERN DATA BYTES TO BE PRINTED ONTO THE
                =1969 ;       DISPLAY.
                =1970 ;       THE END OF THE STRING IS INDICATED WHEN BIT7 =1.
                =1971 ;       CALLS SUBROUTINE 'WDISP'
                =1972 ;       TO ACTUALLY EFFECT WRITING INTO THE DISPLAY REGISTERS.
0100 0319       =1973 OUTUTL: ADD     A,#STRUTL
0102 B4F1       =1974 OUTCLR: CALL    CLEAR
0104 A3         =1975 OUTMSG: MOVP    A,@A
                =1976         MMOV    STRTMP,A
0105 B940       =1989+          MOV     R1,#STRTMP
0107 A1         =1990+          MOV     @R1,A
                =1994 PRNT2:  MMOV    A,STRTMP        ;LOAD NEXT CHARACTER LOCATION
0108 B940       =2003+          MOV     R1,#STRTMP
010A F1         =2004+          MOV     A,@R1
010B A3         =2008         MOVP    A,@A            ;LOAD BIT PATTERN INDIRECT
010C F217       =2009         JB7     PRNT1
010E D4D8       =2010         CALL    WDISP           ;OUTPUT TO NEXT CHARACTER POSITION
                =2011         MINC    STRTMP          ;INDEX POINTER
0110 B940       =2016+          MOV     R1,#STRTMP
0112 F1         =2017+          MOV     A,@R1
0113 17         =2021+          INC     A
0114 A1         =2026+          MOV     @R1,A
0115 2408       =2029         JMP     PRNT2
0117 C4D8       =2030 PRNT1:  JMP     WDISP           ;DONE
                =2031 ;
0019            =2032 STRUTL  EQU     LOW $
0119 31         =2033         DB      LOW(DERROR)     ;UTILITY MESSAGE 0 ADDRESS
011A 37         =2034         DB      LOW(DSGNON)     ;UTILITY MESSAGE 1 ADDRESS
011B 3E         =2035         DB      LOW(DRUN)       ;UTILITY MESSAGE 2 ADDRESS
011C 44         =2036         DB      LOW(DBPNT)      ;UTILITY MESSAGE 3 ADDRESS
001D            =2037 STRCOM  EQU     LOW $
011D 46         =2038         DB      LOW(DMOD)       ;BASIC COMMAND 0 RESPONSE ADDRESS
011E 49         =2039         DB      LOW(DGO)        ;BASIC COMMAND 1 RESPONSE ADDRESS
011F 4B         =2040         DB      LOW(DFILL)      ;BASIC COMMAND 2 RESPONSE ADDRESS
0120 4E         =2041         DB      LOW(DLST)       ;BASIC COMMAND 3 RESPONSE ADDRESS
0121 51         =2042         DB      LOW(DREC)       ;BASIC COMMAND 4 RESPONSE ADDRESS
0122 54         =2043         DB      LOW(DREL)       ;BASIC COMMAND 5 RESPONSE ADDRESS
0123 57         =2044         DB      LOW(DSB)        ;BASIC COMMAND 6 RESPONSE ADDRESS
0124 5A         =2045         DB      LOW(DCB)        ;BASIC COMMAND 7 RESPONSE ADDRESS
0125 5D         =2046         DB      LOW(DGR)        ;BASIC COMMAND 8 RESPONSE ADDRESS
0026            =2047 STRMEM  EQU     LOW $
0126 5F         =2048         DB      LOW(DPRMEM)     ;DATA TYPE MODIFIER 0 RESPONSE ADDRESS
0127 61         =2049         DB      LOW(DDAMEM)     ;DATA TYPE MODIFIER 1 RESPONSE ADDRESS
0128 63         =2050         DB      LOW(DRM)        ;DATA TYPE MODIFIER 2 RESPONSE ADDRESS
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

0129 69         =2051          DB      LOW(DINTRG)     ;DATA TYPE MODIFIER 3 RESPONSE ADDRESS
012A 65         =2052          DB      LOW(DPRBRK)     ;DATA TYPE MODIFIER 4 RESPONSE ADDRESS
012B 67         =2053          DB      LOW(DDABRK)     ;DATA TYPE MODIFIER 5 RESPONSE ADDRESS
002C            =2054 STRGOC   EQU     LOW $
012C 6B         =2055          DB      LOW(DNOBRK)     ;EXECUTION MODE MODIFIER 0
012D 6D         =2056          DB      LOW(DWBRK)      ;EXECUTION MODE MODIFIER 1
012E 6F         =2057          DB      LOW(DSS)        ;EXECUTION MODE MODIFIER 2
012F 72         =2058          DB      LOW(DPA)        ;EXECUTION MODE MODIFIER 3
0130 75         =2059          DB      LOW(DTR)        ;EXECUTION MODE MODIFIER 4
                =2060 ;
                =2061 ;        UTILITY OUTPUT MESSAGES
                =2062 ;
                =2063 DERROR:
0131 79         =2064          DB      01111001B       ;"E"
0132 50         =2065          DB      01010000B       ;"R"
0133 50         =2066          DB      01010000B       ;"R"
0134 5C         =2067          DB      01011100B       ;"O"
0135 50         =2068          DB      01010000B       ;"R"
0136 C0         =2069          DB      11000000B       ;"-."
                =2070 DSGNON:
0137 00         =2071          DB      00000000B       ;" "
0138 76         =2072          DB      01110110B       ;"H"
0139 6D         =2073          DB      01101101B       ;"S"
013A 79         =2074          DB      01111001B       ;"E"
013B 40         =2075          DB      01000000B       ;"-"
013C 66         =2076          DB      01100110B       ;"4"
013D E7         =2077          DB      11100111B       ;"9."(TM)
                =2078 DRUN:
013E 00         =2079          DB      00000000B       ;" "
013F 40         =2080          DB      01000000B       ;"-"
0140 50         =2081          DB      01010000B       ;"R"
0141 1C         =2082          DB      00011100B       ;"U"
0142 54         =2083          DB      01010100B       ;"N"
0143 C0         =2084          DB      11000000B       ;"-."
                =2085 DBPNT:
0144 73         =2086          DB      01110011B       ;"P"
0145 B9         =2087          DB      10111001B       ;"C."
                =2088 $EJECT
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

                =2089 ;
                =2090 ;       PRIMARY COMMAND RESPONSE STRING PATTERNS
                =2091 ;
                =2092 DMOD:
0146 79         =2093         DB      01111001B,00111001B,11110100B   ;"ECH."
0147 39         =
0148 F4         =
                =2094 DGO:
0149 3D         =2095         DB      00111101B,11011100B             ;"GO."
014A DC         =
                =2096 DFILL:
014B 71         =2097         DB      01110001B,00110000B,10111000B   ;"FIL."
014C 30         =
014D B8         =
                =2098 DLST:
014E 38         =2099         DB      00111000B,01101101B,11111000B   ;"LST."
014F 6D         =
0150 F8         =
                =2100 DREC:
0151 3E         =2101         DB      00111110B,01110011B,10111000B   ;"UPL."
0152 73         =
0153 B8         =
                =2102 DREL:
0154 5E         =2103         DB      01011110B,01010100B,10111000B   ;"DNL."
0155 54         =
0156 B8         =
                =2104 DSB:
0157 6D         =2105         DB      01101101B,01111000B,11111100B   ;"STB."
0158 78         =
0159 FC         =
                =2106 DCB:
015A 39         =2107         DB      00111001B,00111000B,11111100B   ;"CLB."
015B 38         =
015C FC         =
                =2108 DGR:
015D 3D         =2109         DB      00111101B,11010000B             ;"GR."
015E D0         =
                =2110 $EJECT
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

                =2111 ;
                =2112 ;       MEMORY SPACE MODIFIER OPTION RESPONSE STRINGS:
                =2113 ;
                =2114 DPRMEM:
015F 73         =2115         DB      01110011B,11010000B             ;"PR."
0160 D0         =
                =2116 DDAMEM:
0161 5E         =2117         DB      01011110B,11101111B             ;"DA."
0162 F7         =
                =2118 DRM:
0163 50         =2119         DB      01010000B,10111101B             ;"RG."
0164 BD         =
                =2120 DPRBRK:
0165 73         =2121         DB      01110011B,11111100B             ;"PB."
0166 FC         =
                =2122 DDABRK:
0167 5E         =2123         DB      01011110B,11111100B             ;"DB."
0168 FC         =
                =2124 DINTRG:
0169 76         =2125         DB      01110110B,11010000B             ;"HR."
016A D0         =
                =2126 ;
                =2127 ;       RESPONSE MESSAGES FOR GO CONDITION MODIFIERS.
                =2128 ;
                =2129 DNOBRK:
016B 54         =2130         DB              01010100B,11111100B     ;"NB."
016C FC         =
                =2131 DWBRK:
016D 7C         =2132         DB      01111100B,11010000B             ;"BR."
016E D0         =
                =2133 DSS:
016F 6D         =2134         DB      01101101B,01101101B,11111000B   ;"SST."
0170 6D         =
0171 F8         =
                =2135 DPA:
0172 77         =2136         DB      01110111B,01111100B,11010000B   ;"ABR."
0173 7C         =
0174 D0         =
                =2137 DTR:
0175 77         =2138         DB      01110111B,01101101B,11111000B   ;"AST."
0176 6D         =
0177 F8         =
                =2139 ;
                =2140         SIZECHK
0078            =2143+  SIZE  SET   120
                =2144+;
                =2145+;*************************************************************
                =2154 $EJECT
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

                =2155          CODEBLK 45
00C0            =2160+           ORG     192
                =2164 ;INPADR INPUT DATA INTO TWO-BYTE PARAMETER BUFFER INDICATED BY R0.
                =2165 ;        RECEIVE NUMERIC KEYS FROM KEYBOARD UNTIL ',' OR '.'.
                =2166 ;        SHIFT INTO ADDRESS BUFFER;
                =2167 ;        RE-WRITE DISPLAY.
                =2168 ;        IF NUMBER OF CONSTANTS NEEDED IS ZERO, NO NEW PARAMETERS ARE ALLOWED.
                =2169 ;
00C0 97         =2170 INPADR: CLR     C
00C1 A7         =2171         CPL     C
                =2172         MMOV    A,NUMCON
00C2 B938       =2181+          MOV     R1,#NUMCON
00C4 F1         =2182+          MOV     A,@R1
00C5 C6D7       =2186         JZ      ELSIF1
00C7 FB         =2187 INPAD1: MOV     A,KEY
00C8 92D7       =2188         JB4     ELSIF1
00CA 20         =2189         XCH     A,@R0
00CB 47         =2190         SWAP    A
00CC 20         =2191         XCH     A,@R0
00CD 30         =2192         XCHD    A,@R0
00CE 18         =2193         INC     R0
00CF 30         =2194         XCHD    A,@R0
00D0 3478       =2195         CALL    UPDADR
00D2 14EC       =2196         CALL    INPKEY
00D4 97         =2197         CLR     C
00D5 04C7       =2198         JMP     INPAD1
                =2199 ;
                =2200 ;ELSIF1 IF KEY=',' OR '.' THEN RETURN.
                =2201 ;
00D7 FB         =2202 ELSIF1: MOV     A,KEY
00D8 D312       =2203         XRL     A,#KEYNXT
00DA C6E5       =2204         JZ      ELSIF2
00DC FB         =2205         MOV     A,KEY
00DD D313       =2206         XRL     A,#KEYEND
00DF C6E5       =2207         JZ      ELSIF2
                =2208 ;
                =2209 ;        ELSE GOTO PERROR.
                =2210 ;
00E1 BA02       =2211         MOV     LDATA,#2
00E3 249A       =2212         JMP     PERROR
00E5 B846       =2213 ELSIF2: MOV     R0,#SEGMAP
00E7 B903       =2214         MOV     R1,#3
00E9 B4F5       =2215         CALL    DBLANK
00EB 83         =2216         RET
                =2217         SIZECHK
002C            =2220+  SIZE  SET   44
                =2221+;
                =2222+;*******************************************************
                =2231 $EJECT
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

                =2232          CODEBLK 35
0178            =2242+           ORG     376
                =2246 ;UPDADR UPDATE ADDRESS FIELD
                =2247 ;        (LAST THREE CHARACTERS OF DISPLAY) WITH ADDRESS BUFFER
                =2248 UPDADR: MMOV    NEXTPL,PLUS3
0178 B93A       =2259+           MOV     R1,#NEXTPL
017A B103       =2260+           MOV     @R1,#PLUS3
                =2264 ;        WRITE ADDR INTO NEXT THREE BUFFER LOCATIONS.
017C F0         =2265 UPDAD1: MOV     A,@R0
017D C8         =2266          DEC     R0
017E 530F       =2267          ANL     A,#0FH
0180 968E       =2268          JNZ     DSPHI
0182 D4D8       =2269          CALL    WDISP
0184 F0         =2270          MOV     A,@R0
0185 47         =2271          SWAP    A
0186 530F       =2272          ANL     A,#0FH
0188 9692       =2273          JNZ     DSPM1
018A D4D8       =2274          CALL    WDISP
018C 2494       =2275          JMP     DSPLO
018E D4D3       =2276 DSPHI:  CALL    DSPACC
0190 F0         =2277 DSPMID: MOV     A,@R0
0191 47         =2278          SWAP    A
0192 D4D3       =2279 DSPM1:  CALL    DSPACC
0194 F0         =2280 DSPLO:  MOV     A,@R0
0195 D4D3       =2281          CALL    DSPACC
0197 83         =2282          RET
                =2283          SIZECHK
0020            =2286+  SIZE  SET   32
                =2287+;
                =2288+;*****************************************************
                =2297 $EJECT
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

                =2298          CODEBLK 35
0198            =2308+           ORG     408
                =2312 ;PERROR:         REPEAT
                =2313 ;            OUTPUT_MESSAGE(PERROR_PROMPT)
                =2314 ;            OUTPUT(LDATA)
                =2315 ;            CALL INPUT_BYTE(KEY)
                =2316 ;          UNTIL KEY='CLEAR/PREVIOUS'
0198 BA04       =2317 RERROR: MOV     LDATA,#4
019A BF02       =2318 PERROR: MOV     XPCODE,#2
019C 74D1       =2319         CALL    XPTEST
019E 27         =2320         CLR     A
019F D7         =2321         MOV     PSW,A
01A0 FB         =2322         MOV     A,KEY
01A1 D317       =2323         XRL     A,#KEYCLR
01A3 C6D6       =2324         JZ      ERROR2
01A5 27         =2325         CLR     A
01A6 3400       =2326         CALL    OUTUTL
01A8 FA         =2327         MOV     A,LDATA
01A9 D4D3       =2328         CALL    DSPACC
                =2329         MMOV    KBDBUF,NEG1
01AB B93B       =2340+          MOV     R1,#KBDBUF
01AD B1FF       =2341+          MOV     @R1,#NEG1
01AF 14EC       =2345         CALL    INPKEY
01B1 FB         =2346         MOV     A,KEY
01B2 D313       =2347         XRL     A,#KEYEND
01B4 9698       =2348         JNZ     RERROR
01B6 0429       =2349 ERROR2: JMP     MAIN
                =2350         SIZECHK
0020            =2353+  SIZE  SET   32
                =2354+;
                =2355+;*************************************************************
                =2364 ;
                =2365         CODEBLK 80
0200            =2380+          ORG     512
                =2384 ;IMPLEM IMPLEMENT COMMAND
0200 2306       =2385 IMPLEM: MOV     A,#LOW(JMPTBL)
                =2386         MADD    A,BCODE
0202 B936       =2392+          MOV     R1,#BCODE
0204 61         =2393+          ADD     A,@R1
0205 B3         =2397         JMPP    @A
                =2398 ;
                =2399 JMPTBL:
0206 0F         =2400         DB      LOW(JTOMOD)
0207 20         =2401         DB      LOW(JTOGO)
0208 22         =2402         DB      LOW(JTOFIL)
0209 1A         =2403         DB      LOW(JTOLST)
020A 11         =2404         DB      LOW(JTOREC)
020B 16         =2405         DB      LOW(JTOREL)
020C 2C         =2406         DB      LOW(COMSBR)
020D 28         =2407         DB      LOW(COMCBR)
020E 26         =2408         DB      LOW(JGORES)
                =2409 ;
020F 444F       =2410 JTOMOD: JMP     EXAMIN
                =2411 ;
0211 85         =2412 JTOREC: CLR     F0      ;F0=0 ==> HEX FORMAT DATA DUMP
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

0212 B472       =2413          CALL    HFILEO
0214 0429       =2414          JMP     MAIN
                =2415 ;
0216 5497       =2416 JTOREL:  CALL    HRECIN
0218 0429       =2417          JMP     MAIN
                =2418 ;
021A 85         =2419 JTOLST:  CLR     F0
021B 95         =2420          CPL     F0
021C B472       =2421          CALL    HFILEO
021E 0429       =2422          JMP     MAIN
                =2423 ;
0220 8400       =2424 JTOGO:   JMP     EPRUN
                =2425 ;
0222 54E5       =2426 JTOFIL:  CALL    COMFIL
0224 0429       =2427          JMP     MAIN
                =2428 ;
0226 8461       =2429 JGORES:  JMP     COMGOR
                =2430 ;
                =2431 ;COMCBR COMMAND TO CLEAR BREAKPOINTS
0228 BA00       =2432 COMCBR:  MOV     LDATA,#0
022A 442E       =2433          JMP     BRKFIL
                =2434 ;
                =2435 ;COMSBR COMMAND TO SET BREAKPOINTS
022C BA01       =2436 COMSBR:  MOV     LDATA,#1
022E 2304       =2437 BRKFIL:  MOV     A,#4
                =2438          MADD    TYPE,A
0230 B937       =2448+           MOV     R1,#TYPE
0232 61         =2449+           ADD     A,@R1
0233 A1         =2455+           MOV     @R1,A
0234 F400       =2459 BRKNXT:  CALL    LSTORE
0236 FB         =2460          MOV     A,KEY
0237 D313       =2461          XRL     A,#KEYEND
0239 C64D       =2462          JZ      BRKEND
023B 14EC       =2463          CALL    INPKEY
                =2464          MMOV    NUMCON,PLUS1
023D B938       =2475+           MOV     R1,#NUMCON
023F B101       =2476+           MOV     @R1,#PLUS1
0241 B830       =2480          MOV     R0,#SMALO
0243 B000       =2481          MOV     @R0,#0
                =2482          MMOV    SMAHI,ZERO
0245 B931       =2493+           MOV     R1,#SMAHI
0247 B100       =2494+           MOV     @R1,#ZERO
0249 14C0       =2498          CALL    INPADR
024B E634       =2499          JNC     BRKNXT
024D 0429       =2500 BRKEND:  JMP     MAIN
                =2501          SIZECHK
004F            =2504+  SIZE   SET   79
                =2505+;
                =2506+;*******************************************************
                =2515 $EJECT
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

                =2516        CODEBLK 75
024F            =2531+         ORG     591
                =2535 ;EXAMIN EXAMINE/MODIFY MEMORY COMMAND.
                =2536 ;       DISPLAYS MEMORY ADDRESS SPACE OPTION, ADDRESS VALUE, AND CURRENT DATA.
                =2537 ;       READS KEYBOARD AND INTERPRETS RESPONSE.
                =2538 ;
                =2539 ;       OUTPUT_MESSAGE(<MEMORY_SPACE_OPTION><SMA>'='<DATA_BYTE>)
024F 85         =2540 EXAMIN: CLR     F0
                =2541 EXAM0:  MMOV    A,TYPE
0250 B937       =2550+          MOV     R1,#TYPE
0252 F1         =2551+          MOV     A,@R1
0253 0326       =2555         ADD     A,#STRMEM       ;OFFSET FOR FIRST MEMORY TYPE STRING
0255 3402       =2556         CALL    OUTCLR
0257 B831       =2557         MOV     R0,#SMALO+1
0259 347C       =2558         CALL    UPDAD1
025B 2348       =2559         MOV     A,#01001000B    ;'='
025D D4D8       =2560         CALL    WDISP
025F 14FC       =2561         CALL    LFETCH
0261 FA         =2562         MOV     A,LDATA
0262 47         =2563         SWAP    A
0263 D4D3       =2564         CALL    DSPACC
0265 FA         =2565         MOV     A,LDATA
0266 D4D3       =2566         CALL    DSPACC
                =2567 ;
                =2568 ;
                =2569 ;       INPUT_KEY(KEY)
                =2570 ;       IF (KEY IS NOT NUMERIC)
                =2571 ;               IF (KEY=KEYEND) GO TO PARSER
                =2572 ;               ELSEIF (KEY=KEYNEXT)
                =2573 ;                       INCREMENT (SMA)
                =2574 ;                       GOTO EXAMIN
                =2575 ;               ELSEIF (KEY=KEYPREVIOUS)
                =2576 ;                       DECREMENT (SMA)
                =2577 ;                       GOTO EXAMIN
                =2578 ;               ELSE GOTO PERROR
                =2579 ;
0268 14EC       =2580         CALL    INPKEY
                =2581         MMOV    A,KEY
026A FB         =2597+          MOV     A,KEY
026B 927B       =2601         JB4     EXAM1
                =2602 ;
                =2603 ;       APPEND DATA WITH (LOWNIB_<KEY>)
                =2604 ;       CALL LSTORE
                =2605 ;       GOTO EXAMIN
                =2606 ;
026D FA         =2607         MOV     A,LDATA
026E 47         =2608         SWAP    A
026F 53F0       =2609         ANL     A,#0F0H
0271 B675       =2610         JF0     EXAM5
0273 27         =2611         CLR     A
0274 95         =2612         CPL     F0
0275 6B         =2613 EXAM5:  ADD     A,KEY
0276 AA         =2614         MOV     LDATA,A
0277 F400       =2615         CALL    LSTORE
0279 4450       =2616         JMP     EXAM0
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

                =2617 ;
027B D313       =2618 EXAM1:  XRL     A,#(KEYEND)
027D 9681       =2619         JNZ     EXAM2
027F 0429       =2620         JMP     MAIN
                =2621 ;
0281 FB         =2622 EXAM2:  MOV     A,KEY
0282 D312       =2623         XRL     A,#KEYNXT
0284 968A       =2624         JNZ     EXAM3
0286 34F2       =2625         CALL    INCSMA
0288 444F       =2626         JMP     EXAMIN
028A FB         =2627 EXAM3:  MOV     A,KEY
028B D317       =2628         XRL     A,#KEYCLR
028D 9693       =2629         JNZ     EXAM4
028F 54F4       =2630         CALL    DECSMA
0291 444F       =2631         JMP     EXAMIN
0293 B803       =2632 EXAM4:  MOV     LDATA,#03H
0295 249A       =2633         JMP     PERROR
                =2634         SIZECHK
0048            =2637+  SIZE  SET   72
                =2638+;
                =2639+;*******************************************************
                =2648 ;
                =2649         CODEBLK 4
00EC            =2654+          ORG     236
00EC D4C2       =2658 INPKEY: CALL    KBDIN   ; RETURNS KEY DEPRESSION IN A
00EE A8         =2659         MOV     KEY,A
00EF 83         =2660         RET
                =2661         SIZECHK
0004            =2664+  SIZE  SET   4
                =2665+;
                =2666+;*******************************************************
                =2675 $EJECT
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

                 2676 $      INCLUDE(:F0:GOCOMS.MOD)
                =2677        CODEBLK 210
0400            =2697+         ORG     1024
                =2701 ;EPRUN RUN EMULATION MODE.
                =2702 ;      RELOAD EP WITH SYSTEM STATUS AND RELEASE.
                =2703 ;      SEQUENCE IS AS FOLLOWS:
                =2704 ;      IF COMMAND WAS TERMINATED BY THE 'NEXT' KEY:
                =2705 ;         STORE SMA INTO EP PC;
                =2706 ;      STORE EP PC INTO TOP-OF-STACK (RELATIVE TO EP PSW);
                =2707 ;      PASS EP R0;
                =2708 ;      PASS EP PSW;
                =2709 ;      PASS EP TIMER;
                =2710 ;      PASS EP ACCUMULATOR;
                =2711 ;
0400 2302       =2712 EPRUN: MOV     A,#2
0402 3400       =2713        CALL    OUTUTL
                =2714        MMOV    A,NUMCON
0404 B938       =2723+         MOV     R1,#NUMCON
0406 F1         =2724+         MOV     A,@R1
0407 9615       =2728        JNZ     EPCONT
                =2729        MMOV    EPPCLO,SMALO
0409 B930       =2745+         MOV     R1,#SMALO
040B F1         =2746+         MOV     A,@R1
040C B924       =2752+         MOV     R1,#EPPCLO
040E A1         =2753+         MOV     @R1,A
                =2756        MMOV    EPPCHI,SMAHI
040F B931       =2772+         MOV     R1,#SMAHI
0411 F1         =2773+         MOV     A,@R1
0412 B925       =2779+         MOV     R1,#EPPCHI
0414 A1         =2780+         MOV     @R1,A
0415 FB         =2783 EPCONT: MOV     A,KEY
0416 D312       =2784        XRL     A,#KEYNXT
0418 C61F       =2785        JZ      EPCON1
041A 2301       =2786        MOV     A,#01H  ;STACK ONE LEVEL DEEP TO HOLD USER STARTING ADDRESS
                =2787        MMOV    EPPSW,A
041C B921       =2800+         MOV     R1,#EPPSW
041E A1         =2801+         MOV     @R1,A
                =2805 EPCON1: MMOV    LDATA,EPPCLO
041F B924       =2821+         MOV     R1,#EPPCLO
0421 F1         =2822+         MOV     A,@R1
0422 AA         =2835+         MOV     LDATA,A
                =2838        MMOV    A,EPPSW
0423 B921       =2847+         MOV     R1,#EPPSW
0425 F1         =2848+         MOV     A,@R1
0426 07         =2852        DEC     A
0427 5307       =2853        ANL     A,#07H
0429 E7         =2854        RL      A
042A 0308       =2855        ADD     A,#08H
                =2856        MMOV    SMALO,A
042C B930       =2869+         MOV     R1,#SMALO
042E A1         =2870+         MOV     @R1,A
042F F4C3       =2874        CALL    EPSTOR
                =2875        MINC    SMALO
0431 B930       =2880+         MOV     R1,#SMALO
0433 F1         =2881+         MOV     A,@R1
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

0434 17         =2885+          INC     A
0435 A1         =2890+          MOV     @R1,A
                =2893           MMOV    A,EPPSW
0436 B921       =2902+          MOV     R1,#EPPSW
0438 F1         =2903+          MOV     A,@R1
0439 53F0       =2907           ANL     A,#0F0H
                =2908           MORL    A,EPPCHI
043B B925       =2914+          MOV     R1,#EPPCHI
043D 41         =2915+          ORL     A,@R1
043E AA         =2919           MOV     LDATA,A
043F F4C3       =2920           CALL    EPSTOR
0441 B8D1       =2921 EPCNT:    MOV     R0,#LOW(OV2BAS+OVSIZE)
0443 746A       =2922           CALL    OVLOAD
                =2923           MMOV    A,EPR0
0445 B923       =2932+          MOV     R1,#EPR0
0447 F1         =2933+          MOV     A,@R1
0448 F4D0       =2937           CALL    EPPASS
                =2938           MMOV    A,EPPSW
044A B921       =2947+          MOV     R1,#EPPSW
044C F1         =2948+          MOV     A,@R1
044D F4D0       =2952           CALL    EPPASS
                =2953           MMOV    A,EPTIMR
044F B922       =2962+          MOV     R1,#EPTIMR
0451 F1         =2963+          MOV     A,@R1
0452 F4D0       =2967           CALL    EPPASS
                =2968           MMOV    A,EPACC
0454 B920       =2977+          MOV     R1,#EPACC
0456 F1         =2978+          MOV     A,@R1
0457 F4D0       =2982           CALL    EPPASS
0459 8903       =2983           ORL     P1,#000000011B
045B F4DB       =2984           CALL    EPSTEP
045D 745A       =2985           CALL    OVSWAP
045F 846B       =2986           JMP     CGO
                =2987 ;
                =2988 ;COMGOR GO FROM RESET COMMAND
                =2989 ;       RESET PROCESSOR
                =2990 ;       RELOAD LOW ORDER PROGRAM BYTES INTO PROGRAM MEMORY
                =2991 ;
0461 2302       =2992 COMGOR:   MOV     A,#2
0463 3400       =2993           CALL    OUTUTL
0465 8910       =2994           ORL     P1,#EPRSET
0467 745A       =2995           CALL    OVSWAP
0469 99EF       =2996           ANL     P1,#(NOT EPRSET)
                =2997 ;
                =2998 ;
                =2999 ;CGO    SET UP BREAK LOGIC FOR APPROPRIATE BREAK CONDITIONS,
                =3000 ;       DEPENDING ON CONTENTS OF 'TYPE'.
                =3001 ;
                =3002 CGO:      MMOV    A,TYPE
046B B937       =3011+          MOV     R1,#TYPE
046D F1         =3012+          MOV     A,@R1
046E 0371       =3016           ADD     A,#LOW GOTBL
0470 B3         =3017           JMPP    @A
                =3018 ;
0471 7C         =3019 GOTBL:    DB      LOW(CGONB)
```

```
LOC  OBJ       LINE       SOURCE STATEMENT

0472 76        =3020          DB      LOW(CGOWB)
0473 80        =3021          DB      LOW(CGOSS)
0474 76        =3022          DB      LOW(CGOPAT)
0475 80        =3023          DB      LOW(CGOTRA)
               =3024 ;
               =3025 CGOPAT:
0476 99FD      =3026 CGOWB:   ANL     P1,#NOT 00000010B
0478 8901      =3027          ORL     P1,#00000001B
047A 8482      =3028          JMP     EPRUN4
               =3029 ;
047C 99FC      =3030 CGOWB:   ANL     P1,#NOT 00000011B
047E 8482      =3031          JMP     EPRUN4
               =3032 ;
               =3033 CGOTRA:
0480 8903      =3034 CGOSS:   ORL     P1,#00000011B
               =3035 ;
               =3036 ;EPRUN4 SET UP CONTROL LOGIC TO RUN USER'S PROGRAM.
               =3037 ;       RELEASE PROCESSOR TO RUN.
               =3038 ;
0482 8A20      =3039 EPRUN4:  ORL     P2,#00100000B           ;DISABLE EP LINK REFERENCES.
0484 9AEF      =3040          ANL     P2,#NOT 00010000B       ;SET ALL REFERENCES TO RAM ARRAY.
0486 99DF      =3041          ANL     P1,#NOT MODOUT
0488 F4F4      =3042          CALL    EPREL
               =3043 ;
               =3044 ;       WAIT FOR KEYSTROKE INPUT OR HARDWARE BREAK TO OCCUR.
               =3045 ;
048A F4AC      =3046 EPRUN1:  CALL    TOFPOL
048C F4AF      =3047          CALL    KBDPOL
048E 37        =3048          CPL     A
048F F295      =3049          JB7     EPRUN3
0491 8699      =3050          JNI     EPRUN2
0493 848A      =3051          JMP     EPRUN1
               =3052 ;
               =3053 ;EPRUN3 A KEYSTROKE WAS DETECTED WHILE EP WAS RUNNING.
               =3054 ;       BREAK EXECUTION,
               =3055 ;       PROCESS KEYSTROKE.
0495 B400      =3056 EPRUN3:  CALL    STSAVE
0497 84B3      =3057          JMP     EPRUN5
               =3058 ;
               =3059 ;EPRUN2 AN ENABLED BREAK CONDITION OCCURRED.
               =3060 ;       BREAK EMULATION MODE,
               =3061 ;       CONTINUE ACCORDING TO GO COMMAND TYPE.
0499 B400      =3062 EPRUN2:  CALL    STSAVE
               =3063          MMOV    A,TYPE
049B B937      =3072+           MOV     R1,#TYPE
049D F1        =3073+           MOV     A,@R1
049E 03A1      =3077          ADD     A,#LOW CNTTBL
04A0 B3        =3078          JMPP    @A
               =3079 ;
04A1 A6        =3080 CNTTBL:  DB      LOW(BRKERR)
04A2 BA        =3081          DB      LOW(EPRUN6)
04A3 BA        =3082          DB      LOW(EPRUN6)
04A4 AA        =3083          DB      LOW(CNTTRA)
04A5 AA        =3084          DB      LOW(CNTTRA)
               =3085 ;
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

                =3086 ;BRKERR BREAKPOINT LATCH WAS SET THOUGH BREAKPOINTS NOT ENABLED.
                =3087 ;       DISPLAY HARDWARE ERROR MESSAGE.
04A6 BA0B       =3088 BRKERR: MOV     LDATA,#0BH
04A8 249A       =3089         JMP     PERROR
                =3090 ;
                =3091 CNTTRA: MMOV    A,DSPTIM
04AA B928       =3100+          MOV     R1,#DSPTIM
04AC F1         =3101+          MOV     A,@R1
04AD 94F2       =3105         CALL    DELAY
04AF F4AF       =3106         CALL    KBDPOL
04B1 F241       =3107         JB7     EPCNT           ;B7 SET INDICATES NO KEYSTROKE.
                =3108 ;
                =3109 ;EPRUN5 INPUT(KEY),
                =3110 ;       IF KEY=END GO TO PARSER,
                =3111 ;       INPUT KEY,
                =3112 ;       IF KEY<>NEXT GO TO PARSER,
                =3113 ;       CONTINUE IN SAME MODE.
                =3114 ;
04B3 14EC       =3115 EPRUN5: CALL    INPKEY
04B5 FB         =3116         MOV     A,KEY
04B6 D313       =3117         XRL     A,#KEYEND
04B8 96C7       =3118         JNZ     EPRET
04BA 14EC       =3119 EPRUN6: CALL    INPKEY
04BC FB         =3120         MOV     A,KEY
04BD D312       =3121         XRL     A,#KEYNXT
04BF 96C7       =3122         JNZ     EPRET
04C1 2302       =3123         MOV     A,#2
04C3 3400       =3124         CALL    OUTUTL
04C5 8441       =3125         JMP     EPCNT
                =3126 ;
                =3127 ;EPRET  EXECUTION MODE IS TO BE TERMINATED.
                =3128 ;       JUMP INTO PARSER TO INTERPRET KEY ALREADY DETECTED.
04C7 0433       =3129 EPRET:  JMP     MAIN2
                =3130 ;
                =3131         SIZECHK
00C9            =3134+  SIZE  SET   201
                =3135+;
                =3136+;*******************************************************
                =3145 $EJECT
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

                =3146          CODEBLK 115
0500            =3171+           ORG     1200
                =3175 ;STSAVE EP STATUS SAVE SUBROUTINE.
                =3176 ;        FORCE CALL TO LOC 014H;
                =3177 ;        SAVE EP ACC;
                =3178 ;        SAVE EP TIMER;
                =3179 ;        SAVE EP PSW;
                =3180 ;        SAVE EP R0;
                =3181 ;        SAVE EP TOP-OF-STACK IN EP PC;
                =3182 ;        RETURN.
0500 744F       =3183 STSAVE:  CALL    EPBRK
0502 2303       =3184          MOV     A,#3
0504 3400       =3185          CALL    OUTUTL
0506 745A       =3186          CALL    OVSWAP
0508 B88F       =3187          MOV     R0,#LOW(OV0BAS+OVSIZE)
050A 746A       =3188          CALL    OVLOAD
050C 8A20       =3189          ORL     P2,#00100000B
050E 2314       =3190          MOV     A,#14H
0510 91         =3191          MOVX    @R1,A
0511 9ADF       =3192          ANL     P2,#NOT 00100000B
0513 8903       =3193          ORL     P1,#00000011B
0515 F4DB       =3194          CALL    EPSTEP
0517 8A20       =3195          ORL     P2,#00100000B
0519 9AEF       =3196          ANL     P2,#NOT 00010000B
051B 8903       =3197          ORL     P1,#(ENBRAM OR ENBLNK)
051D F4DB       =3198          CALL    EPSTEP
                =3199 ;
                =3200 ;        EXECUTION PROCESSOR IS NOW AT LOCATION 009H INTERNAL WITH
                =3201 ;        (RETURN ADDRESS+2) PUSHED ON STACK.
                =3202 ;
051F B8A5       =3203          MOV     R0,#LOW(OV3BAS+OVSIZE)
0521 746A       =3204          CALL    OVLOAD
0523 F4D0       =3205          CALL    EPPASS
                =3206          MMOV    EPACC,A
0525 B920       =3219+           MOV     R1,#EPACC
0527 A1         =3220+           MOV     @R1,A
0528 F4D0       =3224          CALL    EPPASS
                =3225          MMOV    EPTIMR,A
052A B922       =3238+           MOV     R1,#EPTIMR
052C A1         =3239+           MOV     @R1,A
052D F4D0       =3243          CALL    EPPASS
                =3244          MMOV    EPPSW,A
052F B921       =3257+           MOV     R1,#EPPSW
0531 A1         =3258+           MOV     @R1,A
0532 F4D0       =3262          CALL    EPPASS
                =3263          MMOV    EPR0,A
0534 B923       =3276+           MOV     R1,#EPR0
0536 A1         =3277+           MOV     @R1,A
0537 B88B       =3281          MOV     R0,#LOW(OV1BAS+OVSIZE)
0539 746A       =3282          CALL    OVLOAD
                =3283          MMOV    A,EPPSW
053B B921       =3292+           MOV     R1,#EPPSW
053D F1         =3293+           MOV     A,@R1
053E 07         =3297          DEC     A
053F 5307       =3298          ANL     A,#07H
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

0541 E7         =3299         RL      A
0542 0308       =3300         ADD     A,#08H
                =3301         MMOV    SMALO,A
0544 B930       =3314+          MOV     R1,#SMALO
0546 A1         =3315+          MOV     @R1,A
0547 F4B7       =3319         CALL    EPFET
0549 03FE       =3320         ADD     A,#-2
054B AA         =3321         MOV     LDATA,A
                =3322         MMOV    EPPCLO,A
054C B924       =3335+          MOV     R1,#EPPCLO
054E A1         =3336+          MOV     @R1,A
054F F4C3       =3340         CALL    EPSTOR
0551 B930       =3341         MOV     R1,#SMALO
0553 11         =3342         INC     @R1
0554 F4B7       =3343         CALL    EPFET
0556 AA         =3344         MOV     LDATA,A
0557 53F0       =3345         ANL     A,#11110000B
0559 2A         =3346         XCH     A,LDATA
055A 13FF       =3347         ADDC    A,#-1
055C 530F       =3348         ANL     A,#00001111B
                =3349         MMOV    EPPCHI,A
055E B925       =3362+          MOV     R1,#EPPCHI
0560 A1         =3363+          MOV     @R1,A
0561 4A         =3367         ORL     A,LDATA
0562 AA         =3368         MOV     LDATA,A
0563 F4C3       =3369         CALL    EPSTOR
0565 B825       =3370         MOV     R0,#EPPCHI
0567 347C       =3371         CALL    UPDAD1
0569 2340       =3372         MOV     A,#01000000B    ;"-" FOR DISPLAY
056B D4D8       =3373         CALL    WDISP
056D B820       =3374         MOV     R0,#EPACC
056F 3490       =3375         CALL    DSPMID
0571 83         =3376         RET
                =3377         SIZECHK
0072            =3380+  SIZE  SET   114
                =3381+;
                =3382+;*******************************************************
                =3391 $EJECT
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

                 3392 $      INCLUDE(:F0:HFILE.MOD)
000D            =3393 CHARCR EQU    0DH             ;<CR>
000A            =3394 CHARLF EQU    0AH             ;<LF>
001A            =3395 CNTRLZ EQU    1AH             ;CONTROL-Z
                =3396 ;
                =3397        CODEBLK 80
0297            =3412+         ORG    663
                =3416 ;HRECIN HEXFILE RECORD INPUT ROUTINE
0297 34CD       =3417 HRECIN: CALL   CHARIN
0299 D31A       =3418         XRL    A,#CNTRLZ
029B C6E0       =3419         JZ     DONE
029D D31A       =3420         XRL    A,#CNTRLZ
029F D33A       =3421         XRL    A,#(':')
02A1 9697       =3422         JNZ    HRECIN
                =3423         MMOV   CHKSUM,ZERO
02A3 BD00       =3428+          MOV    CHKSUM,#ZERO
02A5 14F0       =3432         CALL   BYTEIN
                =3433         MMOV   BUFCNT,A
02A7 B941       =3446+          MOV    R1,#BUFCNT
02A9 A1         =3447+          MOV    @R1,A
02AA 14F0       =3451         CALL   BYTEIN
                =3452         MMOV   SMAHI,A
02AC B931       =3465+          MOV    R1,#SMAHI
02AE A1         =3466+          MOV    @R1,A
02AF 14F0       =3470         CALL   BYTEIN
                =3471         MMOV   SMALO,A
02B1 B930       =3484+          MOV    R1,#SMALO
02B3 A1         =3485+          MOV    @R1,A
02B4 14F0       =3489         CALL   BYTEIN
                =3490         MMOV   RECTYP,A
02B6 B942       =3503+          MOV    R1,#RECTYP
02B8 A1         =3504+          MOV    @R1,A
                =3508 ;
                =3509 ;HDATIN HEX DATA BYTE IN
                =3510 HDATIN: MMOV   A,BUFCNT
02B9 B941       =3519+          MOV    R1,#BUFCNT
02BB F1         =3520+          MOV    A,@R1
02BC C6CC       =3524         JZ     RECDON
02BE 14F0       =3525         CALL   BYTEIN
02C0 AA         =3526         MOV    LDATA,A
02C1 F400       =3527         CALL   LSTORE
02C3 34F2       =3528         CALL   INCSMA
                =3529         MDEC   BUFCNT
02C5 B941       =3534+          MOV    R1,#BUFCNT
02C7 F1         =3535+          MOV    A,@R1
02C8 07         =3539+          DEC    A
02C9 A1         =3544+          MOV    @R1,A
02CA 44B9       =3547         JMP    HDATIN
                =3548 ;
02CC 34CD       =3549 RECDON: CALL   CHARIN
02CE D33F       =3550         XRL    A,#('?')
02D0 C6D8       =3551         JZ     CKSMOK
02D2 D33F       =3552         XRL    A,#('?')       ;SWITCH BACK TO DATA CHARACTER
02D4 34BA       =3553         CALL   NIBIN2         ;JOIN SUBROUTINE ALREADY IN PROGRESS
02D6 14F2       =3554         CALL   BYTEI1         ;DITTO
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

                =3555                               ;(RESULT FOR NON-'?' CHARACTERS IS AS IF
                =3556                               ;  BYTEIN WAS CALLED.)
                =3557           MMOV    A,CHKSUM
02D8 FD         =3573+            MOV     A,CHKSUM
02D9 96E1       =3577           JNZ     CHKERR
                =3578 CKSMOK:   MMOV    A,RECTYP
02DB B942       =3587+            MOV     R1,#RECTYP
02DD F1         =3588+            MOV     A,@R1
02DE C697       =3592           JZ      HRECIN
                =3593 ;
                =3594 ;DONE     HEX FILE CORRECTLY RECEIVED
02E0 83         =3595 DONE:     RET
                =3596 ;
                =3597 ;CHKERR CHECKSUM ERROR IN INPUT RECORD DETECTED
02E1 BA0C       =3598 CHKERR:   MOV     LDATA,#0CH
02E3 249A       =3599           JMP     PERROR
                =3600           SIZECHK
004E            =3603+  SIZE    SET     78
                =3604+;
                =3605+;*************************************************************
                =3614 ;
                =3615           CODEBLK 12
00F0            =3620+            ORG     240
                =3624 ;BYTEIN BYTE INPUT SUBROUTINE.
                =3625 ;         RECEIVES TWO HEXIDECIMAL CHARACTERS FROM THE TAPE INPUT DEVICE
                =3626 ;         AND ASSEMBLES THEM INTO A SINGLE BYTE OF DATA.
00F0 34B8       =3627 BYTEIN:   CALL    NIBIN
00F2 47         =3628 BYTEI1:   SWAP    A
00F3 AA         =3629           MOV     LDATA,A
00F4 34B8       =3630           CALL    NIBIN
                =3631           MORL    LDATA,A
00F6 4A         =3640+            ORL     A,LDATA
00F7 AA         =3660+            MOV     LDATA,A
00F8 6D         =3664           ADD     A,CHKSUM
00F9 AD         =3665           MOV     CHKSUM,A
00FA FA         =3666           MOV     A,LDATA
00FB 83         =3667           RET
                =3668           SIZECHK
000C            =3671+  SIZE    SET     12
                =3672+;
                =3673+;*************************************************************
                =3682 ;
                =3683           CODEBLK 25
01B8            =3693+            ORG     440
                =3697 ;NIBIN    RECEIVES A HEXIDECIMAL CHARACTER AND PRODUCES A MASKED FOUR BIT VALUE.
                =3698 ;         NOTE- ERROR CHECKING DONE TO VERIFY HEXIDECIMAL VALIDITY
01B8 34CD       =3699 NIBIN:    CALL    CHARIN
01BA 03C6       =3700 NIBIN2:   ADD     A,#-3AH         ;ACC=0F6-0FF FOR CHARACTERS '0'-'9'
                =3701                                   ;CHARACTERS > '9' PRODUCE OVERFLOW
01BC E6C2       =3702           JNC     NIBI3
01BE 03F9       =3703           ADD     A,#-7           ;ACC=0-5 FOR CHARACTERS 'A'-'F'
01C0 E6C9       =3704           JNC     ASCERR          ;ERROR IF CHARACTER BETWEEN '9' AND 'A'
                =3705 ;
                =3706 ;         ACC=0F6H-0FFH FOR CHARACTERS '0'-'9'
                =3707 ;
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

01C2 03FA      =3708 NIBI3:  ADD     A,#-6          ;ACC=0F0H-0FFH FOR CHARACTERS '0'-'F'
01C4 0310      =3709         ADD     A,#10H         ;ACC=00H-0FH FOR CHARACTERS '0'-'F';
               =3710                                ;OVERFLOW IF ABOVE IS TRUE.
01C6 E6C9      =3711         JNC     ASCERR
01C8 83        =3712         RET
               =3713 ;
               =3714 ;ASCERR ILLEGAL HEXIDECIMAL CHARACTER RECEIVED
01C9 BA0A      =3715 ASCERR: MOV     LDATA,#0AH
01CB 249A      =3716         JMP     PERROR
               =3717         SIZECHK
0015           =3720+  SIZE  SET   21
               =3721+;
               =3722+;*************************************************************
               =3731 ;
               =3732 ;
               =3733         CODEBLK 5
01CD           =3743+          ORG     461
               =3747 ;CHARIN CHARACTER INPUT ROUTINE.
               =3748 ;       RECEIVES ONE ASCII CHARACTER FROM THE LOGICAL READER DEVICE.
01CD D449      =3749 CHARIN: CALL    CIN
01CF 537F      =3750         ANL     A,#7FH
01D1 83        =3751         RET
               =3752         SIZECHK
0005           =3755+  SIZE  SET   5
               =3756+;
               =3757+;*************************************************************
               =3766 ;
               =3767 ;
               =3768 $EJECT
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

                =3769         CODEBLK 100
0572            =3794+          ORG     1394
                =3798 ;HFILEO HEX FILE OUTPUT SUBROUTINE
                =3799 ;       WHEN CALLED WITH F0=0 OUTPUT IS STANDARD HEX FILE FORMAT.
                =3800 ;       WHEN CALLED WITH F0=1 OUTPUT IS FORMATTED DATA DUMP TO CRT.
                =3801 HFILEO: MMOV    MEMHI,SMAHI
0572 B931       =3817+          MOV     R1,#SMAHI
0574 F1         =3818+          MOV     A,@R1
0575 B935       =3824+          MOV     R1,#MEMHI
0577 A1         =3825+          MOV     @R1,A
                =3826         MMOV    MEMLO,SMALO
0578 B930       =3844+          MOV     R1,#SMALO
057A F1         =3845+          MOV     A,@R1
057B B934       =3851+          MOV     R1,#MEMLO
057D A1         =3852+          MOV     @R1,A
                =3855         MMOV    CHKSUM,ZERO
057E BD00       =3860+          MOV     CHKSUM,#ZERO
0580 B865       =3864         MOV     R0,#HEXBUF
                =3865 ;
                =3866 ;LDBYTE LOAD NEXT BYTE FROM MEMORY INTO HEX BUFFER
0582 14FC       =3867 LDBYTE: CALL    LFETCH
0584 FA         =3868         MOV     A,LDATA
0585 A0         =3869         MOV     @R0,A
0586 18         =3870         INC     R0
0587 B4E2       =3871         CALL    CMPMAS
0589 E696       =3872         JNC     ENDFIL
058B 34F2       =3873         CALL    INCSMA
058D F8         =3874         MOV     A,R0
058E 0388       =3875         ADD     A,#-(BUFLEN+HEXBUF)
0590 E682       =3876         JNC     LDBYTE
0592 D400       =3877         CALL    HRECO
0594 A472       =3878         JMP     HFILEO
                =3879 ;
                =3880 ;ENDFIL END HEX FILE TRANSMISSION:
                =3881 ;       PRINT OUT BUFFER FOR LAST DATA RECORD
                =3882 ;       PRINT OUT CANNED 'END-OF-FILE' RECORD
                =3883 ;       RETURN.
0596 D400       =3884 ENDFIL: CALL    HRECO
0598 B6A7       =3885         JF0     HFDONE
059A 34D2       =3886         CALL    TCRLFO
059C B8AE       =3887         MOV     R0,#(LOW EOFREC)
059E F8         =3888 ENDF1:  MOV     A,R0
059F A3         =3889         MOVP    A,@A
05A0 C6A7       =3890         JZ      HFDONE
05A2 B4BD       =3891         CALL    CHARO
05A4 18         =3892         INC     R0
05A5 A49E       =3893         JMP     ENDF1
05A7 34D2       =3894 HFDONE: CALL    TCRLFO
05A9 231A       =3895         MOV     A,#CNTRLZ
05AB B4BD       =3896         CALL    CHARO
05AD 83         =3897         RET
                =3898 ;
                =3899 ;EOFREC CHARACTER SRTING FOR CANNED END-OF-FILE RECORD FOR
                =3900 ;       INTEL HEX FILE FORMAT STANDARD.
05AE 203A3030   =3901 EOFREC: DB      ' :00000001FF'
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

05B2 30303030
05B6 30314646
05BA 00         =3902         DB      0               ;END OF STRING CODE BYTE
                =3903         SIZECHK
0049            =3906+  SIZE  SET   73
                =3907+;
                =3908+;*****************************************************************
                =3917 ;
                =3918 ;
                =3919         CODEBLK 90
0600            =3949+          ORG     1536
                =3953 ;HRECO  HEXIDECIMAL RECORD OUTPUT SEQUENCE.
                =3954 ;       HEX BUFFER ALREADY LOADED.
0600 F8         =3955 HRECO:  MOV     A,R0
0601 039B       =3956         ADD     A,#-HEXBUF
                =3957         MMOV    BUFCNT,A
0603 B941       =3970+          MOV     R1,#BUFCNT
0605 A1         =3971+          MOV     @R1,A
0606 34D2       =3975         CALL    TCRLFO
0608 2320       =3976         MOV     A,#' '
060A B4BD       =3977         CALL    CHARO
060C B617       =3978         JF0     FDUMP1
060E 233A       =3979         MOV     A,#':'
0610 B4BD       =3980         CALL    CHARO
                =3981         MMOV    A,BUFCNT
0612 B941       =3990+          MOV     R1,#BUFCNT
0614 F1         =3991+          MOV     A,@R1
0615 34DC       =3995         CALL    BYTEO
                =3996 FDUMP1: MMOV    A,MEMHI
0617 B935       =4005+          MOV     R1,#MEMHI
0619 F1         =4006+          MOV     A,@R1
061A 34DB       =4010         CALL    BYTEO
                =4011         MMOV    A,MEMLO
061C B934       =4020+          MOV     R1,#MEMLO
061E F1         =4021+          MOV     A,@R1
061F 34DB       =4025         CALL    BYTEO
0621 B628       =4026         JF0     FDUMP2
0623 27         =4027         CLR     A
0624 34DB       =4028         CALL    BYTEO
0626 C42C       =4029         JMP     DATO
0628 233D       =4030 FDUMP2: MOV     A,#'='
062A B4BD       =4031         CALL    CHARO
                =4032 ;DATO   DATA OUTPUT
062C B865       =4033 DATO:   MOV     R0,#HEXBUF
062E B632       =4034 DATO1:  JF0     FDUMP5
0630 C436       =4035         JMP     FDUMP3
0632 2320       =4036 FDUMP5: MOV     A,#' '
0634 B4BD       =4037         CALL    CHARO
0636 F0         =4038 FDUMP3: MOV     A,@R0
0637 34DB       =4039         CALL    BYTEO
0639 18         =4040         INC     R0
                =4041         MDJNZ   BUFCNT,DATO1
063A B941       =4046+          MOV     R1,#BUFCNT
063C F1         =4047+          MOV     A,@R1
063D 07         =4051+          DEC     A
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

063E A1         =4056+          MOV     @R1,A
063F 962E       =4060+          JNZ     DAT01
                =4062 ;
                =4063 ;ENDREC END RECORD BEING TRANSMITTED
0641 B648       =4064 ENDREC: JF0       FDUMP4
                =4065         MMOV    A,CHKSUM
0643 FD         =4081+          MOV     A,CHKSUM
0644 37         =4085         CPL     A
0645 17         =4086         INC     A
0646 34DB       =4087         CALL    BYTE0
0648 83         =4088 FDUMP4: RET
                =4089         SIZECHK
0049            =4092+  SIZE  SET   73
                =4093+;
                =4094+;*************************************************************
                =4103 ;
                =4104         CODEBLK 9
01D2            =4114+          ORG     466
                =4118 ;TCRLF0 TAPE <CR><LF> OUTPUT
01D2 230D       =4119 TCRLF0: MOV     A,#CHARCR
01D4 B4BD       =4120         CALL    CHAR0
01D6 230A       =4121         MOV     A,#CHARLF
01D8 B4BD       =4122         CALL    CHAR0
01DA 83         =4123         RET
                =4124         SIZECHK
0009            =4127+  SIZE  SET   9
                =4128+;
                =4129+;*************************************************************
                =4138 ;
                =4139         CODEBLK 11
01DB            =4149+          ORG     475
                =4153 ;BYTE0  BYTE OUTPUT
01DB AA         =4154 BYTE0:  MOV     LDATA,A
01DC 6D         =4155         ADD     A,CHKSUM
01DD AD         =4156         MOV     CHKSUM,A
01DE FA         =4157         MOV     A,LDATA
01DF 47         =4158         SWAP    A
01E0 B4DB       =4159         CALL    NIB0
01E2 FA         =4160         MOV     A,LDATA
01E3 B4BB       =4161         CALL    NIB0
01E5 83         =4162         RET
                =4163         SIZECHK
000B            =4166+  SIZE  SET   11
                =4167+;
                =4168+;*************************************************************
                =4177 ;
                =4178         CODEBLK 12
01E6            =4188+          ORG     486
                =4192 ;HEXASC HEXIDECIMAL NIBBLE TO ASCII CHARACTER CONVERSION.
01E6 530F       =4193 HEXASC: ANL     A,#0FH
01E8 03F6       =4194         ADD     A,#(-10)
01EA F6EF       =4195         JC      HEXNIB
01EC 033A       =4196         ADD     A,#(10+'0')
01EE 83         =4197         RET
01EF 0341       =4198 HEXNIB: ADD     A,#('A')
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

01F1 83         =4199          RET
                =4200          SIZECHK
000C            =4203+  SIZE  SET   12
                =4204+;
                =4205+;*******************************************************
                =4214 ;
                =4215 ;
                =4216 DECLARE BITSO,CONST
0008            =4230 BITSO    EQU     11      ;DATA BITS PUT OUT (INCLUDING TWO STOP BITS)
                =4231 ;
                =4232          CODEBLK 30
04C9            =4252+           ORG     1225
                =4256 ;HBDLAY HALF-BIT TIME DELAY
                =4257 HBDLAY: MMOV    H,HBITHI
04C9 B927       =4273+           MOV     R1,#HBITHI
04CB F1         =4274+           MOV     A,@R1
04CC B945       =4280+           MOV     R1,#H
04CE A1         =4281+           MOV     @R1,A
                =4284          MMOV    R1,HBITLO
04CF B926       =4300+           MOV     R1,#HBITLO
04D1 F1         =4301+           MOV     A,@R1
04D2 A9         =4314+           MOV     R1,A
04D3 84D7       =4317          JMP     HBD1
04D5 B900       =4318 HBD2:    MOV     R1,#0
04D7 E9D7       =4319 HBD1:    DJNZ    R1,HBD1
                =4320          MDJNZ   H,HBD2
04D9 B945       =4325+           MOV     R1,#H
04DB F1         =4326+           MOV     A,@R1
04DC 07         =4330+           DEC     A
04DD A1         =4335+           MOV     @R1,A
04DE 96D5       =4339+           JNZ     HBD2
04E0 83         =4341          RET
                =4342          SIZECHK
0018            =4345+  SIZE  SET   24
                =4346+;
                =4347+;*******************************************************
                =4356 ;
                =4357 $EJECT
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

                =4358       CODEBLK 40
05B8            =4383+        ORG    1467
                =4387 ;NIBO   MASK ACC TO MAKE HEX NIBBLE, TRANSLATE TO ASCII AND OUTPUT
05B8 34E6       =4388 NIBO:   CALL   HEXASC
                =4389 ;
                =4390 ;CHARO  CONSOLE OUTPUT SUBROUTINE
                =4391 ;       WRITES THE CONTENTS OF THE ACC TO THE CRT DISPLAY SCREEN
                =4392 CHARO:  MMOV   REGC,A
05BD B944       =4405+        MOV    R1,#REGC
05BF A1         =4406+        MOV    @R1,A
                =4410         MMOV   B,BITSO ;SET NUMBER OF BITS TO BE TRANSMITTED
05C0 B943       =4421+        MOV    R1,#B
05C2 B108       =4422+        MOV    @R1,#BITSO
05C4 97         =4426         CLR    C       ;CLEAR CARRY
05C5 F6CB       =4427 CO1:    JC     CO2
05C7 998F       =4428         ANL    P1,#NOT TTYOUT
05C9 A4CF       =4429         JMP    CO3
05CB 8940       =4430 CO2:    ORL    P1,#TTYOUT
05CD 00         =4431         NOP           ;EVEN OUT TWO BRANCH EXECUTION TIMES
05CE 00         =4432         NOP
05CF 94C9       =4433 CO3:    CALL   HBDLAY
05D1 94C9       =4434         CALL   HBDLAY
05D3 97         =4435         CLR    C       ;SET WHAT WILL EVENTUALLY BECOME A STOP BIT
05D4 A7         =4436         CPL    C
                =4437         MRRC   REGC    ;ROTATE CHARACTER RIGHT ONE BIT,
05D5 B944       =4442+        MOV    R1,#REGC
05D7 F1         =4443+        MOV    A,@R1
05D8 67         =4447+        RRC    A
05D9 A1         =4452+        MOV    @R1,A
                =4455                        ;\ MOVING NEXT DATA BIT INTO CARRY
                =4456         MDJNZ  B,CO1   ;CHECK IF CHARACTER (AND STOP BIT(S)) DONE
05DA B943       =4461+        MOV    R1,#B
05DC F1         =4462+        MOV    A,@R1
05DD 07         =4466+        DEC    A
05DE A1         =4471+        MOV    @R1,A
05DF 96C5       =4475+        JNZ    CO1
05E1 83         =4477         RET
                =4478         SIZECHK
0027            =4481+  SIZE  SET   39
                =4482+;
                =4483+;*******************************************************
                =4492 ;
                =4493         CODEBLK 47
0649            =4523+        ORG    1609
                =4527 ;CIN    CONSOL INPUT SUBROUTINE WAITS FOR A KEYSTROKE AND
                =4528 ;       RETURNS WITH 8 BITS IN REG ACC.
0649 B943       =4529 CIN:    MOV    R1,#B
064B B108       =4530         MOV    @R1,#8  ;DATA BITS TO BE READ
064D 464D       =4531 CI0:    JNT1   CI0
064F 464D       =4532         JNT1   CI0
0651 5651       =4533 CI1:    JT1    CI1
0653 5651       =4534         JT1    CI1
0655 94C9       =4535         CALL   HBDLAY
0657 5651       =4536         JT1    CI1
0659 94C9       =4537 CI2:    CALL   HBDLAY
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

065B 94C9       =4538          CALL    HBDLAY
065D 5662       =4539          JT1     CI3     ;CHECK SID LINE LEVEL
065F 97         =4540          CLR     C       ;DATA BIT IN CY
0660 C465       =4541          JMP     CI4
0662 97         =4542 CI3:     CLR     C
0663 A7         =4543          CPL     C
0664 00         =4544          NOP             ;EVEN OUT BRANCH EXECUTION TIMES
0665 00         =4545 CI4:     NOP
0666 00         =4546          NOP
0667 00         =4547          NOP
                =4548          MRRC    REGC
0668 B944       =4553+           MOV     R1,#REGC
066A F1         =4554+           MOV     A,@R1
066B 67         =4558+           RRC     A
066C A1         =4563+           MOV     @R1,A
                =4566          MDJNZ   B,CI2
066D B943       =4571+           MOV     R1,#B
066F F1         =4572+           MOV     A,@R1
0670 07         =4576+           DEC     A
0671 A1         =4581+           MOV     @R1,A
0672 9659       =4585+           JNZ     CI2
                =4587          MMOV    A,REGC
0674 B944       =4596+           MOV     R1,#REGC
0676 F1         =4597+           MOV     A,@R1
0677 83         =4601          RET             ;CHARACTER COMPLETE
                =4602          SIZECHK
002F            =4605+  SIZE   SET   47
                =4606+;
                =4607+;*********************************************************
                =4616 $EJECT
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

                 4617 $       INCLUDE(:F0:MEMREF.MOD)
                =4618         CODEBLK 15
02E5            =4633+          ORG     741
                =4637 ;COMFIL COMMAND TO FILL ADDRESS SPACE BETWEEN SMA AND EMA WITH DATA
                =4638 ;       IN LOW BYTE OF MEM.
                =4639 COMFIL: MMOV    LDATA,MEMLO
02E5 B934       =4655+          MOV     R1,#MEMLO
02E7 F1         =4656+          MOV     A,@R1
02E8 AA         =4669+          MOV     LDATA,A
02E9 F400       =4672 LFILL:  CALL    LSTORE
02EB B4E2       =4673         CALL    CMPMAS
02ED E6F3       =4674         JNC     LFILL1
02EF 34F2       =4675         CALL    INCSMA
02F1 44E9       =4676         JMP     LFILL
02F3 83         =4677 LFILL1: RET
                =4678         SIZECHK
000F            =4681+  SIZE  SET   15
                =4682+;
                =4683+;*************************************************************
                =4692 ;
                =4693         CODEBLK 4
00FC            =4698+          ORG     252
                =4702 ;LFETCH FETCHES CONTENTS OF LOGICAL MEMORY ADDRESS DETERMINED BY
                =4703 ;       <TYPE>,<SMAHI>, & <SMALO> INTO <LDATA>.
00FC D478       =4704 LFETCH: CALL    AFETCH
00FE AA         =4705         MOV     LDATA,A
00FF 83         =4706         RET
                =4707         SIZECHK
0004            =4710+  SIZE  SET   4
                =4711+;
                =4712+;*************************************************************
                =4721 ;
                =4722         CODEBLK 75
0678            =4752+          ORG     1656
                =4756 ;
                =4757 ;AFETCH LOGICAL FETCH SUBROUTINE
                =4758 ;       FETCHS CONTENTS OF VARIOUS MEMORY SPACES TO ACC.
                =4759 AFETCH: MMOV    A,TYPE
0678 B937       =4768+          MOV     R1,#TYPE
067A F1         =4769+          MOV     A,@R1
067B 037E       =4773         ADD     A,#LOW LFETBL
067D B3         =4774         JMPP    @A
                =4775 ;
067E 84         =4776 LFETBL: DB      LOW LFEPM
067F 98         =4777         DB      LOW LFEDM
0680 9C         =4778         DB      LOW LFEREG
0681 A9         =4779         DB      LOW LFEINT
0682 B1         =4780         DB      LOW LFEBRK
0683 B1         =4781         DB      LOW LFEBRK
                =4782 ;
                =4783 LFEPM:  MMOV    A,SMAHI
0684 B931       =4792+          MOV     R1,#SMAHI
0686 F1         =4793+          MOV     A,@R1
0687 9698       =4797         JNZ     LFEDM
                =4798         MMOV    A,SMALO
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

0689 B930       =4807+          MOV     R1,#SMALO
068B F1         =4808+          MOV     A,@R1
068C 03E9       =4812         ADD     A,#-OVSIZE
068E F698       =4813         JC      LFEDM
                =4814         MMOV    A,SMALO
0690 B930       =4823+          MOV     R1,#SMALO
0692 F1         =4824+          MOV     A,@R1
0693 034E       =4828         ADD     A,#OVBUF
0695 A9         =4829         MOV     R1,A
0696 F1         =4830         MOV     A,@R1
0697 83         =4831         RET
0698 94E1       =4832 LFEDM:  CALL    LPGSEL
069A 81         =4833         MOVX    A,@R1
069B 83         =4834         RET
                =4835 ;
                =4836 LFEREG: MMOV    A,SMALO
069C B930       =4845+          MOV     R1,#SMALO
069E F1         =4846+          MOV     A,@R1
069F 537F       =4850         ANL     A,#01111111B    ;CHECK IF LOW 7 BITS =0
06A1 C6A5       =4851         JZ      LFER0
06A3 E4B7       =4852         JMP     EPFET
                =4853 ;
                =4854 LFER0:  MMOV    A,EPR0
06A5 B923       =4863+          MOV     R1,#EPR0
06A7 F1         =4864+          MOV     A,@R1
06A8 83         =4868         RET
                =4869 ;
                =4870 LFEINT: MMOV    A,SMALO
06A9 B930       =4879+          MOV     R1,#SMALO
06AB F1         =4880+          MOV     A,@R1
06AC 0320       =4884         ADD     A,#EPACC
06AE A9         =4885         MOV     R1,A
06AF F1         =4886         MOV     A,@R1
06B0 83         =4887         RET
                =4888 ;
                =4889 ;LFEBRK LOGICAL FETCH OF BREAK-POINT DATA
06B1 94E1       =4890 LFEBRK: CALL    LPGSEL
06B3 99F7       =4891         ANL     P1,#NOT 00001000B
06B5 8908       =4892         ORL     P1,#00001000B
06B7 99FD       =4893         ANL     P1,#NOT 00000010B
06B9 8901       =4894         ORL     P1,#00000001B
06BB 81         =4895         MOVX    A,@R1
06BC 2301       =4896         MOV     A,#01H
06BE 86C1       =4897         JNI     LFEBR1
06C0 27         =4898         CLR     A
06C1 83         =4899 LFEBR1: RET
                =4900         SIZECHK
004A            =4903+  SIZE  SET   74
                =4904+;
                =4905+;*******************************************************
                =4914 $EJECT
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

                =4915          CODEBLK 85
0700            =4950+           ORG    1792
                =4954 ;
                =4955 ;LSTORE LOGICAL STORE SUBROUTINE
                =4956 ;       STORES CONTENTS OF LDATA INTO VARIOUS MEMORY SPACES.
                =4957 LSTORE: MMOV   A,TYPE
0700 B937       =4966+          MOV    R1,#TYPE
0702 F1         =4967+          MOV    A,@R1
0703 0306       =4971          ADD    A,#LOW LSTTBL
0705 B3         =4972          JMPP   @A
                =4973 ;
0706 0C         =4974 LSTTBL: DB     LOW LSTPM
0707 21         =4975          DB     LOW LSTDM
0708 26         =4976          DB     LOW LSTREG
0709 34         =4977          DB     LOW LSTINT
070A 3D         =4978          DB     LOW LSTBRK
070B 3D         =4979          DB     LOW LSTBRK
                =4980 ;
                =4981 LSTPM:  MMOV   A,SMAHI
070C B931       =4990+          MOV    R1,#SMAHI
070E F1         =4991+          MOV    A,@R1
070F 9621       =4995          JNZ    LSTDM
                =4996          MMOV   A,SMALO
0711 B930       =5005+          MOV    R1,#SMALO
0713 F1         =5006+          MOV    A,@R1
0714 03E9       =5010          ADD    A,#-OVSIZE
0716 F621       =5011          JC     LSTDM
                =5012          MMOV   A,SMALO
0718 B930       =5021+          MOV    R1,#SMALO
071A F1         =5022+          MOV    A,@R1
071B 034E       =5026          ADD    A,#OVBUF
071D A9         =5027          MOV    R1,A
071E FA         =5028          MOV    A,LDATA
071F A1         =5029          MOV    @R1,A
0720 83         =5030          RET
                =5031 ;
0721 94E1       =5032 LSTDM:  CALL   LPGSEL
0723 FA         =5033          MOV    A,LDATA
0724 91         =5034          MOVX   @R1,A
0725 83         =5035          RET
                =5036 ;
                =5037 LSTREG: MMOV   A,SMALO
0726 B930       =5046+          MOV    R1,#SMALO
0728 F1         =5047+          MOV    A,@R1
0729 537F       =5051          ANL    A,#01111111B    ;CHECK IF LOW ORDER BITS = 0
072B C62F       =5052          JZ     LSTR0
072D E4C3       =5053          JMP    EPSTOR
                =5054 ;
                =5055 LSTR0:  MMOV   EPR0,LDATA
072F FA         =5078+          MOV    A,LDATA
0730 B923       =5084+          MOV    R1,#EPR0
0732 A1         =5085+          MOV    @R1,A
0733 83         =5088          RET
                =5089 ;
                =5090 LSTINT: MMOV   A,SMALO
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

0734 B930       =5099+          MOV     R1,#SMALO
0736 F1         =5100+          MOV     A,@R1
0737 0320       =5104           ADD     A,#EPACC
0739 A9         =5105           MOV     R1,A
073A FA         =5106           MOV     A,LDATA
073B A1         =5107           MOV     @R1,A
073C 83         =5108           RET
                =5109 ;
                =5110 ;LSTBRK LOGICAL STORE OF BREAK-POINT DATA
073D 94E1       =5111 LSTBRK: CALL    LPGSEL
073F FA         =5112         MOV     A,LDATA
0740 1246       =5113         JB0     LSTBR1
0742 8901       =5114         ORL     P1,#00000001B
0744 E448       =5115         JMP     LSTBR2
0746 99FE       =5116 LSTBR1: ANL     P1,#NOT 00000001B
0748 99F7       =5117 LSTBR2: ANL     P1,#NOT 00001000B
074A 81         =5118         MOVX    A,@R1
074B 8908       =5119         ORL     P1,#00001000B
074D 83         =5120         RET
                =5121         SIZECHK
004E            =5124+  SIZE  SET   78
                =5125+;
                =5126+;*************************************************************
                =5135 ;
                =5136         CODEBLK 17
04E1            =5156+          ORG     1249
                =5160 ;LPGSEL LOGICAL PAGE SELECT.
                =5161 ;       SETS UP PORT 2 TO ADDRESS APPROPRIATE BYTE OF RAM BLOCK.
                =5162 LPGSEL: MMOV    A,TYPE
04E1 B937       =5171+          MOV     R1,#TYPE
04E3 F1         =5172+          MOV     A,@R1
04E4 5301       =5176           ANL     A,#00000001B            ;MASK OFF DATA TYPE SELECTOR BIT
04E6 47         =5177           SWAP    A
                =5178           MORL    A,SMAHI
04E7 B931       =5184+          MOV     R1,#SMAHI
04E9 41         =5185+          ORL     A,@R1
04EA 4340       =5189           ORL     A,#01000000B
04EC 3A         =5190           OUTL    P2,A
                =5191           MMOV    A,SMALO
04ED B930       =5200+          MOV     R1,#SMALO
04EF F1         =5201+          MOV     A,@R1
04F0 A9         =5205           MOV     R1,A
04F1 83         =5206           RET
                =5207         SIZECHK
0011            =5210+  SIZE  SET   17
                =5211+;
                =5212+;*************************************************************
                =5221 ;
                =5222 $EJECT
```

```
LOC  OBJ        LINE       SOURCE STATEMENT

                =5223          CODEBLK 11
01F2            =5233+           ORG     498
                =5237 ;INCSMA INCREMENT STARTING MEMORY ADDRESS WORD.
01F2 B930       =5238 INCSMA: MOV     R1,#SMALO
01F4 11         =5239 INCW:   INC     @R1
01F5 F1         =5240         MOV     A,@R1
01F6 96FC       =5241         JNZ     INCW1
01F8 19         =5242         INC     R1
01F9 F1         =5243         MOV     A,@R1
01FA 17         =5244         INC     A
01FB 31         =5245         XCHD    A,@R1
01FC 83         =5246 INCW1:  RET
                =5247         SIZECHK
0008            =5250+  SIZE  SET   11
                =5251+;
                =5252+;*******************************************************
                =5261 ;
                =5262          CODEBLK 12
02F4            =5277+           ORG     756
                =5281 ;DECSMA DECREMENT SMA WORD.
02F4 B930       =5282 DECSMA: MOV     R1,#SMALO
02F6 F1         =5283         MOV     A,@R1
02F7 07         =5284         DEC     A
02F8 21         =5285         XCH     A,@R1
02F9 96FF       =5286         JNZ     DECSM1
02FB 19         =5287         INC     R1
02FC F1         =5288         MOV     A,@R1
02FD 07         =5289         DEC     A
02FE 31         =5290         XCHD    A,@R1
02FF 83         =5291 DECSM1: RET
                =5292         SIZECHK
000C            =5295+  SIZE  SET   12
                =5296+;
                =5297+;*******************************************************
                =5306 ;
                =5307          CODEBLK 15
05E2            =5332+           ORG     1506
                =5336 ;CMPMAS COMPARE MEMORY ADDRESSES
                =5337 ;       COMPARE SMA BYTES WITH EMA BYTES TO DETERMINE RELATIVE MAGNITUDE.
                =5338 ;       RETURNS WITH CARRY=1 IFF (SMA) >= (EMA).
                =5339 ;       IS CALLED AFTER ACTION HAS BEEN PERFORMED ON (SMA) TO DETERMINE IF
                =5340 ;       TASK IS COMPLETED:
                =5341 ;           IF CY=0 THEN (SMA) >= (EMA)   ==>  TERMINATE TASK.
                =5342 ;           IF CY=1 THEN (SMA) <  (EMA)   ==>  INC SMA AND REPEAT.
                =5343 CMPMAS: MMOV    A,SMALO
05E2 B930       =5352+          MOV     R1,#SMALO
05E4 F1         =5353+          MOV     A,@R1
05E5 37         =5357         CPL     A
                =5358         MADD    A,EMALO
05E6 B932       =5364+          MOV     R1,#EMALO
05E8 61         =5365+          ADD     A,@R1
                =5369         MMOV    A,SMAHI
05E9 B931       =5378+          MOV     R1,#SMAHI
05EB F1         =5379+          MOV     A,@R1
05EC 37         =5383         CPL     A
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

                =5384         MADDC   A,EMAHI
05ED B933       =5390+          MOV     R1,#EMAHI
05EF 71         =5391+          ADDC    A,@R1
05F0 83         =5395 CMPRET: RET
                =5396         SIZECHK
000F            =5399+  SIZE  SET   15
                =5400+;
                =5401+;*****************************************************************
                =5410 $EJECT
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

                 5411 $      INCLUDE(:F0:KBD.MOD)
                =5412        CODEBLK 100
074E            =5447+         ORG    1870
                =5451 ;
                =5452 ;      KEYBOARD AND DISPLAY PROCESSING ROUTINE
                =5453 ;      CALLED PERIODICALLY WHEN KBD AND DISPLAY ARE TO BE ALIVE.
074E D5         =5454 TIINT: SEL    RB1
                =5455        MMOV   ASAVE,A
074F B93E       =5468+         MOV    R1,#ASAVE
0751 A1         =5469+         MOV    @R1,A
0752 23F0       =5473        MOV    A,#(-10H)
0754 62         =5474        MOV    T,A             ;RELOAD TIMER INTERVAL
0755 27         =5475        CLR    A
0756 3E         =5476        MOVD   PSEGHI,A        ;WRITE BLANK PATTERN TO SEG DRIVERS
0757 3D         =5477        MOVD   PSEGLO,A
0758 FD         =5478        MOV    A,CURDIG
0759 07         =5479        DEC    A
075A 3F         =5480        MOVD   PDIGIT,A        ;ENERGIZE CHARACTER
075B 0C         =5481        MOVD   A,PINPUT        ;LOAD ANY SWITCH CLOSURES
075C AA         =5482        MOV    ROTPAT,A
                =5483                               ;WRITE NEXT SEGMENT PATTERN
075D FD         =5484        MOV    A,CURDIG
075E 07         =5485        DEC    A
075F 0346       =5486        ADD    A,#SEGMAP       ;ADD CURDIG DISPLACMENT TO BASE
0761 A8         =5487        MOV    R0,A
0762 F0         =5488        MOV    A,@R0           ;LOAD ACC W/ NEXT SEGMENT PATTERN
0763 3D         =5489        MOVD   PSEGLO,A        ;ENABLE APPROPRIATE SEGMENTS
0764 47         =5490        SWAP   A
0765 3E         =5491        MOVD   PSEGHI,A
                =5492 ;
                =5493 ;*******************************************************
                =5494 ;      THE NEXT CHARACTER IS NOW BEING DISPLAYED.
                =5495 ;      THE KEYBOARD SCAN ROUTINE IS INTEGRATED INTO THE DISPLAY SCAN.
                =5496 ;      WITH THE CURRENT ROW ENERGIZED, CHECK IF THERE ARE ANY INPUTS.
                =5497 ;*******************************************************
                =5498 ;
                =5499 ;      ROTATE BITS THROUGH THE CY WHILE INCREMENTING KEYLOC.
                =5500 ;
0766 B804       =5501        MOV    ROTCNT,#NCOLS   ;SET UP FOR <NCOLS> LOOPS THROUGH 'NXTLOC'
                =5502 NXTLOC: MRRC  ROTPAT
0768 FA         =5514+         MOV    A,ROTPAT
0769 67         =5518+         RRC    A
076A AA         =5529+         MOV    ROTPAT,A
076B F68B       =5532        JC     SCAN5           ;ONE BIT IN CY INDICATES KEY NOT DOWN
076D BE01       =5533        MOV    KEYFLG,#1       ;MARK THAT AT LEAST ONE KEY WAS DETECTED
                =5534                               ;\ IN THE CURRENT SCAN
                =5535 ;
                =5536 ;*******************************************************
                =5537 ;      A KEYSTROKE WAS DETECTED FOR THE CURRENT COLUMN.  ITS
                =5538 ;      POSITION IS IN REGISTER KEYLOC.  SEE IF SAME KEY SENSED LAST CYCLE.
                =5539 ;*******************************************************
                =5540 ;
                =5541        MMOV   A,KEYLOC
076F B93C       =5550+         MOV    R1,#KEYLOC
0771 F1         =5551+         MOV    A,@R1
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

0772 2C         =5555           XCH     A,LASTKY
0773 DC         =5556           XRL     A,LASTKY
0774 C67C       =5557           JZ      SCAN3
                =5558 ;
                =5559 ;*********************************************************
                =5560 ;       A DIFFERENT KEY WAS READ ON THIS CYCLE THAN ON THE PREVIOUS CYCLE.
                =5561 ;       SET NREPTS TO THE DEBOUNCE PARAMETER FOR A NEW COUNTDOWN.
                =5562 ;*********************************************************
                =5563 ;
0776 B93D       =5564           MOV     R1,#NREPTS
0778 B106       =5565           MOV     @R1,#6
077A E48B       =5566           JMP     SCAN5
                =5567 ;
                =5568 ;*********************************************************
                =5569 ;       SAME KEY WAS DETECTED 3S ON PREVIOUS CYCLE
                =5570 ;       LOOK AT NREPTS: IF ALREADY ZERO, DO NOTHING.
                =5571 ;       ELSE DECREMENT NREPTS.
                =5572 ;       IF THIS RESULTS IN ZERO, MOVE LASTKY INTO KBDBUF.
                =5573 ;*********************************************************
                =5574 ;
                =5575 SCAN3:  MMOV    A,NREPTS
077C B93D       =5584+          MOV     R1,#NREPTS
077E F1         =5585+          MOV     A,@R1
077F C68B       =5589           JZ      SCAN5           ;IF ALREADY ZERO
0781 07         =5590           DEC     A               ;INDICATE ONE MORE SUCCESIVE KEY DETECTION
                =5591           MMOV    NREPTS,A
0782 B93D       =5604+          MOV     R1,#NREPTS
0784 A1         =5605+          MOV     @R1,A
0785 968B       =5609           JNZ     SCAN5           ;IF DECREMENT DOES NOT RESULT IN ZERO
                =5610           MMOV    KBDBUF,LASTKY   ;TO MARK NEW KEY CLOSURE
0787 FC         =5633+          MOV     A,LASTKY
0788 B93B       =5639+          MOV     R1,#KBDBUF
078A A1         =5640+          MOV     @R1,A
                =5643 ;
078B B93C       =5644 SCAN5:  MOV     R1,#KEYLOC
078D 11         =5645           INC     @R1
078E ED68       =5646           DJNZ    ROTCNT,NXTLOC
0790 EDA8       =5647           DJNZ    CURDIG,TIRET1
0792 BD08       =5648           MOV     CURDIG,#CHARNO
                =5649 ;
                =5650 ;*********************************************************
                =5651 ;       THE FOLLOWING CODE SEGMENT IS USED BY THE KEYBOARD SCANNING ROUTINE.
                =5652 ;       IT IS EXECUTED ONLY AFTER A REFRESH SEQUENCE IS COMPLETED
                =5653 ;*********************************************************
                =5654 ;
                =5655           MMOV    KEYLOC,ZERO
0794 B93C       =5666+          MOV     R1,#KEYLOC
0796 B100       =5667+          MOV     @R1,#ZERO
0798 FE         =5671           MOV     A,KEYFLG
0799 969D       =5672           JNZ     SCAN8           ;JUMP IF ANY KEYS WERE DETECTED
                =5673           MMOV    LASTKY,NEG1     ;CHANGE (LASTKY) WHEN NO KEYS ARE DOWN
079B BCFF       =5678+          MOV     LASTKY,#NEG1
079D BE00       =5682 SCAN8:  MOV     KEYFLG,#0
                =5683 ;
                =5684 ;*********************************************************
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

                =5685 ;
                =5686 ;         KBD/DISP RETURN CODE- RESTORES SYSTEM STATUS.
                =5687           MMOV    A,RDELAY
079F B93F       =5696+            MOV     R1,#RDELAY
07A1 F1         =5697+            MOV     A,@R1
07A2 C6A8       =5701           JZ      TIRET1
07A4 07         =5702           DEC     A
                =5703           MMOV    RDELAY,A
07A5 B93F       =5716+            MOV     R1,#RDELAY
07A7 A1         =5717+            MOV     @R1,A
                =5721 TIRET1:   MMOV    A,ASAVE
07A8 B93E       =5730+            MOV     R1,#ASAVE
07AA F1         =5731+            MOV     A,@R1
07AB 93         =5735           RETR
                =5736 ;
                =5737 ;
                =5738 ;TOFPOL TIMER OVERFLOW POLLING SUBROUTINE.
                =5739 ;         CALLED REPEATEDLY FROM WHEREVER KBD/DISP MUST BE ALIVE.
                =5740 ;         MONITORS THE TIMER OVERFLOW FLAG (TOF) AND CALLS SERVICE
                =5741 ;         ROUTINE WHEN APPROPRIATE.
07AC 164E       =5742 TOFPOL:   JTF     TIINT
07AE 83         =5743           RET
                =5744           SIZECHK
0061            =5747+  SIZE    SET   97
                =5748+;
                =5749+;*********************************************************
                =5750 $EJECT
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

                =5759          CODEBLK 17
06C2            =5789+           ORG     1730
                =5793 ;
                =5794 ;KBDIN   KEYBOARD INPUT SUBROUTINE.
                =5795 ;        RETURNS ONLY AFTER A NEW KEYSTROKE HAS BEEN DETECTED AND DEBOUNCED.
                =5796 ;        VALUE OF KEY POSITION IN SWITCH MATRIX IS
                =5797 ;        RETURNED IN THE ACCUMULATOR.
                =5798 ;        DISPLAY CHARACTER NOW ON BLANKED BEFORE RETURNING.
06C2 BF03       =5799 KBDIN:   MOV     XPCODE,#3
06C4 74D1       =5800          CALL    XPTEST
06C6 F4AC       =5801 KBDI1:   CALL    TOFPOL
                =5802          MMOV    A,KBDBUF
06C8 B93B       =5813+           MOV     R1,#KBDBUF
06CA F1         =5812+           MOV     A,@R1
06CB F2C6       =5816          JB7     KBDI1
06CD 27         =5817          CLR     A
06CE 3E         =5818          MOVD    PSEGHI,A
06CF 3D         =5819          MOVD    PSEGLO,A
06D0 37         =5820          CPL     A
06D1 21         =5821          XCH     A,@R1
06D2 83         =5822          RET
                =5823          SIZECHK
0011            =5826+  SIZE   SET   17
                =5827+;
                =5828+;*************************************************************
                =5837 ;
                =5838          CODEBLK 15
05F1            =5863+           ORG     1521
                =5867 ;CLEAR   WRITES 'BLANK' CHARACTERS INTO ALL DISPLAY REGISTERS.
                =5868 ;        RETURNS WITH NEXTPL SET TO LEFTMOST CHARACTER POSITION
                =5869 ;        DOES NOT AFFECT ACC OR CY.
05F1 B846       =5870 CLEAR:   MOV     R0,#SEGMAP
05F3 B908       =5871          MOV     R1,#CHARNO
05F5 B000       =5872 DBLANK:  MOV     @R0,#0          ;STORE THE BLANK CODE
05F7 18         =5873          INC     R0              ;POINT TO NEXT CHARACTER TO THE LEFT
05F8 E9F5       =5874          DJNZ    R1,DBLANK
                =5875          MMOV    NEXTPL,CHARNO
05FA B93A       =5886+           MOV     R1,#NEXTPL
05FC B108       =5887+           MOV     @R1,#CHARNO
05FE 83         =5891          RET
                =5892          SIZECHK
000E            =5895+  SIZE   SET   14
                =5896+;
                =5897+;*************************************************************
                =5906 ;
                =5907          CODEBLK 44
06D3            =5937+           ORG     1747
                =5941 ;DSPACC DISPLAY VALUE OF LOW NIBBLE OF ACC
06D3 530F       =5942 DSPACC:  ANL     A,#0FH
06D5 03EF       =5943          ADD     A,#DGPATS
06D7 A3         =5944          MOVP    A,@A
                =5945 ;WDISP   WRITES BIT PATTERN NOW IN ACC INTO NEXT CHARACTER POSITION
                =5946 ;        OF THE DISPLAY (NEXTPL).  INCREMENTS NEXTPL.
                =5947 ;        RESULTS IN DISPLAY BEING FILLED LEFT TO RIGHT, THEN RESTARTING
06D8 AE         =5948 WDISP:   MOV     DSPTMP,A
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

06D9 BF04       =5949          MOV     XPCODE,#4
06DB 74D1       =5950          CALL    XPTEST
                =5951          MMOV    A,NEXTPL
06DD B93A       =5960+           MOV     R1,#NEXTPL
06DF F1         =5961+           MOV     A,@R1
06E0 0345       =5965          ADD     A,#SEGMAP-1
06E2 A9         =5966          MOV     R1,A
06E3 FE         =5967          MOV     A,DSPTMP
06E4 A1         =5968          MOV     @R1,A
                =5969          MDJNZ   NEXTPL,WDISP1
06E5 B93A       =5974+           MOV     R1,#NEXTPL
06E7 F1         =5975+           MOV     A,@R1
06E8 07         =5979+           DEC     A
06E9 A1         =5984+           MOV     @R1,A
06EA 96EE       =5988+           JNZ     WDISP1
06EC B108       =5990          MOV     @R1,#CHARNO
06EE 83         =5991 WDISP1:  RET
                =5992 ;
                =5993 ;DGPATS IS THE BASE FOR THE TABLE OF SEGMENT PATTERNS FOR HEX DIGITS.
                =5994 ;HERE THE FULL HEX SET (0-F) IS INCLUDED.
                =5995 ;
00EF            =5996 DGPATS   EQU     $ AND 0FFH
                =5997 ;
                =5998 ;FORMAT IS       PGFEDCBA        IN STANDARD SEVEN-SEGMENT ENCODING CONVENTION
                =5999 ;                                WHERE P REPRESENTS THE DECIMAL POINT
06EF 3F         =6000          DB      00111111B       ;SEGMENT PATTERN FOR DIGIT '0'
06F0 06         =6001          DB      00000110B       ;SEGMENT PATTERN FOR DIGIT '1'
06F1 5B         =6002          DB      01011011B       ;SEGMENT PATTERN FOR DIGIT '2'
06F2 4F         =6003          DB      01001111B       ;SEGMENT PATTERN FOR DIGIT '3'
06F3 66         =6004          DB      01100110B       ;SEGMENT PATTERN FOR DIGIT '4'
06F4 6D         =6005          DB      01101101B       ;SEGMENT PATTERN FOR DIGIT '5'
06F5 7D         =6006          DB      01111101B       ;SEGMENT PATTERN FOR DIGIT '6'
06F6 07         =6007          DB      00000111B       ;SEGMENT PATTERN FOR DIGIT '7'
06F7 7F         =6008          DB      01111111B       ;SEGMENT PATTERN FOR DIGIT '8'
06F8 67         =6009          DB      01100111B       ;SEGMENT PATTERN FOR DIGIT '9'
06F9 77         =6010          DB      01110111B       ;SEGMENT PATTERN FOR DIGIT 'A'
06FA 7C         =6011          DB      01111100B       ;SEGMENT PATTERN FOR DIGIT 'B'
06FB 39         =6012          DB      00111001B       ;SEGMENT PATTERN FOR DIGIT 'C'
06FC 5E         =6013          DB      01011110B       ;SEGMENT PATTERN FOR DIGIT 'D'
06FD 79         =6014          DB      01111001B       ;SEGMENT PATTERN FOR DIGIT 'E'
06FE 71         =6015          DB      01110001B       ;SEGMENT PATTERN FOR DIGIT 'F'
                =6016          SIZECHK
002C            =6019+  SIZE   SET   44
                =6020+;
                =6021+;*******************************************************
                =6030 ;
                =6031          CODEBLK 12
04F2            =6051+           ORG     1266
                =6055 ;DELAY   SUBROUTINE WAITS FOR THE NUMBER OF COMPLETE
                =6056 ;        DISPLAY SCANS CORRESPONDING TO THE ACC CONTENTS.
                =6057 ;        USED WITH CRUDE HUMAN INTERFACES- AS WHEN OPERATOR SHOULD SEE
                =6058 ;        SOME DISPLAY CHANGE WHILE IT IS CHANGING.
                =6059 DELAY:   MMOV    RDELAY,A
04F2 B93F       =6072+           MOV     R1,#RDELAY
04F4 A1         =6073+           MOV     @R1,A
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

04F5 F4AC      =6077 DELAY1: CALL    IOFPOL
               =6078         MMOV    A,RDELAY
04F7 B93F      =6087+          MOV     R1,#RDELAY
04F9 F1        =6088+          MOV     A,@R1
04FA 96F5      =6092         JNZ     DELAY1
04FC 83        =6093         RET
               =6094         SIZECHK
0008           =6097+  SIZE  SET   11
               =6098+;
               =6099+;*******************************************************
               =6100 ;
               =6109         CODEBLK 8
07AF           =6144+          ORG     1967
               =6148 ;KBDPOL POLL STATUS OF KEYBOARD INPUT ROUTINE.
               =6149 ;       RETURN WITH ACC BIT 7 = 0 IF KEYBOARD INPUT HAS BEEN RECEIVED.
07AF BF05      =6150 KBDPOL: MOV     XPCODE,#5
07B1 74D1      =6151         CALL    XPTEST
               =6152         MMOV    A,KBDBUF
07B3 B93B      =6161+          MOV     R1,#KBDBUF
07B5 F1        =6162+          MOV     A,@R1
07B6 83        =6166         RET
               =6167         SIZECHK
0008           =6170+  SIZE  SET   8
               =6171+;
               =6172+;*******************************************************
               =6181 $EJECT
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

                 6182 $       INCLUDE(:F0:LINK.MOD)
                =6183         CODEBLK 15
07B7            =6218+          ORG     1975
                =6222 ;EPFET  FETCH DATA BYTE FROM EP INTERNAL RAM ADDRESSED BY SMALO.
                =6223 EPFET:  MMOV    A,SMALO
07B7 B930       =6232+          MOV     R1,#SMALO
07B9 F1         =6233+          MOV     A,@R1
07BA F4D0       =6237         CALL    EPPASS
07BC 2380       =6238         MOV     A,#10000000B
07BE F4D0       =6239         CALL    EPPASS
07C0 F4D0       =6240         CALL    EPPASS
07C2 83         =6241         RET
                =6242         SIZECHK
000C            =6245+  SIZE  SET   12
                =6246+;
                =6247+;*************************************************************
                =6256 ;
                =6257         CODEBLK 15
07C3            =6292+          ORG     1987
                =6296 ;EPSTOR STORE DATA IN LDATA IN EP INTERNAL RAM AT <SMALO>
07C3 FA         =6297 EPSTOR: MOV     A,LDATA
07C4 F4D0       =6298         CALL    EPPASS
                =6299         MMOV    A,SMALO
07C6 B930       =6308+          MOV     R1,#SMALO
07C8 F1         =6309+          MOV     A,@R1
07C9 537F       =6313         ANL     A,#01111111B
07CB F4D0       =6314         CALL    EPPASS
07CD F4D0       =6315         CALL    EPPASS
07CF 83         =6316         RET
                =6317         SIZECHK
000D            =6320+  SIZE  SET   13
                =6321+;
                =6322+;*************************************************************
                =6331 $EJECT
```

```
LOC  OBJ        LINE       SOURCE STATEMENT

                =6332 ;       THE FOLLOWING UTILITIES INVOLVE INTERCHANGES BETWEEN THE MP AND EP.
                =6333 ;
                =6334         CODEBLK 11
07D0            =6369+          ORG     2000
                =6373 ;EPPASS PASSES A SINGLE PARAMETER BYTE TO THE EP THROUGH THE LINK.
                =6374 ;       WRITE THE CONTENTS OF THE ACC TO THE LINK;
                =6375 ;       RELEASE THE EP;
                =6376 ;       READ THE LINK INTO THE ACC;
                =6377 ;       RETURN.
07D0 8A30       =6378 EPPASS: ORL     P2,#00110000B          ;ENABLE LINK WRITES.
07D2 91         =6379         MOVX    @R1,A                  ;WRITE ACC TO LINK.
07D3 99FE       =6380         ANL     P1,#NOT ENBRAM         ;DISABLE BREAKPOINTS.
07D5 8902       =6381         ORL     P1,#ENBLNK             ;SET TO BREAK ON LINK REFERENCE.
07D7 F4DB       =6382         CALL    EPSTEP
07D9 81         =6383         MOVX    A,@R1
07DA 83         =6384         RET
                =6385         SIZECHK
0008            =6388+  SIZE  SET   11
                =6389+;
                =6390+;*****************************************************************
                =6399 ;
                =6400         CODEBLK 25
07DB            =6435+          ORG     2011
                =6439 ;EPSTEP RELEASES EP TO RUN IN PRESENT MODE UNTIL AN ANTICIPATED
                =6440 ;       HARDWARE BREAK OCCURS.
                =6441 ;       (DUE TO SINGLE STEPPING, LINK OPCODE FETCH, OR LINK DATA FETCH.)
                =6442 ;       MUST OCCUR WITHIN A FINITE NUMBER OF CYCLES (<40 MP CYCLES)
                =6443 ;       OR WATCHDOG TIMER WILL ASSUME A COMMUNICATIONS ERROR
                =6444 ;       BETWEEN THE MP AND EP.
07DB F4F4       =6445 EPSTEP: CALL    EPREL
07DD B90A       =6446         MOV     R1,#10
07DF 86F1       =6447 EPSTE1: JNI     EPSTE2
07E1 E9DE       =6448         DJNZ    R1,EPSTE1
07E3 8910       =6449         ORL     P1,#EPRSET
07E5 744F       =6450         CALL    EPBRK
07E7 B8BB       =6451         MOV     R0,#LOW(OV1BAS+OVSIZE)
07E9 746A       =6452         CALL    OVLOAD
07EB 99EF       =6453         ANL     P1,#NOT EPRSET
07ED BA0E       =6454         MOV     LDATA,#0EH
07EF 249A       =6455         JMP     PERROR
07F1 744F       =6456 EPSTE2: CALL    EPBRK
07F3 83         =6457         RET
                =6458         SIZECHK
0019            =6461+  SIZE  SET   25
                =6462+;
                =6463+;*****************************************************************
                =6472 ;
                =6473 ;
                =6474 $EJECT
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

                =6475           CODEBLK 9
07F4            =6510+            ORG     2036
                =6514 ;EPREL    RELEASES EP TO RUN IN PRESENT MODE.
                =6515 ;         SEQUENCE IS AS FOLLOWS:
                =6516 ;         PUT MEMORY ARRAY IN EP MODE;
                =6517 ;         RAISE /SSTEP;
                =6518 ;         RETURN.
07F4 99F7       =6519 EPREL:    ANL     P1,#NOT CLRBFF          ;CLEAR BREAK F/F.
07F6 8908       =6520           ORL     P1,#CLRBFF              ;RE-ENABLE BREAK F/F.
07F8 9ABF       =6521           ANL     P2,#NOT 010000000B      ;ENABLE EP CONTROL OF MEM ARRAY
07FA 8904       =6522           ORL     P1,#000001000B          ;FREE EP TO RUN UNTIL BREAK.
07FC 83         =6523           RET
                =6524           SIZECHK
0009            =6527+  SIZE    SET   9
                =6528+;
                =6529+;*********************************************************
                =6538 ;
                =6539 ;
                =6540           CODEBLK 11
034F            =6580+            ORG     847
                =6584 ;EPBRK    REGAIN CONTROL OF MEMORY ARRAY FROM EP.
                =6585 ;         DROP /SSTEP;
                =6586 ;         WAIT 30 USECS.;
                =6587 ;         PUT MEMORY ARRAY IN MP MODE;
                =6588 ;         RETURN.
034F 99FB       =6589 EPBRK:    ANL     P1,#NOT 000001000B      ;FREEZE EMULATION PROCESSOR.
0351 8920       =6590           ORL     P1,#MODOUT              ;SIGNAL EP IS NOT RUNNING USER CODE.
0353 B905       =6591           MOV     R1,#5
0355 E955       =6592           DJNZ    R1,$                    ;DELAY FOR EP TO FINISH INSTRUCTION.
0357 8A40       =6593           ORL     P2,#010000000B          ;SEIZE CONTROL OF MEM ARRAY.
0359 83         =6594           RET
                =6595           SIZECHK
000B            =6598+  SIZE    SET   11
                =6599+;
                =6600+;*********************************************************
                =6609 ;
                =6610 ;
                =6611           CODEBLK 16
035A            =6651+            ORG     858
                =6655 ;OVSWAP OVERLAY SWAP.
                =6656 ;         SWAPS BLOCK OF DATABYTES (USER'S PROGRAM) BETWEEN MP RAM & EP PM.
035A B865       =6657 OVSWAP:   MOV     R0,#OVBUF+OVSIZE
035C B917       =6658           MOV     R1,#OVSIZE
035E 2340       =6659           MOV     A,#010000000B
0360 3A         =6660           OUTL    P2,A
0361 C8         =6661 OVSW1:    DEC     R0
0362 C9         =6662           DEC     R1
0363 81         =6663           MOVX    A,@R1
0364 20         =6664           XCH     A,@R0
0365 91         =6665           MOVX    @R1,A
0366 F9         =6666           MOV     A,R1
0367 9661       =6667           JNZ     OVSW1
0369 83         =6668           RET
                =6669           SIZECHK
0010            =6672+  SIZE    SET   16
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

                =6673+;
                =6674+;*******************************************************
                =6683 ;
                =6684           CODEBLK 14
036A            =6724+            ORG     874
                =6728 ;OVLOAD OVERLAY LOAD.
                =6729 ;         MOVES BLOCK OF DATABYTES (ASSEMBLED SOURCE) FROM PG3 TO EP PM.
                =6730 ;         TOP OF DATA BLOCK LOADED AND BLOCK LENGTH DETERMINED BY R0 AND R1.
036A B917       =6731 OVLOAD:   MOV     R1,#OVSIZE
036C 2340       =6732           MOV     A,#01000000B
036E 3A         =6733           OUTL    P2,A
036F C8         =6734 MML01:    DEC     R0
0370 C9         =6735           DEC     R1
0371 F8         =6736           MOV     A,R0
0372 E3         =6737           MOVP3   A,@A
0373 91         =6738           MOVX    @R1,A
0374 F9         =6739           MOV     A,R1
0375 9G6F       =6740           JNZ     MML01
0377 83         =6741           RET
                =6742           SIZECHK
000E            =6745+  SIZE    SET   14
                =6746+;
                =6747+;*******************************************************
                =6756 $EJECT
```

```
LOC  OBJ        LINE      SOURCE STATEMENT

                =6757 ;
                =6758 ;=================================
                =6759 ;
                =6760 ;       THE REST OF THIS MODULE CONTAINS THE MINI-MONITORS WHICH OVERLAY
                =6761 ;       THE EMULATION PROCESSOR PROGRAM RAM TO GIVE THE
                =6762 ;       MASTER PROCESSOR ACCESS TO INTERNAL REGISTERS AND RAM OF THE EP.
                =6763 ;
                =6764 ;=================================
                =6765 ;
                =6766         DATABLK 22
0378            =6771+          ORG     888
                =6775 ;
                =6776 ;OV0-   OVERLAY TO BREAK EP EXECUTION AND JUMP TO LOCATION 009H.
                =6777 ;       LOCATION 009H REACHED WITH TOP-OF-STACK = RETURN ADDRESS+2
                =6778 ;       DUE TO FORCED "CALL" DURING WHICH PC WAS INCREMENTED.
                =6779 ;       LOCS 003H & 007H CALL 009H TO SIMULATE SAME CONDITION
                =6780 ;       IF BREAK OCCURS DURING INTERRUPT CYCLE.
                =6781 ;       SOURCE CODE FOR MINI-MONITOR OVERLAYED OVER LOW ORDER PROGRAM RAM.
                =6782 ;
0378            =6783 OV0BAS  EQU     $
0378            =6784 ORG     OV0BAS
0378 1409       =6785         CALL    009H
037A 00         =6786         NOP
                =6787 ;
037B            =6788 ORG     OV0BAS+003H
037B 1409       =6789         CALL    009H
037D 00         =6790         NOP
037E 00         =6791         NOP
                =6792 ;
037F            =6793 ORG     OV0BAS+007H
037F 1409       =6794         CALL    009H
0381 00         =6795         NOP
0382 00         =6796         NOP
0383 00         =6797         NOP
0384 00         =6798         NOP
0385 00         =6799         NOP
0386 00         =6800         NOP
0387 00         =6801         NOP
0388 00         =6802         NOP
0389 00         =6803         NOP
038A 00         =6804         NOP
038B 00         =6805         NOP
                =6806 ;
038C            =6807 ORG     OV0BAS+014H
038C 0409       =6808         JMP     009H
                =6809 ;
                =6810         SIZECHK
0016            =6813+  SIZE  SET   22
                =6814+;
                =6815+;*********************************************************
                =6824 $EJECT
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

                =6825         DATABLK 22
038E            =6830+          ORG     910
                =6834 ;
                =6835 ;OV3-   OVERLAY TO SAVE STATUS DATA AFTER BREAK.
                =6836 ;       ACC, TIMER/COUNTER, PSW (WITH F1), & RAM LOC 0 PASSED SEQUENTIALLY
                =6837 ;       TO MP.
                =6838 ;       SOURCE CODE FOR MINI-MONITOR OVERLAYED OVER LOW ORDER PROGRAM RAM.
                =6839 ;
038E            =6840 OV3BAS  EQU     $
038E            =6841 ORG     OV3BAS
038E 0400       =6842         JMP     000H
0390 00         =6843         NOP
                =6844 ;
0391            =6845 ORG     OV3BAS+003H
0391 83         =6846         RET
0392 00         =6847         NOP
0393 00         =6848         NOP
0394 00         =6849         NOP
                =6850 ;
0395            =6851 ORG     OV3BAS+007H
0395 83         =6852         RET
0396 00         =6853         NOP
                =6854 ;
0397            =6855 ORG     OV3BAS+009H
0397 90         =6856         MOVX    @R0,A
0398 42         =6857         MOV     A,T
0399 90         =6858         MOVX    @R0,A
039A C7         =6859         MOV     A,PSW
039B 7611       =6860         JF1     OV3B1
039D 53F7       =6861         ANL     A,#11110111B
0311            =6862 OV3B1   EQU     $-(LOW OV3BAS)
039F 90         =6863         MOVX    @R0,A
03A0 C5         =6864         SEL     RB0
03A1 F8         =6865         MOV     A,R0
03A2 0409       =6866         JMP     009H
                =6867 ;
                =6868         SIZECHK
0016            =6871+  SIZE  SET   22
                =6872+;
                =6873+;*********************************************************
                =6882 $EJECT
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

                =6883          DATABLK 22
03A4            =6888+           ORG     932
                =6892 ;
                =6893 ;OV1-    OVERLAY 1 TO GIVE MP ACCESS TO EP RAM LOCS. 01H-7FH.
                =6894 ;        SOURCE CODE FOR MINI-MONITOR OVERLAYED OVER LOW ORDER PROGRAM RAM.
                =6895 ;
03A4            =6896 OV1BAS   EQU     $
                =6897 ;
03A4 040A       =6898          JMP     OV1B1
03A6 00         =6899          NOP
                =6900 ;
03A7            =6901 ORG      OV1BAS+003H
03A7 83         =6902          RET
03A8 00         =6903          NOP
03A9 00         =6904          NOP
03AA 00         =6905          NOP
                =6906 ;
03AB            =6907 ORG      OV1BAS+007H
03AB 83         =6908          RET
03AC 00         =6909          NOP
                =6910 ;
03AD            =6911 ORG      OV1BAS+009H
03AD 90         =6912          MOVX    @R0,A
                =6913 ;
000A            =6914 OV1B1    EQU     $-OV1BAS
                =6915 ;
03AE 80         =6916          MOVX    A,@R0
03AF A8         =6917          MOV     R0,A
03B0 80         =6918          MOVX    A,@R0
03B1 F213       =6919          JB7     OV1B2
03B3 28         =6920          XCH     A,R0
03B4 A0         =6921          MOV     @R0,A
03B5 0409       =6922          JMP     009H
                =6923 ;
0313            =6924 OV1B2    EQU     $-LOW OV1BAS
                =6925 ;
03B7 F0         =6926          MOV     A,@R0
03B8 0409       =6927          JMP     009H
                =6928 ;
                =6929          SIZECHK
0016            =6932+  SIZE   SET   22
                =6933+;
                =6934+;*************************************************************
                =6943 $EJECT
```

```
LOC  OBJ        LINE        SOURCE STATEMENT

                =6944       DATABLK 23
03BA            =6949+        ORG     954
                =6953 ;
                =6954 ;OV2-   OVERLAY TO RESTORE EP STATUS SAVED ON BREAK AND RESUME USER'S PROGRAM.
                =6955 ;       SOURCE CODE FOR MINI-MONITOR OVERLAYED OVER LOW ORDER PROGRAM RAM.
                =6956 ;
03BA            =6957 OV2BAS  EQU     $
03BA            =6958 ORG     OV2BAS
03BA 0400       =6959         JMP     000H
03BC 00         =6960         NOP
                =6961 ;
03BD            =6962 ORG     OV2BAS+003H
03BD 83         =6963         RET
03BE 00         =6964         NOP
03BF 00         =6965         NOP
03C0 00         =6966         NOP
                =6967 ;
03C1            =6968 ORG     OV2BAS+007H
03C1 83         =6969         RET
03C2 00         =6970         NOP
                =6971 ;
03C3            =6972 ORG     OV2BAS+009H
03C3 90         =6973         MOVX    @R0,A
                =6974 ;
03C4 80         =6975         MOVX    A,@R0
03C5 A8         =6976         MOV     R0,A
03C6 80         =6977         MOVX    A,@R0
03C7 D7         =6978         MOV     PSW,A
03C8 A5         =6979         CLR     F1
03C9 B5         =6980         CPL     F1
03CA 7213       =6981         JB3     OV2B1
03CC A5         =6982         CLR     F1
                =6983 ;
0313            =6984 OV2B1   EQU     $-LOW OV2BAS
                =6985 ;
03CD 80         =6986         MOVX    A,@R0
03CE 62         =6987         MOV     T,A
03CF 80         =6988         MOVX    A,@R0
03D0 93         =6989         RETR
                =6990         SIZECHK
0017            =6993+  SIZE  SET   23
                =6994+;
                =6995+;*************************************************************
                =7004 $EJECT
```

```
LOC  OBJ          LINE       SOURCE STATEMENT

                  7005 ;
                  7006          CODEBLK 11
03D1              7046+           ORG     977
03D1 8A80         7050 XPTEST:  ORL     P2,#80H
03D3 0A           7051          IN      A,P2
03D4 9A7F         7052          ANL     P2,#(NOT 80H)
03D6 F2D9         7053          JB7     $+3
03D8 83           7054          RET
03D9 F5           7055          SEL     MB1
03DA 0400         7056          JMP     800H
                  7057          SIZECHK
000B              7060+  SIZE   SET   11
                  7061+;
                  7062+;*******************************************************
                  7071 ;
                  7072          CODEBLK 13
03DC              7112+           ORG     988
03DC 28432931     7116          DB      '(C)1979 INTEL'
03E0 39373920
03E4 494E5445
03E8 4C
                  7117          SIZECHK
000D              7120+  SIZE   SET   13
                  7121+;
                  7122+;*******************************************************
                  7131 ;
                  7132 ;
                  7133          RSOURCE
0100              7135+           PGSIZE  SET   ORGPG0-000H     ;BYTES USED ON PAGE 0
00FD              7136+           PGSIZE  SET   ORGPG1-100H     ;BYTES USED ON PAGE 1
0100              7137+           PGSIZE  SET   ORGPG2-200H     ;BYTES USED ON PAGE 2
00E9              7138+           PGSIZE  SET   ORGPG3-300H     ;BYTES USED ON PAGE 3
00FD              7139+           PGSIZE  SET   ORGPG4-400H     ;BYTES USED ON PAGE 4
00F1              7140+           PGSIZE  SET   ORGPG5-500H     ;BYTES USED ON PAGE 5
00FF              7141+           PGSIZE  SET   ORGPG6-600H     ;BYTES USED ON PAGE 6
00FD              7142+           PGSIZE  SET   ORGPG7-700H     ;BYTES USED ON PAGE 7
                  7143+$EJECT
```

```
LOC  OBJ        LINE       SOURCE STATEMENT

                7145 ;*********************************************************
                7146 ;
                7147 ;       FILL ALL UNUSED MEMORY LOCATIONS WITH NOP OPCODES
                7148 ;
                7149 ;*********************************************************
                7150 ;
                7151 $GEN
                7158 ;
01FD            7160           ORG   ORGPG1
                7161         REPT (200H - ORGPG1)
-               7162           DB    0
                7163         ENDM
01FD 00         7164+          DB    0
01FE 00         7165+          DB    0
01FF 00         7166+          DB    0
                7168 ;
                7175 ;
03E9            7177           ORG   ORGPG3
                7178         REPT (400H - ORGPG3)
-               7179           DB    0
                7180         ENDM
03E9 00         7181+          DB    0
03EA 00         7182+          DB    0
03EB 00         7183+          DB    0
03EC 00         7184+          DB    0
03ED 00         7185+          DB    0
03EE 00         7186+          DB    0
03EF 00         7187+          DB    0
03F0 00         7188+          DB    0
03F1 00         7189+          DB    0
03F2 00         7190+          DB    0
03F3 00         7191+          DB    0
03F4 00         7192+          DB    0
03F5 00         7193+          DB    0
03F6 00         7194+          DB    0
03F7 00         7195+          DB    0
03F8 00         7196+          DB    0
03F9 00         7197+          DB    0
03FA 00         7198+          DB    0
03FB 00         7199+          DB    0
03FC 00         7200+          DB    0
03FD 00         7201+          DB    0
03FE 00         7202+          DB    0
03FF 00         7203+          DB    0
                7205 ;
04FD            7207           ORG   ORGPG4
                7208         REPT (500H - ORGPG4)
-               7209           DB    0
                7210         ENDM
04FD 00         7211+          DB    0
04FE 00         7212+          DB    0
04FF 00         7213+          DB    0
                7215 ;
05FF            7217           ORG   ORGPG5
                7218         REPT (600H - ORGPG5)
```

```
LOC  OBJ          LINE        SOURCE STATEMENT

-                 7219          DB    0
                  7220        ENDM
05FF 00           7221+         DB    0
                  7223 ;
06FF              7225          ORG   ORGPG6
                  7226        REPT (700H - ORGPG6)
-                 7227          DB    0
                  7228        ENDM
06FF 00           7229+         DB    0
                  7231 ;
07FD              7233          ORG   ORGPG7
                  7234        REPT (800H - ORGPG7)
-                 7235          DB    0
                  7236        ENDM
07FD 00           7237+         DB    0
07FE 00           7238+         DB    0
07FF 00           7239+         DB    0
                  7241 ;
                  7242 $EJECT
```

```
LFILL   4672#  4676
LFILL1  4674   4677#
LPGSEL  4832   4890   5032   5111   5162#
LSTBR1  5113   5116#
LSTBR2  5115   5117#
LSTBRK  4978   4979   5111#
LSTDM   4975   4995   5011   5032#
LSTINT  4977   5090#
LSTORE  2459   2615   3527   4672   4957#
LSTPM   4974   4981#
LSTR0   5052   5055#
LSTREG  4976   5037#
LSTTBL  4971   4974#
M0       551#
M1       552#
MADD     430#  1816   2386   2438   5358
MADDC    435#  5384
MAIN    1434   1539#  1546   2349   2414   2417   2422   2427   2500   2620
MAIN2   1544#  3129
MAINA   1594   1609#
MAINB   1672#  1674
MAINB0  1798   1830#
MAINB1  1831#  1847
MAINC1  1716#  1762
MAIND   1741   1801#
MAIND1  1742   1766#
MANL     440#
MBLOCK   165#  1307   1315   1323
MDEC     471#  3529
MDJNZ    475#  4041   4320   4456   4566   5969
MEMHI   1158#  3824   4005
MEMLO   1149#  3851   4020   4655
MERROR  1592   1858#
MINC     467#  1743   2011   2875
MML01   6734#  6740
MMOV     398#  1558   1574   1609   1628   1649   1682   1716   1766   1782   1801   1976   1994   2172   2248   2329
        2464   2482   2541   2581   2714   2729   2756   2787   2805   2838   2856   2893   2923   2938   2953   2968
        3002   3063   3091   3206   3225   3244   3263   3283   3301   3322   3349   3423   3433   3452   3471   3490
        3510   3557   3578   3801   3828   3855   3957   3981   3996   4011   4065   4257   4284   4392   4410   4587
        4639   4759   4783   4798   4814   4836   4854   4870   4957   4981   4996   5012   5037   5055   5090   5162
        5191   5343   5369   5455   5541   5575   5591   5610   5655   5673   5687   5703   5721   5802   5875   5951
        6059   6078   6152   6223   6299
MODOUT   537#  3041   6590
MORL     445#  2908   3631   5178
MPUSEL   553#
MRL      482#
MRLC     494#
MRR      486#
MRRC     490#  4437   4548   5502
MXCH     455#
MXRL     450#
NCOLS    614#  5501
NEG1     729#  2341   5678
NEXTPL  1203#  2253   2259   5800   5886   5960   5974
NIBI3   3702   3708#
NIBIN   3627   3630   3699#
NIBIN2  3553   3700#
```

NIBO 4159 4161 4380#
NOBRK 1521# 1928
NOVALS 1381# 1416
NREPTS 1230# 5564 5584 5604
NUMCON 1185# 1662 1838 2181 2469 2475 2723
NXTLOC 5502# 5646
OPTAB1 1901 1903 1904 1905 1906 1922#
OPTAB2 1907 1908 1925#
OPTAB3 1902 1909 1927#
OPTION 1194# 1641 1698 1791 1810
ORGPG0 128# 1400 1401 1449# 1528 1529 1873# 1947 2157 2158 2230# 2234 2300 2367 2518 2651
2652 2674# 2679 3148 3399 3617 3618 3681# 3685 3735 3771 3921 4106 4141 4180 4234
4360 4495 4620 4695 4696 4720# 4724 4917 5138 5225 5264 5309 5414 5761 5840 5909
6033 6111 6185 6259 6336 6402 6477 6542 6613 6686 7008 7074 7135 7152 7153
ORGPG1 129# 1952 1953 2153# 2239 2240 2296# 2305 2306 2363# 2372 2523 2684 3153 3404 3690
3691 3730# 3740 3741 3765# 3776 3926 4111 4112 4137# 4146 4147 4176# 4185 4186 4213#
4239 4365 4500 4625 4729 4922 5143 5230 5231 5260# 5269 5314 5419 5766 5845 5914
6038 6116 6190 6264 6341 6407 6482 6547 6618 6691 7013 7079 7136 7159 7160
ORGPG2 130# 2377 2378 2514# 2528 2529 2647# 2689 3158 3409 3410 3613# 3781 3931 4244 4370
4505 4630 4631 4691# 4734 4927 5148 5274 5275 5305# 5319 5424 5771 5850 5919 6043
6121 6195 6269 6346 6412 6487 6552 6623 6696 7018 7084 7137 7169 7170
ORGPG3 131# 1334 1335 1395# 1877 1878 1943# 6577 6578 6600# 6648 6649 6682# 6721 6722 6755#
6768 6769 6823# 6827 6828 6881# 6885 6886 6942# 6946 6947 7003# 7043 7044 7070# 7109
7110 7130# 7138 7176 7177
ORGPG4 132# 2694 2695 3144# 3163 3786 3936 4249 4250 4355# 4375 4510 4739 4932 5153 5154
5220# 5324 5429 5776 5855 5924 6048 6049 6107# 6126 6200 6274 6351 6417 6492 6557
6628 6701 7023 7089 7139 7206 7207
ORGPG5 133# 3168 3169 3390# 3791 3792 3916# 3941 4380 4381 4491# 4515 4744 4937 5329 5330
5409# 5434 5781 5860 5861 5905# 5929 6131 6205 6279 6356 6422 6497 6562 6633 6706
7028 7094 7140 7216 7217
ORGPG6 134# 3946 3947 4102# 4520 4521 4615# 4749 4750 4913# 4942 5439 5786 5787 5836# 5934
5935 6029# 6136 6210 6284 6361 6427 6502 6567 6638 6711 7033 7099 7141 7224 7225
ORGPG7 135# 4947 4948 5134# 5444 5445 5757# 6141 6142 6180# 6215 6216 6255# 6289 6290 6330#
6366 6367 6398# 6432 6433 6471# 6507 6508 6537# 6572 6643 6716 7038 7104 7142 7232
7233
OUTCLR 1624 1974# 2556
OUTMSG 1797 1827 1975#
OUTUTL 1542 1973# 2326 2713 2993 3124 3185
OV0BAS 3187 6783# 6784 6788 6793 6807
OV1B1 6898 6914#
OV1B2 6919 6924#
OV1BAS 1426 3281 6451 6896# 6901 6907 6911 6914 6924
OV2B1 6981 6984#
OV2BAS 2921 6957# 6958 6962 6968 6972 6984
OV3B1 6860 6862#
OV3BAS 3203 6840# 6841 6845 6851 6855 6862
OVBUF 1319# 4828 5026 6657
OVLORD 1427 2922 3188 3204 3282 6452 6731#
OVSIZE 646# 1321 1426 2921 3187 3203 3281 4812 5010 6451 6657 6658 6731
OVSW1 6661# 6667
OVSWAP 2985 2995 3186 6657#
PERR 1518# 1924
PDIGIT 517# 5480
PERROR 1859 2212 2318# 2633 3089 3599 3716 6455
PGSIZE 7135# 7136# 7137# 7138# 7139# 7140# 7141# 7142#
PINPUT 520# 5481
PLUS1 699# 2476

```
PLUS3   714#  2260
PRNT1  2009   2030#
PRNT2  1994#  2029
PSEGH1  518#  1414   5476   5491   5818
PSEGLO  519#  1413   5477   5489   5819
RDELAY 1248#  5696   5716   6072   6087
RECDON 3524   3549#
RECTYP 1275#  3503   3587
REGC   1293#  4405   4442   4553   4596
REORG   191#  1335   1401   1529   1878   1948   1953   2158   2235   2240   2301   2306   2368   2373   2378   2519
       2524   2529   2652   2680   2685   2690   2695   3149   3154   3159   3164   3169   3400   3405   3410   3618
       3686   3691   3736   3741   3772   3777   3782   3787   3792   3922   3927   3932   3937   3942   3947   4107
       4112   4142   4147   4181   4186   4235   4240   4245   4250   4361   4366   4371   4376   4381   4496   4501
       4506   4511   4516   4521   4621   4626   4631   4696   4725   4730   4735   4740   4745   4750   4918   4923
       4928   4933   4938   4943   4948   5139   5144   5149   5154   5226   5231   5265   5270   5275   5310   5315
       5320   5325   5330   5415   5420   5425   5430   5435   5440   5445   5762   5767   5772   5777   5782   5787
       5841   5846   5851   5856   5861   5910   5915   5920   5925   5930   5935   6034   6039   6044   6049   6112
       6117   6122   6127   6132   6137   6142   6186   6191   6196   6201   6206   6211   6216   6260   6265   6270
       6275   6280   6285   6290   6337   6342   6347   6352   6357   6362   6367   6403   6408   6413   6418   6423
       6428   6433   6478   6483   6488   6493   6498   6503   6508   6543   6548   6553   6558   6563   6568   6573
       6578   6614   6619   6624   6629   6634   6639   6644   6649   6687   6692   6697   6702   6707   6712   6717
       6722   6769   6828   6886   6947   7009   7014   7019   7024   7029   7034   7039   7044   7075   7080   7085
       7090   7095   7100   7105   7110
RERROR 2317#  2348
RINT   1520#  1923
ROTCNT  886#  5501   5646
ROTPAT  865#  5482   5507   5514   5529
RSOURC  276#  7133
SCAN3  5557   5575#
SCAN5  5532   5566   5589   5609   5644#
SCAN8  5672   5682#
SEGMAP 1311#  2213   5486   5870   5965
SING   1523#  1928
SIZE   1385#  1388   1439#  1442   1863#  1866   1933#  1936   2143#  2146   2220#  2223   2286#  2289   2353#  2356
       2504#  2507   2637#  2640   2664#  2667   3134#  3137   3380#  3383   3603#  3606   3671#  3674   3720#  3723
       3755#  3758   3906#  3909   4092#  4095   4127#  4130   4166#  4169   4203#  4206   4345#  4348   4481#  4484
       4605#  4608   4681#  4684   4710#  4713   4903#  4906   5124#  5127   5210#  5213   5250#  5253   5295#  5298
       5399#  5402   5747#  5750   5826#  5829   5895#  5898   6019#  6022   6097#  6100   6170#  6173   6245#  6248
       6320#  6323   6388#  6391   6461#  6464   6527#  6530   6598#  6601   6672#  6675   6745#  6748   6813#  6816
       6871#  6874   6932#  6935   6993#  6996   7060#  7063   7120#  7123
SIZECH  270#  1382   1436   1860   1930   2140   2217   2283   2350   2501   2634   2661   3131   3377   3600   3668
       3717   3752   3903   4089   4124   4163   4200   4342   4478   4602   4678   4707   4900   5121   5207   5247
       5292   5396   5744   5823   5892   6016   6094   6167   6242   6317   6385   6458   6524   6595   6669   6742
       6810   6868   6929   6990   7057   7117
SMAHI  1122#  2487   2493   2772   3465   3817   4792   4990   5184   5378
SMALO  1113#  1671   1831   2480   2557   2745   2869   2880   3314   3341   3484   3844   4807   4823   4845   4879
       5005   5021   5046   5099   5200   5238   5282   5352   6232   6308
STRCOM 1623   2037#
STRGOC 1927   2054#
STRMEM 1922   1925   2047#  2555
STRTMP 1257#  1989   2003   2016
STRUTL 1973   2032#
STSAVE 3856   3862   3183#
TCRLFO 3886   3894   3975   4119#
TIINT  5454#  5742
TIRET1 5647   5701   5721#
TOFPOL 3846   5742#  5801   6077
```

```
TTYOUT  539#  4428  4430
TYPE   1176#  1429  1579  1585  1748  1771  1777  1822  2448  2550  3011  3072  4768  4966  5171
UPDHD1 2265#  2558  3371
UPDADR 2195   2248#
VERSNO 1050#
WBRK   1522#  1928
WDISP  2010   2030  2269  2274  2560  3373  5948#
WDISP1 5988   5991#
XPCODE  837#  1410  1539  2318  5799  5949  6150
XPTEST 1411   1540  2319  5800  5950  6151  7050#
ZERO    684#  1570  1586  1778  2494  3428  3860  5667

CROSS REFERENCE COMPLETE
```

```
BRKFIL 2433  2437#
BRKNXT 2459# 2499
BUFCNT 1266# 3446  3519  3534  3970  3990  4046
BUFLEN  662# 1329  3875
BYTEI1 3554  3628#
BYTEIN 3432  3451  3470  3489  3525  3627#
BYTEO  3995  4010  4025  4028  4039  4087  4154#
CGO    2986  3002#
CGONB  3019  3030#
CGOPAT 3022  3025#
CGOSS  3021  3034#
CGOTRA 3023  3033#
CGOWB  3020  3026#
CHARCR 3393# 4119
CHARIN 3417  3549  3699  3749#
CHARLF 3394# 4121
CHARNO  598# 1313  1349  1378  5648  5871  5887  5990
CHARO  3891  3896  3977  3980  4031  4037  4120  4122  4392#
CHKERR 3577  3598#
CHKSUM  803# 3428  3566  3573  3664  3665  3860  4074  4081  4155  4156
CI0    4531# 4531  4532
CI1    4533# 4533  4534  4536
CI2    4537# 4585
CI3    4539  4542#
CI4    4541  4545#
CIN    3749  4529#
CKSMOK 3551  3578#
CLEAR  1974  5870#
CLRBFF  534# 6519  6520
CMDINT 1836  1842  1845  1855#
CMPMAS 3871  4673  5343#
CMPRET 5395#
CNTRLZ 3395# 3418  3420  3895
CNTTBL 3077  3080#
CNTTRA 3083  3084  3091#
CO1    4427# 4475
CO2    4427  4430#
CO3    4429  4433#
CODEBL  199# 1398  1526  1945  2155  2232  2298  2365  2516  2649  2677  3146  3397  3615  3683  3733
        3769  3919  4104  4139  4178  4232  4358  4493  4618  4693  4722  4915  5136  5223  5262  5307
        5412  5759  5838  5907  6031  6109  6183  6257  6334  6400  6475  6540  6611  6684  7006  7072
COMCBR 2407  2432#
COMFIL 1428  1432  2426  4639#
COMGOR 2429  2992#
COMSGR 2406  2436#
COMSIZ 1596  1899#
CTAB   1557  1898#
CURDIG  928# 5478  5484  5647  5648
DATABL  244# 1332  1875  6766  6825  6883  6944
DATO   4029  4033#
DATO1  4034# 4060
DBLANK 2215  5872#  5874
DBPNT  2036  2085#
DBRK   1519# 1924
DCB    2045  2106#
DDABRK 2053  2122#
DDAMEM 2049  2116#
```

```
DEBNCE  630#
DECLAR  170#   584   600   616   632   648   670   685   700   715   739   756   773   790   807   824
         848   869   890   911   932   964   973   982   991  1000  1009  1018  1027  1036  1045  1054
        1063  1072  1081  1090  1099  1108  1117  1126  1135  1144  1153  1162  1171  1180  1189  1198
        1207  1216  1225  1234  1243  1252  1261  1270  1279  1288  1297  4216
DECSM1 5286  5291#
DECSMA 2630  5282#
DELAY  3105  6059#
DELAY1 6077#  6092
DERROR 2033  2063#
DFILL  2040  2096#
DGO    2039  2094#
DGPRTS 5943  5996#
DGR    2046  2108#
DINTRG 2051  2124#
DLSI   2041  2098#
DMOD   2038  2092#
DNOBRK 2055  2129#
DONE   3419  3595#
DPA    2058  2135#
DPRBRK 2052  2120#
DPRMEM 2048  2114#
DREC   2042  2100#
DREL   2043  2102#
DRM    2050  2118#
DRUN   2035  2078#
DSB    2044  2104#
DSGNON 2034  2070#
DSPACC 2276  2279  2281  2328  2564  2566  5942#
DSPHI  2268  2276#
DSPLO  2275  2280#
DSPML  2273  2279#
DSPMID 2277#  3375
DSPTIM 1041#  3100
DSPTMP  820#  5948  5967
DSS    2057  2133#
DTR    2059  2137#
DWBRK  2056  2131#
ELSIF1 2186  2188  2202#
ELSIF2 2204  2207  2213#
EMAHI  1140#  5390
EMALO  1131#  5364
ENBLNK  529#  3197  6381
ENBRAM  528#  3197  6380
ENDF1  3888#  3893
ENDFIL 3872  3884#
ENDREC 4064#
EOFREC 3887  3901#
EPACC   969#  2977  3219  3374  4884  5104
EPBRK  1425  3183  6450  6456  6589#
EPCNT  2921#  3107  3125
EPCON1 2785  2805#
EPCONT 2728  2783#
EPFET  3319  3343  4852  6223#
EPPASS 2937  2952  2967  2982  3205  3224  3243  3262  6237  6239  6240  6298  6314  6315  6378#
EPPCHI 1014#  2779  2914  3362  3370
EPPCLO 1005#  2752  2821  3335
```

```
EPPSW   978#  2800   2847   2902   2947   3257   3292
EPR0    996#  2932   3276   4863   5084
EPREL  3042   6445   6519#
EPRET  3118   3122   3129#
EPRSET  536#  1433   2994   2996   6449   6453
EPRUN  2424   2712#
EPRUN1 3046#  3051
EPRUN2 3050   3062#
EPRUN3 3049   3056#
EPRUN4 3028   3031   3039#
EPRUN5 3057   3115#
EPRUN6 3081   3082   3119#
EPSSTP  532#
EPSTE1 6447#  6448
EPSTE2 6447   6456#
EPSTEP 2984   3194   3198   6382   6445#
EPSTOR 2874   2920   3340   3369   5053   6297#
EPTIMR  987#  2962   3238
ERROR   740    765    782    799    816    833    861    882    903    924    945
ERROR2 2324   2349#
EXAM0  2541#  2616
EXAM1  2601   2618#
EXAM2  2619   2622#
EXAM3  2624   2627#
EXAM4  2629   2632#
EXAM5  2610   2613#
EXAMIN 2410   2540#  2626   2631
EXMON   555#
FDUMP1 3978   3996#
FDUMP2 4026   4030#
FDUMP3 4035   4038#
FDUMP4 4064   4088#
FDUMP5 4034   4036#
FINDOP 1590#  1600
GOTBL  3016   3019#
H      1302#  4280   4325
HBD1   4317   4319#  4319
HBD2   4318#  4339
HBDLAY 4257#  4433   4434   4535   4537   4538
HBITHI 1032#  4273
HBITLO 1023#  4300
HDATIN 3510#  3547
HEXASC 4193#  4388
HEXBUF 1327#  3864   3875   3956   4033
HEXNIB 4195   4198#
HFDONE 3885   3890   3894#
HFILEO 2413   2421   3801#  3878
HRECIN 2416   3417#  3422   3592
HRECO  3877   3884   3955#
HREGA  1059#
HREGB  1068#
HREGC  1077#
HREGD  1086#
HREGE  1095#
HREGF  1104#
IMPLEM 1855   2385#
INCSMA 1431   2625   3528   3873   4675   5230#
```

```
INCW    5239#
INCW1   5241   5246#
INIT    1409#
INITLP  1418#  1423
INPAD1  2187#  2198
INPADR  1835   2170#  2498
INPKEY  1543   1675   1828   1846   2196   2345   2463   2580   2658#  3115   3119
INVALS  1346#  1381   1417
ITMP     786#  1557   1590   1595   1597   1625   1626   1646   1647   1705   1712   1715   1725   1732   1761   1830
        1832   1833   1837
JGORES  2408   2429#
JMPTBL  2385   2399#
JTOFIL  2402   2426#
JTOGO   2401   2424#
JTOLST  2403   2419#
JTOMOD  2400   2410#
JTOREC  2404   2412#
JTOREL  2405   2416#
KBDBUF  1212#  2334   2340   5639   5811   6161
KBDI1   5801#  5816
KBDIN   2658   5799#
KBDPOL  3047   3106   6150#
KCLRB   1515#  1908
KEY      769#  1544   1593   1740   1843   2107   2202   2285   2322   2346   2460   2590   2597   2613   2622   2627
        2659   2783   3116   3120
KEYCLR  1505#  2323   2628
KEYDM   1504#  1923   1926
KEYEND  1501#  1545   1844   2206   2347   2461   2618   3117
KEYFIL  1499#  1903
KEYFLG   949#  5533   5671   5682
KEYGO   1512#  1902
KEYLOC  1221#  5550   5644   5660   5666
KEYLST  1510#  1904
KEYMOD  1513#  1901
KEYNX1  1500#  2203   2623   2784   3121
KEYPAT  1503#  1929
KEYPM   1508#  1923   1926
KEYREC  1506#  1905
KEYREG  1509#  1923
KEYREL  1502#  1906
KEYTRA  1507#  1929
KGORES  1511#  1909
KSETB   1514#  1907
LASTKY   907#  5555   5556   5626   5633   5670
LDATR    752#  1858   2211   2317   2327   2432   2436   2562   2565   2607   2614   2632   2828   2835   2919   3088
        3321   3344   3346   3367   3368   3526   3598   3629   3641   3648   3660   3666   3715   3868   4154   4157
        4160   4662   4669   4705   5028   5033   5071   5078   5106   5112   6297   6454
LDBYTE  3867#  3876
LFEBR1  4897   4899#
LFEBRK  4780   4781   4890#
LFEDM   4777   4797   4813   4832#
LFEINT  4779   4870#
LFEPM   4776   4783#
LFER0   4851   4854#
LFEREG  4778   4836#
LFETBL  4773   4776#
LFETCH  2561   3867   4704#
```

```
PSEGLO 000C    RDELAY 003F    RECDON 02CC    RECTYP 0042    REGC   0044    REORG  0005    RERROR 0198    RINT   0011
ROTCNT 0003    ROTPAT 0002    RSOURC 0012    SCAN3  077C    SCAN5  0788    SCAN8  079D    SEGMAP 0046    SING   001A
SIZE   000D    SIZECH 0011    SMAHI  0031    SMALO  0030    STRCOM 001D    STRGOC 002C    STRMEM 0026    STRTMP 0040
STRUTL 0019    STSAVE 0500    TCRLFO 01D2    TIINT  074E    TIRET1 07A8    TOFPOL 07AC    TTYOUT 0040    TYPE   0037
UPDAD1 017C    UPDADR 0178    VERSNO 0029    WBRK   0016    WDISP  06D8    WDISP1 06EE    XPCODE 0007    XPTEST 03D1
ZERO   0000

ASSEMBLY COMPLETE,  NO ERRORS
```

```
?A      105#  1614  1629  1637  1650  1658  1721  1787  1806  1818  1977  1985  1999  2177  2388  2444
        2546  2586  2719  2788  2796  2843  2857  2865  2898  2910  2928  2943  2958  2973  3007  3068
        3096  3207  3215  3226  3234  3245  3253  3264  3272  3288  3302  3310  3323  3331  3350  3358
        3434  3442  3453  3461  3472  3480  3491  3499  3515  3562  3583  3637  3958  3966  3986  4001
        4016  4070  4393  4401  4592  4764  4788  4803  4819  4841  4859  4875  4962  4986  5001  5017
        5042  5095  5167  5180  5196  5348  5360  5374  5386  5456  5464  5546  5500  5592  5600  5692
        5704  5712  5726  5807  5956  6060  6068  6083  6157  6228  6304
?ASAVE  1235#  5460  5466  5722  5728
?B      1280#  4413  4413  4413  4419  4459  4469  4569  4579
?B0PNT  117#  746  754#  763  771#  780  788#  797  805#  814  822#  831  839#
?B0R2   110#  754
?B0R3   111#  771
?B0R4   112#  788
?B0R5   113#  805
?B0R6   114#  822
?B0R7   115#  839
?B1PNT  126#  859  867#  880  888#  901  909#  922  930#  943  951#
?B1R2   119#  867
?B1R3   120#  888
?B1R4   121#  909
?B1R5   122#  930
?B1R6   123#  951
?B1R7   124#
?BCODE  1163#  1561  1561  1561  1567  1610  1616  2390
?BINOP  415#  1817  2387  2439  2909  3632  5179  5359  5385
?BITS0  4217#  4411
?BUFCN  1262#  3438  3444  3511  3517  3532  3542  3962  3968  3982  3988  4044  4054
?BUFLE  649#
?CHARN  585#  5876
?CHKSU  791#  3426  3426  3426  3558  3564  3571  3571  3858  3858  3858  4066  4072  4079  4079
?CONST  104#  585  586  590  594  601  602  606  610  617  618  622  626  633  634  638
        642  649  650  654  658  671  672  676  680  686  687  691  695  701  702  706
        710  716  717  721  725  4217  4218  4222  4226
?CURDI  912#
?DEBNC  617#
?DSPTI  1037#  3092  3098
?DSPTM  808#
?EMAHI  1136#  5388
?EMALO  1127#  5362
?EPACC  965#  2969  2975  3211  3217
?EPPCH  1010#  2761  2777  2912  3354  3360
?EPPCL  1001#  2734  2750  2806  2814  2819  3327  3333
?EPPSW  974#  2792  2798  2839  2845  2894  2900  2939  2945  3249  3255  3284  3290
?EPR0   992#  2924  2930  3268  3274  4855  4861  5060  5082
?EPTIM  983#  2954  2960  3230  3236
?FORM1  295#  1615  1634  1655  1688  1695  1722  1745  1780  1807  1819  1982  2000  2013  2178  2389
        2441  2445  2547  2587  2720  2735  2742  2762  2769  2793  2811  2818  2844  2862  2877  2899
        2911  2929  2944  2959  2974  3008  3069  3097  3212  3231  3250  3269  3289  3307  3328  3355
        3439  3458  3477  3496  3516  3531  3563  3584  3634  3638  3807  3814  3834  3841  3963  3987
        4002  4017  4043  4071  4263  4270  4290  4297  4322  4398  4439  4458  4550  4568  4593  4645
        4652  4765  4789  4804  4820  4842  4860  4876  4963  4987  5002  5018  5043  5061  5068  5096
        5168  5181  5197  5349  5361  5375  5387  5461  5504  5547  5581  5597  5616  5623  5693  5709
        5727  5808  5957  5971  6065  6084  6158  6229  6305
?FORM2  319#  1638  1659  1692  1702  1986  2739  2749  2766  2776  2797  2815  2825  2866  3216  3235
        3254  3273  3311  3332  3359  3443  3462  3481  3500  3811  3821  3838  3848  3967  4267  4277
        4294  4304  4402  4649  4659  5065  5081  5465  5601  5620  5636  5713  6069
?FORM3  339#  1755  2023  2452  2887  3541  3651  4053  4332  4449  4468  4560  4578  5520  5981
```

```
?FORM4  356#
?FORM5  388#  1560  1576  1611  1630  1651  1684  1718  1768  1784  1803  1978  1996  2174  2250  2331
        2466  2484  2543  2583  2716  2731  2758  2789  2807  2840  2858  2895  2925  2940  2955  2970
        3004  3065  3093  3208  3227  3246  3265  3285  3303  3324  3351  3425  3435  3454  3473  3492
        3512  3559  3580  3803  3830  3857  3959  3983  3998  4013  4067  4259  4286  4394  4412  4589
        4641  4761  4785  4800  4816  4838  4856  4872  4959  4983  4998  5014  5039  5057  5092  5164
        5193  5345  5371  5457  5543  5577  5593  5612  5657  5675  5689  5705  5723  5804  5877  5953
        6061  6080  6154  6225  6301
?H      1298#  4262  4278  4323  4333
?HBITH  1028#  4258  4266  4271
?HBITL  1019#  4285  4293  4298
?HEXBU  1324#
?HREGA  1055#
?HREGB  1064#
?HREGC  1073#
?HREGD  1082#
?HREGE  1091#
?HREGF  1100#
?ITMP    774#  1687  1703  1710  1710  1717  1723  1730  1730
?KBDBU  1208#  2332  2332  2332  2338  5615  5637  5803  5809  6153  6159
?KEY     757#  2582  2588  2595  2595
?KEYFL   933#
?KEYLO  1217#  5542  5548  5658  5658  5658  5664
?LASTK   891#  5611  5619  5624  5631  5631  5676  5676  5676
?LDATA   740#  2810  2826  2833  2833  3633  3639  3646  3646  3652  3658  3658  4644  4660  4667  4667
        5056  5064  5069  5076  5076
?LENGT  1333#  1388  1399#  1442  1527#  1866  1876#  1936  1946#  2146  2156#  2223  2233#  2289  2299#  2356
        2366#  2507  2517#  2640  2650#  2667  2678#  3137  3147#  3383  3398#  3606  3616#  3674  3684#  3723
        3734#  3758  3770#  3909  3920#  4095  4105#  4130  4140#  4169  4179#  4206  4233#  4348  4359#  4484
        4494#  4608  4619#  4684  4694#  4713  4723#  4906  4916#  5127  5137#  5213  5224#  5253  5263#  5298
        5308#  5402  5413#  5750  5760#  5829  5839#  5898  5908#  6022  6032#  6100  6110#  6173  6184#  6248
        6258#  6323  6335#  6391  6401#  6464  6476#  6530  6541#  6601  6612#  6675  6685#  6748  6767#  6816
        6826#  6874  6884#  6935  6945#  6996  7007#  7063  7073#  7123
?MEMHI  1154#  3806  3822  3997  4003
?MEMLO  1145#  3833  3849  4012  4018  4640  4648  4653
?MINDX   156#   967   971#   971   976   980#   980   985   989#   989   994   998#   998  1003  1007#  1007
        1012  1016#  1016  1021  1025#  1025  1030  1034#  1034  1039  1043#  1043  1048  1052#  1052  1057
        1061#  1061  1066  1070#  1070  1075  1079#  1079  1084  1088#  1088  1093  1097#  1097  1102  1106#
        1106  1111  1115#  1115  1120  1124#  1124  1129  1133#  1133  1138  1142#  1142  1147  1151#  1151
        1156  1160#  1160  1165  1169#  1169  1174  1178#  1178  1183  1187#  1187  1192  1196#  1196  1201
        1205#  1205  1210  1214#  1214  1219  1223#  1223  1228  1232#  1232  1237  1241#  1241  1246  1250#
        1250  1255  1259#  1259  1264  1268#  1268  1273  1277#  1277  1282  1286#  1286  1291  1295#  1295
        1300  1304#  1304  1309  1313#  1313  1317  1321#  1321  1325  1329#  1329
?MSAVE   158#   587   603   619   635   651   673   688   703   718   742   759   776   793   810   827
         851   872   893   914   935   967   976   985   994  1003  1012  1021  1030  1039  1048  1057
        1066  1075  1084  1093  1102  1111  1120  1129  1138  1147  1156  1165  1174  1183  1192  1201
        1210  1219  1228  1237  1246  1255  1264  1273  1282  1291  1300  1309  1317  1325  4219
?NCOLS   601#
?NEG1    716#  2330  5674
?NEXTP  1199#  2251  2251  2251  2257  5878  5878  5878  5884  5952  5958  5972  5982
?NREPT  1226#  5576  5582  5596  5602
?NUMCO  1181#  1654  1660  2173  2179  2467  2407  2467  2473  2715  2721
?OPTIO  1190#  1633  1639  1683  1691  1696  1783  1789  1802  1808
?OVBUF  1316#
?OVSIZ   633#
?PLUS1   686#  2465
?PLUS3   701#  2249
```

```
?R1      99#  4289  4305  4312  4312
?RAM    103#   965   966   974   975   983   984   992   993  1001  1002  1010  1011  1019  1020  1028
        1029  1037  1038  1046  1047  1055  1056  1064  1065  1073  1074  1082  1083  1091  1092  1100
        1101  1109  1110  1118  1119  1127  1128  1136  1137  1145  1146  1154  1155  1163  1164  1172
        1173  1181  1182  1190  1191  1199  1200  1208  1209  1217  1218  1226  1227  1235  1236  1244
        1245  1253  1254  1262  1263  1271  1272  1280  1281  1289  1290  1298  1299
?RB0    101#   740   741   745   757   758   762   774   775   779   791   792   796   808   809   813
         825   826   830
?RB1    102#   849   850   854   858   870   871   875   879   891   892   896   900   912   913   917
         921   933   934   938   942
?RDELA 1244#  5688  5694  5708  5714  6064  6070  6079  6085
?RECTY 1271#  3495  3501  3579  3585
?REGC  1289#  4397  4403  4440  4450  4551  4561  4508  4594
?ROTCN  870#
?ROTPA  849#  5505  5512  5512  5521  5527  5527
?RSAVE  144#   591   595   607   611   623   627   639   643   655   659   677   681   692   696   707
         711   722   726   746   763   780   797   814   831   855   859   876   880   897   901   918
         922   939   943  4223  4227
?SEGMA 1308#
?SIZE   255#  1383  1437  1861  1931  2141  2218  2284  2351  2502  2635  2662  3132  3378  3601  3669
        3718  3753  3904  4090  4125  4164  4201  4343  4479  4603  4679  4708  4901  5122  5208  5248
        5293  5397  5745  5824  5893  6017  6095  6168  6243  6318  6386  6459  6525  6596  6670  6743
        6811  6869  6930  6991  7058  7118
?SMAHI 1118#  2405  2485  2485  2491  2757  2765  2770  3457  3463  3802  3810  3815  4784  4790  4982
        4988  5182  5370  5376
?SMALO 1109#  2730  2738  2743  2861  2867  2878  2888  3306  3312  3476  3482  3829  3837  3842  4799
        4805  4815  4821  4837  4843  4871  4877  4997  5003  5013  5019  5038  5044  5091  5097  5192
        5198  5344  5350  6224  6230  6300  6306
?START 1339#  1383  1383  1391  1405#  1437  1437  1445  1533#  1861  1861  1869  1882#  1931  1931  1939
        1957#  2141  2141  2149  2162#  2218  2218  2226  2244#  2284  2284  2292  2310#  2351  2351  2359
        2382#  2502  2502  2510  2533#  2635  2635  2643  2656#  2662  2662  2670  2699#  3132  3132  3140
        3173#  3378  3378  3386  3414#  3601  3601  3609  3622#  3669  3669  3677  3695#  3718  3718  3726
        3745#  3753  3753  3761  3796#  3904  3904  3912  3951#  4090  4090  4098  4116#  4125  4125  4133
        4151#  4164  4164  4172  4190#  4201  4201  4209  4254#  4343  4343  4351  4385#  4479  4479  4487
        4525#  4603  4603  4611  4635#  4679  4679  4687  4700#  4708  4708  4716  4754#  4901  4901  4909
        4952#  5122  5122  5130  5158#  5208  5208  5216  5235#  5248  5248  5256  5279#  5293  5293  5301
        5334#  5397  5397  5405  5449#  5745  5745  5753  5791#  5824  5824  5832  5865#  5893  5893  5901
        5939#  6017  6017  6025  6053#  6095  6095  6103  6146#  6168  6168  6176  6220#  6243  6243  6251
        6294#  6318  6318  6326  6371#  6386  6386  6394  6437#  6459  6459  6467  6512#  6525  6525  6533
        6582#  6596  6596  6604  6653#  6670  6670  6678  6726#  6743  6743  6751  6773#  6811  6811  6819
        6832#  6869  6869  6877  6890#  6930  6930  6938  6951#  6991  6991  6999  7048#  7058  7058  7066
        7114#  7118  7118  7126
?STRTN 1253#  1981  1987  1995  2001  2014  2024
?TYPE  1172#  1577  1577  1577  1583  1746  1756  1769  1769  1769  1775  1820  2440  2446  2453  2542
        2548  3003  3009  3064  3070  4760  4766  4958  4964  5163  5169
?UNARY  459#  1744  2012  2876  3530  4042  4321  4438  4457  4549  4567  5503  5970
?VERSN 1046#
?XPCOD  825#
?ZERO   671#  1559  1575  1767  2483  3424  3856  5656
AFETCH 4704  4759#
ASAVE  1239#  5468  5730
ASCERR 3704  3711  3715#
B      1284#  4415  4421  4461  4529  4571
BCODE  1167#  1563  1569  1598  1618  2392
BITSO  4230#  4422
BRKEND 2462  2500#
BRKERR 3080  3080#
```

```
LOC  OBJ          LINE        SOURCE STATEMENT

                  7243        END

USER SYMBOLS
?A       0004   ?ASAVE 0002   ?B       0002   ?B0PNT 0008   ?B0R2  0003   ?B0R3  0004   ?B0R4  0005   ?B0R5  0006
?B0R6    0007   ?B0R7  0008   ?B1PNT 0007   ?B1R2  0003   ?B1R3  0004   ?B1R4  0005   ?B1R5  0006   ?B1R6  0007
?B1R7    0008   ?BCODE 0002   ?BINOP 0022   ?BITS0 0003   ?BUFCN 0002   ?BUFLE 0003   ?CHARN 0003   ?CHKSU 0000
?CONST   0003   ?CURDI 0001   ?DEBNC 0003   ?DSPTI 0002   ?DSPTM 0000   ?EMAHI 0002   ?EMALO 0002   ?EPACC 0002
?EPPCH   0002   ?EPPCL 0002   ?EPPSW 0002   ?EPR0  0002   ?EPTIM 0002   ?FORM1 0016   ?FORM2 0018   ?FORM3 001A
?FORM4   001C   ?FORM5 001E   ?H      0002   ?HBITH 0002   ?HBITL 0002   ?HEXBU 0003   ?HREGA 0002   ?HREGB 0002
?HREGC   0002   ?HREGD 0002   ?HREGE 0002   ?HREGF 0002   ?ITMP  0000   ?KBDBU 0002   ?KEY   0000   ?KEYFL 0001
?KEYLO   0002   ?LASTK 0001   ?LDATA 0000   ?LENGT 000D   ?MEMHI 0002   ?MEMLO 0002   ?MINDX 0075   ?MSAVE 0001
?NCOLS   0003   ?NEG1  0003   ?NEXTP 0002   ?NREPT 0002   ?NUMCO 0002   ?OPTIO 0002   ?OVBUF 0003   ?OVSIZ 0003
?PLUS1   0003   ?PLUS3 0003   ?R1     0000   ?RAM   0002   ?RB0   0000   ?RB1   0001   ?RDELA 0002   ?RECTY 0002
?REGC    0002   ?ROTCN 0001   ?ROTPA 0001   ?RSAVE 0000   ?SEGMA 0003   ?SIZE  000E   ?SMAHI 0002   ?SMALO 0002
?START   03DC   ?STRTM 0002   ?TYPE   0002   ?UNARY 002A   ?VERSN 0002   ?XPCOD 0000   ?ZERO  0003   AFETCH 0678
ASAVE    003E   ASCERR 01C9   B      0043   BCODE  0036   BITS0  000B   BRKEND 024D   BRKERR 04A6   BRKFIL 022E
BRKNXT   0234   BUFCNT 0041   BUFLEN 0010   BYTEI1 00F2   BYTEIN 00F0   BYTEO  01DB   CGO    046B   CGONB  047C
CGOPAT   0476   CGOSS  0480   CGOTRA 0480   CGOWB  0476   CHARCK 000D   CHARIN 01CD   CHARLF 000A   CHARNO 0008
CHARO    05BD   CHKERR 02E1   CHKSUM 0005   CI0    064D   CI1    0651   CI2    0659   CI3    0662   CI4    0665
CIN      0649   CKSMOK 02DB   CLEAR  05F1   CLRBFT 0008   CMDINT 00BA   CMFMAS 05E2   CMPRET 05F0   CNTRLZ 001A
CNTTBL   04A1   CNTTRA 04AA   CO1    05C5   CO2    05CB   CO3    05CF   CODEBL 0006   COMCBR 0228   COMFIL 02E5
COMGOR   0461   COMSBR 022C   COMSIZ 0003   CTAB   0023   CURDIG 0005   DATABL 000C   DATO   062C   DATO1  062E
DBLANK   05F5   DBPNT  0144   DBRK   0015   DCB    015A   DDABRK 0167   DDAMEM 0161   DEBNCE 0008   DECLAR 0003
DECSM1   02FF   DECSMA 02F4   DELAY  04F2   DELAY1 04F5   DERROR 0131   DFILL  014B   DGO    0149   DGPATS 00EF
DGR      015D   DINTRG 0169   DLST   014E   DMOD   0146   DNOBRK 01CB   DONE   02E0   DPA    0172   DPRBRK 0165
DPRMEM   015F   DREC   0151   DREL   0154   DRM    0163   DRUN   013E   DSB    0157   DSGNON 0137   DSPACC 06D3
DSPHI    018E   DSPLO  0194   DSPM1  0192   DSPMID 0190   DSPTIM 0028   DSPTMP 0006   DSS    016F   DTR    0175
DWBRK    016D   ELSIF1 00D7   ELSIF2 00E5   EMAHI  0033   EMALO  0032   ENBLNK 0002   ENBRAM 0001   ENDF1  059E
ENDFIL   0596   ENDREC 0641   EOFREC 05AE   EPACC  0020   EPBRK  034F   EPCNT  0441   EPCON1 041F   EPCONT 0415
EPFET    07B7   EPPASS 07D0   EPPCHI 0025   EPPCLO 0024   EPPSW  0021   EPR0   0023   EPREL  07F4   EPRET  04C7
EPRSET   0010   EPRUN  0400   EPRUN1 048A   EPRUN2 0499   EPRUN3 0495   EPRUN4 0482   EPRUN5 04B3   EPRUN6 04BA
EPSSTP   0004   EPSTE1 07DF   EPSTE2 07F1   EPSTEP 07DB   EPSTOR 07C3   EPTIMR 0022   ERROR2 01B6   EXAM0  0250
EXAM1    027B   EXAM2  0261   EXAM3  028A   EXAM4  0293   EXAM5  0275   EXAMIN 024F   EXPMON 0080   FDUMP1 0617
FDUMP2   0628   FDUMP3 0636   FDUMP4 0648   FDUMP5 0632   FINDOP 0042   GOTBL  0471   H      0045   HBD1   04D7
HBD2     04D5   HBDLAY 04C9   HBITHI 0027   HBITLO 0026   HDATIN 02B9   HEXASC 01E6   HEXBUF 0065   HEXNIB 01EF
HFDONE   05A7   HFILEO 0572   HRECIN 0297   HRECO  0600   HREGA  002A   HRLGB  002B   HREGC  002C   HREGD  002D
HREGE    002E   HREGF  002F   IMPLEM 0200   INCSMR 01F2   INCW   01F4   INCW1  01FC   INIT   0000   INITLP 000E
INPAD1   00C7   INPADR 00C8   INPKEY 00EC   INVALS 0300   ITMP   0004   JGORES 0226   JMPTBL 0206   JTOFIL 0222
JTOGO    0220   JTOLST 021A   JTOMOD 020F   JTOREC 0211   JTOREL 0216   KBDBUF 003B   KBDI1  06C6   KBDIN  06C2
KBDPOL   07AF   KCLRB  000C   KEY    0003   KEYCLR 0017   KEYDM  0016   KEYEND 0013   KEYFIL 0010   KEYFLG 0006
KEYGO    001E   KEYLOC 003C   KEYLST 001C   KEYMOD 001F   KEYNXT 0012   KEYPAT 0015   KEYPM  001A   KEYREC 0018
KEYREG   001B   KEYREL 0014   KEYTRA 0019   KGORES 001D   KSETB  000B   LASTKY 0004   LDATA  0002   LDBYTE 0582
LFEBR1   06C1   LFEBRK 06B1   LFEDM  0698   LFEINT 06A9   LFEPM  0684   LFER0  06A5   LFEREG 069C   LFETBL 067E
LFETCH   00FC   LFILL  02E9   LFILL1 02F3   LPGSEL 04E1   LSTBR1 0746   LSTBR2 0748   LSTBRK 073D   LSTDM  0721
LSTINT   0734   LSTORE 0700   LSTPM  070C   LSTR0  072F   LSTREG 0726   LSTTBL 0706   M0     0010   M1     0020
MADD     0024   MADDC  0025   MAIN   0029   MAIN2  0033   MAINA  0052   MAINB  0069   MAINB0 009E   MAINB1 00A0
MAINC1   0075   MAIND  0093   MAIND1 0087   MANL   0026   MBLOCK 0002   MDEC   002C   MDJNZ  002D   MEMHI  0035
MEMLO    0034   MERROR 008C   MINC   002B   MMLO1  036F   MMOV   0020   MODOUT 0020   MORL   0027   MPUSEL 0040
MRL      002E   MRLC   0031   MRR    002F   MRRC   0030   MXCH   0029   MXRL   0028   NCOLS  0004   NEG1   FFFF
NEXTPL   003A   NIBI3  01C2   NIBIN  01B8   NIBIN2 01BA   NIBO   05BB   NOBRK  001B   NOVALS 0023   NREPTS 003D
NUMCON   0038   NXTLOC 0768   OPTAB1 033F   OPTAB2 0346   OPTAB3 0349   OPTION 0039   ORGPG0 0100   ORGPG1 01FD
ORGPG2   0300   ORGPG3 03E9   ORGPG4 04FD   ORGPG5 05FF   ORGPG6 06FF   ORGPG7 07FD   OUTCLR 0102   OUTMSG 0104
OUTUTL   0100   OV0BAS 0378   OV1B1  000A   OV1B2  0313   OV1BAS 03A4   OV2B1  0313   OV2BAS 03BA   OV3B1  0311
OV3BAS   038E   OVBUF  004E   OVLOAD 036A   OVSIZE 0017   OVSW1  0361   OVSWAP 035A   PBRK   0019   PDIGIT 000E
PERROR   019A   PGSIZE 00FD   PINPUT 000B   PLUS1  0001   PLUS3  0003   PRNT1  0117   PRNT2  0108   PSEGHI 000D
```

Appendix C and D

APPENDIX C
COMMAND SUMMARY

The following is a summary of the commands implemented by the HSE-49 emulator monitor. Within each command group, tokens in each column indicate options the user has when invoking those commands.

Tokens in square brackets indicate dedicated keys on the keyboard (some keys having shared functions); angle brackets enclose hex digit strings used to specify an address or data parameter. Parameters in parentheses are optional, with the effects explained above. The notation used is as follows:

<SMA> — Starting Memory Address for block command,
<EMA> — Ending Memory Address for block command,
<LOC> — LOCation for individual accesses,
<DATA> — DATA byte.

Asterisks (*) indicate the default condition for each command; thus that token is optional and serves to regularize the command syntax.

Program/data entry and verification commands:

[EXAM]	[PROG MEM]*	<LOC> [,]	[NEXT]
	[DATA MEM]		[PREV]
	[REGISTER]		[.]
	[HWRE REG]		
	[PROG BRK]		
	[DATA BRK]		

Program/data initialization commands:

[FILL]	[PROG MEM]*	<SMA>	[,]	<EMA>	[,]	<DATA> [.]
	[DATA MEM]					
	[REGISTER]					
	[HWRE REG]					
	[PROG BRK]					
	[DATA BRK]					

Intellec® development system or TTY interface commands (for transferring HEX format files):

[UPLOAD]	[PROG MEM]*	<SMA>	[,]	<EMA>	[.]
	[DATA MEM]				
	[REGISTER]				
	[HWRE REG]				
	[PROG BRK]				
	[DATA BRK]				

[DNLOAD]	[PROG MEM]* [.]
	[DATA MEM]
	[REGISTER]
	[HWRE REG]
	[PROG BRK]
	[DATA BRK]

Formatted data dump to TTY or CRT:

[LIST]	[PROG MEM]*	<SMA>	[,]	<EMA>	[.]
	[DATA MEM]				
	[REGISTER]				
	[HWRE REG]				
	[PROG BRK]				
	[DATA BRK]				

Program execution commands:

[GO]	[NO BREAK]*	(<SMA>)	[.]
	[W/ BREAK]		[,]
	[SING STP]		
	[AUTO BRK]		
	[AUTO STP]		
[GO/RST]	[NO BREAK]*	[.]	
	[W/ BREAK]		
	[SING STP]		
	[AUTO BRK]		
	[AUTO STP]		

Breakpoint setting and clearing:

[SET BRK]	[PROG MEM]*	<LOC>	([,] <LOC> ...)	[.]
	[DATA MEM]			
[CLR BRK]	[PROG MEM]*	<LOC>	([,] <LOC> ...)	[.]
	[DATA MEM]			

APPENDIX D
ERROR MESSAGES

The following error message codes are used by the monitor software to report an operator or hardware error. Errors may be cleared by pressing [CLR/PREV] or [END/.]. The format used for reporting errors is "Error – .n" where "n" is a hex digit.

Operator Errors

1. Illegal command initiator.
2. Illegal command modifier or parameter digit.
3. Illegal terminator for Examine command.
4. Illegal attempt to clear Error mode.

5-9. Not used.

Hardware Errors

A. ASCII error — non-hex digit encountered in data field of hex format record.

B. Breakpoint error. Break logic activated though breakpoints not enabled.

C. Hex format record checksum error. Note — the checksum will not be verified if the first character of the checksum field is a question mark ("?") rather than a hexidecimal digit. This allows object files to be patched using the ISIS text editor without the necessity of manually recomputing the checksum value.

D. Not used.

E. Execution processor failed to respond to a command or parameter passed to it by the master processor. EP automatically reset. EP internal status may be lost. Program memory not affected.

F. Not used.

Using the 8049 as an 80 Column Printer Controller

Contents

INTRODUCTION 5-218

PRINT MECHANISM DESCRIPTION 5-218

INTERFACE CIRCUITRY 5-219

SOFTWARE 5-221

HANDLING THE I/O BUFFER 5-222

TIMING 5-222

CONCLUSION 5-224

APPENDIX A. SCHEMATIC DIAGRAM 5-225

APPENDIX B. MONITOR LISTING 5-226

USING THE 8049 AS AN 80 COLUMN PRINTER CONTROLLER

I. INTRODUCTION

This Application Note details using INTEL's 8049 microcomputer as a dot matrix printer controller. Previous INTEL Application notes, (e.g. AP-27 and AP-54) described using intelligent processors and peripherals to control single printer mechanisms. This Application note expands upon the theme established in these prior notes and extends the concept to include a complete bi-directional 80 column printer using a single line buffer. For convenience this application note is divided into six sections:

1. INTRODUCTION
2. PRINT MECHANISM DESCRIPTION
3. INTERFACE CIRCUITRY
4. SOFTWARE
5. CONCLUSION
6. APPENDIX

Over the last few years 80 column output devices have become somewhat of a defacto output standard for business and some data processing applications. It should be mentioned that by no means is the 80 column format a "new" standard. 80 column computer cards have been around for more than 20 years and perhaps the existence of these cards in the early days of computers is why the 80 column format is a standard today.

Many CRT terminals use the 80 by N format and to complement this a number of printers use this same format. One reason, aside from those historic in nature, for the 80 column standard is that 80 columns of 12 pitch text on standard typewritten 8.5 inch by 11 inch paper completely fills up an entire line and allow ample room for margins. So, the 80 column format is an aesthetically convenient format.

Printers are usually divided into either impact or non-impact and a character or line oriented device. Impact printers actually use some type of "striker" to place ink on the paper. More often than not the ink is contained on a ribbon which is placed between the striker and the paper. Non-impact printers use some means other than direct pressure to place the characters on the paper. This type of printer is very fast because there is very little mechanical motion associated with placing the characters on the paper. However, because the paper is required to be treated with a special substance, it is not as convenient as an impact printer.

Character printers are capable of printing one character at a time. (Any standard home typewriter is in effect a character printer.) Line printers must print an entire line at a time. Line printers are usually quite a bit faster than character printers, but they usually don't offer the print quality of character printers.

In recent years, the "computer boom" has caused the price of printers to tumble markedly. High volume production, competition, and the tremendous demand for reliable print mechanisms have all contributed to the decrease in price. Because of their simplicity, line printer mechanisms have decreased in price faster than other mechanisms. Therefore, when high quality print is not needed, a line printer is a very attractive choice.

This application note describes how to control an 80 column impact-line printer with an 8049/8039. The complete software listing is included in the appendix. The 8049 is the high-performance member of the MCS-48™ microcontroller family. The Processor has all of the features of the 8048 plus twice the amount of program and data memory and an 11MHz clock speed. For details about the 8049, please refer to the MCS-48 user's manual.

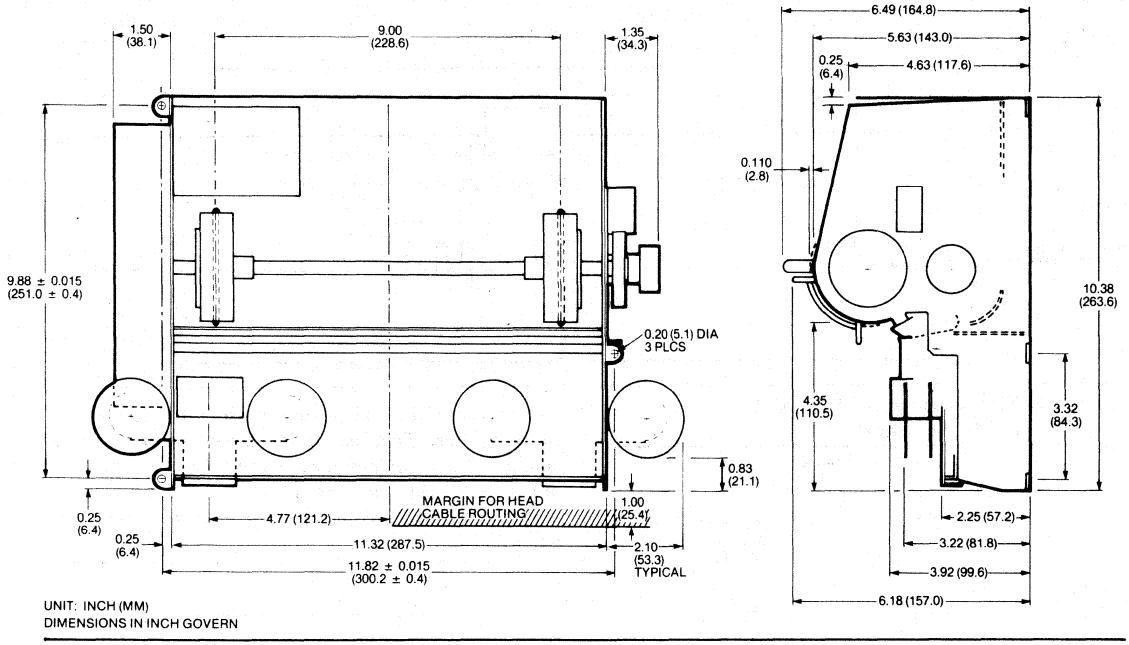
II. PRINT MECHANISM DESCRIPTION

The model 820 printer is available from C. ITOH ELECTRONICS (5301 BEETHOVEN STREET, LOS ANGELES, CA 90066). This inexpensive and simple printer is ideal for applications requiring 80 columns of dot matrix alpha-numeric information.

The model 820 printer is comprised of three basic sub-assemblies; the chassis or frame, the paper feed mechanism, and the print head. The diagram in Figure 2.1 gives the physical dimensions of the basic print mechanism. The basic chassis for the printer is constructed out of four sheet metal stampings. These stampings are screwed together to form a sturdy base on which all other components of the printer are mounted.

The paper feed mechanism consists of a toothed wheel, a solenoid, a tension spring, and a "catcher." When the solenoid is activated, the arm of the solenoid pulls against the spring and drags over the toothed wheel. When the solenoid is released, its arm is pulled by the spring, but this time the arm grabs a tooth on the wheel and pulls the wheel forward which advances the paper. A "catcher," which is merely a piece of plastic held against the toothed wheel, is added to assure that the paper is advanced only one "tooth" position each time the solenoid is activated.

The print head is comprised of seven solenoids which are mounted in a common housing. The solenoids are physically mounted in a circle, but their hammers are positioned linearly along the vertical axis. These seven vertically positioned hammers are the strikers that actually do the printing.



UNIT: INCH (MM)
DIMENSIONS IN INCH GOVERN

Figure 2.1 Physical Dimensions of C. ITOH Model 820 Printer

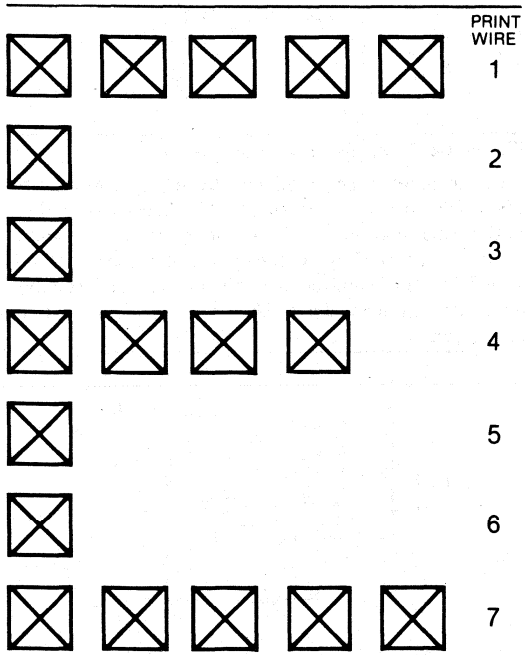


Figure 2.2 "Formation" of a Character by a Dot Matrix Printer

A motor, mounted toward the back of the print mechanism, drives a rubber toothed belt which turns a roller guide. A motor turns a guide that moves the print head from right to left and left to right. By properly timing the current flow through the solenoids while the print head is moving across the paper, characters can be formed. Figure 2.2 illustrates how the dot matrix printer "forms" its characters.

The timing pulses for the print head mechanism are generated by an opto-electronic sensor. This sensor, located on the left side plate of the printer, informs the print controller when to apply current to the print head mechanism. This "on-board timing wheel" assures that all characters will be properly spaced and that they will all be "in-line" in a vertical sense.

The print mechanism is also equipped with two additional sensors. These are the left home position sensor, located near the left front of the mechanism, and the right home position sensor, located near the right front of the print mechanism. These sensors simply tell the controller when the print head is in either the left or right home position. A complete timing chart for the printer is shown in Figure 2.3.

III. INTERFACE CIRCUITRY

The manual supplied with the printer recommends some specific interface circuitry. For the most part the circuitry used in this Application Note followed these suggestions. The circuitry needed to drive the print head solenoid is shown in Figure 3.1. This same

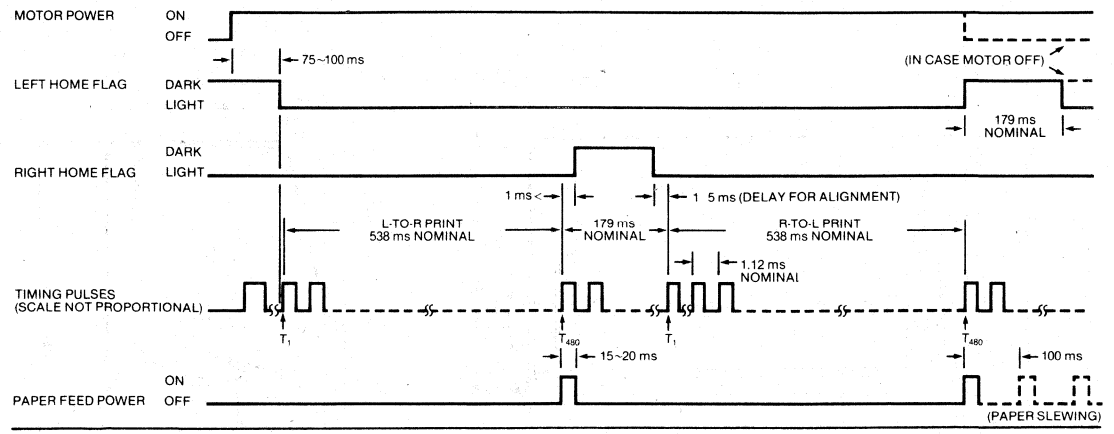


Figure 2.3 Timing Diagram of C. ITOH Model 820 Printer

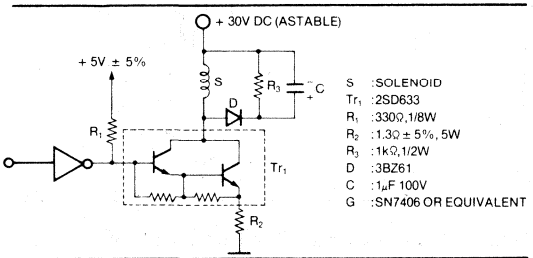


Figure 3.1 Solenoid Drive Circuit (Eliminate R2 for Line Feed Solenoid)

circuit is used to drive the line feed solenoid except that the current limiting resistor R2 is eliminated. This resistor is not needed because the line feed solenoid is physically much larger than the print head solenoids and can tolerate much higher levels of current.

The print head drivers are connected to an 8212 latch. The latch is interfaced to the BUS PORT on the 8049 and is enabled whenever the WR pin and the BIT 4 of PORT 1 are coincidentally low. The line feed driver is connected to PORT 1 BIT 1 of the 8049.

Note that the driver is simply a Darlington transistor that is driven by an open collector TTL gate. Resistor R2 is the current limiting resistor and diode D, capacitor C, and resistor R3 are used to "dampen" the inductive spike that occurs when driving solenoid S. This circuit is repeated for each of the seven solenoids in the print head. It should be mentioned that, although the type of Darlington transistor needed to drive the print head is not critical, a collector current rating of at least 5 amps and a breakdown voltage (Vceo) of at least 100 volts is needed. Transistors that do not meet these requirements will be damaged by the inductive kickback of the solenoids.

As mentioned in Section 2, the printer provides some sensor interface signals that are derived via three opto-electronic sensors. These signals must be amplified and converted to TTL levels in order to interface to the controller. This conversion is accomplished with a simple voltage comparator. Figure 3.2 is a schematic of the sensor interface circuitry. Note that hysterisis is employed on the voltage comparators. This eliminates "false" sensing.

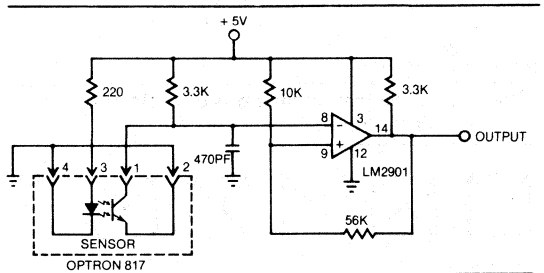


Figure 3.2 Example of Sensor Circuit

Motor control is accomplished by using a Monsanto MCS-6200 optically-coupled TRIAC. This part is ideal in this kind of application because it provides a simple means of controlling a line-operated motor without sacrificing the isolation needed for safe and reliable operation. Figure 3.3 is a schematic of the motor driving circuit.

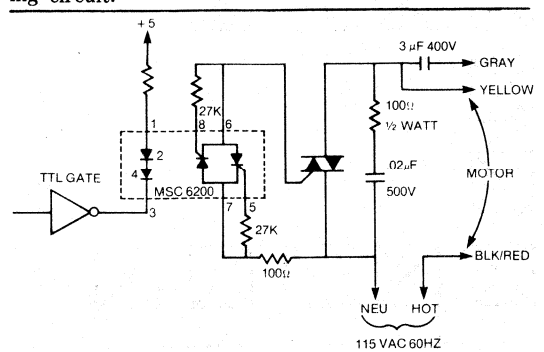


Figure 3.3 Motor Driving Circuit

To interface 8049 to the outside world one 8212 latch was used. This latch was connected to the BUS PORT and is enabled by an INS or MOVX instruction coincident with BIT 4 of PORT 1 being in a logical zero state. In this configuration, the 8212 was used to hold the data until read by the 8049. The connection of the 8212 to the 8049 is shown in Figure 3.4 and the parallel port timing diagram is shown in Figure 3.5. The 8212 parallel port was connected to the LINE PRINTER OUTPUT of an INTELLEC MICROCOMPUTER DEVELOPMENT SYSTEM.

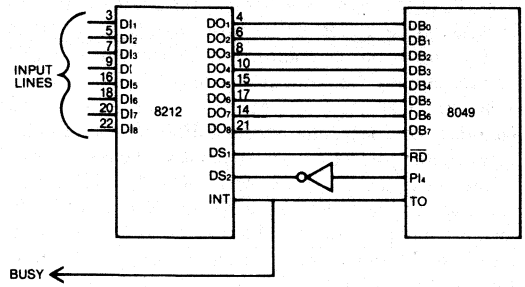


Figure 3.4 Connection of the 8212 Input Port to the 8049

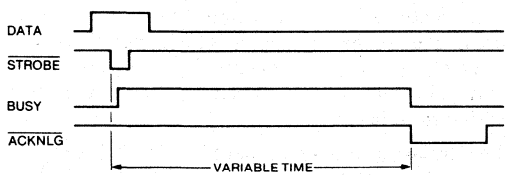


Figure 3.5 Parallel Port Timing

IV. SOFTWARE

As mentioned in Section 2, the bulk of the timing needed to control the printer is actually generated by the printer itself. Therefore, all the software must do is harness these timing signals and turn on and off the right solenoids at the right time.

To make things easy, the software needed to drive the printer is broken into four separate routines. These are:

1. INITIALIZATION ROUTINE
2. INPUT ROUTINE
3. OUTPUT ROUTINE
4. LOOKUP ROUTINE

The INITIALIZATION ROUTINE turns the motor on and checks the opto-electronic sensors. If a failure is found, the routine turns off the motor and loops on itself. This insures that the print mechanism is cycled properly before characters are accepted for printing. This routine also initializes all of the variables used by the printer.

The INPUT ROUTINE reads the characters that are present in the 8212 input port and writes them into the 8049's buffer memory. The routine then checks the characters to see if a CARRIAGE RETURN (ASCII OCH) has been transmitted. If a CR is detected, the input routine automatically inserts a LINE FEED as the next character. When the input routine detects a LINE FEED, it stops reading characters and sets the direction bits and the print bit in the status register. This action evokes the OUTPUT ROUTINE. A detailed flowchart of the INPUT ROUTINE is shown in Figure 4.1.

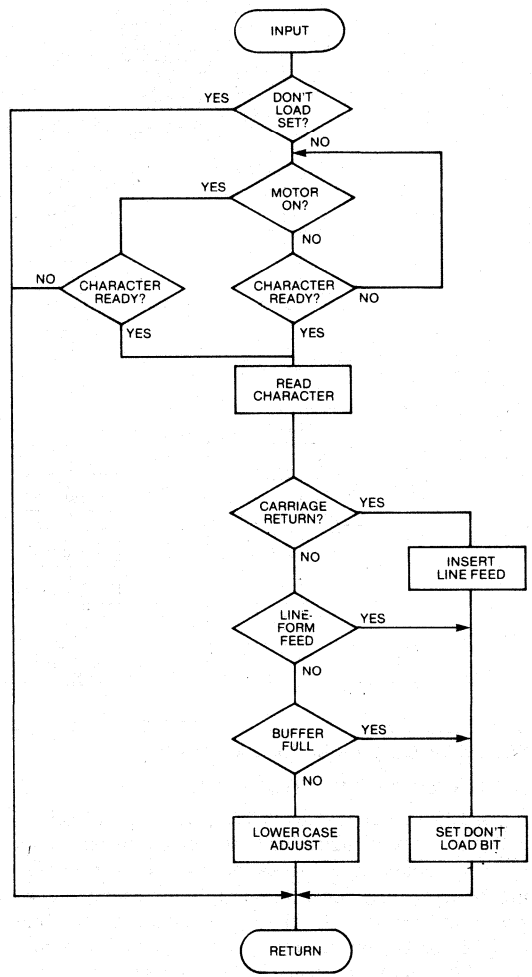


Figure 4.1 Input Routine Flowchart

The OUTPUT ROUTINE initializes both the input and output buffer pointers and then reads the characters from the 8049's buffer memory. After a character is read the OUTPUT ROUTINE calls the LOOKUP ROUTINE which reads the proper bit pattern to form that character. This bit pattern is then used to strobe the solenoids. After each character is printed, the OUTPUT ROUTINE calls the INPUT ROUTINE and another character is placed into the buffer memory. This type of operation guarantees that the input buffer cannot "overrun" the output buffer. A flowchart of the OUTPUT ROUTINE is shown in Figure 4.2.

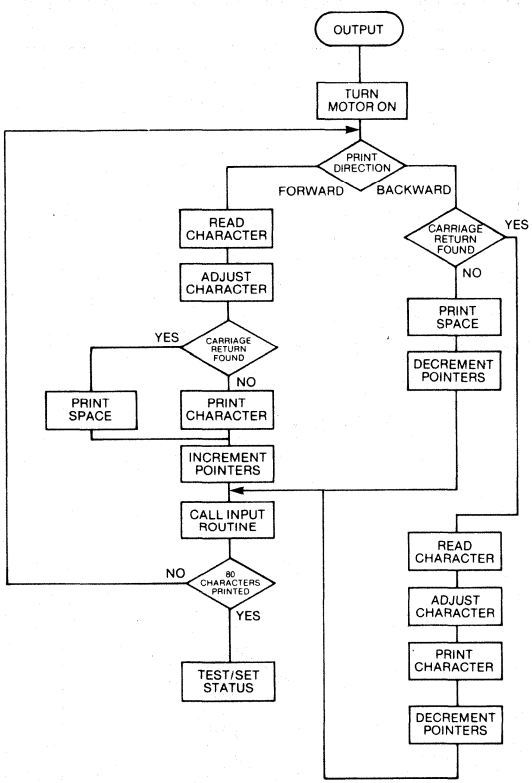


Figure 4.2 Output Routine Flowchart

IV-I. HANDLING THE I/O BUFFER

Since the C. ITOH Model 820 printer is capable of printing in both directions the 80 character buffer must be manipulated in a manner as to allow maximum input-output efficiency. This is accomplished by reversing the "direction" of the buffer memory each time the printer is printing from right to left. For simplicity, if it is assumed that the buffer is only five bytes long, Figure 4.3 can be used to help explain the buffer operation.

Initially the input buffer pointer is loaded with the address of the first location in the buffer memory. As characters are read, the input buffer pointer increments and fills the buffer memory as shown in Figure 4.3(b) through 4.3(f). When a CARRIAGE RETURN-LINE FEED (CRLF) is encountered the input buffer pointer and the output buffer pointer are reset back to the first location. The OUTPUT ROUTINE then reads the character from the first location in the buffer memory, increments the output buffer pointer and calls the INPUT ROUTINE, which reads another character from the parallel input port.

The OUTPUT ROUTINE reads the entire buffer, inserting space codes (20H) after a CR is detected, and the input buffer pointer follows the output buffer pointer as they "increment" up to the buffer memory. When the OUTPUT ROUTINE has printed the last character or space, the output buffer pointer and the input buffer pointer are set to point at the last location of the buffer memory. The OUTPUT ROUTINE then reads the character from the last location of the buffer memory and proceeds to "decrement" down the buffer memory. Space codes are inserted until a CR is found. Figure 4.3(1) to 4.3(0).

The input buffer pointer follows the output buffer pointer just as in the previous case. When the last, or in this case the first character is printed, the output buffer pointer and the input buffer pointer are set to point at the last location of the buffer memory. Now the pointers are "decrementing" down the buffer memory, but the printer is actually printing in a "normal" left to right fashion.

When the last character or space is printed, the output buffer and the input buffer pointer are set to the first location of the buffer memory and printing takes place in a reverse or right to left manner. After this line is printed, the print head and both buffer pointers are in the same position as they were initially. So, four lines must be printed before the buffer pointers and the print head complete a cycle. Each of these situations is handled separately by four different subroutines: CASE0, CASE1, CASE2, and CASE3.

IV-II. TIMING

All critical timing for the printer controller came from two basic sources; the timing sensors on the printer and the internal eight-bit timer of the 8049.

The internal timer of the 8049 was used to control the length of time the solenoids were fired (600 microseconds) and was also used as a "one-shot" to align the printer. This alignment is needed to make the "backward" printing line up vertically with the normal or forward printing. The "one-shot" is used to measure the time from the last column of the last character position until the right sensor flag is covered.

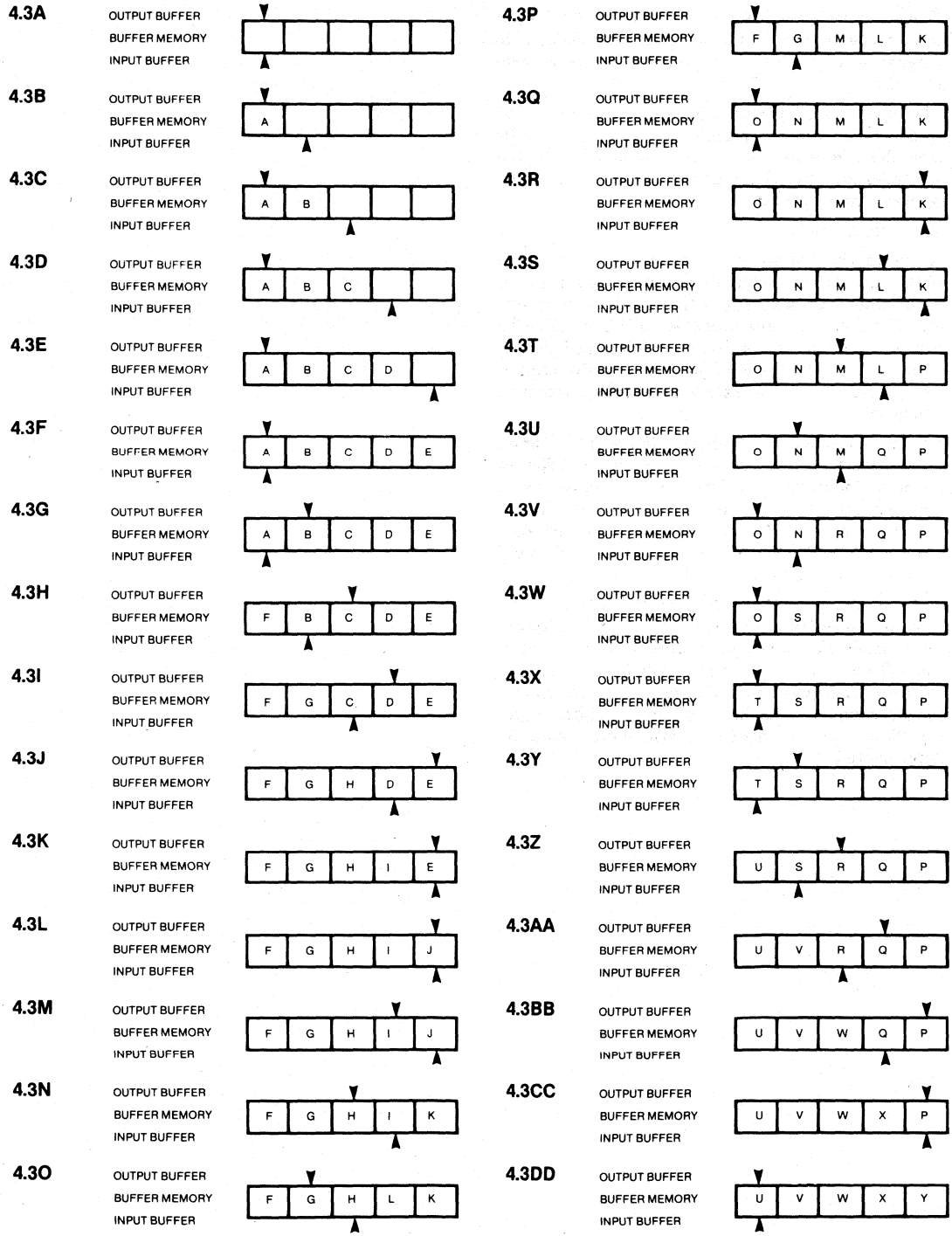


Figure 4.3 I/O Buffer Handler

When the print head reverses direction and the right sensor flag is uncovered, the timer is then used to determine where to start printing in the reverse direction.

The timer and the print wheel on the printer are used to determine when to place a character. The strobe from the print wheel informs the 8049 when to fire the solenoids and the timer allows the proper spacing between the characters.

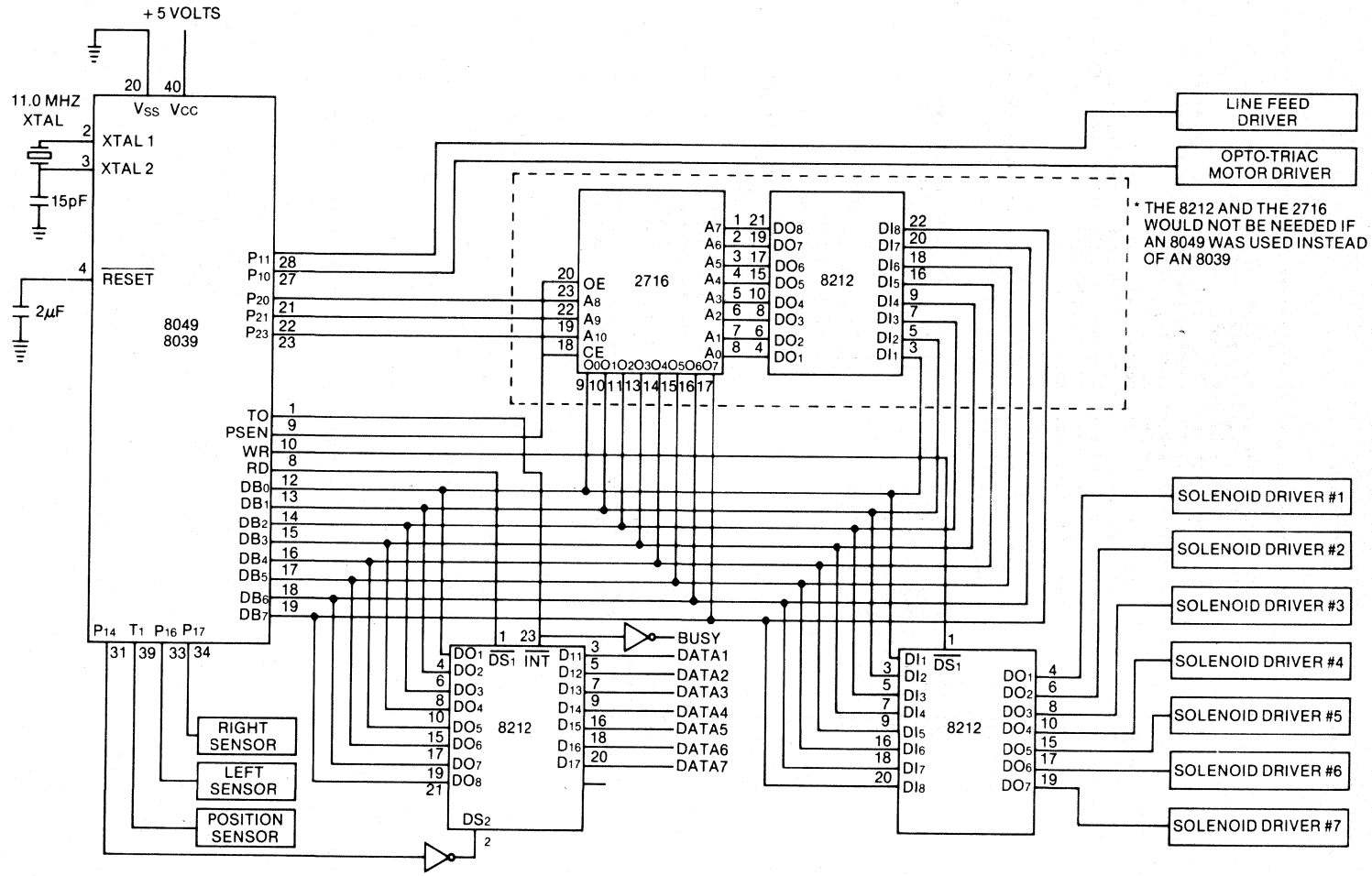
V. CONCLUSION

Although the full speed of the 8049 was not used in this application, the high speed of the 8049 makes it possible to "fine-tune" any critical timing parameters. Additionally, the extra available CPU time could be used to add an interrupt driven keyboard and display, such as the ones discussed in AP-40, to the printer. This would allow the printer to function as a complete "terminal".

Very little attempt was made to optimize the software, but still the entire program fits easily in 1.25K of memory; 750 bytes for printer control and 500 bytes for character lookup. Adding lower case to the printer would require an additional 500 bytes of lookup table. The remaining 250 bytes should be used to add "user" features such as tabs, double width printing, etc.

The high speed of the 8049 combined with its hardware and software architecture make it an ideal choice for controlling an 80 column, bi-directional line printer. The I/O structure of the 8049 minimizes the amount of external hardware needed to control the printer and the large amount of on-board program and data memory allow quite a sophisticated control program to be implemented.

APPENDIX A. SCHEMATIC DIAGRAM



APPENDIX B. MONITOR LISTING

```
LOC  OBJ          SEQ          SOURCE STATEMENT

                    1          ;
                    2          ;*****************************************************************
                    3          ;
                    4          ;THIS PROGRAM IMPLEMENTS CONTROL OF THE C.ITOH MODEL 820
                    5          ;PRINTER.  THE HARDWARE CONFIGURATION IS AS SUCH:
                    6          ;8212 INPUT PORT ON BUS = DATA INPUT
                    7          ;8212 OUTPUT PORT ON BUS = OUTPUT TO SOLENOID HAMMERS
                    8          ;T1 INPUT = CHARACTER POSITIONING SENSOR ON PRINTER
                    9          ;T0 INPUT = INTERRUPT FROM 8212 INPUT PORT
                   10          ;PORT 10 = MOTOR ON, LOW = ON
                   11          ;PORT 11 = LINE FEED STROBE, LOW = ON
                   12          ;PORT 16 = LEFT MARGIN SENSOR, LOW WHEN COVERED, HIGH WHEN OPEN
                   13          ;PORT 17 = RIGHT MARGIN SENSOR, LOW WHEN COVERED, HIGH WHEN OPEN
                   14          ;T1 = PIN 2 OF LM339, PRINT WHEEL SENSOR
                   15          ;PORT 16 = PIN 13 OF LM339
                   16          ;PORT 17 = PIN 14 OF LM339
                   17          ;
                   18          ;*****************************************************************
                   19          ;
                   20          ;SYSTEM EQUATES
                   21          ;
0000               22 INBUF    EQU     R0              ;POINTS AT INPUT LOCATION
0001               23 OUTBUF   EQU     R1              ;POINTS AT OUTPUT LOCATION
0002               24 SAVPNT   EQU     R2              ;STATUS FOR PRINTING
0003               25 STBCNT   EQU     R3              ;STROBE COUNTER
0004               26 TEMP1    EQU     R4
0005               27 STATUS   EQU     R5              ;BIT 0 = LINE FEED SET
                   28                                  ;BIT 1 = PRINT
                   29                                  ;BIT 2 = CONTINUE
                   30                                  ;BIT 3 = CR FOUND
                   31                                  ;BIT 4 = LF FOUND
                   32                                  ;BIT 5 = LF FOUND IN PRINTING
                   33                                  ;BIT 6 = PRINT DIRECTION
                   34                                  ;0 = RIGHT TO LEFT
                   35                                  ;1 = LEFT TO RIGHT
                   36                                  ;BIT 7 = BUFFER LOAD DIRECTION
                   37                                  ;0 = FIRST TO MAX
                   38                                  ;1 = MAX TO FIRST
0006               39 LINCNT   EQU     R6              ;THE LINE COUNTER
0007               40 JUNK1    EQU     R7
006F               41 MAX      EQU     6FH             ;MAX BUFFER LOCATION
0020               42 FIRST    EQU     20H             ;BOTTOM OF BUFFER
                   43 $EJECT
```

```
LOC  OBJ          SEQ          SOURCE STATEMENT

                    44          ;
0800                45          ORG     800H
                    46          ;
                    47          ;JUMP OVER THE INTERRUPT LOCATIONS
                    48          ;
0800 15             49          DIS     I                   ;DON'T USE INTERRUPTS
0801 3400           50          JMP     BGIN                ;BEGIN THE PROGRAM
                    51          ;
080A                52          ORG     8AH
                    53          ;
                    54          ;START THE PROGRAM
                    55          ;
                    56          ;LOOP UNTIL THE BUFFER FILLS UP
                    57          ;
080A FD             58 PRNT:    MOV     A,STATUS            ;GET THE STATUS
080B 3211           59          JB1     LPRNT               ;IF PRINTING, CONTINUE
080D 3408           60          CALL    LDBUF               ;READ INTO THE BUFFER
080F 840A           61          JMP     PRNT                ;LOOP
                    62          ;
                    63          ;THIS ROUTINE PRINTS A LINE
                    64          ;IT FIRST SAVES THE STATUS
                    65          ;AND THEN DETERMINES WHICH DIRECTION TO PRINT
                    66          ;AND HOW TO MANIPULATE THE BUFFER
                    67          ;
0811 34C9           68 LPRNT:   JMP     STACHK              ;GO FIX UP THE STATUS
0813 F224           69 LPRNT1:  JB7     CASE23              ;JUMP TO CASE 2 AND 3
0815 8417           70          JMP     CASE01              ;JUMP TO CASE 0 AND 1
                    71          ;
                    72          ;CASE01, LOADING THE BUFFER FROM FIRST TO MAX
                    73          ;
0817 B920           74 CASE01:  MOV     OUTBUF,#FIRST       ;SET UP OUTBUF
0819 B820           75          MOV     INBUF,#FIRST        ;SET UP INBUF
081B FA             76          MOV     A,SAVPNT            ;GET THE SAVED STATUS
081C 94DC           77          CALL    MOTON               ;TURN ON THE MOTOR
081E D252           78          JB6     CASE1               ;PRINT FOWARD
0820 94B3           79          CALL    PRNTBK              ;GET READY TO PRINT BACKWARDS
0822 8431           80          JMP     CASE0               ;PRINT BACKWARDS
                    81          ;
                    82          ;CASE23, LOADING BUFFER FROM MAX TO FIRST
                    83          ;
0824 B96F           84 CASE23:  MOV     OUTBUF,#MAX         ;SET UP OUTBUF
0826 B86F           85          MOV     INBUF,#MAX          ;SET UP INBUF
0828 FA             86          MOV     A,SAVPNT            ;GET THE PRINT STATUS
0829 94DC           87          CALL    MOTON               ;TURN ON THE MOTOR
082B D2C2           88          JB6     CASE3               ;PRINT LEFT TO RIGHT
082D 94B3           89          CALL    PRNTBK              ;GET READY TO PRINT BACKWARDS
082F 848D           90          JMP     CASE2               ;PRINT RIGHT TO LEFT
                    91          ;
                    92 $EJECT
```

```
LOC  OBJ          SEQ          SOURCE STATEMENT

0831 F1            93 CASE0:   MOV     A,@OUTBUF       ;GET THE CHARACTER
0832 3491          94          CALL    FXPRNT          ;ADJUST FOR PRINTING
0834 B128          95          MOV     @OUTBUF,#28H    ;PUT A SPACE IN BUFFER RAM
0836 F242          96          JB7     FDC             ;FOUND A CR
0838 945E          97          CALL    INCTST          ;UPDATE OUTBUF
083A C6AE          98          JZ      WATCHD          ;WAIT FOR END
083C BF28          99          MOV     JUNK1,#28H      ;GET A SPACE TO PRINT
083E 9463         100          CALL    GTPRNT          ;GO PRINT A SPACE
0840 8431         101          JMP     CASE0           ;LOOP
0842 BF28         102 FDC:     MOV     JUNK1,#28H      ;GO PRINT THE LAST SPACE
0844 9463         103 FDC1:    CALL    GTPRNT          ;GO PRINT A CHARACTER
0846 945E         104          CALL    INCTST          ;CHECK OUT BUFFER
0848 C6AE         105          JZ      WATCHD          ;WAIT FOR THE END
084A F1           106          MOV     A,@OUTBUF       ;GET THE CHARACTER
084B B128         107          MOV     @OUTBUF,#28H    ;PUT A SPACE THERE
084D 3491         108          CALL    FXPRNT          ;FIX THE CHARACTER UP
084F AF           109          MOV     JUNK1,A         ;SAVE IT
0850 8444         110          JMP     FDC1            ;LOOP
                  111          ;
                  112          ;
                  113          ;CASE 1, PRINTING LEFT TO RIGHT, LOADING BUFFER FROM
                  114          ;FIRST TO MAX
                  115          ;
0852 F1           116 CASE1:   MOV     A,@OUTBUF       ;GET THE CHARACTER
0853 3491         117          CALL    FXPRNT          ;ADJUST FOR PRINTING
0855 AF           118          MOV     JUNK1,A         ;SAVE ACC
0856 B128         119          MOV     @OUTBUF,#28H    ;PUT A SPACE IN THE BUFFER
0858 F262         120          JB7     CRFOND          ;FOUND A CR?
085A 9463         121          CALL    GTPRNT          ;GO PRINT THE CHARACTER
085C 945E         122          CALL    INCTST          ;CHECK THE BUFFER
085E C675         123          JZ      WATCH           ;IS THE LAST CHARACTER BEING PRINTED?
0860 8452         124          JMP     CASE1           ;LOOP
0862 B128         125 CRFOND:  MOV     @OUTBUF,#28H    ;PUT A SPACE IN THE BUFFER MEMORY
0864 BF28         126          MOV     JUNK1,#28H      ;PUT A SPACE IN TEMP LOCATION
0866 9463         127          CALL    GTPRNT          ;GO PRINT THE SPACE
0868 945E         128          CALL    INCTST          ;CHECK THE BUFFER
086A C675         129          JZ      WATCH           ;LAST CHARACTER PRINTED?
086C F1           130          MOV     A,@OUTBUF       ;GET THE NEXT CHARACTER
086D 3491         131          CALL    FXPRNT          ;ADJUST IT
086F 8462         132          JMP     CRFOND          ;LOOP
                  133 $EJECT
```

```
LOC  OBJ          SEQ          SOURCE STATEMENT

                  134          ;
                  135          ;THIS ROUTINE CALLS THE LINE FEED
                  136          ;
0071 947B         137 DOLF:    CALL    LINEFD          ;STROBE LINE FEED SOLENOID
0073 048A         138          JMP     PRNT            ;GO BACK TO THE PRINT ROUTINE
                  139          ;
                  140          ;THIS ROUTINE COMPLETES A LINE WHEN THE PRINT
                  141          ;HEAD IS MOVING LEFT TO RIGHT
                  142          ;
0075 27           143 WATCH:   CLR     A               ;ZERO ACC
0076 62           144          MOV     T,A             ;ZERO TIMER
0077 55           145          STRT    T               ;START THE TIMER
0078 3488         146          CALL    LDBUF           ;GO READ THE LAST CHARACTER
007A 09           147 LOOPW:   IN      A,P1            ;EXAMIN PORT ONE
007B F27A         148          JB7     LOOPW           ;CHECK RIGHT HAND SENSOR
007D 65           149          STOP    TCNT            ;STOP THE TIMER
007E FD           150          MOV     A,STATUS        ;GET THE STATUS
007F 5285         151          JB2     OVR1            ;JUMP IF CONTINUE IS SET
0081 94DF         152          CALL    MOTOF           ;TURN MOTOR OFF
0083 53FD         153          ANL     A,#0FDH         ;RESET BIT ONE
0085 53FB         154 OVR1:    ANL     A,#0FBH         ;RESET CONTINUE BIT
0087 AD           155          MOV     STATUS,A        ;RESTORE STATUS
0088 FA           156          MOV     A,SAVPNT        ;GET THE SAVED STATUS
0089 8271         157          JB3     DOLF            ;DO A LINE FEED IF BIT IS SET
008B 048A         158          JMP     PRNT            ;GO BACK TO PRINT ROUTINE
                  159          ;
                  160          ;
                  161          ;CASE 2, PRINTING RIGHT TO LEFT, LOADING BUFFER FROM
                  162          ;MAX TO FIRST
                  163          ;
                  164          ;
008D F1           165 CASE2:   MOV     A,@OUTBUF       ;GET THE CHARACTER
008E 3491         166          CALL    FXPRNT          ;ADJUST FOR PRINTING
0090 B128         167          MOV     @OUTBUF,#28H    ;PUT A SPACE IN BUFFER RAM
0092 F29E         168          JB7     FDCR            ;FIND A CR YET
0094 9472         169          CALL    DECTST          ;CHECK THE BUFFER
0096 C6AE         170          JZ      WATCHD          ;IF ZERO WAIT FOR SENSOR FLAG
0098 BF28         171          MOV     JUNK1,#28H      ;PUT SPACE IN TEMP LOCATION
009A 9463         172          CALL    GTPRNT          ;GO PRINT SPACE
009C 048D         173          JMP     CASE2           ;LOOP
009E BF28         174 FDCR:    MOV     JUNK1,#28H      ;GET A SPACE
00A0 9463         175 FDCR1:   CALL    GTPRNT          ;GO PRINT THE CHARACTER
00A2 9472         176          CALL    DECTST          ;CHECK THE BUFFER
00A4 C6AE         177          JZ      WATCHD          ;LEAVE IF DONE
00A6 F1           178          MOV     A,@OUTBUF       ;GET A CHARACTER
00A7 3491         179          CALL    FXPRNT          ;ADJUST THE CHARACTER FOR PRINTING
00A9 AF           180          MOV     JUNK1,A         ;SAVE IT
00AA B128         181          MOV     @OUTBUF,#28H    ;PUT A SPACE WHERE THE CHARACTER WAS
00AC 04A0         182          JMP     FDCR1           ;LOOP
                  183 $EJECT
```

```
LOC  OBJ          SEQ          SOURCE STATEMENT

                   184          ;
                   185          ;THIS ROUTINE WAITS FOR THE SENSOR FLAGS TO BE COVERED
                   186          ;WHEN PRINTING RIGHT TO LEFT
                   187          ;
0BAE 348B          188 WATCHD:  CALL    LDBUF           ;GO READ THE LAST CHARACTER
0BB0 09            189          IN      A,P1            ;GET SENSOR INFORMATION
0BB1 D2AE          190          JB6     WATCHD          ;LOOP IF SENSOR IS NOT COVERED
0BB3 FD            191          MOV     A,STATUS        ;GET THE STATUS
0BB4 52BA          192          JB2     OVR             ;SEE IF CONTINUE IS SET
0BB6 94DF          193          CALL    MOTOF           ;TURN THE MOTOR OFF
0BB8 53FD          194          ANL     A,#0FDH         ;RESET BIT 1
0BBA 53F8          195 OVR:     ANL     A,#0F8H         ;RESET BIT 3
0BBC AD            196          MOV     STATUS,A        ;RESTORE STATUS
0BBD FA            197          MOV     A,SAVPNT        ;GET THE SAVED STATUS
0BBE B271          198          JB5     DOLF            ;DO A LINE FEED
0BC0 B48A          199          JMP     PRNT            ;EXIT
                   200          ;
                   201          ;CASE 3, PRINTING LEFT TO RIGHT, LOADING BUFFER FROM
                   202          ;MAK TO FIRST
                   203          ;
0BC2 F1            204 CASE3:   MOV     A,@OUTBUF       ;GET A CHARACTER
0BC3 3491          205          CALL    FXPRNT          ;FIX FOR PRINTING
0BC5 AF            206          MOV     JUNK1,A         ;SAVE CHARACTER
0BC6 B128          207          MOV     @OUTBUF,#28H    ;PUT A SPACE IN THE BUFFER
0BC8 F2D2          208          JB7     CRFND           ;LEAVE IF A CR IS FOUND
0BCA 9463          209          CALL    GTPRNT          ;GO PRINT THE CHARACTER
0BCC 9472          210          CALL    DECTST          ;CHECK THE BUFFER
0BCE C675          211          JZ      WATCH           ;LEAVE IF DONE
0BD0 B4C2          212          JMP     CASE3           ;LOOP
0BD2 B128          213 CRFND:   MOV     @OUTBUF,#28H    ;PUT A SPACE IN THE BUFFER RAM
0BD4 BF28          214          MOV     JUNK1,#28H      ;GET A SPACE
0BD6 9463          215          CALL    GTPRNT          ;PRINT A SPACE
0BD8 9472          216          CALL    DECTST          ;CHECK THE BUFFER
0BDA C675          217          JZ      WATCH           ;LEAVE IF DONE
0BDC F1            218          MOV     A,@OUTBUF       ;GET NEXT CHARACTER
0BDD 3491          219          CALL    FXPRNT          ;ADJUST IT
0BDF B4D2          220          JMP     CRFND           ;LOOP
                   221 $EJECT
```

```
LOC  OBJ          SEQ          SOURCE STATEMENT

0100                222          ORG     100H
                    223          ;
0100 89             224 LDBUF:   IN      A,P1            ;READ PORT 1
0101 B21C           225          JB5     LNMODE          ;BIT 5 = H = LINE MODE
0103 1207           226          JB0     ARND            ;JUMP AROUND IF MOTOR IS ON
0105 8901           227          ORL     P1,#01H         ;TURN THE MOTOR OFF
0107 920F           228 ARND:    JB4     NOFF            ;NO FORM FEED
0109 FE             229          MOV     A,LINCNT        ;GET THE LINE COUNTER
010A 4380           230          ORL     A,#80H          ;SET MSB
010C AE             231          MOV     LINCNT,A        ;RESTORE THE LINE COUNTER
010D 23FF           232          MOV     A,#0FFH         ;SET ACC
010F 721A           233 NOFF:    JB3     NOLF            ;JUMP IF NO LINE FEED
0111 9478           234          CALL    LINEFD          ;GO DO A LF OR FF
0113 89             235 BUTLOP:  IN      A,P1            ;READ THE PORT
0114 721A           236          JB3     NOLF            ;WAIT FOR SWITCH TO BE RELEASED
0116 921A           237          JB4     NOLF            ;WAIT FOR SWITCH TO BE RELEASED
0118 2413           238          JMP     BUTLOP          ;LOOP
011A 2400           239 NOLF:    JMP     LDBUF           ;LOOP
                    240          ;
                    241          ;FIRST SEE IF A CHARACTER IS PRESENT IN THE BUFFER
                    242          ;
011C 261F           243 LNMODE:  JNTB    CHAR            ;IF CHARACTER PRESENT, READ IT
011E 83             244          RET                     ;IF NOT, EXIT ROUTINE
                    245          ;
                    246          ;IF THERE IS A CHARACTER, READ IT
                    247          ;
011F FD             248 CHAR:    MOV     A,STATUS        ;GET THE STATUS
0120 5249           249          JB2     ARNDJP          ;IF CONTINUE IS SET, DON'T LOAD
0122 9249           250          JB4     ARNDJP          ;IF LF IS SET, DON'T LOAD
0124 724A           251          JB3     LFCRCK          ;WAS CR SET, SEE IF NEXT CHAR IS LF
0126 94D6           252          CALL    GTCAR           ;GO READ A CHARACTER
0128 3461           253 GOOD:    CALL    FXCHAR          ;MAKE SURE IT IS OK
012A A8             254          MOV     @INBUF,A        ;SAVE CHARACTER IN BUFFER MEMORY
012B FD             255          MOV     A,STATUS        ;GET THE STATUS
012C F239           256          JB7     SUB1            ;IF BIT 7 IS SET DECREMENT BUFFER
012E 18             257          INC     INBUF           ;UPDATE INBUF
012F 2378           258          MOV     A,#MAX+1        ;GET TOP
0131 D8             259          XRL     A,INBUF         ;ARE WE AT THE TOP?
0132 9649           260          JNZ     ARNDJP          ;IF NOT GET THE STATUS
0134 F8             261          MOV     A,INBUF         ;GET INBUF
0135 07             262          DEC     A               ;CHANGE BY ONE
0136 A8             263          MOV     INBUF,A         ;PUT IT BACK
0137 2449           264          JMP     ARNDJP          ;GET THE STATUS
0139 F8             265 SUB1:    MOV     A,INBUF         ;GET INBUF
013A 07             266          DEC     A               ;CHANGE BY ONE
013B A8             267          MOV     INBUF,A         ;PUT INBUF BACK
013C 231F           268          MOV     A,#FIRST-1      ;GET THE BOTTOM OF THE BUFFER
013E D8             269          XRL     A,INBUF         ;TEST THE BUFFER
013F 9649           270          JNZ     ARNDJP          ;IF NOT ZERO READ THE STATUS
0141 18             271          INC     INBUF           ;MOVE INBUF BACK
0142 2449           272          JMP     ARNDJP          ;GO GET STATUS
0144 FD             273 GETSTA:  MOV     A,STATUS        ;GET THE STATUS
0145 1249           274          JB0     ARNDJP          ;IF BIT 0 SET, BYPASS
0147 9258           275          JB4     STBIT1          ;IF LF IS FOUND, SET THE STATUS
0149 83             276 ARNDJP:  RET                     ;EXIT
                    277          ;
                    278          ;THIS ROUTINE "FORCES" A LF AFTER A CR
                    279          ;
014A 94D6           280 LFCRCK:  CALL    GTCAR           ;READ A CHARACTER
014C 238A           281          MOV     A,#8AH          ;GET A LINE FEED
014E 2428           282          JMP     GOOD            ;JUMP BACK
                    283          ;
                    284          ;THIS ROUTINE SETS THE STATUS BITS
                    285          ;
0150 FD             286 STBIT1:  MOV     A,STATUS        ;LOAD THE STATUS
0151 3259           287          JB1     STPRNT          ;IF STILL PRINTING, LEAVE
0153 4382           288          ORL     A,#82H          ;SET PRINT BIT
0155 8348           289          ADD     A,#48H          ;UPDATE POSITION COUNTER
0157 AD             290          MOV     STATUS,A        ;PUT STATUS BACK
0158 83             291          RET                     ;EXIT ROUTINE
0159 5268           292 STPRNT:  JB2     BYEBYE          ;CHECK CONTINUE BIT
015B 4384           293          ORL     A,#84H          ;SET CONTINUE BIT
015D 8348           294          ADD     A,#48H          ;UPDATE PRINT DIRECTION
015F AD             295          MOV     STATUS,A        ;PUT THE STATUS BACK
0160 83             296 BYEBYE:  RET                     ;EXIT
                    297          ;
```

```
LOC  OBJ          SEQ          SOURCE STATEMENT

                   298          ;THIS ROUTINE "CONVERTS" LOWER CASE LETTERS TO
                   299          ;UPPER CASE
                   300          ;
0161 97            301 FXCHAR:  CLR     C                  ;CLEAR THE CARRY
0162 537F          302          ANL     A,#7FH             ;STRIP MSB
0164 AF            303          MOV     JUNK1,A            ;SAVE ACC
0165 03A8          304          ADD     A,#0A8H            ;SEE IF NUMBER IS 60H
0167 E670          305          JNC     FINE               ;IF CARRY ISN'T SET, JUMP
0169 FF            306          MOV     A,JUNK1            ;GET ACC BACK
016A 37            307          CPL     A                  ;SUBTRACT 20H FROM THE ACC
016B 0320          308          ADD     A,#20H
016D 37            309          CPL     A
016E 2474          310          JMP     FIXDUN             ;JUMP TO TEST CR LF
0170 37            311 FINE:    CPL     A                  ;NOW SUBTRACT A0H FROM ACC
0171 03A8          312          ADD     A,#0A8H
0173 37            313          CPL     A
0174 AF            314 FIXDUN:  MOV     JUNK1,A            ;SAVE A
0175 D30D          315          XRL     A,#0DH             ;IS CHARACTER A CR
0177 967F          316          JNZ     LFTEST             ;IF IT IS NOT TEST LF
0179 FD            317          MOV     A,STATUS           ;GET THE STATUS
017A 4308          318          ORL     A,#08H             ;SET BIT 3
017C AD            319          MOV     STATUS,A           ;RESTORE THE STATUS
017D 248F          320          JMP     FIXFIN             ;LEAVE
017F FF            321 LFTEST:  MOV     A,JUNK1            ;GET CHARACTER BACK
0180 D30A          322          XRL     A,#0AH             ;IS IT A LF
0182 C689          323          JZ      FIXUP              ;IF ITS NOT, WE ARE DONE
0184 FF            324          MOV     A,JUNK1            ;GET THE CHARACTER BACK
0185 D30C          325          XRL     A,#0CH             ;IS IT A FORM FEED
0187 968F          326          JNZ     FIXFIN             ;IF NOT FORM FEED, JUMP
0189 FD            327 FIXUP:   MOV     A,STATUS           ;GET THE STATUS
018A 4310          328          ORL     A,#10H             ;SET BIT 4
018C AD            329          MOV     STATUS,A           ;RETURN THE STATUS
018D 3458          330          CALL    STBIT1             ;SET THE STATUS
018F FF            331 FIXFIN:  MOV     A,JUNK1            ;GET THE CHARACTER
LOC  OBJ          SEQ          SOURCE STATEMENT

0190 83            332          RET                        ;EXIT FIXCHAR
                   333          ;
                   334          ;THIS ROUTINE RECOGNIZES A LF, FF, AND CR
                   335          ;DURING THE PRINT OPERATION
                   336          ;IT ALSO FORCES A SPACE IF A CHARACTER FOUND
                   337          ;IN THE BUFFER IS NOT IN THE LOOKUP TABLE
                   338          ;
0191 AF            339 FXPRNT:  MOV     JUNK1,A            ;SAVE ACC
0192 D30C          340          XRL     A,#0CH             ;FORM FEED
0194 C6B2          341          JZ      FFFIX              ;GO SET FORM FEED
0196 FF            342          MOV     A,JUNK1            ;RESTORE CHARACTER
0197 D30D          343          XRL     A,#0DH             ;SEE IF IT IS A CR
0199 C6A8          344          JZ      CRFIX              ;LEAVE IF IT IS
019B FF            345          MOV     A,JUNK1            ;GET ACC BACK
019C D30A          346          XRL     A,#0AH             ;SEE IF IT IS A LF
019E C6AB          347          JZ      LFFIX              ;LEAVE IF IT IS
01A0 FF            348          MOV     A,JUNK1            ;GET CHARACTER BACK
01A1 53E0          349          ANL     A,#0E0H            ;SEE IF IT IS A CHARACTER
01A3 96BD          350          JNZ     ISCHAR             ;IF IT IS JUMP
01A5 2320          351          MOV     A,#20H             ;PUT A SPACE IN ACC
01A7 83            352          RET                        ;EXIT
01A8 4380          353 CRFIX:   ORL     A,#80H             ;SET BIT 7
01AA 83            354          RET                        ;EXIT
01AB FD            355 LFFIX:   MOV     A,STATUS           ;GET THE STATUS
01AC 4320          356          ORL     A,#20H             ;SET LF BIT IN STATUS
01AE AD            357          MOV     STATUS,A           ;PUT THE STATUS BACK
01AF 2320          358          MOV     A,#20H             ;GET A SPACE
01B1 83            359          RET                        ;EXIT
01B2 FD            360 FFFIX:   MOV     A,STATUS           ;GET THE STATUS
01B3 4320          361          ORL     A,#20H             ;SET LINE FEED BIT
01B5 AD            362          MOV     STATUS,A           ;PUT THE STATUS BACK
01B6 FE            363          MOV     A,LINCNT           ;GET THE LINE COUNT
01B7 4380          364          ORL     A,#80H             ;SET BIT 7
01B9 AE            365          MOV     LINCNT,A           ;PUT LINE COUNT BACK
01BA 2320          366          MOV     A,#20H             ;GET A SPACE
01BC 83            367          RET                        ;EXIT
01BD FF            368 ISCHAR:  MOV     A,JUNK1            ;GET CHARACTER BACK
01BE 533F          369          ANL     A,#3FH             ;STRIP THE TWO MSB
01C0 83            370          RET                        ;EXIT
```

```
LOC  OBJ          SEQ          SOURCE STATEMENT

                  371          ;
                  372          ;THIS ROUTINE PRINTS THE CHARACTER IN THE ACC
                  373          ;
01C1 AC           374 PRNTIT:  MOV     TEMP1,A          ;SAVE CHARACTER
01C2 E7           375          RL      A                ;MULTIPLY BY TWO
01C3 E7           376          RL      A                ;MULTIPLY BY FOUR
01C4 6C           377          ADD     A,TEMP1          ;ADD ONCE TO MULTIPLY BY 5
                  378          ;
                  379          ;NOW SEE WHAT PART OF THE LOOKUP TABLE TO USE
                  380          ;
01C5 2C           381          XCH     A,TEMP1          ;PUT CHARACTER IN A, TARGET IN TEMP1
01C6 B2CA         382          JB5     SHORT            ;JUMP TO HIGH ADDRESS IF BIT 5 SET
01C8 44AB         383          JMP     PAGE1            ;GO TO FIRST PART OF LOOKUP TABLE
01CA 64AB         384 SHORT:   JMP     PAGE2            ;GO TO SECOND PAGE OF LOOKUP TABLE
                  385          ;
                  386          ;THIS ROUTINE TRIGGERS THE SOLENOIDS FOR 600 MICROSECONDS
                  387          ;AFTER WAITING FOR THE TRIGGER SIGNAL FROM THE PRINTER
                  388          ;*
01CC AF           389 FIRE:    MOV     JUNK1,A          ;SAVE THE ACC
01CD FD           390          MOV     A,STATUS         ;GET THE STATUS
01CE D2D4         391          JB6     NT1              ;SEE IF FORWARD OR BACKWARDS
01D0 56D0         392 FIREX:   JT1     FIREX            ;WAIT FOR T1
01D2 24D6         393          JMP     FIREY            ;LEAVE
01D4 46D4         394 NT1:     JNT1    NT1              ;LOOP
01D6 FF           395 FIREY:   MOV     A,JUNK1          ;GET ACC BACK
01D7 90           396          MOVX    @R0,A            ;TRIGGER THE SOLENOID
                  397          ;
                  398          ;NOW KILL 600 MICROSECONDS
                  399          ;
01D8 23F3         400          MOV     A,#0F3H          ;LOAD DELAY NUMBER
01DA 62           401          MOV     T,A              ;PUT IT IN TIMER
01DB 55           402          STRT    T                ;START THE TIMER
01DC 16E0         403 TSJTF:   JTF     KTDUN            ;LOOP ON TIMER FLAG
01DE 24DC         404          JMP     TSJTF            ;
01E0 27           405 KTDUN:   CLR     A                ;ZERO ACC
01E1 90           406          MOVX    @R0,A            ;TURN OFF SOLENOIDS
01E2 65           407          STOP    TCNT             ;STOP THE TIMER
01E3 83           408          RET                      ;EXIT FIRE ROUTINE
                  409 $EJECT
```

```
LOC  OBJ          SEQ          SOURCE STATEMENT

                  410          ;
                  411          ;*****************************************************************
                  412          ;
                  413          ;THIS IS THE LOOKUP TABLE. THE MSB IS NOT USED, THE MSB - 1
                  414          ;IS THE DOT THAT IS THE TOP OF ANY GIVEN CHARACTER AND THE
                  415          ;LSB IS THE DOT THAT IS THE BOTTOM OF ANY GIVEN CHARACTER
                  416          ;
                  417          ;*****************************************************************
                  418          ;
0200              419          ORG     200H
                  420          ;*
0200 3E           421 TABLE1:  DB      3EH            ;  *****
0201 41           422          DB      41H            ; *     *
0202 5D           423          DB      5DH            ; * *** *
0203 59           424          DB      59H            ; * **  *
0204 4E           425          DB      4EH            ; *  ***
                  426
0205 7C           427          DB      7CH            ; *****
0206 12           428          DB      12H            ;    *  *
0207 11           429          DB      11H            ;    *   *
0208 12           430          DB      12H            ;    *  *
0209 7C           431          DB      7CH            ; *****
                  432
020A 7F           433          DB      7FH            ; *******
020B 49           434          DB      49H            ; *  *  *
020C 49           435          DB      49H            ; *  *  *
020D 49           436          DB      49H            ; *  *  *
020E 36           437          DB      36H            ;  ** **
                  438
020F 3E           439          DB      3EH            ;  *****
0210 41           440          DB      41H            ; *     *
0211 41           441          DB      41H            ; *     *
0212 41           442          DB      41H            ; *     *
0213 22           443          DB      22H            ;  *   *
                  444
0214 7F           445          DB      7FH            ; *******
0215 41           446          DB      41H            ; *     *
0216 41           447          DB      41H            ; *     *
0217 41           448          DB      41H            ; *     *
0218 3E           449          DB      3EH            ;  *****
                  450
0219 7F           451          DB      7FH            ; *******
021A 49           452          DB      49H            ; *  *  *
021B 49           453          DB      49H            ; *  *  *
021C 49           454          DB      49H            ; *  *  *
021D 41           455          DB      41H            ; *     *
                  456 $EJECT
```

```
LOC  OBJ          SEQ          SOURCE STATEMENT

                   457
021E 7F            458          DB       7FH             ; *******
021F 09            459          DB       09H             ;    *  *
0220 09            460          DB       09H             ;    *  *
0221 09            461          DB       09H             ;    *  *
0222 01            462          DB       01H             ;       *
                   463
0223 3E            464          DB       3EH             ;  *****
0224 41            465          DB       41H             ; *     *
0225 41            466          DB       41H             ; *     *
0226 51            467          DB       51H             ; * *   *
0227 71            468          DB       71H             ; ***   *
                   469
0228 7F            470          DB       7FH             ; *******
0229 08            471          DB       08H             ;    *
022A 08            472          DB       08H             ;    *
022B 08            473          DB       08H             ;    *
022C 7F            474          DB       7FH             ; *******
                   475
022D 00            476          DB       00H             ;
022E 41            477          DB       41H             ; *     *
022F 7F            478          DB       7FH             ; *******
0230 41            479          DB       41H             ; *     *
0231 00            480          DB       00H             ;
                   481
0232 20            482          DB       20H             ;  *
0233 40            483          DB       40H             ; *
0234 40            484          DB       40H             ; *
0235 40            485          DB       40H             ; *
0236 3F            486          DB       3FH             ;  ******
                   487
0237 7F            488          DB       7FH             ; *******
0238 08            489          DB       08H             ;    *
0239 14            490          DB       14H             ;   * *
023A 22            491          DB       22H             ;  *   *
023B 41            492          DB       41H             ; *     *
                   493
023C 7F            494          DB       7FH             ; *******
023D 40            495          DB       40H             ; *
023E 40            496          DB       40H             ; *
023F 40            497          DB       40H             ; *
0240 40            498          DB       40H             ; *
                   499
0241 7F            500          DB       7FH             ; *******
0242 02            501          DB       02H             ;      *
0243 0C            502          DB       0CH             ;    **
0244 02            503          DB       02H             ;      *
0245 7F            504          DB       7FH             ; *******
                   505
0246 7F            506          DB       7FH             ; *******
0247 04            507          DB       04H             ;     *
0248 08            508          DB       08H             ;    *
0249 10            509          DB       10H             ;   *
024A 7F            510          DB       7FH             ; *******
                   511 $EJECT
```

```
LOC  OBJ          SEQ          SOURCE STATEMENT

                   512
0248 3E            513          DB      3EH          ;  *****
024C 41            514          DB      41H          ; *     *
024D 41            515          DB      41H          ; *     *
024E 41            516          DB      41H          ; *     *
024F 3E            517          DB      3EH          ;  *****
                   518
0250 7F            519          DB      7FH          ; *******
0251 09            520          DB      09H          ;    *  *
0252 09            521          DB      09H          ;    *  *
0253 09            522          DB      09H          ;    *  *
0254 06            523          DB      06H          ;     **
                   524
0255 3E            525          DB      3EH          ;  *****
0256 41            526          DB      41H          ; *     *
0257 51            527          DB      51H          ; * *   *
0258 21            528          DB      21H          ;  *    *
0259 5E            529          DB      5EH          ; * ****
                   530
025A 7F            531          DB      7FH          ; *******
025B 09            532          DB      09H          ;    *  *
025C 19            533          DB      19H          ;   **  *
025D 29            534          DB      29H          ;  * * *
025E 46            535          DB      46H          ; *   **
                   536
025F 26            537          DB      26H          ;  *  **
0260 49            538          DB      49H          ; *  *  *
0261 49            539          DB      49H          ; *  *  *
0262 49            540          DB      49H          ; *  *  *
0263 32            541          DB      32H          ;  **  *
                   542
0264 01            543          DB      01H          ;       *
0265 01            544          DB      01H          ;       *
0266 7F            545          DB      7FH          ; *******
0267 01            546          DB      01H          ;       *
0268 01            547          DB      01H          ;       *
                   548
0269 3F            549          DB      3FH          ;  ******
026A 40            550          DB      40H          ; *
026B 40            551          DB      40H          ; *
026C 40            552          DB      40H          ; *
026D 3F            553          DB      3FH          ;  ******
                   554
026E 1F            555          DB      1FH          ;   *****
026F 20            556          DB      20H          ;  *
0270 40            557          DB      40H          ; *
0271 20            558          DB      20H          ;  *
0272 1F            559          DB      1FH          ;   *****
                   560
0273 7F            561          DB      7FH          ; *******
0274 20            562          DB      20H          ;  *
0275 18            563          DB      18H          ;   **
0276 20            564          DB      20H          ;  *
0277 7F            565          DB      7FH          ; *******
                   566 $EJECT
```

```
LOC  OBJ          SEQ          SOURCE STATEMENT

                   567
0278 63            568          DB      63H          ; **    **
0279 14            569          DB      14H          ;   * *
027A 08            570          DB      08H          ;     *
027B 14            571          DB      14H          ;   * *
027C 63            572          DB      63H          ; **    **
                   573
027D 03            574          DB      03H          ;       **
027E 04            575          DB      04H          ;     *
027F 78            576          DB      78H          ; ****
0280 04            577          DB      04H          ;     *
0281 03            578          DB      03H          ;       **
                   579
0282 61            580          DB      61H          ; **    *
0283 51            581          DB      51H          ; * *   *
0284 49            582          DB      49H          ; *  *  *
0285 45            583          DB      45H          ; *   * *
0286 43            584          DB      43H          ; *    **
                   585
0287 7F            586          DB      7FH          ; *******
0288 7F            587          DB      7FH          ; *******
0289 41            588          DB      41H          ; *     *
028A 41            589          DB      41H          ; *     *
028B 41            590          DB      41H          ; *     *
                   591
028C 02            592          DB      02H          ;      *
028D 04            593          DB      04H          ;     *
028E 08            594          DB      08H          ;    *
028F 10            595          DB      10H          ;   *
0290 20            596          DB      20H          ;  *
                   597
0291 41            598          DB      41H          ; *     *
0292 41            599          DB      41H          ; *     *
0293 41            600          DB      41H          ; *     *
0294 7F            601          DB      7FH          ; *******
0295 7F            602          DB      7FH          ; *******
                   603
0296 10            604          DB      10H          ;   *
0297 08            605          DB      08H          ;    *
0298 04            606          DB      04H          ;     *
0299 08            607          DB      08H          ;    *
029A 10            608          DB      10H          ;   *
                   609
029B 40            610          DB      40H          ; *
029C 40            611          DB      40H          ; *
029D 40            612          DB      40H          ; *
029E 40            613          DB      40H          ; *
029F 40            614          DB      40H          ; *
                   615 $EJECT
```

```
LOC  OBJ          SEQ          SOURCE STATEMENT

                  616          ;
02A8 BBBB         617 PAGE1:   MOV     STBCNT,#00H      ;ZERO STROBE COUNTER
02A2 FA           618          MOV     A,SAVPNT         ;GET DIRECTION
02A3 37           619          CPL     A                ;FLIP BITS
02A4 D2B3         620          JB6     BAKWRD           ;IF BACKWARD JUMP OUT
02A6 FC           621 LKL0:    MOV     A,TEMP1          ;GET THE TARGET
02A7 A3           622          MOVP    A,@A             ;GET THE DATA
02A8 34CC         623          CALL    FIRE             ;STROBE THE SOLENOIDS
02AA 1C           624          INC     TEMP1            ;INCREMENT THE POINTER
02AB 1B           625          INC     STBCNT           ;INCREMENT THE STROBE COUNTER
02AC FB           626          MOV     A,STBCNT         ;GET THE STROBE COUNTER
02AD D3B5         627          XRL     A,#B5H           ;IS IT FIVE
02AF 96A6         628          JNZ     LKL0             ;REPEAT IF NOT FIVE
02B1 84AE         629          JMP     SETTIM           ;GO BACK
02B3 FC           630 BAKWRD:  MOV     A,TEMP1          ;GET THE TARGET
02B4 03B4         631          ADD     A,#B4H           ;COMPENSATE FOR GOING BACKWARDS
02B6 AC           632          MOV     TEMP1,A          ;SAVE IT
02B7 FC           633 LKL01:   MOV     A,TEMP1          ;GET THE TARGET
02B8 A3           634          MOVP    A,@A             ;GET THE DATA
02B9 34CC         635          CALL    FIRE             ;STROBE THE SOLENOIDS
02BB FC           636          MOV     A,TEMP1          ;GET TEMP1
02BC B7           637          DEC     A                ;DECREASE BY ONE
02BD AC           638          MOV     TEMP1,A          ;PUT IT BACK
02BE 1B           639          INC     STBCNT           ;INCREMENT THE STROBE COUNTER
02BF FB           640          MOV     A,STBCNT         ;GET THE STROBE COUNTER
02C0 D3B5         641          XRL     A,#B5H           ;IS IT FIVE
02C2 96B7         642          JNZ     LKL01            ;REPEAT IF NOT FIVE
02C4 84AE         643          JMP     SETTIM           ;GO BACK, CHARACTER IS DONE
                  644 $EJECT
```

```
LOC  OBJ          SEQ          SOURCE STATEMENT

                   645          ;*
0300               646          ORG     300H
                   647          ;*
                   648
0300 00            649          DB      00H             ;
0301 00            650          DB      00H             ;
0302 00            651          DB      00H             ;
0303 00            652          DB      00H             ;
0304 00            653          DB      00H             ;
                   654
0305 00            655          DB      00H             ;
0306 00            656          DB      00H             ;
0307 5F            657          DB      5FH             ;  * *****
0308 00            658          DB      00H             ;
0309 00            659          DB      00H             ;
                   660
030A 00            661          DB      00H             ;
030B 07            662          DB      07H             ;       ***
030C 00            663          DB      00H             ;
030D 07            664          DB      07H             ;       ***
030E 00            665          DB      00H             ;
                   666
030F 14            667          DB      14H             ;    * *
0310 7F            668          DB      7FH             ;  *******
0311 14            669          DB      14H             ;    * *
0312 7F            670          DB      7FH             ;  *******
0313 14            671          DB      14H             ;    * *
                   672
0314 24            673          DB      24H             ;   *   *
0315 2A            674          DB      2AH             ;   * * *
0316 7F            675          DB      7FH             ;  *******
0317 2A            676          DB      2AH             ;   * * *
0318 12            677          DB      12H             ;    *  *
                   678
0319 23            679          DB      23H             ;   *   **
031A 13            680          DB      13H             ;    *  **
031B 08            681          DB      08H             ;     *
031C 64            682          DB      64H             ;  **  *
031D 62            683          DB      62H             ;  **   *
                   684
031E 36            685          DB      36H             ;   ** **
031F 49            686          DB      49H             ;  *  *  *
0320 56            687          DB      56H             ;  * * **
0321 20            688          DB      20H             ;   *
0322 5D            689          DB      5DH             ;  * * *
                   690 $EJECT
```

```
LOC  OBJ          SEQ          SOURCE STATEMENT

                   691
0323 00            692          DB       00H             ;
0324 00            693          DB       00H             ;
0325 07            694          DB       07H             ;     ***
0326 00            695          DB       00H             ;
0327 00            696          DB       00H             ;
                   697
0328 1C            698          DB       1CH             ;    ***
0329 22            699          DB       22H             ;   *   *
032A 41            700          DB       41H             ; *     *
032B 00            701          DB       00H             ;
032C 00            702          DB       00H             ;
                   703
032D 00            704          DB       00H             ;
032E 00            705          DB       00H             ;
032F 41            706          DB       41H             ; *     *
0330 22            707          DB       22H             ;   *   *
0331 1C            708          DB       1CH             ;    ***
                   709
0332 22            710          DB       22H             ;   *   *
0333 14            711          DB       14H             ;    * *
0334 7F            712          DB       7FH             ; *******
0335 14            713          DB       14H             ;    * *
0336 22            714          DB       22H             ;   *   *
                   715
0337 00            716          DB       00H             ;     *
0338 00            717          DB       00H             ;     *
0339 7F            718          DB       7FH             ; *******
033A 00            719          DB       00H             ;     *
033B 00            720          DB       00H             ;     *
                   721
033C 00            722          DB       00H             ;
033D 40            723          DB       40H             ; *
033E 30            724          DB       30H             ;  **
033F 00            725          DB       00H             ;
0340 00            726          DB       00H             ;
                   727
0341 00            728          DB       00H             ;     *
0342 00            729          DB       00H             ;     *
0343 00            730          DB       00H             ;     *
0344 00            731          DB       00H             ;     *
0345 00            732          DB       00H             ;     *
                   733
0346 00            734          DB       00H             ;
0347 00            735          DB       00H             ;
0348 40            736          DB       40H             ; *
0349 00            737          DB       00H             ;
034A 00            738          DB       00H             ;
                   739
034B 20            740          DB       20H             ;   *
034C 10            741          DB       10H             ;    *
034D 08            742          DB       08H             ;     *
034E 04            743          DB       04H             ;      *
034F 02            744          DB       02H             ;       *
                   745
0350 3E            746          DB       3EH             ;  *****
0351 51            747          DB       51H             ; *  *   *
0352 49            748          DB       49H             ; *  *  *
0353 45            749          DB       45H             ; *   * *
0354 3E            750          DB       3EH             ;  *****
                   751
0355 00            752          DB       00H             ;
0356 42            753          DB       42H             ; *    *
0357 7F            754          DB       7FH             ; *******
0358 40            755          DB       40H             ; *
0359 00            756          DB       00H             ;
                   757
035A 62            758          DB       62H             ; **   *
035B 51            759          DB       51H             ; *  *   *
035C 49            760          DB       49H             ; *  *  *
035D 49            761          DB       49H             ; *  *  *
035E 46            762          DB       46H             ; *   **
                   763
035F 21            764          DB       21H             ;  *    *
0360 41            765          DB       41H             ; *     *
```

```
LOC  OBJ          SEQ          SOURCE STATEMENT

0361 49           766          DB      49H           ;  *  *  *
0362 4D           767          DB      4DH           ;  *  ** *
0363 33           768          DB      33H           ;  **  **
                  769
0364 18           770          DB      18H           ;    **
0365 14           771          DB      14H           ;   * *
0366 12           772          DB      12H           ;  *  *
0367 7F           773          DB      7FH           ; *******
0368 10           774          DB      10H           ;   *
                  775
0369 27           776          DB      27H           ;  *  ***
036A 45           777          DB      45H           ; *   * *
036B 45           778          DB      45H           ; *   * *
036C 45           779          DB      45H           ; *   * *
036D 39           780          DB      39H           ;  ***  *
                  781
036E 3C           782          DB      3CH           ;  ****
036F 4A           783          DB      4AH           ;  *  * *
0370 49           784          DB      49H           ;  *  *  *
0371 49           785          DB      49H           ;  *  *  *
0372 31           786          DB      31H           ;  **   *
                  787
0373 01           788          DB      01H           ;       *
0374 71           789          DB      71H           ;  ***   *
0375 09           790          DB      09H           ;     *  *
0376 05           791          DB      05H           ;      * *
0377 03           792          DB      03H           ;       **
                  793
0378 36           794          DB      36H           ;  ** **
0379 49           795          DB      49H           ;  *  *  *
037A 49           796          DB      49H           ;  *  *  *
037B 49           797          DB      49H           ;  *  *  *
037C 36           798          DB      36H           ;  ** **
                  799 $EJECT
```

```
LOC  OBJ          SEQ          SOURCE STATEMENT

                  800
037D 46           801          DB        46H          ;  *    **
037E 49           802          DB        49H          ;  *  *   *
037F 49           803          DB        49H          ;  *  *   *
0380 29           804          DB        29H          ;   * *   *
0381 1E           805          DB        1EH          ;    ****
                  806
0382 00           807          DB        00H          ;
0383 00           808          DB        00H          ;
0384 14           809          DB        14H          ;    * *
0385 00           810          DB        00H          ;
0386 00           811          DB        00H          ;
                  812
0387 00           813          DB        00H          ;
0388 40           814          DB        40H          ;  *
0389 34           815          DB        34H          ;   ** *
038A 00           816          DB        00H          ;
038B 00           817          DB        00H          ;
                  818
038C 00           819          DB        00H          ;     *
038D 14           820          DB        14H          ;    * *
038E 22           821          DB        22H          ;   *   *
038F 41           822          DB        41H          ;  *     *
0390 00           823          DB        00H          ;
                  824
0391 14           825          DB        14H          ;    * *
0392 14           826          DB        14H          ;    * *
0393 14           827          DB        14H          ;    * *
0394 14           828          DB        14H          ;    * *
0395 14           829          DB        14H          ;    * *
                  830
0396 00           831          DB        00H          ;
0397 41           832          DB        41H          ;  *     *
0398 22           833          DB        22H          ;   *   *
0399 14           834          DB        14H          ;    * *
039A 00           835          DB        00H          ;     *
                  836
039B 02           837          DB        02H          ;       *
039C 01           838          DB        01H          ;        *
039D 59           839          DB        59H          ;  * **  *
039E 05           840          DB        05H          ;      * *
039F 02           841          DB        02H          ;       *
                  842 $EJECT
```

```
LOC  OBJ          SEQ          SOURCE STATEMENT

03A8 B8B8          843 PAGE2:  MOV     STBCNT,#B8H     ;ZERO STROBE COUNTER
03A2 FA            844         MOV     A,SAVPNT        ;GET DIRECTION
03A3 37            845         CPL     A               ;FLIP BITS
03A4 D2B5          846         JB6     BKWRD           ;IF BACKWARD JUMP OUT
03A6 FC            847 LKHI:   MOV     A,TEMP1         ;GET THE TARGET
03A7 B360          848         ADD     A,#60H          ;ADJUST THE TARGET
03A9 A3            849         MOVP    A,@A            ;GET THE DATA
03AA 34CC          850         CALL    FIRE            ;STROBE THE SOLENOIDS
03AC 1C            851         INC     TEMP1           ;INCREMENT THE POINTER
03AD 1B            852         INC     STBCNT          ;INCREMENT THE STROBE COUNTER
03AE FB            853         MOV     A,STBCNT        ;GET THE STROBE COUNTER
03AF D3B5          854         XRL     A,#B5H          ;IS IT FIVE
03B1 96A6          855         JNZ     LKHI            ;REPEAT IF NOT FIVE
03B3 84AE          856         JMP     SETTIM          ;GO BACK
03B5 FC            857 BKWRD:  MOV     A,TEMP1         ;GET THE TARGET
03B6 B364          858         ADD     A,#64H          ;COMPENSATE FOR GOING BACKWARDS
03B8 AC            859         MOV     TEMP1,A         ;SAVE IT
03B9 FC            860 LKHI1:  MOV     A,TEMP1         ;GET THE TARGET
03BA A3            861         MOVP    A,@A            ;GET THE DATA
03BB 34CC          862         CALL    FIRE            ;STROBE THE SOLENOIDS
03BD FC            863         MOV     A,TEMP1         ;GET TEMP1
03BE B7            864         DEC     A               ;DECREASE BY ONE
03BF AC            865         MOV     TEMP1,A         ;PUT IT BACK
03C0 1B            866         INC     STBCNT          ;INCREMENT THE STROBE COUNTER
03C1 FB            867         MOV     A,STBCNT        ;GET THE STROBE COUNTER
03C2 D3B5          868         XRL     A,#B5H          ;IS IT FIVE
03C4 96B9          869         JNZ     LKHI1           ;REPEAT IF NOT FIVE
03C6 84AE          870         JMP     SETTIM          ;GO BACK, CHARACTER IS DONE
                   871 $EJECT
```

```
LOC  OBJ          SEQ          SOURCE STATEMENT

                   872          ;
0400               873          ORG     400H
                   874          ;
0400 27            875 BGIN:    CLR     A                ;ZERO ACC
0401 98            876          MOVX    @RB,A            ;TURN OFF THE SOLENOIDS
0402 9408          877          CALL    SETUP            ;SET UP THE PRINTER
0404 943F          878          CALL    VARSET           ;SET UP THE SOFTWARE
0406 8408          879          JMP     PRNT             ;GO START
                   880          ;
0408 23FE          881 SETUP:   MOV     A,#0FEH          ;LOAD ACC WITH VALUE TO TURN ON MOTOR
040A 39            882          OUTL    P1,A             ;TURN ON MOTOR
                   883          ;
                   884          ;NOW DELAY 3.2 SECONDS WHILE CHECKING RIGHT SENSOR
                   885          ;
040B BC05          886          MOV     TEMP1,#05H       ;LOAD DELAY VALUE ONE
040D BFFF          887 SELFC:   MOV     JUNK1,#0FFH      ;LOAD DELAY VALUE TWO
040F BEFF          888 SELFB:   MOV     LINCNT,#0FFH     ;LOAD DELAY VALUE THREE
0411 89            889 SELFA:   IN      A,P1             ;READ PORT ONE
0412 37            890          CPL     A                ;MAKE THINGS RIGHT
0413 F21D          891          JB7     DONER            ;IS BIT 7 SET?
0415 EE11          892          DJNZ    LINCNT,SELFA     ;SMALL LOOP
0417 EF0F          893          DJNZ    JUNK1,SELFB      ;BIGGER LOOP
0419 EC0D          894          DJNZ    TEMP1,SELFC      ;BIGGEST LOOP
041B 845A          895          JMP     ERROR            ;SOMETHING IS WRONG
                   896          ;
                   897          ;NOW MAKE SURE THE RIGHT SENSOR IS CLEARED
                   898          ;
041D BFFF          899 DONER:   MOV     JUNK1,#0FFH      ;SET UP DELAY
041F BEFF          900 SELF:    MOV     LINCNT,#0FFH     ;SOME MORE DELAY
0421 89            901 SELF1:   IN      A,P1             ;GET THE FLAG INFORMATION
0422 F22A          902          JB7     DONEF            ;IS FLAG CLEARED?
0424 EE21          903          DJNZ    LINCNT,SELF1     ;IF NOT LOOP
0426 EF1F          904          DJNZ    JUNK1,SELF       ;LOOP SOME MORE
0428 845A          905          JMP     ERROR            ;LEAVE IF FLAG IS NOT UNCOVERED
                   906          ;
                   907          ;NOW CHECK THE LEFT SENSOR IN THE SAME MANNER AS THE
                   908          ;RIGHT SENSOR, EXCEPT DELAY ONLY 2.5 SECONDS
                   909          ;
042A BC04          910 DONEF:   MOV     TEMP1,#04H       ;LOAD DELAY 1
042C BFFF          911 SELFCC:  MOV     JUNK1,#0FFH      ;LOAD DELAY 2
042E BEFF          912 SELFBB:  MOV     LINCNT,#0FFH     ;LOAD DELAY 3
0430 89            913 SELFAA:  IN      A,P1             ;READ THE PORT
0431 37            914          CPL     A                ;CHANGE THINGS AROUND
0432 D23C          915          JB6     DONEL            ;OK IF BIT 6 IS A ZERO
0434 EE30          916          DJNZ    LINCNT,SELFAA    ;SMALL LOOP
0436 EF2E          917          DJNZ    JUNK1,SELFBB     ;BIGGER LOOP
0438 EC2C          918          DJNZ    TEMP1,SELFCC     ;BIGGEST LOOP
043A 845A          919          JMP     ERROR            ;SOMETHING IS WRONG
043C 8981          920 DONEL:   ORL     P1,#81H          ;TURN MOTOR OFF
043E 83            921          RET                      ;GO BACK
                   922          ;
                   923          ;NOW SET UP THE VARIABLES
                   924          ;
043F 23FE          925 VARSET:  MOV     A,#0FEH          ;LOAD THE TIMER
0441 62            926          MOV     T,A
0442 55            927          STRT    T                ;START THE TIMER
0443 B820          928          MOV     INBUF,#FIRST     ;LOAD INPUT BUFFER
0445 BE08          929          MOV     LINCNT,#08H      ;SET LINE COUNT
0447 BD08          930          MOV     STATUS,#08H      ;SET FORWARD BIT
                   931          ;
                   932          ;NOW CLEAR THE RAM AREA BY WRITING SPACE CODES
                   933          ;
0449 B920          934          MOV     OUTBUF,#FIRST    ;LOAD OUTBUF
044B 2320          935 CLRMEM:  MOV     A,#20H           ;PUT SPACE CODE IN ACC
044D A1            936          MOV     @OUTBUF,A        ;PUT SPACE CODE IN DATA MEMORY
044E 19            937          INC     OUTBUF           ;UPDATE THE POINTER
044F F9            938          MOV     A,OUTBUF         ;MOVE THE POINTER INTA ACC
0450 D378          939          XRL     A,#MAX+1         ;SEE IF DONE
0452 9648          940          JNZ     CLRMEM           ;LOOP IF NOT CLEARED
                   941          ;
                   942          ;NOW CLEAR THE 8212
                   943          ;
0454 99EF          944          ANL     P1,#0EFH         ;SET ENABLE BIT
0456 80            945          MOVX    A,@INBUF         ;CLEAR THE 8212 INPUT BUFFER
0457 8910          946          ORL     P1,#10H          ;RESET ENABLE BIT
                   947          ;
```

```
LOC  OBJ          SEQ          SOURCE STATEMENT
                   948          ;NOW EXIT VARSET
                   949          ;
0459 83            950          RET                         ;LEAVE INITIALIZATION
                   951          ;
                   952          ;THIS ROUTINE TURNS THE MOTOR OFF AND LOOPS
                   953          ;
045A 89FF          954 ERROR:   ORL     P1,#0FFH            ;TURN OFF MOTOR
045C 045C          955 DEAD:    JMP     DEAD                ;LOOP BECAUSE SOMETHING IS WRONG
                   956          ;
                   957          ;THESE ARE ALL SUBROUTINES THAT ARE CALLED
                   958          ;
045E 19            959 INCTST:  INC     OUTBUF              ;UPDATE THE POINTER
045F 237B          960          MOV     A,#MAX+1            ;GET THE VALUE FOR THE LAST CHARACTER
0461 D9            961          XRL     A,OUTBUF            ;DO THE TEST
0462 83            962          RET                         ;EXIT
0463 B9            963 GTPRNT:  IN      A,P1                ;READ PORT ONE
0464 37            964          CPL     A                   ;FLIP BITS
0465 D263          965          JB6     GTPRNT              ;LOOP UNTIL SENSOR IS UNCOVERED
0467 166B          966 TSTJTF:  JTF     PIT                 ;SEE IF TIMER FLAG IS SET
0469 0467          967          JMP     TSTJTF              ;TEST FLAG
046B 65            968 PIT:     STOP    TCNT                ;STOP THE TIMER
046C FF            969          MOV     A,JUNK1             ;GET THE CHARACTER
046D 34C1          970          CALL    PRNTIT              ;PRINT THE CHARACTER
046F 341C          971          CALL    LNMODE              ;GET ANOTHER CHARACTER
0471 83            972          RET                         ;EXIT
0472 F9            973 DECTST:  MOV     A,OUTBUF            ;GET OUTBUF
0473 07            974          DEC     A                   ;REDUCE BY ONE
0474 A9            975          MOV     OUTBUF,A            ;PUT BACK IN OUTBUF
0475 D31F          976          XRL     A,#FIRST-1          ;SEE IF IT IS ALL THE WAY DOWN
0477 83            977          RET                         ;EXIT
                   978          ;
                   979          ;THIS ROUTINE DOES A LINE FEED
                   980          ;
0478 FE            981 LINEFD:  MOV     A,LINCNT            ;GET THE LINE COUNT
0479 F298          982          JB7     DOFF                ;IF BIT 7 IS SET, DO A FORMFEED
047B 99FD          983 LFDO:    ANL     P1,#0FDH            ;TURN ON THE SOLENOID
047D BC4D          984          MOV     TEMP1,#4DH          ;LOAD ONE DELAY
047F BF93          985 LFLP1:   MOV     JUNK1,#93H          ;LOAD ANOTHER DELAY
0481 EF81          986 LFLP2:   DJNZ    JUNK1,LFLP2         ;LOOP
0483 EC7F          987          DJNZ    TEMP1,LFLP1         ;LOOP SOME MORE
0485 8982          988          ORL     P1,#02H             ;TURN OFF LF SOLENOID
0487 1E            989          INC     LINCNT              ;UPDATE THE LINE COUNTER
0488 FE            990          MOV     A,LINCNT            ;GET THE LINE COUNT
0489 D328          991          XRL     A,#28H              ;IS PAGE DONE
048B 968F          992          JNZ     NOTDON              ;SKIP OVER
048D BE00          993          MOV     LINCNT,#00H         ;ZERO LINE COUNTER
                   994          ;
                   995          ;NOW DELAY 90 MILLISECONDS
                   996          ;
048F BC8B          997 NOTDON:  MOV     TEMP1,#8BH          ;LOAD DELAY VALUES
0491 BFFF          998 LOP1:    MOV     JUNK1,#0FFH         ;
0493 EF93          999 LOP2:    DJNZ    JUNK1,LOP2          ;GENERATE DELAY
0495 EC91         1000          DJNZ    TEMP1,LOP1          ;
0497 83           1001          RET                         ;LINE FEED IS DONE
                  1002          ;
                  1003          ;THIS ROUTINE DOES A FORM FEED
                  1004          ;
0498 B9           1005 DOFF:    IN      A,P1                ;GET THS STATUS
0499 37           1006          CPL     A                   ;FLIP ACC
049A 53C8         1007          ANL     A,#0C8H             ;LEAVE ONLY TWO MSB'S
049C C698         1008          JZ      DOFF                ;IF A FLAG ISN'T COVERED, LOOP
049E 8981         1009          ORL     P1,#81H             ;TURN THE MOTOR OFF
04A0 947B         1010          CALL    LFDO                ;GO DO ONE LINE FEED
04A2 FE           1011 FFCK:    MOV     A,LINCNT            ;GET THE LINE COUNT
04A3 537F         1012          ANL     A,#7FH              ;STRIP BIT SEVEN
04A5 D300         1013          XRL     A,#00H              ;IS IT DONE
04A7 C6AD         1014          JZ      FFDONE              ;LEAVE IF IT IS
04A9 947B         1015          CALL    LFDO                ;STROBE THE SOLENOIDS
04AB 04A2         1016          JMP     FFCK                ;CHECK THE FORM FEED OUT
04AD 83           1017 FFDONE:  RET                         ;EXIT FORM FEED
                  1018          ;
04AE 23EB         1019 SETTIM:  MOV     A,#0EBH             ;GET DELAY VALUE
04B0 62           1020          MOV     T,A                 ;PUT IN TIMER
04B1 55           1021          STRT    T                   ;START THE TIMER
04B2 83           1022          RET                         ;EXIT
                  1023          ;
```

```
LOC  OBJ          SEQ          SOURCE STATEMENT

04B3 42           1024 PRNTBK: MOV     A,T              ;GET THE TIMER
04B4 37           1025         CPL     A                ;TWOS COMPLEMENT ACC
04B5 17           1026         INC     A
04B6 17           1027         INC     A
04B7 17           1028         INC     A
04B8 17           1029         INC     A
04B9 17           1030         INC     A                ;ADJUST TIMER
04BA 62           1031         MOV     T,A              ;PUT IT BACK IN THE TIMER
04BB 89           1032 INLOOP: IN      A,P1             ;READ PORT 1
04BC F2C8         1033         JB7     CONPBK           ;IF SENSOR IN NOT COVERED, LEAVE
04BE 84BB         1034         JMP     INLOOP           ;OTHERWISE LOOP
04C0 55           1035 CONPBK: STRT    T                ;START THE TIMER
04C1 16C5         1036 CONPB:  JTF     RDTOPT           ;SEE IF READY TO PRINT
04C3 84C1         1037         JMP     CONPB            ;OTHERWISE LOOP
04C5 23FF         1038 RDTOPT: MOV     A,#0FFH          ;LOAD A
04C7 62           1039         MOV     T,A              ;PUT IT IN THE TIMER
04C8 83           1040         RET                      ;EXIT
                  1041         ;
                  1042         ;THIS ROUTINE ADJUSTS AND SAVES THE STATUS DURING PRINTING
                  1043         ;
04C9 FD           1044 STACHK: MOV     A,STATUS         ;GET THE STATUS
04CA 92D2         1045         JB4     LFSET            ;SET LINE FEED BIT
04CC AA           1046 B4RET:  MOV     SAVPNT,A         ;SAVE THE STATUS
04CD 53C2         1047         ANL     A,#0C2H          ;RESET EVERYTHING EXCEPT
                  1048                                  ;DIRECTION AND PRINT
04CF AD           1049         MOV     STATUS,A         ;PUT THE STATUS BACK
04D0 8413         1050         JMP     LPRNT1           ;EXIT
04D2 4320         1051 LFSET:  ORL     A,#20H           ;SET BIT 5
04D4 84CC         1052         JMP     B4RET            ;JUMP BACK
                  1053         ;
                  1054         ;THIS ROUTINE READS A CHARACTER AND PUTS IT IN THE ACC
                  1055         ;
04D6 99EF         1056 GTCAR:  ANL     P1,#0EFH         ;SET ENABLE BIT
04D8 88           1057         MOVX    A,@INBUF         ;READ THE CHARACTER
04D9 8910         1058         ORL     P1,#10H          ;RESET ENABLE BIT
04DB 83           1059         RET                      ;EXIT GTCHAR
                  1060         ;
                  1061         ;THIS ROUTINE TURNS THE MOTOR ON
                  1062         ;
04DC 99FE         1063 MOTON:  ANL     P1,#0FEH         ;TURN MOTOR ON
04DE 83           1064         RET                      ;EXIT
                  1065         ;
                  1066         ;THIS ROUTINE TURNS THE MOTOR OFF
                  1067         ;
04DF 8901         1068 MOTOF:  ORL     P1,#01H          ;TURN MOTOR OFF
04E1 83           1069         RET                      ;EXIT
                  1070         ;
                  1071         END                      ;DONE

USER SYMBOLS
ARND   0107   ARNDJP 0149   B4RET  04CC   BAKWRD 02B3   BGIN   0488   BKWRD  03B5   BUTLOP 0113   BYEBYE 0160
CASE0  0031   CASE01 0017   CASE1  0052   CASE2  00B0   CASE23 0024   CASE3  00C2   CHAR   011F   CLRMEM 0448
CONPB  04C1   CONPBK 04C0   CRFIX  01A8   CRFND  00D2   CRFOND 0062   DEAD   045C   DECTST 0472   DOFF   0498
DOLF   0071   DONEF  042A   DONEL  043C   DONER  0410   ERROR  045A   FDC    0042   FDC1   0044   FDCR   009E
FDCR1  00A8   FFCK   04A2   FFDONE 04AD   FFFIX  01B2   FINE   0178   FIRE   01CC   FIREX  01D0   FIREY  01D6
FIRST  0020   FIXDUN 0174   FIXFIN 013F   FIXUP  0109   FXCHAR 0161   FXPRNT 0191   GETSTA 0144   GOOD   0128
GTCAR  04D6   GTPRNT 0463   INBUF  0000   INCTST 045E   INLOOP 04BB   ISCHAR 01BD   JUNK1  0007   KTDUN  01E0
LDBUF  0188   LFCRCK 014A   LFDO   0478   LFFIX  01AB   LFLP1  047F   LFLP2  0481   LFSET  04D2   LFTEST 017F
LINCNT 0006   LINEFD 0478   LKHI   03A6   LKHI1  03B9   LKLO   02A6   LKLO1  02B7   LNMODE 011C   LOOPW  007A
LOP1   0491   LOP2   0493   LPRNT  0011   LPRNT1 0013   MAX    006F   MOTOF  04DF   MOTON  04DC   NOFF   010F
NOLF   011A   NOTDON 048F   NT1    01D4   OUTBUF 0001   OVR    00BA   OVR1   0005   PAGE1  02A0   PAGE2  03A0
PIT    0068   PRNT   000A   PRNTBK 04B3   PRNTIT 01C1   RDTOPT 04C5   SAVPNT 0002   SELF   041F   SELF1  0421
SELFA  0411   SELFAA 0438   SELFB  040F   SELFBB 042E   SELFC  040D   SELFCC 042C   SETTIM 04AE   SETUP  0408
SHORT  01CA   STACHK 04C9   STATUS 0005   STBCNT 0003   STBIT1 0158   STPRNT 0159   SUB1   0139   TABLE1 0200
TEMP1  0004   TSJTF  01DC   TSTJTF 0467   VARSET 043F   WATCH  0075   WATCHD 00AE

ASSEMBLY COMPLETE,  NO ERRORS
```

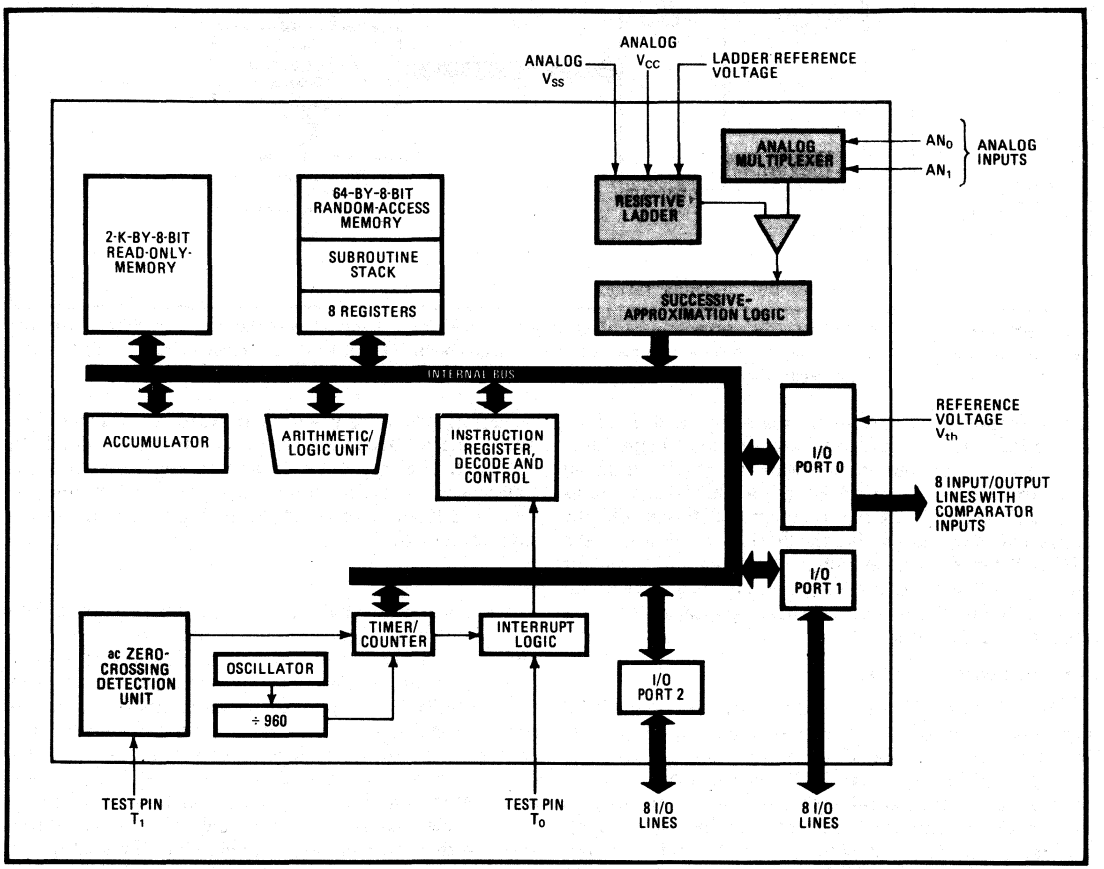
Microcontroller includes a-d converter for lowest-cost analog interfacing

Adding hardware for analog-to-digital conversion to a single-chip microcomputer cuts interface software and component count for high-volume control applications

by W. Check, E. Cheng, G. Hill, M. Hollen, and J. Miller, *Intel Corp., Santa Clara, Calif.*

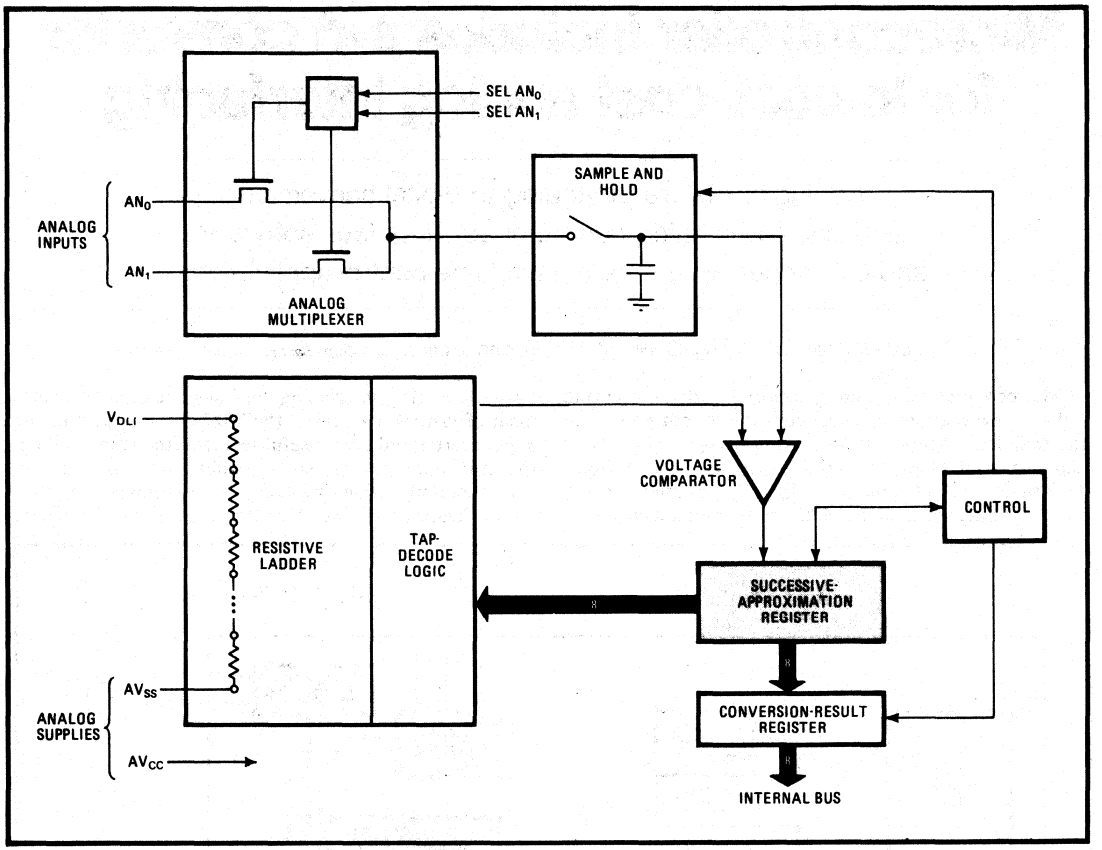
☐ Microcomputers' plunging size and cost are creating a rising new market: low-cost controllers that end up in automobiles, appliances, and consumer products. Now that the technology is available to integrate a high-performance 8-bit analog-to-digital converter and a microcomputer on a single chip, the tremendous need for low-cost analog interfacing has hastened the development of just such a device: the 8022. By integrating the a-d converter and other useful features, the chip achieves the minimum system cost possible for high-volume controller applications involving analog signals.

The heart of the 8022 is the 8021 general-purpose



1. All aboard. The first single-chip microcomputer with a built-in 8-bit analog-to-digital converter is Intel's 8022. Around a foundation of the 8021, the chip packs several features that suit it to control applications: two multiplexed analog inputs, a zero-crossing detector, two 7-mA digital outputs that are part of Port 1, and a total of 26 digital input/output lines, eight of which have voltage-comparator inputs.

Reprinted from **Electronics**/ May 25, 1978



2. The converter. The 8022's a-d converter uses successive approximation. A multiplexer selects either of two inputs, which is sampled and held. The successive-approximation register holds a byte that taps off a voltage from a 256-resistor divider through decoding logic. Input is compared with tapped voltage; when the two are equal, the held byte is sent to the conversion-result register.

microcomputer with built-in read-only and random-access memories, which go a long way since many functions are carried out in hardware or require minimal software. The 8021's modular design facilitates its use as a cornerstone for more highly integrated designs like the 8022. This new design, like the 8021, is a member of the MCS-48 family of single-chip microcomputers, and its on-chip a-d converter makes the family even more useful in such high-volume, cost-sensitive application areas as household appliances.

A microcomputer plus

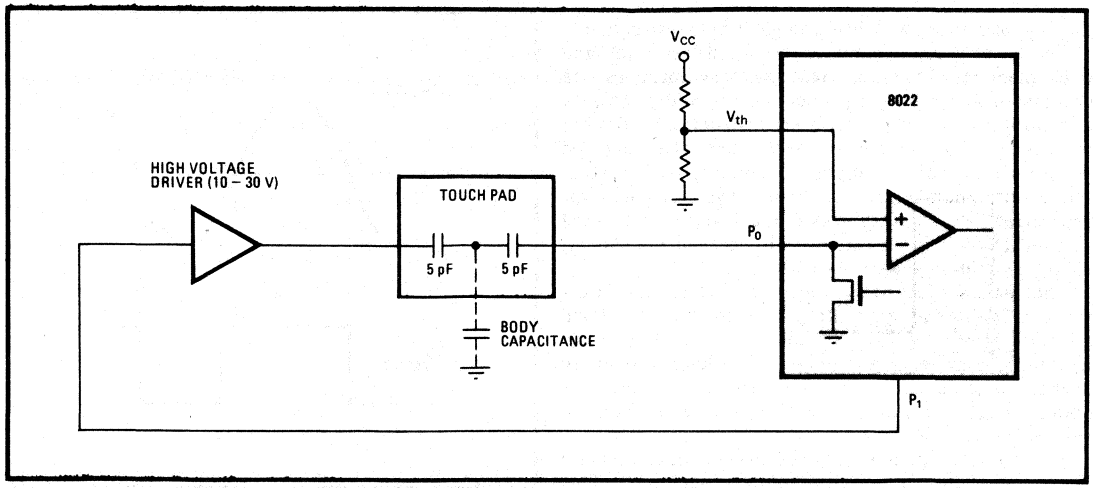
Operating on a single +5-volt power supply, the 8022 contains all the functions necessary for digital processing, plus digital or analog control. On the chip, as diagrammed in Fig. 1, are 2 kilobytes of ROM, 64 bytes of RAM, an 8-bit central processor with more than 70 instructions (a subset of the higher-performance 8048), an internal timer/event counter, a clock and oscillator, the 8-bit a-d converter with two analog inputs, and 26 digital input/output lines.

All parts of the a-d converter are integrated onto the chip—no external components are required. Conversion is performed entirely with hardware by a successive-approximation technique and takes 40 microseconds to complete. The only software is three single-byte instructions: select analog-input 0 (SEL AN0), select analog-input 1 (SEL AN1), and read the analog-to-digital conversion result (RAD).

Flexible I/O lines

The 26 digital input/output lines are organized into three 8-bit general-purpose ports and two test pins, T_0 and T_1. The three ports are quasi-bidirectional—each line can be programmed for input or output. Adding to the flexibility is an optional mask operation that eliminates the pull-up resistor for the metal-oxide-semiconductor drive transistor on each line, creating an open-drain output. The open drains are useful in driving analog circuits and for certain loads such as keyboards.

Port 0 also has variable-threshold voltage-comparator inputs with a common reference pin (V_{th}). This setup can accommodate such input situations as high noise margins, low-voltage (10-to-15-v) touch switching, and expansion of the analog inputs. Two input/output pins ($P1_0$ and $P1_1$) provide for high-current drive; each sinks



3. Low-voltage touch. Because port 0 has variable-threshold comparator inputs on each of its eight lines, new input configurations are possible, such as this low-voltage touch switch. Touching the panel momentarily pulls down the comparator input. The high-voltage driver, which may be a single transistor or part of a hex driver, then recharges the panel. The port is read by the microcomputer as is any other.

7 milliamperes, more than four times the 1.6-mA load of standard transistor-transistor-logic outputs. In many applications of the 8022, 7 mA can eliminate the need for discrete drive transistors.

The lower half of port 2, in addition to serving as general input/output, may be hooked up as a bus for attaching I/O expander units, such as the 8243, or discrete TTL parts for low-cost I/O expansion. Operations of the 8243 are synchronized by the port-expander strobe pin, a feature that is especially useful for input/output expansions designed with standard transistor-transistor logic gates.

The two test-pin inputs can be tested directly with two conditional-branch instructions. T_0 can interrupt the system, while T_1 also can detect the zero crossing of ac signals—a plus when it comes to firing triacs for phase control of motors.

The a-d converter

The 8022's a-d converter has two multiplexed input channels. Channel selection by either the SEL AN0 or SEL AN1 restarts the conversion sequence. A valid digital value can be read with the RAD instruction during the fourth instruction cycle after a select instruction. Conversions occur continuously, and RAD may be executed at any time with confidence that the sample is no more than 40 μs old. Typical software for reading two sequential a-d conversions would be:

```
SEL AN0        Starts conversion
MOV R0,#24     Setup memory pointer
RAD            First conversion to accumulator
MOV @R0,A      Store first value
INC R0         Ready for next conversion
RAD            Second conversion to accumulator
MOV @R0,A      Store second value
```

As shown in Fig. 2, the conversion hardware itself has three parts: a series string of resistors, a voltage comparator, and successive-approximation logic. The string of 256 resistors divides the voltage between V_{ss} and V_{DLI} (the reference pin) into 256 voltage steps. This configuration gives the converter inherent monotonicity. Decode logic selects the appropriate tap and transfers that voltage to the comparator block.

The conversion logic

The comparator amplifies the difference between the analog input and the voltage tap. This difference is presented to the successive-approximation logic. Eight comparisons result in a fully converted byte being transferred to the conversion-result register. All comparisons are performed automatically by on-chip hardware; executing the RAD instruction moves the contents of the CRR to the accumulator.

Novel circuit design (see "The a-d converter: how it was done," p. 27) gives the converter 8-bit resolution over the full input range of V_{ss} to V_{cc}. This capability simplifies direct connection to sensors, reduces software, and provides fast, 40-microsecond conversions. The separate power-supply pins complete the analog block and keep the converter isolated from digital-noise sources.

The instruction set

To conserve memory and maximize throughput, most instructions in the 8022 are single-byte and single-cycle; no instructions are longer than 2-byte, two-cycle. The cycle time is 10 μs.

The overall efficiency of the instruction set is enhanced for control applications by the extensive conditional-branch logic that has been built into the microprocessor. For example, the instruction to decrement a register and jump if not zero (DJNZ) allows loops to be formed in one 2-byte instruction. Similarly, the instruction to move to the accumulator from the current page (MOVP A, @ A) allows table look-up for constants or

display formatting with just a single 2-byte instruction.

The 64-byte RAM integrates the hardware stack and data memory. The first eight memory locations are designated as working registers and are addressable by any of the 11 direct-register instructions. Besides increasing the variety of operations that can be performed on data in memory, this approach further reduces the number of instruction bytes required for processing. Working registers 0 and 1 also may be used as pointers to indirectly address all locations in memory, using the indirect-register instructions.

The next 16 bytes of RAM may be used as the address stack to enable the processor to keep track of the return addresses generated from call instructions and to handle interrupts. Since each address is 11 bits long, 2 bytes are needed to store each address. Thus, the 16 bytes of address stack allow a total of altogether eight levels of subroutine nesting.

A 3-bit stack pointer supplies the locations that are loaded with the next return address generated. This stack pointer is incremented when a return address is stored and decremented when an address is fetched during a return. If an application does not require all eight levels of subroutine nesting, the free portion of the address stack may be used as standard RAM.

Other on-chip features

The 8022 contains its own clock and oscillator circuitry and requires only an external timing control element to generate all internal timing signals. For highly cost-sensitive applications an inductor may be used as this element. If a more precise clock is required, the designer may specify a crystal or external clock for the application.

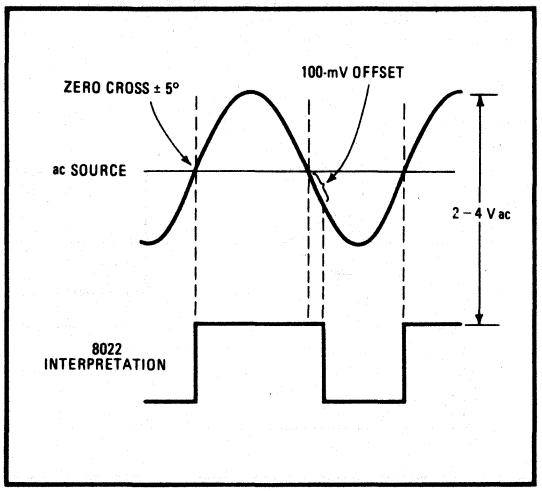
To further reduce the user's system cost and to permit use of the chip in noisy environments, the power-supply tolerance has been increased, permitting a range from 4.5 to 6.5 v. Less filtering and regulation is necessary, therefore, and the microcomputer's immunity to noisy power supplies is greater, as well.

The programmable 8-bit timer/event counter accurately monitors elapsed time, avoiding the software overhead of timing loops. Once it has been loaded with the contents of the accumulator, its divide-by-32 prescaler is incremented for each system clock cycle and at prescaler overflow. A timer flag is set at overflow. Once activated, it can be tested by a conditional-branch instruction to generate an interrupt. Total count capacity is 8,192 instruction cycles or 81.9 milliseconds, for the 10-μs cycle time.

The timer may also be used as an event counter where the test pin T_1 serves as a counter input. Upon command, the chip will respond to a low-to-high transition on the pin by incrementing its timer.

Comparator inputs

The input/output port 0 of the 8022 has several properties that ease analog interfacing problems. Two of these features are moderate-gain voltage comparators and pull-up resistors on each line that either may serve as standard TTL outputs or may be masked out to give open-drain outputs.



4. Zero-crossing detector. Useful in timing the firing of triacs for ac phase control of appliances or getting a real-time clock, the 8022's T_1 test pin detects the crossing of a waveform's dc level by its rising edge. One hundred millivolts of hysteresis prevents chattering, and the ac frequency is limited to 1 kilohertz.

The comparators are especially handy for troublesome inputs. The comparator at each pin accurately compares that line to the threshold-voltage reference pin, V_{th}, within about 100 millivolts in the range from V_{ss} to $V_{cc}/2$. Allowed to float, V_{th} will bias itself to the digital switch point of the other ports, and port 0 then behaves as a set of conventional digital inputs.

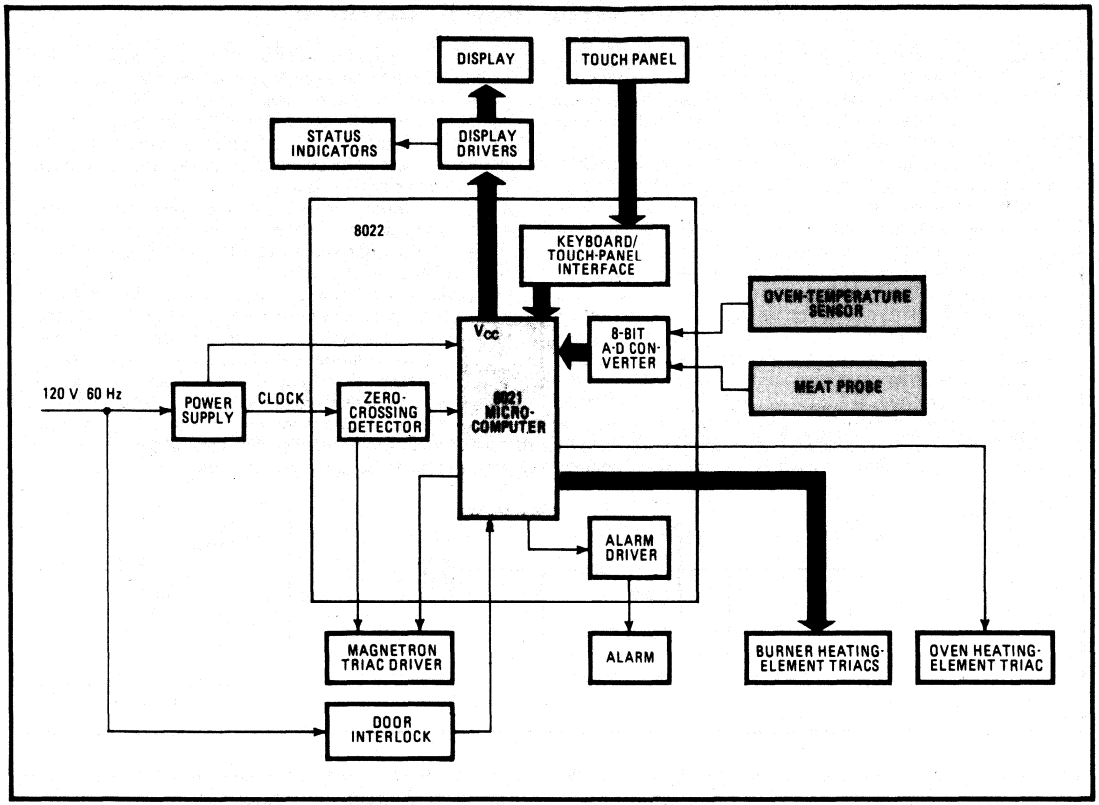
However, the switch point can be both tightly controlled and adjusted by specially biasing V_{th}. Uses for this would include high-noise-margin inputs (up to $V_{cc}/2$), unusual logic-level inputs as from a diode-isolated keyboard, analog-channel extension, and direct interfacing of capacitive touch panels. The comparator action is automatic, and the port is read just as is any other port.

Three advantages

Since the on-chip comparators allow small voltage changes to be detected, a cost-effective and safe touch panel can be built. Many appliances using touch panels have as much as 100 volts at the panel, albeit with extremely low power. The comparators in the 8022, however, permit appliance touch panels to be operated in the 10-to-15-v range.

The advantages of a low-voltage touch panel are three. First, it costs less to generate and switch the lower voltage. Then, since the keyboard operates at below 30 v, it is an Underwriters Laboratories' class II system, which can sharply cut the time required for approval. Finally, the possible product-liability problems associated with high-voltage operation disappear.

Simplified capacitive touch-panel operation is shown in Fig. 3. Contact with the panel drives both the voltage buffer and input to ground. When port 0 is read, a 0 on any line indicates a touched switch. The microcomputer drives the voltage buffer to recharge the panel. Matrix



5. Oven controller. The use of the 8022 is demonstrated in this controller for a combination microwave and conventional oven. The chip needs no assistance in figuring temperatures from thermistors connected to its analog inputs, reading inputs from a touch panel, detecting zero-crossing of ac for firing triacs and gating clocks and timers, direct-driving an alarm, and storing cooking-time instructions.

switch panels may also be sensed by the comparators.

Each pin on port 0 may or may not have an internal pull-up resistor: the option is chosen during selection of the ROM program code. If a resistor is left out for a given pin, the output appears as a true open drain for the range V_{ss} to V_{cc}. There is no temporary low-impedance drive to V_{cc}, as is the case with the remaining quasi-bidirectional ports. With open drains, accurate output waveforms can be generated, and operational amplifiers can be driven directly, for example.

The zero-crossing detector

Although the T_1 test pin on the 8022 may be driven directly by a digital input, it has special circuitry to detect an ac signal crossing its average direct-current level. The signal required for the zero-cross detection mode must be 2 to 4 v peak to peak and have a maximum frequency of 1 kilohertz. It couples to T_1 through an external capacitor.

Figure 4 shows the waveforms for zero-crossing detection. The internal digital state of T_1 is sensed as a 0, until the wave's rising edge crosses the average dc level, when it becomes a 1. The digital transition takes place within a 5° phase from the zero point. The digital level then remains at 1 until the input goes approximately 100 mv below the zero point on the falling edge. The 100-mv hysteresis keeps noise from causing chattering of the internal signal.

The zero-crossing detection capability allows the applications designer to make the 60-hertz power signal the basis for system timing. All timing routines, including time of day, can be implemented with the signal and just a few conditional jump instructions.

Moreover, since T_1 is also an input to the external event counter, the detection feature may be combined with this counter to interrupt processing at the critical zero-crossing point. Thus the user can trigger phase-sensitive devices, such as triacs and silicon-controlled rectifiers, and use the 8022 in such applications as shaft-angle measurement and speed control of motors—anywhere that the zero crossing of a waveform provides timing information.

An oven controller

The 8022's high level of functional integration provides a single-chip solution to sophisticated, high-volume controller applications that have required relatively expensive multichip designs. An example is a controller (Fig. 5) for a stove with a combined microwave and conventional oven and range-top burners.

The a-d converter: how it was done

The drive to increase the density of large-scale integration leads to continually improving control of small geometries. In fact, self-aligned silicon-gate processes now allow arrays of identical resistors and access transistors to be almost as densely packed as memory arrays.

The resistive ladder on the 8022 is a string of 256 matched diffusion resistors with access gates to each tap. Process geometry and resistivity control matches these within 8-bit accuracy without trimming or special processing. Any mismatched resistors simply expand or contract the voltage between taps. Even shorted resistors cannot cause nonmonotonic voltage outputs.

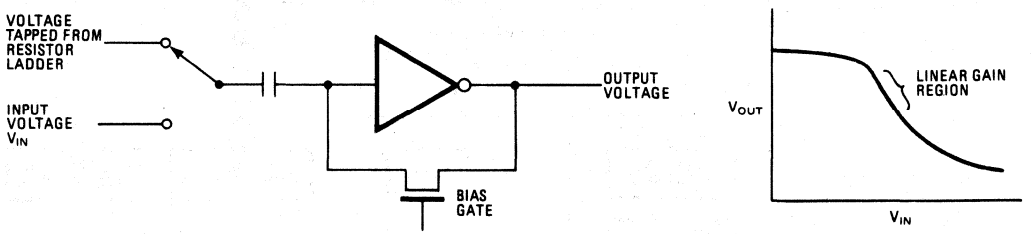
Design of the voltage comparator requires offset voltages smaller than could be expected from the standard memory/microprocessor process. So a chopper-stabilized design is used to compensate for offset inherently. Similarly, the low supply voltage of 4.5 to 6.5 volts does not allow sufficient gain or operating range from a differential stage. Thus a single-ended approach is used to increase gain. Carefully devised circuit tricks are enough to convert this stage into a differential comparator.

As shown, the basic gain stage is a logic element biased into its linear-gain region. Biasing is done while the input voltage is forced to the other side of the sample capacitor. When the bias gate is turned off and the ladder voltage is selected, the stage essentially amplifies the difference between the two voltage levels.

A string of these stages forms the comparator block. The input voltage has no effect on the amplifier bias point and therefore will not affect gain. This allows comparison down to voltages as low as V_{ss}.

Comparison with V_{cc} was made possible by judicious use of bootstrap circuitry. To limit bootstrap drivers, the voltage comparison actually occurs at half this external level. This allows all ladder select voltages to be simply V_{ss} or V_{cc}. Both resistive and capacitive dividers are used to drop the two comparison voltages to their internal level.

Finally, the capacitors inherent in the amplifier become the sample-and-hold mechanism that allows only one voltage sample to be taken per conversion.



Twenty keys enter timing and cooking instructions, and a four-digit display shows cooking time, temperature, and the time of day. Two temperature-sensing thermistors are employed, one for standard use and the other for microwave use.

While such a system could be controlled by a conventional 4-bit or 8-bit microcomputer, external circuitry would be required to interface the keyboard, convert the analog signals to digital data, drive an audio alarm, and determine the zero-crossing point of the 60-Hz power wave for timing functions and magnetron control. The 8022 reduces this multichip system to a single chip. The computer-plus-converter chip can save the oven maker upwards of several dollars in parts costs.

In this application, the 8022 program memory stores all control programs, cooking and power-cycling algorithms, and timing routines. Its 2-kilobyte ROM is large enough to provide for easy expansion of oven features and product differentiation. The on-chip RAM stores temperatures, power-level and timing settings, and all intermediate computational results.

The analog signals from the conventional temperature sensor and the microwave meat probe feed directly into the two analog inputs on the 8022 without any additional circuitry. What's more, the chip's 8-bit a-d converter gives more accurate temperature sensing than most existing discrete configurations.

The keyboard interfaces directly to the device through port 0. The keyboard in this application can be either a capacitive touch panel or a conventional switch type, since the 8022 directly interfaces either.

The T_1 pin in the zero-crossing detection mode establishes an accurate time base for all timing routines, including cooking cycles, presetting functions, and time of day. To accomplish this, the chip detects a zero crossing using the two conditional-jump instructions associated with T_1: JT1 and JNT1. Then it increments a register in data memory, effectively keeping track of elapsed time. Using this technique, a time-of-day routine can be written for most applications in less than 30 bytes of code.

Control of the magnetron

The zero-crossing detection capability also efficiently controls the microwave's magnetron. To minimize current surges through the system, the magnetron should be fired at the peak of the ac wave (90°). To achieve this performance, the 8022 detects the zero crossing point with its T_1 pin and delays the 90° phase shift with the internal timer.

The high-current drive pins, $P1_0$ and $P1_1$, are tied together to directly drive a piezoelectric alarm, which requires 10 to 15 mA of current. The remaining I/O lines are used to drive the display and status indicators, to monitor the door interlock, and to control the triacs that switch the burner and oven heating elements. The internal timer controls the refreshing of the displays and the scanning of the keyboard. □

Microcomputer's on-chip functions ease users' programming chores

The one-chip 8022 includes hardware, such as an a-d converter, that combines with the instruction set for easy development of routines

by William F. Ittner and Jeffrey A. Miller, *Intel Corp., Santa Clara, Calif.*

☐ A single-chip microcomputer that incorporates analog-to-digital conversion, comparator inputs, and ac zero-crossing detection is a strong candidate for low-cost, high-volume applications. Moreover, to maintain its front-runner position, the new 8022 has been designed for ease of programming: many common routines are invisible to the user because they are performed in on-chip hardware.

The 8022's instruction set, in conjunction with its hardware features, affords programming ease in the development of routines for translating analog signal levels, monitoring temperatures, reading capacitive-touch-panel inputs, controlling phase-sensitive thyristors, and calculating the time of day. For example, performing an a-d conversion requires software only to select the appropriate analog input; the actual conversion is performed entirely in hardware. This leaves room in the program memory for additional system functions. Furthermore, the instruction set accommodates bit handling, binary and binary-coded-decimal arithmetic, and direct table look-up, and it has extensive facilities for input selection and input-based program jumps.

The 8022 [*Electronics*, May 25, p. 122] is the first general-purpose single-chip microcomputer to offer an on-chip a-d converter. While retaining the 8-bit central processing unit, 64 bytes of random-access memory, clock, zero-crossing detection, and timer/event counter featured in its 8021 predecessor, the new chip doubles the read-only memory to 2 kilobytes and provides comparator inputs on eight input/output lines, five more digital I/O lines (including an extra test pin), full interrupt capability, and two 8-bit a-d input channels. Such a decrease in system component count cannot help but minimize cost and increase reliability.

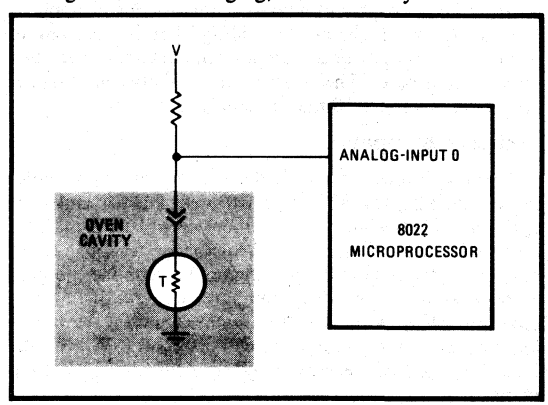
Easy a-d conversion

The 8022's a-d converter has two multiplexed channels, selectable with the SEL AN0 (select analog input 0) or SEL AN1 (select analog input 1) instructions. Built-in successive-approximation hardware accomplishes the conversion. The select instructions and the RAD command (read a-d conversion result) are the only software instructions necessary. The select instructions restart the continuously occurring conversion process, but do not affect the conversion-result register. The new valid digital value can be read from the CRR during the fourth cycle after a select instruction and every fourth instruction cycle thereafter.

An application that points up the advantages of this easy-to-use on-chip converter is monitoring temperature in an oven controller. The temperature is sensed by a thermistor probe located in the oven (Fig. 1). In such a system, noisy analog signals are apt to prevail, obscuring the readings. But since so few instructions are needed for each sampling, a software filtering technique can be added at little expense for increased accuracy. One software filtering method is to average each reading with the previous samples:

```
SEL AN0          ;Start conversion
MOV R0, #30      ;Point to storage location of previous
                  a-d sample average
RAD              ;Read second sample result
ADD A,@R0        ;Add last sample to new sample
RRC A            ;Divide by 2
MOV @R0,A        ;Store new average
```

Excessive noise may require averaging of many readings taken over a short period of time. Program 1 illustrates a method of computing the average of 16 readings. In such averaging, it is necessary to select the



1. Talk about simple. To sense temperature with the 8022, all that is needed is a thermistor pulled up to the supply. Simpler yet are the instructions to sense the voltage divider's potential: select analog input 0 (SEL AN_0), and read conversion-result register (RAD).

PROGRAM 1: 16 CONSECUTIVE READINGS OF SAME ANALOG INPUT

```
        MOV   R4, #00     clear temp. MSB result register
        MOV   R0, #26     set up pointer
        MOV   @R0, #00    clear result register
        SEL   AN0         select & start conversion
        MOV   R2, #16     16 readings
LOOP:   RAD               read result
        ADD   A, @R0      LSB
        MOV   @R0, A      save new LSB
        CLR   A
        ADDC  A, R4       MSB add carry to R4
        MOV   R4, A       save new MSB
        DJNZ  R2, LOOP    next reading
        SWAP  A           MSB into MSN
        XCH   A, @R0
        SWAP  A           divide by 16
        XCHD  A, @R0      location 26 in
                          RAM now contains
                          the average value
                          of 16 conversions
                          over a period of
                          1.44 msec
```

PROGRAM 2: MAGNITUDE-COMPARISON ROUTINE

```
        SEL   AN0         start conversion
        MOV   R0, #24     set up pointer
        RAD               read conversion result
        SEL   AN1         start other conversion
        CPL   A
        INC   A
        MOV   @R0, A      save first conversion
        RAD               read second conversion

        ADD   A, @R0      add first conversion
                          A equals the differential
                          in ones complement
        JZ    EQUAL       AN0 = AN1
        JC    LESTHN      AN0 <AN1
                          AN0 >AN1
```

appropriate analog channel only once. Thereafter, a new conversion result is available every four instruction cycles.

Often noise on the analog input is due to 60-hertz ac pickup. To minimize this interference, analog signals should be synchronized with the line voltage, accomplished in the 8022 by the on-board zero-crossing-detection circuitry. The combination of line synchronization with simple filtration renders digital values impervious to line-generated noise.

Signal averaging may be used for more than noise filtering. Since it can be applied to either channel 1 or channel 0 (whichever is selected with a SEL ANx instruction), each channel may monitor different functions in one system, such as temperature and pressure in a process-control application. The fast a-d conversion time permits rapid switching between the two channels.

A similar software technique permits measuring the same variable in two different locations and comparing the two results, as in checking the internal and external temperatures in an automotive climate-control application. Program 2 shows the coding that is required to perform a magnitude comparison between the two analog channels. The time elapsed between channel switching is a mere 50 microseconds.

Comparator inputs

To ease interfacing with devices presenting troublesome I/O links, the 8022's port 0 incorporates comparator inputs controlled by a common voltage-reference pin and an option of a pull-up resistor or an open-drain output. Each of port 0's eight pins has a moderate-gain voltage comparator, which compares to a common reference pin (V_{th}) with ±100-millivolt accuracy, within a analog reference voltage range of V_{ss} to $V_{cc}/2$. The biased V_{th} pin will ensure a tightly controlled switching point.

A typical use for port 0 is in the interfacing with the capacitive touch panels on microwave ovens and other new appliances. A touch-panel switch consists of two capacitors in series. One lead is attached to a high-voltage buffer (10 to 30 volts). The other is attached to the port 0 sense input. As a finger touches the common point, the drive signal is shunted by body capacitance, attenuating the signal reaching the input.

Low-voltage touch-panel operation (less than 30 v) is possible, since the comparators allow small voltage changes to be detected. Most of the present touch-panel designs require a 50-to-100-v drive on the touch panel.

Capacitive touch panels can be multiplexed in the same manner as can mechanical keyboards (Fig. 2). The vacuum fluorescent display and the touch panel are integrated to optimize hardware through shared high-voltage buffers.

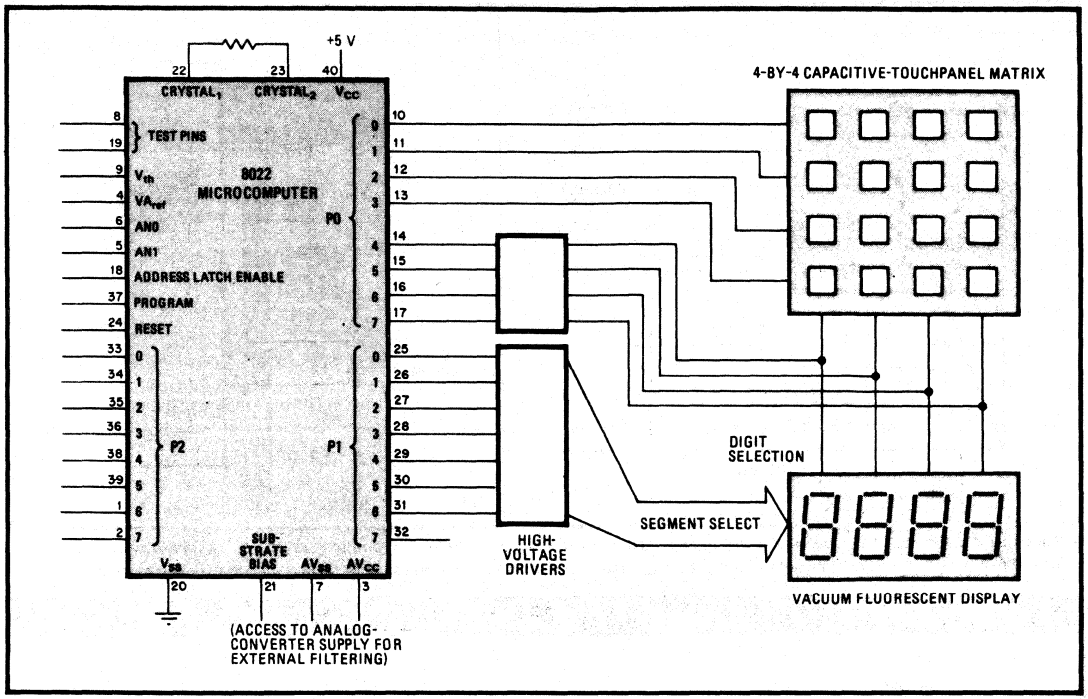
Program 3 lists the software that is necessary to refresh the display and scan the touch-panel matrix. This routine could be adapted to serve as part of a timer/interrupt scheme that would generate an interrupt at precise intervals for a flicker-free display. Another portion of the software would check for any touched input pads, test for valid entry, and enter key-depression codes into the main program.

Correcting pad imbalance

A common problem with capacitive touch panels is their imbalance. Layout process, aging, and surface impurities all cause the capacitance to vary from touch pad to touch pad, resulting in a family of curves (Fig. 3a) of voltage levels from each column of touch pads reaching the sense inputs. As the curves show, if threshold voltage V_{th1} alone were used, one column would always appear touched; if V_{th2} were used exclusively, three of the columns would never appear touched.

To compensate for such varying capacitance levels, the on-chip analog-input circuit may be used to allow multiple input voltage levels. Figure 3b depicts the 8022 version of such a circuit. AN0 and V_{th} are tied together with a capacitor to line 0 of port 0. The pull-up resistor option is used on line 0 to provide an RC timing network connected to AN0, V_{th}, and $P0_0$. The remaining seven lines of port 0 are the sense lines for the touch panel and use the open-drain-output option.

Handling the different voltage levels would begin with initializing the data by plotting the family of curves with the AN0 input. This is done by writing a 0 to $P0_0$ (grounding $P0_0$) which initializes V_{th} to 0 v. A logic 1 is then written to $P0_0$, which begins to pull the RC network



2. Multiplexed touch panel. A capacitive touch panel and high-voltage display may be combined in much the same way as a mechanical keyboard and light-emitting-diode array. To save hardware, an obvious choice is to share the high-voltage drivers.

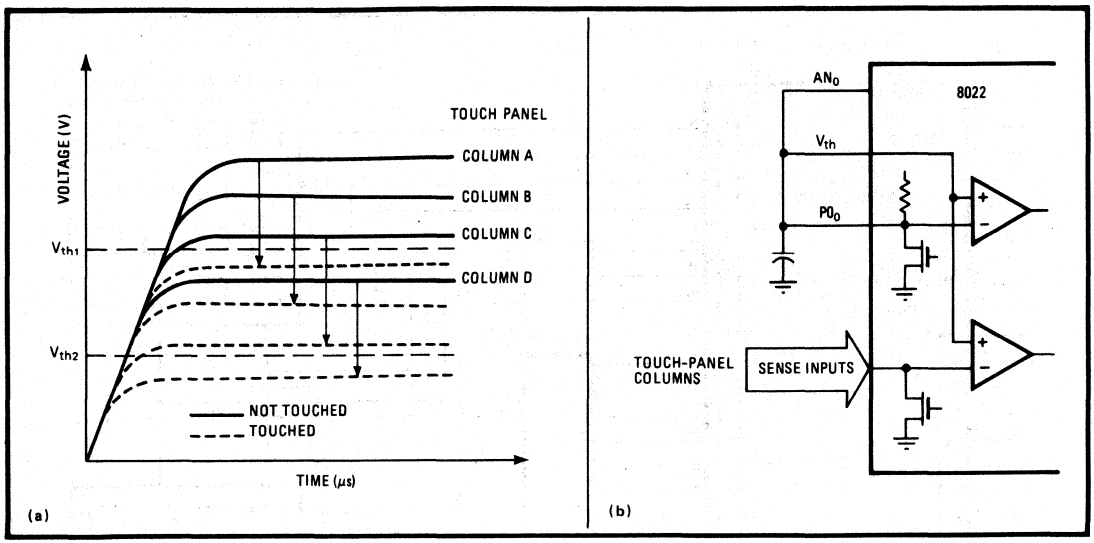
PROGRAM 3: DISPLAY REFRESH, KEYBOARD SCAN ROUTINE

```
        CLR     A
        OUTL    P1, A       turn off segment drivers
        OUTL    P0, A       turn off digit drivers and panel strobes
                            and initialize sense input to ground

        MOV     A, #0FH
        OUTL    P0, A       float sense inputs
        ORL     A, R3       new strobe position
        OUTL    P0, A       turn on strobe
        IN      A, P0       read sense inputs
        MOV     R4, A       save sense inputs

        MOV     R0, #D1SP4-1    RAM location of MSB of 7 segment pattern
        MOV     A, R3       strobe position into A
        OUTL    P0, A       GND sense inputs

LOOP:   RLC     A           rotate digit strobe into carry
        INC     R0          next digit location
        JNC     LOOP        loop until carry
        MOV     A, @R0      retrieve pattern from RAM
        OUTL    P1, A       new segment pattern
```

toward 5 v. As V_{th} ramps upward, the sense lines are monitored for input changes, which will go from 1 to 0 as the not-touched voltage curve for each input is intersected. As the changes occur, the a-d value for each sense line can be read and stored.

Thus the threshold reference voltage for each sense line can be determined by establishing the not-touched voltage levels and by placing the threshold reference voltage below this level. As each row of the keyboard is scanned, the RC network is initialized to 0 v and ramps upward, varying the V_{th} level. The a-d converter monitors this level looking for the calculated threshold points. As the points are intersected, the corresponding sense inputs are read, with a 0 indicating a touched input and a 1 indicating a not-touched input. Thus, a multiplexed capacitive touch panel can be scanned and balanced without adding external components to the inputs.

For systems requiring more than two analog inputs to



3. Balancing. Dissimilar pad capacitances are represented by the curves in (a). With a fixed reference (V_{th1} or V_{th2}), false sensing will occur. Individual thresholds can be determined using (b): a rising edge is placed on $P0_0$ through software; AN0 is then read until switching.

PROGRAM 4 "90° PHASE ANGLE ROUTINE

```
NINDEG    EQU     13              13x32x10 usec = 4.160M msec

LOC 7:    XCH     A, R7           save Acc and get flag byte
          INC     A
          JNZ     NINETY          is it zero cross or 90 deg.
          MOV     A, #NINDEG      zero cross"
          MOV     T, A            set up for ninety degrees interrupt
          STRT    T
          XCH     A, R7           load R7 with non-FF number
          RETI                    and restore A"

NINETY:
          IN      A, P1
          MOV     A, #11101111B   set P14 low (TRIAC PORT)
          OUTL    P1, A

          MOV     A, #0FFH        set up for zero cross interrupt
          MOV     T, A            next time"
          STRT    CNT
          XCH     A, R7           load FF into R7 and resotre Acc
          RETI
```

the 8022, port 0's comparator may be reconfigured to permit forming of pseudo-analog inputs from variable-threshold digital inputs. The hardware configuration can be identical to that of Fig. 3b. In this scheme, sense inputs act as additional analog inputs of less accuracy than AN0 and AN1 (about 6 bits).

The sequence for implementing the extension is essentially the same found in the variable-threshold touch panel. As V_{th} ramps upward, a port 0 bit is monitored for a change from 1 to 0. At the change, AN0 is read, corresponding to the value of the analog input into port 0 (with some error due to the time lag, which can be subtracted). This configuration can be utilized when it is possible to trade off accuracy for cost improvements: one analog input with 8-bit accuracy and seven analog inputs with 6-bit accuracy. Such may be the case in a range controller that monitors temperature in two ovens, a meat probe, and two of the four burners, all to an accuracy within 10°F.

To establish a reliable time base and to switch ac

PROGRAM 5: COMPUTING THE TIME OF DAY

```
                ORG     7               T1 timer interrupt vector
                MOV     A, #0FFH        initialize timer
                MOV     T, A

                MOV     R0, #TIME0 - 1  TIME0 = LSB of timer register
                MOV     R1, #TABLE      LSB of ROM look-up

LOOP:
                INC     R0              point to next byte
                MOV     A, @R0          retrieve BCD byte
                ADD     A, #1           increment
                DA      A               decimal adjust
                MOV     @R0, A          restore BCD byte
                MOV     A, R1           test for carry
                MOVP    A, @A           using table entry
                XRL     A, @R0          and"
                INC     R1              bump pointer
                JNZ     DONE            no carry, wait for next tick
                XCH     A, @R0          carry, set digit pair to 00
                ANL     A, #1           was overflow hours?
                JZ      LOOP            no, increment next byte
                MOV     @R0, A          yes, set hours to 1

DONE:
                RETI    \

TABLE:
                DB      60 H            sixtieth
                DB      60 H            minutes
                DB      13 H            hours use 25 for 24 hour operation
```

loads, the 8022 has circuitry built into the T_1 pin to detect an ac signal crossing its average dc level. The switching is at predetermined points of the sine wave to reduce inrush currents or radio-frequency interference.

Zero-crossing detection

There are several methods by which software can monitor the input. The simplest method involves the jump instructions JT_1 and JNT_1, which correspond to jump on T_1 high, and jump on T_1 low, respectively. The rising edge of the T_1 input is the most accurate: the falling edge contains 100 mV of hysteresis to increase noise margin. The two jump instructions can be used back to back to find this zero-crossing point:

```
HERE1: JT1 HERE1    ;Wait here if line high
HERE2: JNT1HERE2    ;Wait here if line low
                    ; Zero cross
```

To reduce loop time, the T_1 pin may also be coupled to the event counter. The start-counting instruction couples the rising edge into the 8022's internal 8-bit timer. With each rising edge, the timer increments by one, and when it increments from FF Hex to 00, an overflow flag is set. If the interrupt line is activated, an interrupt vector at location 7 will occur. Since the timer may be preloaded with any value, it is possible to cause an interrupt to occur on the next zero crossing rather than waiting in the jump loop.

The following routine will initialize the timer to accomplish this. All other processing may be performed while waiting for the zero crossing. The timer could be reloaded with FF Hex during the interrupt routine to generate an interrupt on each zero crossing.

```
MOV A, #0FFH    ;Full count into accumulator
MOV T,A         ;Load timer
STRT CNT        ;T1 pin is source to timer
EN TCNTI        ;Enable timer interrupt
```

Of course, the zero crossing is not always the best point to gate a control device. For example, an application involving a highly inductive device such as a magnetron transformer will produce inrush currents that are at their maximum at the zero-crossing point.

Low inrush

To minimize inrush in such a system, the triac is turned on at a 90° phase angle in the 60-Hz sine wave. Program 4 provides the software necessary to accomplish this task. The timer detects the zero-crossing point and times out to the 90° point, where the leading current will just be at a minimum. An interrupt occurs at both the zero and 90° points to prevent interference with normal processing. Both interrupts use the same interrupt vector location. Software determines the source of the interrupt and acts accordingly.

Another use of the T_1 input is generating the timing base for a time-of-day routine. The software implementing this routine is in program 5. The time parameters listed in the accompanying data table could be modified to accommodate either 12- or 24-hour operation. □

Designing with Intel's 8022 Microcomputer

Contents

INTRODUCTION 5-260

PRODUCT OVERVIEW 5-260

PRODUCT FEATURES 5-261

- System Clock 5-261
- Timer/Counter 5-264
- Test and Interrupt Inputs 5-265
- Analog to Digital Converter 5-267
- Port 0 Comparator Inputs 5-270

APPLICATION IDEAS 5-272

- Power Supply Controller 5-272
- DC Motor Control 5-274
- Automotive Dashboard 5-275
- Darkroom Timer 5-276

CONCLUSION 5-276

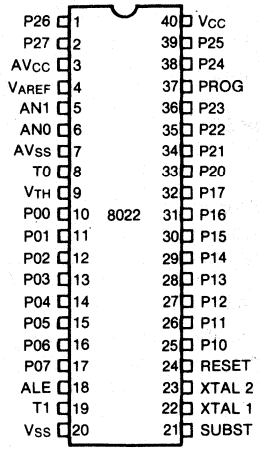


Figure 1. Pin Configuration

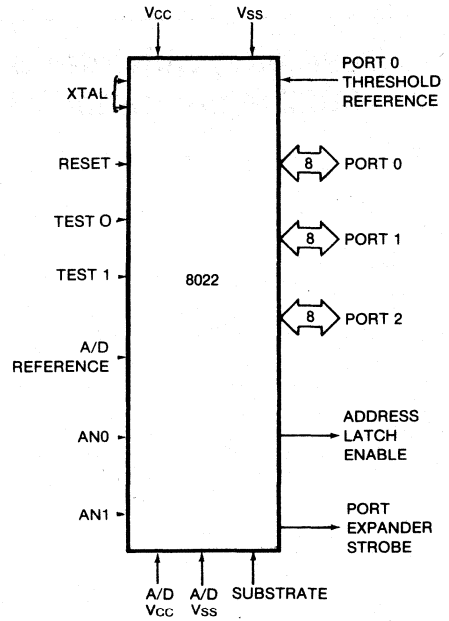


Figure 2. Logic Symbol

INTRODUCTION

Taking advantage of the latest advances in silicon technology, Intel has developed a complete control system on a chip, the 8022, the first 8-bit microcomputer with an A/D converter on-chip. Whereas in the past microcomputers relied on external circuits for analog interfacing, it is now possible to build a one chip control system with analog interfacing, digital interfacing, and computer processing capabilities. Tackling the high volume, low cost controller market, the Intel 8022 microcomputer fits cost and space sensitive applications such as automobiles, appliances, and consumer products previously dominated by electromechanical controls. Its use, however, is not confined only to these applications. In medium volume applications, the 8022 provides the system designer with a simplified solution to many control problems. No longer is it necessary to expend valuable engineering time designing wheel spokes and axles; the whole cart is available.

This note is intended to answer some design questions concerning the 8022 and to suggest to the reader possible applications and system configurations. The reader should refer to the 8022 Data Sheet for electrical specifications and details. It is also suggested that the reader consult with the MCS-48 User's Manual (July 1978 or later) for a complete description of the entire MCS-48 family of microprocessors of which the 8022 is the newest member.

The note is divided into two main sections. The first is a product description of the 8022, including a detailed discussion of the main features, their characteristics and how to use them, The second section discusses several possible applications, their configurations and design considerations.

Product Overview

The heart of the 8022 is the Intel 8021, a general purpose single chip microcomputer, which is a lower performance, lower cost version of the 8048. Added to this central core are interrupts, additional I/O, and linear functions. Like the 8021, the 8022 is designed to operate over a power supply range of 4.5 to 6.5 volts.

The 8022 instruction set contains over 70 instructions and is a subset of the 8048 instruction set. To conserve memory and maximize throughput, most instructions are single-byte, single-cycle. No instructions are longer than two-byte, two-cycle. The instruction cycle time is 10 microseconds at a 3MHz clock rate. Extensive conditional branch logic is built into the processor to increase the overall efficiency of the instruction set for control applications. As examples, the DJNZ instruction (decrement register and jump if not zero) allows loops to be formed in just one instruction and the MOVP A, @A allows single instruction table look-up of constants from program storage. Program storage in the 8022 consists of 2048 eight bit bytes of mask programmable ROM.

Hardware stack and data memory are integrated in the 64 byte RAM to enhance processing flexibility and memory utilization. The first eight RAM locations are designated as working registers and are directly addressable by any of the 11 direct register instructions. Besides increasing the variety of operations that can be performed on data in memory, this approach further reduces the number of instruction bytes required for processing. In addition to being used as working registers, Registers 0 and 1 can be used as Pointer registers to indirectly address all locations in memory using the indirect register instructions.

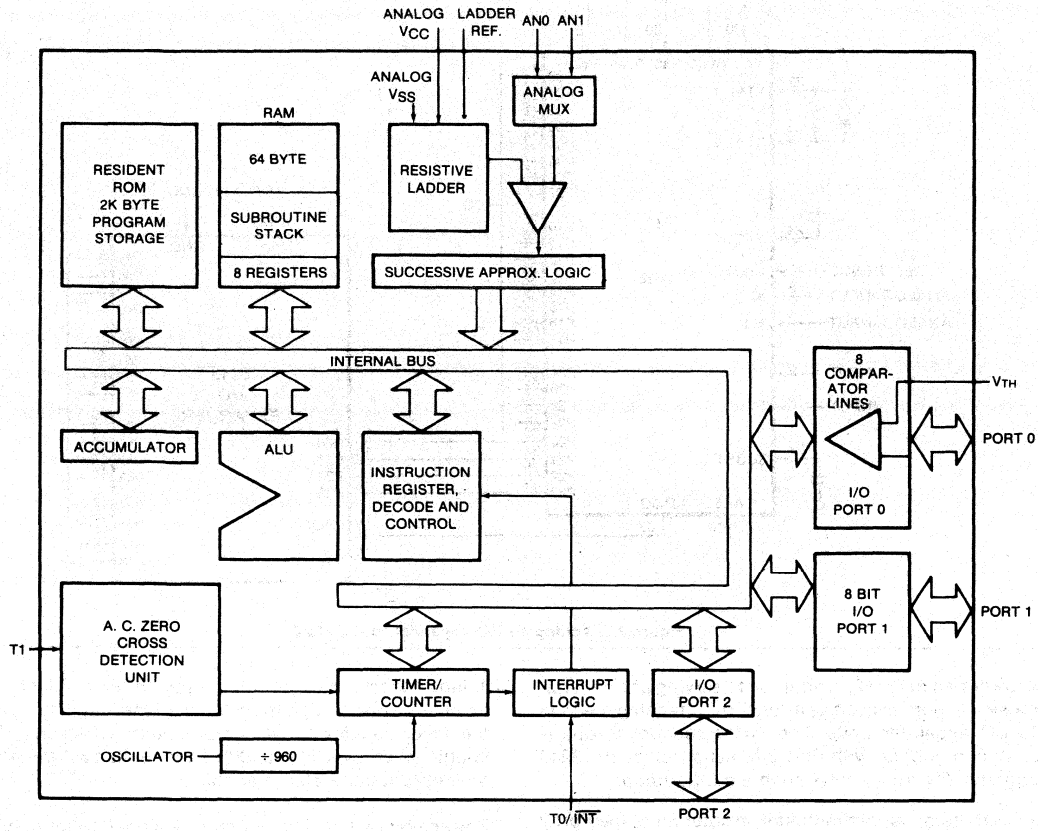


Figure 3. 8022 Block Diagram

The next 16 bytes of RAM may be used as the address stack to enable the processor to keep track of the return addresses generated from instructions and in handling interrupts. Since two bytes are needed to store each address, the 16 bytes of address stack allow up to a total of eight levels of subroutine nesting. A 3-bit stack pointer supplies the address of the locations to be loaded with the next return address generated. This stack pointer is incremented when a return address is stored and decremented when an address is fetched during a subroutine or interrupt return. If all eight levels of subroutine nesting are not required by an application, the unused portion of the address stack may be used as standard RAM.

The 8022 has an extremely flexible and powerful I/O structure. The 26 digital I/O lines are configured into three 8-bit general-purpose ports and two test pins, T0 and T1. All three ports are quasi-bidirectional, meaning all lines are useable as inputs or outputs on a line-by-line basis under software control.

To increase the user's flexibility, any line of Port 0 can also be designated an open drain output by removing the pullup device present on the line via mask option. This is useful in driving analog circuits and interfacing to high impedance digital I/O. In addition to the open drain option, Port 0 has voltage comparator inputs with a common reference pin (V_{TH}). In appliance control and other applications, this allows direct glass touchpanel interfacing with relatively low voltage (10-15V) drive, thus limiting product liablility problems and easing U.L. approval. The Port 0 comparator inputs are also generally useful in many other ways from expanding analog inputs to maximizing margin on noisy signals.

To further increase user flexibility and reduce system cost, two I/O pins (P10 and P11) have been designated as high current drive pins with the ability to sink 7ma each, instead of the standard TTL load of 1.6ma. This can eliminate the need for discrete drive transistors in many applications.

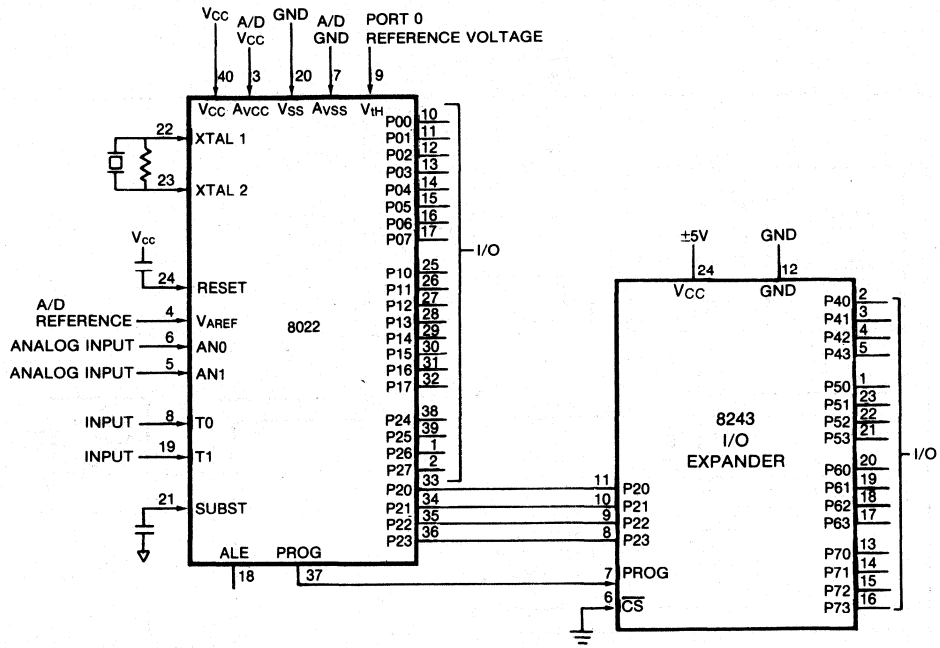


Figure 4. Adding an I/O Expander to the 8022

The lower half of Port 2, in addition to serving as a general-purpose I/O port, is used as a "bus" for attaching the Intel 8243 I/O expander units. The Port Expander Strobe is used in conjunction with Port 2 to synchronize the 8243 operations. Figure 4 shows such a configuration.

Note that the quasi-bidirectional structure and the Port 2 expansion bus are consistent with all MCS-48 products and are fully described in the MCS-48 User's Manual.

Frequently in control applications, the state of one or two signals must be monitored so that a fast response can be accomplished. The 8022's two test pins offer this capability. Both test pins, T0 and T1, are directly testable via two conditional branch instructions. The T0 pin can also cause an interrupt. The T1 pin, in addition to being directly testable, has the ability to detect the zero crossing of slowly moving AC inputs. This is useful in controlling 50/60Hz power. It also enables the 8022 to precisely control phase sensitive devices, such as triacs and SCRs. Again external circuitry is reduced.

The 8022 contains its own clock and oscillator circuitry and requires only an external timing control element to generate all internal timing signals. An inductor, a crystal, or an external clock may be used as the timing control device.

The programmable 8-bit timer/event counter enables the user to accurately monitor elapsed time by providing a hardware replacement for software overhead such as timing loops. Total count capacity is 8192 instruction cycles or 81.9 msec at a 10 microsecond cycle time. The timer may also be used as an event counter where the Test 1 input serves as a counter input. After a STRT CNT command, low to high transitions on the T1 pin will cause the timer/counter to be incremented. When the timer counter overflows (FFH to 00), the timer flag will be set and an interrupt generated if enabled.

The analog to digital converter is designed to simplify and cost reduce interfacing to analog sources. All parts of the converter are integrated onto the chip, with the exception of the voltage reference. Conversion is completely hardware controlled using a successive approximation technique and occurs in four instruction cycles or 40 microseconds. Three single byte instructions, SEL AN0 (select analog input 0), SEL AN1 (select analog input 1), and RAD (read A/D conversion result) are added to the 8021 instruction set to allow the programmer to interface to the converter conveniently.

Product Features

This next section will delve deeper into some of the functions which comprise the 8022 architecture. Chip architecture will be discussed along with design considerations, software routines, and hardware configurations. The specific items covered are CPU timing, the Timer/Counter, the TEST and Interrupt inputs, Zero Cross detection, the A/D converter, and the Port 0 comparator inputs.

System Clock

One of the first considerations in the system design is what frequency source should be used. The on-board oscillator can use a variety of elements to determine system fre-

quency. Depending on the accuracy needed, the element can be an inductor and capacitor, or a crystal and resistor. If necessary, the oscillator inputs can also be driven by an external source.

It should be noted that the values given in this section are approximate values based on a sampling of parts. In no case are these to be interpreted as guaranteed specifications. They are here as an aid in system design. Consult the final Data sheet or contact Intel direct if more information is needed for a critical design.

Inductor Mode

Figure 5 shows the proper configuration for the inductor mode. A parallel capacitor of 20 to 50pf is recommended for best frequency tolerance.

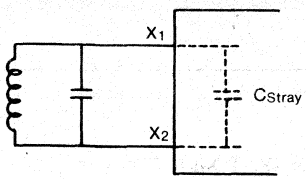


Figure 5.

Table 1 shows the effects of changes in parameters based on a sampling of parts. Part to part input capacitance differences (Cstray) will effect the tolerance. A less than 0.2% part to part tolerance can be expected with a parallel capacitance of 50pf. (see fig. #5). An additional 0.5% variation comes about when only 20pf is used in the tank circuit. This is because the stray capacitance in the 8022 and the PCB becomes a larger proportion of the total capacitance.

Vcc =	4.5v	5.5v	6.5v
f =	≃±0.2%	0	≃0.2%
Temp =	-40°C	25°C	85°C
f =	±0.6%	0	≃0.6%

Table 1. Inductor Mode

To determine the inductance and capacitance required for a given frequency, the equation

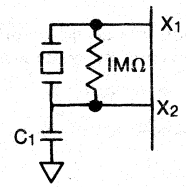
$$f = \frac{1}{2\pi\sqrt{LC}}$$

can be used. Due to the effects of stray capacitance the calculated frequency may be slightly high. It should be noted that the tolerances given in Table 1 do not include the tolerances of the inductor and capacitor used in the system. Mathematical analysis of the above equation will show that the frequency will change roughly proportional to the tolerances of L and C on a worst case situation. That is if both L and C are ±5% parts, the frequency will vary approximately ±5%.

Crystal Mode

Figure 6 shows the proper installation of a crystal. A one meg-ohm parallel resistor is required for operation with an 8021 or 8022. Application note AP-35 "CRYSTALS: Specifications for Intel Components" should be consulted for information on using and specifying crystals.

A 20pf capacitor is optional, but recommended, on X2. It has been found that using the capacitor increases the immunity of the microcomputer to line transient noise or spurious signals which may find their way into the system.



C_1 is an optional 20pf capacitor

Figure 6.

Which One?

Which timing source to use is dependent on several factors. In most applications cost is of primary importance. The lowest cost device, but one which still gets the job accomplished, is the logical choice. Selecting the device which gets the job accomplished is the next task.

A Case Study

To exemplify the design tradeoffs in choosing a timing element consider the detection of 50Hz or 60Hz line frequency as may be needed in many consumer products being sold in the U.S. and overseas. Traditionally two products are produced, one for the U.S. market and one for the overseas market. A jumper selection to tell the processor which frequency source is being used is the only difference. This costs one I/O pin plus the costs of insertion and inventorying two products. All of these costs can be saved by allowing the processor to compute which frequency is coming in on the T1 pin. Figure 7 lists the software which could be used during a power-up routine to determine whether 50Hz or 60Hz timing should be used.

The timer is used to time the interval of one line cycle. If everything were perfectly accurate, one count would equal 50Hz while another count would equal 60 Hz, but it's not. The power company frequency may shift slightly, plus the 8022 oscillator may drift as discussed earlier. The maximum allowable oscillator change must be calculated from the input source. Assuming the power companies may drift ±2 cycles, then the processor must be able to detect a difference of 58Hz-52Hz = 6Hz or less than 10.3% change. This means that the oscillator frequency itself cannot change more than 10.3% or ±5.15%. The crystal would definitely work but may be overkill. The Inductor/capacitor combination could be the most economical solution.

The equation

$$\frac{\frac{1}{LF}}{\frac{1 \times 30 \times 32}{f}} = \text{count}$$

where LF = line freq,
f = osc. freq.

will give the value of the time at the end of one line cycle. Plugging in the values for an oscillator of 3MHz ±5% and a ±2 cycle deviation in line freqency, the counter will yield counts of:

47-56 = 60Hz
57-68 = 50Hz

Inductor and capacitor components could be picked to yield the required tolerance, saving the costs previously mentioned.

Timer/Counter

An 8-bit interval timer/counter is available to enable the user to keep track of time elapsed or number of events occured while normal program execution and flow continues. The Auto 50/60Hz detection routine previously discussed is one of many possible applications of the timer/counter.

The timer/counter consists of a divide by 32 prescaler (only used in the timer mode) and an eight bit main timer. The STRT T command clears the prescaler and thereafter it increments once each 30 input clocks (once each single cycle instruction, twice each double cycle instruction). At the (11111) to (00000) transition the timer is incremented. A timer overflow from (FFH) to (00H) will set the timer flag along with the timer interrupt, if enabled (see below). A conditional branch instruction (JTF) is available for testing

```
LOC  OBJ        LINE        SOURCE STATEMENT

                   1 ;
                   2 ;       AUTO 50/60HZ DETECTION ON POWER UP
                   3 ;       ==================================
                   4 ;
                   5 PWRUP:
0000 27            6         CLR     A          ;CLEAR ACCM
0001 0414          7         JMP     PWRDET     ;JUMP AROUND INTERRUPT ROUTINES
                   8 ;
                   9 ;
                  10 ;       INTERRUPT ROUTINES HERE
                  11 ;
                  12 ;
0014              13         ORG     20
                  14 ;
                  15 ;       MEASURE ONE LINE CYCLE
                  16 ;       ----------------------
                  17 ;
                  18 PWRDET:
0014 5614         19         JT1     PWRDET     ;WAIT FOR NEXT RISING EDGE
                  20 LINLOW:
0016 4616         21         JNT1    LINLOW     ;WAIT FOR NEXT RISING EDGE
0018 62           22         MOV     T,A        ;CLEAR TIMER
0019 55           23         STRT    T          ;START TIMER
                  24 LINEHI:
001A 561A         25         JT1     LINEHI     ;WAIT HERE FOR LINE TO GO LOW
                  26 RISEDG:
001C 461C         27         JNT1    RISEDG     ;WAIT HERE FOR RISING EDGE
001E 65           28         STOP    TCNT       ;STOP TIMER AT END OF ONE LINE CYCLE
001F 42           29         MOV     A,T        ;READ TIMER VALUE
0020 03D1         30         ADD     A,#-47     ;SUBTRACT 47
0022 E62C         31         JNC     ORANGE     ;ERROR-NOT WITHIN RANGE
0024 03F6         32         ADD     A,#-10     ;SUBTRACT 10
0026 E62C         33         JNC     HZ60       ;JUMP TO 60HZ ROUTINE
0028 03F4         34         ADD     A,#-12     ;SUBTRACT 12
002A F62C         35         JC      ORANGE     ;ERROR-NOT WITHIN RANGE
                  36 HZ50:
                  37                            ;SET 50HZ FLAG
                  38 HZ60:
                  39                            ;SET 60HZ FLAG
                  40 ORANGE:
                  41                            ;LINE FREQUENCY ABNORMAL TRY AGAIN
                  42
                  43
                  44
                  45
```

Figure 7.

this flag, the flag being reset each test. This instruction must also be used to initialize the timer overflow flag after a RESET, as RESET does not perform this function. Total count capacity for the timer is $2^5 \times 2^8 = 8192$ or 81.9 ms at a 10 micro second cycle time. Contents of the timer are moved to the accumulator by the MOV A,T instruction without disturbing the counting process. Conversely, the MOV T,A instruction loads the timer with the contents of the accumulator. Notice that the 8-bit timer can be read from and written to. The prescaler, however, can not. It is a separate 5-bit counter which is cleared only by a STRT T command.

The timer may also be used as an event counter. After a STRT CNT command the 8022 will respond to low-to-high transitions on the Test 1 pin by incrementing the timer. Transitions can occur no faster than once each three instruction cycles (every 30 microseconds when using a 3 MHz clock)—there is no minimum frequency. In this mode the prescaler is not used. The timer will contain the number of positive transitions occuring on T1 since a STRT CNT command.

The timer and event counter functions are mutually exclusive. Counting or timing may be started (STRT CNT, STRT T) or stopped (STOP TCNT) under program control.

The T1 pin, besides being an input to the counter, can also function as a testable input, detect the zero crossing of an AC signal, and interrupt processing. These functions, as well as those of the Test 0 pin and the interrupt structure, will be discussed in the next section.

Test And Interrupt Inputs

In addition to the 24 general purpose I/O lines which comprise ports 0, 1, and 2, the 8022 has two special inputs, T0 and T1, which are testable via conditional jump instructions. These pins allow inputs to cause program branches without the necessity to load an input port into the accumulator. The instructions JT0, JNT0, JT1, JNT1 will cause program flow to be modified depending on the state of the T0 or T1 pin. For instance, JT0 will cause a jump to the specified address if the T0 pin is high (a 1 level). Conversely, JNT0 will jump if T0 is low (a 0 level). If the jump does not occur, program flow continues with the next instruction.

The Test 0 pin serves as an external interrupt input as well as a testable input. An interrupt sequence is initiated by applying a low "0" level input to the T0 pin when the external interrupt is enabled (EN I). The interrupt is level triggered and active low to allow "WIRE ORING" of several interrupt sources at the input pin. When an interrupt is detected it causes a "call to subroutine" to location 3 in program memory as soon as all other cycles of the current instruction are complete. At this time, the program counter contents are saved in the program counter stack, but the remaining status of the processor is not.

Unlike the 8048, the 8022 does not contain a program status word. Thus, when appropriate, the carry and auxiliary carry flags must be saved by the software, as must be the accumulator. The routine shown below saves the accumulator and the carry flags.

```
Instructions    Comments
MOV R6,A        ;save accumulator in register 6
CLR A           ;clear accumulator
DA A            ;convert carry flags into sixes
MOV R7,A        ;save representation of carry flags
```

The end of an interrupt service subroutine is marked by the execution of a Return from Interrupt instruction (RETI). Prior to returning from the interrupt subroutine however, the status of the accumulator and the carry flags must be restored. The following routine restores the status of the accumulator and the carry flags, which were previously saved by the above program segment.

```
Instructions    Comments
MOV A,R7        ;restore carry flags status to
ADD A,#0AAH     ;accumulator and set/clear
                ;carry flags
MOV A,R6        ;restore accumulator
RETI            ;return from interrupt
```

An interrupt or CALL to a subroutine causes the contents of the program counter to be stored in one of the eight register pairs of the Program Counter Stack. During a CALL instruction the program counter, when saved, points to the second byte of the CALL instruction (or the return address minus one). The stack contents are then incremented before being loaded into the program counter during a return (RET) from subroutine. During an interrupt the program counter, when saved, points directly to the return address. Thus, during a return (RETI) from interrupt, the stack contents are not incremented but loaded directly into the program counter. This difference makes it imperative to use only RETI's to return from interrupts, and RET's to return from subroutines.

The interrupt system is single level in that once an interrupt is detected all further interrupt requests are ignored until execution of a RETI re-enables the interrupt input logic. This sequence holds true also for an internal interrupt generated by timer overflow. If an external interrupt and an internal timer/counter generated interrupt are detected at the same time, the external source will be recognized first, if enabled. The timer/counter interrupt will be recognized, if enabled, after the return (RETI) from the external interrupt. Timer/counter generated internal interrupts and T0 generated external interrupts have separate vector locations. The external interrupt will vector to location 3, whereas an internal interrupt will vector to location 7.

If needed, a second external interrupt can be created by enabling the timer/counter interrupt (EN TCNTI), loading FFH into the counter (one less than terminal count) and enabling the event counter mode (STRT CNT). A low-to-high transition on the T1 input will then cause an interrupt vector to location 7.

Zero Cross Detect

The Test 1 pin, in addition to being a testable input and a counter input, also serves one other important function. It can be used to detect the zero crossing point of slow moving AC signals. Execution of the STRT CNT instruction puts the T1 pin in the counter input mode by connecting T1 to the counter and enabling the counter. Subsequent low-to-high transitions on T1 will cause the counter to increment. Note that this operation differs from the rest of the MCS-48 devices, which increment the counter on high-to-low transitions. This change was made on the 8022 to take advantage of the accuracy of the rising edge detection on the zero cross circuitry.

When driven directly, this pin responds as a normal digital input. To utilize the zero cross detection mode, an AC signal of approximately 1-3 VAC p-p magnitude and a maximum frequency of 1kHz is coupled through an external capacitor (1 microfarad) to the T1 pin. The internal digital state is sensed as a zero until the rising edge crosses the DC average level, when it becomes a one. This is accomplished by the self-biasing high gain amplifier which is included in the T1 input. This circuit biases the T1 input exactly at its switching point, such that a small change will cause a digital transition to occur. This digital transition takes place within 5 degrees of the zero point.

The digital value of T1 remains a one until the falling edge of the AC input drops approximately 100mV below the switching point of the rising edge (100mV below the zero point, if the digital transition occured exactly at the zero point). The 100 mV offset is created by hysterisis and eliminates chattering of the internal signal caused by external noise.

The accuracy of the zero crossing will be a function of the capacitor used (see Fig. 10). On critical systems the capacitor can be adjusted to improve overall accuracy.

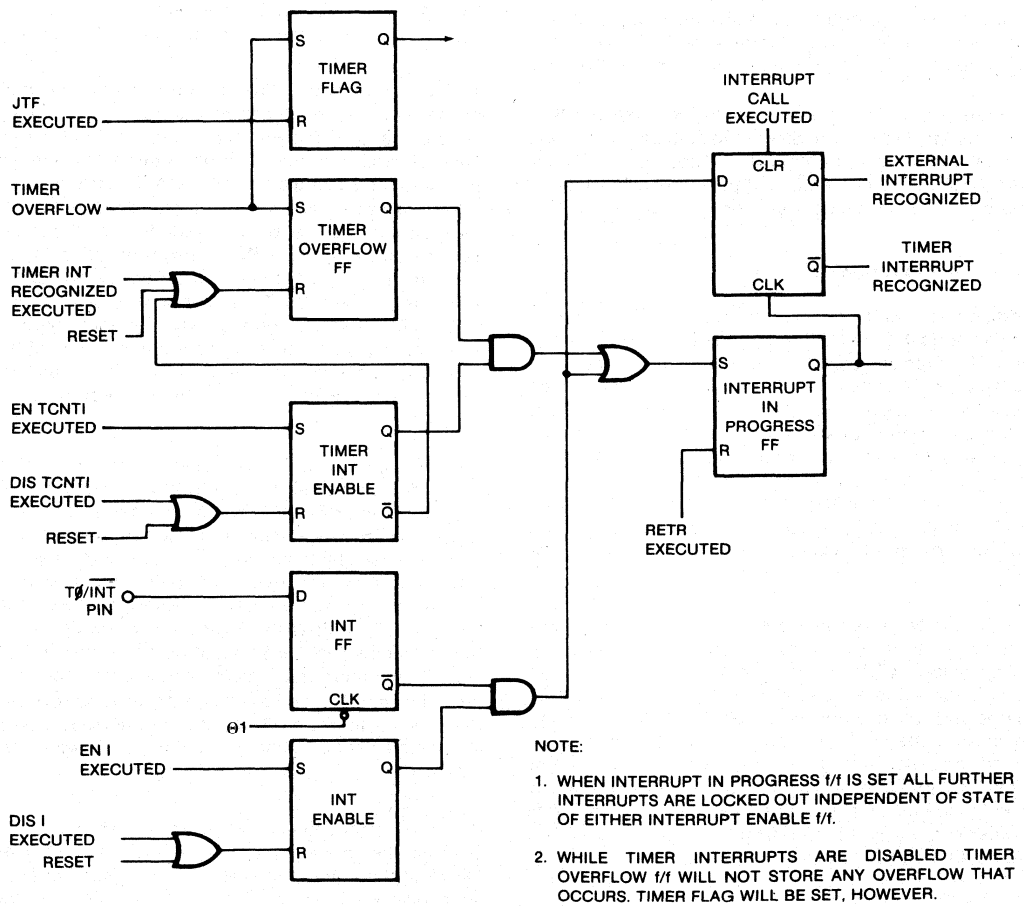


Figure 8. Interrupt Logic

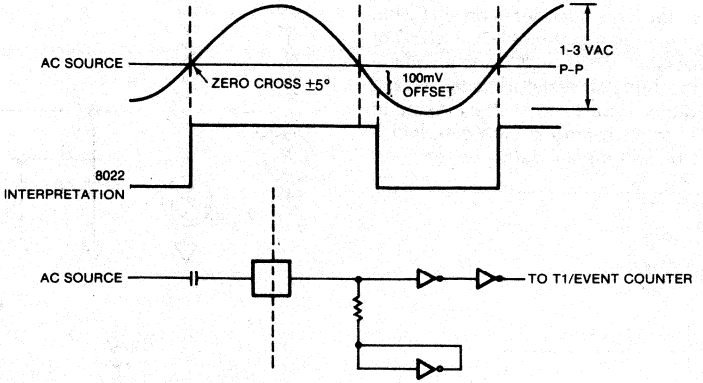


Figure 9. Zero Cross Detection

The phase angle at the T1 input can be expressed as

$$\Theta = \arctan \frac{X_C}{R}$$

where $X_C = \frac{1}{2\pi fC}$

$R = 150K\Omega$ (see fig. 10)

Solving the equation using the recommended one microfarad capacitor and 60Hz

$$X_C = \frac{1}{2\pi\ (60)\ (1\mu f)}$$

$$= 2652.6$$

$$\Theta = \arctan \frac{2652.6}{150K\Omega}$$

$$= -1.01^0$$

shows the voltage at the pin slightly leading the true AC voltage. Internally the circuit adds up to another five degrees before the processor can detect that a zero crossing occured. Software can also add several degrees before outputing a signal. To compensate for all of this delay, a smaller capacitor could be chosen to give a -5 degree shift in hardware before the processor.

The zero cross detection capability allows the user to make the 50/60 Hz power signal the basis for his system timing. All timing routines, including time-of-day, can be implemented using the zero cross detection capability of T1 and its conditional jump instructions. In addition, the zero cross detection feature can be used in conjunction with the timer interrupt, as discussed earlier, to interrupt processing at the zero voltage point. This enables the user to control voltage phase sensitive devices such as triacs and SCRs, and to use the 8022 in applications such as shaft speed and angle measurement.

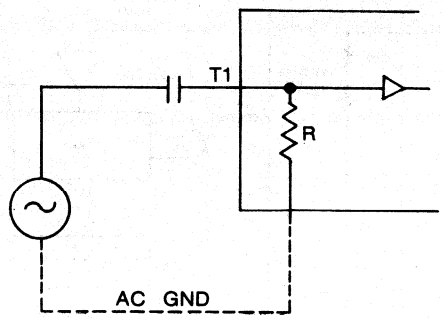


Figure 10. AC Equivalent of Zero Cross Input

Analog To Digital Converter

The T1 zero cross function is only one of the linear functions incorporated into the 8022 architecture. The most noted linear function is that of a complete analog to digital converter.

The analog to digital converter is a complete successive approximation converter with two multiplexed input channels. Either channel is selected by software with the SEL AN0 or SEL AN1 instruction. These instructions also restart the conversion sequences. A valid digital value can be read with the RAD (read A/D) instruction during the fourth instruction cycle following a select instruction. Conversions occur continuously, and RAD may be executed at any time with confidence that the sample is no more than 40 microseconds old.

The converter hardware has three parts as shown in Figure 11, a series string of resistors, a voltage comparator, and successive approximation logic. A series string of 256 matched resistors divides the voltage between AVss and V_{AREF} (the reference pin) into 256 voltage steps. This configuration gives the converter its inherent monotonicity.

The voltage tap on the series resistor string is selected by

the resistor ladder decoder. This decoder is driven by the 8-bit successive approximation register (SAR). Each bit of the SAR is set in succession MSB to LSB and a voltage comparison between the selected resistor ladder voltage and the analog input voltage is performed after the setting of each bit. The result of each comparison determines whether the particular bit will remain set or be reset. All

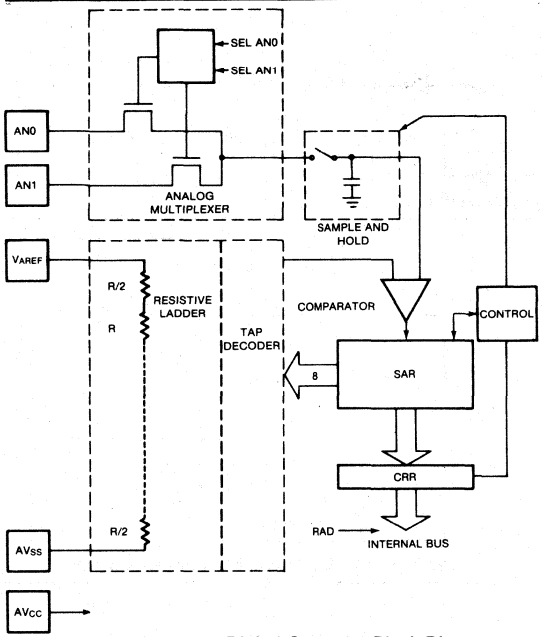


Figure 11. Analog to Digital Converter Block Diagram

comparisons are performed automatically by the on-chip A/D hardware. At the end of eight comparisons the SAR contains a valid digital representation of the analog voltage. This result is then latched into the conversion result registor (CRR). The RAD instruction can then load the conversion result from the CRR to the accumulator.

To insure maximum accuracy from the A/D converter, separate power supply pins (Avcc and Avss) and a substrate pin (SUBST) have been provided. Unless there is excessive noise on the digital power supply, both Vcc and Avcc can be tied together and still maintain maximum accuracy. Figure 12 shows a typical analog configuration for sensing temperature in two thermistors. The substrate has both low frequency and high frequency bypass for noise immunity. The power supply pins (Vcc, Avcc) are bypassed with a .01 microfarad capacitor close to the chip. All other analog signals are bypassed with .001 microfarad capacitors for added noise rejection. (See also Software Noise Rejection)

As figure 11 shows, V_{AREF} is connected to the top of the resistive ladder. When the selected analog channel is equal to or greater than V_{AREF} the conversion result will equal 255 decimal (FF hexadecimal). The V_{AREF} voltage can be generated in a number of ways depending on the

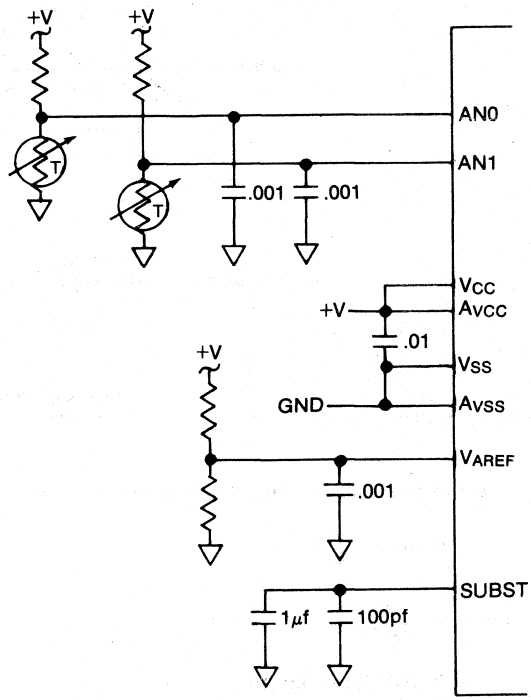


Figure 12. Typical Analog Schematic

system. It could be connected directly to Vcc giving a A/D range of GND to Vcc, or a simple resistor divider could be used to balance the reference voltage with the analog signals as in Figure 12. In calculating the impedance of the divider, the ladder impedance must be considered (see Figure 13). The total impedance of the ladder ranges from approximately 15K to 20K. This includes part to part differences and variance as a function of temperature. The resistor impedance should be chosen such that the 15K ohm parallel resistance is a small percentage of the divider impedance.

Input impedance of the converter can also be an important

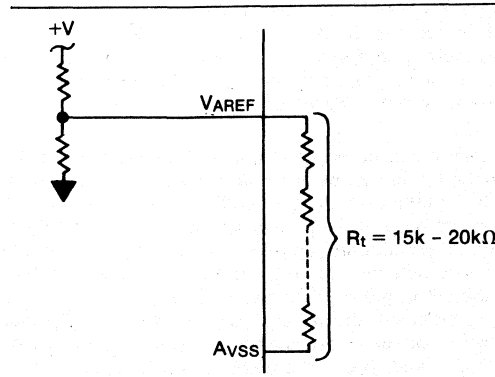


Figure 13. Ladder Impedance

factor. Figure 14 is an equivalent circuit of an analog input. Capacitance C1 is package capacitance which may range from 1pf to 3pf. Capacitance C2 is the sample and hold capacitance of 1.2pf to 1.4pf. This capacitance is only connected into the circuit by the sample and hold switch. The switch is closed for 0.3 tcy every four instruction cycles. Resistor R1 is package leakage which is approximately 2.5-5.0M ohms.

Software Noise Rejection

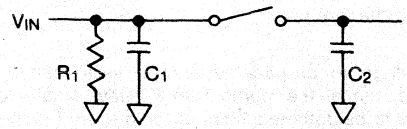


Figure 14. Analog Input Equivalent Circuit

Noise can be a problem in any system. Capacitors can be used to filter the noise but may not filter all of it. Capacitors also add cost to the system but can be eliminated by software filtering. One technique is simply to average two readings:

$$\frac{V_{IN1} + V_{IN2}}{2} = V_{OUT}$$

or keep a running average by averaging each reading with the previous average:

```
SEL AN0      ;Start conversion
MOV R0,#30   ;Point to storage location
RAD          ;Read current A/D sample
ADD A,@R0    ;Add current sample to previous average
RRC A        ;Divide by two
MOV @R0,A    ;Store new average
```

This method will eliminate small fluctuations in the input voltage and reduce the effect of large fluctuations. Often, however, noise may be more severe. Excessive noise may require averaging of many readings taken over a short period of time.

$$\frac{V_{IN1} + V_{IN2} + \ldots + V_{IN16}}{16} = V_{OUT}$$

Figure 15 lists the software required to average 16 successive A/D samples, as the above equation suggests. In such averaging, it is necessary to select the appropriate channel only once. Thereafter, a new conversion result is available every four instruction cycles.

Still another type of filtering is "exponential averaging." Similar to the running average method, current readings are averaged with the previous average.

$$\frac{V_{IN} - Voldavg}{K} + Voldavg = Vavg$$

Where Vavg = current average
Voldavg = previous average
Vin = current reading
K = constant

This method has the advantage of large signal to noise ratios, but has slower dynamic response. In many systems, especially those involving temperature measurement, dynamic response is not a problem. Signal noise will be of a much higher frequency than any change in temperature. The constant, K, can be chosen to yield any desired signal to noise ratio. The larger the constant, the higher the ratio. The lower the constant, the higher the dynamic response.

To increase the effectiveness in reducing line generated noise, any of the above methods should be synchronized to the line frequency. As previously discussed, an interrupt can be generated when the 50Hz or 60Hz line frequency crosses AC zero. The A/D filtering routine should be part of the interrupt routine. Reading of the A/D will then occur at the same point of each line cycle, thus ignoring any line generated fluctuations in the analog inputs.

```
LOC  OBJ        LINE        SOURCE STATEMENT

                 47 ;
                 48 ;       AVERAGE 16 A/D READINGS
                 49 ;       ======================
                 50 ;
                 51 AVG16:
002C BC00        52          MOV     R4,#00   ;CLEAR TEMP. MSB RESULT REGISTER
002E B81A        53          MOV     R0,#26   ;SET UP POINTER
0030 B000        54          MOV     @R0,#00  ;CLEAR RESULT REGISTER
0032 85          55          SEL     AN0      ;SELECT AND START CONVERSION
0033 BA10        56          MOV     R2,#16   ;16 READINGS
                 57 LOOP:
0035 80          58          RAD              ;READ RESULT
0036 60          59          ADD     A,@R0    ;LSB
0037 A0          60          MOV     @R0,A    ;SAVE NEW LSB
0038 27          61          CLR     A
0039 7C          62          ADDC    A,R4     ;ADD CARRY TO MSB
003A AC          63          MOV     R4,A     ;SAVE NEW MSB
003B EA35        64          DJNZ    R2,LOOP  ;NEXT READING
003D 47          65          SWAP    A        ;MSB INTO MSN
003E 20          66          XCH     A,@R0
003F 47          67          SWAP    A        ;DIVIDE BY 16
0040 30          68          XCHD    A,@R0    ;LOCATION 26 NOW CONTAINS
                 69                           ;THE AVERAGE
                 70
                 71
                 72
```

Figure 15.

Port 0 Comparator Inputs

Intel, in its commitment to add analog features to microcomputers, did not stop with A/D conversion and zero cross detection. Also added to the 8022 were eight comparators for easing the interface to non-digital inputs.

Port 0 has been modified from the standard quasi-bidirectional structure to allow an optional open drain configuration with comparator inputs. The low impedance pullup device has been eliminated and the high impedance pullup is optional. Thus, the user can choose via a mask programmable selection each line of Port 0 to be either quasi-bidirectional with a high impedance or true open-drain. The open drain configuration allows the line to sink current through the low impedance pulldown device or to float in the high output state. More importantly, the open drain configuration makes Port 0 very easy to drive when it is used as inputs. The input circuitry for each line of Port 0 includes a voltage comparator which amplifies the voltage difference between the input port line and the Port 0 threshold reference pin (V_{TH}). The voltage gain of the comparator is sufficient to sense a 100mV input differential within the range Vss to Vcc/2.

If V_{TH} is allowed to float, it will bias itself to the digital switch point of the other ports, and Port 0 behaves as a set of normal digital inputs. However, by biasing V_{TH}, the switch point can be both tightly controlled and adjusted. Common uses for this would include unusual logic level inputs as from a diode isolated keyboard, analog channel expansion, and direct capacitive touchpanel interface. The comparator action is automatic and the port is read just as any other port.

A typical use for Port 0 is in the interfacing with capacitive touch panels on microwave ovens and other new appliances. A touch-panel switch consists of two capacitors in series. One lead is attached to a high voltage buffer (10 to 30 volts). The other is attached to the Port 0 sense input. As a finger touches the common point, the drive signal is shunted by body capacitance, attenuating the signal reaching the input.

Low-voltage touch-panel operation (less than 30V) is possible, since the comparators allow small voltage changes to be detected. Most of the present touch-panel designs require a 50–100V drive on the touch panel.

Capacitive touch panels can be multiplexed in the same manner as mechanical keyboards (Fig. 16). The vacuum flourescent display and the touch panel drivers are integrated to optimize hardware through shared high voltage buffers.

Figure 17 lists the software necessary to refresh the display and scan the touch-panel matrix. This routine could be adapted to serve as part of a timer/interrupt

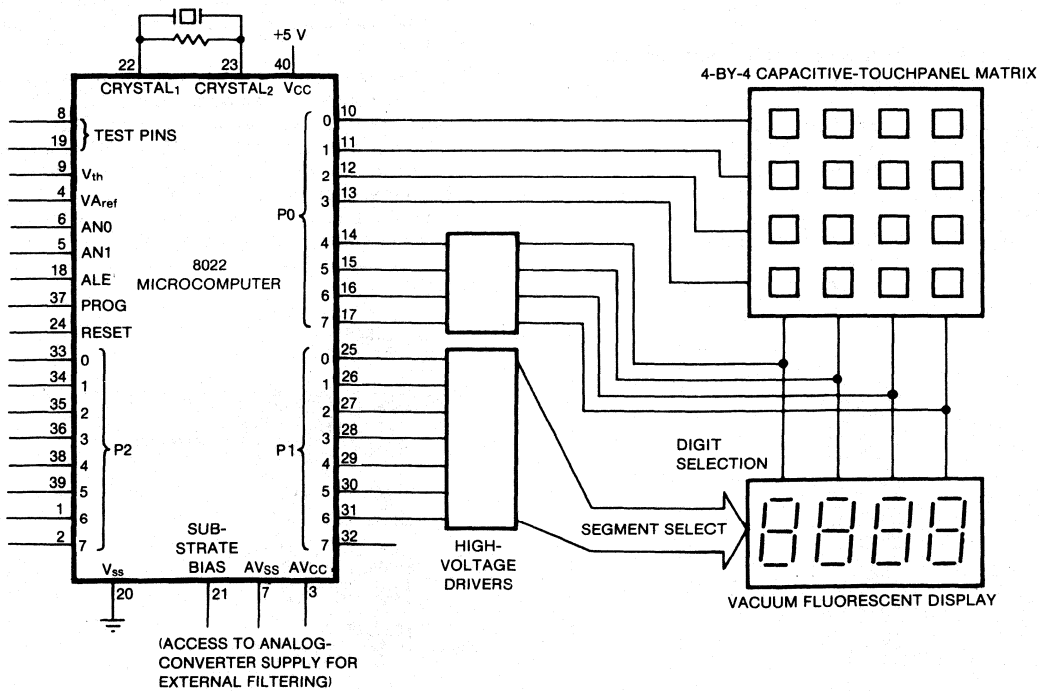


Figure 16. Typical Keyboard/Display Schematic

```
LOC  OBJ          LINE        SOURCE STATEMENT

                    74 ;
                    75 ;        KEYBOARD DISPLAY ROUTINE
                    76 ;        ========================
                    77 ;
003C                78 D4       EQU     3CH     ;MSD OF DISPLAY
                    79 KEYDIS:
0041 27             80          CLR     A
0042 39             81          OUTL    P1,A    ;TURN OFF SEGMENT DRIVERS
0043 90             82          OUTL    P0,A    ;TURN OFF DIGIT DRIVERS AND
                    83                          ;PANEL STROBES
                    84                          ;INITIALIZE SENSE INPUTS TO GND
0044 230F           85          MOV     A,#0FH
0046 90             86          OUTL    P0,A    ;FLOAT SENSE INPUTS
0047 4B             87          ORL     A,R3    ;NEW STROBE POSITION
0048 90             88          OUTL    P0,A    ;TURN ON STROBE
0049 08             89          IN      A,P0    ;READ SENSE INPUTS
004A AC             90          MOV     R4,A    ;SAVE SENSE INPUTS
004B B83B           91          MOV     R0,#D4-1        ;RAM LOCATION OF MSB OF 7-SEG PATTERN
004D FB             92          MOV     A,R3    ;STROBE POSITION INTO A
004E 90             93          OUTL    P0,A    ;GND SENSE INPUTS
                    94 LOOP1:
004F F7             95          RLC     A       ;ROTATE DIGIT STROBE INTO CARRY
0050 18             96          INC     R0      ;NEXT DIGIT LOCATION
0051 E64F           97          JNC     LOOP1   ;LOOP UNTIL CARRY
0053 F0             98          MOV     A,@R0   ;RETRIEVE PATTERN FROM RAM
0054 39             99          OUTL    P1,A    ;OUTPUT NEW PATTERN
                   100
                   101
```

Figure 17.

scheme that would generate an interrupt at precise intervals for a flicker-free display. Another portion of the software would check for any touched input pads, test for valid entry, and enter key-depression codes into the main program.

Correcting Pad Imbalance

A common problem with capacitive touch panels is their imbalance. Layout, process, aging, and surface impurities all cause the capacitance to vary from touch pad to touch pad, resulting in a family of curves (Fig. 18) of voltage levels from each column of touch pads reaching the sense inputs. As the curves show, if threshold voltage Vth1 alone were used, one column would always appear touched; if Vth2 were used exclusively three of the columns would never appear touched.

To compensate for such varying capacitance levels, the on-chip analog input circuit may be used to allow multiple input voltage levels. Figure 19 depicts the 8022 version of such a circuit. AN0 and V_{TH}, are tied together with a capacitor to line 0 of Port 0. The pull-up resistor and capacitor are used on line 0 to provide an RC timing network connected to ANO, VTH, and P00. The remaining seven lines of Port 0 are the sense lines for the touch panel and use the open drain output option.

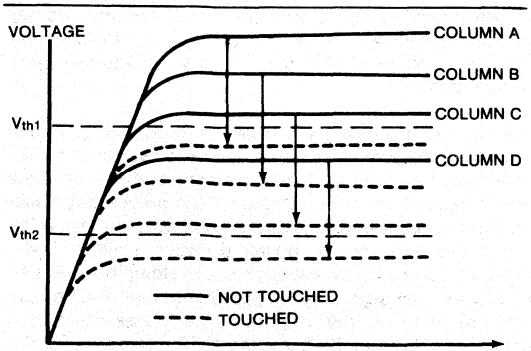


Figure 18.

Handling the different voltage levels would begin with initializing the data by plotting the family of curves with the AN0 input. This is done by writing a 0 to P00 (grounding P00) which initializes V_{TH} to 0v. A logic 1 is then written to P00, which begins to pull the RC network toward 5 v. As V_{TH} ramps upward, the sense lines are monitored for input changes, which will go from 1 to 0 as the not-touched voltage curve for each input is intersected. As the changes occur, the A/D value for each input is intersected. As the changes occur, the A/D value for each sense line can be read and stored. Thus the threshold reference voltage for each sense line can be determined by establishing the not touched voltage levels and by placing the threshold reference voltage below this level. As each row of the keyboard is scanned, the RC network is initialized to 0 v and ramps upward, varying the V_{TH} level. The A/D converter monitors this level looking for the calculated threshold points. As

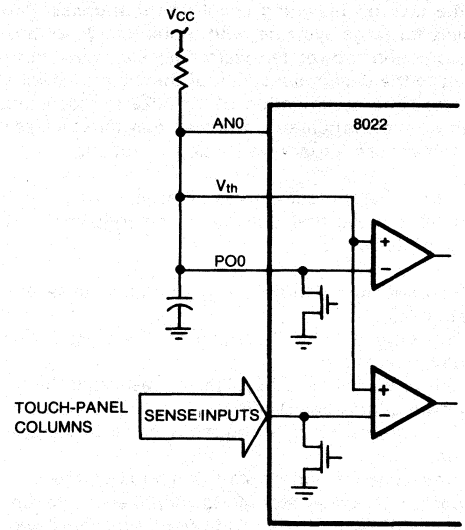


Figure 19.

the points are intersected, the corresponding sense inputs are read, with a 0 indicating a touched input and a 1 indicating a not-touched input. Thus, a multiplexed capacitive touch panel can be scanned and balanced without adding external components to the inputs.

For systems requiring more than two analog inputs to the 8022, Port 0 comparator inputs may be reconfigured to permit formation of pseudo-analog inputs from variable threshold digital inputs. The hardware configuration can be identical to that of Fig. 19. In this scheme, sense inputs act as additional analog inputs of less accuracy than AN0 and AN1 (about 6 bits).

The sequence for implementing the extension is essentially the same found in the variable-threshold touch panel. As V_{TH} ramps upward, a Port 0 bit is monitored for a change from 1 to 0. At the change, AN0 is read corresponding to the value of the analog input into Port 0 (with some error due to the time lag, which can be subtracted). This configuration can be utilized when it is possible to trade off accuracy for such cost improvements: one analog input with 8-bit accuracy and seven analog inputs with 6-bit accuracy. Such may be the case in a range controller that monitors temperature in two ovens, a meat probe, and two of the four burners, all to an accuracy within 10°F.

Application Ideas

This section will discuss some possible applications of the 8022. These applications are discussed in general terms and are believed to be feasible applications of the 8022. None of these applications, however, have been built and checked out.

Power Supply Controller

The three terminal voltage regulator, with its built-in current limiting and overload protection, has vastly simplified the task of designing small power supplies. Power supplies for large systems, with requirements for brown out protection, power fail warnings, etc., have not yet yielded to the design simplicity of the integrated voltage regulator. The combination of an 8022 microcomputer and these same regulators, however, may make it feasible to simplify these larger power supply systems.

There are several requirements of larger power supplies which have to be met outside of the regulation itself. Typical of these are:

1. Sequencing the turn on and shut down of several supplies.
2. Providing an early warning to the system that power is failing.
3. The ability to hold the system in a reset state during power supply sequencing.
4. Generation of a line frequency clock to the system.
5. Provisions for remote start up and shut down.
6. Sufficient energy storage to keep the system running long enough to provide an orderly shut down.
7. High efficiencies to minimize power requirements and heat dissipation.

These requirements can be met by a combination of raw DC supply, multiple three-terminal regulators, and an 8022 microcomputer. Figure 20 shows a raw supply which is capable of generating DC voltages suitable for regulation to five, plus twelve, and minus twelve voltages. (These are arbitrary, but common voltages). In addition, a separate winding is provided which generates a five-volt supply which will be used to supply power to the 8022 itself. The normal rectifiers in the RAW5 and RAW12 supplies are replaced by silicon controlled rectifiers which will be phase angle controlled by the 8022.

Figure 21 shows the connections to the 8022. The RAW5 and RAW12 supplies are applied to simple voltage dividers which feed the analog inputs of the 8022. The signal

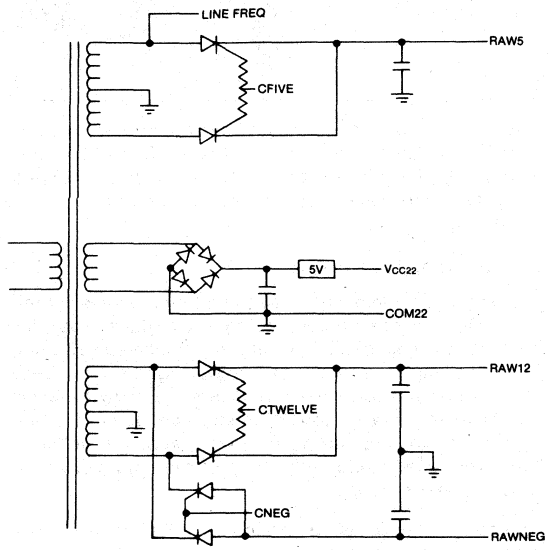


Figure 20.

LINEFREQ is taken from a convenient winding of a transformer, divided down, and applied to the zero cross input of the 8022. A strap is provided to configure the unit for 50/60 Hz operation. In addition to being connected to the basic power supply, the 8022 is also connected to the system receiving the power. The on/off switch becomes an input to the 8022. The 8022 provides outputs for a 10Hz interrupt, a power fail interrupt, cold/warm indicator and system reset.

The 8022 can perform many of the functions normally done by hardware sequencers in the power supply. On power-up, it can hold off the three main supplies until the main supply is firmly established. This prevents the system from responding to short power restorations which frequently occur during power outages. Having determined that it is safe to power up the system, the 8022 can assert the reset signal and the cold start signal. The cold start indication tells the system that power was interrupted at the mains rather than by the OFF switch—a useful function if any amount of battery backed up RAM exists in the system. Having set up these signals, the 8022 waits for

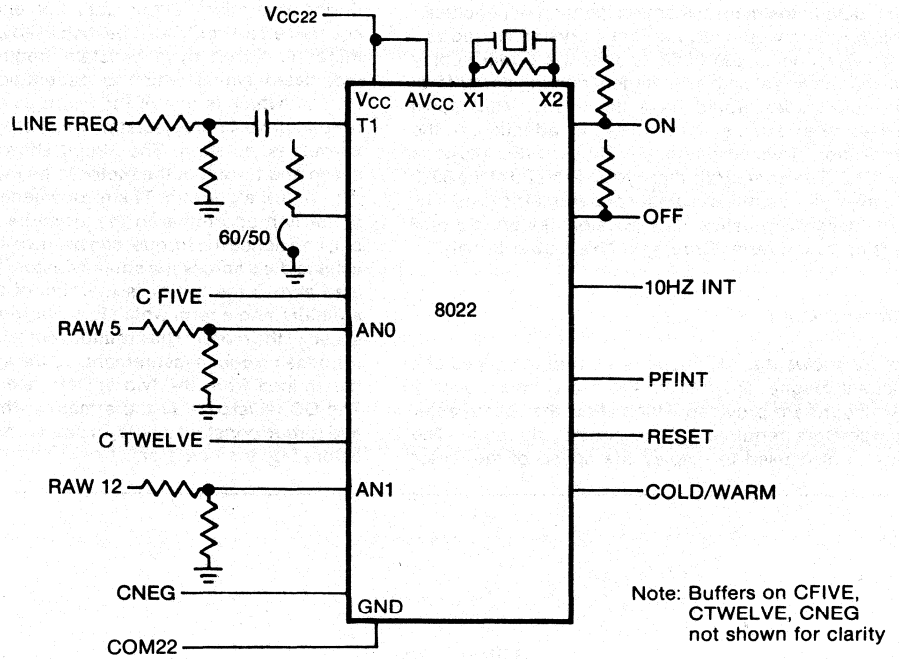


Figure 21.

a zero crossing (to minimize inrush) and then turns on the SCRs for the three supplies one at a time (again to minimize inrush). Any sequencing of the three supplies that is required by the system can also be allowed for. After some programmable time delay, the reset signal can be released and the system allowed to start operation.

During normal operation the 8022 can monitor the two major raw supplies and use phase angle control of the SCRs to regulate them. The regulation would be used to ensure that the three terminal regulators had minimum input voltage requirements met under all line voltage variations while at the same time minimizing the voltage drop across them. This increases the efficiency of the power supply and allows it to be capable of handling brown outs without dissipating excessive power in the regulators.

The line frequency input is used not only for the basis for the phase angle control, but also for two other functions; power fail detect and generation of the 10 Hz interrupt. The 10Hz interrupt can be generated by simply dividing the power line frequency by 5 for 50 Hz and 6 for 60 Hz operation. Performing this division in the power supply itself allows the system to be run on 50 or 60 cycle power with no change external to the power supply. In some situations it should even be possible to have the power supply adapt to either of these inputs by measuring the period of the incoming power on startup (see section "Which One?"). This would be an easy function to incorporate in the software and would require no additional hardware since provision is already made for zero cross detect.

Power fail detection can be done by running the timer while waiting for the line to zero cross. If an excessive time elapses it can be assumed that the power has failed and the power fail interrupt asserted. Note that this will detect total power failure but not a dip in the line voltage below the specifications of the power supply. This condition can be detected by keeping track of the phase angle that is required to maintain the RAW supplies at the proper level. If the SCR's have to be turned on for too high a portion of the total line cycle it is an indication of a brown-out condition and the powerfail interrupt should be generated. Whenever the powerfail interrupt is generated the 8022 should turn on the SCRs continuously to ensure maximum possible energy storage in the filter capacitors. After generation of the powerfail interrupt, the 8022 can again delay (depending, of course, on the energy storage of the power supply) and then assert reset. Once reset is asserted the SCRs are turned off, and left off, until the supplies have dropped down to a point which guarantees that any reset circuitry residing outside of the power supply will see a full power transition when power is reapplied. If the power is shut down by the 8022 in response to the on/off switch, the sequence would be similar except that the cold/warm start signal would indicate a warm start.

The above discussion should make it clear that the 8022 would make the task of designing a power supply system far easier, particularly for those designers more familiar with digital than analog design. If, in addition, the 8022 supply were put on a battery back-up, it would be possible to add many features to the system at virtually zero cost. The 8022 could be programmed to become the system clock and send, perhaps in serial ASCII, the time of day

and the date to the main system on demand or periodically. This function would require that a crystal be used as a timing reference to the 8022 so that the power supply could still track real time even if the incoming power fails. Other possibilities would have the system shut down unless some external event required its attention, or the incorporation of system diagnostic checks within the code of the 8022. The comparator inputs on PORT 0 of the 8022 would even allow some capability of parametric testing as part of these diagnostics. The possibilities bring a new dimension to the term "Programmable Power Supply".

DC Motor Control

Figure 22 shows the 8022 used to control the speed of a permanent magnet DC motor. A seven segment display and keyboard are provided which allow the user to enter the parameters required by the control algorithm. The display is also used to display the speed of the motor during operation. Other data (for example root mean squared error) could also be displayed upon demand. The motor is driven by a constant frequency pulse width modulated signal which is generated programatically. Port 11 (which is one of the two high current outputs) is used to drive a photoisolator which provides level shifting as well as isolation. The circuit shown allows both the speed and torque of the motor to be measured for use by the control algorithm. The torque generated by a PM DC motor is proportional to the armature current. This current, and hence the torque, can be measured by reading the voltage drop across the shunt resistor. The voltage generated across the motor is the sum of the IR drop in the armature and a term which is proportional to the angular speed of the motor. The armature current is already known from the torque measurement, so the speed can easily be determined from the two analog measurements shown. The DC resistance of the armature, the speed constant, and torque constant would, of course, have to be known or entered by the operator.

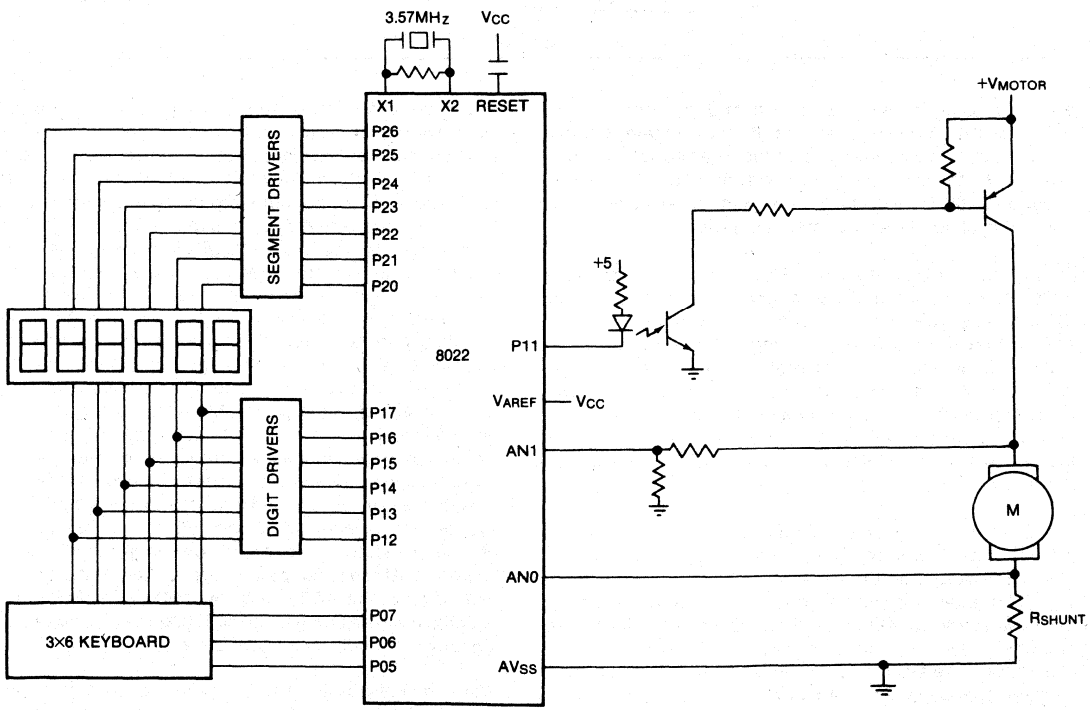


Figure 22. DC Motor Control

Automotive Dashboard

Figure 23 shows an automotive dashboard controlled by the 8022. Provisions are made to measure the oil pressure, water temperature, and vehicle speed. A connection to the distributor points allows the engine RPM and point dwell to be measured. Outputs are provided to control the ignition and starter (allowing the ignition switch to be eliminated in favor of a combination lock). Drive to a vacuum servo is also possible to allow cruise control to be cheaply implemented. The display can be used for a speedometer, tachometer, oil pressure guage, or water temperature guage depending on the current desire of the driver. There are several uncommitted I/O pins which could be used to implement functions such as intermittent action windshield wipers or delayed action light circuits.

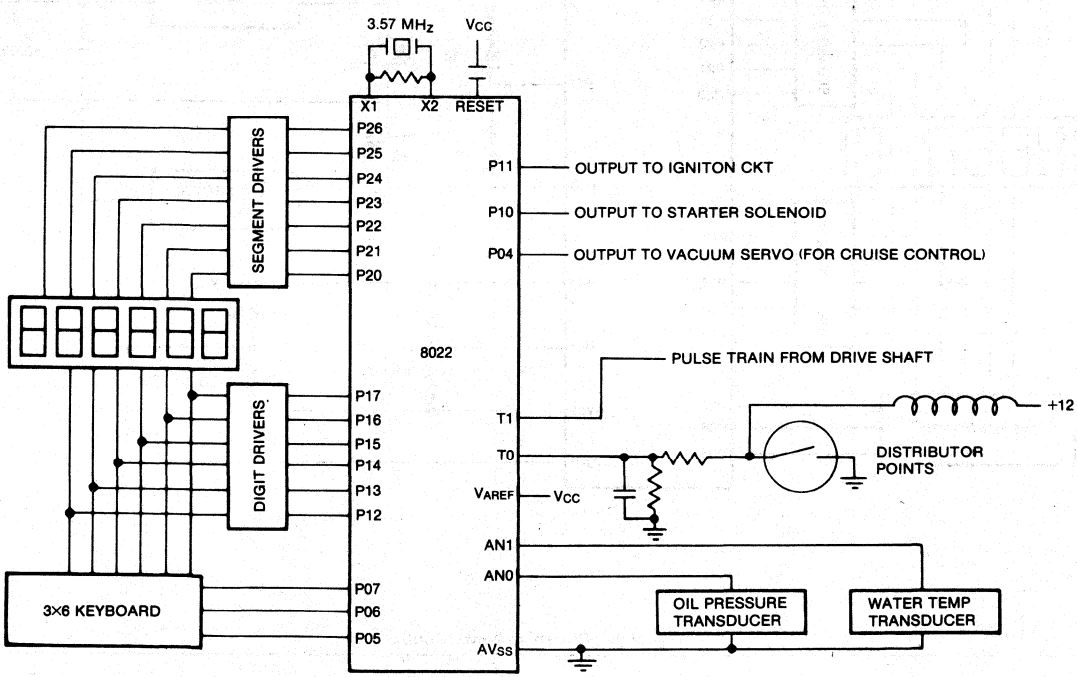


Figure 23. Automotive Dashboard

A darkroom timer based on the 8022 is shown in Figure 24. In addition to the keyboard and display this diagram incorporates drive to two TRIACs, an input to monitor the line frequency crossings, and two analog measurements. The analog inputs are used to monitor and display the temperature of the chemical bath and the light output of the enlarger, both of which can be controlled by the microcomputer. The 8022 could be used to run several timers concurrently while also maintaining the temperature of the chemical bath at the required level. Several uncommited I/O pins are available for additional functions.

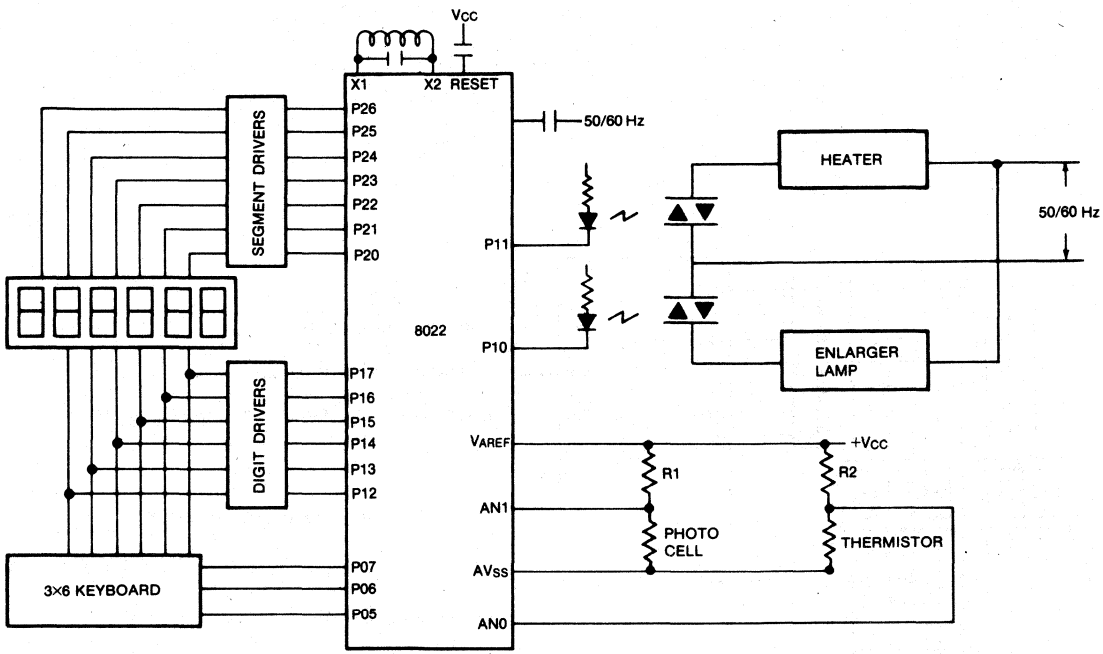


Figure 24. Darkroom Timer/Control

Conclusions

This application note has introduced the reader to the Intel 8022 microcomputer. It has described the main features of the 8022 and discussed some of the design considerations encountered in designing with the 8022.

The reader has also been exposed to several possible applications which show the versatility and cost effectiveness of a microcomputer with on-board analog features.

Energy Savings in an Induction Motor Using the 8022 Microcontroller

Contents

INTRODUCTION 5-278

THEORY OF OPERATION 5-278

THE 8022 5-280

HARDWARE DESCRIPTION 5-281

SOFTWARE DESCRIPTION 5-281

CONCLUSION 5-283

PROGRAM LISTING 5-285

Introduction

A recent NASA invention enables considerable energy savings when using the common induction electric motor. It can be used with existing motors, since it requires no modification to the motor. Typical energy savings of 10 to 60% can be realized, depending on the amount of motor loading present. This invention is the direct result of an analysis, by NASA, of Solar Heating and Cooling Systems to reduce the power consumed by pump and fan motors used in these systems. It is applicable to both single phase and 3 phase motors.

Since the induction motor is widely used in many of the same systems that can take advantage of the 8022 microcontroller, this invention can provide a significant benefit to many commercial and consumer products. Examples include the following:

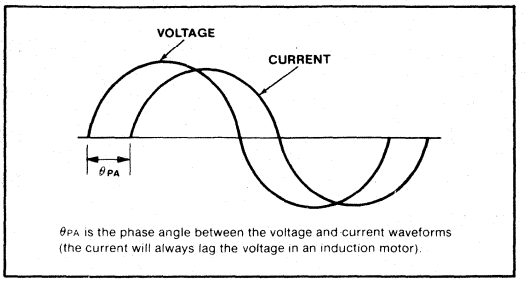
—space heating and cooling systems
—heat pumps
—solar collector controllers
—liquid or chemical process control
—large industrial motor control
—refrigeration units
—swimming pool controllers
—washing machines
—dishwashers.

This application note will explain how this invention can be implemented with an 8022 microcontroller with a minimum amount of external hardware. The note is organized into the following sections:

1. Power Factor Controller theory of operation.
2. Description of the 8022 microcontroller in this application.
3. Hardware description.
4. Software description and listing.
5. Conclusion.

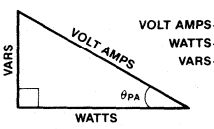
Theory of Operation

The concept of the Power Factor Controller (PFC), conceived and developed by NASA aerospace engineer Frank Nola,[1] is to reduce the voltage applied to the motor when it is partially loaded, resulting in significant energy savings. At full load, the induction motor must have a high flux density in order to perform adequately. Under this condition the motor is running at peak efficiency with a power factor of about 0.8. At less than full load, however, this high flux density is still supported by the high current set up in the field coils. The resulting power factor can drop to as low as 0.1 (FIG. 2). The losses associated with this high current, under the lightly loaded condition, is primarily in the form of heat. This is a loss which can be paid for twice if the motor is in a refrigerated or otherwise cooled system.



θ_{PA} is the phase angle between the voltage and current waveforms (the current will always lag the voltage in an induction motor).

Figure 1.



VOLT AMPS—APPARANT POWER—I_{EFF} V_{EFF}
WATTS— AVERAGE POWER —I_{EFF} V_{EFF} COS θ_{PA}
VARS— REACTIVE POWER —I_{EFF} V_{EFF} SIN θ_{PA}

The Power Factor is equal to the cosine of the phase angle. It represents the ratio of the total power required to be supplied by the power company to the power recorded by a watt meter.

$$\text{Power Factor} = \frac{\text{Average Power}}{\text{Apparent Power}} = \text{Cos}\,\theta_{PA}$$

The Power Factor ranges from 0 to 1. For a purely resistive load, the voltage and current are in phase, θ_{PA} is zero, and the P.F. is unity, the Apparent Power and Average Power are equal.

For a purely reactive load, that is, one containing no resistance, the phase difference between the voltage and current is either plus or minus 90° (purely inductive being plus 90°, purely capacitive being minus 90°), the P.F. is therefore zero, resulting in the Average Power being zero.

Between these two extreme cases, there are the general networks for which the P.F. can range from zero to unity. Therefore, for a constant magnitude of the current and voltage, the Average Power (Watts) can range from zero to a maximum value equal to the Apparent Power, depending on the phase angle.

Figure 2. Complex Power and the Power Triangle

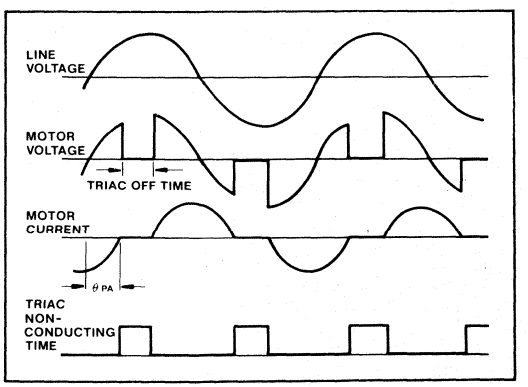


Figure 3.

[1] NASA Tech Brief MFS-23280, Power Factor Controller. Copies may be obtained by writing to: Director, Technology Utilization Office, Marshall Space Flight Center, AL, 35812. The patent for the PFC is owned by the U.S. Government. Licenses for commercial development are available at no charge. Contact: Patent Counsel, Marshall Space Flight Center, AL, 35812.

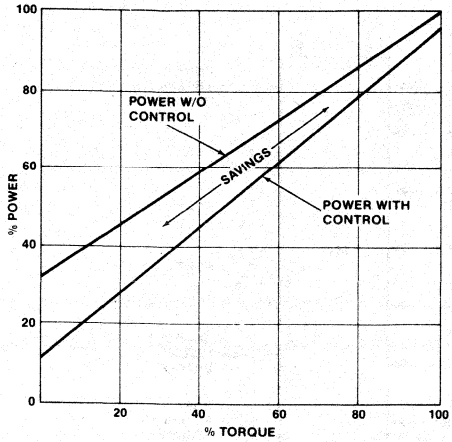


Figure 4. Typical Power Savings [2] for Single Phase Motor

In this partial load condition, the motor would produce the same torque at essentially the same speed even if much weaker magnetic forces were set up, by decreasing the current which produces these fields. The electric motor on its own, can't recognize this condition, however, and will continue to draw near the high current used under full load, even when operating under no load.

The principle of operation of the PFC is to measure the shift in phase angle between motor voltage and current

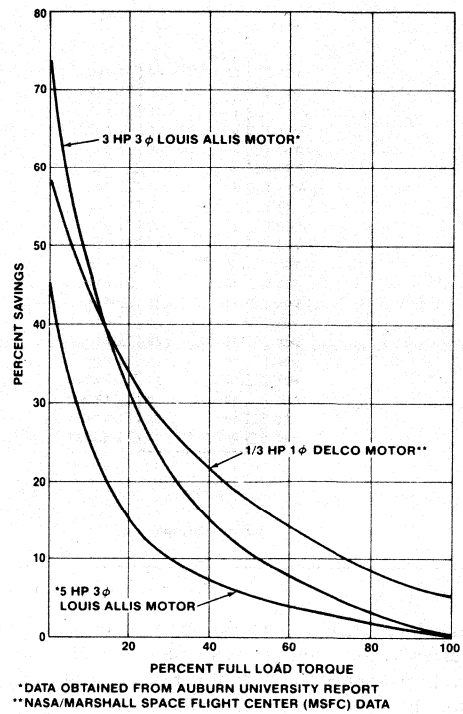


Figure 5. Power Factor Controller Percent Savings Vs. Torque for Various Motors [3]

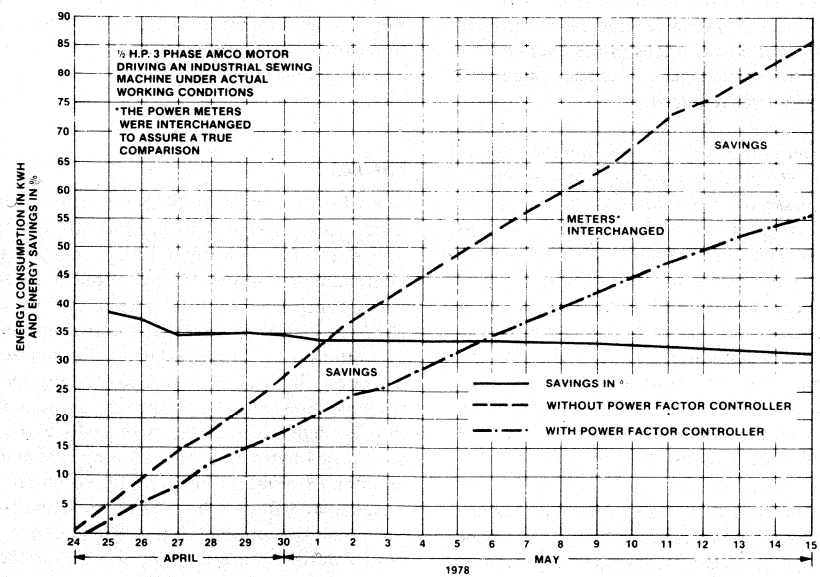


Figure 6. Energy Consumption as a Function of Time [4]

[2] NASA TECH BRIEF MFS-23280
[3] Ibid.
[4] Ibid.

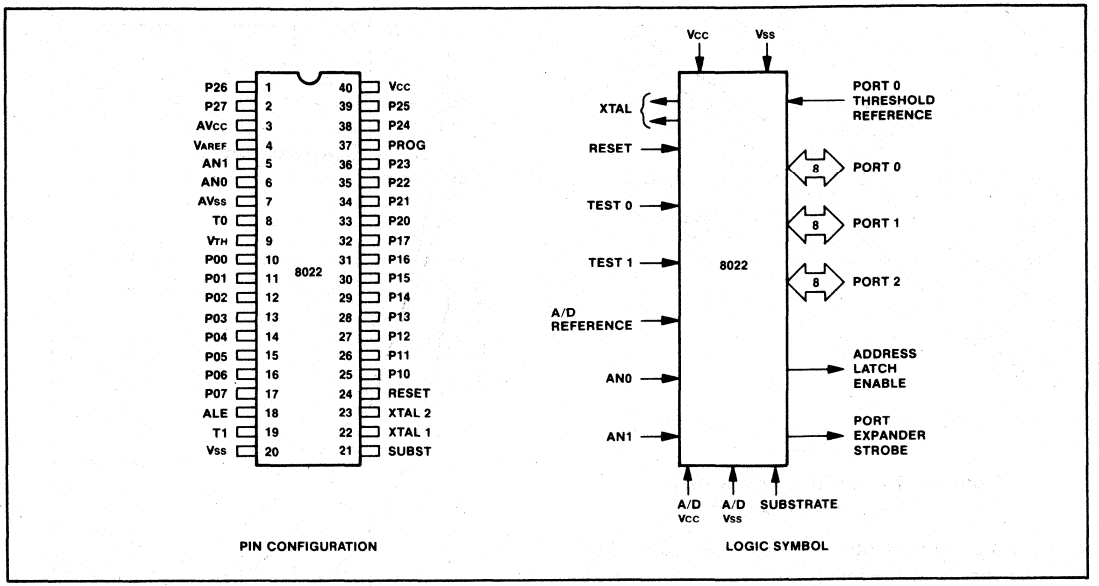


Figure 7.

as the load changes. As the monitored phase angle increases, a solid-state switch (triac) is used to reduce the voltage applied to the motor (FIG. 3). This increases motor slip and reduces the phase angle to a predetermined value. The feedback loop forces the motor to run at a constant, optimum, phase angle selected by the user, thereby minimizing wasted power. Since phase shift can be sampled as often as every power-line cycle, the 8022 can immediately respond to changes in the load and the motor will always run at near-peak efficiency, supplying only the amount of power required.

The PFC was originally tested at the Marshall Space Flight Center on 40 motors ranging from $^1/_{12}$ hp to 5 hp, both single phase and 3 phase.[5] Most motors exhibited a 40-60% savings at no load and 0-10% savings when fully loaded. The more expensively built 3-phase motors being the more efficient. When the device was used in 15 typical applications with single phase motors, including a drill press and vacuum pump, the typical savings were 30-40% when idling and 10% when loaded. The vacuum pump even ran about 25 degrees C cooler with the device. In a 500 hour test of two identical machines performing the same task, the piece of equipment with the PFC used 33% less energy (FIG. 6).

The device was also tested extensively by the Auburn University and many companies, all coming to the same conclusion. The Power Factor Controller is proven to offer significant savings to the user that, by the way, go beyond the actual direct power reduction mentioned. These multiple savings include the motor running cooler and quieter, extending the motor's life as well as saving in an air conditioned environment since there is less heat generated, and also by actually controlling the power factor to a desired value. This will benefit large users of motors with cyclic loads who are often charged by the Power Company for a poor power factor and are forced to correct this condition with large capacitor banks or a synchronous condenser.

[5] Ibid.

The 8022

The PFC developed by NASA consists of discrete electronic parts, designed to produce an analog controller. The 8022 microcontroller brings the benefit of a programmable computer to this application so that intelligent control of the total system can be effected. The 8022 is especially suited for an analog environment as it contains a 2-channel 8-bit A/D converter, zero-cross detection circuitry, and comparitor inputs with a controllable threshold. All of these capabilities are integrated into a single chip, along with 2048 bytes of program storage, 64 bytes of RAM, an 8-bit internal timer/event counter, external interrupt input, two high current drive outputs, and three 8-bit I/O ports. The result is a complete microcomputer system on a single chip. If the reader desires more information about the 8022, (s)he may refer to the MCS-48 User's Manual and application note AP-56, "Designing With Intel's 8022 Microcontroller," for a complete description of the 8022.

The 8022 is already being used to control many systems which utilize an induction motor. The extra code required for the PFC operation is only 154 bytes (less than 8% of the available program store). The main program would, of course, have to be modified to facilitate the PFC's function. This can easily be performed by placing the PFC operation in an interrupt routine. The total number of pins delegated to the PFC operation is 4: three I/O pins and the T1 zero-cross input. One of these I/O pins is dedicated to control Vth.

These three interface lines: the voltage-cross input, the current-cross input, and the triac gate control, are all that is required for the PFC application. The 8022 simply measures the amount of lag time between the voltage zero-crossing and the current zero-crossing. This measured time is then converted to a phase angle and subsequently compared to the desired phase angle. As the load changes and the measured phase angle shifts either greater than or less than the desired value, the 8022 will either lengthen or shorten the triac off time. The result is that the motor only gets the amount of current needed to drive the instantaneous load.

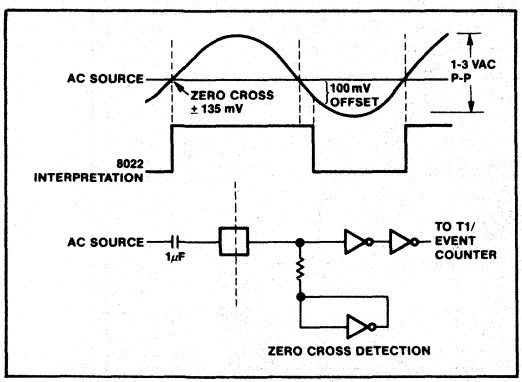


Figure 8.

Hardware Description

To perform the PFC function, the motor was placed in series with a triac which cuts out portions of the applied voltage, and a small series resistor which was used for measuring the current (FIG. 9). To isolate the 8022 from the AC line voltage, a small audio transformer was placed across the current sensing resistor, and an opto-isolator was used between the gate of the triac and the pin driving it. The voltage signal was taken from the secondary side of the power supply transformer.

Aside from the isolation, the only other interface hardware was some simple signal conditioning. Since the T1 zero-cross input requires a 1-3 VAC p-p signal, two diodes to DC ground clipped the AC signal nicely at 1.5 VAC p-p. A resistor for current limiting and a 1 microfarad capacitor to T1 complete the voltage zero-cross.

One of the port 0 comparator inputs was used for the current zero-cross detection. The 8022 had to be able to recognize when the current waveform was approaching zero from both the positive side and the negative side since it doesn't cross zero at any particular point, but rather approaches it from either side then remains zero until the triac fires again (FIG. 3). This was done by having two separate thresholds for the comparator. One for the negative slope crossing and one for the positive. To accomplish this, the current waveform was first boosted up with a DC level so it would be entirely positive. VTH was then biased to this DC level by two resistors. The threshold would be changed by about 0.1v either side of this DC level under software control by writing either a logic "1" or "0" to P17 when anticipating either a positive approach or a negative approach, respectively. This would place the 100k ohm resistor in parallel with either resistor thereby decreasing its value and subsequently raising or lowering the threshold.

The remaining hardware was included to aid in the development of this application note, but is not necessary for the system's operation. The remaining 6 pins on port zero were used to input the desired phase angle. The remaining 7 pins on port 1 were used to control the 7 segments of a display. The upper nibble of port 2 was used to control up to 10 digits of a display. The user would then select the desired phase angle at will, with the actual phase angle being written to the display.

With this pin assignment the remaining pins include the lower nibble of port 2 which can be used with the Intel 8243 I/O expander, adding four 4-bit I/O ports. The T0 pin can still be used as either a testable input, the interrupt request pin, or both. Finally, the two channels of the A/D converter are available, allowing the chip to interface directly with analog transducers. Aside from the Power Factor Controller and the direct user interface with the 8022 (via a keypad or thumbwheel switches and the display), there are still enough pins left over for almost any controller application.

Hardware not required for the PFC operation is shaded in the schematic.

Software Description

To simplify the software, the triac "offtime" is quantized in units of 0.27ms. This represents one time-unit of the timer (operating with a 3.58 MHz TV crystal). The "offtime" is therefore incremented or decremented by one of these units (or remained unchanged) when it is

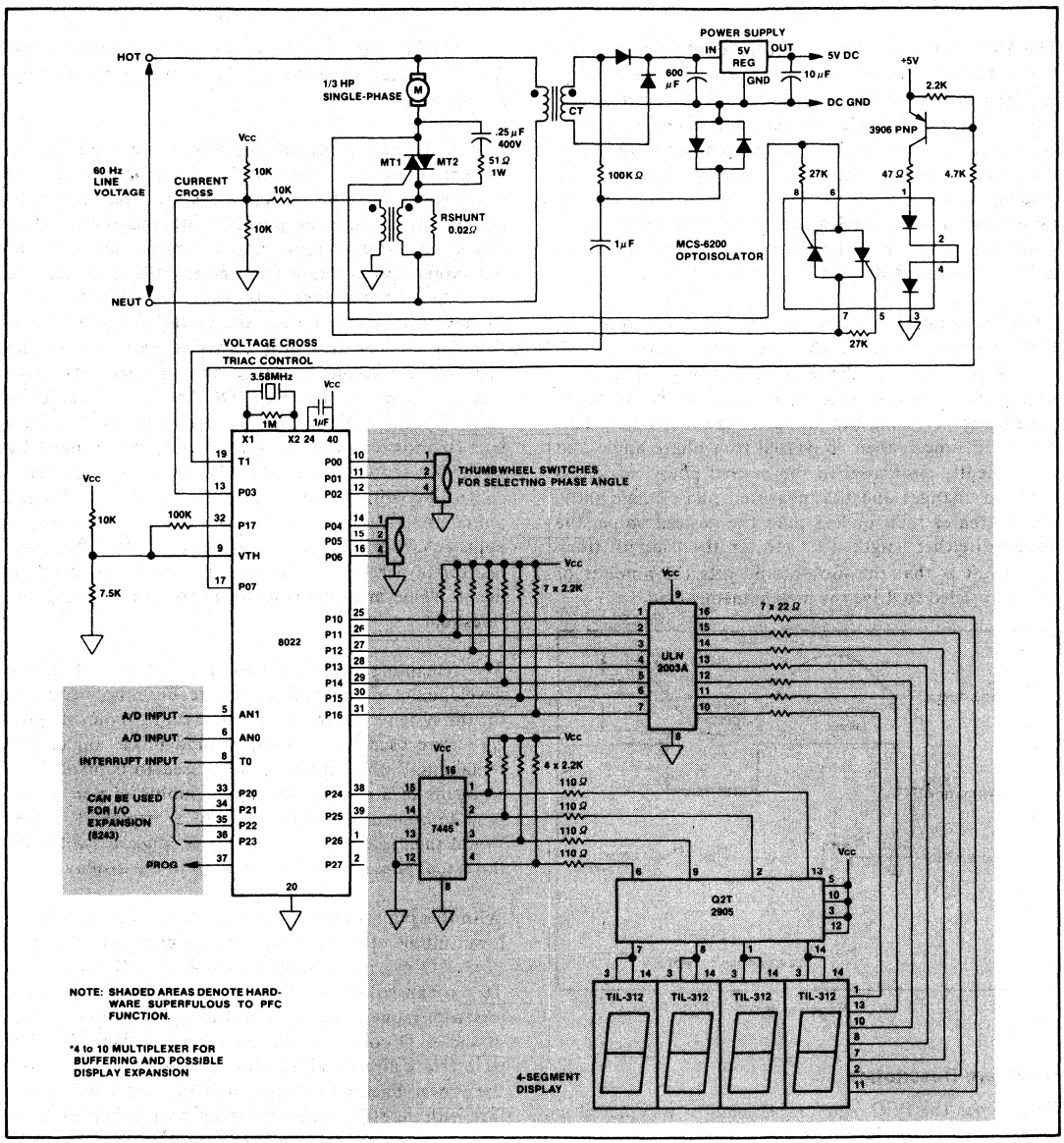


Figure 9.

reviewed for adjustment every cycle. This proves to be an adequate reaction to abrupt changes in the load. If however, the phase angle will only be sampled every 3 cycles, for example, on an interrupt, a simple routine has been included to change the "offtime" by more than one increment should the phase angle change more than 12 degrees between samplings. This is included to compensate for extremely abrupt load changes.

For simplicity, the phase angle is also measured in units of approximately 6 degrees, to coincide with the timer unit of 0.27ms. The lookup table consists of these quantized values. For a steady phase angle, the desired phase angle should be chosen to be one of these values. If any value between two of these are chosen, the phase angle oscillates between these two values and will never equal the chosen value, since this value is not in the lookup

table. This can be rectified by using a table lookup and interpolation instead of a simple table lookup (see AP-24; "Application Techniques for the MCS-48 Family"). There is no apparent benefit in doing this, however, as the value-searching over a 6 degree range doesn't seem to cause any problems.

The software also distinguishes between that necessary for the PFC only, and that used to include the user interface features. The BCD-to-binary routine was adapted from AP-49, "Serial I/O and Math Utilities for the 8049 Microcomputer," as was the binary-to-BCD routine. The total bytes of program storage consumed by the PFC only, is 154, and the total for this entire application is 328.

The software flowchart and complete listing follow.

Conclusion

The advantages of controlling a system with a programable microcontroller are evident. The added advantage of conserving power with the 8022 in the same application is realized in this application note. As the Power Factor Controller will be used in more and more applications as an energy-saving feature, the 8022 is ideally suited to implement it.

To illustrate this, the following application shows the 8022 controlling a Heat Pump / Solar Heating system.

Figure 10 illustrates the type of solar heating system that was the target of the NASA invention. The 8022 can control this system, and in addition, implement the PFC function on the motors driving the heatpump compressor and fan. The A/D can be used to monitor the temperatures in the system, and the various pumps and fans can be activated so as to maximize the heat extracted from the solar radiation. As an example of this, the flow rate in the solar-collector circuit can be adjusted by the 8022 so that heat can be collected even on cloudy days. (Lowering the flow rate lets the fluid remain in the collectors longer so that the heat radiation has a longer period in which to heat the fluid.) The 2K bytes of program storage available on the 8022 would make the addition of features such as a real-time clock for controlling the temperature to different levels at different times of the day, and dual-level thermostat settings, possible at little or no incremental product cost.

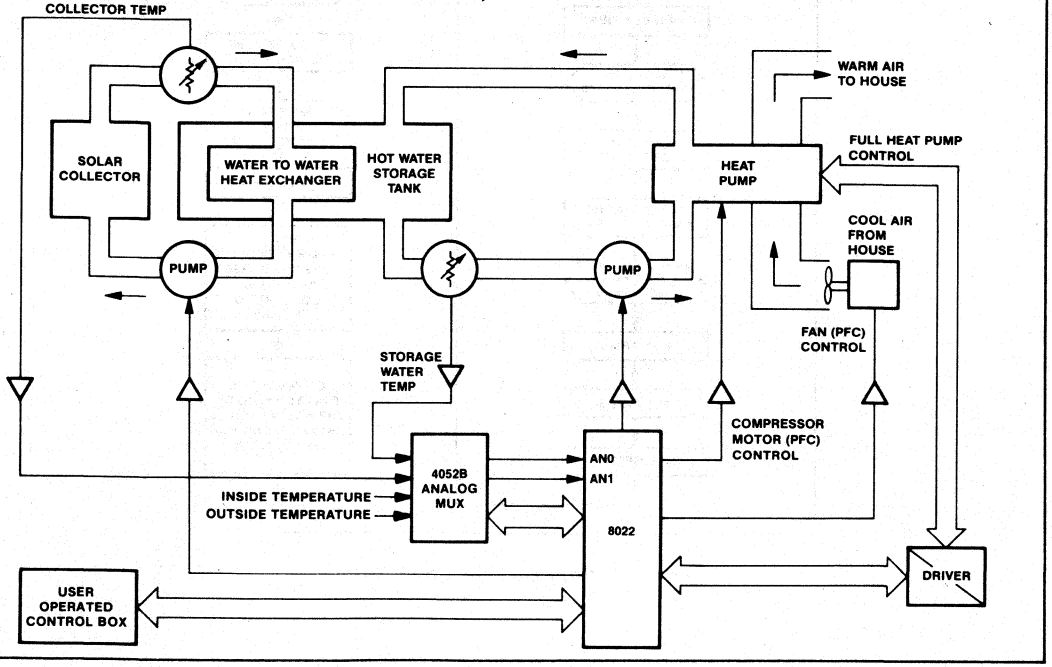
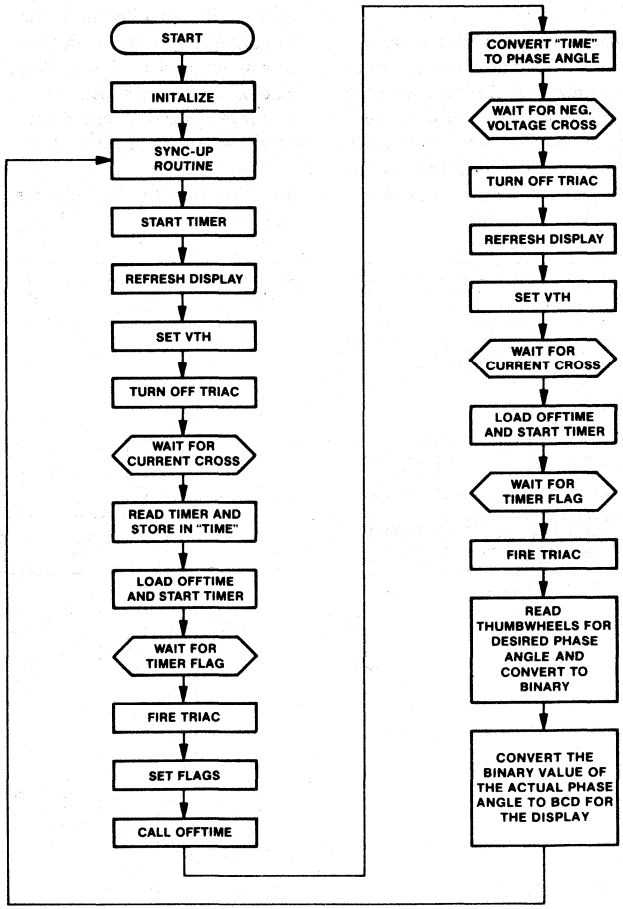


Figure 10.



Program Flow Chart

```
ASM48 AP PAGELENGTH(88)

ISIS-II MCS-48/UPI-41 MACRO ASSEMBLER, V2.0                PAGE    1

  LOC  OBJ          SEQ          SOURCE STATEMENT
                      1 $MOD22 SYMBOLS MACROFILE XREF
                      2
                      3 ;*****************************************************************
                      4 ;           DEFINITIONS OF MACROS
                      5 ;*****************************************************************
                      6
                      7 LDA       MACRO   PAR1
                      8           MOV     R0,#PAR1
                      9           MOV     A,@R0
                     10           ENDM
                     11
                     12 STA       MACRO   PAR1
                     13           MOV     R0,#PAR1
                     14           MOV     @R0,A
                     15           ENDM
                     16
0000 1498            17 INIT:     CALL    INITZ              ;INITIALIZE
                     18
                     19 ;*****************************************************************
                     20 ;             THE FOLLOWING IS A SYNC-UP ROUTINE TO ASSURE
                     21 ;              A START ON A RAISING VOLTAGE ZERO-CROSS
                     22 ;*****************************************************************
                     23
0002 5606            24 PWRON:    JT1     VOWAIT             ;JUMP ON T1=1
0004 0402            25           JMP     PWRON
                     26
0006 460A            27 VOWAIT:   JNT1    WAITST             ;JUMP ON T1=0
0008 0406            28           JMP     VOWAIT
                     29
000A 560E            30 WAITST:   JT1     START              ;JUMP ON T1=1
000C 040A            31           JMP     WAITST
                     32
                     33 ;*****************************************************************
                     34 ;           END OF SYNC-UP                ==START PROGRAM==
                     35 ;*****************************************************************
                     36
000E 27              37 START:    CLR     A
000F 62              38           MOV     T,A                ;CLEAR PRESCALER
0010 55              39           STRT    T                  ;START TIMER
                     40
0011 3400            41           CALL    REFRSH             ;REFRESH THE DISPLAY
                     42                                      ;VTH SET FOR UPCOMING CURRENT AUTO-
                     43                                      ;MATICALLY IN REFRSH ROUTINE (P17=0)
                     44
0013 23FF            45           MOV     A,#0FFH
0015 90              46           OUTL    P0,A               ;TURN OFF TRIAC
                     47
0016 08              48 IPWAIT:   IN      A,P0
0017 5308            49           ANL     A,#00001000B       ;WAIT FOR CURRENT CROSS
0019 C616            50           JZ      IPWAIT
                     51
001B 42              52           MOV     A,T                ;READ TIMER
001C AA              53           MOV     TIME,A             ;STORE TIMER VALUE
                     54
001D 14BB            55           CALL    TRIAC
                     56
                     57 $EJECT
```

NOTE: SHADED AREAS DENOTE SOFTWARE SUPERFULOUS TO PFC FUNCTION.

```
LOC  OBJ          SEQ         SOURCE STATEMENT

                    58
                    59 ;*****************************************************************
                    60 ;                          SET THE FLAGS
                    61 ;*****************************************************************
                    62
001F 97             63          CLR     C               ;CLEAR THE CARRY FLAG
                    64
0020 FC             65          MOV     A,SETVAL
0021 37             66          CPL     A
0022 17             67          INC     A               ;2'S COMPLIMENT SETVAL
0023 6D             68          ADD     A,PHSANG        ;PHSANG-SETVAL = DELTA
                    69
0024 C64F           70          JZ      LOOK            ;IF PHSANG=SETVAL, DONT CHANGE ANYTHING
                    71
                    72          STA     DELTA
0026 B826           73+         MOV     R0,#DELTA
0028 A0             74+         MOV     @R0,A
                    75
0029 5380           76          ANL     A,#10000000B    ;TEST FOR NEG
002B C638           77          JZ      OTCALC          ;IF POS DO NOT SET FLAG
002D 1E             78          INC     PHLTSV          ;SET PHSANG L.T. SETVAL FLAG
                    79
                    80          LDA     DELTA
002E B826           81+         MOV     R0,#DELTA
0030 F0             82+         MOV     A,@R0
0031 030A           83          ADD     A,#10D          ;10-DELTA
0033 5380           84          ANL     A,#10000000B
0035 C638           85          JZ      OTCALC
0037 1F             86          INC     DLGT10          ;SET DELTA G.T. 10 FLAG IF NEG
                    87
                    88 ;*****************************************************************
                    89 ;                CHANGE OFFTIME PER FLAGS NEWLY SET
                    90 ;*****************************************************************
                    91
0038 FE             92 OTCALC:  MOV     A,PHLTSV        ;DECREMENT OR INCREMENT?
0039 C649           93          JZ      DECREM          ;IF PHLTSV IS CLEAR THEN DECREMENT
                    94
003B FB             95 INCREM:  MOV     A,OFFTIM        ;IF OFFTIME=255 OR 254 DON'T INCREMENT
003C 37             96          CPL     A
003D C64C           97          JZ      CLRFLG
003F 07             98          DEC     A
0040 C64C           99          JZ      CLRFLG
                   100
0042 1B            101          INC     OFFTIM          ;INCREMENT
                   102
0043 FF            103          MOV     A,DLGT10        ;SHOULD IT INCREMENT TWICE?
0044 C64C          104          JZ      CLRFLG
0046 1B            105          INC     OFFTIM
0047 044C          106          JMP     CLRFLG
                   107
0049 FB            108 DECREM: MOV     A,OFFTIM
004A 07            109          DEC     A
004B AB            110          MOV     OFFTIM,A        ;DECREMENT
                   111
004C 27            112 CLRFLG: CLR     A
004D AE            113          MOV     PHLTSV,A
004E AF            114          MOV     DLGT10,A        ;CLEAR FLAGS
                   115
                   116 $EJECT
```

```
LOC  OBJ          SEQ          SOURCE STATEMENT

                   117
                   118 ;*************************************************************************
                   119 ;                    LOOKUP VALUE FOR "PHSANG" FROM "TIME"
                   120 ;*************************************************************************
                   121
004F 23C9          122 LOOK:    MOV      A,#(LOW TAB)    ;BOTTOM OF TABLE
0051 6A            123          ADD      A,TIME
0052 A3            124          MOVP     A,@A            ;GET PHSANG VALUE
0053 AD            125          MOV      PHSANG,A
                   126
                   127 ;*************************************************************************
                   128 ;                    WAIT FOR NEGATIVE GOING VOLTAGE CROSS
                   129 ;*************************************************************************
                   130
0054 4658          131 HOLD:    JNT1     CONTON
0056 0454          132          JMP      HOLD
                   133
0058 23FF          134 CONTON:  MOV      A,#0FFH         ;TURN OFF TRIAC
005A 90            135          OUTL     P0,A
                   136
005B 3400          137          CALL     REFRSH          ;REFRESH THE DISPLAY
                   138          LDA      SEGDAT
005D B824          139+         MOV      R0,#SEGDAT
005F F0            140+         MOV      A,@R0
0060 4380          141          ORL      A,#10000000B    ;SET VTH FOR DOWN GOING CURRENT-
0062 39            142          OUTL     P1,A            ;BY SETTING P17
                   143
0063 08            144 INWAIT:  IN       A,P0            ;WAIT FOR CURRENT CROSS
0064 5308          145          ANL      A,#00001000B
0066 9663          146          JNZ      INWAIT
                   147
0068 14BB          148          CALL     TRIAC
                   149
                   150 ;*************************************************************************
                   151 ;                    READ THUMBWHEELS
                   152 ;                  AND CONVERT TO BINARY
                   153 ;*************************************************************************
                   154
006A 08            155          IN       A,P0
006B 5377          156          ANL      A,#01110111B
                   157          STA      BINTR           ;STORE ACC IN BINTR
006D B820          158+         MOV      R0,#BINTR
006F A0            159+         MOV      @R0,A
                   160
0070 14A7          161          CALL     SELRB1
                   162
                   163          LDA      BINTR
0072 B820          164+         MOV      R0,#BINTR
0074 F0            165+         MOV      A,@R0
                   166
0075 344A          167          CALL     CONBIN          ;CONVERT THUMBWHEELS TO BINARY
                   168
                   169          STA      BINTR           ;STORE ACC IN BINTR
0077 B820          170+         MOV      R0,#BINTR
0079 A0            171+         MOV      @R0,A
                   172
007A 14B1          173          CALL     SELRB0
                   174
                   175          LDA      BINTR           ;MOV BINTR TO ACC
007C B820          176+         MOV      R0,#BINTR
007E F0            177+         MOV      A,@R0
007F AC            178          MOV      SETVAL,A
                   179
                   180 $EJECT
```

```
LOC  OBJ          SEQ          SOURCE STATEMENT

                  181
                  182 ;*****************************************************************
                  183 ;       CONVERT BINARY VALUE OF THE PHASE ANGLE (PHSANG)
                  184 ;                       TO BCD (ANGLE)
                  185 ;*****************************************************************
                  186
0080 FD           187         MOV     A,PHSANG
                  188         STA     BCDTR           ;STORE PHASE ANGLE
0081 B821         189+        MOV     R0,#BCDTR
0083 A0           190+        MOV     @R0,A
                  191
0084 14A7         192         CALL    SELRB1
                  193
                  194         LDA     BCDTR
0086 B821         195+        MOV     R0,#BCDTR
0088 F0           196+        MOV     A,@R0
                  197
0089 3437         198         CALL    CNBCD
                  199
                  200         STA     BCDTR
008B B821         201+        MOV     R0,#BCDTR
008D A0           202+        MOV     @R0,A
                  203
008E 14B1         204         CALL    SELRB0
                  205
                  206         LDA     BCDTR
0090 B821         207+        MOV     R0,#BCDTR
0092 F0           208+        MOV     A,@R0
                  209         STA     ANGLE
0093 B823         210+        MOV     R0,#ANGLE
0095 A0           211+        MOV     @R0,A
                  212
                  213
0096 040A         214         JMP     WAITST          ;GO BACK TO BEGINNING AND REPEAT
                  215
                  216 ;*****************************************************************
                  217 ;*****************************************************************
                  218 ;*****************************************************************
                  219 ;   ------------------- END OF MAIN PROGRAM ---------------------
                  220 ;*****************************************************************
                  221 ;*****************************************************************
                  222 ;*****************************************************************
                  223
                  224 $EJECT
```

```
LOC  OBJ          SEQ          SOURCE STATEMENT

                  225
                  226 INITZ:
                  227
                  228 ;*****************************************************************************
                  229 ;                    ALLOCATE    RAM
                  230 ;*****************************************************************************
                  231
0020              232 BINTR    EQU     20H
0021              233 BCDTR    EQU     21H
0022              234 PASTOR   EQU     22H
0023              235 ANGLE    EQU     23H
0024              236 SEGDAT   EQU     24H
0025              237 DISFLG   EQU     25H
0026              238 DELTA    EQU     26H
                  239
0002              240 TIME     EQU     R2
0003              241 OFFTIM   EQU     R3
0004              242 SETVAL   EQU     R4
0005              243 PHSANG   EQU     R5
0006              244 PHLTSV   EQU     R6
0007              245 DLGT10   EQU     R7
                  246
                  247 ;*************************************************
                  248 ;       FOR USE IN CONBIN AND CONBCD
                  249 ;       ****************************
                  250
0002              251 XA       EQU     R2
0003              252 COUNT    EQU     R3
0004              253 ICNT     EQU     R4
0001              254 DIGPR    EQU     1
                  255
                  256 ;*************************************************
                  257
0098 BC29         258          MOV     R4,#29H         ;SETVAL=41
009A BD29         259          MOV     R5,#29H         ;PHSANG=41
009C BBFF         260          MOV     R3,#0FFH        ;SET OFFTIM TO 255
                  261
009E 237F         262          MOV     A,#7FH          ;TURN ON TRIAC
00A0 90           263          OUTL    P0,A
                  264
00A1 97           265          CLR     C
00A2 27           266          CLR     A
                  267          STA     DISFLG          ;CLEAR DISPLAY FLAG
00A3 B825         268+         MOV     R0,#DISFLG
00A5 A0           269+         MOV     @R0,A
                  270
00A6 83           271          RET
                  272
                  273 $EJECT
```

```
LOC  OBJ          SEQ          SOURCE STATEMENT

                  274
                  275 ;*****************************************************************
                  276 ;        THESE NEXT TWO SUBROUTINES ACT LIKE A BANK SELECT
                  277 ;              EXCEPT THAT R0 AND R1 GET WIPED OUT
                  278 ;*****************************************************************
                  279
                  280
                  281
                  282
                  283
                  284
                  285 SELRB1:
00A7 B918         286          MOV     R1,#18H
00A9 B807         287          MOV     R0,#07H
00AB F0           288 BAK1:    MOV     A,@R0
00AC A1           289          MOV     @R1,A
00AD 19           290          INC     R1
00AE E8AB         291          DJNZ    R0,BAK1
00B0 83           292          RET
                  293
                  294 ;*****************************************************************
                  295
                  296
                  297 SELRB0:
00B1 B918         298          MOV     R1,#18H
00B3 B807         299          MOV     R0,#07H
00B5 F1           300 BAK2:    MOV     A,@R1
00B6 A0           301          MOV     @R0,A
00B7 19           302          INC     R1
00B8 E8B5         303          DJNZ    R0,BAK2
00BA 83           304          RET
                  305
                  306 $EJECT
```



```
LOC  OBJ          SEQ          SOURCE STATEMENT

                  307
                  308 ;*****************************************************************
                  309 ;                  THIS ROUTINE FIRES THE TRIAC AFTER A
                  310 ;                        PREDETERMINED OFFTIME
                  311 ;*****************************************************************
                  312
                  313
                  314
                  315
                  316 TRIAC:
                  317
00BB 65           318          STOP    TCNT            ;STOP TIMER
00BC 16BE         319          JTF     $+2             ;CLEAR TIMER FLAG
                  320
00BE FB           321          MOV     A,OFFTIM        ;SET PRESCALER TO OFFTIME
00BF 62           322          MOV     T,A
                  323
00C0 55           324          STRT    T               ;START TIMER
                  325
00C1 16C5         326 CNTDWN:  JTF     FIRE            ;WAIT FOR TIMER FLAG
00C3 04C1         327          JMP     CNTDWN
                  328
00C5 237F         329 FIRE:    MOV     A,#7FH          ;FIRE
00C7 90           330          OUTL    P0,A            ;TRIAC
                  331
00C8 83           332          RET
                  333
00C9              334 TAB      EQU     $
                  335
                  336 $EJECT
```

```
 LOC  OBJ          SEQ          SOURCE STATEMENT

                    337 ;*****************************************************************
                    338 ;
                    339 ;          THIS ROUTINE REFRESHES THE DISPLAY
                    340 ;*****************************************************************
                    341
 0100               342          ORG     100H
                    343 REFRSH:
 0100 23F0          344          MOV     A,#0F0H
 0102 3A            345          OUTL    P2,A            ;TURN OFF DIGITS
                    346
                    347          LDA     DISFLG
                    348+         MOV     R0,#DISFLG
 0103 B825          349+         MOV     A,@R0
 0105 F0            350          JZ      NOSWAP          ;CHECK DISPLAY FLAG
 0106 C60E          351          LDA     ANGLE
                    352+         MOV     R0,#ANGLE
 0108 B823          353+         MOV     A,@R0
 010A F0            354          SWAP    A
 010B 47            355          JMP     CONT2
 010C 2411          356
                    357 NOSWAP: LDA      ANGLE
                    358+         MOV     R0,#ANGLE
 010E B823          359+         MOV     A,@R0
 0110 F0            360 CONT2:   ANL     A,#0FH          ;MASK UPPER NIBBLE
 0111 530F          361          ADD     A,#(LOW TABL)
 0113 032D          362          MOVP    A,@A            ;GET CHAR CODE
 0115 A3            363
                    364          STA     SEGDAT
                    365+         MOV     R0,#SEGDAT
 0116 B824          366+         MOV     @R0,A
 0118 A0            367          OUTL    P1,A            ;TURN ON THE SEGMENTS
 0119 39            368
                    369          LDA     DISFLG
                    370+         MOV     R0,#DISFLG
 011A B825          371+         MOV     A,@R0
 011C F0            372          JZ      DIG1
 011D C623          373
                    374 DIG2:    MOV     A,#10H
 011F 2310          375          JMP     CONT3
 0121 2424          376
                    377 DIG1:    CLR     A
 0123 27            378
                    379 CONT3:   OUTL    P2,A            ;TURN ON THE DIGIT
 0124 3A            380
                    381          LDA     DISFLG
                    382+         MOV     R0,#DISFLG
 0125 B825          383+         MOV     A,@R0
 0127 F0            384          CPL     A
 0128 37            385          STA     DISFLG          ;CHANGE DISPLAY FLAG
                    386+         MOV     R0,#DISFLG
 0129 B825          387+         MOV     @R0,A
 012B A0            388
 012C 83            389          RET
                    390
                    391 TABL:
 012D 3F            392          DB      00111111B       ; 0
 012E 06            393          DB      00000110B       ; 1
 012F 5B            394          DB      01011011B       ; 2
 0130 4F            395          DB      01001111B       ; 3
 0131 66            396          DB      01100110B       ; 4
 0132 6D            397          DB      01101101B       ; 5
 0133 7D            398          DB      01111101B       ; 6
 0134 07            399          DB      00000111B       ; 7
 0135 7F            400          DB      01111111B       ; 8
 0136 67            401          DB      01100111B       ; 9
                    402
 0137               403 HERE1    EQU     $
                    404
                    405 $EJECT
```

```
LOC  OBJ          SEQ         SOURCE STATEMENT

                  406
                  407 ;*****************************************************************
                  408 ;
                  409 ;                 LOOKUP  TABLE  FOR THE PHASE ANGLE
                  410 ;
                  411 ;*****************************************************************
                  412
00C9              413         ORG     TAB
                  414
00C9 00           415         DB      00H
00CA 06           416         DB      06H
00CB 0C           417         DB      0CH
00CC 11           418         DB      11H
00CD 17           419         DB      17H
00CE 1D           420         DB      1DH
00CF 23           421         DB      23H
00D0 29           422         DB      29H
00D1 2E           423         DB      2EH
00D2 34           424         DB      34H
00D3 3A           425         DB      3AH
00D4 40           426         DB      40H
00D5 46           427         DB      46H
00D6 4B           428         DB      4BH
00D7 51           429         DB      51H
00D8 57           430         DB      57H
00D9 5D           431         DB      5DH
00DA 63           432         DB      63H
00DB 68           433         DB      68H
00DC 6E           434         DB      6EH
00DD 74           435         DB      74H
00DE 7A           436         DB      7AH
00DF 80           437         DB      80H
00E0 85           438         DB      85H
00E1 8B           439         DB      8BH
00E2 91           440         DB      91H
00E3 97           441         DB      97H
00E4 97           442         DB      97H
00E5 97           443         DB      97H
00E6 97           444         DB      97H
                  445
                  446 $EJECT
```

```
LOC  OBJ          SEQ          SOURCE STATEMENT

                    447
0137                448            ORG     HERE1
                    449
                    450 $          INCLUDE(CONBCD)
                  = 451 ;*****************************************************************
                  = 452 ;*
                  = 453 ;*                  CONBCD
                  = 454 ;*
                  = 455 ;*================================================================
                  = 456 ;*
                  = 457 ;*                  THIS UTILITY CONVERTS AN 8 BIT BINARY VALUE TO BCD
                  = 458 ;*                  AT ENTRY:
                  = 459 ;*                        A = 8 BIT BINARY VALUE
                  = 460 ;*
                  = 461 ;*                  AT EXIT:
                  = 462 ;*                        A = BCD VALUE
                  = 463 ;*****************************************************************
                  = 464
0005              = 465 TEMP1   SET     R5
0006              = 466 TEMP2   SET     R6
                  = 467 ;
                  = 468 ;1 CONVERT TO BCD
                  = 469 CNBCD:
                  = 470 ;1 BCDACC:=0
0137 BE00         = 471         MOV     TEMP2,#00
                  = 472 ;1 COUNT:=8
0139 BB08         = 473         MOV     COUNT,#08
                  = 474 ;1 REPEAT
                  = 475 BCDCOR:
                  = 476 ;2      BIN:=BIN*2
013B 97           = 477         CLR     C
013C F7           = 478         RLC     A
                  = 479 ;2      BCD:=BCD*2+CARRY
013D AD           = 480         MOV     TEMP1,A
013E FE           = 481 BCDOC:  MOV     A,TEMP2
013F 7E           = 482         ADDC    A,TEMP2
0140 57           = 483         DA      A
0141 AE           = 484         MOV     TEMP2,A
0142 FD           = 485         MOV     A,TEMP1
                  = 486 ;2      IF CARRY FROM BCDACC GOTO ERROR EXIT
0143 F649         = 487         JC      BCDCOD
                  = 488 ;2      COUNT:=COUNT-1
                  = 489 ;1 UNTIL COUNT=0
0145 EB3B         = 490         DJNZ    COUNT,BCDCOR
0147 97           = 491         CLR     C       ; CLEAR CARRY TO INDICATE NORMAL TERMINATION
0148 FE           = 492         MOV     A,TEMP2 ; PUT BCD VALUE IN ACCUMULATOR FOR RETURN
                  = 493 ;1 END CONVERT TO BCD
0149 83           = 494 BCDCOD: RET
                  = 495 $EJECT
```

```
LOC  OBJ          SEQ          SOURCE STATEMENT

                    496 $       INCLUDE(CONB2)
                  = 497 ;*****************************************************************
                  = 498 ;*
                  = 499 ;*      CONBIN
                  = 500 ;*
                  = 501 ;*===============================================================
                  = 502 ;*
                  = 503 ;*      THIS UTILITY CONVERTS A 2 DIGIT BCD VALUE TO BINARY
                  = 504 ;*      AT ENTRY:
                  = 505 ;*        A = BCD VALUE
                  = 506 ;*
                  = 507 ;*      AT EXIT:
                  = 508 ;*        A = EIGHT BITS OF THE BINARY RESULT
                  = 509 ;*
                  = 510 ;*****************************************************************
                  = 511 ;
                  = 512 ;
                  = 513 ;
0005              = 514 TEMP1   SET     R5
0006              = 515 TEMP2   SET     R6
0007              = 516 TEMP3   SET     R7
                  = 517 ;
                  = 518 ;1 CONVERT_TO_BINARY
                  = 519 CONBIN:
                  = 520 ;1 COUNT:=DIGITPAIR
014A BB01         = 521         MOV     COUNT,#DIGPR
                  = 522 ;  REPEAT
                  = 523 CONBLP:
                  = 524 ;     BIN:=BIN*10
014C AF           = 525         MOV     TEMP3,A
014D 47           = 526         SWAP    A
014E 530F         = 527         ANL     A,#0FH
0150 345A         = 528         CALL    CONB10
                  = 529 ;     BIN:=BIN+MEM(R0)[7-4]
0152 AE           = 530         MOV     TEMP2,A
0153 FF           = 531         MOV     A,TEMP3
0154 530F         = 532         ANL     A,#0FH
0156 6E           = 533         ADD     A,TEMP2
                  = 534 ;     COUNT:=COUNT-1
                  = 535 ;  UNTIL COUNT=0
0157 EB4C         = 536         DJNZ    COUNT,CONBLP
                  = 537 ;  END CONVERT_TO_BINARY
0159 83           = 538 CONBER: RET
                  = 539 ;
                  = 540 $EJECT
```

```
LOC  OBJ          SEQ          SOURCE STATEMENT

                  = 541 ;
                  = 542 ;       UTILITY TO MULTIPLY BIN BY 10
                  = 543 ;       CARRY WILL BE SET IF OVERFLOW OCCURS
                  = 544 ;
015A AD           = 545 CONB10: MOV     TEMP1,A ; SAVE A
                  = 546
015B 97           = 547         CLR     C
015C F7           = 548         RLC     A       ; BIN:=BIN*2
                  = 549
015D F7           = 550         RLC     A       ; BIN:=BIN*4
                  = 551
015E 6D           = 552         ADD     A,TEMP1 ; BIN:=BIN*5
                  = 553
015F F7           = 554         RLC     A       ; BIN:=BIN*10
                  = 555
0160 83           = 556         RET
                  = 557
                  = 558 ;
                  = 559 $EJECT
```



```
LOC  OBJ          SEQ          SOURCE STATEMENT

                  560          END
```

```
USER SYMBOLS
ANGLE  0023    BAK1   00AB    BAK2   00B5    BCDCOB 013B    BCDCOD 0149
BCDOC  013E    BCDTR  0021    BINTR  0020    CLRFLG 004C    CNBCD  0137
CNTDWN 00C1    CONB10 015A    CONBER 0159    CONBIN 014A    CONBLP 014C
CONT2  0111    CONT3  0124    CONTON 0058    COUNT  0003    DECREM 0049
DELTA  0026    DIG1   0123    DIG2   011F    DIGPR  0001    DISFLG 0025
DLGT10 0007    FIRE   00C5    HERE1  0137    HOLD   0054    ICNT   0004
INCREM 003B    INIT   0000    INITZ  0098    INWAIT 0063    IPWAIT 0016
LDA    0000    LOOK   004F    NOSWAP 010E    OFFTIM 0003    OTCALC 0038
PASTOR 0022    PHLTSV 0006    PHSANG 0005    PWRON  0002    REFRSH 0100
SEGDAT 0024    SELRB0 00B1    SELRB1 00A7    SETVAL 0004    STA    0001
START  000E    TAB    00C9    TABL   012D    TEMP1  0005    TEMP2  0006
TEMP3  0007    TIME   0002    TRIAC  00BB    VOWAIT 0006    WAITST 000A
XA     0002

ASSEMBLY COMPLETE,  NO ERRORS
```

```
ANGLE   210    235#   352    358
BAK1    288#   291
BAK2    300#   303
BCDCOB  475#   490
BCDCOD  487    494#
BCDOC   481#
BCDTR   189    195    201    207    233#
BINTR   158    164    170    176    232#
CLRFLG   97     99    104    106    112#
CNBCD   198    469#
CNTDWN  326#   327
CONB10  528    545#
CONBER  538#
CONBIN  167    519#
CONBLP  523#   536
CONT2   355    360#
CONT3   375    379#
CONTON  131    134#
COUNT   252#   473    490    521    536
DECREM   93    108#
DELTA    73     81    238#
DIG1    372    377#
DIG2    374#
DIGPR   254#   521
DISFLG  237#   268    348    370    382    386
DLGT10   86    103    114    245#
FIRE    326    329#
HERE1   403#   448
HOLD    131#   132
ICNT    253#
INCREM   95#
INIT     17#
INITZ    17    226#
INWAIT  144#   146
IPWAIT   48#    50
LDA       7#    80    138    163    175    194    206    347    351    357    369    381
LOOK     70    122#
NOSWAP  350    357#
OFFTIM   95    101    105    108    110    241#   321
OTCALC   77     85     92#
PASTOR  234#
PHLTSV   78     92    113    244#
PHSANG   68    125    187    243#
PWRON    24#    25
REFRSH   41    137    343#
SEGDAT  139    236#   365
SELRB0  173    204    297#
SELRB1  161    192    285#
SETVAL   65    178    242#
STA      12#    72    157    169    188    200    209    267    364    385
START    30     37#
TAB     122    334#   413
TABL    361    391#
TEMP1   465#   480    485    514#   545    552
TEMP2   466#   471    481    482    484    492    515#   530    533
TEMP3   516#   525    531
TIME     53    123    240#
TRIAC    55    148    316#
VOWAIT   24     27#    28
WAITST   27     30#    31    214
XA      251#
```

CROSS REFERENCE COMPLETE

MCS®-51 Architecture 6

CHAPTER 6
MCS®-51 ARCHITECTURE

6.0 INTRODUCTION

The major MCS®-51 features:

- 8-Bit CPU
- On-Chip oscillator and clock circuitry
- 32 I/O lines
- 64K address space for external data memory
- 64K address space for external program memory
- Two 16-bit timer/counters (three on 8032/8052)
- A five-source interrupt structure (six sources on 8032/8052) with two priority levels
- Full duplex serial port
- Boolean processor

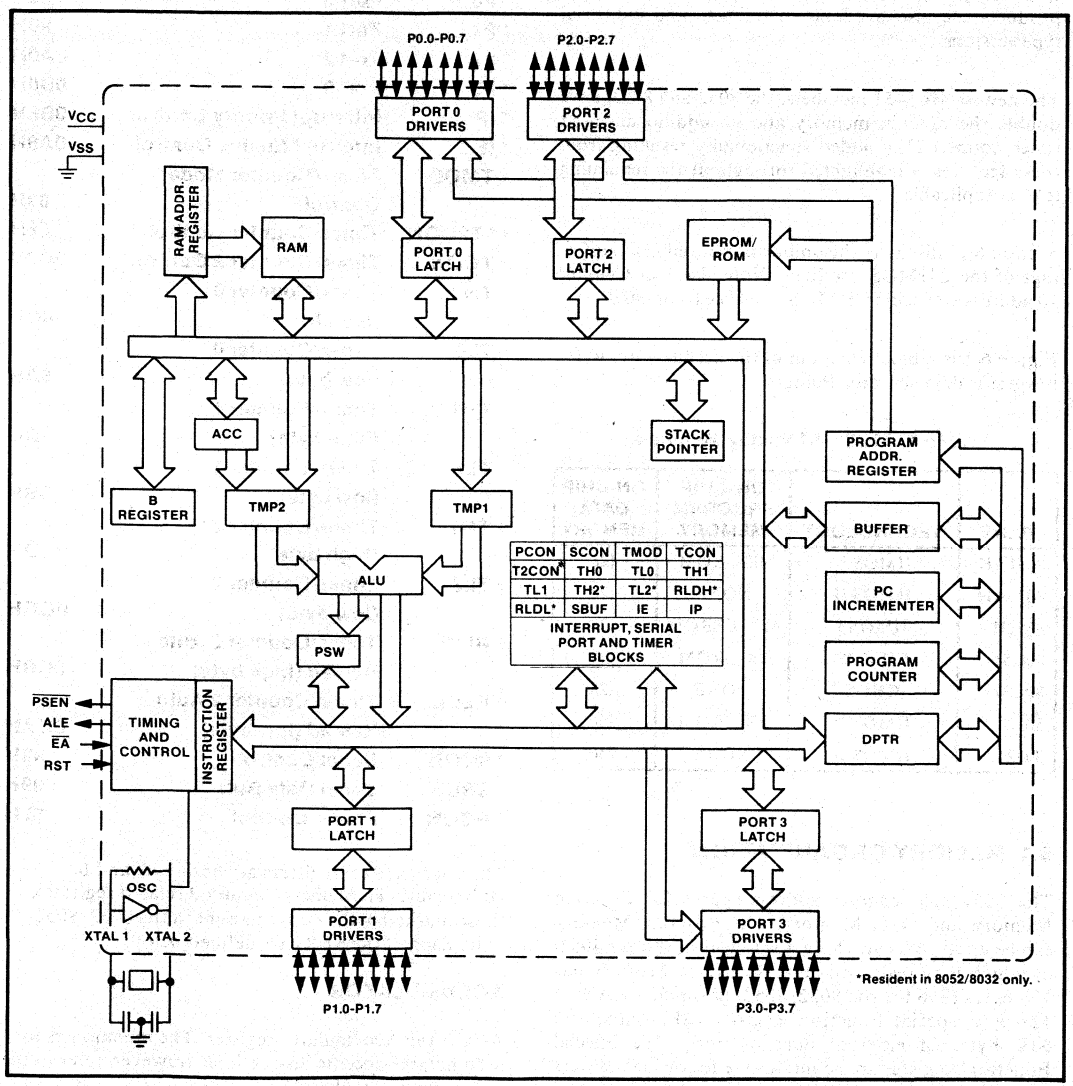


Figure 6-1. MCS-51 Architectural Block Diagram

The MCS®-51 family is a collection of functional and pin compatible ROM, ROMless and EPROM single component 8-bit microcontrollers based on the MCS-51 architecture. Table 1 lists each of the commercial grade members, their respective fabrication technology and memory attributes. The term "8051", however, is often used generically to refer to all of the family members. This is the case throughout this manual, except where specifically stated otherwise. Many of these MCS-51 products are also available in military and industrial type versions.

The newest MCS-51 members, the 8032 and 8052, have double the on-chip memory and an additional 16-bit timer/counter. The added functionality resulting from these features is highlighted throughout the remaining text as applicable.

Section 6.11 discusses the unique power reduction offerings of the CHMOS products. Note also that Chapter 10 addresses specific CHMOS interface techniques.

Figure 6-1 is a block diagram of the architecture. Refer to specific data sheet for Pinouts.

Table 1. MCS®-51 Family Members

PART	TECHNOLOGY	ON-CHIP PROGRAM MEMORY	ON-CHIP DATA MEMORY
8051AH	HMOS II	4K—ROM	128
8031AH	HMOS II	NONE	128
8751H	HMOS I	4K—EPROM	128
80C51	CHMOS	4K—ROM	128
80C31	CHOS	NONE	128
8052	HMOS II	8K—ROM	256
8032	HMOS II	NONE	256

6.1 MEMORY ORGANIZATION

The 8051 has separate address spaces for Program Memory and Data Memory. The Program Memory can be up to 64K bytes long, the lower 4K (8K for 8052) of which may reside on-chip. The Data Memory has 128 bytes (256 for the 8032/8052) of on-chip RAM, a 128-byte Special Function Register (SFR) area and 64K bytes of external data memory. The Special Function Register area contains the following registers in the indicated locations:

* ACC	Accumulator	0E0H
* B	B Register	0F0H
* PSW	Program Status Word	0D0H
SP	Stack Pointer	81H
DPTR	Data Pointer (consisting of DPH	83H and
	DPL	82H
* P0	Port 0	80H
* P1	Port 1	90H
* P2	Port 2	0A0H
* P3	Port 3	0B0H
* IP	Interrupt Priority Control	0B8H
* IE	Interrupt Enable Control	0A8H
TMOD	Timer/Counter Mode Control	89H
+* T2CON	Timer/Counter Control	88H
TCON	Timer/Counter 2 Control	0C8H
TH0	Timer/Counter 0 (high byte)	8CH
TL0	Timer/Counter 0 (low byte)	8AH
TH1	Timer/Counter 1 (high byte)	8DH
TL1	Timer/Counter 1 (low byte)	8BH
+ TH2	Timer/Counter 2 (high byte)	0CDH
+ TL2	Timer/Counter 2 (low byte)	0CCH
+ RLDH	Timer/Counter 2 Auto-Reload (high byte)	0CBH
+ RLDL	Timer/Counter 2 Auto-Reload (low byte)	0CAH
* SCON	Serial Control	98H
SBUF	Serial Data Buff	99H
PCON	Power Control	97H

The astericked (*) registers are both byte and bit addressable. The Timer/Counter 2 related registers, those marked with (+), are present in the 8032/8052 only. Each of the SFRs are defined below.

ACCUMULATOR

ACC is the Accumulator register. The mnemonics for accumulator-specific instructions, however, refer to the accumulator simply as A.

B REGISTER

The B register is used during multiply and divide operations. For other instructions it can be treated as another scratch pad register.

PROGRAM STATUS WORD

The PSW register contains program status information as detailed in Figure 6-2.

STACK POINTER

The Stack Pointer register is 8 bits wide. It is incremented before data is stored during PUSH and CALL executions. While the stack may reside anywhere in on-chip RAM, the Stack Pointer is initialized to 07H after a reset. This causes the stack to begin at location 08H.

DATA POINTER

The Data Pointer (DPTR) consists of a high byte (DPH) and a low byte (DPL). Its function is to hold a 16-bit address. It may be manipulated as a 16-bit register or as two independent 8-bit registers.

PORTS 0 to 3

The SFRs P0, P1, P2 and P3 are the latches of P0, P1, P2 and P3, respectively.

SERIAL DATA BUFFER

The Serial Data Buffer is actually two separate registers, a transmit buffer and a receive buffer register. When data is moved to SBUF, it goes to the transmit buffer where it is held for serial transmission. (Moving a byte to SBUF is what initiates the transmission). When data is moved from SBUF, it comes from the receive buffer.

CONTROL REGISTERS

Special Function Registers IP, IE, TMOD, TCON, T2CON, SCON, and PCON contain control and status bits for the interrupt system, the timer/counters, and the serial port. They will be described in later sections.

Chapter 2 is dedicated to a full description of the memory organization, addressing modes and data manipulation.

6.2 OSCILLATOR AND CLOCK CIRCUIT

XTAL1 and XTAL2 are the input and output, respectively, of an inverting amplifier which can be configured for use with a crystal oscillator, in the Pierce configuration, a ceramic resonator, or an external source. Crystal/ceramic resonator mode operation is as shown in Figure 6-3A.

(MSB) (LSB)

CY	AC	F0	RS1	RS0	OV	—	P

Symbol	Position	Name and Significance
CY	PSW.7	Carry flag. Set/cleared by hardware or software during certain arithmetic and logical instructions.
AC	PSW.6	Auxiliary Carry flag. Set/cleared by hardware during addition or subtraction instructions to indicate carry or borrow out of bit 3.
F0	PSW.5	Flag 0 Set/cleared/tested by software as a user-defined status flag.
RS1	PSW.4	Register bank Select control bits 1 & 0. Set/cleared by software to determine working register bank (see Note).
RS0	PSW.3	
OV	PSW.2	Overflow flag. Set/cleared by hardware during arithmetic instructions to indicate overflow conditions.
—	PSW.1	(reserved)
P	PSW.0	Parity flag. Set/cleared by hardware each instruction cycle to indicate an odd/even number of "one" bits in the accumulator, i.e., even parity.

Note— the contents of (RS1, RS0) enable the working register banks as follows:

(0.0)—Bank 0	(00H-07H)
(0.1)—Bank 1	(08H-0FH)
(1.0)—Bank 2	(10H-17H)
(1.1)—Bank 3	(18H-1FH)

Figure 6-2. PSW: Program Status Word Register

To drive HMOS versions of the 8051 externally, XTAL1 should be grounded, while XTAL2 is driven, (see Figure 6-3B). For CHMOS versions, XTAL1 is driven, while XTAL2 is floated, as shown in Figure 6-3C. There are no requirements on the duty cycle of the external oscillator signal, as the input to the clock generator is a divide-by-two flip-flop. Minimum high and low times (20 ns), however, must be observed.

The clock generator divides the oscillator frequency by 2, to provide a two phase clock signal to the chip. The Phase 1 signal is active during the first half of each clock period, and the Phase 2 signal is active during the second half of each clock period. XTAL2 is the input to this clock generator.

6.3 CPU TIMING

A machine cycle consists of 6 states (12 oscillator periods). Each state is divided into a Phase 1 half, during which the Phase 1 clock is active, and a Phase 2 half, during which the Phase 2 clock is active. Thus, a machine cycle consists of 12 oscillator periods, numbered S1P1 (State 1, Phase 1), through S6P2 (State 6, Phase 2). Each phase lasts for one oscillator period. Each state lasts for two oscillator periods. Typically, arithmetic and logical operations take place during Phase 1 and internal register-to-register transfers take place during Phase 2.

The diagrams in Figure 6-4 show the fetch/execute timing referred to the internal states and phases. Since these internal clock signals are not user accessible, the XTAL2 oscillator signal and the ALE (Address Latch Enable) signal are shown for external reference. ALE is normally activated twice during each machine cycle: once during S1P2 and S2P1, and again during S4P2 and S5P1.

Execution of a one-cycle instruction begins at S1P2, when the opcode is latched into the Instruction Register. If it is a two-byte instruction, the second byte is read during S4 of the same machine cycle. If it is a one-byte instruction, there is still a fetch at S4, but the byte read (which would be the next opcode), is ignored, and the Program Counter is not incremented. In any case, execution is complete at the end of S6P2. Figures 6-4A and 6-4B show the timing for a 1-byte, 1-cycle instruction and for a 2-byte, 1-cycle instruction.

Most 8051 instructions execute in one cycle. MUL (multiply) and DIV (divide) are the only instructions that take more than two cycles to complete, they take four cycles.

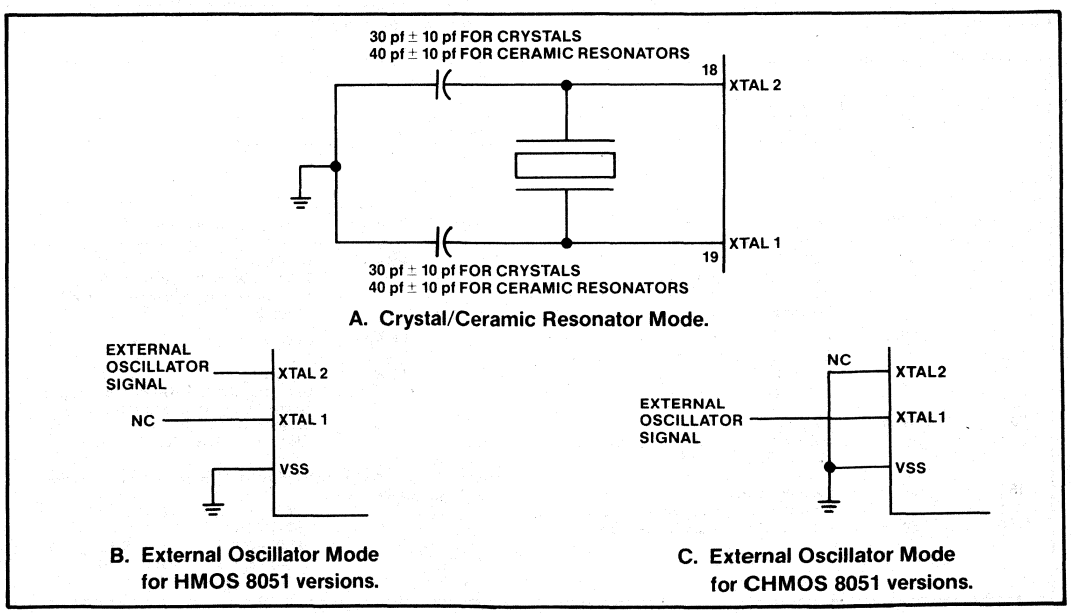


Figure 6-3. Crystal/Ceramic Resonator and External Oscillator

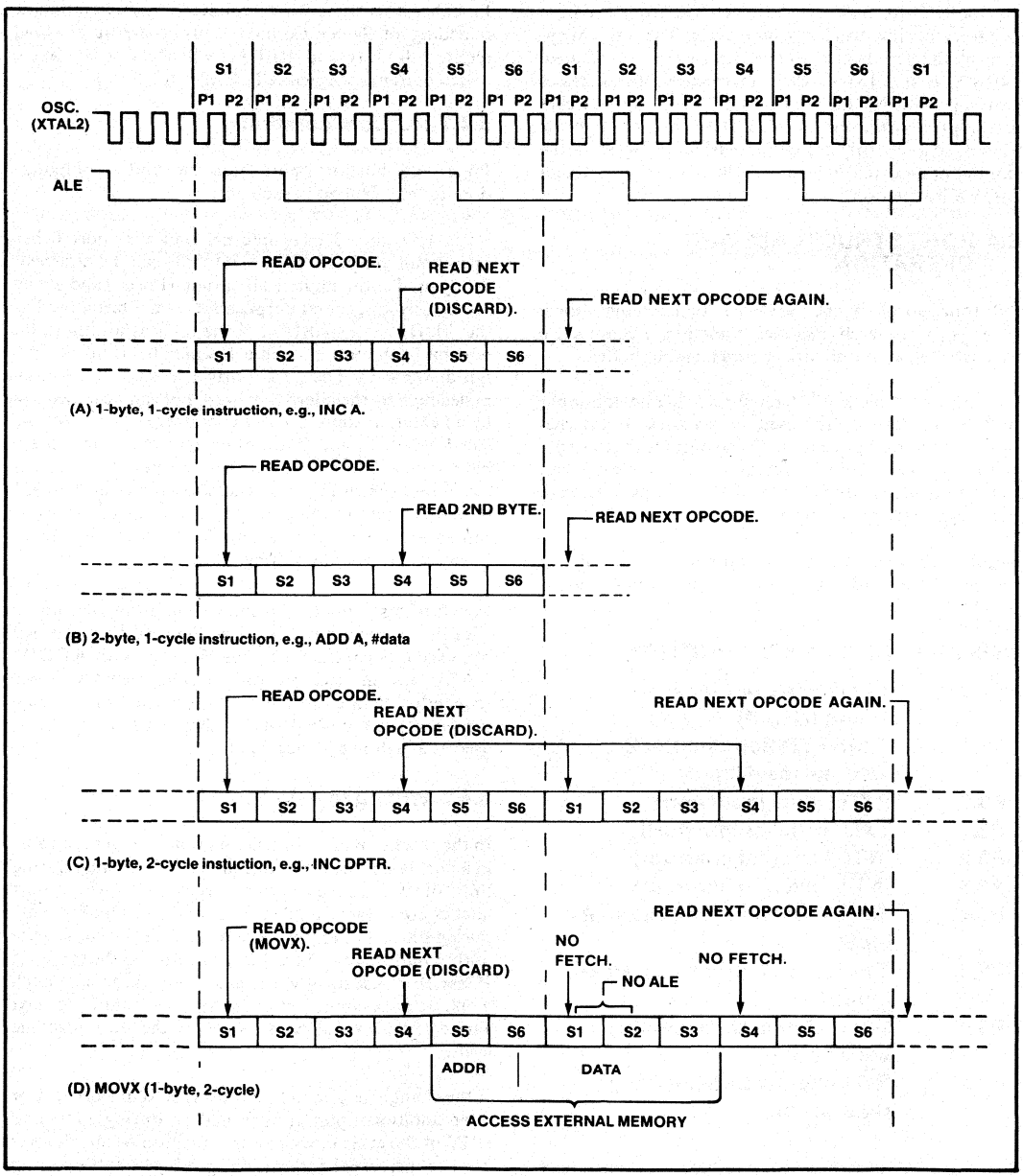


Figure 6-4. 8051 Fetch/Execute Sequences

Normally, two code bytes are fetched from Program Memory during every machine cycle. The only exception to this is when a MOVX instruction is executed. MOVX is a 1-byte 2-cycle instruction that accesses external Data Memory. During a MOVX, two fetches are skipped while the external Data Memory is being addressed and strobed. Figures 6-4C and 6-4D show the timing for a normal 1-byte, 2-cycle instruction and for a MOVX instruction.

6.4 PORT STRUCTURES AND OPERATION

All four ports in the 8051 are bidirectional. Each consists of a latch (Special Function Registers P0 through P3), an output driver, and an input buffer.

The output drivers of Ports 0 and 2, and the input buffers of Port 0, are used in accesses to external memory. In this application, Port0 outputs the low byte of the external memory address, time-multiplexed with the byte being written or read. Port 2 outputs the high byte of the external memory address.

Some of the output drivers and input buffers of Port 1 and all of those of Port 3 are also multifunctional, as listed below:

PORT PIN	ALTERNATE FUNCTION
*P1.0	T2 (Timer/Counter 2 external input)
*P1.1	T2RST (Timer/Counter 2 external reset input)
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	$\overline{INT0}$ (external interrupt)
P3.3	$\overline{INT1}$ (external interrupt)
P3.4	T0 (Timer/Counter 0 external input)
P3.5	T1 (Timer/Counter 1 external input)
P3.6	$\overline{WR}$ (external Data memory write strobe)
P3.7	$\overline{RD}$ (external Data memory read strobe)

The Timer/Counter2 related alternate functions, those asterisked (*) are applicable to the 8032/8052 only. The output latch corresponding to a secondary function must be programmed to a one (1) for that function to operate.

In either the "timer" or "counter" mode, automatic reloading of Timer/Counter 2 from the auto-reload registers RLDH and RLDL, will occur if the auto-reload option is selected ($CP/\overline{RL2} = 0$).

6.4.1 I/O Configurations

Figure 6-5 illustrates port latch and buffer configurations for a typical bit in each port.

Ports 1, 2, and 3 have internal pull-ups. Port 0 has open-drain outputs. Each I/O line can be independently used as an input or an output. (Ports 0 and 2 may not be used as general purpose I/O when being used as the ADDR/DATA BUS). To be used as an input, the port bit latch must contain a 1, which turns off the output driver FET. Then, for Ports 1, 2, and 3, the pin is pulled high by the internal pull-up, but can be pulled low by an external source. These characteristics are the reason Ports 1, 2, and 3 are often referred to as "quasi-bidirectional". For Port 0, a 1 in the port latch causes the output pin to float. All the port latches in the 8051 have 1s written to them by the reset function. If a 0 is subsequently written to a port latch, it can be reconfigured as an input by writing a 1 to it.

Port 0 differs in not having internal pull-ups. The upper FET in the P0 output driver (see Figure 6-5A) is turned off, except when the port is being used as an ADDR/DATA bus in accesses to external memory. Consequently, P0 latch results in both output FETs being turned off, so the pin floats. In that condition it can be used as a high-impedence input.

6.4.2 Writing to a Port

In the execution of an instruction that changes the value in a port latch, the new value arrives at the latch during S6P2 of the final cycle of the instruction. However, port latches are in fact sampled by their output buffers only during Phase 1 of any clock period. (During Phase 2 the output buffer holds the value it saw during the previous Phase 1). Consequently, the new value in the port latch won't actually appear at the output pin until the next Phase 1, which will be at S1P1 of the next machine cycle.

If the change requires a 0-to-1 transition in Port 1, 2, or 3, an additional pull-up is turned on during S1P1 and S1P2 of the cycle in which the transition occurs. This is done to increase the transition speed. The extra pull-up can source about 100 times the current that the normal pull-up can. It should be noted that the internal pull-ups are field-effect transistors, not linear resistors. The pull-up arrangements are shown in Figure 6-6.

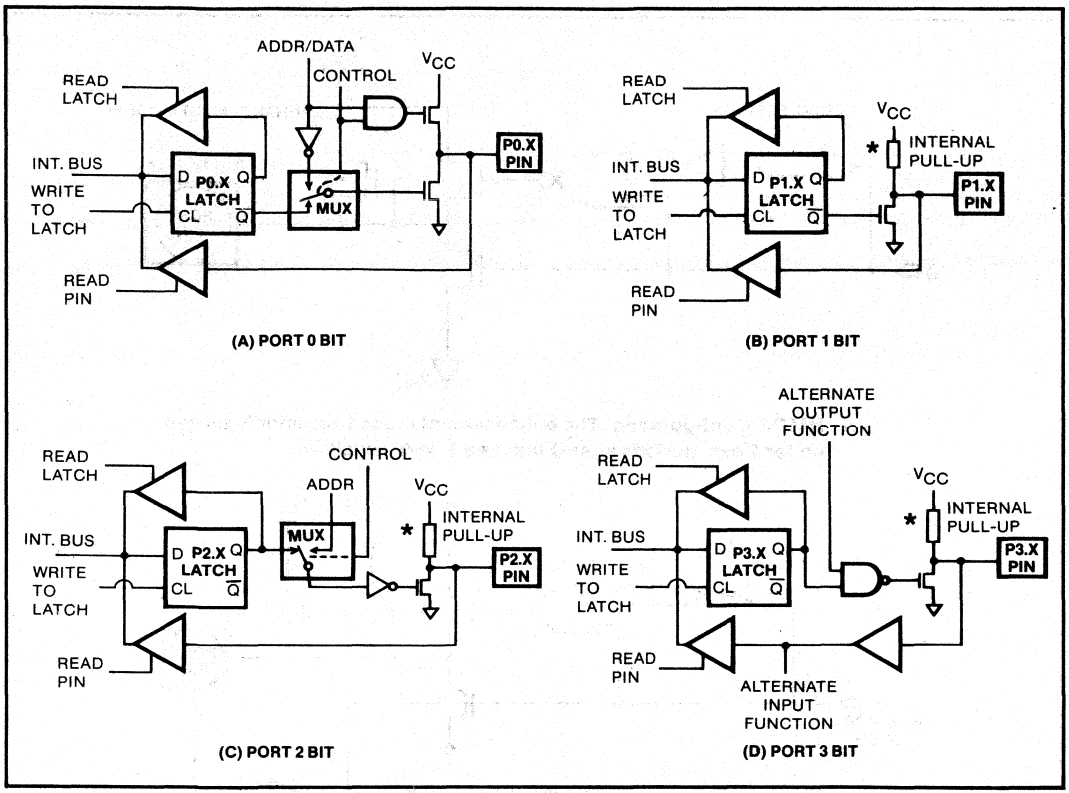


Figure 6-5. 8051 Port Latches and Buffers

In HMOS versions of the 8051, the fixed part of the pull-up is a depletion-mode transistor with the gate wired to the source. This transistor will allow the pin to source about 0.25 mA when shorted to ground. In parallel with the fixed pull-up is an enhancement-mode transistor, which is activated during S1 whenever the port bit does a 0-to-1 transition. During this interval, if the port pin is shorted to ground, this extra transistor will allow the pin to source an additional 30 mA.

In the CHMOS versions, the pull-up consists of three pFETs. It should be noted that an n-channel FET (nFET) is turned on when a logical 1 is applied to its gate electrode, and is turned off when a logical 0 is applied to its gate electrode. A p-channel FET (pFET) is the opposite: it is on when its gate sees a 0, and off when its gate sees a 1.

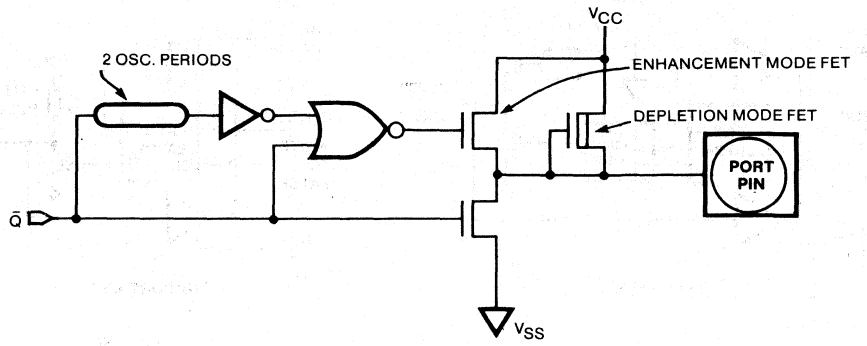
pFET 1 in Figure 6-6B is the transistor that is turned on for 2 oscillator periods after a 0-to-1 transition in the port latch. While it's on, it turns on pFET 3 (a weak pull-up), through the inverter. This inverter and pFET form a latch which hold the 1. pFET 2 is also on, the nFET is off.

6.4.3 Port Loading and Interfacing

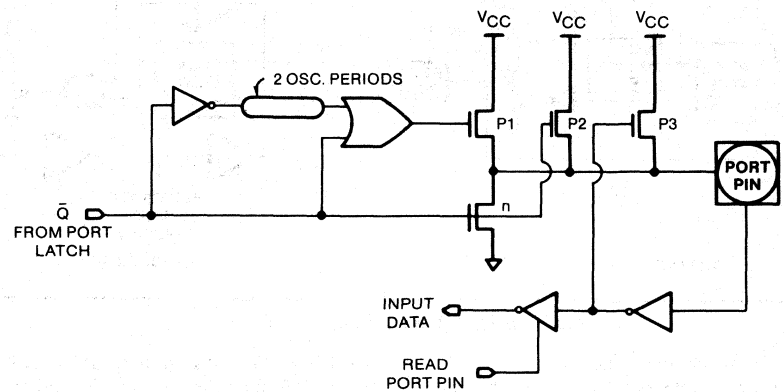
The output buffers of Ports 1, 2, and 3 can each drive 3 LS TTL inputs. These ports on HMOS versions can be driven in a normal manner by any TTL or NMOS circuit. Both HMOS and CHMOS versions can be driven by open-collector and open-drain outputs, without the need for external pull-ups.

Port 0 output buffers can each drive 8 LS TTL inputs. They do, however, require external pull-ups to drive NMOS inputs, except when being used as the ADDRESS/DATA bus.

CHMOS design interface techniques are discussed in Chapter 10.



A. HMOS Configuration. The enhancement mode transistor is turned on for 2 osc. periods after Q makes a 1-to-0 transition.



B. CHMOS Configuration. pFET 1 is turned on for 2 osc. periods after Q makes a 1-to-0 transition. During this time, pFET 1 also turns on pFET 3 through the inverter to form a latch which holds the 1. pFET 2 is also on.

Figure 6-6. Ports 1, 2 and 3 HMOS and CHMOS Internal Pull-up Configurations

6.4.4 Read-Modify-Write Feature

There are two ways to read a port: an instruction reads either the latch or the pin, (refer to Figure 6-5). In the 8051, some instructions read the latch and some read the pin. The instructions that read the latch rather than the pin are the ones that read a value, possibly change it, and then rewrite it to the latch. These are called "read-modify-write" instructions. The instructions listed below are read-modify-write instructions. When the destination operand is a port, or a port bit, these instructions read the latch rather than the pin:

ANL	**(logical AND, e.g., ANL P1,A)**
ORL	**(logical OR, e.g., ORL P2,A)**
XRL	**(logical EX-OR, e.g., XRL P3,A)**
JBC	**(jump if bit = 1 and clear bit, e.g., JBC P1.1, LABEL)**
CPL	**(complement bit, e.g., CPL P3.0)**

INC (increment, e.g., INC P2)
DEC (decrement, e.g., DEC P2)
DJNZ (decrement and jump if not zero, e.g., DJNZ P3, LABEL)
MOV PX.Y,C (move carry bit to bit Y of Port X)
CLR PX.Y (clear bit Y of Port X)
SET PX.Y (set bit Y of Port X)

It is not obvious that the last three instructions in this list are read-modify-write instructions: They read the port byte, all 8 bits, modify the addressed bit, then write the new byte back to the latch.

The reason that read-modify-write instructions are directed to the latch rather than the pin is to avoid a possible misinterpretation of the voltage level at the pin. For example, a port bit might be used to drive the base of a transistor. When a 1 is written to the bit, the transistor is turned on. If the CPU then reads the same port bit at the pin rather than the latch, it will read the base voltage of the transistor and interpret it as a 0. Reading the latch rather than the pin will return the correct value of 1.

6.5 ACCESSING EXTERNAL MEMORY

Accesses to external memory are of two types: accesses to external Program Memory and accesses to external Data Memory. Accesses to external Program Memory use signal $\overline{PSEN}$ (program store enable) as the read strobe. Accesses to external Data Memory use $\overline{RD}$ or $\overline{WR}$ (alternate functions of P3.7 and P3.6) to strobe the memory.

Fetches from external Program Memory always use a 16-bit address. Accesses to external Data Memory can use either a 16-bit address (MOVX @DPTR) or an 8-bit address (MOVX @Ri).

Whenever a 16-bit address is used, the high byte of the address comes out on Port 2, where it is held for the duration of the read or write cycle. During this time the Port 2 latch (the Special Function Register) does not have to contain 1s, and the contents of the Port 2 SFR are not modified. If the external memory cycle is not immediately followed by another external memory cycle, the undisturbed contents of the Port 2 SFR will reappear in the next cycle.

If an 8-bit address is being used (MOVX @Ri), the contents of the Port 2 SFR remain at the Port 2-pins throughout the external memory cycle. This will facilitate paging.

In any case, the low byte of the address is time-multiplexed with the data byte on Port 0. The ADDR/DATA signal drives both FETs in the Port 0 output buffers. Thus, in this application the Port 0 pins are not open-drain outputs, and hence, do not require external pull-ups. Signal ALE (address latch enable) should be used to capture the address byte into an external latch. The address byte is valid at the negative transition of ALE. Then, in a write cycle, the data byte to be written appears on Port 0 just before $\overline{WR}$ is activated, and remains there until after $\overline{WR}$ is deactivated. In a read cycle, the incoming byte is accepted at Port 0 just before the read strobe is deactivated.

During any access to external memory, the CPU writes 0FFH to the Port 0 latch (the Special Function Register), thus obliterating whatever information the Port 0 SFR may have been holding.

External Program Memory is accessed under two conditions:

1) Whenever signal $\overline{EA}$ is active; or
2) Whenever the program counter (PC) contains a number that is larger than 0FFH (1FFH for the 8052).

This requires that the ROMless versions have $\overline{EA}$ wired low to enable the lower 4K (8K for the 8032) program bytes to be fetched from external memory.

When the CPU is executing out of external Program Memory, all 8 bits of Port 2 are dedicated to an output function and may not be used for general purpose I/O. During external program fetches they output the high byte of the PC, and during accesses to external Data Memory they output either DPH or the Port 2 SFR (depending on whether the external Data Memory access is a MOVX @DPTR or a MOVX @Ri).

6.5.1 $\overline{PSEN}$

The read strobe for external fetches is $\overline{PSEN}$. $\overline{PSEN}$ is not activated for internal fetches. When the CPU is accessing external Program Memory, $\overline{PSEN}$ is activated twice every cycle (except during a MOVX instruction) whether or not the byte fetched is actually needed for the current instruction. When $\overline{PSEN}$ is activated its timing is not the same as $\overline{RD}$. A complete $\overline{RD}$ cycle, including activation and deactivation of ALE and $\overline{RD}$, takes 12 oscillator periods. A complete $\overline{PSEN}$ cycle, including activation and deactivation of ALE and $\overline{PSEN}$, takes 6 oscillator periods. The execution sequence for these two types of read cycles are shown in Figure 6-7 for comparison.

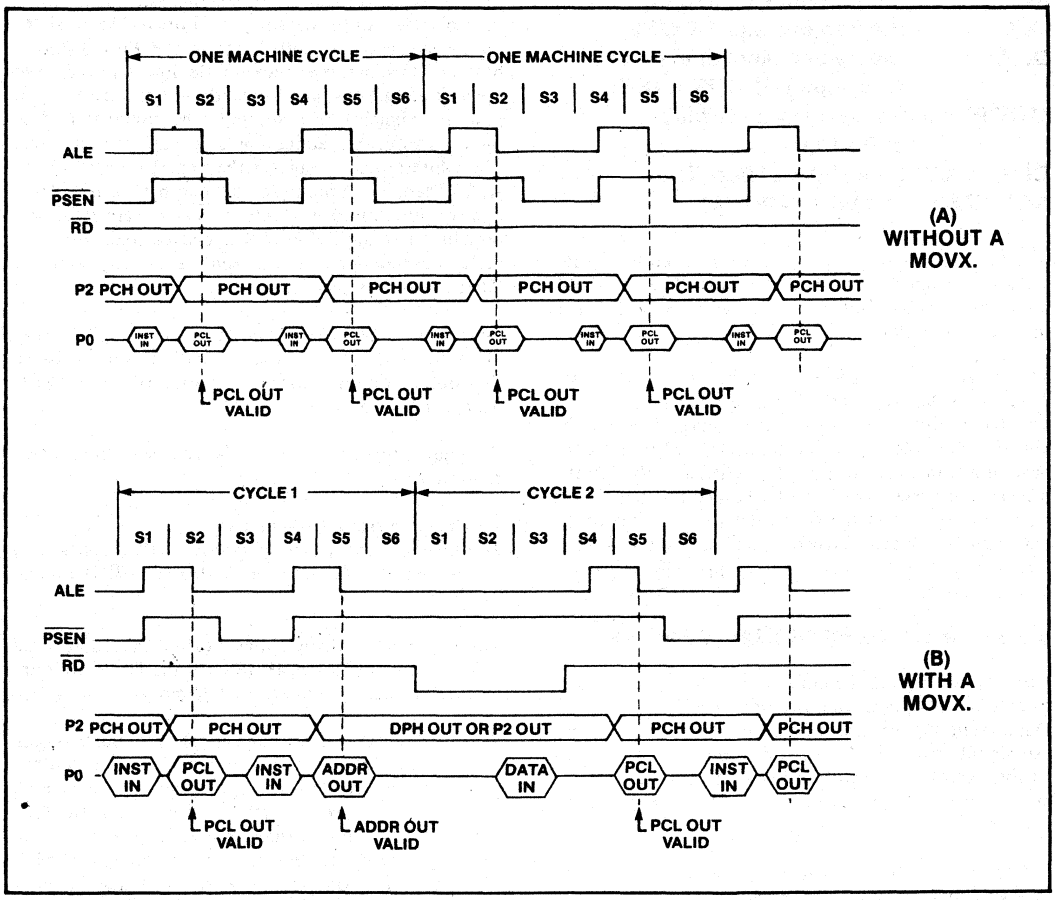


Figure 6-7. External Program Memory Execution

6.5.2 ALE

The main function of ALE is to provide a properly timed signal to latch the low byte of an address from P0 to an external latch during fetches from external is activated Program Memory. For that purpose ALE twice every machine cycle. This activation takes place even when the cycle involves no external fetch. The only time an ALE pulse doesn't come out is during an access to external Data Memory. The first ALE of the second cycle of a MOVX instruction is missing (see Figure 6-7). Consequently, in any system that does not use external Data Memory, ALE is activated at a constant rate of 1/6 the oscillator frequency, and can be used for external clocking or timing purposes.

6.5.3 Overlapping External Program and Data Memory Spaces

In some applications it is desirable to execute a program from the same physical memory that is being used to store data. In the 8051, the external Program and Data Memory spaces can be combined by ANDing $\overline{PSEN}$ and $\overline{RD}$. A positive-logic AND of these two signals produces an active-low read strobe that can be used for the combined physical memory. Since the $\overline{PSEN}$ cycle is faster than the $\overline{RD}$ cycle, the external memory needs to be fast enough to accommodate the $\overline{PSEN}$ cycle.

6.6 TIMER/COUNTERS

The 8051 has two 16-bit timer/counters, Timer/Counter 0 and Timer/Counter 1. The 8032/8052 has these two timer/counters plus one: Timer/Counter 2. While each timer/counter can be configured as either a timer or event counter, Timer/Counter 2 has capabilities over Timer/Counters 0 and 1 which are described in Section 6.6.2.

6.6.1 Timer/Counters 0 and 1

Timer/Counter 0 and 1 each have a control bit in SFR TMOD that selects the timer/counter to be "timer" or "counter".

In the "timer" function, the register is incremented every machine cycle. Thus, one can think of it as counting machine cycles. Since a machine cycle consists of 12 oscillator periods, the count rate is 1/12 of the oscillator frequency.

In the "counter" function, the register is incremented in response to a 1-to-0 transition at its corresponding external input pin, T0 or T1. In this function, the external input is sampled during S5P2 of every machine cycle. When the samples show a high in one cycle and a low in the next cycle, the count is incremented. The new count value appears in the register during S3P1 of the cycle following the one in which the transition was detected. Since it takes 2 machine cycles (24 oscillator periods) to recognize a 1-to-0 transition, the maximum count rate is 1/24 of the oscillator frequency. There are no restrictions on the duty cycle of the external input signal, but to insure that a given level is sampled at least once before it changes, it should be held for at least one full machine cycle.

In addition to the "timer" or "counter" selection, each timer/counter has four operating modes. Modes 0, 1, and 2 are the same for both timer/counters. Mode 3 is different.

MODE 0

Putting either Timer/Counter 0 or 1 into mode 0 makes it look like an 8048AH Timer, which is an 8-bit counter with a divide-by-32 prescaler. Figure 6-8 shows the mode 0 operation as it applies to Timer/Counter 1.

In this mode, the timer register is configured as a 13-bit register. As the count rolls over from all 1s to all 0s, it sets the timer interrupt flag TF1. The counted input is enabled to the Timer/Counter when TR1 = 1 and either GATE = 0 or $\overline{INT1}$ = 1. (Setting GATE = 1 allows the Timer to be controlled by external input $\overline{INT1}$, to facilitate pulse width measurements). TR1 is a control bit in Special Function Register TCON. GATE is a control bit in Special Function Register TMOD.

The 13-bit register consists of all 8 bits of TH1 and the lower 5 bits of TL1. The upper 3 bits of TL1 are indeterminate and should be ignored. Setting the run flag (TR1) does not clear the registers.

Mode 0 operation is the same for Timer Counter 0 as for Timer/Counter 1. Substitute TR0, TF0 and $\overline{INT0}$ for the corresponding Timer/Counter 1 signals in Figure 6-8. There are two different GATE bits, one for Timer/Counter 1 (TMOD.7) and one for Timer/Counter 0 (TMOD.3).

MODE 1

Mode 1 is the same as Mode 0, except that the Timer register is being run with all 16 bits.

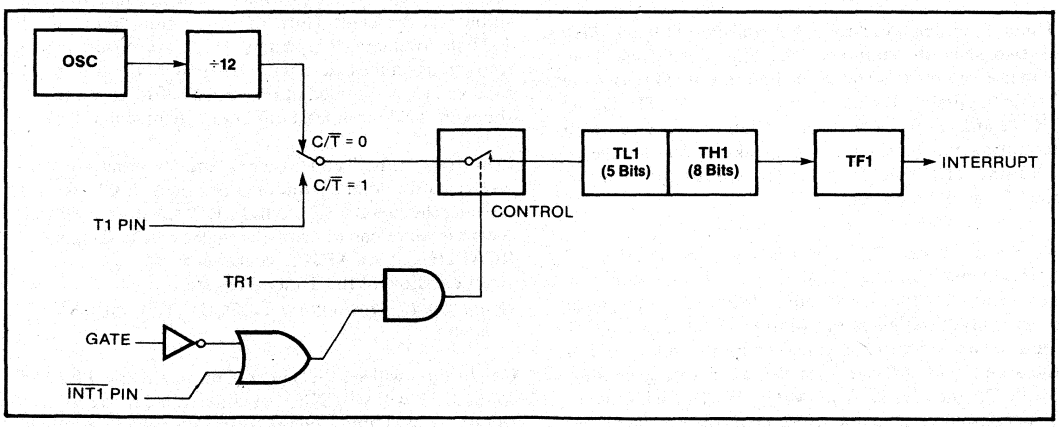


Figure 6-8. Timer/Counter 1 Mode 0: 13-bit Counter

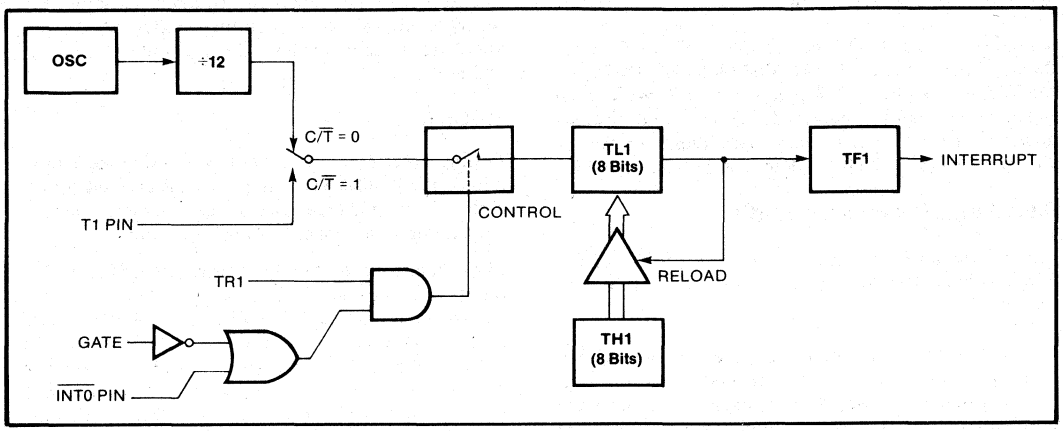


Figure 6-9. Timer/Counter 1 Mode 2: 8-bit Auto-reload

MODE 2

Mode 2 configures the timer register as an 8-bit counter (TL1) with automatic reload, as shown in Figure 6-9. Overflow from TL1 not only sets TF1, but also reloads TL1 with the contents of TH1, which is preset by software. The reload leaves TH1 unchanged.

Mode 2 operation is the same for Timer/Counter 0.

MODE 3

Timer/Counter 1 in Mode 3 holds its count. The effect is the same as setting TR1 = 0.

Timer/Counter 0 in Mode 3 establishes TL0 and TH0 as two separate counters. The logic for Mode 3 on Timer/Counter 0 is shown in Figure 6-9. TL0 uses the Timer/Counter 0 control bits: C/$\overline{T}$, GATE, TR0, $\overline{INT0}$, and TF0. TH0 is locked into a timer function (counting machine cycles) and takes over the use of TR1 and TF1 from Timer 1. Thus, TH0 now controls the "Timer 1" interrupt.

Normally, Timer/Counter 0 is not used in Mode 3 unless Timer/Counter 1 is already in use as a baud rate generator for the serial port. Mode 3 is provided specifically for applications (non 8032/8052 based) that require two independent timer/counters and the use of the serial port. (Timer 1 as the baud rate generator, Mode 2, and Timer 0 in Mode 3). The addition of Timer/Counter 2 in the 8032/8052 further facilitates design of such applications.

6.6.2 Timer/Counter 2

Timer/Counter 2 is a 16-bit timer/counter with 16-bit auto-reload and capture capability. The control register is SFR T2CON.

When Timer/Counter 2 operates as a "timer," the Timer/Counter 2 register is incremented once per machine cycle. As a "counter", Timer/Counter 2 is incremented in response to a 1-to-0 transition at its external input pin T2EX (P1.0). The input is sampled during S5P2 of each machine cycle in which the counter function is active. Therefore, when the sample shows a high in one cycle and a low in the next cycle, the count is incremented. The new count value appears in the Timer/Counter 2 register during S3P1 of the cycle following the one in which the transition was detected. Thus, the maximum count rate is 1/24 the oscillator frequency. There are no duty cycle restrictions on the external input signal, but to insure that a given level is sampled at least once before it changes, it should be held for at least one complete machine cycle.

In both the "timer" and "counter" modes, either the capture or auto-reload option may be selected. CP/$\overline{RL2}$=1 activates the capture feature and CP/$\overline{RL2}$=0, auto-reload. A capture or reload to/from the capture/reload registers, RCAP2H and RCAP2L, occurs with:

1) an overflow of the Timer 2 register; or
2) a negative transition on T2EX (P1.1) when EXEN2 is set.

Condition 1 will set the Timer/Counter 2 flag TF2 and condition 2 will set EXF2. In either case, if the Timer/Counter 2 interrupt is enabled, an interrupt request will be generated. The interrupt vector location is 2BH. Both

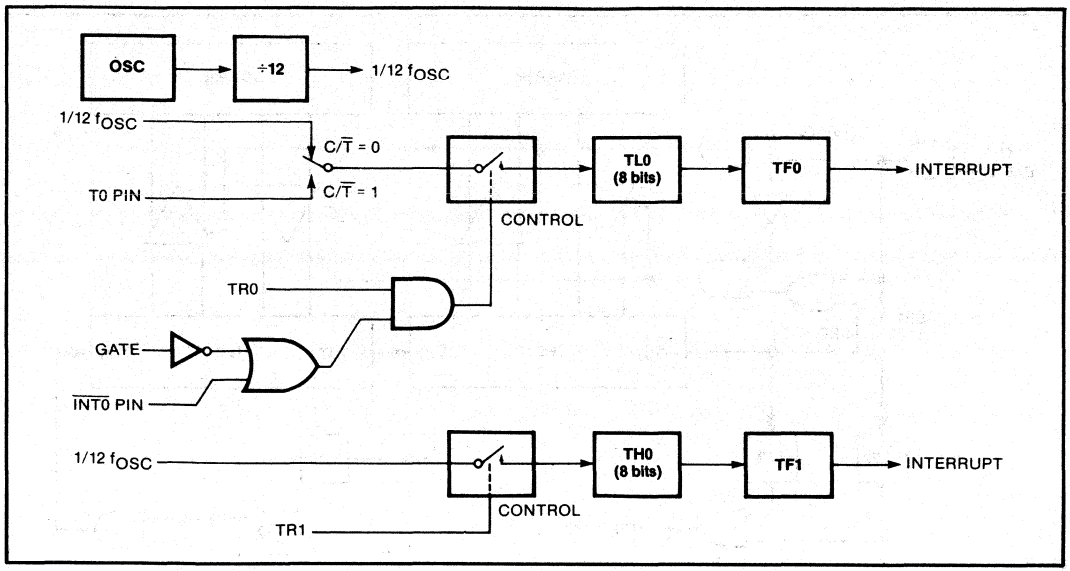


Figure 6-10. Timer/Counter 0 Mode 3: Two 8-bit Counters

TF2 and EXF2 must be cleared by software. Figure 6-11 illustrates Timer/Counter 2 operation in the "timer" and "counter" modes.

Timer/Counter 2 can also be used as a baud rate clock source in serial port modes 1 and 3 by setting RCLK or TCLK. (RCLK=TCLK=0 selects Timer/Counter 1 as the baud rate generator.) In this mode, Timer/Counter 2 overflow pulses, rather than Timer/Counter 1's, clock the serial port. If C/$\overline{T2}$=0, the counter increments at 1/2 the oscillator frequency. If C/$\overline{T2}$=1, the counter increments in response to 1-to-0 transitions on T2EX (P1.0). (The maximum count frequency is 1/24 the oscillator frequency.)

If EXEN2 is set while Timer/Counter 2 is being used to clock the serial port, a negative transition on T2EX, P1.1, will set EXF2, but no reload or capture will occur. Consequently, while Timer/Counter 2 is in the baud rate generator mode, T2EX may be used as an additional external interrupt input. TF2 is not affected by timer overflows. Automatic reloading from RCAP2H and RCAP2L will occur regardless of the state of CP/$\overline{RL2}$.

TH2 and TL2 cannot be written or read without causing error while Timer/Counter 2 is in the baud rate generator mode. RCAP2H and RCAP2L also may not be written in this mode.

6.6.3 Timer/Counter Control and Status Registers

The Special Function Registers TMOD, TCON and T2CON (8032/8052 only), are used to define the operating modes and control the functions of the timer/counters. When an instruction changes the content of TMOD, TCON, or T2CON, the change is latched into the SFR and takes effect at S1P1 of the next instruction's first cycle. The registers are shown in Figures 6-13, 6-14 and 6-15, respectively. All the bits of each register are cleared by reset.

6.7 SERIAL INTERFACE

The serial port is full duplex, meaning it can transmit and receive simultaneously. It is also receive-buffered, meaning it can commence reception of a second byte before a previously received byte has been read from the receive register. (However, if the first byte still hasn't been read by the time reception of the second byte is complete, one of the bytes will be lost). The serial port receive and transmit registers are both accessed at Special Function Register SBUF, although they are physically separate.

SCON is the SFR that is used in determining the operatoring mode of the serial port. It receives the 9th data bit (RB8), and contains other status flags. Figure 6-16 summarizes the register.

The serial port can operate in 4 modes, as indicated in Table 2.

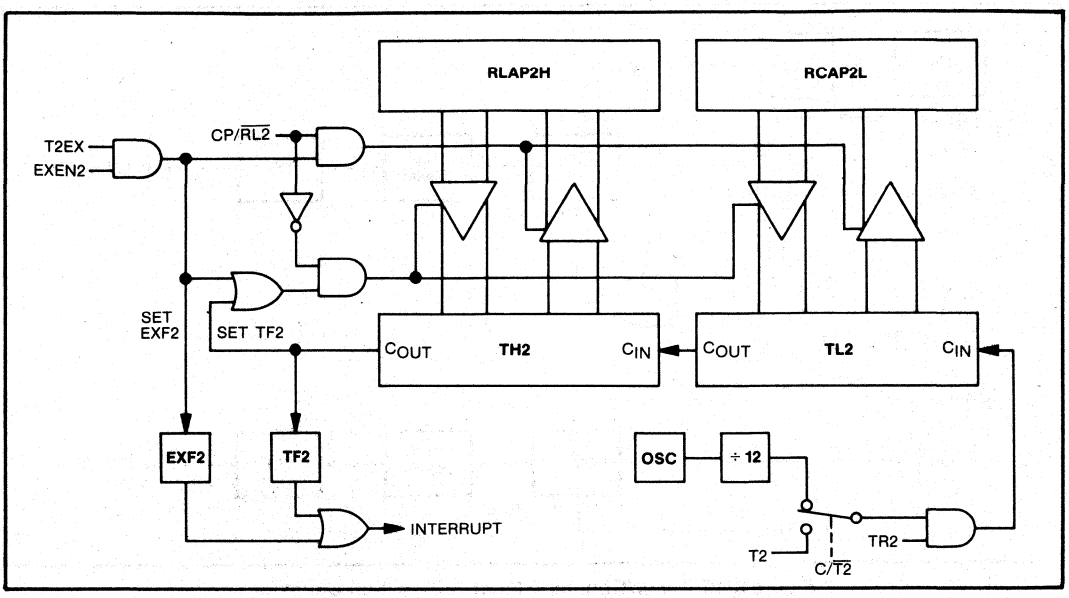


Figure 6-11. Timer/Counter 2 in Timer/Counter Mode (RCLK = TCLK = 0)

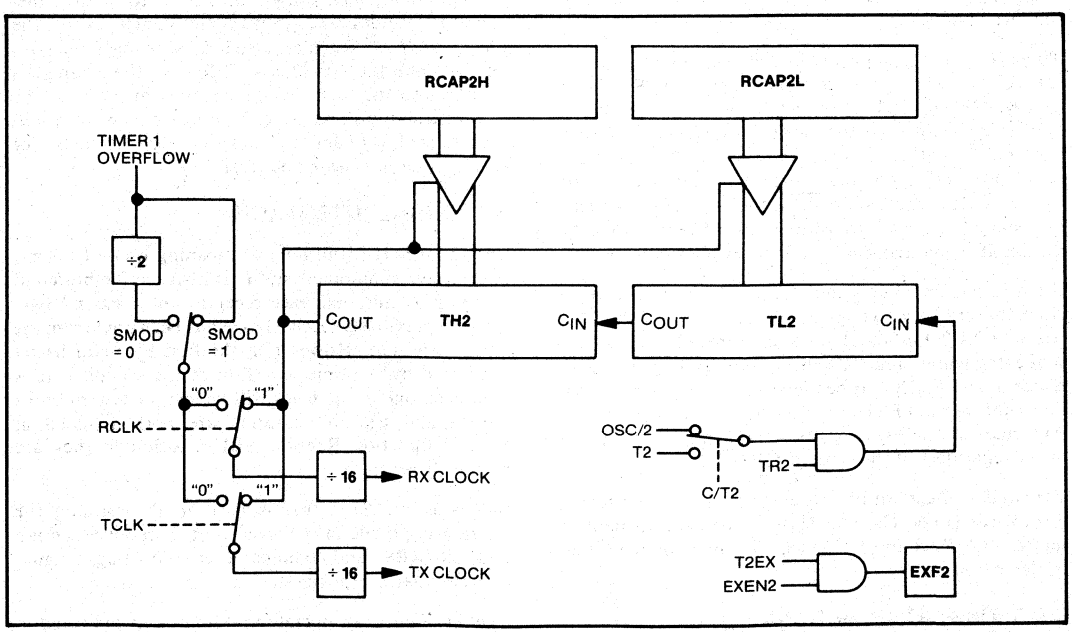
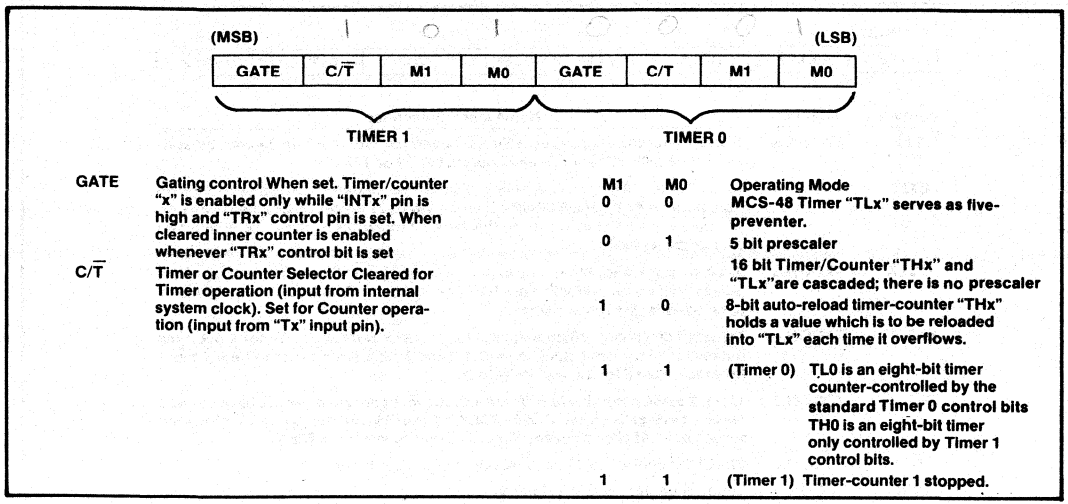


Figure 6-12. Timer/Counter 2 in Baud Rate Generator Mode (RCLK + TCLK = 1)

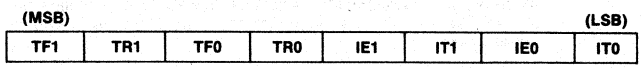


GATE Gating control When set. Timer/counter "x" is enabled only while "INTx" pin is high and "TRx" control pin is set. When cleared inner counter is enabled whenever "TRx" control bit is set

C/T Timer or Counter Selector Cleared for Timer operation (input from internal system clock). Set for Counter operation (input from "Tx" input pin).

M1	M0	Operating Mode
0	0	MCS-48 Timer "TLx" serves as five-preventer.
0	1	5 bit prescaler
		16 bit Timer/Counter "THx" and "TLx"are cascaded; there is no prescaler
1	0	8-bit auto-reload timer-counter "THx" holds a value which is to be reloaded into "TLx" each time it overflows.
1	1	(Timer 0) TL0 is an eight-bit timer counter-controlled by the standard Timer 0 control bits TH0 is an eight-bit timer only controlled by Timer 1 control bits.
1	1	(Timer 1) Timer-counter 1 stopped.

Figure 6-13. TMOD: Timer/Counter Mode Control Register



Symbol	Position	Name and Significance	Symbol	Position	Name and Significance
TF1	TCON.7	Timer 1 overflow Flag. Set by hardware on timer/counter overflow. Cleared when interrupt processed.	IE1	TCON.3	Interrupt 1 Edge flag. Set by hardware when external interrupt edge detected. Cleared when interrupt processed.
TR1	TCON.6	Timer 1 Run control bit. Set/cleared by software to turn timer/counter on/off.	IT1	TCON.2	Interrupt 1 Type control bit. Set/cleared by software to specify falling edge/low level triggered external interrupts.
TF0	TCON.5	Timer 0 overflow Flag. Set by hardware on timer/counter overflow. Cleared when interrupt processed.	IE0	TCON.1	Interrupt 0 Edge flag. Set by hardware when external interrupt edge detected. Cleared when interrupt processed.
TR0	TCON.4	Timer 0 Run control bit. Set/cleared by software to turn timer/counter on/off.	IT0	TCON.0	Interrupt 0 Type control bit. Set/cleared by software to specify falling edge/low level triggered external interrupts.

Figure 6-14. TCON: Timer/Counter Control Register

(MSB)							(LSB)
TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/$\overline{T2}$	CP/$\overline{RL2}$

Symbol	Position	Name and Significance
TF2	T2CON.7	Timer 2 overflow flag set by a Timer 2 overflow and must be cleared by software. TF2 will not be set when either RCLK = 1 or TCLK = 1.
EXF2	T2CON.6	Timer 2 external flag set when either a capture or reload is caused by a negative transition on T2EX and EXEN2 = 1. When Timer 2 interrupt is enabled, EXF2 = 1 will cause the CPU to vector to the Timer 2 interrupt routine. EXF2 must be cleared by software.
RCLK	T2CON.5	Receive clock flag. When set, causes the serial port to use Timer 2 overflow pulses for its receive clock in modes 1 and 3. RCLK = 0 causes Timer 1 overflow to be used for the receive clock.
EXEN2	T2CON.4	Transmit clock flag. When set, causes the serial port to use Timer 2 overflow pulses for its transmit clock in modes 1 and 3. TCLK = 0 causes Timer 1 overflows to be used for the transmit clock.
TCLK	T2CON.3	Timer 2 external enable flag. When set, allows a capture or reload to occur as a result of a negative transition on T2EX if Timer 2 is not being used to clock the serial port. EXEN2 = 0 causes Timer 2 to ignore events at T2EX.
TR2	T2CON.2	Start/stop control for Timer 2. A logic 1 starts the timer.
C/$\overline{T2}$	T2CON.1	Timer or counter select. (Timer 2) 0 = Internal timer (OSC/12) 1 = External event counter (falling edge triggered).
CP/$\overline{RL2}$	T2CON.0	Capture/Reload flag. When set, captures will occur on negative transitions at T2EX if EXEN2 = 1. When cleared, auto reloads will occur either with Timer 2 overflows or negative transitions at T2EX when EXEN2 = 1. When either RCLK = 1 or TCLK = 1, this bit is ignored and the timer is forced to auto-reload on Timer 2 overflow.

Figure 6-15. T2CON: Timer/Counter 2 Control Register

(MSB)							(LSB)
SM0	SM1	SM2	REN	TB8	RB8	TI	RI

where SM0, SM1 specify the serial port mode, as follows:

SM0	SM1	Mode	Description	Baud Rate
0	0	0	shift register	$f_{osc.}/12$
0	1	1	8-bit UART	variable
1	0	2	9-bit UART	$f_{osc.}/64$ or $f_{osc.}/32$

- SM2 enables the multiprocessor communication feature in modes 2 and 3. In mode 2 or 3, if SM2 is set to 1 then RI will not be activated if the received 9th data bit (RB8) is 0. In mode 1, if SM2 = 1 then RI will not be activated if a valid stop bit was not received. In mode 0, SM2 should be 0.
- REN enables serial reception. Set by software to enable reception. Clear by software to disable reception.
- TB8 is the 9th data bit that will be transmitted in modes 2 and 3. Set or clear by software as desired.
- RB8 in modes 2 and 3, is the 9th data bit that was received. In mode 1, if SM2 = 0, RB8 is the stop bit that was received. In mode 0, RB8 is not used.
- TI is transmit interrupt flag. Set by hardware at the end of the 8th bit time in mode 0, or at the beginning of the stop bit in the other modes, in any serial transmission. Must be cleared by software.
- RI is receive interrupt flag. Set by hardware at the end of the 8th bit time in mode 0, or halfway through the stop bit time in the other modes, in any serial reception (except see SM2). Must be cleared by software.

Figure 6-16. SCON: Serial Port Control Register

6.7.1 Mode 0

Serial data enters and exits through RXD. TXD outputs the shift clock. 8 bits are transmitted/received: 8 data bits (LSB first). The baud rate is fixed at 1/12 the oscillator frequency.

Figure 6-17 shows a simplified functional diagram of the serial port in mode 0, and associated timing.

Transmission is initiated by any instruction that uses SBUF as a destination register. The "write to SBUF" signal at S6P2 also loads a 1 into the 9th bit position of the transmit shift register and tells the TX Control block to commence a transmission. The internal timing is such that one full machine cycle will elapse between "write to SBUF", and activation of SEND.

SEND enables the output of the shift register to the alternate output function line of P3.0, and also enables SHIFT CLOCK to the alternate output function line of P3.1. SHIFT CLOCK is low during S3, S4, and S5 of every machine cycle, and high during S6, S1 and S2. At S6P2 of every machine cycle in which SEND is active, the contents of the transmit shift register are shifted to the right one position.

As data bits shift out to the right, zeros come in from the left. When the MSB of the data byte is at the output position of the shift register, then the 1 that was initially loaded into the 9th position, is just to the left of the MSB, and all positions to the left of that contain zeros. This condition flags the TX Control block to do one last shift and then deactivate SEND and set TI. Both of these actions occur at S1P1 of the 10th machine cycle after "write to SBUF".

Reception is initiated by the condition REN 1 and RI = 0. At S6P2 of the next machine cycle, the RX Control unit writes the bits 11111110 to the receive shift register, and in the next clock phase activates RECEIVE.

RECEIVE enables SHIFT CLOCK to the alternate output function line of P3.1. SHIFT CLOCK makes transitions at S3P1 and S6P1 of every machine cycle. At S6P2 of every machine cycle in which RECEIVE is active, the contents of the receive shift register are shifted to the left one position. The value that comes in from the right is the value that was sampled at the P3.0 pin at S5P2 of the same machine cycle.

As data bits come in from the right, 1s shift out to the left. When the 0 that was initially loaded into the rightmost position arrives at the leftmost position in the shift register, it flags the RX Control block to do one last shift and load SBUF. At S1P1 of the 10th machine cycle, after the write to SCON that cleared RI, RECEIVE is cleared and RI is set.

6.7.2 Mode 1

Ten bits are transmitted (through TXD), or received (through RXD): a start bit (0), 8 data bits (LSB first), and a stop bit (1). On receive, the stop bit goes into RB8 in SCON. The baud rate is variable. Either Timer 1 or 2 may be used to clock the serial port to generate the variable baud rate by setting and/or clearing T2CON bits TCLK and RCLK.

Figure 6-18 shows a simplified functional diagram of the serial port in Mode 1, and associated timings for transmit and receive.

Transmission is initiated by any instruction that uses SBUF as a destination register. The "write to SBUF" signal also loads a 1 into the 9th bit position of the transmit shift register and flags the TX Control unit that a transmission is requested. Transmission actually commences at S1P1 of the machine cycle following the next rollover in the divide-by-16 counter. (Thus, the bit times are synchronized to the divide-by-16 counter, not to the "write to SBUF" signal).

The transmission begins with activation of SEND, which puts the start bit at TXD. One bit time later, DATA is activated, which enables the output bit of the transmit shift register to TXD. The first shift pulse occurs one bit time after that.

As data bits shift out to the right, zeros are clocked in from the left. When the MSB of the data byte is at the output position of the shift register, then the 1 that was initially loaded into the 9th position is just to the left of the MSB, and all positions to the left of that contain zeroes. This condition flags the TX Control unit to do one last shift and then deactivate SEND and set TI. This occurs at the 10th divide-by-16 rollover after "write to SBUF".

Reception is initiated by a detected 1-to-0 transition at RXD. For this purpose RXD is sampled at a rate of 16 times whatever baud rate has been established. When a transition is detected, the divide-by-16 counter is immediately reset, and 1FFH is written into the input shift register. Resetting the divide-by-16 counter aligns its rollovers with the boundaries of the incoming bit times.

The 16 states of the counter divide each bit time into 16ths. At the 7th, 8th, and 9th counter states of each bit time, the bit detector samples the value of RXD.

value accepted is the value that was seen in at least 2 of the 3 samples. This is done for noise rejection. If the value accepted during the first bit time is not 0, the receive circuits are reset and the unit goes back to looking for another 1-to-0 transition. This is to provide rejection of false start bits. If the start bit proves valid, it is shifted into the input shift register, and reception of the rest of the frame will proceed.

As data bits come in from the right, 1s shift out to the left. When the start bit arrives at the leftmost position in the shift register, (which in mode 1 is a 9-bit register), it flags the RX Control block to do one last shift, load SBUF and RB8, and set RI. The signal to load SBUF and RB8, and to set RI, will be generated if, and only if, the following conditions are met at the time the final shift pulse is generated.

1) RI = 0, and
2) Either SM2 = 0, or the received stop bit = 1

If either of these two conditions are not met, the received frame is irretrievably lost. If both conditions are met, the stop bit goes into RB8, the 8 data bits go into SBUF, and RI is activated. At this time, whether the above conditions are met or not, the unit goes back to looking for a 1-to-0 transition in RXD.

6.7.3 Modes 2 and 3

Eleven bits are transmitted (through TXD), or received (through RXD): a start bit (0), 8 data bits (LSB first), a programmable 9th data bit, and a stop bit (1). On transmit, the 9th data bit (TB8) can be assigned the value of 0 or 1. On receive, the 9th data bit goes into RB8 in SCON. The baud rate is programmable to either 1/32 or 1/64 the oscillator frequency in mode 2. Mode 3 may have a variable baud rate generated from either Timer 1 or 2 depending on the state of TCLK and RCLK.

Figure 6-19 shows a functional diagram of the serial port in modes 2 and 3. The receive portion is exactly the same as in mode 1. The transmit portion differs from mode 1 only in the 9th bit of the transmit shift register.

Transmission is initiated by any instruction that uses SBUF as a destination register. The "write to SBUF" signal also loads TB8 into the 9th bit position of the transmit shift register and flags the TX Control unit that a transmission is requested. Transmission commences at S1P1 of the maching cycle following the next rollover in the divide-by-16 counter. (Thus, the bit times are synchronized to the divide-by-16 counter, not the "write to SBUF" signal"

The transmission begins with activation of SEND, which puts the start bit at TXD. One bit time later, DATA is activated, which enables the output bit of the transmit shift register to TXD. The first shift pulse occurs one bit time after that. The first shift clocks a 1 (the stop bit) into the 9th bit position of the shift register. Thereafter, only zeroes are clocked in. Thus, as data bits shift out to the right, zeroes are clocked in from the left. When TB8 is at the output position of the shift register, then the stop bit is just to the left of TB8, and all positions to the left of that contain zeroes. This condition flags the TX Control unit to do one last shift and then deactivate SEND and set TI. This occurs at the 11th divide-by-16 rollover after "write to SBUF".

Reception is initiated by a detected 1-to-0 transition at RXD. For this purpose RXD is sampled at a rate of 16 times whatever baud rate has been established. When a transition is detected, the divide-by-16 counter is immediately reset, and 1FFH is written to the input shift register.

At the 7th, 8th and 9th counter states of each bit time, the bit detector samples the value of RXD. The value accepted is the value that was seen in at least 2 of the 3 samples. If the value accepted during the first bit time is not 0, the receive circuits are reset and the unit goes back to looking for another 1-to-0 transition. If the start bit proves valid, it is shifted into the input shift register, and reception of the rest of the frame will proceed.

As data bits come in from the right, 1s shift out to the left. When the start bit arrives at the leftmost position in the shift register (which in modes 2 and 3 is a 9-bit register), it flags the RX Control block to do one last shift, load SBUF and RB8, and set RI. The signal to load SBUF and RB8, and to set RI, will be generated if, and only if, the following conditions are met at the time the final shift pulse is generated:

1) RI = 0, and
2) Either SM2 = 0 or the received 9th data bit =1

If either of these conditions are not met, the received frame is irretrievably lost, and RI is not set. IF both conditions are met, the received 9th data bit goes into RB8, and the first 8 data bits go into SBUF. One bit time later, whether the above conditions were met or not, the unit goes back to looking for a 1-to-0 transition at the RXD input.

Note that the value of the received stop bit is irrelevant to SBUF, RB8, or RI.

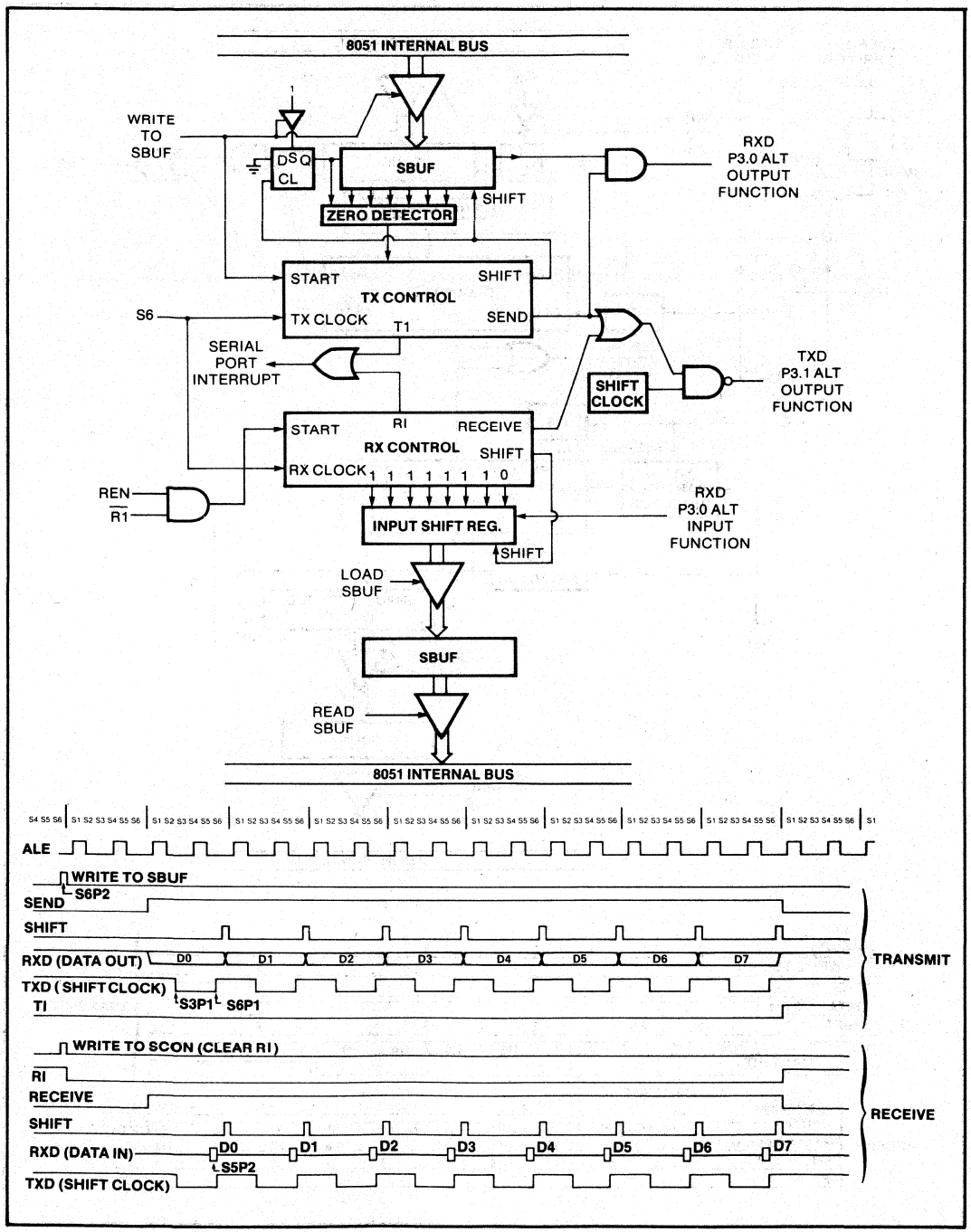


Figure 6-17. Serial Port Mode 0

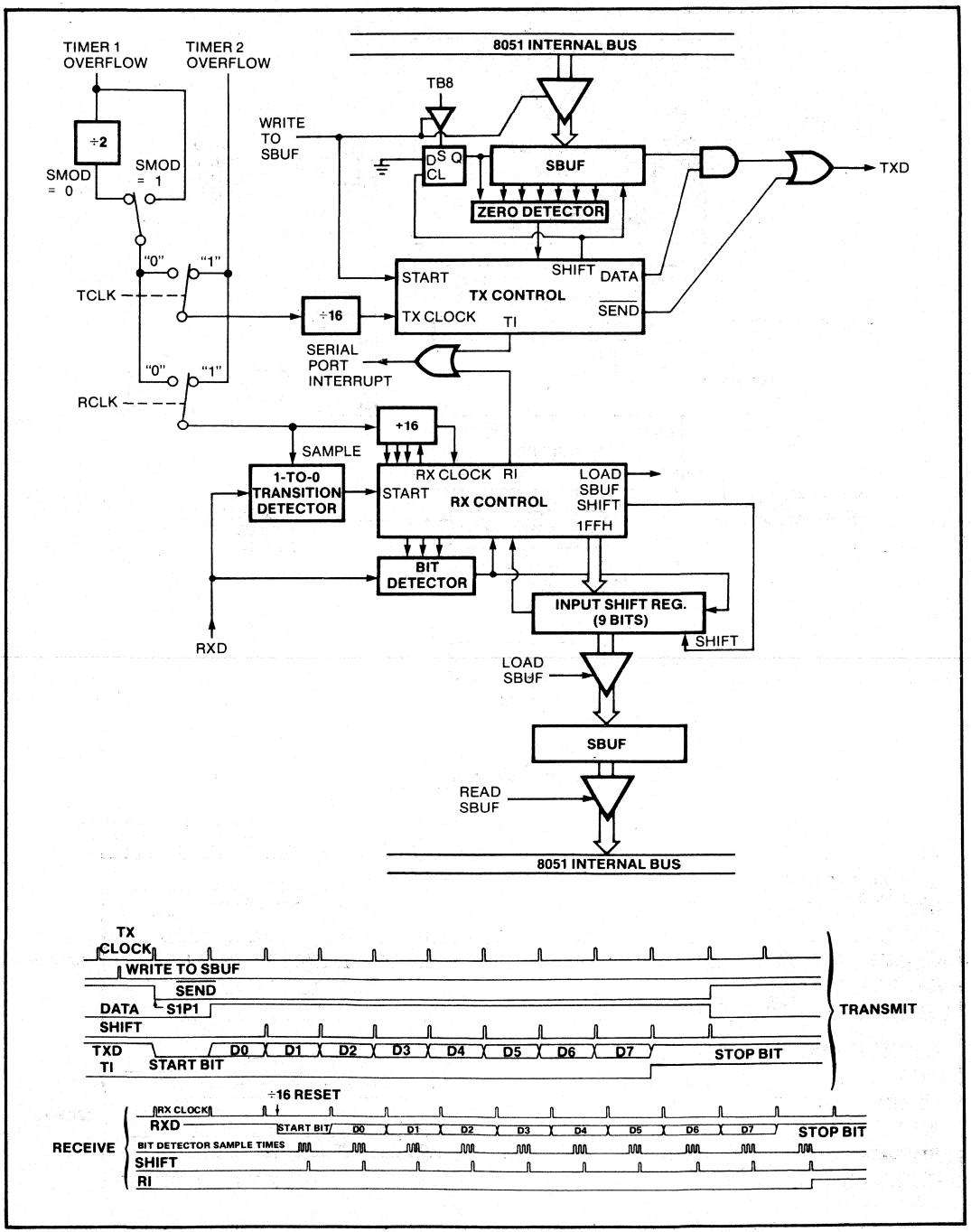


Figure 6-18. Serial Port Mode 1

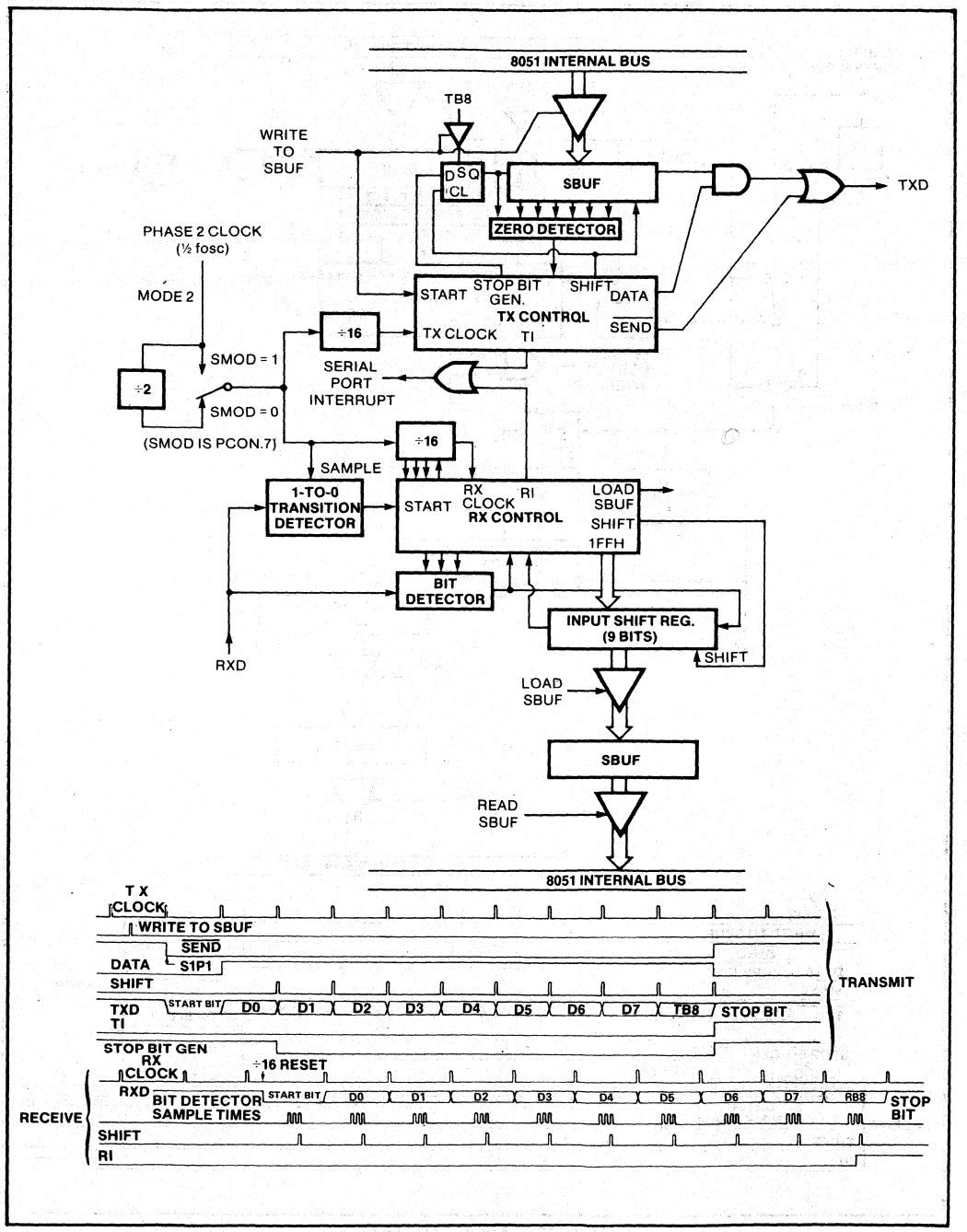


Figure 19-A. Serial Port Mode 2

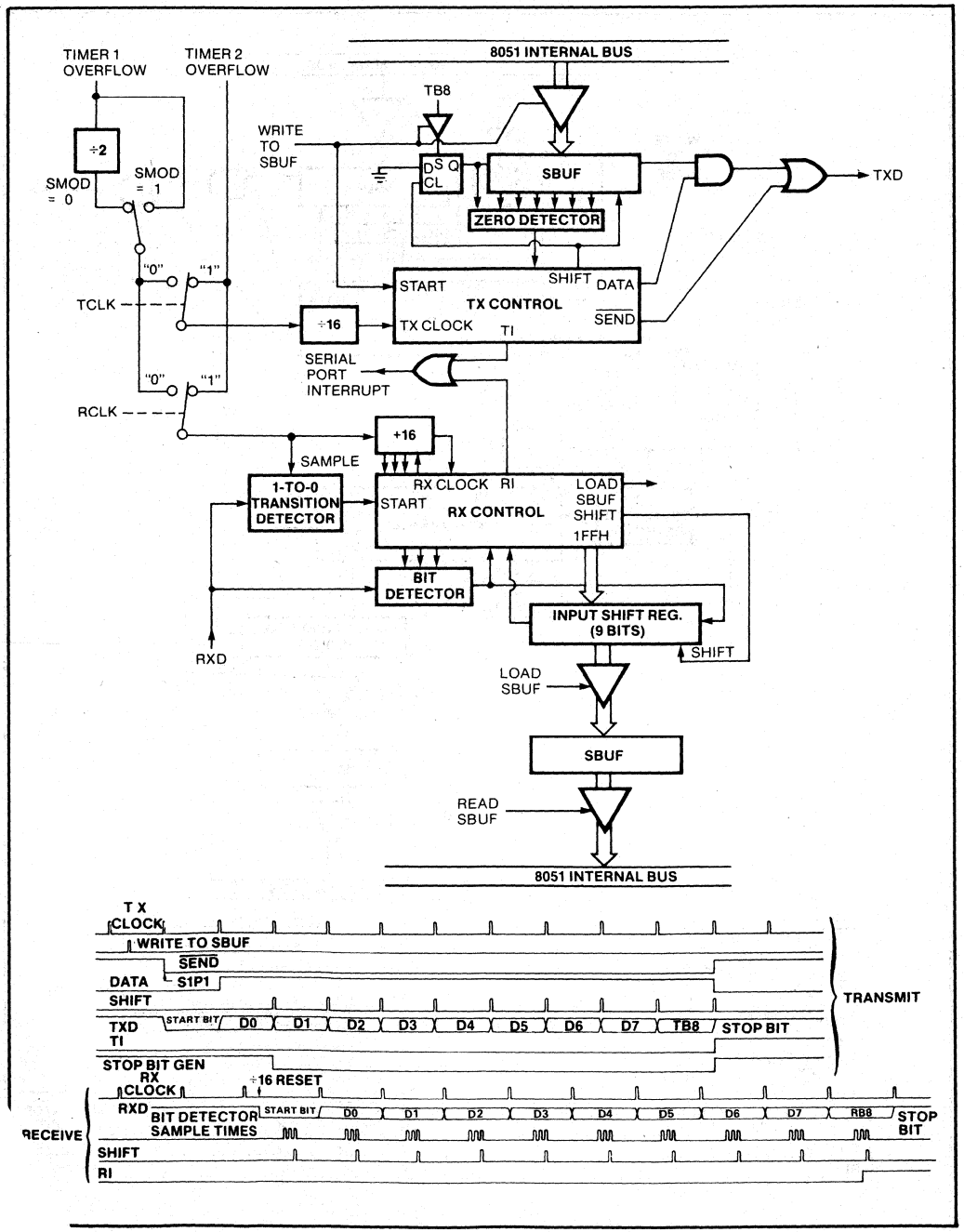


Figure 19B. Serial Port Mode 3

Table 2. Serial Port Operation Modes

MODE	SM0	SM1	DESCRIPTION
0	0	0	Shift Register I/O Expansion
1	0	1	8-bit UART, variable baud rate
2	1	0	9-bit UART, baud rate is either 1/32 or 1/64 oscillator frequency
3	1	1	9-bit UART, variable baud rate

6.7.4 Multiprocessor Communications

Modes 2 and 3 have a special provision for multiprocessor communications. In these modes, 9 data bits are received. The 9th one goes into RB8. Then comes a stop bit. The port can be programmed such that when the stop bit is received, the serial port interrupt will be activated only if RB8 = 1. This feature is enabled by setting bit SM2 in SCON. This feature is used in multiprocessor systems as follows:

When the master processor wants to transmit a block of data to one of several slaves, it first sends out an address byte to identify the target slave. An address byte differs from a data byte in that its 9th bit is 1, while the 9th bit of a data byte is a 0 data byte. SM2 = 1 enables slaves not to be interrupted by a data byte. An address byte, however, will interrupt all slaves. This allows each slave to examine the received byte to see if it is the target slave. The addressed slave clears its SM2 bit and prepares to receive the incoming data bytes. The slaves that were not addressed leave their SM2 bits set and go on about their business, ignoring the coming data bytes. SM2 should be cleared for operations in Mode 0 or 1.

6.7.5 Baud Rates

The baud rate in Mode 0 is fixed at 1/12 the oscillator frequency. The baud rate in Mode 2 is either 1/64 or 1/32 of the oscillator frequency, depending on the value of bit SMOD in Special Function Register PCON. If SMOD = 0 (which is its value after reset), the baud rate is 1/64 the oscillator frequency. If SMOD = 1, the baud rate is 1/32 of the oscillator frequency.

Baud rates in Modes 1 and 3 are determined by either the Timer/Counter 1 or 2 overflow rate. The reception rate may be different from the transmission rate depending on the state of bits TCLK and RCLK in T2CON.

TIMER/COUNTER 1 CLOCK SOURCE

When Timer/Counter 1 is used as the baud rate clock source to the serial port, the baud rate is determined as follows:

Baud Rate = (Timer/Counter 1 overflow rate)/n

where n is an integer whose value is either 32 or 16, depending on the value of bit SMOD in Special Function Register PCON (PCON will be discussed in Section 6.11.2). If SMOD = 0 (which is its value on reset), n = 32. If SMOD = 1, n = 16. Timer/Counter 1 can be configured in any mode. The Timer/Counter 1 overflow rate is determined by its count rate and how many counts it takes to reach overflow.

For example, Timer/Counter 1 can be configured in the auto reload mode (TMOD.5 = 1, TMOD.4 = 0). The Timer must be running (TCON.6 = 1), and to keep the overflows from generating unnecessary interrupts, the Timer/Counter 1 interrupt is disabled (IE.3 = 0). The overflow rate then depends on the reload value in TH1, as follows:

Overflow Rate = (count rate)/[256–(TH1)].

For very low baud rates one might select the 16-bit Timer 1 mode (TMOD.5 = 0, TMOD.4 = 1), and use the Timer 1 interrupt to do a software reload. In this case, one would want to have the Timer 1 interrupt enabled (IE.3 = 1).

In any case, if Timer 1 is running with bit $C/\overline{T}$ = 0, the count rate is 1/12 the oscillator frequency. If the timer is running with $C/\overline{T}$ = 1, the count rate is the external input frequency, whose maximum usable value is 1/24 the oscillator frequency.

Figure 6-20 lists various commonly used baud rates and how they can be obtained when using Timer 1 to clock the serial port.

BAUD RATE	f_{osc}	SMOD	TIMER 1		
			C/T	MODE	RELOAD VALUE
MODE 0 MAX: 1MHZ	12 MHZ	X	X	X	X
MODE 2 MAX: 375K	12 MHZ	1	X	X	X
MODES 1,3: 62.5K	12 MHZ	1	0	2	FFH
19.2K	11.059 MHZ	1	0	2	FDH
9.6K	11.059 MHZ	0	0	2	FDH
4.8K	11.059 MHZ	0	0	2	FAH
2.4K	11.059 MHZ	0	0	2	F4H
1.2K	11.059 MHZ	0	0	2	E8H
137.5	11.986 MHZ	0	0	2	1DH
110	6 MHZ	0	0	2	72H
110	12 MHZ	0	0	1	FEEBH

Figure 6-20. Timer 1 Generated Commonly Used Baud Rates

TIMER/COUNTER 2 CLOCK SOURCE (8032/8052 ONLY)

Timer/Counter 2 may be used in the timer or counter mode when clocking the serial port. In either case, the baud rate is determined by the overflow rate regardless of the state of SMOD. Since the serial port requires 16 clocks per bit, the baud rate is calculated by the following equation when $C/\overline{T2}=0$:

$$\text{Baud Rate} = \frac{\text{Oscillator frequency.}}{2 \times 16 \times \text{Divisor}}$$

Divisor is 65536 minus the 16-bit auto-reload value programmed into RCAP2H and RCAP2L. This allows baud rates from 5.72 baud to 375 kbaud at 12MHz operation.

When $C/\overline{T2}=1$, the overflow rate is equal to the count rate of the external input frequency divided by Divisor. The maximum usable value is 1/24 the oscillator frequency.

6.8 INTERRUPTS

The 8051 has five interrupt sources, (six for the 8032/8052), each of which can be programmed to one of two priority levels. The interrupt sources are from the two external interrupt inputs; one each from the three timer/counter overflow flags (and capture on the 8032/8052) and one from the serial port.

Each source can be individually enabled or disabled by setting or clearing a bit in Special Function Register IE. Figure 6-21 details the IE register.

(MSB)							(LSB)
EA	X	ET2	ES	ET1	EX1	ET0	EX0

Symbol	Position	Function
EA	IE.7	disables all interrupts. If EA = 0, no interrupt will be acknowledged. If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.
—	IE.6	reserved
ET2	IE.5	enables or disables the Timer 2 overflow or capture interrupt. If ET2 = 0, the Timer 2 interrupt is disabled.
ES	IE.4	enables or disables the Serial Port interrupt. If ES = 0, the Serial Port interrupt is disabled.
ET1	IE.3	enables or disables the Timer 1 Overflow interrupt. If ET1 = 0, the Timer 1 interrupt is disabled.
EX1	IE.2	enables or disables External Interrupt 1. If EX1 = 0, External Interrupt 1 is disabled.
ET0	IE.1	enables or disables the Timer 0 Overflow interrupt. If ET0 = 0, the Timer 0 interrupt is disabled.
EX0	IE.0	enables or disables External Interrupt 0. If EX0 = 0, External Interrupt 0 is disabled.

Figure 6-21. IE: Interrupt Enable Register

Each source can also be programmed to a high-priority level or a low-priority level by setting or clearing a bit in SFR IP. Figure 6-22 details IP. All of the interrupt flags can be set or cleared by software with the same effect as if by hardware.

(MSB)							(LSB)
EA	X	ET2	ES	ET1	EX1	ET0	EX0

Symbol	Position	Function
—	IP.7	reserved
—	IP.6	reserved
PT2	IP.5	defines the Timer 2 interrupt priority level. PT2 = 1 programs it to the higher priority level.
PS	IP.4	defines the Serial Port interrupt priority level. PS = 1 programs it to the higher priority level.
PT1	IP.3	defines the Timer 1 interrupt priority level. PT1 = 1 programs it to the higher priority level.
PX1	IP.2	defines the External Interrupt 1 priority level. PX1 = 1 programs it to the higher priority level.
PT0	IP.1	defines the Timer 0 interrupt priority level. PT0 = 1 programs it to the higher priority level.
PX0	IP.0	defines the External Interrupt 0 priority level. PX0 = 1 programs it to the higher priority level.

Figure 6-22. IP: Interrupt Priority Register

6.8.1 Priority Level Structure

A low-priority interrupt can itself be interrupted by a high-priority interrupt, but not by another low-priority interrupt. A high-priority interrupt cannot be interrupted. To implement these rules, the interrupt system contains two non-addressable "priority level active" "flip-flops". One indicates that a high-priority interrupt is being serviced, and blocks all further interrupts. The other indicates that a low-priority interrupt is being serviced, and blocks all but high priority interrupts.

In the event that requests of the same priority level are received simultaneously, an internal polling sequence determines which request is serviced. Thus, within each priority level there is a second priority structure determined by the following polling sequence:

SOURCE	PRIORITY WITHIN LEVEL
External Interrupt 0	(highest)
Timer/Counter 0 Overflow	
External Interrupt 1	
Timer/Counter 1 Overflow	
Serial Port	
Timer/Counter 2 Overflow	
Timer 2 Overflow/Negative Transition on T2EX (8032/8052 only)	(lowest)

All the interrupt sources are examined sequentially during each cycle, such that by S6 of any cycle all active interrupt requests have been found and prioritized. Response to the active request of highest priority will commence with state 1 of the next machine cycle, provided the response is not blocked by any of the following conditions:

1) An interrupt of equal or higher priority level is already in progress.
2) The current machine cycle is not the final cycle in the execution of the instruction in progress. (In other words, no interrupt request will be responded to until the instruction in progress is completed).
3) The instruction in progress is RETI or an access to Special Function Registers IE or IP. (in other words an interrupt request will not be responded to after RETI or after a read or write to IE or IP until at least one other instruction has been executed.

If any of the above conditions exists, the result of the interrupt poll is discarded. If none of the above conditions exists, the result of the interrupt poll is acted on with the very next machine cycle.

6.8.2 Interrupt Response Protocol

The processor acknowledges a request by first setting the appropriate "priority level active" flip-flop. Then it executes a hardware subroutine call to the servicing routine. It also clears the flag that requested the interrupt (with exceptions: it doesn't clear $\overline{\text{INT0}}$ or $\overline{\text{INT1}}$, since it has no control over the sources of these signals, and it doesn't clear TI, RI, TF2 or EXF2). The hardware subroutine call pushes the contents of the Program Counter onto the stack (but it does not save the PSW) and reloads the PC with an address that depends on the source of the interrupt request, as shown below:

SOURCE	VECTOR LOCATION
External Interrupt 0	0003H
Timer 0 Overflow	000BH
External Interrupt 1	0013H
Timer 1 Overflow	001BH
Serial Port	0023H
Timer 2 Overflow/ Negative Transition on T2EX (8032/8052 only)	002BH

Execution proceeds from that address until the RETI instruction is encountered. The RETI instruction clears the "priority level active" flip-flop that was set when this interrupt was acknowledged. Then it pops the top two bytes from the stack and reloads the Program Counter. Execution of the interrupted program commences from where it left off.

6.8.3 External Interrupts

The external sources can be programmed to be level-activated or transition-activated by setting or clearing bit IT1 or IT0 in Register TCON. If ITx = 0, external interrupt x is triggered by a detected low at the INTx pin. If ITx = 1, external interrupt x is edge-triggered. In this mode if successive samples of the INTx pin show a high in one cycle and a low in the next cycle, interrupt request flag IEx in TCON is set. Flag bit IEx then requests the interrupt.

Since the external interrupt pins are sampled once each machine cycle, an input high or low should hold for at least 12 oscillator periods to ensure sampling. If the external interrupt is transition-activated, the external source has to hold the request pin high for at least one cycle, and then hold it low for at least one cycle to ensure that the transition is seen so that interrupt request flag IEx will be set. IEx will be automatically cleared by the CPU when the service routine is called.

If the external interrupt is level-activated, the external source has to hold the request active until the requested interrupt is actually generated. Then it has to deactivate the request before the interrupt service routine is completed, or else another interrupt will be generated.

6.8.4 Response Time

The $\overline{\text{INT0}}$ and $\overline{\text{INT1}}$ levels are latched into an internal holding register at S5P2 of every machine cycle. The values are not actually polled by the circuitry until the next machine cycle. If a request is active and conditions are right for it to be acknowledged, a hardware subroutine call to the requested service routine will be the next instruction to be executed. The call itself takes two cycles. Thus, a minimum of three complete machine cycles elapse between activation of an external interrupt request and the beginning of execution of the first instruction of the service routine. Figure 6-23 shows interrupt response timings.

A longer response time would result if the request is blocked by one of the 3 previously listed conditions. If an interrupt of equal or higher priority level is already in progress, the additional wait time obviously depends on the nature of the other interrupt's service routine. If the instruction in progress is not in its final cycle, the additional wait time cannot be more than 3 cycles, since the longest instructions (MUL and DIV) are only 4

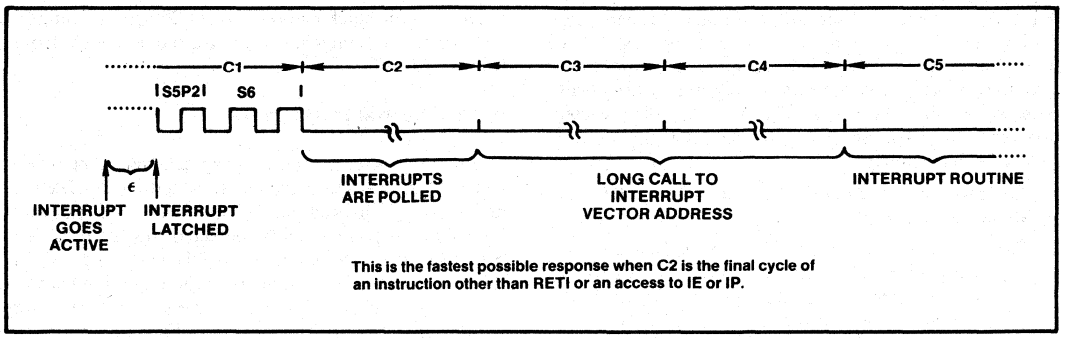


Figure 6-23. Interrupt Response Timing Diagram

cycles long, and if the instruction in progress is RETI or an access to IE or IP, the additional wait time cannot be more than 5 cycles (a maximum of one more cycle to complete the instruction in progress, plus 4 cycles to complete the next instruction if the next instruction is MUL or DIV).

Thus, in a single-interrupt system, the response time is always more than 3 cycles and less than 8 cycles.

6.9 SINGLE-STEP OPERATION

The 8051 interrupt structure allows single-step execution with very little software overhead. As previously noted, an interrupt request will not be responsed to while an interrupt of equal priority level is still in progress, nor will it be responded to after RETI until at least one other instruction has been executed. Thus, once an interrupt routine has been entered, it cannot be re-entered until at least once instruction of the interrupted program is executed. One way to use this feature for single-step operation is to program one of the external interrupts (say, $\overline{INT0}$) to be level-activated. The service routine for the interrupt will terminate with the following code:

```
JNB     P3.2,$      ;WAIT HERE TILL
                    INTO GOES HIGH
JB      P3.2,$      ;NOW WAIT HERE
                    TILL IT GOES LOW
RETI                :GO BACK AND
                    EXECUTE ONE
                    INSTRUCTION
```

Now if the $\overline{INT0}$ pin, which is also the P3.2 pin, is held normally low, the CPU will go right into the External Interrupt 0 routine and stay there until $\overline{INT0}$ is pulsed (from low to high to low). Then it will execute RETI, go back to the taskprogram, execute one instruction, and immediately re-enter the External Interrupt 0 routine to await the next pulsing of P3.2. One step of the task program is executed each time P3.2 is pulsed.

6.10 RESET

The reset circuitry for HMOS versions of the 8051 is connected to the reset pin, RST/VPD, as shown in Figure 6-24. A Schmitt Trigger is used at the input for noise rejection. The output of the Schmitt Trigger is sampled by the reset circuitry at S5P2 of every machine cycle.

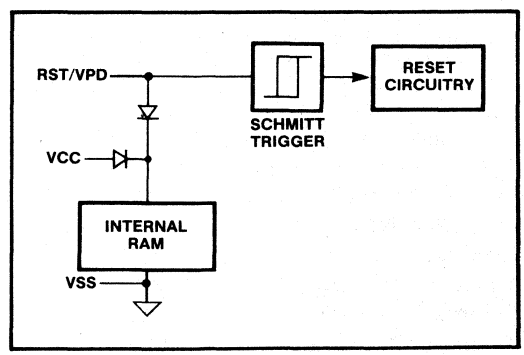


Figure 6-24. HMOS Reset Configuration at RST/VPD

The configuration for CHMOS versions is as shown in Figure 6-25. Note that the internal RAM draws backup power from VCC, not the reset pin. Hence in the 80C31/80C51, this pin is called simply RST. Backup power options are discussed in Section 6.11.

A reset in both HMOS and CHMOS versions is accomplished by holding the RST/VPD pin high for at

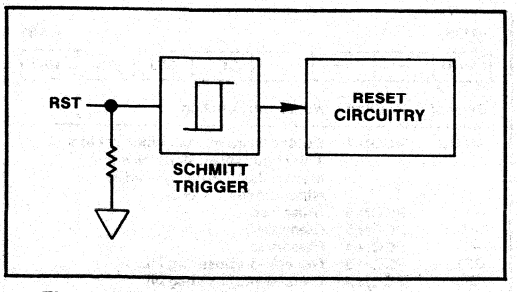


Figure 6-25. CHMOS Reset Configuration at RST/VPD

least two machine cycles (24 oscillator periods), *while the oscillator is running*. The CPU responds by executing an internal reset. It also configures the ALE and PSEN pins as inputs. (They are quasi-bidirectional). The internal reset is executed during the second cycle in which RST is high and is repeated every cycle until RST goes low. It leaves the internal registers as follows:

REGISTER	CONTENT
PC	000H
ACC	00H
B	00H
PSW	00H
SP	07H
DPTR	0000H
P0 - P3	0FFH
IP	(XX000000)
IE	(0X000000)
TMOD	00H
TCON	00H
T2CON	00H
TH0	00H
TL0	00H
TH1	00H
TL1	00H
TH2	00H
TL2	00H
RLDH	00H
RLDL	00H
SCON	00H
SBUF	Intermediate
PCON	(0XXX0000)

The internal RAM is not affected by reset. When VCC is turned on, the RAM content is indeterminate unless the part is returning from a reduced power mode of operation.

POWER-ON RESET

An automatic reset can be obtained when VCC is turned on by connecting the RST pin to VCC through a 10 μf capacitor and VSS through a 8.2KΩ resistor, providing the VCC risetime does not exceed a millisecond and the oscillator start-up time does not exceed 10 milliseconds. This power-on reset circuit is shown in Figure 6-26. When power comes on, the current drawn by RST is the difference between VCC and the capacitor voltage, and decreased from VCC as the cap charges. The larger the capacitor, the more slowly VRST decreases. VRST must remain above the lower threshhold of the Schmitt Trigger long enough to effect a complete reset. The time required is the oscillator start-up time, plus 2 machine cycles.

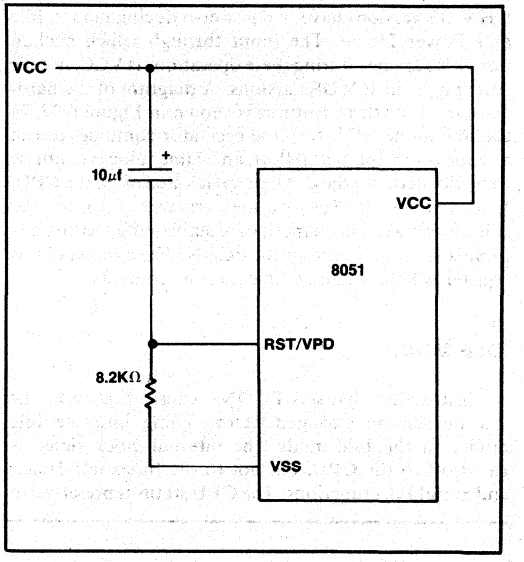


Figure 6-26. Power on Reset Circuit

6.11 POWER SUPPLY AND CONSUMPTION FEATURES

The 8051 has provisions for reduced component power consumption which allows for backup supplies.

In HMOS versions, this is achieved through the use of RST/VPD. CHMOS versions use the Power Control register PCON.

6.11.1 HMOS Power Down Operation

During normal operation the internal RAM draws its power from VCC. However, as was seen in Figure 6-24, if the voltage at RST/VPD exceeds VCC, it becomes the source of power for the RAM. This allows a backup power supply to be used to hold RAM data in the event of a power failure.

To take advantage of this feature, the user's system, upon detecting that a power failure is imminent, would interrupt the processor via $\overline{INT0}$ or $\overline{INT1}$ to transfer relevant data to the RAM and enable the backup power supply to RST/VPD pin before VCC falls below its operating limit. When power returns, VPD needs to stay on long enough to effect a reset (oscillator start-up time, plus two machine cycles), and normal operation can resume.

6.11.2 CHMOS Power Reduction Modes

CHMOS versions have two power-reducing modes, Idle and Power Down. The input through which backup power is supplied during these operations is VCC, not the reset pin as in HMOS versions. A diagram of the hardware used for these features is shown in Figure 6-27. In the Idle mode (IDL = 1), the oscillator continues to run and the Interrupt, Serial Port, and Timer blocks continue to be clocked, but the clock signal is gated off to the CPU. In Power Down (PD = 1), the oscillator is frozen. The Idle and Power Down modes are activated by setting bits in Special Function Register PCON. The address of this register is 87H. Figure 6-28 details its contents.

(MSB) (LSB)

SMOD	—	—	—	GF1	GF0	PD	IDL

Symbol	Position	Name and Function
SMOD	PCON.7	Double Baud rate bit. When set to a 1, the baud rate is doubled when the serial port is being used in either modes 1, 2 or 3.
—	PCON.6	(Reserved)
—	PCON.5	(Reserved)
—	PCON.4	(Reserved)
GF1	PCON.3	General-purpose flag bit.
GF0	PCON.2	General-purpose flag bit.
PD	PCON.1	Power Down bit. Setting this bit activates power down operation.
IDL	PCON.0	Idle mode bit. Setting this bit activates idle mode operation.

If 1s are written to PD and IDL at the same time, PD takes precedence. The reset value of PCON is (0XXX0000).

Figure 6-28. PCON: Power Control Register

IDLE MODE

An instruction that sets PCON.0 causes that to be the last instruction executed before going into the Idle mode. In the Idle mode, the internal clock signal is gated off to the CPU, but not to the Interrupt, Timer, and Serial Port functions. The CPU status is preserved in its entirety: the Stack Pointer, Program Counter, Program Status Word, Accumulator, and all other registers maintain their data during Idle.

There are two ways to terminate the Idle. Activation of any enabled interrupt will cause PCON.0 to be cleared by hardware, terminating the Idle mode. The interrupt will be serviced, and following RETI the next instruction to be executed will be the one following the instruction that put the device into Idle.

The flag bits GF0 and GF1 can be used to give an indication if an interrupt occurred during normal operation or

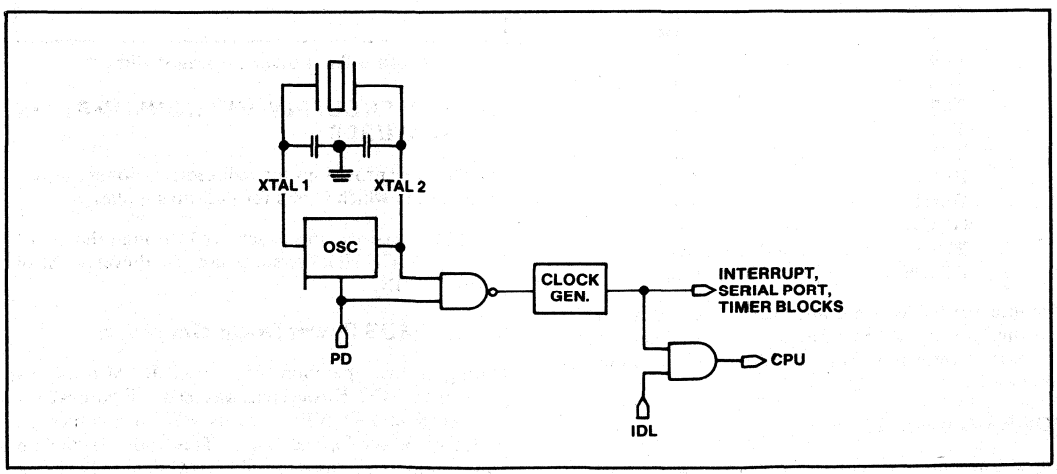


Figure 6-27. Idle and Power Down Hardware

during an Idle. For example, an instruction that activates Idle can also set one or both flag bits. When Idle is terminated by an interrupt, the interrupt service routine can examine the flag bits.

The other way of terminating the Idle mode is with a hardware reset. Since the clock oscillator is still running, the hardware reset needs to be held active for only two machine cycles (24 oscillator periods) to complete the reset.

POWER DOWN MODE

An instruction that sets PCON.1 causes that to be the last instruction executed before going into the Power Down mode. In the Power Down mode, the on-chip oscillator is stopped. With the clock frozen, all functions are stopped and only the on-chip RAM is held. (The Special Function Registers are not held). The only exit from Power Down is a hardware reset.

In the Power down mode of operation, VCC can be reduced to minimize power consumption. Care must be taken, however, to ensure that VCC is not reduced before the Power Down mode is invoked, and that VCC is restored to its normal operating level, before the Power Down mode is terminated. The reset that terminates Power Down also frees the oscillator. The reset should not be activated before VCC is restored to its normal operating level, and must be held active long enough to allow the oscillator to restart and stabilize (normally less than 10 msec).

MCS®-51 Memory Organization, Addressing Modes and Boolean Processor 7

CHAPTER 7
MCS®-51 MEMORY ORGANIZATION, ADDRESSING MODES AND BOOLEAN PROCESSOR

7.0 INTRODUCTION

The MCS®-51 architecture provides on-chip memory as well as off-chip memory expansion capabilities. Several addressing mechanisms are incorporated to allow for an optimal instruction set.

7.1 MEMORY ORGANIZATION

The 8051 has three basic memory address spaces:

- 64K-byte Program Memory;
- 64K-byte External Data Program Memory; and
- 256-byte (384 bytes for the 8032/8052 Internal Data Memory

Figure 7-1 shows MCS®-51 memory maps.

PROGRAM MEMORY ADDRESS SPACE

The 64K-byte Program Memory space consists of an internal and an external memory portion. If the $\overline{EA}$ pin is held high, the 8051 executes out of internal program memory unless the address exceeds 0FFFH (1FFFH for the 8052). Locations 1000H through 0FFFFH (2000H through 0FFFFH for the 8052), are then fetched from external Program memory. If the $\overline{EA}$ pin is held low, the 8051 fetches all instructions from external Program Memory. In either case, the 16-bit Program Counter is the addressing mechanism.

Locations 00 through 42H (00 through 2BH for the 8032/8052) in Program Memory are reserved for interrupt service routines as indicated in Table 1.

DATA MEMORY ADDRESS SPACE

The Data Memory address space consists of an internal and an external memory space. External Data Memory is accessed when a MOVX instruction is executed.

Internal Data Memory is divided into three physically separate and distinct blocks: the lower 128 bytes of RAM; the upper 128 bytes of RAM (accessable in the 8032/8052 only); and the 128-byte Special Function Register (SFR) area. While the upper RAM area and the SFR area share the same address locations, they are accessed through different addressing modes. These modes are discussed in a later section.

Figure 7-2 shows a mapping of Internal Data Memory. Four 8-Register Banks occupy locations 0 through 31 in the lower RAM area. Only one of these banks may be enabled at a time (through a two-bit field in the PSW). The next sixteen bytes, locations 32 through 47 contain 128 bit addressable locations. Figure 7-3 shows the RAM bit addresses. The SFR area also has bit addressable locations. They are shown in Figure 7-4.

Note that reading from unused locations in Internal Data Memory will yield random data.

Table 1. Addressing Method and Associated Memory Spaces

Register Addressing
- R0-R7
- ACC, B, CY(bit), DPTR

Direct Addressing
- Lower 128 bytes of internal RAM
- Special Function Registers

Register Indirect Addressing
- Internal RAM (@R1, @R0, SP)
- External Data Memory (@R1, R0, @DPTR)

Immediate Addressing
- Program Memory

Base-Register plus Index-Register Indirect Addressing
- Program Memory (@DPTR + A, @ PC + A)

7.2 ADDRESSING MODES

The 8051 uses five addressing modes:
- Register;
- Direct;
- Register Indirect;
- Immediate; and
- Base-Register plus Index-Register Indirect.

Table 2 summarizes which memory spaces may be accessed by each of the addressing modes.

Table 2.

Source	Address
External Interrupt 0	0003H
Timer 0 Overflow	000BH
External Interrupt 1	0013H
Timer 1 Overflow	001BH
Serial Port	0023H
Timer 2 Overflow/TZEX Negative Transition	002BH

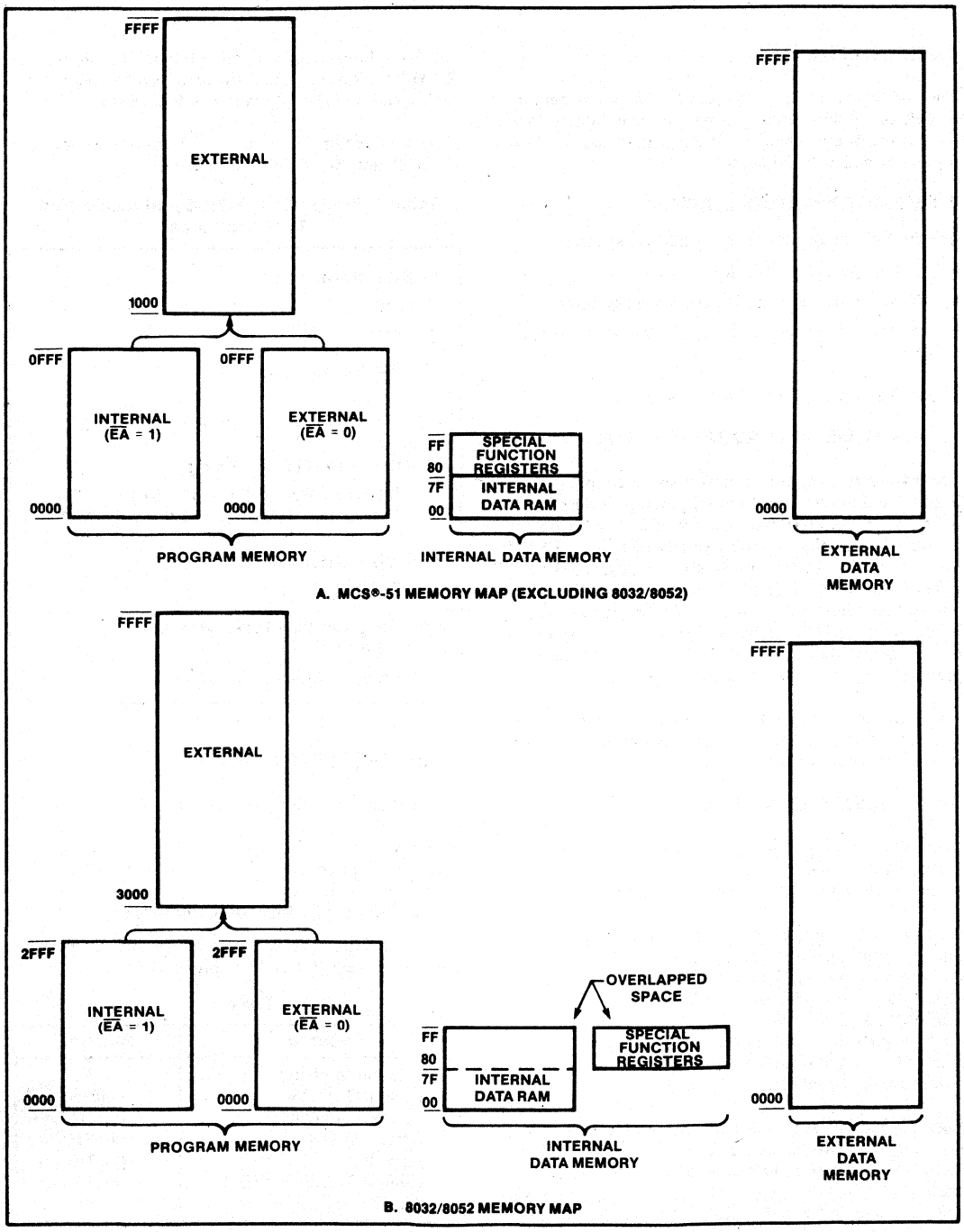


Figure 7-1. MCS®-51 Memory Maps

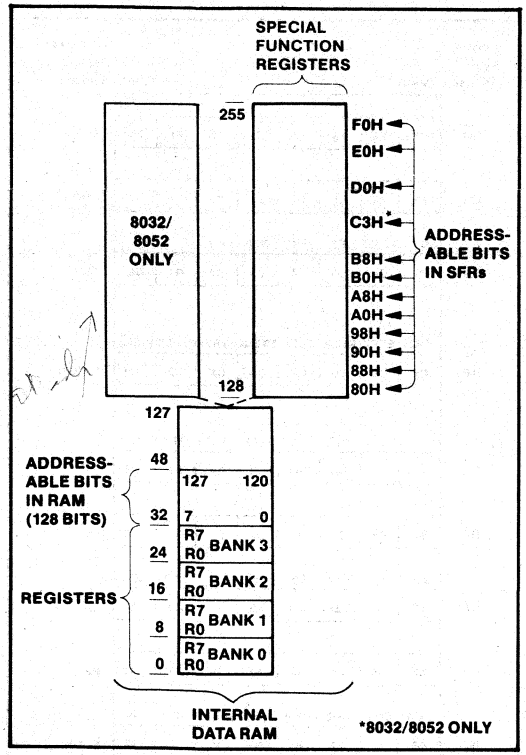


Figure 7-2. Internal Data Memory Address Space

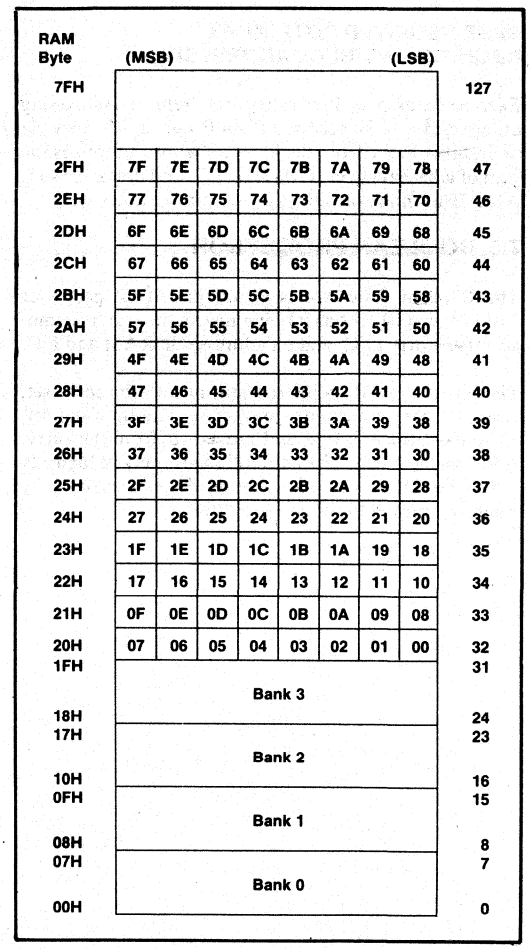


Figure 7-3. Special Function Bit Addressable Locations

REGISTER ADDRESSING

Register Addressing accesses the eight working registers (R0-R7) of the selected Register Bank. The least significant three bits of the instruction op code indicate which register is to be used. ACC, B, DPTR and CY, the Boolean Processor accumulator, can also be addressed as registers.

DIRECT ADDRESSING

Direct Addressing is the only method of accessing the Special Function Registers. The lower 128 bytes of Internal RAM are also directly addressable.

REGISTER-INDIRECT ADDRESSING

Register-Indirect Addressing uses the contents of either R0 or R1 (in the selected Register Bank) as a pointer to locations in a 256-byte block: the lower 128 bytes of internal RAM; the upper 128 bytes of internal RAM (8032/8052 only); or the lower 256 bytes of external Data Memory. Note that the Special Function Registers are not accessable by this method. Access to the full 64K external Data Memory address space is accomplished by using the 16-bit Data Pointer.

Execution of PUSH and POP instructions also use Register-Indirect addressing. The Stack Pointer may reside anywhere in Internal RAM.

IMMEDIATE ADDRESSING

Immediate Addressing allows constants to be part of the op code instruction in Program Memory.

BASE-REGISTER PLUS INDEX REGISTER-INDIRECT ADDRESSING

Base-Register plus Index Register-Indirect Addressing allows a byte to be accessed from Program Memory via an indirect move from the location whose address is the sum of a base register (DPTR or PC) and index register, ACC. This mode facilitates look-up-table accesses.

7.3 BOOLEAN PROCESSOR

The Boolean Processor is an integrated bit processor within the 8051. It has its own instruction set, accumulator (the carry flag), and bit addressable RAM and I/O.

The bit-manipulation instructions allow a bit to be set, cleared, complimented, jump-if-set, jump-if-not-set, jump-if-set-then-cleared and moved to/from the carry. Addressable bits, or their compliments, may be logically ANDed or ORed with the contents of the carry flag. The result is returned to the carry register.

Direct Byte Address	Bit Addresses (MSB)							(LSB)	Hardware Register Symbol
240	F7	F6	F5	F4	F3	F2	F1	F0	B
224	E7	E6	E5	E4	E3	E2	E1	E0	ACC
	CY	AC	F0	RS1	RS0	OV		P	
208	D7	D6	D5	D4	D3	D2	D1	D0	PSW
	TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/$\overline{T2}$	CP/$\overline{RL2}$	
200	CF	CE	CD	CC	CB	CA	C9	C8	T2CON
			PT2	PS	PT1	PX1	PT0	PX0	
184	—	—	BD	BC	BB	BA	B9	B8	IP
176	B7	B6	B5	B4	B3	B2	B1	B0	P3
	EA		ET2	ES	ET1	EX1	ET0	EX0	
168	AF	—	AD	AC	AB	AA	A9	A8	IE
160	A7	A6	A5	A4	A3	A2	A1	A0	P2
	SM0	SM1	SM2	REN	TB8	RB8	TI	RI	
152	9F	9E	9D	9C	9B	9A	99	98	SCON
144	97	96	95	94	93	92	91	90	P1
	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	
136	8F	8E	8D	8C	8B	8A	89	88	TCON
128	87	86	85	84	83	82	81	80	P0

Figure 7-4. Special Function Register Bit Address

CHAPTER 8
MCS®-51 INSTRUCTION SET

8.0 INTRODUCTION

The MCS®-51 instruction set includes 111 instructions, 49 of which are single-byte, 45 two-byte and 17 three byte. The instruction op code format consists of a function mnemonic followed by a "destination, source" operand field. This field specifies the data type and addressing method(s) to be used.

By Intel convention, multiple byte address and data operands are stored least significant byte in the high-order address and most significant byte in the low-order address.

8.1 FUNCTIONAL OVERVIEW

The MCS-51 instruction set is divided into four functional groups:

- Data Transfer
- Arithmetic
- Logic
- Control Transfer

8.1.1 Data Transfer

Data transfer operations are divided into three classes:

- General Purpose
- Accumulator-Specific
- Address-Object

None of these operations affect the PSW flag settings except a POP or MOV directly to the PSW.

GENERAL-PURPOSE TRANSFERS

- MOV performs a bit or a byte transfer from the source operand to the destination operand.
- PUSH increments the SP register and then transfers a byte from the source operand to the stack location currently addressed by SP.
- POP transfer a byte operand from the stack location addressed by SP to the destination operand and then decrements SP.

ACCUMULATOR SPECIFIC TRANSFERS

- XCH exchanges the byte source operand with register A (accumulator).
- XCHD exchanges the low-order nibble of the byte source operand with the low-order nibble of A.
- MOVX performs a byte move between the External Data Memory and the accumulator. The external address can be specified by the DPTR register (16-bit) or the R1 or R0 register (8-bit).
- MOVC moves a byte from Program memory to the accumulator. The operand in A is used as an index into a 256-byte table pointed to by the base register (DPTR or PC). The byte operand accessed is transferred to the accumulator.

ADDRESS-OBJECT TRANSFER

- MOV DPTR, #data loads 16-bits of immediate data into a pair of destination registers, DPH and DPL.

8.1.2 Arithmetic

The 8051 has four basic mathematical operations. Only 8-bit operations using unsigned arithmetic are supported directly. The overflow flag, however, permits the addition and subtraction operation to serve for both unsigned and signed binary integers. Arithmetic can also be performed directly on packed decimal (BCD) representations.

ADDITION

- INC (increment) adds one to the source operand and puts the result in the operand.
- ADD adds A to the source operand and returns the result to A.
- ADDC (add with Carry) adds A and the source operand, then adds one (1) if CY is set, and puts the result in A.
- DA (decimal-add-adjust for BCD addition) corrects the sum which results from the binary addition of two two-digit decimal operands. The packed decimal sum formed by DA is returned to A. CY is set if the BCD result is greater than 99; otherwise, it is cleared.

SUBTRACTION

- SUBB (subtract with borrow) subtracts the second source operand from the first operand (the accumulator), subtracts one (1) if CY is set and returns the result to A.
- DEC (decrement) subtracts one (1) from the source operand and returns the result to the operand.

MULTIPLICATION

- MUL performs an unsigned multiplication of the A register by the B register, returning a double-byte result. A receives the low-order byte, B receives the high-order byte. OV is cleared if the top half of the result is zero and is set if it is non-zero. CY is cleared. AC is unaffected.

DIVISION

- DIV performs an unsigned division of the A register by the B register and returns the integer quotient to A and returns the fractional remainder to the B register. Division by zero leaves indeterminate data in registers A and B and sets OV; otherwise OV is cleared. CY is cleared. AC is unaffected.

  Unless otherwise stated in the above descriptions, the flags of PSW are affected as follows:

- CY is set if the operation causes a carry to or from the resulting high-order bit. Otherwise CY is cleared.

- AC is set if the operation results in a carry from the low-order four bits of the result (during addition), or a borrow from the high-order bits to the low-order bits (during subtraction); otherwise AC is cleared.

- OV is set if the operation results in a carry to the high-order bit of the result but not a carry from the high-order bit, or vice versa; otherwise OV is cleared. OV is used in two's-complement arithmetic, because it is set when the signed result cannot be represented in 8 bits.

- P is set if the modulo 2 sum of the eight bits in the accumulator is 1 (odd parity); otherwise P is cleared (even parity). When a value is written to the PSW register, the P bit remains unchanged, as it always reflects the parity of A.

8.1.3 Logic

The 8051 performs basic logic operations on both bit and byte operands.

SINGLE-OPERAND OPERATIONS

- CLR sets A or any directly addressable bit to zero (0).

- SETB sets any directly addressable bit to one (1).

- CPL is used to compliment the contents of the A register without affecting any flags, or any directly addressable bit location.

- RL, RLC, RR, RRC, SWAP are the five rotate operations that can be performed on A. RL, rotate left, RR, rotate right, RLC, rotate left through C, RRC, rotate right through C, and SWAP, rotate left four. For RLC and RRC the CY flag becomes equal to the last bit rotated out. SWAP rotates A left four places to exchange bits 3 through 0 with bits 7 through 4.

TWO-OPERAND OPERATIONS

- ANL performs bitwise logical and of with two source operands (for both bit and byte operands) and returns the result to the location of the first operand.

- ORL performs bitwise logical or of two source operands (for both bit and byte operands) and returns the result of the location of the first operand.

- XRL performs bitwise logical or of two source operands (byte operands) and returns the result to the location of the first operand.

8.1.4 Control Transfer

There are three classes of control transfer operations: unconditional calls, returns and jumps; conditional jumps; and interrupts. All control transfer operations cause, some upon a specific condition, the program execution to continue at a non-sequential location in program memory.

UNCONDITIONAL CALLS, RETURNS AND JUMPS

Unconditional calls, returns and jumps transfer control from the current value of the Program Counter to the target address. Both direct and indirect transfers are supported.

- ACALL and LCALL push the address of the next instruction onto the stack and then transfer control to the target address. ACALL is a 2-byte instruction used when the target address is in the current 2K page. LCALL is a 3-byte instruction that addresses the full 64K program space. In ACALL, immediate data (i.e. an 11 bit address field) is concatenated to the five most significant bits of the PC (which is pointing to the next instruction). If ACALL is in the last 2 bytes of a 2K page then the call will be made to the next page since the PC will have been incremented to the next instruction prior to execution.

- RET transfers control to the return address saved on the stack by a previous call operation and decrements the SP register by two (2) to adjust the SP for the popped address.

- AJMP, LJMP and SJMP transfer control to the target operand. The operation of AJMP and LJMP are analogous to ACALL and LCALL. The SJMP (short jump) instruction provides for transfers within a 256 byte range centered about the starting address of the next instruction (-128 to +127).

- JMP @A+DPTR performs a jump relative to the DPTR register. The operand in A is used as the offset (0-255) to the address in the DPTR register. Thus, the effective destination for a jump can be anywhere in the Program Memory space.

CONDITIONAL JUMPS

Conditional jumps perform a jump contingent upon a specific condition. The destination will be within a 256-byte range centered about the starting address of the next instruction (-128 to +127).

- JZ performs a jump if the accumulator is zero.
- JNZ performs a jump if the accumulator is not zero.
- JC performs a jump if the carry flag is set.
- JNC performs a jump if the carry flag is not set.
- JB performs a jump if the Direct Addressed bit is set.
- JNB performs a jump if the Direct Addressed bit is not set.
- JBC performs a jump if the Direct Addressed bit is set and then clears the Direct Addressed bit.
- CJNE compares the first operand to the second operand and performs a jump if they are not equal. CY is set if the first operand is less than the second operand; otherwise it is cleared. Comparisons can be made between A directly addressable bytes in Internal Data Memory or between an immediate value and either A, a register in the selected Register Bank, or a Register-Indirect addressed byte of Internal RAM.
- DJNZ decrements the source operand and returns the result to the operand. A jump is performed if the result is not zero. The source operand of the DJNZ instruction may be any byte in the Internal Data Memory. Either Direct or Register Addressing may be used to address the source operand.

INTERRUPT RETURNS

- RETI transfers control as does RET, but additionally enables interrupts of the current priority level.

8.2 INSTRUCTION DEFINITIONS

Each of the 51 basic MCS-51 operations, ordered alphabetically according to the operation mnemonic are described beginning page 8-8.

A brief example of how the instruction might be used is given as well as its effect on the PSW flags. The number of bytes and machine cycles required, the binary machine-language encoding, and a symbolic description or restatement of the function is also provided.

Note: Only the carry, auxiliary-carry, and overflow flags are discussed. The parity bit is computed after every instruction cycle that alters the accumulator. Similarly, instructions which alter directly addressed registers could affect the other status flags if the instruction is applied to the PSW. Status flags can also be modified by bit-manipulation.

For details on the MCS-51 assembler, ASM51, refer to the MCS-51 Macro Assembler User's Guide, publication number 9800937.

Table 8-1 summarized the MCS-51 instruction set.

Table 8-1. 8051 Instruction Set Summary

Interrupt Response Time: To finish execution of current instruction, respond to the interrupt request, push the PC and to vector to the first instruction of the interrupt service program requires 38 to 81 oscillator periods (3 to 7μs @ 12 MHz).

INSTRUCTIONS THAT AFFECT FLAG SETTINGS[1]

INSTRUCTION	FLAG			INSTRUCTION	FLAG		
	C	OV	AC		C	OV	AC
ADD	X	X	X	CLR C	0		
ADDC	X	X	X	CPL C	X		
SUBB	X	X	X	ANL C,bit	X		
MUL	0	X		ANL C,/bit	X		
DIV	0	X		ORL C,bit	X		
DA	X			ORL C,bit	X		
RRC	X			MOV C,bit	X		
RLC	X			CJNE	X		
SETB C	1						

[1]Note that operations on SFR byte address 208 or bit addresses 209-215 (i.e., the PSW or bits in the PSW) will also affect flag settings.

Notes on instruction set and addressing modes:

Rn —Register R7-R0 of the currently selected Register Bank.

direct —8-bit internal data location's address. This could be an Internal Data RAM location (0-127) or a SFR [i.e., I/O port, control register, status register, etc. (128-255)].

@Ri —8-bit internal data RAM location (0-255) addressed indirectly through register R1 or R0.

#data —8-bit constant included in instruction.

#data 16 —16-bit constant included in instruction

addr 16 —16-bit destination address. Used by LCALL & LJMP. A branch can be anywhere within the 64K-byte Program Memory address space.

addr 11 —11-bit destination address. Used by ACALL & AJMP. The branch will be within the same 2K-byte page of program memory as the first byte of the following instruction.

rel —Signed (two's complement) 8-bit offset byte. Used by SJMP and all conditional jumps. Range is -128 to +127 bytes relative to first byte of the following instruction.

bit —Direct Addressed bit in Internal Data RAM or Special Function Register.

* —New operation not provided by 8048AH/8049AH.

ARITHMETIC OPERATIONS

Mnemonic		Description	Byte	Oscillator Period
ADD	A,Rn	Add register to Accumulator	1	12
ADD	A,direct	Add direct byte to Accumulator	2	12
ADD	A,@Ri	Add indirect RAM to Accumulator	1	12
ADD	A,#data	Add immediate data to Accumulator	2	12
ADDC	A,Rn	Add register to Accumulator with Carry	1	12
ADDC	A,direct	Add direct byte to Accumulator with Carry	2	12
ADDC	A,@Ri	Add indirect RAM to Accumulator with Carry	1	12
ADDC	A,#data	Add immediate data to Acc with Carry	2	12
SUBB	A,Rn	Subtract register from Acc with borrow	1	12
SUBB	A,direct	Subtract direct byte from Acc with borrow	2	12

ARITHMETIC OPERATIONS Cont.

Mnemonic		Description	Byte	Oscillator Period
SUBB	A,@Ri	Subtract indirect RAM from Acc with borrow	1	12
SUBB	A,#data	Subtract immediate data from Acc with borrow	2	12
INC	A	Increment Accumulator	1	12
INC	Rn	Increment register	1	12
INC	direct	Increment direct byte	2	12
INC	@Ri	Increment indirect RAM	1	12
DEC	A	Decrement Accumulator	1	12
DEC	Rn	Decrement Register	1	12
DEC	direct	Decrement direct byte	2	12
DEC	@Ri	Decrement indirect RAM	1	12
INC	DPTR	Increment Data Pointer	1	24
MUL	AB	Multiply A & B	1	48
DIV	AB	Divide A by B	1	48
DA	A	Decimal Adjust Accumulator	1	12

Table 8-1. 8051 Instruction Set Summary (Continued)

LOGICAL OPERATIONS

Mnemonic		Description	Byte	Oscillator Period
ANL	A,Rn	AND register Accumulator	1	12
ANL	A,direct	AND direct byte to Accumulator	2	12
ANL	A,@Ri	AND indirect RAM to Accumulator	1	12
ANL	A,#data	AND immediate data to Accumulator	2	12
ANL	direct,A	AND Accumulator to direct byte	2	12
ANL	direct,#data	AND immediate data to direct byte	3	24
ORL	A,Rn	OR register to Accumulator	1	12
ORL	A,direct	OR direct byte to Accumulator	2	12
ORL	A,@Ri	OR indirect RAM to Accumulator	1	12
ORL	A,#data	OR immediate data to Accumulator	2	12
ORL	direct,A	OR Accumulator to direct byte	2	12
ORL	direct,#data	OR immediate data to direct byte	3	24
XRL	A,Rn	Exclusive-OR register to Accumulator	1	12
XRL	A,direct	Exclusive-OR direct byte to Accumulator	2	12

LOGICAL OPERATIONS Cont.

Mnemonic		Description	Byte	Oscillator Period
XRL	A,@Ri	Exclusive-OR indirect RAM to Accumulator	1	12
XRL	A,#data	Exclusive-OR immediate data to Accumulator	2	12
XRL	direct,A	Exclusive-OR Accumulator to direct byte	2	12
XRL	direct,#data	Exclusive-OR immediate data to direct byte	3	24
CLR	A	Clear Accumulator	1	12
CPL	A	Complement Accumulator	1	12
RL	A	Rotate Accumulator Left	1	12
RCL	A	Rotate Accumulator Left through the Carry	1	12
RR	A	Rotate Accumulator Right	1	12
RRC	A	Rotate Accumulator Right through the Carry	1	12
SWAP	A	Swap nibbles within the Accumulator	1	12

Table 8-1. 8051 Instruction Set Summary (Continued)

DATA TRANSFER

Mnemonic		Description	Byte	Oscillator Period
MOV	A,Rn	Move register to Accumulator	1	12
MOV	A,direct	Move direct byte to Accumulator	2	12
MOV	A,@Ri	Move indirect RAM to Accumulator	1	12
MOV	A,#data	Move immediate data to Accumulator	2	12
MOV	Rn,A	Move Accumulator to register	1	12
MOV	Rn,direct	Move direct byte to register	2	24
MOV	Rn,#data	Move immediate data to register	2	12
MOV	direct,A	Move Accumulator to direct byte	2	12
MOV	direct,Rn	Move register to direct byte	2	24
MOV	direct,direct	Move direct byte to direct	3	24
MOV	direct,@Ri	Move indirect RAM to direct byte	2	24
MOV	direct,#data	Move immediate data to direct byte	3	24
MOV	@Ri,A	Move Accumulator to indirect RAM	1	12
MOV	@Ri,direct	Move direct byte to indirect RAM	2	24
MOV	@Ri,#data	Move immediate data to indirect RAM	2	12

DATA TRANSFER Cont.

Mnemonic		Description	Byte	Oscillator Period
MOV	DPTR,#data16	Load Data Pointer with a 16-bit constant	3	24
MOVC	A,@A+DPTR	Move Code byte relative to DPTR to Acc	1	24
MOVC	A,@A+PC	Move Code byte relative to PC and Acc	1	24
MOVX	A,@Ri	Move External RAM (8-bit addr) to Acc	1	24
MOVX	A,@DPTR	Move External RAM (16-bit addr) to Acc	1	24
MOVX	@Ri,A	Move Acc to External RAM (8-bit addr)	1	24
MOVX	@DPTR,A	Move Acc to External RAM (16-bit addr)	1	24
PUSH	direct	Push direct byte onto stack	2	24
POP	direct	Pop direct byte from stack	2	24
XCH	A,Rn	Exchange register with Accumulator	1	12
XCH	A,direct	Exchange direct byte with Accumulator	2	12
XCH	A,@Ri	Exchange indirect RAM with Accumulator	1	12
XCHD	A,@Ri	Exchange low-order Digit indirect RAM with Acc	1	12

Table 8-1. 8051 Instruction Set Summary (Continued)

BOOLEAN VARIABLE MANIPULATION

Mnemonic		Description	Byte	Oscillator Period
CLR	C	Clear Carry	1	12
CLR	bit	Clear direct bit	2	12
SETB	C	Set Carry	1	12
SETB	bit	Set direct bit	2	12
CPL	C	Complement Carry	1	12
CPL	bit	Complement direct bit	2	12
ANL	C,bit	AND direct bit to Carry	2	24
ANL	C,/bit	AND complement of direct bit to Carry	2	24
ORL	C,bit	OR direct bit to Carry	2	24
ORL	C,/bit	OR complement of direct bit to Carry	2	24
MOV	C,bit	Move direct bit to Carry	2	12
MOV	bit,C	Move Carry to direct bit	2	24
JC	rel	Jump if Carry is set	2	24
JNC	rel	Jump if Carry not set	2	24
JB	bit,rel	Jump if direct Bit is set	3	24
JNB	bit,rel	Jump if direct Bit is Not set	3	24
JBC	bit,rel	Jump if direct Bit is set & clear bit	3	24

PROGRAM BRANCHING

Mnemonic		Description	Byte	Oscillator Period
ACALL	addr11	Absolute Subroutine Call	2	24
LCALL	addr16	Long Subroutine Call	3	24
RET		Return from Subroutine	1	24

PROGRAM BRANCHING Cont.

Mnemonic		Description	Byte	Oscillator Period
RETI		Return from interrupt	1	24
AJMP	addr11	Absolute Jump	2	24
LJMP	addr16	Long Jump	3	24
SJMP	rel	Short Jump (relative addr)	2	24
JMP	@A+DPTR	Jump indirect relative to the DPTR	1	24
JZ	rel	Jump if Accumulator is Zero	2	24
JNZ	rel	Jump if Accumulator is Not Zero	2	24
CJNE	A,direct,rel	Compare direct byte to Acc and Jump if Not Equal	3	24
CJNE	A,#data,rel	Compare immediate to Acc and Jump if Not Equal	3	24
CJNE	Rn,#data,rel	Compare immediate to register and Jump If Not Equal	3	24
CJNE	@Ri,#data,rel	Compare immediate to indirect and Jump if Not Equal	3	24
DJNZ	Rn,rel	Decrement register and Jump if Not Zero	3	24
DJNZ	direct,rel	Decrement direct byte and Jump if Not Zero	3	24
NOP		No Operation	1	12

ACALL addr11

Function: Absolute Call

Description: ACALL unconditionally calls a subroutine located at the indicated address. The instruction increments the PC twice to obtain the address of the following instruction, then pushes the 16-bit result onto the stack (low-order byte first) and increments the stack pointer twice. The destination address is obtained by successively concatenating the five high-order bits of the incremented PC, op-code bits 7-5, and the second byte of the instruction. The subroutine called must therefore start within the same 2K block of the program memory as the first byte of the instruction following ACALL. No flags are affected.

Example: Initially SP equals 07H. The label "SUBRTN" is at program memory location 0345H. After executing the instruction,

```
ACALL    SUBRTN
```

at location 0123H, SP will contain 09H, internal RAM locations 08H and 09H will contain 25H and 01H, respectively, and the PC will contain 0345H.

Bytes: 2

Cycles: 2

Encoding:

a10	a9	a8	1	0	0	0	1

a7	a6	a5	a4	a3	a2	a1	a0

Operation:

ACALL
(PC) ← (PC) + 2
(SP) ← (SP) + 1
((SP)) ← (PC_{7-0})
(SP) ← (SP) + 1
((SP)) ← (PC_{15-8})
(PC_{10-0}) ← page address

ADD A,<src-byte>

Function: Add

Description: ADD adds the byte variable indicated to the accumulator, leaving the result in the accumulator. The carry and auxiliary-carry flags are set, respectively, if there is a carry-out from bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occurred.

OV is set if there is a carry-out of bit 6 but not out of bit 7, or a carry-out of bit 7 but not bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands, or a positive sum from two negative operands.

Four source operand addressing modes are allowed: register, direct, register-indirect, or immediate.

Example: The accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B). The instruction,

```
ADD     A,R0
```

will leave 6DH (01101101B) in the accumulator with the AC flag cleared and both the carry flag and OV set to 1.

ADD A,Rn

Bytes: 1

Cycles: 1

Encoding:

0	0	1	0	1	r	r	r

Operation: ADD

(A) ← (A) + (Rn)

ADD A,direct

Bytes: 2

Cycles: 1

Encoding:

0	0	1	0	0	1	0	1	direct address

Operation: ADD

(A) ← (A) + (direct)

ADD A,@Ri

Bytes: 1
Cycles: 1

Encoding:

0	0	1	0	0	1	1	i

Operation: ADD
(A) ← (A) + ((Ri))

ADD A,#data

Bytes: 2
Cycles: 1

Encoding:

0	0	1	0	0	1	0	0	immediate data

Operation: ADD
(A) ← (A) + #data

ADDC A, <src-byte>

Function: Add with Carry

Description: ADDC simultaneously adds the byte variable indicated, the carry flag and the accumulator contents, leaving the result in the accumulator. The carry and auxiliary-carry flags are set, respectively, if there is a carry-out from bit 7 or bit 3, and cleared otherwise. When adding unsigned integers, the carry flag indicates an overflow occurred.

OV is set if there is a carry-out of bit 6 but not out of bit 7, or a carryout of bit 7 but not out of bit 6; otherwise OV is cleared. When adding signed integers, OV indicates a negative number produced as the sum of two positive operands or a positive sum from two negative operands.

Four source operand addressing modes are allowed: register, direct, register-indirect, or immediate.

Example: The accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B) with the carry flag set. The instruction,

ADDC A,R0

will leave 6EH (01101110B) in the accumulator with AC cleared and both the carry flag and OV set to 1.

ADDC A,Rn

Bytes: 1

Cycles: 1

Encoding:

0 0 1 1	1 r r r

Operation: ADDC

$(A) \leftarrow (A) + (C) + (R_n)$

ADDC A,direct

Bytes: 2

Cycles: 1

Encoding:

0 0 1 1	0 1 0 1	direct address

Operation: ADDC

$(A) \leftarrow (A) + (C) + (direct)$

ADDC A,@Ri

Bytes: 1
Cycles: 1

Encoding:

0	0	1	1	0	1	1	i

Operation: ADDC
(A)←(A) + (C) + ((Ri))

ADDC A,#data

Bytes: 2
Cycles: 1

Encoding:

0	0	1	1	0	1	0	0	immediate data

Operation: ADDC
(A)←(A) + (C) + #data

AJMP addr11

Function: Absolute Jump

Description: AJMP transfers program execution to the indicated address, which is formed at run-time by concatenating the high-order five bits of the PC (*after* incrementing the PC twice), opcode bits 7-5, and the second byte of the instruction. The destination must therefore be within the same 2K block of program memory as the first byte of the instruction following AJMP.

Example: The label "JMPADR" is at program memory location 0123H. The instruction,

AJMP JMPADR

is at location 0345H and will load the PC with 0123H.

Bytes: 2
Cycles: 2

Encoding:

a10	a9	a8	0	0	0	0	1	a7	a6	a5	a4	a3	a2	a1	a0

Operation: AJMP
(PC)←(PC) + 2
(PC10-0)← page address

ANL <dest-byte> , <src-byte>

Function: Logical-AND for byte variables

Description: ANL performs the bitwise logical-AND operation between the variables indicated and stores the results in the destination variable. No flags are affected.

The two operands allow six addressing mode combinations. When the destination is the accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the accumulator or immediate data.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

Example: If the accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B) then the instruction,

```
ANL     A,R0
```

will leave 41H (01000001B) in the accumulator.

When the destination is a directly addressed byte, this instruction will clear combinations of bits in any RAM location or hardware register. The mask byte determining the pattern of bits to be cleared would either be a constant contained in the instruction or a value computed in the accumulator at run-time. The instruction,

```
ANL     P1,#01110011B
```

will clear bits 7, 3, and 2 of output port 1.

ANL A,Rn

Bytes: 1

Cycles: 1

Encoding:

0	1	0	1	1	r	r	r

Operation:

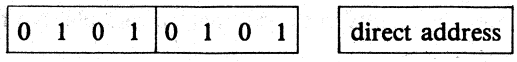
ANL

(A) ← (A) ∧ (Rn)

ANL A,direct

Bytes: 2
Cycles: 1

Encoding:

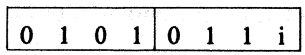


Operation: ANL
(A) ← (A) ∧ (direct)

ANL A,@Ri

Bytes: 1
Cycles: 1

Encoding:

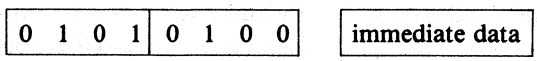


Operation: ANL
(A) ← (A) ∧ ((Ri))

ANL A,#data

Bytes: 2
Cycles: 1

Encoding:

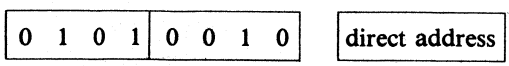


Operation: ANL
(A) ← (A) ∧ #data

ANL direct,A

Bytes: 2
Cycles: 1

Encoding:

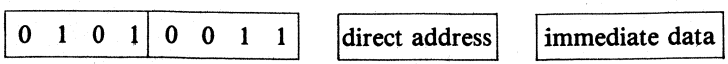


Operation: ANL
(direct) ← (direct) ∧ (A)

ANL direct,#data

Bytes: 3
Cycles: 2

Encoding:



Operation: ANL
(direct) ← (direct) ∧ #data

ANL C, <src-bit>

Function: Logical-AND for bit variables

Description: If the Boolean value of the source bit is a logical 0 then clear the carry flag; otherwise leave the carry flag in its current state. A slash ("/") preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, *but the source bit itself is not affected.* No other flags are affected.

Only direct bit addressing is allowed for the source operand.

Example: Set the carry flag if, and only if, P1.0 = 1, ACC. 7 = 1, and OV = 0:

```
MOV  C,P1.0        ;LOAD CARRY WITH INPUT PIN STATE
ANL  C,ACC.7       ;AND CARRY WITH ACCUM. BIT 7
ANL  C,/OV         ;AND WITH INVERSE OF OVERFLOW
                   FLAG
```

ANL C,bit

Bytes: 2

Cycles: 2

Encoding:

1 0 0 0	0 0 1 0	bit address

Operation: ANL

(C) ← (C) ∧ (bit)

ANL C,/bit

Bytes: 2

Cycles: 2

Encoding:

1 0 1 1	0 0 0 0	bit address

Operation: ANL

(C) ← (C) ¬ (bit)

CJNE <dest-byte>,<src-byte>, rel

Function: Compare and Jump if Not Equal.

Description: CJNE compares the magnitudes of the first two operands, and branches if their values are not equal. The branch destination is computed by adding the signed relative-displacement in the last instruction byte to the PC, after incrementing the PC to the start of the next instruction. The carry flag is set if the unsigned integer value of <dest-byte> is less than the unsigned integer value of <src-byte>; otherwise, the carry is cleared. Neither operand is affected.

The first two operands allow four addressing mode combinations: the accumulator may be compared with any directly addressed byte or immediate data, and any indirect RAM location or working register can be compared with an immediate constant.

Example: The accumulator contains 34H. Register 7 contains 56H. The first instruction in the sequence,

```
                CJNE    R7,#60H, NOT_EQ
;               ...     .....                   ; R7 = 60H.
NOT_EQ:         JC      REQ_LOW                 ; IF R7 < 60H.
;               ...     .....                   ; R7 > 60H.
```

sets the carry flag and branches to the instruction at label NOT_EQ. By testing the carry flag, this instruction determines whether R7 is greater or less than 60H.

If the data being presented to port 1 is also 34H, then the instruction,

```
WAIT:  CJNE  A,P1,WAIT
```

clears the carry flag and continues with the next instruction in sequence, since the accumulator does equal the data read from P1. (If some other value was being input on P1, the program will loop at this point until the P1 data changes to 34H.)

CJNE A,direct,rel

Bytes: 3

Cycles: 2

Encoding:

1 0 1 1	0 1 0 1	direct address	rel. address

Operation:

```
CJNE
(PC) ← (PC) + 3
 IF (direct) < (A)
    THEN  (PC) ← (PC) + rel and (C) ← 0
                OR
 IF (direct) < (A)
    THEN (PC) ← (PC) + rel and (C) ← 1
```

CJNE A,#data,rel

Bytes: 3
Cycles: 2
Encoding:

1 0 1 1 0 1 0 0	immediate data	rel. address

Operation:

```
CJNE
(PC) ←— (PC) + 3
 IF  #data < (A)
     THEN  (PC) ←— (PC) + rel and (C) ←— 0
           OR
 IF  #data > (A)
     THEN  (PC) ←— (PC) + rel and (C) ←— 1
```

CJNE Rn,#data,rel

Bytes: 3
Cycles: 2
Encoding:

1 0 1 1 1 r r r	immediate data	rel. address

Operation:

```
CJNE
(PC) ←— (PC) + 3
 IF  #data < (Rn)
     THEN  (PC) ←— (PC) + rel and (C) ←— 0
           OR
 IF  #data > (Rn)
     THEN  (PC) ←— (PC) + rel and (C) ←— 1
```

CJNE @Ri,#data,rel

Bytes: 3
Cycles: 2
Encoding:

1 0 1 1 0 1 1 i	immediate data	rel. address

Operation:

```
CJNE
(PC) ←— (PC) + 3
 IF  #data < ((Ri))
     THEN  (PC) ←— (PC) + rel and (C) ←— 1
           OR
 IF  #data > ((Ri))
     THEN  (PC) ←— (PC) + rel and (C) ←— 0
```

CLR A

Function: Clear Accumulator
Description: The accumulator is cleared (all bits set to zero). No flags are affected.
Example: The accumulator contains 5CH (01011100B). The instruction,

CLR A

will leave the accumulator set to 00H (00000000B).
Bytes: 1
Cycles: 1

Encoding:

1	1	1	0	0	1	0	0

Operation: CLR
(A)←0

CLR bit

Function: Clear bit
Description: The indicated bit is cleared (reset to zero). No other flags are affected. CLR can operate on the carry flag or any directly addressable bit.
Example: Port 1 has previously been written with 5DH (01011101B). The instruction,

CLR P1.2

will leave the port set to 59H (01011001B).

CLR C

Bytes: 1
Cycles: 1

Encoding:

1	1	0	0	0	0	1	1

Operation: CLR
(C)←0

CLR bit

Bytes: 2
Cycles: 1

Encoding:

1	1	0	0	0	0	1	0	bit address

Operation: CLR
(bit)←0

CPL A

Function: Complement Accumulator

Description: Each bit of the accumulator is logically complemented (one's complement). Bits which previously contained a one are changed to zero and vice-versa. No flags are affected.

Example: The accumulator contains 5CH (01011100B). The instruction,

```
CPL     A
```

will leave the accumulator set to 0A3H (10100011B).

Bytes: 1

Cycles: 1

Encoding:

1	1	1	1	0	1	0	0

Operation:

```
CPL
(A)←¬(A)
```

CPL bit

Function: Complement bit

Description: The bit variable specified is complemented. A bit which had been a one is changed to zero and vice-versa. No other flags are affected. CLR can operate on the carry or any directly addressable bit.

Note: When this instruction is used to modify an output pin, the value used as the original data will be read from the output data latch, *not* the input pin.

Example: Port 1 has previously been written with 5BH (01011101B). The instruction sequence,

```
CPL     P1.1
CPL     P1.2
```

will leave the port set to 5BH (01011011B).

CPL C

Bytes: 1

Cycles: 1

Encoding:

1	0	1	1	0	0	1	1

Operation:

```
CPL
(C)←¬(C)
```

CPL bit

Bytes: 2
Cycles: 1

Encoding:

1	0	1	1	0	0	1	0	bit address

Operation: CPL
(bit)←¬(bit)

DA A

Function: Decimal-adjust Accumulator for Addition

Description: DA A adjusts the eight-bit value in the accumulator resulting from the earlier addition of two variables (each in packed-BCD format), producing two four-bit digits. Any ADD or ADDC instruction may have been used to perform the addition.

If accumulator bits 3-0 are greater than nine (xxxx1010-xxxx1111), or if the AC flag is one, six is added to the accumulator producing the proper BCD digit in the low-order nibble. This internal addition would set the carry flag if a carry-out of the low-order four-bit field propagated through all high-order bits, but it would not clear the carry flag otherwise.

If the carry flag is now set, or if the four high-order bits now exceed nine (1010xxxx-1111xxxx), these high-order bits are incremented by six, producing the proper BCD digit in the high-order nibble. Again, this would set the carry flag if there was a carry-out of the high-order bits, but wouldn't clear the carry. The carry flag thus indicates if the sum of the original two BCD variables is greater than 100, allowing multiple precision decimal addition. OV is not affected.

All of this occurs during the one instruction cycle. Essentially, this instruction performs the decimal conversion by adding 00H, 06H, 60H, or 66H to the accumulator, depending on initial accumulator and PSW conditions.

Note: DA A *cannot* simply convert a hexadecimal number in the accumulator to BCD notation, nor does DA A apply to decimal subtraction.

Example: The accumulator holds the value 56H (01010110B) representing the packed BCD digits of the decimal number 56. Register 3 contains the value 67H (01100111B) representing the packed BCD digits of the decimal number 67. The carry flag is set. The instruction sequence,

```
ADDC    A,R3
DA      A
```

will first perform a standard twos-complement binary addition, resulting in the value 0BEH (10111110) in the accumulator. The carry and auxiliary carry flags will be cleared.

The Decimal Adjust instruction will then alter the accumulator to the value 24H (00100100B), indicating the packed BCD digits of the decimal number 24, the low-order two digits of the decimal sum of 56, 67, and the carry-in. The carry flag will be set by the Decimal Adjust instruction, indicating that a decimal overflow occurred. The true sum 56, 67, and 1 is 124.

BCD variables can be incremented or decremented by adding 01H or 99H. If the accumulator initially holds 30H (representing the digits of 30 decimal), then the instruction sequence,

```
ADD     A,#99H
DA      A
```

will leave the carry set and 29H in the accumulator, since 30 + 99 = 129. The low-order byte of the sum can be interpreted to mean 30 − 1 = 29.

Bytes: 1

Cycles: 1

Encoding:

1	1	0	1	0	1	0	0

Operation:

```
DA
-contents of Accumulator are BCD
IF  [[(A3-0) > 9] ∨ [(AC) = 1]]
    THEN  (A3-0) ← (A3-0) + 6
          AND
IF  [[(A7-4) >9] ∨ [(C) = 1]]
    THEN  (A7-4) ← (A7-4) + 6
```

DEC byte

Function: Decrement

Description: The variable indicated is decremented by 1. An original value of 00H will underflow to 0FFH. No flags are affected. Four operand addressing modes are allowed: accumulator, register, direct, or register-indirect.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

Example: Register 0 contains 7FH (01111111B). Internal RAM locations 7EH and 7FH contain 00H and 40H, respectively. The instruction sequence,

```
DEC  @R0
DEC  R0
DEC  @R0
```

will leave register 0 set to 7EH and internal RAM locations 7EH and 7FH set to 0FFH and 3FH.

DEC A

Bytes: 1

Cycles: 1

Encoding:

0 0 0 1	0 1 0 0

Operation: DEC

(A)←(A) − 1

DEC Rn

Bytes: 1

Cycles: 1

Encoding:

0 0 0 1	1 r r r

Operation: DEC

(Rn)←(Rn) − 1

DEC direct

Bytes: 2
Cycles: 1

Encoding:

0 0 0 1	0 1 0 1	direct address

Operation: DEC
(direct) ← (direct) − 1

DEC @Ri

Bytes: 1
Cycles: 1

Encoding:

0 0 0 1	0 1 1 i

Operation: DEC
((Ri)) ← ((Ri)) − 1

DIV AB

Function: Divide

Description: DIV AB divides the unsigned eight-bit integer in the accumulator by the unsigned eight-bit integer in register B. The accumulator receives the integer part of the quotient; register B receives the integer remainder. The carry and OV flags will be cleared.

Exception: if B had originally contained 00H, the values returned in the accumulator and B-register will be undefined and the overflow flag will be set. The carry flag is cleared in any case.

Example: The accumulator contains 251 (0FBH or 11111011B) and B contains 18 (12H or 00010010B). The instruction,

DIV AB

will leave 13 in the accumulator (0DH or 00001101B) and the value 17 (11H or 00010001B) in B, since 251 = (13 x 18) + 17. Carry and OV will both be cleared.

Bytes: 1
Cycles: 4

Encoding:

1 0 0 0	0 1 0 0

Operation: DIV
$(A)_{15-8} \leftarrow (A) / (B)$
$(B)_{7-0}$

DJNZ <byte>,<rel-addr>

Function: Decrement and Jump if Not Zero

Description: DJNZ decrements the location indicated by 1, and branches to the address indicated by the second operand if the resulting value is not zero. An original value of 00H will underflow to 0FFH. No flags are affected. The branch destination would be computed by adding the signed relative-displacement value in the last instruction byte to the PC, after incrementing the PC to the first byte of the following instruction.

The location decremented may be a register or directly addressed byte.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

Example: Internal RAM locations 40H, 50H, and 60H contain the values 01H, 70H, and 15H, respectively. The instruction sequence,

```
DJNZ    40H,LABEL_1
DJNZ    50H,LABEL_2
DJNZ    60H,LABEL_3
```

will cause a jump to the instruction at label LABEL_2 with the values 00H, 6FH, and 15H in the three RAM locations. The first jump was *not* taken because the result was zero.

This instruction provides a simple way of executing a program loop a given number of times, or for adding a moderate time delay (from 2 to 512 machine cycles) with a single instruction. The instruction sequence,

```
        MOV   R2,#8
TOGGLE: CPL   P1.7
        DJNZ  R2,TOGGLE
```

will toggle P1.7 eight times, causing four output pulses to appear at bit 7 of output port 1. Each pulse will last three machine cycles; two for DJNZ and one to alter the pin.

DJNZ Rn,rel

Bytes: 2
Cycles: 2

Encoding:

1 1 0 1	1 r r r	direct address

Operation:

```
DJNZ
(PC)←(PC) + 2
(Rn)←(Rn) − 1
IF (Rn) > 0 or (Rn) < 0
    THEN
        (PC)←(PC) + rel
```

DJNZ direct,rel

Bytes: 3
Cycles: 2

Encoding:

1 1 0 1	0 1 0 1	direct address	rel. address

Operation:

```
DJNZ
(PC)←(PC) + 2
(direct)←(direct) − 1
IF (direct) > 0 or (direct) < 0
    THEN
        (PC)←(PC) + rel
```

INC <byte>

Function: Increment

Description: INC increments the indicated variable by 1. An original value of 0FFH will overflow to 00H. No flags are affected. Three addressing modes are allowed: register, direct, or register-indirect.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

Example: Register 0 contains 7EH (011111110B). Internal RAM locations 7EH and 7FH contain 0FFH and 40H, respectively. The instruction sequence,

```
INC  @R0
INC  R0
INC  @R0
```

will leave register 0 set to 7FH and internal RAM locations 7EH and 7FH holding (respectively) 00H and 41H.

INC A

Bytes: 1

Cycles: 1

Encoding:

0 0 0 0	0 1 0 0

Operation: INC

(A) ← (A) + 1

INC Rn

Bytes: 1

Cycles: 1

Encoding:

0 0 0 0	1 r r r

Operation: INC

(Rn) ← (Rn) + 1

INC direct

Bytes: 2

Cycles: 1

Encoding:

0 0 0 0	0 1 0 1	direct address

Operation: INC

(direct) ← (direct) + 1

INC @Ri

Bytes: 1
Cycles: 1

Encoding:

0 0 0 0	0 1 1 i

Operation: INC
((Ri)) ← ((Ri)) + 1

INC DPTR

Function: Increment Data Pointer

Description: Increment the 16-bit data pointer by 1. A 16-bit increment (modulo 2^{16}) is performed; an overflow of the low-order byte of the data pointer (DPL) from 0FFH to 00H will increment the high-order byte (DPH). No flags are affected.

This is the only 16-bit register which can be incremented.

Example: Registers DPH and DPL contain 12H and 0FEH, respectively. The instruction sequence,

```
INC     DPTR
INC     DPTR
INC     DPTR
```

will change DPH and DPL to 13H and 01H.

Bytes: 1
Cycles: 2

Encoding:

1 0 1 0	0 0 1 1

Operation: INC
(DPTR) ← (DPTR) + 1

JB bit,rel

Function: Jump if Bit set

Description: If the indicated bit is a one, jump to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. *The bit tested is not modified.* No flags are affected.

Example: The data present at input port 1 is 11001010B. The accumulator holds 56 (01010110B). The instruction sequence,

```
JB    P1.2,LABEL1
JB    ACC.2,LABEL2
```

will cause program execution to branch to the instruction at label LABEL2.

Bytes: 3

Cycles: 2

Encoding:

0 0 1 0	0 0 0 0	bit address	rel. address

Operation:

```
JB
(PC) ← (PC) + 3
IF  (bit) = 1
    THEN
            (PC) ← (PC) + rel
```

JBC bit,rel

Function: Jump if Bit is set and Clear bit

Description: If the indicated bit is one, branch to the address indicated; otherwise proceed with the next instruction. *In either case, clear the designated bit.* The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. No flags are affected.

Note: When this instruction is used to test an output pin, the value used as the original data will be read from the output data latch, *not* the input pin.

Example: The accumulator holds 56H (01010110B). The instruction sequence,

```
JBC   ACC.3,LABEL1
JBC   ACC.2,LABEL2
```

will cause program execution to continue at the instruction identified by the label LABEL2, with the accumulator modified to 52H (01010010B).

Bytes: 3
Cycles: 2

Encoding:

0 0 0 1	0 0 0 0	bit address	rel. address

Operation:

```
JBC
(PC) ←— (PC) + 3
IF  (bit) = 1
    THEN
            (bit) ←— 0
            (PC) ←— (PC) + rel
```

JC rel

Function: Jump if Carry is set

Description: If the carry flag is set, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. No flags are affected.

Example: The carry flag is cleared. The instruction sequence,

```
JC    LABEL1
CPL   C
JC    LABEL2
```

will set the carry and cause program execution to continue at the instruction identified by the label LABEL2.

Bytes: 2
Cycles: 2

Encoding:

0 1 0 0	0 0 0 0	rel. address

Operation:

```
JC
(PC) ←— (PC) + 2
IF  (C) = 1
    THEN
            (PC) ←— (PC) + rel
```

JMP @A + DPTR

Function: Jump indirect

Description: Add the eight-bit unsigned contents of the accumulator with the sixteen-bit data pointer, and load the resulting sum to the program counter. This will be the address for subsequent instruction fetches. Sixteen-bit addition is performed (modulo 2^{16}): a carry-out from the low-order eight bits propagates through the higher-order bits. Neither the accumulator nor the data pointer is altered. No flags are affected.

Example: An even number from 0 to 6 is in the accumulator. The following sequence of instructions will branch to one of four AJMP instructions in a jump table starting at JMP_TBL:

```
        MOV     DPTR,#JMP_TBL
        JMP     @A+DPTR
JMP_TBL: AJMP   LABEL0
        AJMP    LABEL1
        AJMP    LABEL2
        AJMP    LABEL3
```

If the accumulator equals 04H when starting this sequence, execution will jump to label LABEL2. Remember that AJMP is a two-byte instruction, so the jump instructions start at every other address.

Bytes: 1

Cycles: 2

Encoding:

0	1	1	1	0	0	1	1

Operation: JMP

(PC) ← (A) + (DPTR)

JNB bit,rel

Function: Jump if Bit Not set

Description: If the indicated bit is a zero, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. *The bit tested is not modified.* No flags are affected.

Example: The data present at input port 1 is 11001010B. The accumulator holds 56H (01010110B). The instruction sequence,

```
JNB     P1.3,LABEL1
JNB     ACC.3,LABEL2
```

will cause program execution to continue at the instruction at label LABEL2.

Bytes: 3

Cycles: 2

Encoding:

0 0 1 1	0 0 0 0	bit address	rel. address

Operation:

```
JNB
(PC) ← (PC) + 3
IF (bit) = 0
    THEN (PC) ← (PC) + rel.
```

JNC rel

Function: Jump if Carry not set

Description: If the carry flag is a zero, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice to point to the next instruction. The carry flag is not modified.

Example: The carry flag is set. The instruction sequence,

```
JNC   LABEL1
CPL   C
JNC   LABEL2
```

will clear the carry and cause program execution to continue at the instruction identified by the label LABEL2.

Bytes: 2
Cycles: 2

Encoding:

0 1 0 1	0 0 0 0	rel. address

Operation:
```
JNC
(PC)←(PC) + 2
IF  (C) = 0
    THEN  (PC)←(PC) + rel
```

JNZ rel

Function: Jump if accumulator Not Zero

Description: If any bit of the accumulator is a one, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The accumulator is not modified. No flags are affected.

Example: The accumulator originally holds 00H. The instruction sequence,

```
JNZ  LABEL1
INC  A
JNZ  LABEL2
```

will set the accumulator to 01H and continue at label LABEL2.

Bytes: 2
Cycles: 2

Encoding:

0 1 1 1	0 0 0 0	rel. address

Operation:
```
JNZ
(PC) ←— (PC) + 2
IF  (A) ≠0
    THEN  (PC)←— (PC) + rel
```

JZ rel

Function: Jump if Accumulator Zero

Description: If all bits of the accumulator are zero, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The accumulator is not modified. No flags are affected.

Example: The accumulator originally contains 01H. The instruction sequence,

```
JZ    LABEL1
DEC   A
JZ    LABEL2
```

will change the accumulator to 00H and cause program execution to continue at the instruction identified by the label LABEL2.

Bytes: 2

Cycles: 2

Encoding:

0 1 1 0	0 0 0 0	rel. address

Operation:

```
JZ
(PC)←—(PC) + 2
IF  (A) = 0
    THEN  (PC)←—(PC) + rel
```

LCALL addr16

Function: Long Call

Description: LCALL calls a subroutine located at the indicated address. The instruction adds three to the program counter to generate the address of the next instruction and then pushes the 16-bit result onto the stack (low byte first), incrementing the stack pointer by two. The high-order and low-order bytes of the PC are then loaded, respectively, with the second and third bytes of the LCALL instruction. Program execution continues with the instruction at this address. The subroutine may therefore begin anywhere in the full 64K-byte program memory address space. No flags are affected.

Example: Initially the stack pointer equals 07H. The label "SUBRTN" is assigned to program memory location 1234H. After executing the instruction,

LCALL SUBRTN

at location 0123H, the stack pointer will contain 09H, internal RAM locations 08H and 09H will contain 26H and 01H, and the PC will contain 1235H.

Bytes: 3

Cycles: 2

Encoding:

0 0 0 1	0 0 1 0	addr15 - addr8	addr7 - addr0

Operation:

LCALL
(PC) ← (PC) + 3
(SP) ← (SP) + 1
((SP)) ← (PC_{7-0})
(SP) ← (SP) + 1
((SP)) ← (PC_{15-8})
(PC) ← $addr_{15-0}$

LJMP addr16

Function: Long Jump

Description: LJMP causes an unconditional branch to the indicated address, by loading the high-order and low-order bytes of the PC (respectively) with the second and third instruction bytes. The destination may therefore be anywhere in the full 64K program memory address space. No flags are affected.

Example: The label "JMPADR" is assigned to the instruction at program memory location 1234H. The instruction,

LJMP JMPADR

at location 0123H will load the program counter with 1234H.

Bytes: 3

Cycles: 2

Encoding:

0 0 0 0	0 0 1 0	addr15 - addr8	addr7 - addr0

Operation:

LJMP
(PC) ← $addr_{15-0}$

MOV <dest-byte>,<src-byte>

Function: Move byte variable

Description: The byte variable indicated by the second operand is copied into the location specified by the first operand. The source byte is not affected. No other register or flag is affected.

This is by far the most flexible operation. Fifteen combinations of source and destination addressing modes are allowed.

Example: Internal RAM location 30H holds 40H. The value of RAM location 40H is 10H. The data present at input port 1 is 11001010B (0CAH).

```
MOV   R0,#30H          ;R0<= 30H
MOV   A,@R0            ;A < = 40H
MOV   R1,A             ;R1 < = 40H
MOV   R,@R1            ;B < = 10H
MOV   @R1,P1           ;RAM (40H) < = 0CAH
MOV   P2, P1           ;P2 #0CAH
```

leaves the value 30H in register 0, 40H in both the accumulator and register 1, 10H in register B, and 0CAH (11001010B) both in RAM location 40H and output on port 2.

MOV A,Rn

Bytes: 1

Cycles: 1

Encoding:

1	1	1	0	1	r	r	r

Operation:
MOV
(A) ← (Rn)

*MOV A,direct

Bytes: 2

Cycles: 1

Encoding:

1	1	1	0	0	1	0	1	direct address

Operation:
MOV
(A) ← (direct)

***MOV A,ACC is not a valid instruction.**

MOV A,@Ri

Bytes: 1
Cycles: 1

Encoding:

1	1	1	0	0	1	1	i

Operation: MOV
(A) ← ((Ri))

MOV A,#data

Bytes: 2
Cycles: 1

Encoding:

0	1	1	1	0	1	0	0	immediate data

Operation: MOV
(A) ← #data

MOV Rn,A

Bytes: 1
Cycles: 1

Encoding:

1	1	1	1	1	r	r	r

Operation: MOV
(Rn) ← (A)

MOV Rn,direct

Bytes: 2
Cycles: 2

Encoding:

1	0	1	0	1	r	r	r	direct addr.

Operation: MOV
(Rn) ← (direct)

MOV Rn,#data

Bytes: 2
Cycles: 1

Encoding:

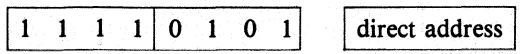
0	1	1	1	1	r	r	r	immediate data

Operation: MOV
(Rn) ← #data

MOV direct,A

Bytes: 2
Cycles: 1

Encoding:

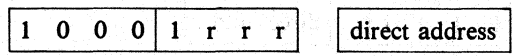


Operation: MOV
(direct)←(A)

MOV direct,Rn

Bytes: 2
Cycles: 2

Encoding:

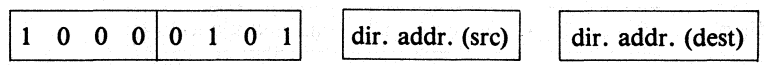


Operation: MOV
(direct)←(Rn)

MOV direct,direct

Bytes: 3
Cycles: 2

Encoding:

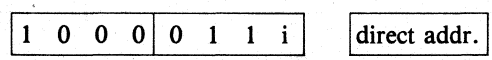


Operation: MOV
(direct)←(direct)

MOV direct,@Ri

Bytes: 2
Cycles: 2

Encoding:

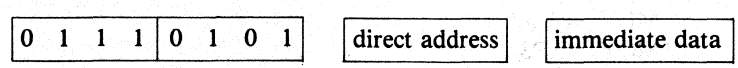


Operation: MOV
(direct)←((Ri))

MOV direct,#data

Bytes: 3
Cycles: 2

Encoding:



Operation: MOV
(direct)←#data

MOV @Ri,A

Bytes: 1
Cycles: 1

Encoding:

1 1 1 1	0 1 1 i

Operation: MOV
((Ri)) ← (A)

MOV @Ri,direct

Bytes: 2
Cycles: 2

Encoding:

1 0 1 0	0 1 1 i	direct addr.

Operation: MOV
((Ri)) ← (direct)

MOV @Ri,#data

Bytes: 2
Cycles: 1

Encoding:

0 1 1 1	0 1 1 i	immediate data

Operation: MOV
((RI)) ← #data

MOV <dest-bit>,<src-bit>

Function: Move bit data

Description: The Boolean variable indicated by the second operand is copied into the location specified by the first operand. One of the operands must be the carry flag; the other may be any directly addressable bit. No other register or flag is affected.

Example: The carry flag is originally set. The data present at input port 3 is 11000101B. The data previously written to output port 1 is 35H (00110101B).

```
MOV  P1.3,C
MOV  C,P3.3
MOV  P1.2,C
```

will leave the carry cleared and change port 1 to 39H (00111001B).

MOV C,bit

Bytes: 2
Cycles: 1

Encoding:

1 0 1 0	0 0 1 0	bit address

Operation: MOV
(C)←(bit)

MOV bit,C

Bytes: 2
Cycles: 2

Encoding:

1 0 0 1	0 0 1 0	bit address

Operation: MOV
(bit)←(C)

MOV DPTR,#data16

Function: Load Data Pointer with a 16-bit constant

Description: The data pointer is loaded with the 16-bit constant indicated. The 16-bit constant is loaded into the second and third bytes of the instruction. The second byte (DPH) is the high-order byte, while the third byte (DPL) holds the low-order byte. No flags are affected.

Example: This is the only instruction which moves 16 bits of data at once. The instruction,

MOV DPTR,#1234H

will load the value 1234H into the data pointer: DPH will hold 12H and DPL will hold 34H.

Bytes: 3
Cycles: 2

Encoding:

1 0 0 1	0 0 0 0	immed. data15 - 8	immed. data7 - 0

Operation: MOV
(DPTR)←#data$_{15\text{-}0}$
DPH □ DPL←#data$_{15\text{-}8}$ □ #data$_{7\text{-}0}$

MOVC A,@A + <base-reg>

Function: Move Code byte

Description: The MOVC instructions load the accumulator with a code byte, or constant from program memory. The address of the byte fetched is the sum of the original unsigned eight-bit accumulator contents and the contents of a sixteen-bit base register, which may be either the data pointer or the PC. In the latter case, the PC is incremented to the address of the following instruction before being added with the accumulator: otherwise the base register is not altered. Sixteen-bit addition is performed so a carry-out from the low-order eight bits may propagate through higher-order bits. No flags are affected.

Example: A value between 0 and 3 is in the accumulator. The following instructions will translate the value in the accumulator to one of four values defined by the DB (define byte) directive.

```
REL_PC: INC     A
        MOVC    A,@A+PC
        RET
        DB      66H
        DB      77H
        DB      88H
        DB      99H
```

If the subroutine is called with the accumulator equal to 01H, it will return with 77H in the accumulator. The INC A before the MOVC instruction is needed to "get around" the RET instruction above the table. If several bytes of code separated the MOVC from the table, the corresponding number would be added to the accumulator instead.

MOVC A,@A+DPTR

Bytes: 1

Cycles: 2

Encoding:

1 0 0 1	0 0 1 1

Operation: MOVC

(A) ← ((A) + (DPTR))

MOVC A,@A+PC

Bytes: 1
Cycles: 2

Encoding:

1	0	0	0	0	0	1	1

Operation: MOVC
(PC)←(PC) + 1
(A)←((A) + (PC))

MOVX ⟨**dest-byte**⟩,⟨**src-byte**⟩

Function: Move External

Description: The MOVX instructions transfer data between the accumulator and a byte of external data memory, hence the "X" appended to MOV. There are two types of instructions, differing in whether they provide an eight-bit or sixteen-bit indirect address to the external data RAM.

In the first type, the contents of R0 or R1 in the current register bank provide an eight-bit address multiplexed with data on P0. Eight bits are sufficient for external I/O expansion decoding or a relatively small RAM array. For somewhat larger arrays, any output port pins can be used to output higher-order address bits. These pins would be controlled by an output instruction preceding the MOVX.

In the second type of MOVX instruction, the data pointer generates a sixteen-bit address. P2 outputs the high-order eight address bits (the contents of DPH) while P0 multiplexes the low-order eight bits (DPL) with data. The P2 Special Function Register retains its previous contents while the P2 output buffers are emitting the contents of DPH. This form is faster and more efficient when accessing very large data arrays (up to 64K bytes), since no additional instructions are needed to set up the output ports.

It is possible in some situations to mix the two MOVX types. A large RAM array with its high-order address lines driven by P2 can be addressed via the data pointer, or with code to output high-order address bits to P2 followed by a MOVX instruction using R0 or R1.

Example: An external 256 byte RAM using multiplexed address/data lines (e.g., an Intel® 8155 RAM/I/O/Timer) is connected to the 8051 Port 0. Port 3 provides control lines for the external RAM. Ports 1 and 2 are used for normal I/O. Registers 0 and 1 contain 12H and 34H. Location 34H of the external RAM holds the value 56H. The instruction sequence,

```
MOVX    A,@R1
MOVX    @R0,A
```

copies the value 56H into both the accumulator and external RAM location 12H.

MOVX A,@Ri

Bytes: 1
Cycles: 2

Encoding:

1	1	1	0	0	0	1	i

Operation: MOVX
(A) ← ((Ri))

MOVX A,@DPTR

Bytes: 1
Cycles: 2

Encoding:

1	1	1	0	0	0	0	0

Operation: MOVX
(A) ← ((DPTR))

MOVX @Ri,A

Bytes: 1
Cycles: 2

Encoding:

1	1	1	1	0	0	1	i

Operation: MOVX
((Ri)) ← (A)

MOVX @DPTR,A

Bytes: 1
Cycles: 2

Encoding:

1	1	1	1	0	0	0	0

Operation: MOVX
(DPTR) ← (A)

MUL AB

Function: Multiply

Description: MUL AB multiplies the unsigned eight-bit integers in the accumulator and register B. The low-order byte of the sixteen-bit product is left in the accumulator, and the high-order byte in B. If the product is greater than 255 (0FFH) the overflow flag is set; otherwise it is cleared. The carry flag is always cleared.

Example: Originally the accumulator holds the value 80 (50H). Register B holds the value 160 (0A0H). The instruction,

MUL AB

will give the product 12,800 (3200H), so B is changed to 32H (00110010B) and the accumulator is cleared. The overflow flag is set, carry is cleared.

Bytes: 1

Cycles: 4

Encoding:

1 0 1 0	0 1 0 0

Operation:

MUL
(A)$_{7-0}$ ← (A) X (B)
(B)$_{15-8}$

NOP

Function: No Operation

Description: Execution continues at the following instruction. Other than the PC, no registers or flags are affected.

Example: It is desired to produce a low-going output pulse on bit 7 of port 2 lasting exactly 5 cycles. A simple SETB/CLR sequence would generate a one-cycle pulse, so four additional cycles must be inserted. This may be done (assuming no interrupts are enabled) with the instruction sequence,

```
CLR     P2.7
NOP
NOP
NOP
NOP
SETB    P2.7
```

Bytes: 1

Cycles: 1

Encoding:

0	0	0	0	0	0	0	0

Operation: NOP

(PC)←(PC) + 1

ORL <dest-byte> <src-byte>

Function: Logical-OR for byte variables

Description: ORL performs the bitwise logical-OR operation between the indicated variables, storing the results in the destination byte. No flags are affected.

The two operands allow six addressing mode combinations. When the destination is the accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the accumulator or immediate data.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

Example: If the accumulator holds 0C3H (11000011B) and R0 holds 55H (01010101B) then the instruction,

```
ORL     A,R0
```

will leave the accumulator holding the value 0D7H (11010111B).

When the destination is a directly addressed byte, the instruction can set combinations of bits in any RAM location or hardware register. The pattern of bits to be set is determined by a mask byte, which may be either a constant data value in the instruction or a variable computed in the accumulator at run-time. The instruction,

```
ORL     P1,#00110010B
```

will set bits 5, 4, and 1 of output port 1.

ORL A,Rn

Bytes: 1

Cycles: 1

Encoding:

0	1	0	0	1	r	r	r

Operation: ORL

(A)←—(A) ∨ (Rn)

ORL A,direct

Bytes: 2
Cycles: 1

Encoding:

0 1 0 0	0 1 0 1	direct address

Operation: ORL
(A) ← (A) ∨ (direct)

ORL A,@Ri

Bytes: 1
Cycles: 1

Encoding:

0 1 0 0	0 1 1 i

Operation: ORL
(A) ← (A) ∨ ((Ri))

ORL A,#data

Bytes: 2
Cycles: 1

Encoding:

0 1 0 0	0 1 0 0	immediate data

Operation: ORL
(A) ← (A) ∨ #data

ORL direct,A

Bytes: 2
Cycles: 1

Encoding:

0 1 0 0	0 0 1 0	direct address

Operation: ORL
(direct) ← (direct) ∨ (A)

ORL direct,#data

Bytes: 3
Cycles: 2

Encoding:

0 1 0 0	0 0 1 1	direct addr.	immediate data

Operation: ORL
(direct) ← (direct) ∨ #data

ORL C, <src-bit>

Function: Logical-OR for bit variables

Description Set the carry flag if the Boolean value is a logical l; leave the carry in its current state otherwise. A slash ("/") preceding the operand in the assembly language indicates that the logical complement of the addressed bit is used as the source value, but the source bit itself is not affected. No other flags are affected.

Example: Set the carry flag if and only if P1.0 = 1, ACC.7 = 1, or OV = 0:

```
MOV  C,P1.0        ;LOAD CARRY WITH INPUT PIN P10
ORL  C,ACC.7       ;OR CARRY WITH THE ACC. BIT 7
ORL  C,/OV         :OR CARRY WITH THE INVERSE OF OV
```

ORL C,bit

Bytes: 2

Cycles: 2

Encoding:

0 1 1 1	0 0 1 0	bit address

Operation: ORL

(C)←(C) ∨ (bit)

ORL C,/bit

Bytes: 2

Cycles: 2

Encoding:

1 0 1 0	0 0 0 0	bit address

Operation: ORL

(C)←(C) ∨ ($\overline{\text{bit}}$)

POP direct

Function: Pop from stack.

Description: The contents of the internal RAM location addressed by the stack pointer is read, and the stack pointer is decremented by one. The value read is the transfer to the directly addressed byte indicated. No flags are affected.

Example: The stack pointer originally contains the value 32H, and internal RAM locations 30H through 32H contain the values 20H, 23H, and 01H, respectively. The instruction sequence,

```
POP     DPH
POP     DPL
```

will leave the stack pointer equal to the value 30H and the data pointer set to 0123H. At this point the instruction,

```
POP     SP
```

will leave the stack pointer set to 20H. Note that in this special case the stack pointer was decremented to 2FH before being loaded with the value popped (20H).

Bytes: 2
Cycles: 2

Encoding:

1 1 0 1	0 0 0 0	direct address

Operation:

```
POP
(direct)←((SP))
  (SP)←(SP) – 1
```

PUSH direct

Function: Push onto stack

Description: The stack pointer is incremented by one. The contents of the indicated variable is then copied into the internal RAM location addressed by the stack pointer. Otherwise no flags are affected.

Example: On entering an interrupt routine the stack pointer contains 09H. The data pointer holds the value 0123H. The instruction sequence,

```
PUSH    DPL
PUSH    DPH
```

will leave the stack pointer set to 0BH and store 23H and 01H in internal RAM locations 0AH and 0BH, respectively.

Bytes: 2
Cycles: 2

Encoding:

1 1 0 0	0 0 0 0	direct address

Operation:

```
PUSH
 (SP)←(SP) + 1
((SP))←(direct)
```

RET

Function: Return from subroutine

Description: RET pops the high- and low-order bytes of the PC successively from the stack, decrementing the stack pointer by two. Program execution continues at the resulting address, generally the instruction immediately following an ACALL or LCALL. No flags are affected.

Example: The stack pointer originally contains the value 0BH. Internal RAM locations 0AH and 0BH contain the values 23H and 01H, respectively. The instruction,

RET

will leave the stack pointer equal to the value 09H. Program execution will continue at location 0123H.

Bytes: 1

Cycles: 2

Encoding:

0	0	1	0	0	0	1	0

Operation:

RET
$(PC_{15-8}) \leftarrow ((SP))$
$(SP) \leftarrow (SP) - 1$
$(PC_{7-0}) \leftarrow ((SP))$
$(SP) \leftarrow (SP) - 1$

RETI

Function: Return from interrupt

Description: RETI pops the high- and low-order bytes of the PC successively from the stack, and restores the interrupt logic to accept additional interrupts at the same priority level as the one just processed. The stack pointer is left decremented by two. No other registers are affected; the PSW is *not* automatically restored to its pre-interrupt status. Program execution continues at the resulting address, which is generally the instruction immediately after the point at which the interrupt request was detected. If a lower- or same-level interrupt had been pending when the RETI instruction is executed, that one instruction will be executed before the pending interrupt is processed.

Example: The stack pointer originally contains the value 0BH. An interrupt was detected during the instruction ending at location 0122H. Internal RAM locations 0AH and 0BH contain the values 23H and 01H, respectively. The instruction,

RETI

will leave the stack pointer equal to 09H and return program execution to location 0123H.

Bytes: 1

Cycles: 2

Encoding:

0	0	1	1	0	0	1	0

Operation:

RETI
$(PC_{15-8}) \leftarrow ((SP))$
$(SP) \leftarrow (SP) - 1$
$(PC_{7-0}) \leftarrow ((SP))$
$(SP) \leftarrow (SP) - 1$

RL A

Function: Rotate accumulator Left

Description: The eight bits in the accumulator are rotated one bit to the left. Bit 7 is rotated into the bit 0 position. No flags are affected.

Example: The accumulator holds the value 0C5H (11000101B). The instruction,

```
RL    A
```

leaves the accumulator holding the value 8BH (10001011B) with the carry unaffected.

Bytes: 1

Cycles: 1

Encoding:

0	0	1	0	0	0	1	1

Operation:

RL

$(A_{n+1}) \leftarrow (An)$ $n = 0-6$

$(A0) \leftarrow (A7)$

RLC A

Function: Rotate accumulator Left through the Carry flag

Description: The eight bits in the accumulator and the carry flag are together rotated one bit to the left. Bit 7 moves into the carry flag; the original state of the carry flag moves into the bit 0 position. No other flags are affected.

Example: The accumulator holds the value 0C5H (11000101B), and the carry is zero. The instruction,

```
RLC    A
```

leaves the accumulator holding the value 8BH (10001010B) with the carry set

Bytes: 1

Cycles: 1

Encoding:

0	0	1	1	0	0	1	1

Operation:

RLC

$(An+1) \leftarrow (An)$ $n = 0-6$

$(A0) \leftarrow (C)$

$(C) \leftarrow (A7)$

RR A

Function: Rotate accumulator Right

Description: The eight bits in the accumulator are rotated one bit to the right. Bit 0 is rotated into the bit 7 position. No flags are affected.

Example: The accumulator holds the value 0C5H (11000101B). The instruction,

```
RR    A
```

leaves the accumulator holding the value 0E2H (11100010B) with the carry unaffected.

Bytes: 1

Cycles: 1

Encoding:

0	0	0	0	0	0	1	1

Operation:

RR

$(An) \leftarrow (A_{n+1})$ $n = 0-6$

$(A7) \leftarrow (A0)$

RRC A

Function: Rotate accumulator Right through Carry flag

Description: The eight bits in the accumulator and the carry flag are together rotated one bit to the right. Bit 0 moves into the carry flag; the original value of the carry flag moves into the bit 7 position. No other flags are affected.

Example: The accumulator holds the value 0C5H (11000101B), the carry is zero. The instruction,

```
RRC    A
```

leaves the accumulator holding the value 62 (01100010B) with the carry set.

Bytes: 1

Cycles: 1

Encoding:

0	0	0	1	0	0	1	1

Operation:

RRC

$(An) \leftarrow (A_{n+1})$ $n = 0-6$

$(A7) \leftarrow (C)$

$(C) \leftarrow (A0)$

SETB <bit>

Function: Set Bit

Description: SETB sets the indicated bit to one. SETB can operate on the carry flag or any directly addressable bit. No other flags are affected.

Example: The carry flag is cleared. Output port 1 has been written with the value 34H (00110100B). The instructions,

```
SETB    C
SETB    P1.0
```

will leave the carry flag set to 1 and change the data output on port 1 to 35H (00110101B).

SETB C

Bytes: 1

Cycles: 1

Encoding:

1 1 0 1	0 0 1 1

Operation:

```
SETB
(C)←—1
```

SETB bit

Bytes: 2

Cycles: 1

Encoding:

1 1 0 1	0 0 1 0	bit address

Operation:

```
SETB
(bit)←—1
```

SJMP rel

Function: Short Jump

Description: Program control branches unconditionally to the address indicated. The branch destination is computed by adding the signed displacement in the second instruction byte to the PC, after incrementing the PC twice. Therefore, the range of destinations allowed is from 128 bytes preceding this instruction to 127 bytes following it.

Example: The label "RELADR" is assigned to an instruction at program memory location 0123H. The instruction,

```
SJMP    RELADR
```

will assemble into location 0100H. After the instruction is executed, the PC will contain the value 0123H.

(Note: Under the above conditions the instruction following SJMP will be at 102H. Therefore, the displacement byte of the instruction will be the relative offset (0123H-0102H) = 21H. Put another way, an SJMP with a displacement of 0FEH would be a one-instruction infinite loop.)

Bytes: 2

Cycles: 2

Encoding:

1 0 0 0	0 0 0 0	rel. address

Operation: SJMP

(PC) ← (PC) + 2

(PC) ← (PC) + rel

SUBB A, <src-byte>

Function: Subtract with borrow

Description: SUBB subtracts the indicated variable and the carry flag together from the accumulator, leaving the result in the accumulator. SUBB sets the carry (borrow) flag if a borrow is needed for bit 7, and clears C otherwise. (If C was set *before* executing a SUBB instruction, this indicates that a borrow was needed for the previous step in a multiple precision subtraction, so the carry is subtracted from the accumulator along with the source operand.) AC is set if a borrow is needed for bit 3, and cleared otherwise. OV is set if a borrow is needed into bit 6, but not into bit 7, or into bit 7, but not bit 6.

When subtracting signed integers OV indicates a negative number produced when a negative value is subtracted from a positive value, or a positive result when a positive number is subtracted from a negative number.

The source operand allows four addressing modes: register, direct, register-indirect, or immediate.

Example: The accumulator holds 0C9H (11001001B), register 2 holds 54H (01010100B), and the carry flag is set. The instruction,

```
SUBB    A,R2
```

will leave the value 74H (01110100B) in the accumulator, with the carry flag and AC cleared but OV set.

Notice that 0C9H minus 54H is 75H. The difference between this and the above result is due to the carry (borrow) flag being set before the operation. If the state of the carry is not known before starting a single or multiple-precision subtraction, it should be explicitly cleared by a CLR C instruction.

SUBB A,Rn

Bytes: 1

Cycles: 1

Encoding:

1	0	0	1	1	r	r	r

Operation: SUBB

(A) ← (A) − (C) − (Rn)

SUBB A,direct

Bytes: 2
Cycles: 1

Encoding:

1 0 0 1	0 1 0 1	direct address

Operation: SUBB
(A)←(A) – (C) – (direct)

SUBB A,@Ri

Bytes: 1
Cycles: 1

Encoding:

1 0 0 1	0 1 1 i

Operation: SUBB
(A)←(A) – (C) – ((Ri))

SUBB A,#data

Bytes: 2
Cycles: 1

Encoding:

1 0 0 1	0 1 0 0	immediate data

Operation: SUBB
(A)←(A) – (C) – #data

SWAP A

Function: Swap nibbles within the Accumulator

Description: SWAP A interchanges the low- and high-order nibbles (four-bit fields) of the accumulator (bits 3-0 and bits 7-4). The operation can also be thought of as a four-bit rotate instruction. No flags are affected.

Example: The accumulator holds the value 0C5H (11000101B). The instruction,

```
SWAP    A
```

leaves the accumulator holding the value 5CH (01011100B).

Bytes: 1
Cycles: 1

Encoding:

1 1 0 0	0 1 0 0

Operation: SWAP
$(A_{3-0}) \rightleftarrows (A_{7-4})$, $(A_{7-4}) \leftarrow (A_{3-0})$

XCH A,<byte>

Function: Exchange Accumulator with byte variable

Description: XCH loads the accumulator with the contents of the indicated variable, at the same time writing the original accumulator contents to the indicated variable. The source/destination operand can use register, direct, or register-indirect addressing.

Example: R0 contains the address 20H. The accumulator holds the value 3FH (00111111B). Internal RAM location 20H holds the value 75H (01110101B). The instruction,

XCH A,@R0

will leave RAM location 20H holding the values 3FH (00111111B) and 75H (01110101B) in the accumulator.

XCH A,Rn

Bytes: 1

Cycles: 1

Encoding:

1	1	0	0	1	r	r	r

Operation: XCH

(A) $\rightleftarrows$ (Rn)

XCH A,direct

Bytes: 2

Cycles: 1

Encoding:

1	1	0	0	0	1	0	1	direct address

Operation: XCH

(A) $\rightleftarrows$ (direct)

XCH A,@Ri

Bytes: 1

Cycles: 1

Encoding:

1	1	0	0	0	1	1	i

Operation: XCH

(A) $\rightleftarrows$ ((Ri))

XCHD A,@Ri

Function: Exchange Digit

Description: XCHD exchanges the low-order nibble of the accumulator (bits 3-0), generally representing a hexadecimal or BCD digit), with that of the internal RAM location indirectly addressed by the specified register. The high-order nibbles (bits 7-4) of each register are not affected. No flags are affected.

Example: R0 contains the address 20H. The accumulator holds the value 36H (00110110B). Internal RAM location 20H holds the value 75H (01110101B). The instruction,

XCHD A,@R0

will leave RAM location 20H holding the value 76H (01110110B) and 35H (00110101B) in the accumulator.

Bytes: 1

Cycles: 1

Encoding:

1	1	0	1	0	1	1	i

Operation: XCHD

$(A_{3-0}) \rightleftarrows ((Ri_{3-0}))$

XRL <dest-byte>, <src-byte>

Function: Logical Exclusive-OR for byte variables

Description: XRL performs the bitwise logical Exclusive-OR operation between the indicated variables, storing the results in the destination. No flags are affected.

The two operands allow six addressing mode combinations. When the destination is the accumulator, the source can use register, direct, register-indirect, or immediate addressing; when the destination is a direct address, the source can be the accumulator or immediate data.

(Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.)

Example: If the accumulator holds 0C3H (11000011B) and register 0 holds 0AAH (10101010B) then the instruction,

```
XRL     A,R0
```

will leave the accumulator holding the value 69H (01101001B).

When the destination is a directly addressed byte, this instruction can complement combinations of bits in any RAM location or hardware register. The pattern of bits to be complemented is then determined by a mask byte, either a constant contained in the instruction or a variable computed in the accumulator at run-time. The instruction,

```
XRL     P1,#00110001B
```

will complement bits 5, 4, and 0 of output port 1.

XRL A,Rn

Bytes: 1

Cycles: 1

Encoding:

0	1	1	0	1	r	r	r

Operation:

```
XRL
(A) ← (A) ∀ (Rn)
```

XRL A,direct

Bytes: 2

Cycles: 1

Encoding:

0	1	1	0	0	1	0	1	direct address

Operation:

```
XRL
(A) ← (A) ∀ (direct)
```

XRL A,@Ri

Bytes: 1

Cycles: 1

Encoding:

0	1	1	0	0	1	1	i

Operation: XRL

(A)←(A) ∀ ((Ṙi))

XRL A,#data

Bytes: 2

Cycles: 1

Encoding:

0	1	1	0	0	1	0	0	immediate data

Operation: XRL

(A)←(A) ∀ #data

XRL direct,A

Bytes: 2

Cycles: 1

Encoding:

0	1	1	0	0	0	1	0	direct address

Operation: XRL

(direct)← (direct) ∀ (A)

XRL direct,#data

Bytes: 3

Cycles: 2

Encoding:

0	1	1	0	0	0	1	1	direct address	immediate data

Operation: XRL

(direct)← (direct) ∀ #data

MCS®-51 Application Examples 9

CHAPTER 9
MCS®-51 APPLICATION EXAMPLES

Chapter 9 is divided into three sections:

- 8051 Programming Techniques
- Peripheral Interfacing Techniques
- Peripheral Interface Examples

The first section has 8051 software examples for common routines in controller applications. Some of the routines are multiple-precision arithmetic and table look-up techniques.

Peripheral Interfacing Techniques include routines for handling the 8051's I/O ports, serial channel and timer/counters. Discussed in this section is I/O port reconfiguration, software delay timing, and transmitting serial port character strings along with other routines.

The Peripheral Interface examples consist of block diagram schematics.

9.0 8051 PROGRAMMING TECHNIQUES

9.0.1 Radix Conversion Routines

The divide instruction can be used to convert a number from one radix to another. BINBCD is a short subroutine to convert an eight-bit unsigned binary integer in the accumulator (between 0 & 255) to a three-digit (two byte) BCD representation. The hundred's digit is returned in one variable (HUND) and the ten's and one's digits returned as packed BCD in another (TENONE).

```
;
; BINBCD  CONVERT 8-BIT BINARY VARIABLE IN ACCUMULATOR
;         TO 3-DIGIT PACKED BCD FORMAT.
;         HUNDREDS' PLACE LEFT IN VARIABLE 'HUND',
;         TENS' AND ONES' PLACES IN 'TENONE'.
;
HUND    DATA  21H
TENONE  DATA  22H
;
BINBCD: MOV   B,#100        ;DIVIDED BY 100 TO
        DIV   AB            ;DETERMINE NUMBER OF HUNDREDS
        MOV   HUND,A
        MOV   A,#10         ;DIVIDE REMAINDER BY TEN TO
        XCH   A,B           ;DETERMINE NUMBER OF TENS LEFT
        DIV   AB            ;TEN'S DIGIT IN ACC, REMAINDER IS
                            ;ONE'S DIGIT
        SWAP  A
        ADD   A,B           ;PACK BCD DIGITS IN ACC
        MOV   TENONE,A
        RET
;
```

The divide instruction can also separate data in the accumulator into sub-fields. For example, dividing packed BCD data by 16 will separate the two nibbles, leaving the high-order digit in the accumulator and the low-order digit (remainder) in B. Each is right-justified, so the digits can be processed individually. This example receives two packed BCD digits in the accumulator, separates the digits, computes their product, and returns the product in packed BCD format in the accumulator.

```
;
;MULBCD  UNPACK TWO BCD DIGITS RECEIVED IN ACCUMULATOR,
;        FIND THEIR PRODUCT, AND RETURN PRODUCT
;        IN PACKED BCD FORMAT IN ACCUMULATOR
;
MULBCD: MOV   B,#10H        ;DIVIDE INPUT BY 16
        DIV   AB            ;A & B HOLD SEPARATED DIGITS
                            ;(EACH RIGHT JUSTIFIED IN REGISTER).
        MUL   AB            ;A HOLDS PRODUCT IN BINARY FORMAT (0-
                            ;99 (DECIMAL) = 0 - 63H)
        MOV   B,#10         ;DIVIDE PRODUCT BY 10
        DIV   AB            ;A HOLDS NUMBER OF TENS, B HOLDS
                            ;REMAINDER
        SWAP  A
        ORL   A,B           ;PACK DIGITS
        RET
```

9.0.2 Multiple Precision Arithmetic

The ADDC and SUBB instructions incorporate the previous state of the carry (borrow) flag to allow multiple-precision calculations by repeating the operation with successively higher-order operand bytes. If the input data for a multiple-precision operation is an unsigned string of integers, the carry flag will be set upon completion if an overflow (for ADDC) or underflow (for SUBB) occurs. With two's complement signed data, the most significant bit of the original input data's most significant byte indicates the sign of the string, so the overflow flag (OV) will indicate if overflow or underflow occurred.

```
;
;SUBSTR  SUBTRACT STRING INDICATED BY R1
;        FROM STRING INDICATED BY R0 TO
;        PRECISION INDICATED BY R2.
;        CHECK FOR SIGNED UNDERFLOW WHEN DONE.
;
SUBSTR: CLR   C             ;BORROW = 0.
```

```
SUBS1:  MOV   A,@R0          ;LOAD MINUEND BYTE
        SUBB  A,@R1          ;SUBTRACT SUBTRAHEND BYTE
        MOV   @R0,A          ;STORE DIFFERENCE BYTE
        INC   R0             ;BUMP POINTERS TO NEXT PLACE
        INC   R1
        DJNZ  R2,SUBS1       ;LOOP UNTIL DONE
;
;       WHEN DONE, TEST IF OVERFLOW OCCURRED
;       ON LAST ITERATION OF LOOP.
;
        JNB   OV,OV_OK
;       ...   .......        (OVERFLOW RECOVERY ROUTINE)
        OV_OK: RET           ;RETURN
```

9.0.3 Table Look-Up Sequences

The two versions of the MOVC instructions are used as part of a three-step sequence to access look-up tables in ROM. To use the DPTR version, load the Data Pointer with the starting address of a look-up table; load the accumulator with (or compute) the index of the entry desired; and execute MOVC A,@A+DPTR. The data pointer may be loaded with a constant for short tables, or to allow more complicated data structures, and tables with more than 256 entries, the values for DPH and DPL may be computed or modified with the standard arithmetic instruction set.

The PC-based version is used with smaller, "local" tables, and has the advantage of not affecting the data pointer. This makes it useful in interrupt routines or other situations where the DPTR contents might be significant. Again, a look-up sequence takes three steps: load the accumulator with the index; compensate for the offset from the look-up instruction's address to the start of the table by adding that offset to the accumulator; then execute the MOVC A,@A+PC instruction.

As a non-trivial situation where this instruction would be used, consider applications which store large multi-dimensional look-up tables of dot matrix patterns, non-linear calibration parameters, and so on in the linear (one-dimensional) program memory. To retrieve data from the tables, variables representing matrix indices must be converted to the desired entry's memory address. For a matrix of dimensions (MDIMEN x NDIMEN) starting at address BASE and respective indices INDEXI and INDEXJ, the address of element (INDEXI, INDEXJ) is determined by the formula,

$$\text{Entry Address} = [\text{BASE} + (\text{NDIMEN} \times \text{INDEXI}) + \text{INDEXJ}]$$

The subroutine MATRIX1 can access an entry in any array with less than 255 elements (e.g., an 11 x 21 array with 231 elements). The table entries are defined using the Data Byte ("DB") directive, and will be contained in the assembly object code as part of the accessing subroutine itself.

To handle the more general case, subroutine MATRX2 allows tables to be unlimited in size, by combining the MUL instruction, double-precision addition, and the data pointer-based version of MOVC. The only restriction is that each index be between 0 and 255.

```
;
;MATRX1  LOAD CONSTANT READ FROM TWO DIMENSIONAL LOOK-UP
;        TABLE IN PROGRAM MEMORY INTO ACCUMULATOR
;        USING LOCAL TABLE LOOK-UP INSTRUCTION, 'MOVC A,@A+PC'.
;        THE TOTAL NUMBER OF TABLE ENTRIES IS ASSUMED TO
;        BE SMALL, I.E., LESS THAN ABOUT 255 ENTRIES.
;        TABLE USED IN THIS EXAMPLE IS 11 x 21.
;        DESIRED ENTRY ADDRESS IS GIVEN BY THE FORMULA,
;
;        [(BASE ADDRESS) = (21 X INDEXI) = (INDEXJ)]
;
INDEXI  EQU   R6                ;FIRST COORDINATE OF ENTRY (0-10).
INDEXJ  DATA  23H               ;SECOND COORDINATE OF ENTRY (0-20).
;
MATRX1: MOV   A,INDEXI
        MOV   B, #21
        MUL   AB                ;(21 X INDEXI)
        ADD   A,INDEXJ          ; ADD IN OFFSET WITHIN ROW
;
;       ALLOW FOR INSTRUCTION BYTE BETWEEN "MOVC" AND
;       ENTRY (0,0).
;
        INC   A
        MOVC  A,@A+PC
        RET
BASE1:  DB    1                 ;(entry 0,0)
        DB    2                 ;(entry 0,1)
;       ..    ......
        DB    21                ;(entry 0,20)
        DB    22                ;(entry 1,0)
;       ..    ......
        DB    42                ;(entry 1,20)
;       ..    ......
;       ..    ......
;       DB    231               ;(entry 10,20)
```

```
MATRX2:  MOV   A,INDEXI          ;LOAD FIRST COORDINATE
         MOV   B,#NDIMEN
         MUL   AB                :INDEXI X NDIMEN
         ADD   A,#LOW(BASE2)     ;ADD IN 16-BIT BASE ADDRESS
         MOV   DPL,A
         MOV   A,B
         ADDC  A,#HIGH(BASE2)
         MOV   DPH,A             ;DPTR=(BASE ADDR) + (INDEXI + NDIMEN)
         MOV   A,INDEXJ
         MOVC  A,@A+DPTR         ;ADD INDEXJ AND FETCH BYTE
         RET
         ...   .....
BASE2:   DB    0                 ;(entry 0,0)
         DB    0                 ;(entry 0,1)
;        ...   ......
         DB    0                 ;(entry 0, NDIMEN-1)
         DB    0                 ;(entry 1,0)
;        ...   ......
         DB    0                 ;(entry 1, NDIMEN-1)
;        ...   ......
;        ...   ......
         DB    0                 ;(entry MDIMEN-1, NDIMEN-1)
```

9.0.4 Saving CPU Status during Interrupts

When the 8051 hardware recognizes an interrupt request, program control branches automatically to the corresponding service routine, by forcing the CPU to process a Long CALL (LCALL) instruction to the appropriate address. The return address is stored on the top of the stack. After completing the service routine, an RETI instruction returns the processor to the background program at the point from which it was interrupted.

Interrupt service routines must not change any variable or hardware registers modified by the main program, or else the program may not resume correctly. (Such a change might look like a spontaneous random error. An example of this will be given later in this section, in the second method of I/O port reconfiguration.) Resources used or altered by the service routine (Accumulator, PSW, etc.) must be saved and restored to their previous value before returning from the service routine. PUSH and POP provide an efficient and convenient way to save such registers on the stack.

```
;
LOC_TMP  EQU    $                ;REMEMBER LOCATION COUNTER
;
         ORG    0003H            ;STARTING ADDRESS FOR INTERRUPT ROUTINE
         LJMP   SERVER           ;JUMP TO ACTUAL SERVICE ROUTINE LOCATE
                                 ;ELSEWHERE
;        ...    ......
         ORG    LOC_TMP          ;RESTORE LOCATION COUNTER
SERVER:  PUSH   PSW
         PUSH   ACC              ;SAVE ACCUMULATOR (NOTE DIRECT ADDRESS
                                 ;NOTATION)
         PUSH   B                ;SAVE B REGISTER
         PUSH   DPL              :SAVE DATA POINTER
         PUSH   DPH              ;
         MOV    PSW,#00001000B   ;SELECT REGISTER BANK 1
;        ...    ......
;        ...    ......
         POP    DPH              ;RESTORE REGISTERS IN REVERSE ORDER
         POP    DPL
         POP    B
         POP    ACC
         POP    PSW              ;RESTORE PSW AND RE-SELECT ORIGINAL
                                 ;REGISTER BANK
         RETI                    ;RETURN TO MAIN PROGRAM AND RESTORE
                                 ;INTERRUPT LOGIC
```

If the SP register held 1FH when the interrupt was detected, then while the service routine was in progress the stack would hold the registers shown in Figure 9-1; SP would contain 26H. This is the most general case; if the service routine doesn't alter the B-register and data pointer, for example, the instructions saving and restoring those registers could be omitted.

9.0.5 Passing Parameters on the Stack

The stack may also pass parameters to and from subroutines. The subroutine can indirectly address the parameters derived from the contents of the stack pointer, or simply pop the stack into registers before processing.

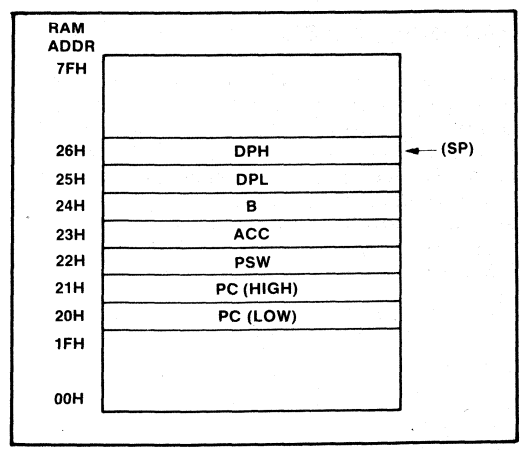


Figure 9-1.

```
HEXASC: MOV   R0,SP
        DEC   R0            ;ACCESS LOCATION PARAMETER PUSHED
                             INTO
        DEC   R0
        XCH   A,@R0         ;READ INPUT PARAMETER AND SAVE AC-
                             CUMULATOR
        ANL   A,#0FH        ;MASK ALL BUT LOW-ORDER 4 BITS
        ADD   A,#2          ;ALLOW FOR OFFSET FROM MOVC TO TABLE
        MOVC  A,@A+PC       ;READ LOOK-UP TABLE ENTRY
        XCH   A,@R0         ;PASS BACK TRANSLATED VALUE AND
                             ;RESTORE ACCUMULATOR
        RET                 ;RETURN TO BACKGROUND PROGRAM
ASCTBL: DB    '0'           ;ASCII CODE FOR 00H
        DB    '1'           ;ASCII CODE FOR 01H
        DB    '2'           ;ASCII CODE FOR 02H
        DB    '3'           ;ASCII CODE FOR 03H
        DB    '4'           ;ASCII CODE FOR 04H
        DB    '5'           ;ASCII CODE FOR 05H
        DB    '6'           ;ASCII CODE FOR 06H
        DB    '7'           ;ASCII CODE FOR 07H
        DB    '8'           ;ASCII CODE FOR 08H
        DB    '9'           ;ASCII CODE FOR 09H
        DB    'A'           ;ASCII CODE FOR 0AH
        DB    'B'           ;ASCII CODE FOR 0BH
        DB    'C'           ;ASCII CODE FOR 0CH
        DB    'D'           ;ASCII CODE FOR 0DH
        DB    'E'           ;ASCII CODE FOR 0EH
        DB    'F'           ;ASCII CODE FOR 0FH
```

One advantage here is simplicity. Variables need not be allocated for specific parameters, a potentially large number of parameters may be passed, and different calling programs may use different techniques for determining or handling the variables.

For example, the subroutine HEXASC converts a hexadecimal value to ASCII code for its low-order digit. It first reads a parameter stored on the stack by calling program, then uses the low-order bits to access a local 16-entry look-up table holding ASCII codes, stores the appropriate code back in the stack and then returns. The accumulator contents are left unchanged.

The background program may reach this subroutine with several different calling sequences, all of which PUSH a value before calling the routine and POP the result to any destination register or port later. There is even the option of leaving a value on the stack if it won't be needed until later. The example below converts the three-digit BCD value computed in the Radix Conversion example above to a three-character string, calling a subroutine SP_OUT to output an eight-bit code in the accumulator.

```
...     ......
PUSH    HUND
CALL    HEXASC          ;CONVERT HUNDREDS DIGIT
POP     ACC
CALL    SP_OUT          ;TRANSMIT HUNDREDS CHARACTER
PUSH    TENONE
CALL    HEXASC          ;CONVERT ONE'S PLACE DIGIT
                        ;BUT LEAVE ON STACK!
MOV     A, TENONE
SWAP    A               ;RIGHT-JUSTIFY TEN'S PLACE
PUSH    ACC             ;CONVERT TEN'S PLACE DIGIT
CALL    HEXASC
POP     ACC
CALL    SP_OUT          ;TRANSMIT TEN'S PLACE CHARACTER
POP     ACC
CALL    SP_OUT          ;TRANSMIT ONE'S PLACE CHARACTER
...     ......
```

9.0.6 N-Way Branching

There are several different means for branching to sections of code determined or selected at run time. (The single destination addresses incorporated into conditional and unconditional jumps are, of course, fixed at assembly time.) Each has advantages for different applications.

In a typical N-way branch situation, the potential destinations are generally known at assembly time. One of a number of small routines is selected according to the value of an index variable determined while the program is running. The most efficient way to solve this problem is with the MOVC and an indirect jump instruction, using a short table of offset values in ROM to indicate the relative starting addresses of the several routines.

JMP @A+DPTR is an instruction which performs an indirect jump to an address determined during program execution. The instruction adds the eight-bit unsigned accumulator contents with the contents of the sixteen-bit data pointer, just like MOVC A,@A+DPTR. The resulting sum is loaded into the program counter and is used as the address for subsequent instruction fetches. Again, a sixteen-bit addition is performed: a carry-out from the low-order eight-bits may propagate through the higher-order bits. In this case, neither the accumulator contents nor the data pointer is altered.

The example subroutine below reads a byte of RAM into the accumulator from one of four alternate address spaces, as selected by the contents of the variable MEMSEL. The address of the byte to be read is determined by the contents of R0 (and optionally R1). It might find use in a printing terminal application, where four different model printers all use the same ROM code but use different types (and sizes) of buffer memory for different speeds and options.

```
;
MEMSEL  EQU     R3
;
JUMP_4: MOV     A,MEMSEL
        MOV     DPTR,#JMPTBL
        MOVC    A,@A+DPTR
        JMP     @A+DPTR
JMPTBL: DB      MEMSP0-JMPTBL
        DB      MEMSP1-JMPTBL
        DB      MEMSP2-JMPTBL
        DB      MEMSP3-JMPTBL
MEMSP0: MOV     A,@R0           ;READ FROM INTERNAL RAM
        RET
MEMSP1: MOVX    A,@R0           ;READ 256 BYTE EXTERNAL RAM
        RET
MEMSP2: MOV     DPL,R0          ;READ 64K BYTE EXTERNAL RAM
        MOV     DPH,R1
        MOVX    A,@DPTR
        RET
MEMSP3: MOV     A,R1            ;READ 4K BYTE EXTERNAL RAM
        ANL     A,#07H
        ANL     P1,#11111000B
        ORL     P1,A
        MOVX    A,@R0
        RET
```

To use this approach, the size of the jump table plus the length of the alternate routines must be less than 256 bytes. The jump table and routines may be located anywhere in program memory and are independent of 256-byte program memory pages.

For applications where up to 128 destinations must be selected, all residing in the same 2K page of program memory, the following technique may be used. In the printing terminal example, this sequence could process 128 different codes for ASCII characters arriving via the 8051 serial port.

```
;
OPTION   EQU   R3
;        ...   ......
;        ...   ......
JMP128:  MOV   A,OPTION
         RL    A                ;MULTIPLY BY 2 FOR 2-BYTE JUMP TABLE
         MOV   DPTR,#INSTBL     ;FIRST ENTRY IN JUMP TABLE
         JMP   @A+DPTR          ;JUMP INTO JUMP TABLE
;        ...   ......           ;
INSTBL:  AJMP  PROC00           ;128 CONSECUTIVE
         AJMP  PROC01           ;AJMP INSTRUCTIONS
         AJMP  PROC02
;        ...   ......
;        ...   ......
         AJMP  PROC7E
         AJMP  PROC7F
```

The destinations in the jump table (PROC00-PROC7F) are not all necessarily unique routines. A large number of special control codes could each be processed with their own unique routine, with the remaining printing characters all causing a branch to a common routine for entering the character into the output queue.

9.0.7 Computing Branch Destinations at Run Time

In some rare situations, 128 options are insufficient, the destination routines may cross a 2K page boundary, or a branch destination is not known at assembly time (for whatever reason), and therefore cannot be easily included in the assembled code. These situations can all be handled by computing the destination address at run-time with standard arithmetic or table look-up instructions, then performing an indirect branch to that address. There are two simple ways to execute this last step, assuming the 16-bit destination address has already been computed. The first is to load the address into the DPH and DPL registers, clear the accumulator and branch using the JMP @A+DPTR instruction; the second is to push the destination address onto the stack, low-order byte first (so as to mimic a call instruction) then pop that address into the PC by performing a return instruction. This also adjusts the stack pointer to its previous value. The code segment below illustrates the latter possibility.

```
;
RTEMP    EQU   R7
;        ...   ......
JMP256:  MOV   DPTR,#ADRTBL    ;FIRST ADDRESS TABLE ENTRY
         MOV   A,OPTION        ;LOAD INDEX INTO TABLE
         CLR   C
         RLC   A               ;MULTIPLY BY 2 FOR 2-BYTE JUMP TABLE
         JNC   LOW128
         INC   DPH             ;FIX BASE IF INDEX>127
```

```
LOW128:  MOV    RTEMP,A          ;SAVE ADJUSTED ACC FOR SECOND READ
         INC    A                ;READ LOWORDER BYTE FIRST
         MOVC   A,@A+DPTR        ;GET LOW-ORDER BYTE FROM TABLE
         PUSH   ACC
         MOV    A,RTEMP          ;RELOAD ADJUSTED ACC
         MOVC   A,@A+DPTR        ;GET HIGH-ORDERED BYTE FROM TABLE
         PUSH   ACC
;
;        THE TWO ACC PUSHES HAVE PRODUCED
;        A "RETURN ADDRESS" ON THE STACK WHICH CORRESPONDS
;        TO THE DESIRED STARTING ADDRESS.
;        IT MAY BE REACHED BY POPPING THE STACK
;        INTO THE PC.
         RET
;        ...     ......
;        ...     ......
;        ...     ......
ADRTBL:  DW     PROC00           ;UP TO 256 CONSECUTIVE DATA
         DW     PROC01           ;WORDS INDICATING STARTING ADDRESSES
;        ...     ......
;        ...     ......
         DW     PROCFF
;
```

9.0.8 In-Line-Code Parameter Passing

Parameters can be passed by loading appropriate registers with values before calling the subroutine. This technique is inefficient if a lot of the parameters are constants, since each would require a separate register to carry it, and a separate instruction to load the register each time the routine is called.

If the routine is called frequently, a more code-efficient way to transfer constants is "in-line-code" parameter-passing. The constants are actually part of the program code, immediately following the call instruction. The subroutine determines where to find them from the return address on the stack, and then reads the parameters it needs from program memory.

For example, assume a utility named ADDBCD adds a 16-bit packed-BCD constant with a two-byte BCD variable in internal RAM and stores the sum in a different two-byte buffer. The utility must be given the constant and both buffer addresses. Rather than using four working registers to carry this information, all four bytes could be inserted into program memory each time the utility is called. Specifically, the calling sequence below invokes the utility to add 1234 (decimal) with the string at internal RAM address 56H, and store the sum in a buffer at location 78H.

The ADDBCD subroutine determines at what point the call was made by popping the return address from the stack into the data pointer high- and low-order bytes. A MOVC instruction then reads the parameters from program memory as they are needed. When done, ADDBCD resumes execution by jumping to the instruction following the last parameter.

```
        ...     ......
        CALL    ADDBCD
        DW      1234H           ;BCD CONSTANT
        DB      56H             ;SOURCE STRING ADDRESS
        DB      78H             ;DESTINATION STRING ADDRESS
;       ...     ......          ;CONTINUATION OF PROGRAM
;
;
;
ADDBCD: POP     DPH             ;POP RETURN ADDRESS INTO DPTR
        POP     DPL
        MOV     A,#2            ;INDEX FOR SOURCE STRING PARAMETER
        MOVC    A,@A+DPTR       ;GET SOURCE STRING LOCATION
        MOV     R0,A
        MOV     A,#3            ;INDEX FOR DESTINATION STRING
                                ;PARAMETER
        MOVC    A,@A+DPTR       ;GET DESTINATION ADDRESS
        MOV     R1,A
        MOV     A,#1            ;INDEX FOR 16-BIT CONSTANT LOW BYTE
        MOVC    A,@A+DPTR       ;GET LOW-ORDER VALUE
        ADD     A,@R0           ;COMPUTE LOW-ORDER BYTE OF SUM
        DA      A               ;DECIMAL ADJUST FOR ADDITION
        MOV     @R1,A           ;SAVE IN BUFFER
        INC     R0
        INC     R1
        CLR     A               ;INDEX FOR HIGH-BYTE = 0
        MOVC    A,@A+DPTR       ;GET HIGH-ORDER CONSTANT
        ADDC    A,@R0
        DA      A               ;DECIMAL ADJUST FOR ADDITION
        MOV     @R1,A           ;SAVE IN BUFFER
        MOV     A,#4            ;INDEX FOR CONTINUATION OF PROGRAM
        JMP     @A+DPTR         ;JUMP BACK INTO MAIN PROGRAM
```

This example illustrates several points:

1) The "subroutine" does not end with a normal return statement; instead, an indirect jump relative to the data pointer returns execution to the first instruction following the parameter list. The two initial POP instructions correct the stack pointer contents.

2) Either an ACALL or LCALL works with the subroutine, since each pushes the address of the *next* instruction or data byte onto the stack. The call may be made from anywhere in the full 8051 address space, since the MOVC instruction access all 64K bytes.

3) The parameters passed to the utility can be listed in whatever order is most convenient, which may not be that in which they're used. The utility has essentially "random access" to the paramater list, by loading the appropriate constant into the accumulator before each MOVC instruction.

4) Other than the data pointer, the whole calling and processing sequence only affects the accumulator, PSW and pointer registers. The utility could have pushed these registers onto the stack (after popping the parameter list starting address), and popped before returning.

Passing parameters through in-line-code can be used in conjunction with other variable passing techniques.

The utility can also get input variables from working registers or from the stack, and return output variables to registers or to the stack.

9.1 PERIPHERAL INTERFACING TECHNIQUES

9.1.1 I/O Port Reconfiguration (First Approach)

I/O ports must often transmit or receive parallel data in formats other than as eight-bit bytes. For example, if an application requires three five-bit latched output ports (called X, Y, and Z), these "virtual" ports could be mapped onto the pins of "physical" ports 1 and 2 (see example at bottom of page).

This pin assignment leaves P2.7 free for use as a test pin, input data pin, or control output through software.

Notice that the bits of port Z are reversed. The highest-order port Z pin corresponds to pin P2.2, and the lowest-order pin of port Z is P2.6, due to P.C. board layout considerations. When connecting an 8051 to an immediately adjacent keyboard column decoder or another device with weighted inputs, the corresponding pins may not be aligned. The interconnections must be "scrambled" to compensate either with interwoven circuit board traces or through software (as shown on the next page).

	PORT "Z"					PORT "Y"					PORT "X"				
	PZ0	PZ1	PZ2	PZ3	PZ4	PY4	PY3	PY2	PY1	PY0	PX4	PX3	PX2	PX1	PX0
P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0

```
PX_MAP  DATA   20H
PY_MAP  DATA   21H
PZ_MAP  DATA   22H
;       ...    ....
OUT_PX: ANL    A,#00011111B      ;CLEAR BITS ACC.7 - ACC. 5
        MOV    PX_MAP,A          ;SAVE DATA IN MAP BYTE
        ACALL  OUT_P1            ;UPDATE PORT 1 OUTPUT LATCH
        RET
;       ....   ....
OUT_PY: MOV    PY_MAP,A          ;SAVE IN MAP BYTE
        ACALL  OUT_P1            ;UPDATE PORT 1
        ACALL  OUT_P2            ;AND PORT 2 OUTPUT LATCHES
        RET
;       ....   ....
OUT_PZ: MOV    PZ_MAP,A          ;SAVE DATA IN MAP BYTE
        ACALL  OUT_P2            ;UPDATE PORT 2.
        RET
;       ....   ....
;       ....   ....
OUT_P1: MOV    A,PY_MAP          ;OUTPUT ALL P1 BITS
        SWAP   A
        RL     A                 ;SHIFT PY_MAP LEFT 5 BITS
        ANL    A,#11100000B      ;MASK OUT GARBAGE
        ORL    A,PX_MAP          ;INCLUDE PX_MAP BITS
        MOV    P1,A
        RET
;       ....   ....
OUT_P2: MOV    C,PZ_MAP.0        ;LOAD CY WITH P2.6 BIT
        RLC    A                 ;AND SHIFT INTO ACC.
        MOV    C,PZ_MAP.1        ;LOAD CY WITH P2.5 BIT
        RLC    A                 ;AND SHIFT INTO ACC.
        MOV    C,PZ_MAP.2        ;LOAD CY WITH P2.4 BIT
        RLC    A                 ;AND SHIFT INTO ACC.
        MOV    C,PZ_MAP.3        ;LOAD CY WITH P2.3 BIT
        RLC    A                 ;AND SHIFT INTO ACC.
        MOV    C,PZ_MAP.4        ;LOAD CY WITH P2.2 BIT
        RLC    A                 ;AND SHIFT INTO ACC.
        MOV    C,PZ_MAP.4        ;LOAD CY WITH P2.1 BIT
        RLC    A                 ;AND SHIFT INTO ACC.
        MOV    C,PZ_MAP.3        ;LOAD CY WITH P2.0 BIT
        RLC    A                 ;AND SHIFT INTO ACC.
        SETB   ACC.7             ;(ASSUMING INPUT ON P2.7)
        MOV    P2.A
        RET
```

Writing to the virtual ports must not affect any other pins. Since the virtual output algorithms are non-trivial, a subroutine is needed for each port: OUT_PX, OUT_PY and OUT_PZ. Each is called with data to output right-justified in the accumulator, and any data in bits ACC.7-ACC.5 is insignificant. Each subroutine saves the data in a "map" variable for the virtual port, then calls other subroutines which use the data in the various map bytes to compute and output the eight-bit pattern needed for each physical port affected. The two level structure of the above subroutines can be modified somewhat if code efficiency and execution speed are critical: incorporate the code shown as subroutines OUT_P1 and OUT_P2 directly into the code for OUT_PX and OUT_PZ, in place of the corresponding ACALL instructions. OUT_PY would not be changed, but now the destinations for its ACALL instructions would be alternate entry points in OUT_PX and OUT_PZ, instead of isolated subroutines.

9.1.2 I/O Port Configuration (Second Approach)

A trickier situation arises if two sections of code which write to the same port or register, or call virtual output routines like those above, need to be executed at different interrupt levels. For example, suppose the background program wants to rewrite Port X (using the port associations in the previous example), and has computed the bit pattern needed for P1. An interrupt is detected just before the MOV P1,A instruction, and the service routine tries to write to Port Y. The service routine would correctly update P1 and P2, but upon returning to the background program P1 is immediately *re*-written with the data computed *before* the interrupt! Now pins P2.1 and P2.0 indicate (correctly) data written to port Y in the interrupt routine, but the earlier data written to P1.7-P1.5 is no longer valid. The same sort of confusion could arise if a high-level interrupt disrupted such an output sequence.

One solution is to disable interrupts around any section of code which must not be interrupted (called "critical section") but this would adversely affect interrupt latency. Another is to have interrupt routines set or clear a flag ("semaphore") when a common resource is altered — a rather complex and elaborate system.

An easier way to ensure that any instruction which writes the port X field of P1 does not change the port Y field pins from their state *at the beginning of that instruction,* is shown next. A number of 8051 operations read, modify, and write the output port latches all in one instruction. These are the arithmetic and logical instructions (INC, DEC, ANL, ORL, etc.), where an addressed byte is both the destination variable and one of the source operands. Using these instructions, instead of data moves, eliminates the critical section problem entirely.

```
OUT_PX: ANL   P1,#11100000B     ;CLEAR BITS P1.4 - P1.0
        ORL   P1,A              ;SET P1 PIN FOR EACH ACC BIT SET
        RET
;       ...   ......
;       ...   ......
OUT_PY: MOV   B#20H
        MUL   AB                ;SHIFT B A LEFT 5 BITS.
        ANL   P1,#00011111B     ;CLEAR PY FIELD OF PORT 1
        ORL   P1,A              ;SET PY PITS ON PORT 1
        MOV   A,B               ;LOAD 2 BITS SHIFTED INTO B
        ANL   P2,#11111100B     ;AND UPDATE P2
        ORL   P2,A
        RET
;       ...   ......
;       ...   ......
OUT_PZ: RRC   A                 ;MOVE ORIGINAL ACC.0 INTO CY
        MOV   P2.6,C            ;AND STORE TO PIN P2.6.
        RRC   A                 ;MOVE ORIGINAL ACC.1 INTO CY
        MOV   P2.5,C            ;AND STORE TO PIN P2.5.
        RRC   A                 ;MOVE ORIGINAL ACC.2 INTO CY
        MOV   P2.4,C            ;AND STORE TO PIN P2.4.
        RRC   A                 ;MOVE ORIGINAL ACC.3 INTO CY
        MOV   P2.3,C            ;AND STORE TO PIN P2.3.
        RRC   A                 ;MOVE ORIGINAL ACC.4 INTO CY
        MOV   P2.2,C            ;AND STORE TO PIN P2.2.
        RET
```

9.1.3 8243 Interfacing

The 8051's quasi-bidirectional port structure lets each I/O pin input data, output data, or serve as a test pin or output strobe under software control. An example of these modes operating in conjunction is the host-processor interface expected by an 8243 I/O expander. Even though the 8051 does not include 8048-type instructions for interfacing with an 8243, the parts can be interconnected and the protocol may be emulated with simple software; see Figure 9-2.

```
;
;IN8243   INPUT DATA FROM AN 8243 I/O EXPANDER
;         CONNECTED TO P23-P20.
;         P25 & P24 MIMIC CS & PROG.
;         P27-P26 USED AS INPUTS. CODE FOR
;         PORT TO BE READ IN ACC.1-ACC.0
;
```

```
PROG    BIT     P2.4                ;SYMBOLIC PIN DESCRIPTION
;
IN8243: ORL     A,#11010000B        ;SET PROG AND PINS USED AS INPUT
        MOV     P2,A                ;OUTPUT PORT CODE AND OPERATION CODE
        CLR     PROG                ;LOWER PROG TO LATCH ADDRESS
        ORL     P2,#00001111B       ;SET LOW ORDER PINS FOR INPUT
        MOV     A,P2                ;READ IN PORT DATA
        ORL     P2,#00110000B       ;SET PROG AND CS HIGH
```

9.1.4 Software Delay Timing

Many 8051 applications involve exact control over output timing. A software-generated output strobe, for instance, might have to be *exactly* 50μsec. wide. The DJNZ operation can insert an one instruction software delay into a piece of code, adding a moderate delay of two instruction cycles per iteration. For example, two instructions can add a 49-μsec software delay loop to code to generate a pulse on the WR pin.

```
CLR     WR
MOV     R2,#24
DJNZ    R2,$
SETB    WR
```

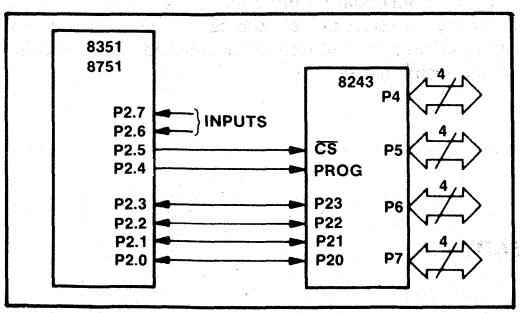


Figure 9-2. Connecting an 8051 with an 8243 I/O Expander

The dollar sign in this example is a special character meaning "the address of this instruction." It can be used to eliminate instruction labels on nearby source lines.

9.1.5 Serial Port and Timer Configuration

Configuring the 8051's Serial Port for a given data rate and protocol requires essentially three short sections of software. On power-up or hardware reset the serial port and timer control words must be initialized to the appropriate values. Additional software is also needed in the transmit routine to load the serial port data register and in the receive routine to unload the data as it arrives.

To choose one arbitrary example, assume the 8051 should communicate with a standard CRT operating at 2400 baud (bits per second). Each character is transmitted as seven data bits, odd parity, and one stop bit. The resulting character rate is 2400 baud/9 bits, approximately 265 characters per second.

For the sake of clarity, the transmit and receive subroutines here are driven by simple-minded software status polling code rather than interrupts. The serial port must be initialized to 8-bit UART mode (SM0, SM1 = 01), enabled to receive all messages (SM2 = 0, REN = 1). The flag indicating that the transmit register is free for more data will be artificially set in order to let the output software know the output register is available. All this can be set up with instruction at label SPINIT.

Timer 1 will be used in auto-reload mode as a baud rate generator. To achieve a data rate of 2400 baud, the timer must divide the 1MHz internal clock by

$$\frac{1 \times 10^6}{(32)\ (2400)}$$

which equals 13 (actually, 13.02) instruction cycles. The timer must reload the value −13, or 0F3H, as shown by the code at label TIINIT. (ASM51 will accept both the signed decimal or hexadecimal representations.)

```
;
;           INITIALIZE SERIAL PORT
;           FOR 8-BIT UART MODE
;           & SET TRANSMIT READY FLAG.
SPINIT:     MOV     SCON,#01010010B
;           INITIALIZE TIMER 1 FOR
;           AUTO-RELOAD AT 32 X 2400HZ
;           (T0 USED AS GATED 16-BIT COUNTER.)
;TIINIT:    MOV     TCON,#1101001B
            MOV     TH1,#13
            SETB    TR1
;           ...     ......
;
```

9.1.6 Simple Serial I/O Drivers

SP_OUT is a simple subroutine to transmit the character passed to it in the accumulator. First it must compute the parity bit, insert it into the data byte, wait until the transmitter is available, output the character and then return.

SP_IN is an equally simple routine which waits until a character is received, sets the carry flag if there is an odd-parity error, and returns the masked seven-bit code in the accumulator.

```
;
;SP_OUT  ADD ODD PARITY TO ACC AND
;           TRANSMIT WHEN SERIAL PORT READY
;
SP_OUT:     MOV     C,P
            CPL     C
            MOV     ACC.7,C
            JNB     TI,$
            CLR     TI
            MOV     SBUF,A
            RET
;
;
```

```
;SP_IN    INPUT NEXT CHARACTER FROM SERIAL PORT.
:         SET CARRY IF ODD-PARITY ERROR
;
SP_IN:    JNB     RI,$
          CLR     RI
          MOV     A,SBUF
          MOV     C,P
          CPL     C
          ANL     A,#7FH
          RET
```

9.1.7 Transmitting Serial Port Character Strings

Any application which transmits characters through a serial port to an ASCII output device will on occasion need to output "canned" messages, including error messages, diagnostics, or operator instructions. These character strings are most easily defined with in-line data bytes defined with the DB directive.

```
CR        EQU     0DH                 ;ASCII CARRIAGE RET
LF        EQU     0AH                 ;ASCII  LINE-FEED
ESC       EQU     1BH                 ;ASCII ESCAPE CODE
;         ...     ......
          CALL    XSTRING
          DB      CR,LF               ;NEW LINE
          DB      'INTEL  DELIVERS'   ;MESSAGE
          DB      ESC                 ;ESCAPE CHARACTER
;
;         (CONTINUATION OF PROGRAM)
;
;         ...     ......
XSTRING:  POP     DPH                 ;LOAD DPTR WITH FIRST CHARACTER
          POP     DPL
XSTR_1:   CLR     A                   ;(ZERO OFFSET)
          MOVC    A,@A+DPTR           ;FETCH FIRST CHARACTER OF STRING
XSTR_2:   JNB     TI,$                ;WAIT UNTIL TRANSMITTER READY
          CLR     TI                  ;MARK AS NOT READY
          MOV     SBUF,A              ;OUTPUT NEXT CHARACTER
          INC     DPTR                ;BUMP POINTER
          CLR     A
          MOVC    A,@A+DPTR           ;GET NEXT OUTPUT CHARACTER
          CJNE    A,#ESC,XSTR_2       ;LOOP UNTIL ESCAPE READ
          MOV     A,#1
          JMP     @A+DPTR             ;RETURN TO CODE AFTER ESCAPE
```

9.1.8 Recognizing and Processing Special Cases

Before operating on the data it receives, a subroutine might give "special handling" to certain input values. Consider a word processing device which receives ASCII characters through the 8051 serial port and drives a thermal hard-copy printer. A standard routine translates most printing characters to bit patterns, but certain control characters ([DEL], [CR], [LF], [BEL], [ESC], or [SP]) must invoke corresponding special routines. Any other character with an ASCII code less than 20H should be translated into the [NUL] value, 00H, and processed with the printing characters. The CJNE operation provides essentially a one-instruction CASE statement.

```
;
CHAR     EQU   R7                  ;CHARACTER CODE VARIABLE
;
INTERP:  CJNE  CHAR,#7FH, INTP_1   ;SKIP UNLESS RUBOUT
;        ...   . . . . .           (SPECIAL ROUTINE FOR RUBOUT CODE)
         RET
INTP_1:  CJNE  CHAR,#07H,INTP_2    ;SKIP UNLESS BELL
;        ...   . . . . .           (SPECIAL ROUTINE FOR BELL CODE)
         RET
INTP_2:  CJNE  CHAR,#0AH,INTP_3    ;SKIP UNLESS LFEED
;        ...   . . . . .           (SPECIAL ROUTINE FOR LFEED CODE)
         RET
INTP_3:  CJNE  CHAR,#0DH,INTP_4    ;SKIP UNLESS RETURN
;        ...   . . . . .           (SPECIAL ROUTINE FOR RETURN CODE)
         RET
INTP_4:  CJNE  CHAR,#1BH,INTP_5    ;SKIP UNLESS ESCAPE
;        ...   . . . . .           (SPECIAL ROUTINE FOR ESCAPE CODE)
         RET
INTP_5:  CJNE  CHAR,#20H,INTP_6    ;SKIP UNLESS SPACE
;        ...   . . . . .           (SPECIAL ROUTINE FOR SPACE CODE)
         RET
INTP_6:  JC    PRINTC              ;JUMP IF CODE 20H
         MOV   CHAR,#0             ;REPLACE CONTROL CHARACTER WITH
                                   ;NULL CODE
PRINTC:                            ;PROCESS STANDARD PRINTING
;        ...   . . . . .           ;CHARACTER
         RET
;
```

9.1.9 Synchronizing Timer Overflows

8051 timer overflows automatically generate an internal interrupt request, which will vector program execution to the appropriate interrupt service routine if interrupts are enabled and no other service routines are in progress at the time. However, it is not predictable *exactly* how long it will take to reach the service routine. The service routine call takes two instruction cycles, but 1, 2, or 4 additional cycles may be needed to complete the instruction in progress. If the background program ever disables interrupts, the response latency could further increase by a few instruction cycles. (Critical sections generally involve simple instruction sequences — rarely multiplies or divides.) Interrupt response delay is generally negligible, but certain time-critical applications must take the exact delay into account. For example, generating interrupts with timer 1 every millisecond (1000 instruction cycles) or so would normally call for reloading it with the value −1000 (0FC30H). But if the interrupt interval (average over time) must be accurate to 1 instruction cycle, the 16-bit value reload into the timer must be computed, taking into account when the timer actually overflowed.

This simply requires reading the appropriate timer, which has been incremented each cycle since the overflow occurred. A sequence like the one below can stop the timer, compute how much time should elapse before the next interrupt, and reload and restart the timer. The double-precision calculation shown here compensates for any amount of timer overrun within the maximum interval. Note that it also takes into account that the timer is stopped for seven instruction cycles in the process. All interrupts are disabled, so a higher priority request will not be able to disrupt the time-critical code section.

```
...     . . . . .
CLR     EA                  ;DISABLE ALL INTERRUPTS
CLR     TR1                 ;STOP TIMER 1
MOV     A,#LOW(—1000+7)     ;LOAD LOW-ORDER DESIRED COUNT
ADD     A,TL1               ;CORRECT FOR TIMER OVERRUN
MOV     TL1,A               ;RELOAD LOW-ORDER BYTE.
MOV     A,#HIGH(—1000+7)    ;REPEAT FOR HIGH-ORDER BYTE.
ADDC    A,TH1
MOV     TH1,A
SETB    TR1                 ;RESTART TIMER
...     . . . . .
```

9.1.10 Reading a Timer/Counter "On-the-Fly"

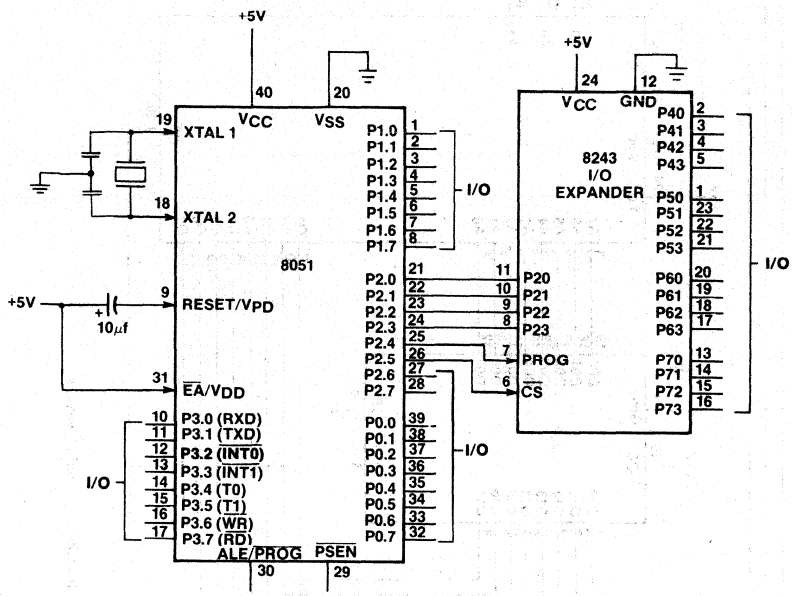
The preceding example simply stopped the timer before changing its contents. This is normally done when reloading a timer so that the time at which the timer is started (i.e. the "run" flag is set) can be exactly controlled. There are situations, though, when it is desired to read the current count without disrupting the timing process. The 8051 timer/counter registers can all be read or written while they are running, but a few precautions must be taken.

Suppose the subroutine RDTIME should return in [R1], [R0] a sixteen-bit value indicating the count in timer 0. The instant at which the count was sampled is not as critical as the fact that the value returned must have been valid at some point while the routine was in progress. There is a potential problem that between reading the two halves, a low-order register overflow might increment the high-order register, and the two data bytes returned would be "out of phase." The solution is to read the high-order byte first, then the low-order byte, and then confirm that the high-order byte has not changed. If it has, repeat the whole process.

```
RDTIME:  MOV   A,TH0          ;SAMPLE TIMER0 (HIGH)
         MOV   R0,TL0         ;SAMPLE TIMER0 (LOW)
         CJNE  A,TH0,RDTIME   ;REPEAT IF NECESSARY
         MOV   R1,A           ;STORE VALID READ
         RET
```

9.2 PERIPHERAL INTERFACE EXAMPLES

This section should give the designer an insight into connecting external peripherals and memories to the MCS-51 family.



The following software driver is required to interface to the 8243.

```
                Mixing Parallel Output, Input, and
                Control Strobes on Port 2.

IN8243          INPUT DATA FROM AN 8243 I/O EXPANDER
                CONNECT TO P23-P20
                P25 & P24 MIMIC CS/ & PROG
                P27-P26 USED AS INPUTS
                PORT TO BE READ IN ACC

IN8243  ORL     A#11010000B
        MOV     P2, A       ;OUTPUT INSTRUCTION CODE
        CLR     P2, 4       ;FALLING EDGE OF PROG
        ORL     P2,#00001111B   ;SET FOR INPUT
        MOV     A, P2       ;READ INPUT DATA
        SETB    P2, 4       ;RETURN PROG HIGH
```

Figure 9-3. I/O Expansion Using an 8243

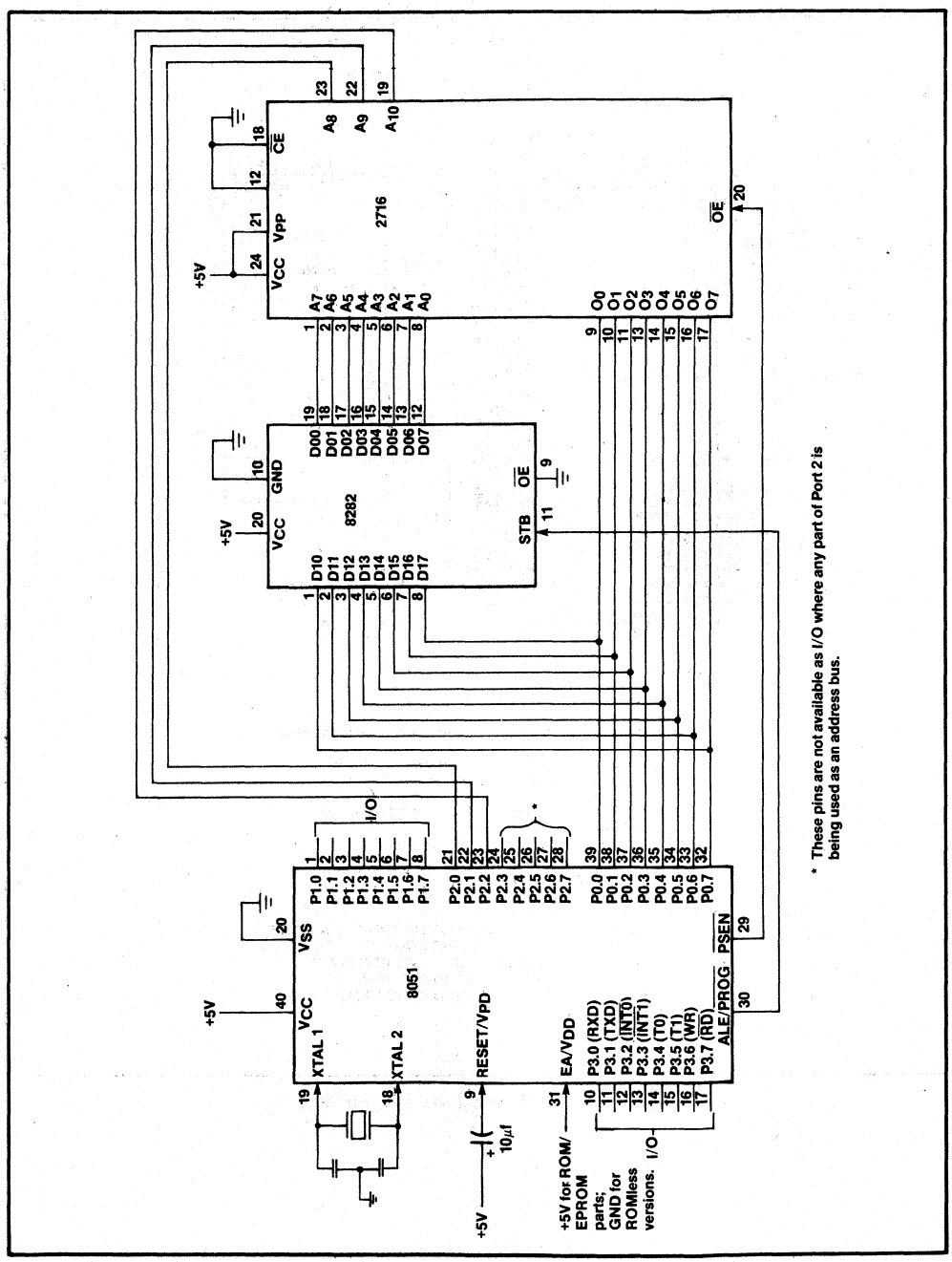


Figure 9-4. External Program Memory Using a 2716

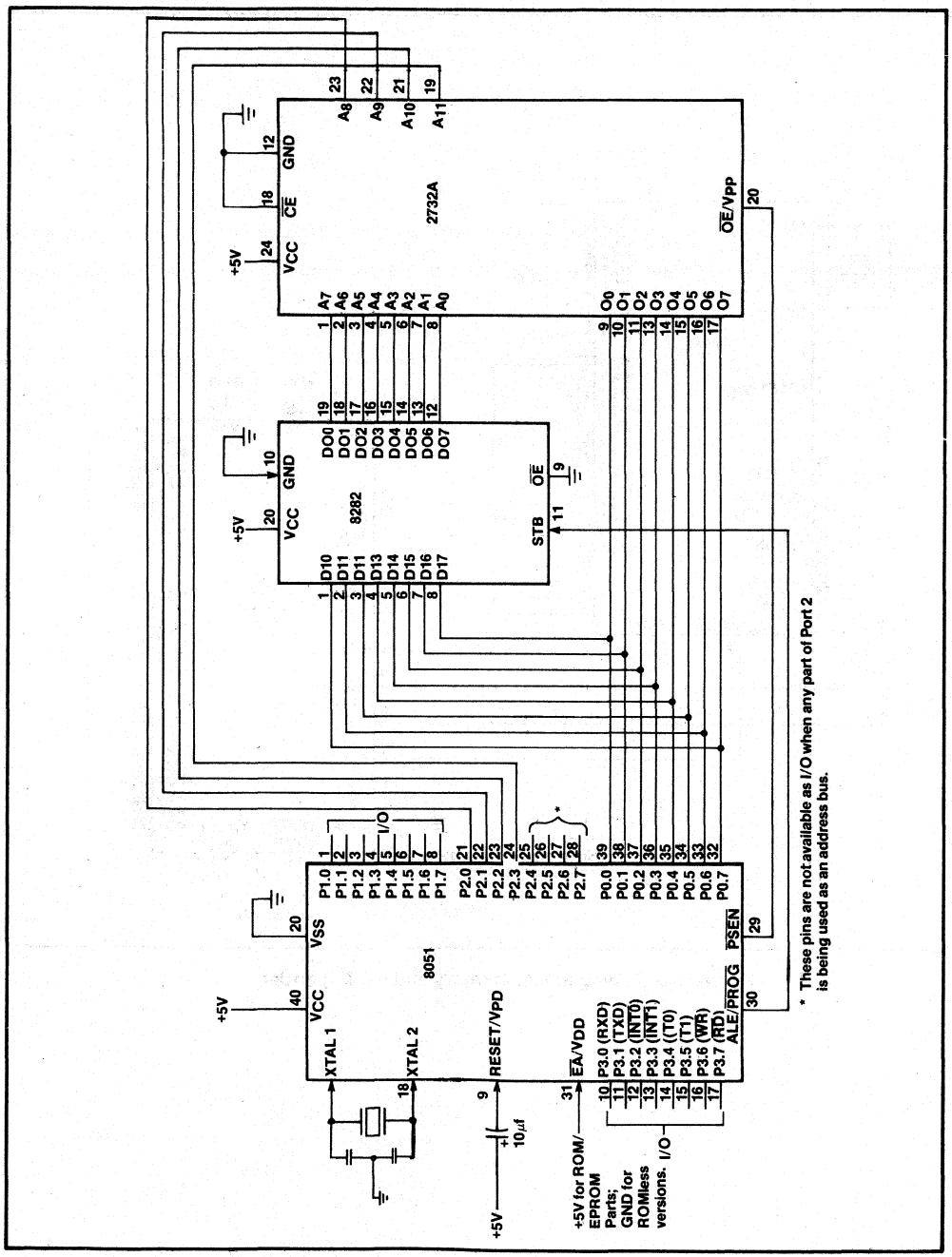


Figure 9-5. External Program Memory Using a 2732

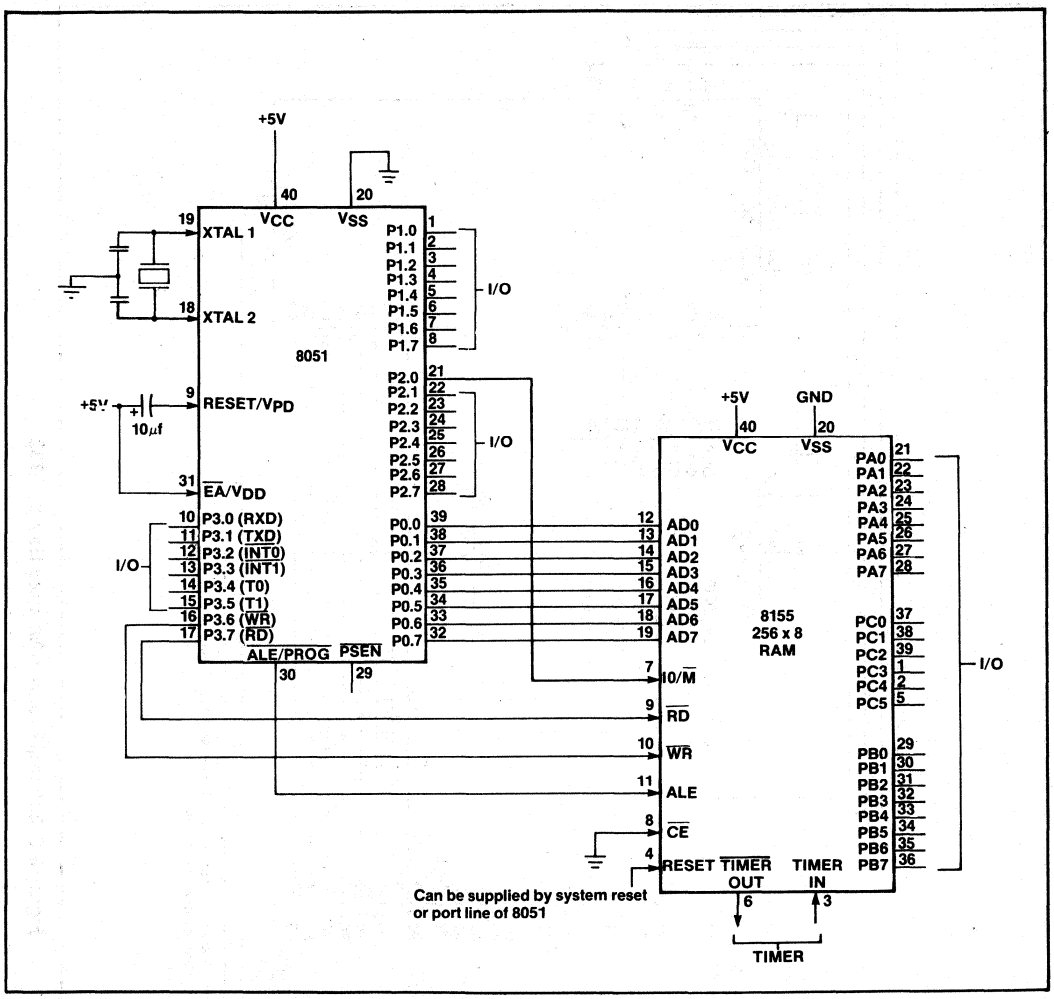


Figure 9-6. Adding a Data Memory and I/O Expander

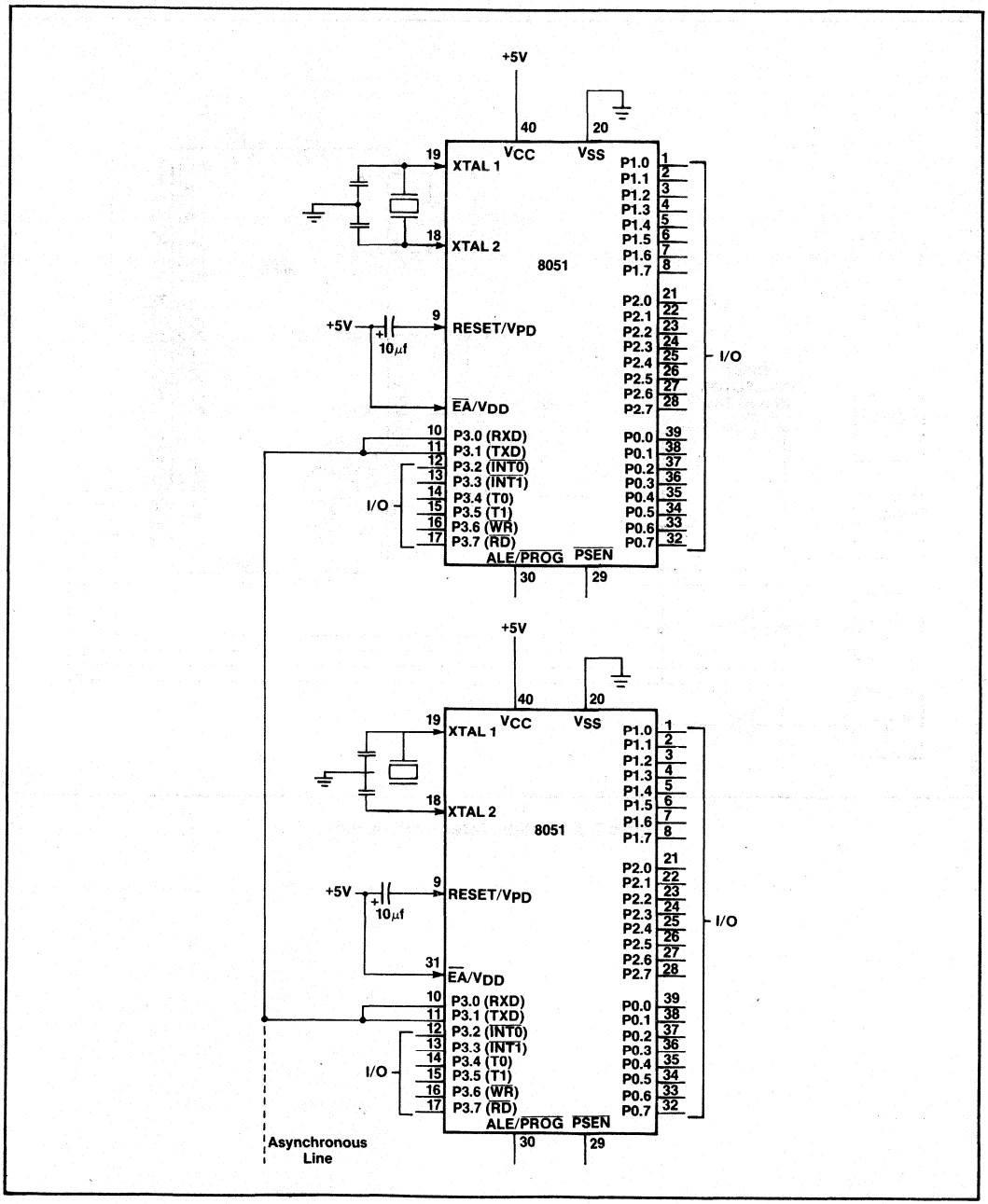


Figure 9-7. Multiple 8051's Using Half-Duplex Serial Communication

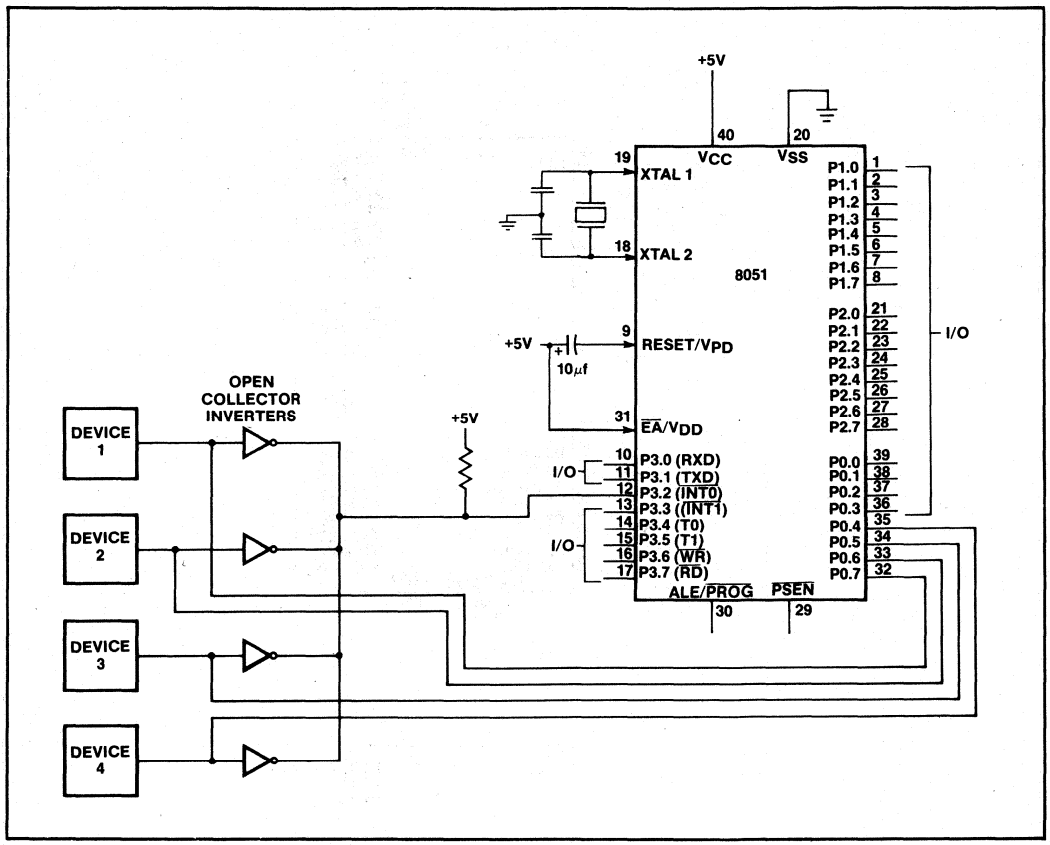


Figure 9-8. Multiple Interrupt Sources

ELECTRONIC PRODUCT DESIGN

MCS-51: a microcomputer optimised for control

A microcomputer optimised for control applications needs fast, real-time operation, rapid context switching facilities and efficient bit and byte operations, writes Intel's Howard Kornstein who describes how the MCS-51 family fits these requirements.

Two unique structures form the foundation of microelectronic computers—the microprocessor and the microcomputer. Microprocessors utilise most of a silicon die for central processing unit resource. Here, the driving philosophy of chip design is to maximise the CPU architecture to serve applications problems requiring high performance processing. The microcomputer, on the other hand, utilises the available silicon die to provide a total computer structure. This will encompass the central processing unit as well as program memory, data memory, input/output, interrupt unit and ancillary functions. The emergence of an entire computer structure on a chip was a significant advance in semiconductor technology—certainly as significant an advance as the creation of the microprocessor itself.

With the introduction of the single chip microcomputer, the application range available for computerisation was increased in a highly significant way. The most popular applications of microcomputers was in electronic logic replacement or electro-mechanical control system replacement. Typical examples of such applications are replacement of electro-mechanical washing machine controller elements or discrete logic traffic light control systems.

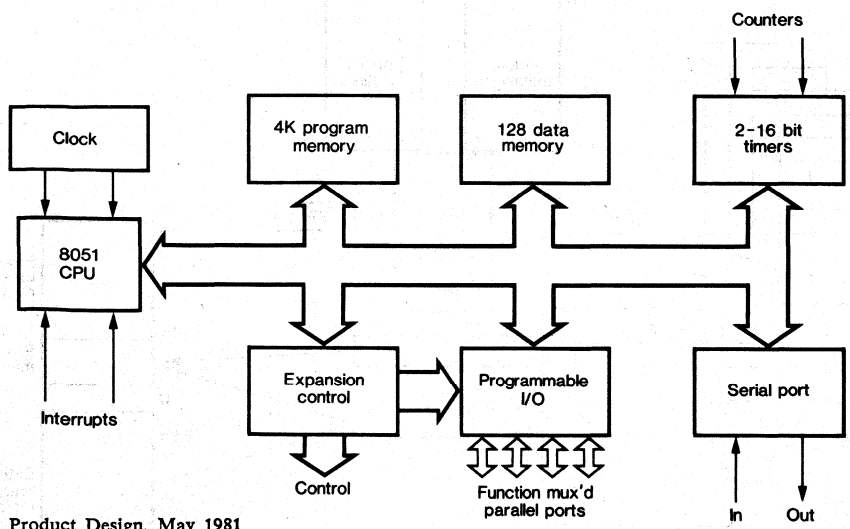
Single chip microcomputers were also very successful in applications involving control of peripherals, as a slave computer for a minicomputer or a master microcomputer. Sequencing systems of different types were a common implementation for the single chip computer, and so were dedicated closed loop systems. Microcomputers have also been used to produce energy systems for heating control in both industrial applications and in conventional central heating systems.

Looking at these diverse applications of single chip microcomputers we can see that "typical" microcomputer applications represents a control-orientated type of activity—where the microcomputer is involved in a closed loop system, operating in real-time, dedicated toward some specific type of control function. Because of this strong orientation towards control, successful microcomputer families have been structured with an architecture that is well suited for dealing with the control environment. Control in general is typified by real-time processing, high speed execution, fast context change and facilities for effective decision making and logic implementation.

The first such microcomputer family was Intel's MCS-48 range. Because this family of devices was particularly orientated towards control applications it was dubbed a "micro-controller" class of computer. The central member of the MCS-48 family was the 8048 which provided one thousand words of pro-

Fig. 1: 8051 block diagram

Fig 1



gram memory, 64 bytes of read/write data memory, a comprehensive instruction set that was control oriented, an 8 bit central processing unit, 27 input/output lines, a real time clock, clock generator and interrupt unit all on a single chip. This device became the industry's standard microcomputer.

Many other family members soon followed; these included the 8021, a very low cost version of the 8048. Another device, the 8022, not only has an 8048 structure on chip but also a complete analogue sub-system including A/D converter, analogue mulitplexer and signal conditioning ports. The 8041 is a slave microcomputer for hierarchical systems; the 8049 is a high performance 8048 with twice the program memory and data memory area as the 8048. Establishing a family of devices with a common architecture provided to systems designers the ability to pick a processor particularly cost effective in their application.

It is worthwhile to look at the facilities necessary in a microcomputer and see how the 8051, Intel's fourth generation microcomupter, provides these facilities. The most critical requirement of a microcomputer's design is to maximise the effective use of the on-chip resources. After all, a single chip microcomputer has a fundamentally finite resource.

For this reason, every resource on the chip must be used to maximum effect. This is particularly important in the instruction set, where one has to consider the fixed amount of program memory space on the chip. There is no room for inefficiency in code compaction for a device that has such finite resources.

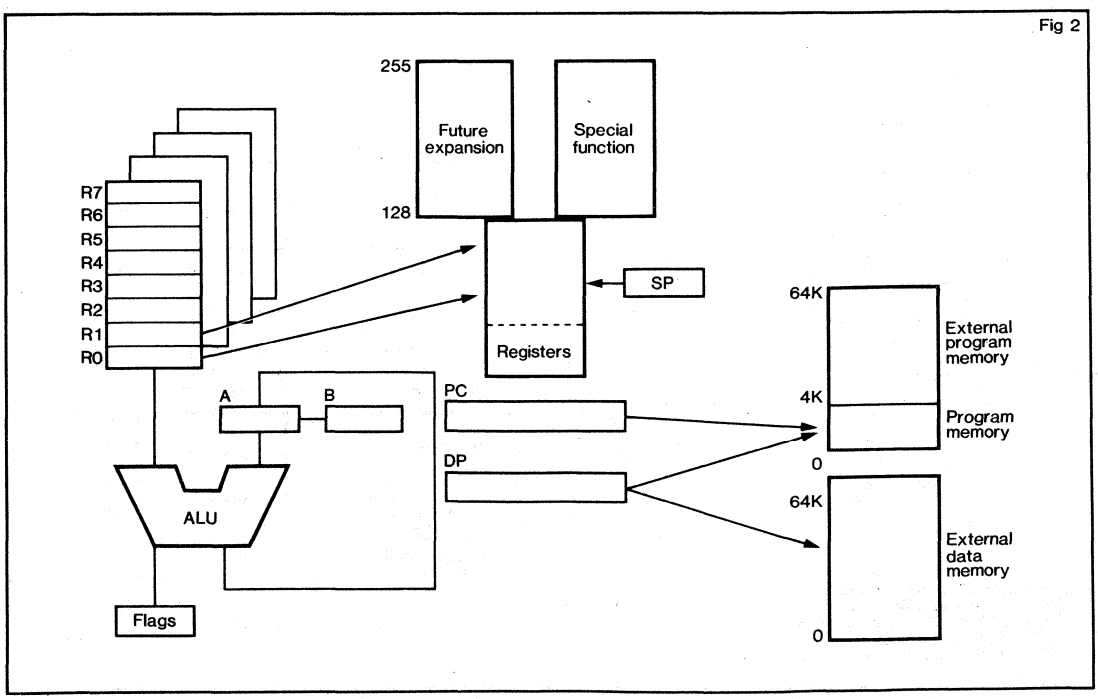
The 8051 provides several approaches for maximising the efficiency of this memory resource. The simplest way that the 8051 addresses the finite resource problem is to quadruple the amount of program memory space provided above the basic 8048 (4k of program memory on chip). The instruction set of the 8048, which has always been efficient in areas of code compaction and compaction efficiency, has been improved in the 8051. Over 50 per cent of the instructions of the 8051 are single byte, and over 80 per cent are no more than two bytes. Many "efficient" new instructions have been added to the 8051; for example, memory to memory moves, compare and branch if not equal, and multiply and divide.

By not employing a convential Von Neuman structure (common data and program space), very compact addressing forms are possible for on-chip data memory. The input/output resources of the 8051 are memory mapped and comprise four input/output ports totalling 32 I/O bits in a 40 pin package. The I/O can be configured by a "quasi bi-directional" port which allows any bit on the port to be used either as input or output, and allows for read-modify-write operations.

To match the control orientation of most single chip applications the 8051 provides specific facilities that enhance decision making. The 8051 provides a "jump-on-comparison" within a decision table (the "do case" statement), a very fast multiply and divide (4 microseconds) which facilitates high speed servocontrol computations, enhanced tale lock-up facilities and a very sophisticated instruction subset dealing with single bit variables. So comprehensive are the 8051 bit-orientated instructions that this group of instructions,

Fig. 2: 8051 registers

Fig 2



associated bit memory and I/O provides a "processor within a processor"—the Boolean processor.

The Boolean processor consists of a one bit accumulator, the carry flag, and 128 directly addressable bit memory variables in the data memory. Another 128 bit variables are specified in the input/output or functional control address space. The Boolean operators include clear, set, complement, and, or, move and jump-on bit. With the Boolean processor, it is possible to directly convert complex combinational or sequential logic equations into a program implementation executed at high speeds. An example of such an implementation is shown in Fig. 3.

One of the facilities that allows high speed control in the 8051 is its advanced interrupt structure. The interrupt unit of the 8051 provides five sources of interrupt, two of which are from external conditions, two from on-chip counter timers and one from the serial I/O port. The interrupt logic provides two levels of interrupt priority, as well as having individual interrupt masking and global interrupt enable or disable.

To provide very fast context switching in the 8051, four on-chip register banks are incorporated in the device. This elimates the need for saving registers on the occurrence of an interrupt. The device also contains a conventional stack and stack pointer register. Perhaps what makes the 8051 so effective in control is the basic execution cycle time—most instructions for the 8051 are executed in one microsecond. The device is implemented in Intel's HMOS high-speed technology.

An important aspect of microcomputer implementation is the provision of portability of software over a range of applications. As was mentioned, the MCS-48 family includes many devices, the 8021, 8022, 8048, 8041, 8049, all aimed at providing a suitable architecture to match a differing range of cost and performance criteria in varying microcontroller applications. These computers use a common architecture, and a common instruction subset runs through all of the devices, although the 8048 and 8049 provide a more enhanced instruction set. The 8051 provides an instruction set for very high performance single chip microcomputer applications but still provides code compatibility with its predecessors in the MCS-48 range. A subset of the 8051 instruction set covers the entire range of Intel's MCS-48 line from the 8021 through to the 8049 device. This allows for portability of 8051 code upward from these 802X and 804X devices and portability downward from the 8051 into these other members of the Intel single chip family.

BFUNC3 Solve a random logic function of 6 variables using straight-line logical instructions on MCS-51 Boolean variables.

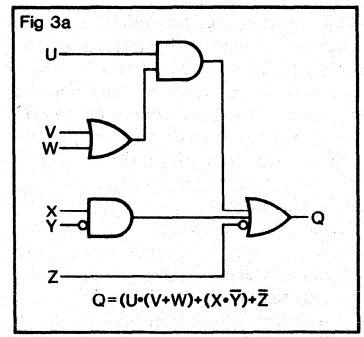
```
MOV  C,V
ORL  C,W    ; Output of OR gate
ANL  C,U    ; Output of top AND gate
MOV  FO,C   ; Save intermediate state
MOV  C,X
ANL  C,/Y   ; Output of bottom AND
              gate
ORL  C,FO   ; Include value saved above
ORL  C,/Z   ; Include last input variable
MOV  G,C    ; Output computed result
```

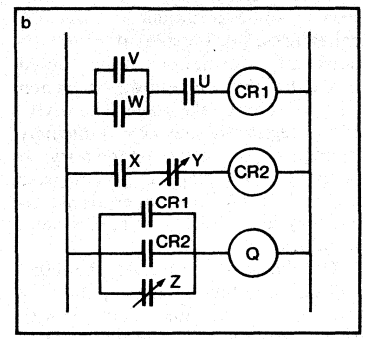
Fig. 3: TTL (a) and relay logic (b) implementations of a Boolean function. The table above is a software version of the same function using the 8051's direct bit addressing to achieve maximum coding efficiency

It is important that single chip microcomputers have expansion capabilities beyond the single chip level, otherwise a single chip system can be a potential bottleneck to configuration flexibility. If any of the on-chip resources must be exceeded, whether it be program memory, data memory, I/O, interrupt facility, number of counter timers and if no expansion path is provided beyond the single chip itself, a designer can run into a catastrophic constraint. The single chip microcomputer must be easily expanded in any of these resource areas.

The 8051 is expandable in all of these directions. The code space and data memory space can be extended to 128k. Program memory of data memory external to the 8051 can be standard ROM, EPROM or RAM memory, or can be highly integrated members of the 8085 family. These provide not only memory expansion but also I/O expansion with every chip added.

Furthermore, the 8051 provides expandability through networking. This is in keeping with recent trends in providing distributed computing in microcomputer systems. The 8051





provides a full duplex serial communications port, on-chip. One of the ways that this port can be utilised is in a multi-processor configuration, using a protocol in which one of the bits in the serial word is defined as an "attention bit" which notifies 8051s in the network to look at a received control word to determine if it is being addressed to perform a function. To facilitate ease of design, there is a pin compatible EPROM version of the 8051 designated the 8751. This allows a design to progress from prototyping into production without any complications in testing and also allows trial-marketing of an 8051 based product.

Architecture

The 8051 microcomputer is a register orientated architecture. The device has an accumulator and eight working registers, two of which are indirect pointers into the data memory area. The registers are designated R0 to R7. The organisation is in keeping with the philosophy that a register orientated machine is particularly effective in both control and general purpose computing applications by providing an effective way of separating the temporaries and variables in a system program.

The working registers allow a convenient and fast work space area for temporaries (a "scratchpad"). More permanent variables are maintained in regular data memory area. Aside from speed improvement, this organisation provides a more natural way to structure information for any program module.

The 8051 provides four register banks for very fast context switching when an interrupt occurs. If all banks are not needed the register banks can act as normal data memory. The 8051 has no "accumulator bottleneck" seen in some accumulator based architectures, i.e. the 8051 does not require all information to be routed through the accumulator when transfers are to take place. The device allows register to memory, immediate to register or memory, memory to memory, or register to register moves. Immediate operations to memory are used to initialise system variables.

The basic structure of the 8051 CPU can be seen in Fig. 2, showing the four register banks, data memory area, program memory area, arithmetic logic unit and the accumulator. Register B is used as an extension register during the operations of multiply and divide. The program counter of the 8051 is 16 bits long, thus providing facility of addressing 64k of program memory The 8051 automatically recognises the address space which is on-chip and that which is off-chip. A program reference into address 4096 or greater is automatically generated off-chip. A member of the MCS-51 family, the 8031, has all program memory off-chip. The 8031 is a low-cost member of the 8051 family and provides another way for low production runs to be realised in EPROM based memory.

A second pointer register in the 8051 is the data pointer. This is a 16-bit register pointing into either program or data memory off-chip. The data pointer provides a base register for table look-up in ROM, external data transfers, or into jump tables which are held in the program memory area. The data pointer may be manipulated as a 16-bit number or as a separate data pointer high/data pointer low locations. The data pointer may initialise with a 16-bit immediate move and it can be indexed.

Two other memory address registers have been mentioned, R0 and R1. These registers can be utilised as pointers for indirect moves in the on-chip data memory area. An 8-bit stack pointer register also points into the RAM area. The 8051 thus provides conventional push and pop operations, subroutine calls and returns which can be nested, and the ability to save program counter and status information on the occurrence of interrupt.

The data memory area of the 8051 is separated into normal data memory and an area known as special function registers. The special function registers provide a memory mapped register facility for controlling resources and accessing input/output. Registers in the special functions area include the accumulator, B register, program status work, interrupt unit control registers, timer-counter registers and the input/output registers of the 8051.

Memory mapped I/O is a very useful capability in I/O handling. It allows instructions such as compare, logical instructions or arithmetic operations to take place between input/output ports and the accumulator, or immediate operations to manipulate I/O.

Instruction set

The 8051 provides an instruction set which is a superset of the MCS-48 family, which is highly compact and symmetrical. Instruction classes for the 8051 include arithmetic operations, logical operations, data transfers, Boolean variable manipulation and program and machine control instructions. The arithmetic operations include the add, subtract, multiply, divide, increment and decrement. Logical operations include and, or, exclusive or, complement as well as the rotate instructions. Data transfers include move operations, push and pop and the exchange instruction. Boolean variables allow the programmer to set or clear a bit, complement a bit and perform and, or and move bit.

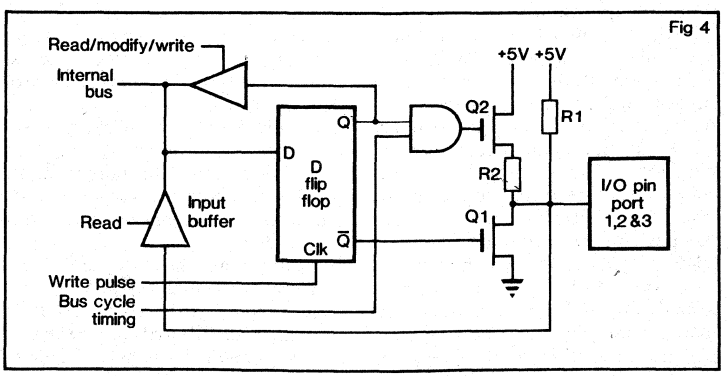
Program control instructions include jumps, calls, special jump instructions that allow jump tables or short "looping" jumps. The 8051 has four basic addressing modes, register, direct, register indirect, and immediate. The device also provides comprehensive addressing dealing with tables. This is particularly important in microcontroller-type applications.

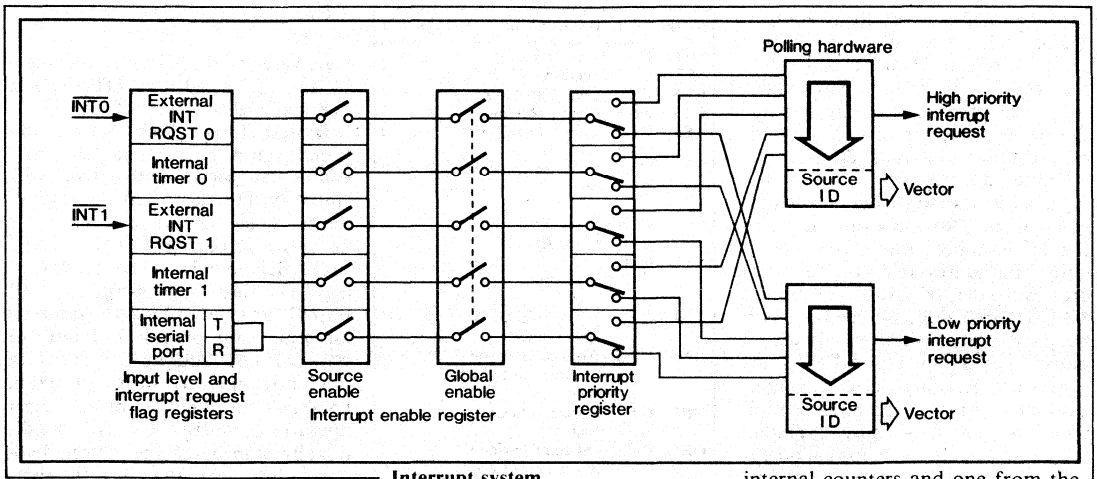
Bit operations

One of the factors determining how long it will take a microcomputer to complete a given chore is the number of instructions it must execute. What makes a given computer architecture particularly well—or poorly—suited

Fig. 4: The I/O port versatility results from the "quasi-bidirectional" output structure depicted below. An output latch bit associated with each pin is updated by direct addressing instructions when that port is the destination. The latch state is buffered to the outside world by R1 and Q1, which may drive a standard TTL input. (In TTL terms, Q1 and R1 resemble an open-collector output with a pull-up resistor to Vcc).

Operations using an input port or pin as the source open and use the logic level of the pin itself, rather than the ouput latch contents. This level is affected by both the microcomputer itself and whatever device the pin is connected to externally. The value read is essentially the "OR-tied" function of Q1 and the external device. If the external device is high-impedence, such as a logic gate input or a three state output in the third state, then reading a pin will reflect the logic level previously ouput.





Interrupt system

External events and the real-time-driven on-chip peripherals require service by the CPU asynchronous to the execution of any particular section of code. To tie the asynchronous activities of these functions to normal program execution, a sophisticated multiple-source, two-priority-level, nested interrupt system is provided. Interrupt response latency ranges from 3μs to 7μs when using a 12 MHz crystal. The 8051 acknowledges interrupt requests from five sources: Two from external sources via the INT0 and INT1 pins, one from each of the two internal counters and one from the serial I/O port. Each interrupt vectors to a separate location in program memory for its service program. Each of the five sources can be assigned to either of two priority levels and can be independently enabled and disabled. Additionally all enabled sources can be globally disabled or enabled. Each external interrupt is programmable as either level- or transition-activated and is active-low to allow the "wire or-ing" of several interrupt sources to the input pin. The interrupt system is shown diagrammatically above.

for a class of problems is how well its instruction set matches the tasks to be performed. The better the 'primative' operations correspond to the steps taken by the control algorithm, the lower the number of instructions needed, and the quicker the program will run. All else being equal, a CPU supporting 64-bit arithmetic directly could clearly perform floating-point maths faster than a machine bogged-down by multiple-precision subroutines. In the same way, direct support for bit manipulation naturally leads to more efficient programs handling the binary input and output conditions inherent in digital control problems.

Let's see how the four basic elements of a digital computer—a CPU with associated registers, program memory, addressable data RAM, and I/O capability—relate to Boolean variables.

CPU. The 8051 CPU incorporates special logic devoted to executing several bit-wide operations. All told, there are 17 such instructions, all listed in the table. Not shown are 94 other (mostly byte-oriented) 8051 instructions.

Program Memory. Bit-processing instructions are fetched from the same program memory as other arithmetic and logical operations. Several sophisticated program control features like multiple addressing modes, subroutine nesting, and a two-level interrupt structure are useful in structuring Boolean processor-based programs.

Boolean instructions are one, two, or three bytes long, depending on what function they perform. Those involving only the carry flag have either a single-byte opcode or an opcode followed by a conditional-branch destination byte. The more general instructions add a "direct address" byte after the opcode to specify the bit affected, yielding two or three byte encodings. Though this format allows potentially 256 directly addressable bit locations, not all of them are implemented in the 8051 family.

Data Memory. The bit instructions can operate directly upon 144 general purpose bits forming the Boolean processor "RAM." These bits can be used as software flags or to store program variables. Two operand instructions use the CPU's carry flag ("C") as a special one-bit register; in a sense, the carry is a "Boolean accumulator" for logical operations and data transfers.

Input/Output. All 32 I/O pins can be addressed as individual inputs, outputs, or both, in any combination. Any pin can be a control strobe output, status (test) input, or serial I/O link implemented via software. An additional 33 individually addressable bits reconfigure, control, and monitor the status of the CPU and all on-chip peripheral functions (timer/counters, serial port modes, interrupt logic, and so forth).

Direct bit addressing

The most significant bit of the direct address byte selects one of two groups of bits. Values between 0 and 127 (00H and 7FH) define bits in a block of 32 bytes of on-chip RAM, between RAM addresses 20H and 2FH. They are numbered consecutively from the lowest-order byte's lowest-order bit through the highest-order byte's highest-order bit.

Bit addresses between 128 and 255 (80H and 0FFH) correspond to bits in a number of special registers, mostly

used for I/O or peripheral control. These positions are numbered with a different scheme than RAM: the five high-order address bits match those of the register's own address, while the three low-order bits identify the bit position within that register.

There are 20 special function registers in the 8051, but the advantages of bit addressing only relate to the 11 described below. Five potentially bit-addressable register addresses (0C0H, 0C8H, 0D8H, 0E8H, & 0F8H) are being reserved for possible future expansion in microcomputers based on the MCS-51 architecture. Reading or writing nonexistent registers in the 8051 series is pointless, and may cause unpredictable results. Byte-wide logical operations can be used to manipulate bits in all *non*-bit addressable registers and RAM.

The accumulator and B registers (A and B) are normally involved in byte-wide arithmetic, but their individual bits can also be used as 16 general software flags. Added with the 128 flags in RAM, this gives 144 general purpose variables for bit-intensive programs. The program status word (PSW) is a collection of flags and machine status bits including the carry flag itself. Byte operations acting on the PSW can therefore affect the carry.

Having looked at the bit variables available to the Boolean Processor, we will now look at the four classes of instructions that manipulate these bits.

State Control. Addressable bits or flags may be set, cleared, or logically complemented in one instruction cycle with the two-byte instructions SETB, CLR, and CPL. (The "B" affixed to SETB distinguishes it from the assembler "SET" directive used for symbol definition.) SETB and CLR are analogous to loading a bit with a constant: 1 or 0. Single byte versions perform the same three operations on the carry.

The MCS-51 assembly language specifies a bit address in any of three ways:

- by a number or expression corresponding to the direct bit address (0–255).
- by the name or address of the register containing the bit, the *dot operator* symbol (a period: "."), and the bit's position in the register (7–0).
- in the case of control and status registers, by the predefined assembler symbols.

Mnemonic		Description	Byte	n
SETB	C	Set Carry flag	1	1
SETB	bit	Set direct Bit	2	1
CLR	C	Clear Carry flag	1	1
CLR	bit	Clear direct bit	2	1
CPL	C	Complement Carry flag	1	2
CPL	bit	Complement direct bit	2	1
MOV	C.bit	Move direct bit to Carry flag	2	1
MOV	bit.C	Move Carry flag to direct bit	2	2
ANL	C.bit	AND direct bit to Carry flag	2	2
ANL	C.bit	AND complement of direct bit to Carry flag	2	2
ORL	C.bit	OR direct to Carry flag	2	2
ORL	C.bit	OR complement of direct bit to Carry flag	2	2
JC	rel	Jump if Carry is flag is set	2	2
JNC	rel	Jump if No Carry flag	2	2
JB	bit.rel	Jump if direct Bit set	3	2
JNB	bit.rel	Jump if direct Bit Not set	3	2
JBC	bit.rel	Jump if direct Bit is set & Clear bit	3	2

n = no. of execution cycles

Address mode abbreviations:

C — Carry flag.
bit — 128 software flags, any I/O pin, control or status bit.
rel — All conditional jumps include an 8-bit offset byte. Range is +127/–128 bytes relative to first byte of the following instruction.

MCS-51 Boolean processing instruction subset.

Data Transfers: The two-byte MOV instructions can transport any addressable bit to the carry in one cycle, or copy the carry to the bit in two cycles. A bit can be moved between two arbitrary locations via the carry by combining the two instructions. (If necessary, push and pop the PSW to preserve the previous contents of the carry.) These instructions can replace the multi-instruction sequences appearing in controller applications whenever flags or outputs are conditionally switched on or off.

Logical Operations. Four instructions perform the logical AND and logical-OR operations between the carry and another bit, and leave the results in the carry. The instruction mnemonics are ANL and ORL; the absence or presence of a slash mark ("/") before the source operand indicates whether to use the positive-logic value or the logical complement of the addressed bit. (The source operand itself is never affected.)

Bit-test Instructions. The conditional jump instruction "JC rel" (jump on carry) and "JNC rel" (jump on not carry) test the state of the carry flag, branching if it is a one or zero, respectively. (The letters "rel" denote relative code addressing.) The three-byte instruction "JB bit,rel" and "JNB bit,rel" (jump on bit and jump on not bit) test the state of *any* addressable bit in a similar manner. A fifth instruction combines the jump on bit and clear operations. "JBC bit,rel" conditionally branches to the indicated address, then clears the bit in the same two cycle instruction.

All 8051 conditional jump instructions use program counter-relative addressing, and all execute in two cycles. The last instruction byte encodes a signed displacement ranging from −128 to +127. During execution, the CPU adds this value to the incremented program counter to produce the jump destination. Put another way, a conditional jump to the immediately following instruction would encode 00H in the offset byte.

A section of program or subroutine written using only relative jumps to nearby addresses will have the same machine code independent of the code's location. An assembled routine may be repositioned anywhere in memory, even crossing memory page boundaries, without having to modify the program or recompute destination addresses. To facilitate this flexibility, there is an unconditional "short jump" (SJMP) which uses relative addressing as well. Since a programmer would have quite a chore trying to compute relative offset values from one instruction to another, ASM51 automatically computes the displacement needed given only the destination address or label. An error message will alert the programmer if the destination is "out of range."

By combining general purpose bit operations with certain addressable bits, one can "custom build" several hundred useful instructions. All eight bits of the PSW can be tested directly with conditional jump instructions to monitor (among other things) parity and overflow status.

Design

Besides packing enough density to quadruple the memory capacity of the 8048, a new single-chip microcontroller works on bits and bytes.

Microcontroller doubles as Boolean processor

The latest single-chip microcontroller family from Intel brings forth a controller whose dual personality sets it well apart. Based on HMOS, the 8051 (and family) packs enough density to offer four times the memory of the 8048; in addition, the instruction set is much more comprehensive. Not only that, the 8051 can also handle Boolean variables—a major breakthrough for microcomputers that makes the 8051 a byte and bit processor.

Boasting 60,000 transistors, compared to the 8048's 17,000, the 8051 offers 4 kbytes of ROM and 128 bytes of RAM. Other members of the new HMOS family include the 8751 EPROM version, meant for prototyping, and the 8031, which relies on external program memory. All three chips are pin-compatible; they can address 64 kbytes each of program and data memory.

But while the 8051 is a capable processor for 8-bit binary as well as BCD arithmetic, along with 8-bit logic operations, the real standout feature is its ability to handle Boolean variables. Individual bits in special-function registers (SFRs) and 128 software flags can be the operands for logical conditional-branch and transfer operations.

Although integrated with the 8051's architecture (Fig. 1), the Boolean processor may be regarded as an independent bit processor. It has its own instruction set, its own accumulator (the carry flag), its own bit-addressable RAM and its own I/O. The bit-manipulation instructions allow the direct addressing of 128 bits within the internal data RAM and 128 bits within the SFRs.

On any addressable bit, the Boolean processor can perform the following bit operations: Set, Clear, Complement, Jump If Set, Jump If Not Set, Jump If Set Then Clear, and Move To/From Carry. Be-

Bob Koehler, Marketing Product Manager
Intel Microcontroller Operation
Reprinted with permission from *Electronic Design,* Vol. 28, No. 11; copyright Hayden Publishing Co., Inc., 1980.

tween any addressable bit (or its complement) and the carry flag, the Boolean processor can perform the bit operations of AND and OR, the result going into the carry flag.

The bit-manipulation instructions provide excellent code and speed efficiency in bit-intensive applications, such as controlling the 8051's on-chip peripherals. The Boolean processor also provides a direct way to convert into software the logic equations used in random-logic design. Complex combinatorial-logic functions can be resolved without extensive data movement, byte masking and test-and-branch trees (Fig. 2).

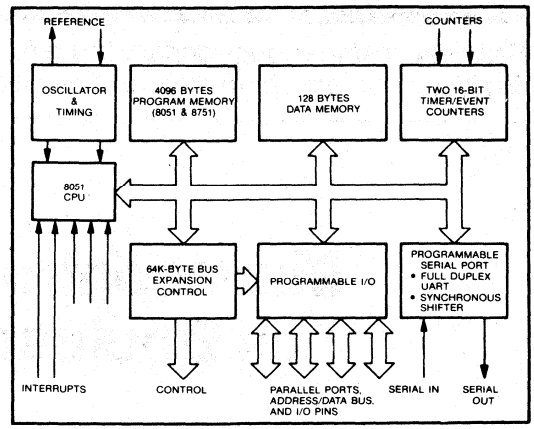
A better architecture

The 8051 CPU manipulates operands in four memory spaces: the 64-kbyte program memory, 64-kbyte external data memory, 16-bit program counter and 384-byte internal data memory, which is further divided into the 256-byte internal data RAM and 128-byte special-function register (Fig. 3). Internal data RAM contains four register banks of eight registers each as well as the stack and 128 addressable bits.

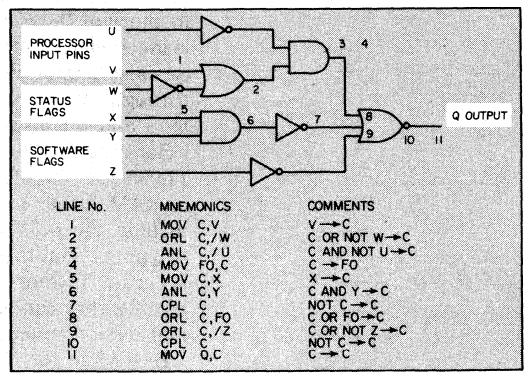
Stack depth is limited only by the available internal data RAM; the starting address is determined by an 8-bit stack pointer. Currently, the lower 128 bytes of internal RAM address space are filled with on-chip RAM—the upper 128 bytes can be added in later products without affecting existing software.

All registers (except for the program counter) reside in the SFR address space. Memory-mapped, they include arithmetic registers, pointers, I/O ports, interrupt system registers, timers and the serial port (Fig. 4).

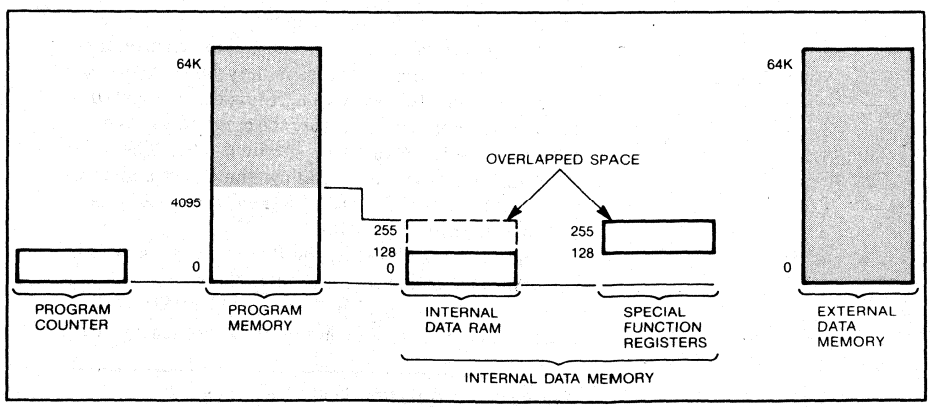
Users can directly address 128 bits in the SFR address space. The 8051 uses only 20 of the 128 bytes in the space for SFRs, so future members of the microcontroller family have room for up to 108 more



1. The architecture of the 8051 family is strongly bus-oriented. On-board counters and a programmable serial port enhance the chip's controller capabilities.



2. The Boolean operations built into the 8051 directly solve problems like this example, which might represent timing control for a keyboard. Variables in the program and Boolean operations are cued to the schematic (white).



3. Both the on-board data and program memories (white) can be supplemented by external RAM to provide about 64 kbytes for both types.

counters, ports, a/d converters and the like, without affecting existing software.

The 8051's program-memory space is not paged and accommodates relocatable code. Conditional branches are offset to the program counter. The register-indirect jump permits branching relative to a 16-bit base register; an 8-bit index register provides offset. Sixteen-bit jumps and calls allow branching to any location within the contiguous 64-kbyte program-memory address space.

Addressing modes for all needs

Source operands can be addressed five ways: *register, direct, register-indirect, immediate* and *base-register plus index-register indirect.* The first three can be used for addressing destination operands. Most instructions have a Destination, Source field that specifies data type, addressing methods and operands. Except for move instructions, the destination operand is also a source operand.

The *register, direct,* and *register indirect* addressing modes access registers in the register bank, located in internal data RAM. *Direct* and *register-indirect* modes access the 128 bytes of internal data RAM while *direct* addressing provides access to the SFRs. To access external data memory, *register-indirect* addressing is used, while the fifth mode, *base-register plus index-register indirect,* provides access to lookup tables residing in program memory.

The 8051's internal ROM, RAM, SFRs, ALU and external data bus are all 8 bits wide, but the processor can handle single-bit, 4-bit and 16-bit data types, too. Facilities for 8-bit data transfer, logic and integer-arithmetic operations are supplemented by data transfer, logic and conditional branch operations performed directly on Boolean variables.

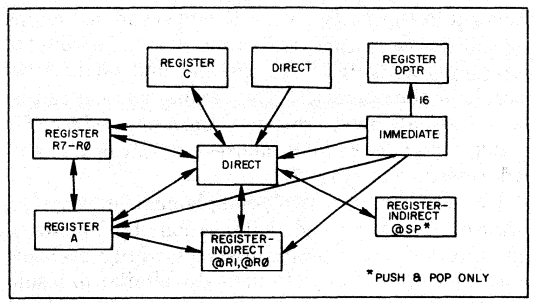
The instruction set, while similar to that of the 8048 family, is enhanced to allow expansion of on-chip CPU peripherals, to sharpen byte efficiency and execution speed, and to contain new high-power operations and new addressing modes that provide a more symmetrical instruction set. It includes 49 single-byte, 45 double-byte and 17 triple-byte instructions. At a 12-MHz clock rate, 64 of the instructions execute in 1 μs and 45 in 2 μs while multiplications and divisions take 4 μs.

Comprehensive Move instructions

The 8051 boasts other improvements as well. For one thing, data transfer, logic manipulation, arithmetic processing and real-time control capabilities have all been enhanced. Lookup tables, residing in program memory, can be accessed by indirect moves, which enables a byte to be transferred from the location whose address is the sum of a 16-bit base register and the 8-bit index register. This feature is

Arithmetic registers	:	Accumulator, B register, program-status word
Pointers	:	Stack pointer, data pointer (high and low)
Parallel-I/O ports	:	Port 0, port 1, port 2, port 3
Interrupt system	:	Interrupt-priority control, interrupt-enable control
Timers	:	Timer mode, timer control, timers 0 and 1 (high and low)
Serial-I/O port	:	Serial control, serial-data buffer

4. Special-function registers, shown in Fig. 3, are used for arithmetic, I/O, timers and the interrupt system.



5. Move operations are possible from nearly any storage location to nearly any other. Except where noted, all moves operate on bytes.

convenient for programming translation algorithms such as ASCII-to-seven-segment conversions.

A byte location within a 256-byte block of external data memory can be accessed through *register-indirect* addressing via an 8-bit base register. Furthermore, any location within the full 64-kbyte external data memory address space can be accessed through *register-indirect* addressing via a 16-bit base register.

Byte-constant moves (or immediate moves) and byte-variable moves are highly orthogonal (Fig. 5). In other words, a byte operand located anywhere in the internal data memory can be transferred to nearly any other location in the memory by a single instruction, as can immediate operands. The direct-address-to-direct-address move permits the value in a port to be moved to the internal data RAM without using any registers or the accumulator.

In addition, the accumulator can be exchanged with a register in a selected register bank, with a *register-indirect*-addressed byte in the internal data RAM, or with a *direct*-addressed byte in the internal data RAM or SFR. The least significant half-byte of the accumulator and of a *register-indirect*-addressed byte in internal data RAM can also be exchanged.

The 8051 permits a second operand to perform the logic operations of AND, OR and XOR on the accumulator. This second operand can be an immediate value, a register in a selected register bank, a *register-indirect*-addressed byte of internal data RAM, a *direct*-addressed byte of internal data RAM

or an SFR.

The three logic operations can also be performed on a *direct*-addressed byte of internal data RAM or an SFR, using the accumulator as the second operand. Any bit anywhere in the internal data RAM or SFRs can be set, cleared or complemented using the logic operations with *immediate*-addressing to *direct* addressing (Fig. 6). The remaining logic operations work only on the accumulator.

Only unsigned binary-integer arithmetic is performed in the ALU. In the two-operand operations of add, add-with-carry and subtract-with-borrow, the accumulator is the first operand and receives the result of the operation. The second operand can be an immediate byte, a register from a selected register bank, a *register-indirect*-addressed byte or a *direct*-addressed byte (Fig. 7).

The 8051 handles two's-complement-integer (i.e., signed) addition and subtraction by software-monitoring of the program-status word's overflow flag. One operation arithmetically similar to a subtraction is the instruction Compare And Jump If Not Equal, which performs a conditional branch if a register in a selected bank (or an *indirect*-addressed byte of internal data RAM) does not equal an immediate value, or if the accumulator does not equal a byte of *direct*-addressable internal data RAM or an SFR.

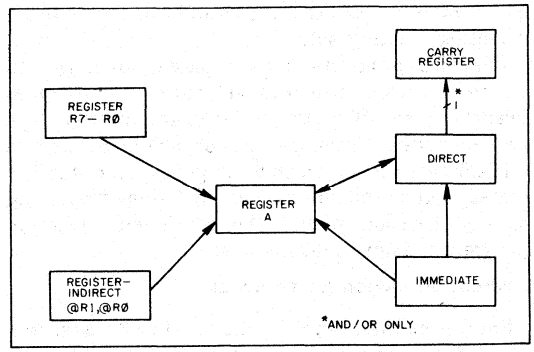
Three arithmetic functions operate exclusively on the accumulator: Decimal Adjust For Addition, Jump If Accumulator Is Zero and Jump If Accumulator Is Not Zero. Conditional branches may be taken, based on the value in the accumulator, whether zero or not.

Software counters are easy to implement with the 8051 because increment and decrement operations can be performed on the accumulator, a register in a selected register bank, an *indirect*-addressed byte in the internal data RAM, or a byte in the *direct*-addressed internal RAM or SFR. The operation Decrement And Jump If Not Zero facilitates efficient loop control. This operation can test a register in a selected register bank, any SFR, or any byte or internal data RAM accessible through *direct* addressing, and then force a branch if it is not zero. In addition, the 16-bit data pointer can be incremented.

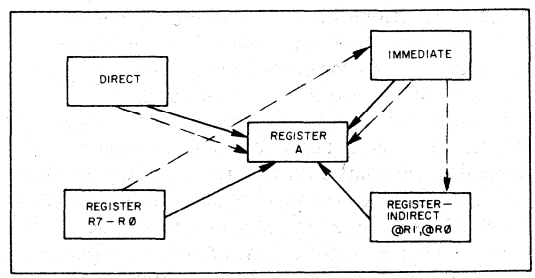
Peripherals galore

Counting, timing and communications in a real-time, priority-oriented environment are the keystones of control applications. External events and peripherals require asynchronous servicing by the CPU. A sophisticated nested interrupt system ties these asynchronous activities to regular program execution with response latencies of 3 to 7 μs.

The 8051 services interrupt requests coming from



6. Logic operations include many source and destination locations and —except where noted—AND, OR and XOR.



7. Various operations involve different registers: Add, Add With Carry and Subtract With Borrow are indicated by black lines, Compare And Jump If Not Equal by broken lines.

five sources—two via the INT 0 and INT 1 pins, one from each of the two internal counters and one from the serial-I/O port. Each vectored interrupt accesses a location in program memory for its service routine. Each of the five interrupt sources can not only be assigned to one of two priority levels but can also be enabled and disabled independently. Finally, all enabled sources can be disabled or enabled globally (Fig. 8).

The 8051 has instructions that treat its 32 I/O lines as 32 individually addressable bits or as four parallel 8-bit ports (Fig. 9). Each pin of port 0 can be configured as an open drain output or as a high-impedance input. Ports 1, 2, and 3 are quasi-bidirectional buffers that can sink or source one TTL load. Port 0 also provides a microprocessor bus with multiplexed low-order address and data; it serves as an interface with standard MCS-85 memories and MCS-80 peripherals. Port 2 provides the high-order address bus needed to expand the 8051 with external program or data memory. The bus driver can sink or source two TTL loads.

At 12 MHz, the program-memory cycle is 500 ns and the access times from stable address and program-store enable are approximately 320 ns and 150 ns, respectively. For external memory, the cycle

is 1 μs and the corresponding access times are approximately 600 ns and 250 ns (pins $\overline{RD}$ and $\overline{WR}$ of port 3).

The remaining pins of port 3 can be used for other functions: Interrupt-request inputs (INT0 and INT1), counter inputs (T0 and T1) and the serial port's receiver input (RXD) and transmitter output (TXD).

Timing is essential

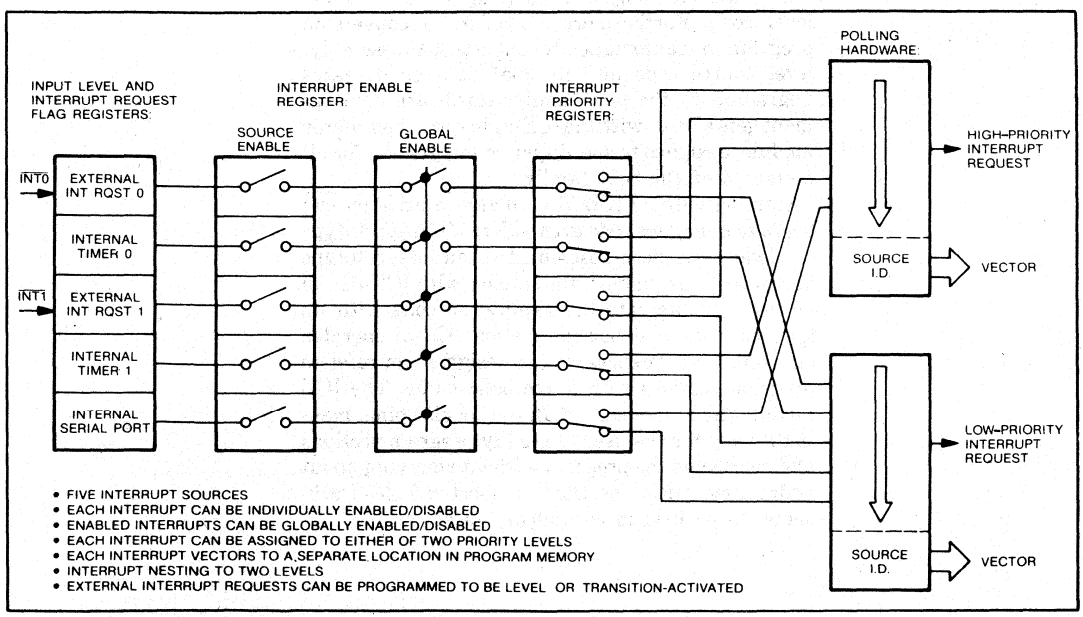
The 8051 contains two 16-bit counters for measuring time intervals, counting events, measuring pulse widths and generating periodic interrupt requests—a big help in control applications. Each can be programmed to operate in one of three modes: Mode 0 is similar to an 8038's 8-bit timer with prescaler and 8-bit event counter; mode 1 is a 16-bit time-interval or event counter; and mode 2 is an 8-bit time-interval or event counter with automatic reload upon overflow. Counter 0 can also be programmed in a fourth mode that divides it into one 8-bit time interval or event counter and one 8-bit time-interval counter.

The 8051's counters not only offer flexibility and 16-bit precision, they also can handle very high input frequencies. For external inputs, the frequency range is from 0 Hz to 0.5 MHz; when programmed for an input of the internal oscillator, they cover 0.1 to 1 MHz. To measure pulse width directly, an external input can be gated to the counter by a second external source. The counters are started and stopped under software control. Each counter sets its interrupt request flag when it overflows from all ONEs to all ZEROs, or to the auto-reload value.

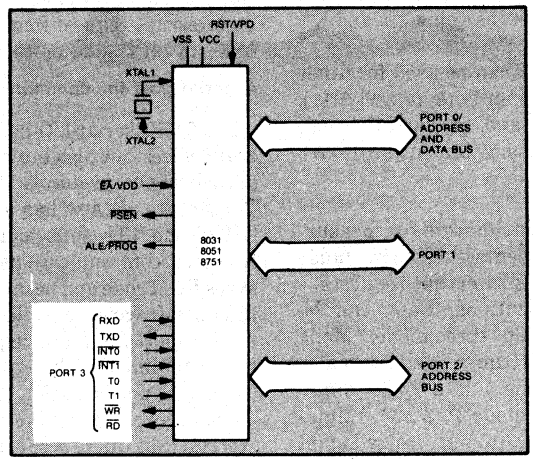
A serial port for communictions

The 8051's serial-I/O port can link not only peripheral devices but also multiple 8051s, using standard asynchronous protocols in a full-duplex operation. The serial port also has a synchronous mode, using CMOS and TTL shift registers to expand I/O lines. A serial-communications interface in hardware saves ROM code and permits a much higher data rate than a software-controlled interface. In response to a serial-port interrupt request, the CPU can service the serial link simply by reading from or writing to the serial port's buffer.

The receiver is double-buffered to eliminate the overrun that would occur if the CPU failed to respond to the receiver's interrupt before the beginning of the next frame. Furthermore, the 8051 offers false-start-bit rejection on received frames and, for noise rejection, takes a best-two-out-of-three poll on three samples near the center of each received bit. When interfacing with standard UART devices, the serial channel can be programmed to transmit or receive a 10-bit frame at communication rates of 122 to 31,250 baud; 11-bit frames can, in addition, be transmitted at 187,500 baud. The interprocessor-communications mode is similar to the two standard UART modes, but provides automatic wakeup of



8. The 8051 family's interrupt system is unusually versatile. Global and individual enables can be combined with both software and hardware-priority assignments.



9. Four 8-bit ports (white) take up most of the 8051's 40 pins. The data bus is split between port 0 and port 2.

slave processors.

Even the best μC is in trouble without development support, but the 8051 family, new as it is, can count on Intel's line of Intellec development systems. Support software, development systems and a user library stocked with 8051 utility and applications programs also support software development.

Users can write 8051 code with ASM-51, an absolute macroassembler, that converts the mnemonic assembly-level language into machine-level code. Furthermore, CONV-51, a conversion program that automatically converts 8048 assembly-level source code into its 8051 equivalent, eases upgrading to the new family. Hardware development gets a boost with the ICE-51 in-circuit emulator module, designed to speed system integration for all members of the 8051 family.

Starting with an ICE-51 emulator, a designer can begin to debug his code even before all the prototype hardware has been assembled. Then, as prototype hardware is assembled, the Intellec with ICE-51 can be used to integrate the application code with its hardware and to debug the system. ICE-51 operates on symbolic references, so the designer can refer to program sections with a symbolic name. The ICE module then takes care of all the bookkeeping, making it easy for the user to modify program sections and rearrange the program without worrying about address modifications. Other support includes workshops to be held in several key locations.□

An Introduction to the Intel MCS-51® Single-Chip Microcomputer Family

Contents

1. **INTRODUCTION** 9-42
   - Family Overview 9-42
   - Microcomputer Background Concepts 9-43

2. **ARCHITECTURE AND ORGANIZATION** 9-45
   - Central Processing Unit 9-46
   - Memory Spaces 9-49
   - Input/Output Ports 9-50
   - Special Peripheral Functions 9-51

3. **INSTRUCTION SET AND ADDRESSING MODES** 9-55
   - Data Addressing Modes 9-55
   - Addressing Mode Combinations 9-58
   - Advantages of Symbolic Addressing 9-58
   - Arithmetic Instruction Usage 9-59
   - Multiplication and Division 9-60
   - Logical Byte Operations 9-60
   - Program Control 9-61
   - Operate-and-Branch Instructions 9-62
   - Stack Operations 9-62
   - Table Look-Up Instructions 9-63

4. **BOOLEAN PROCESSING INSTRUCTIONS** 9-65
   - Direct Bit Addressing 9-65
   - Bit Manipulation Instructions 9-65
   - Solving Combinatorial Logic Equations 9-66

5. **ON-CHIP PERIPHERAL FUNCTIONS** 9-68
   - I/O Ports 9-68
   - Serial Port and Timer 9-69

6. **SUMMARY** 9-70

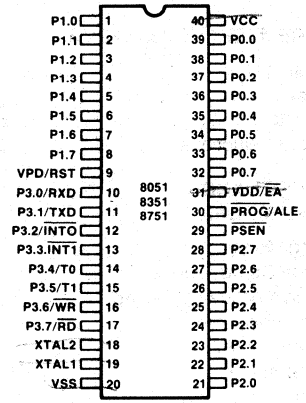


Figure 1a. 8051 Microcomputer Pinout Diagram

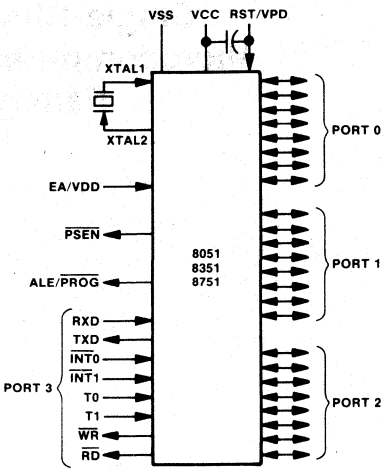


Figure 1b. 8051 Microcomputer Logic Symbol

1. INTRODUCTION

In 1976 Intel introduced the MCS-48™ family, consisting of the 8048, 8748, and 8035 microcomputers. These parts marked the first time a complete microcomputer system, including an eight-bit CPU, 1024 8-bit words of ROM or EPROM program memory, 64 words of data memory, I/O ports and an eight-bit timer/counter could be integrated onto a single silicon chip. Depending only on the program memory contents, one chip could control a limitless variety of products, ranging from appliances or automobile engines to text or data processing equipment. Follow-on products stretched the MCS-48™ architecture in several directions: the 8049 and 8039 doubled the amount of on-chip memory and ran 83% faster; the 8021 reduced costs by executing a subset of the 8048 instructions with a somewhat slower clock; and the 8022 put a unique two-channel 8-bit analog-to-digital converter on the same NMOS chip as the computer, letting the chip interface directly with analog transducers.

Now three new high-performance single-chip microcomputers—the Intel® 8051, 8751, and 8031—extend the advantages of Integrated Electronics to whole new product areas. Thanks to Intel's new HMOS technology, the MCS-51™ family provides four times the program memory and twice the data memory as the 8048 on a single chip. New I/O and peripheral capabilities both increase the range of applicability and reduce total system cost. Depending on the use, processing throughput increases by two and one-half to ten times.

This Application Note is intended to introduce the reader to the MCS-51™ architecture and features. While it does not assume intimacy with the MCS-48™ product line on the part of the reader, he/she should be familiar with some microprocessor (preferably Intel's, of course) or have a background in computer programming and digital logic.

Family Overview

Pinout diagrams for the 8051, 8751, and 8031 are shown in Figure 1. The devices include the following features:

- Single-supply 5 volt operation using HMOS technology.
- 4096 bytes program memory on-chip (not on 8031).
- 128 bytes data memory on-chip.
- Four register banks.
- 128 User-defined software flags.
- 64 Kilobytes each program and external RAM addressability.
- One microsecond instruction cycle with 12 MHz crystal.
- 32 bidirectional I/O lines organized as four 8-bit ports (16 lines on 8031).
- Multiple mode, high-speed programmable Serial Port.
- Two multiple mode, 16-bit Timer/Counters.
- Two-level prioritized interrupt structure.
- Full depth stack for subroutine return linkage and data storage.
- Augmented MCS-48™ instruction set.
- Direct Byte and Bit addressability.
- Binary or Decimal arithmetic.
- Signed-overflow detection and parity computation.
- Hardware Multiple and Divide in 4 μsec.
- Integrated Boolean Processor for control applications.
- Upwardly compatible with existing 8048 software.

AFN-01502A-04

All three devices come in a standard 40-pin Dual In-Line Package, with the same pin-out, the same timing, and the same electrical characteristics. The primary difference between the three is the on-chip program memory—different types are offered to satisfy differing user requirements.

The 8751 provides 4K bytes of ultraviolet-Erasable, Programmable Read Only Memory (EPROM) for program development, prototyping, and limited production runs. (By convention, 1K means $2^{10} = 1024$. 1k—with a lower case "k"—equals $10^3 = 1000$.) This part may be individually programmed for a specific application using Intel's Universal PROM Programmer (UPP). If software bugs are detected or design specifications change the same part may be "erased" in a matter of minutes by exposure to ultraviolet light and reprogrammed with the modified code. This cycle may be repeated indefinitely during the design and development phase.

The final version of the software must be programmed into a large number of production parts. The 8051 has 4K bytes of ROM which are mask-programmed with the customer's order when the chip is built. This part is considerably less expensive, but cannot be erased or altered after fabrication.

The 8031 does not have any program memory on-chip, but may be used with up to 64K bytes of external standard or multiplexed ROMs, PROMs, or EPROMs. The 8031 fits well in applications requiring significantly larger or smaller amounts of memory than the 4K bytes provided by its two siblings.

(The 8051 and 8751 automatically access external program memory for all addresses greater than the 4096 bytes on-chip. The External Access input is an override for all internal program memory—the 8051 and 8751 will each emulate an 8031 when pin 31 is low.)

Throughout this Note, "8051" is used as a generic term. Unless specifically stated otherwise, the point applies equally to all three components. Table 1 summarizes the quantitative differences between the members of the MCS-48™ and MCS-51™ families.

The remainder of this Note discusses the various MCS-51™ features and how they can be used. Software and/or hardware application examples illustrate many of the concepts. Several isolated tasks (rather than one complete system design example) are presented in the hope that some of them will apply to the reader's experiences or needs.

A document this short cannot detail all of a computer system's capabilities. By no means will all the 8051 instructions be demonstrated; the intent is to stress new or unique MCS-51™ operations and instructions generally used in conjunction with each other. For additional hardware information refer to the Intel **MCS-51™ Family User's Manual,** publication number 121517. The assembly language and use of ASM51, the MCS-51™ assembler, are further described in the **MCS-51™ Macro Assembler User's Guide,** publication number 9800937.

The next section reviews some of the basic concepts of microcomputer design and use. Readers familiar with the 8048 may wish to skim through this section or skip directly to the next, "ARCHITECTURE AND ORGANIZATION."

Microcomputer Background Concepts

Most digital computers use the binary (base 2) number system internally. All variables, constants, alphanumeric characters, program statements, etc., are represented by groups of binary digits ("bits"), each of which has the value 0 or 1. Computers are classified by how many bits they can move or process at a time.

The MCS-51™ microcomputers contain an eight-bit central processing unit (CPU). Most operations process variables eight bits wide. All internal RAM and ROM, and virtually all other registers are also eight bits wide. An eight-bit ("byte") variable (shown in Figure 2) may assume one of $2^8 = 256$ distinct values, which usually represent integers between 0 and 255. Other types of numbers, instructions, and so forth are represented by one or more bytes using certain conventions.

For example, to represent positive and negative values, the most significant bit (D7) indicates the sign of the other seven bits—0 if positive, 1 if negative—allowing integer variables between -128 and +127. For integers with extremely large magnitudes, several bytes are manipulated together as "multiple precision" signed or unsigned integers—16, 24, or more bits wide.

Table 1. Features of Intel's Single-Chip Microcomputers

EPROM Program Memory	ROM Program Memory	External Program Memory	Program Memory (Int/Max)	Data Memory (Bytes)	Instr. Cycle Time	Input/ Output Pins	Interrupt Sources	Reg. Banks
—	8021	—	1K/1K	64	8.4 μSec	21	0	1
—	8022	—	2K/2K	64	8.4 μSec	28	2	1
8748	8048	8035	1K/4K	64	2.5 μSec	27	2	2
—	8049	8039	2K/4K	128	1.36 μSec	27	2	2
8751	8051	8031	4K/64K	128	1.0 μSec	32	5	4

AFN-01502A-05

The letters "MCS" have traditionally indicated a system or family of compatible Intel® microcomputer components, including CPUs, memories, clock generators, I/O expanders, and so forth. The numerical suffix indicates the microprocessor or microcomputer which serves as the cornerstone of the family. Microcomputers in the MCS-48™ family currently include the 8048-series (8035, 8048, & 8748), the 8049-series (8039 & 8049), and the 8021 and 8022; the family also includes the 8243, an I/O expander compatible with each of the microcomputers. Each computer's CPU is derived from the 8048, with essentially the same architecture, addressing modes, and instruction set, and a single assembler (ASM48) serves each.

The first members of the MCS-51™ family are the 8051, 8751, and 8031. The architecture of the 8051-series, while derived from the 8048, is not strictly compatible; there are more addressing modes, more instructions, larger address spaces, and a few other hardware differences. In this Application Note the letters "MCS-51" are used when referring to *architectural* features of the 8051-series—features which would be included on possible future microcomputers based on the 8051 CPU. Such products could have different amounts of memory (as in the 8048/8049) or different peripheral functions (as in the 8021 and 8022) while leaving the CPU and instruction set intact. ASM51 is the assembler used by all microcomputers in the 8051 family.

Two digit decimal numbers may be "packed" in an eight-bit value, using four bits for the binary code of each digit. This is called Binary-Coded Decimal (BCD) representation, and is often used internally in programs which interact heavily with human beings.

Alphanumeric characters (letters, numbers, punctuation marks, etc.) are often represented using the American Standard Code for Information Interchange (ASCII) convention. Each character is associated with a unique seven-bit binary number. Thus one byte may represent a single character, and a word or sequence of letters may be represented by a series (or "string") of bytes. Since the ASCII code only uses 128 characters, the most significant bit of the byte is not needed to distinguish between characters. Often D7 is set to 0 for all characters. In some coding schemes, D7 is used to indicate the "parity" of the other seven bits—set or cleared as necessary to ensure that the total number of "1" bits in the eight-bit code is even ("even parity") or odd ("odd parity"). The 8051 includes hardware to compute parity when it is needed.

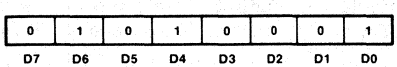


Figure 2. Representation of Bits Within an Eight-Bit "Byte" (Value shown = 01010001 Binary = 81 decimal).

A computer program consists of an ordered sequence of specific, simple steps to be executed by the CPU one-at-a-time. The method or sequence of steps used collectively to solve the user's application is called an "algorithm."

The program is stored inside the computer as a sequence of binary numbers, where each number corresponds to one of the basic operations ("opcodes") which the CPU is capable of executing. In the 8051, each program memory location is one byte. A complete instruction consists of a sequence of one or more bytes, where the first defines the operation to be executed and additional bytes (if needed) hold additional information, such as data values or variable addresses. No instruction is longer than three bytes.

The way in which binary opcodes and modifier bytes are assigned to the CPU's operations is called the computer's "machine language." Writing a program directly in machine language is time-consuming and tedious. Human beings think in words and concepts rather than encoded numbers, so each CPU operation and resource is given a name and standard abbreviation ("mnemonic"). Programs are more easily discussed using these standard mnemonics, or "assembly language," and may be typed into an Intel® Intellec® 800 or Series II® microcomputer development system in this form. The development system can mechanically translate the program from assembly language "source" form to machine language "object" code using a program called an "assembler." The MCS-51™ assembler is called ASM51.

There are several important differences between a computer's machine language and the assembly language used as a tool to represent it. The machine language or instruction set is the set of operations which the CPU can perform while a program is executing ("at run-time"), and is strictly determined by the microcomputer hardware design.

The assembly language is a standard (though more-or-less arbitrary) set of symbols including the instruction set mnemonics, but with additional features which further simplify the program design process. For example, ASM51 has controls for creating and formatting a program listing, and a number of directives for allocating variable storage and inserting arbitrary bytes of data into the object code for creating tables of constants.

AFN-01502A-06

In addition, ASM51 can perform sophisticated mathematical operations, computing addresses or evaluating arithmetic expressions to relieve the programmer from this drudgery. However, these calculations can only use information known at "assembly time."

For example, the 8051 performs arithmetic calculations at run-time, eight bits at a time. ASM51 can do similar operations 16 bits at a time. The 8051 can only do one simple step per instruction, while ASM51 can perform complex calculations in each line of source code. However, the operations performed by the assembler may only use parameter values fixed at assembly-time, not variables whose values are unknown until program execution begins.

For example, when the assembly language source line,

```
ADD     A,#(LOOP_COUNT + 1) * 3
```

is *assembled*, ASM51 will find the value of the previously-defined constant "LOOP_COUNT" in an internal symbol table, increment the value, multiply the sum by three, and (assuming it is between -256 and 255 inclusive) truncate the product to eight bits. When this instruction is *executed*, the 8051 ALU will just add that resulting constant to the accumulator.

Some similar differences exist to distinguish number system ("radix") specifications. The 8051 does all computations in binary (though there are provisions for then converting the result to decimal form). In the course of writing a program, though, it may be more convenient to specify constants using some other radix, such as base 10. On other occasions, it is desirable to specify the ASCII code for some character or string of characters without refering to tables. ASM51 allows several representations for constants, which are converted to binary as each instruction is assembled.

For example, binary numbers are represented in the assembly language by a series of ones and zeros (naturally), followed by the letter "B" (for Binary); octal numbers as a series of octal digits (0-7) followed by the letter "O" (for Octal) or "Q" (which doesn't stand for anything, but *looks* sort of like an "O" and is less likely to be confused with a zero).

Hexadecimal numbers are represented by a series of hexadecimal digits (0-9,A-F), followed by (you guessed it) the letter "H." A "hex" number must begin with a decimal digit; otherwise it would look like a user-defined symbol (to be discussed later). A "dummy" leading zero may be inserted before the first digit to meet this constraint. The character string "BACH" could be a legal label for a Baroque music synthesis routine; the string "0BACH" is the hexadecimal constant BAC_{16}. This is a case where adding 0 makes a big difference.

Decimal numbers are represented by a sequence of decimal digits, optionally followed by a "D." If a number has no suffix, it is assumed to be decimal—so it had better not contain any non-decimal digits. "0BAC" is not a legal representation for anything.

When an ASCII code is needed in a program, enclose the desired character between two apostrophes (as in '#') and the assembler will convert it to the appropriate code (in this case 23H). A string of characters between apostrophes is translated into a series of constants; 'BACH' becomes 42H, 41H, 43H, 48H.

These same conventions are used throughout the associated Intel documentation. Table 2 illustrates some of the different number formats.

2. ARCHITECTURE AND ORGANIZATION

Figure 3 blocks out the MCS-51™ internal organization. Each microcomputer combines a Central Processing Unit, two kinds of memory (data RAM plus program ROM or EPROM), Input/Output ports, and the mode,

Table 2. Notations Used to Represent Numbers

Bit Pattern	Binary	Octal	Hexa-Decimal	Decimal	Signed Decimal
0 0 0 0 0 0 0 0	0B	0Q	00H	0	0
0 0 0 0 0 0 0 1	1B	1Q	01H	1	+1
.....	..	...	...	..	....
0 0 0 0 0 1 1 1	111B	7Q	07H	7	+7
0 0 0 0 1 0 0 0	1000B	10Q	08H	8	+8
0 0 0 0 1 0 0 1	1001B	11Q	09H	9	+9
0 0 0 0 1 0 1 0	1010B	12Q	0AH	10	+10
.....	..	...	...	..	....
0 0 0 0 1 1 1 1	1111B	17Q	0FH	15	+15
0 0 0 1 0 0 0 0	10000B	20Q	10H	16	+16
.....	..	...	...	..	....
0 1 1 1 1 1 1 1	1111111B	177Q	7FH	127	+127
1 0 0 0 0 0 0 0	10000000B	200Q	80H	128	-128
1 0 0 0 0 0 0 1	10000001B	201Q	81H	129	-127
.....	.....	...	...	...	....
1 1 1 1 1 1 1 0	11111110B	376Q	0FEH	254	-2
1 1 1 1 1 1 1 1	11111111B	377Q	0FFH	255	-1

AFN-01502A-07

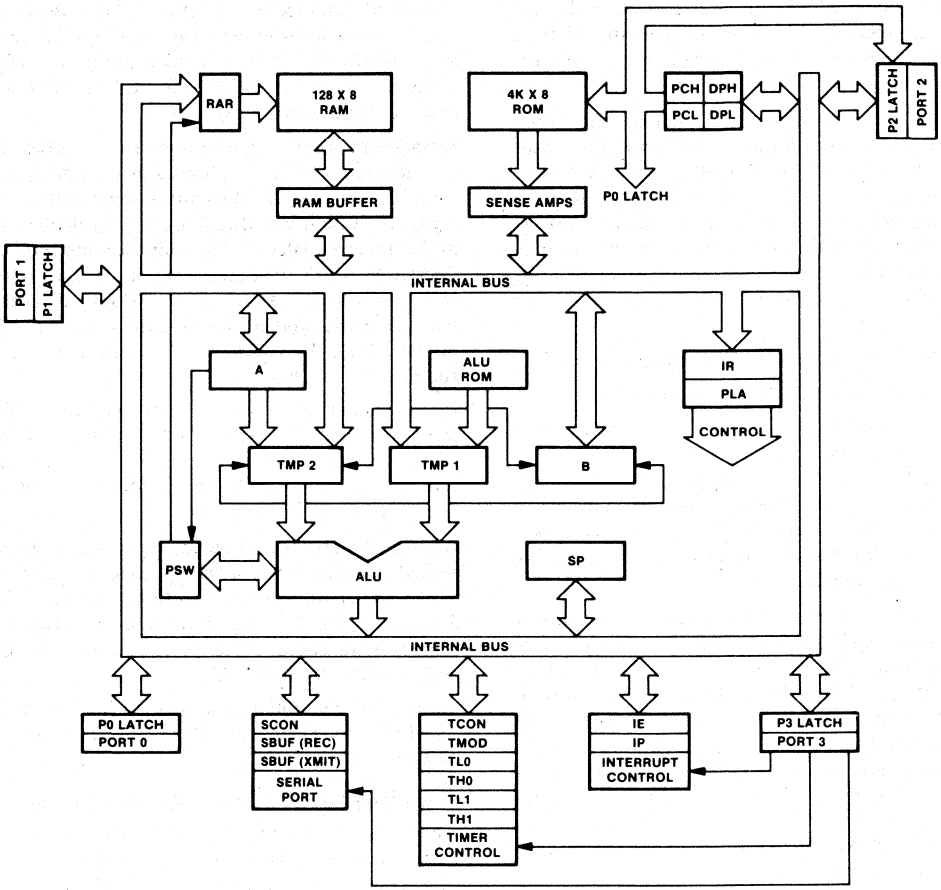


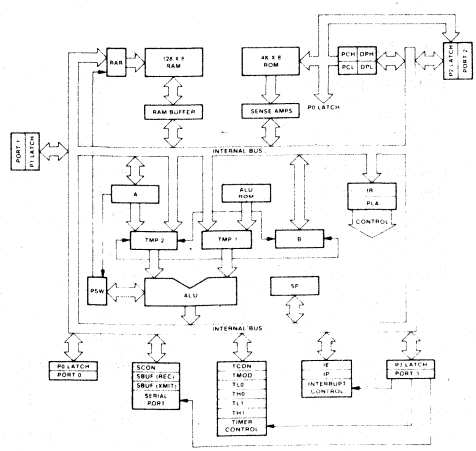
Figure 3. Block Diagram of 8051 Internal Structure

status, and data registers and random logic needed for a variety of peripheral functions. These elements communicate through an eight-bit data bus which runs throughout the chip, somewhat akin to indoor plumbing. This bus is buffered to the outside world through an I/O port when memory or I/O expansion is desired.

Let's summarize what each block does; later chapters dig into the CPU's instruction set and the peripheral registers in much greater detail.

Central Processing Unit

The CPU is the "brains" of the microcomputer, reading the user's program and executing the instructions stored therein. Its primary elements are an eight-bit Arithmetic/Logic Unit with associated registers A, B, PSW, and SP, and the sixteen-bit Program Counter and "Data Pointer" registers.



AFN-01502A-08

Arithmetic Logic Unit

The ALU can perform (as the name implies) arithmetic and logic functions on eight-bit variables. The former include basic addition, subtraction, multiplication, and division; the latter include the logical operations AND, OR, and Exclusive-OR, as well as rotate, clear, complement, and so forth. The ALU also makes conditional branching decisions, and provides data paths and temporary registers used for data transfers within the system. Other instructions are built up from these primitive functions: the addition capability can increment registers or automatically compute program destination addresses; subtraction is also used in decrementing or comparing the magnitude of two variables.

These primitive operations are automatically cascaded and combined with dedicated logic to build complex instructions such as incrementing a sixteen-bit register pair. To execute one form of the compare instruction, for example, the 8051 increments the program counter three times, reads three bytes of program memory, computes a register address with logical operations, reads internal data memory twice, makes an arithmetic comparison of two variables, computes a sixteen-bit destination address, and decides whether or not to make a branch—all in two microseconds!

An important and unique feature of the MCS-51 architecture is that the ALU can also manipulate one-bit as well as eight-bit data types. Individual bits may be set, cleared, or complemented, moved, tested, and used in logic computations. While support for a more primitive data type may initially seem a step backwards in an era of increasing word length, it makes the 8051 especially well suited for controller-type applications. Such algorithms *inherently* involve Boolean (true/false) input and output variables, which were heretofore difficult to implement with standard microprocessors. These features are collectively referred to as the MCS-51™ "Boolean Processor," and are described in the so-named chapter to come.

Thanks to this powerful ALU, the 8051 instruction set fares well at both real-time control and data intensive algorithms. A total of 51 separate operations move and manipulate three data types: Boolean (1-bit), byte (8-bit), and address (16-bit). All told, there are eleven addressing modes—seven for data, four for program sequence control (though only eight are used by more than just a few specialized instructions). Most operations allow several addressing modes, bringing the total number of instructions (operation/addressing mode combinations) to 111, encompassing 255 of the 256 possible eight-bit instruction opcodes.

Instruction Set Overview

Table 4 lists these 111 instructions classified into five groups:

- Arithmetic Operations
- Logical Operations for Byte Variables
- Data Transfer Instructions
- Boolean Variable Manipulation
- Program Branching and Machine Control

MCS-48™ programmers perusing Table 4 will notice the absence of special categories for Input/Output, Timer/Counter, or Control instructions. These functions are all still provided (and indeed many new functions are added), but as special cases of more generalized operations in other categories. To explicitly list all the useful instructions involving I/O and peripheral registers would require a table approximately four times as long.

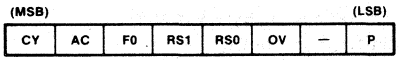
Observant readers will also notice that all of the 8048's page-oriented instructions (conditional jumps, JMPP, MOVP, MOVP3) have been replaced with corresponding but non-paged instructions. The 8051 instruction set is entirely *non*-page-oriented. The MCS-48™ "MOVP" instruction replacement and all conditional jump instructions operate relative to the program counter, with the actual jump address computed by the CPU during instruction execution. The "MOVP3" and "JMPP" replacements are now made relative to another sixteen-bit register, which allows the effective destination to be anywhere in the program memory space, regardless of where the instruction itself is located. There are even three-byte jump and call instructions allowing the destination to be *anywhere* in the 64K program address space.

The instruction set is designed to make programs efficient both in terms of code size and execution speed. No instruction requires more than three bytes of program memory, with the majority requiring only one or two bytes. Virtually all instructions execute in either one or two instruction cycles—one or two microseconds with a 12-MHz crystal—with the sole exceptions (multiply and divide) completing in four cycles.

Many instructions such as arithmetic and logical functions or program control, provide both a short and a long form for the same operation, allowing the programmer to optimize the code produced for a specific application. The 8051 usually fetches two instruction bytes per instruction cycle, so using a shorter form can lead to faster execution as well.

For example, any byte of RAM may be loaded with a constant with a three-byte, two-cycle instruction, but the commonly used "working registers" in RAM may be initialized in one cycle with a two-byte form. Any bit anywhere on the chip may be set, cleared, or complemented by a single three-byte logical instruction using two cycles. But critical control bits, I/O pins, and software flags may be controlled by two-byte, single cycle instructions. While three-byte jumps and calls can "go anywhere" in program memory, nearby sections of code may be reached by shorter relative or absolute versions.

AFN-01502A-09



Symbol	Position	Name and Significance
CY	PSW.7	Carry flag. Set/cleared by hardware or software during certain arithmetic and logical instructions.
AC	PSW.6	Auxiliary Carry flag. Set/cleared by hardware during addition or subtraction instructions to indicate carry or borrow out of bit 3.
F0	PSW.5	Flag 0 Set/cleared/tested by software as a user-defined status flag.
RS1	PSW.4	Register bank Select control bits 1 & 0. Set/cleared by software to determine working register bank (see Note).
RS	PSW.3	
OV	PSW.2	Overflow flag. Set/cleared by hardware during arithmetic instructions to indicate overflow conditions.
—	PSW.1	(reserved)
P	PSW.0	Parity flag. Set/cleared by hardware each instruction cycle to indicate an odd/even number of "one" bits in the accumulator, i.e., even parity.

Note— the contents of (RS1, RS0) enable the working register banks as follows:

(0,0)—Bank 0	(00H-07H)
(0,1)—Bank 1	(08H-0FH)
(1,0)—Bank 2	(10H-17H)
(1,1)—Bank 3	(18H-1FH)

Figure 4. PSW—Program Status Word Organization

A significant side benefit of an instruction set more powerful than those of previous single-chip microcomputers is that it is easier to generate applications-oriented software. Generalized addressing modes for byte and bit instructions reduce the number of source code lines written and debugged for a given application. This leads in turn to proportionately lower software costs, greater reliability, and faster design cycles.

Accumulator and PSW

The 8051, like its 8048 predecessor, is primarily an accumulator-based architecture: an eight-bit register called the accumulator ("A") holds a source operand and receives the result of the arithmetic instructions (addition, subtraction, multiplication, and division). The accumulator can be the source or destination for logical operations and a number of special data movement instructions, including table look-ups and external RAM expansion. Several functions apply exclusively to the accumulator: rotates, parity computation, testing for zero, and so on.

Many instructions implicitly or explicitly affect (or are affected by) several status flags, which are grouped together to form the Program Status Word shown in Figure 4.

(The period within entries under the Position column is called the "dot operator," and indicates a particular bit position within an eight-bit byte. "PSW.5" specifies bit 5 of the PSW. Both the documentation and ASM51 use this notation.)

The most "active" status bit is called the carry flag (abbreviated "C"). This bit makes possible multiple precision arithmetic operations including addition, subtraction, and rotates. The carry also serves as a "Boolean accumulator" for one-bit logical operations and bit manipulation instructions. The overflow flag (OV) detects when arithmetic overflow occurs on signed integer operands, making two's complement arithmetic possible. The parity flag (P) is updated after every instruction cycle with the even-parity of the accumulator contents.

The CPU does not control the two register-bank select bits, RS1 and RS0. Rather, they are manipulated by software to enable one of the four register banks. The usage of the PSW flags is demonstrated in the Instruction Set chapter of this Note.

Even though the architecture is accumulator-based, provisions have been made to bypass the accumulator in common instruction situations. Data may be moved from any location on-chip to any register, address, or indirect address (and vice versa), any register may be loaded with a constant, etc., all without affecting the accumulator. Logical operations may be performed against registers or variables to alter fields of bits—without using or affecting the accumulator. Variables may be incremented, decremented, or tested without using the accumulator. Flags and control bits may be manipulated and tested without affecting anything else.

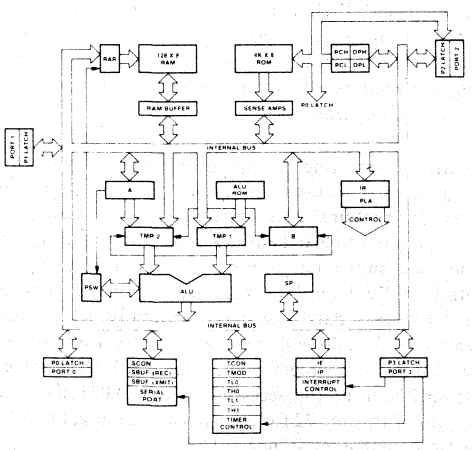
Other CPU Registers

A special eight-bit register ("B") serves in the execution of the multiply and divide instructions. This register is used in conjunction with the accumulator as the second input operand and to return eight-bits of the result.

The MCS-51 family processors include a hardware stack within internal RAM, useful for subroutine linkage,

passing parameters between routines, temporary variable storage, or saving status during interrupt service routines. The Stack Pointer (SP) is an eight-bit pointer register which indicates the address of the last byte pushed onto the stack. The stack pointer is automatically incremented or decremented on all push or pop instructions and all subroutine calls and returns. In theory, the stack in the 8051 may be up to a full 128 bytes deep. (In practice, even simple programs would use a handful of RAM locations for pointers, variables, and so forth—reducing the stack depth by that number.) The stack pointer defaults to 7 on reset, so that the stack will start growing up from location 8, just like in the 8048. By altering the pointer contents the stack may be relocated anywhere within internal RAM.

Finally, a 16-bit register called the data pointer (DPTR) serves as a base register in indirect jumps, table look-up instructions, and external data transfers. The high- and low-order halves of the data pointer may be manipulated as separate registers (DPH and DPL, respectively) or together using special instructions to load or increment all sixteen bits. Unlike the 8048, look-up tables can therefore start anywhere in program memory and be of arbitrary length.



Memory Spaces

Program memory is separate and distinct from data memory. Each memory type has a different addressing mechanism, different control signals, and a different function.

The program memory array (ROM or EPROM), like an elephant, is extremely large and never forgets information, even when power is removed. Program memory is used for information needed each time power is applied: initialization values, calibration constants, keyboard layout tables, etc., as well as the program itself. The program memory has a sixteen-bit address bus; its elements are addressed using the Program Counter or instructions which generate a sixteen-bit address.

To stretch our analogy just a bit, data memory is like a mouse: it is smaller and therefore quicker than program memory, and it goes into a random state when electrical power is applied. On-chip data RAM is used for variables which are determined or may change while the program is running.

A computer spends most of its time manipulating variables, not constants, and a relatively small number of variables at that. Since eight-bits is more than sufficient to uniquely address 128 RAM locations, the on-chip RAM address register is only one byte wide. In contrast to the program memory, data memory accesses need a single eight-bit value—a constant or another variable—to specify a unique location. Since this is the basic width of the ALU and the different memory types, those resources can be used by the addressing mechanisms, contributing greatly to the computer's operating efficiency.

The partitioning of program and data memory is extended to off-chip memory expansion. Each may be added independently, and each uses the same address and data busses, but with different control signals. External program memory is gated onto the external data bus by the $\overline{\text{PSEN}}$ (Program Store Enable) control output, pin 29. External data memory is read onto the bus by the $\overline{\text{RD}}$ output, pin 17, and written with data supplied from the microcomputer by the $\overline{\text{WR}}$ output, pin 16. (There is no control pin to write external program ROM, which is by definition Read Only.) While both types may be expanded to up to 64K bytes, the external data memory may optionally be expanded in 256 byte "pages" to preserve the use of P2 as an I/O port. This is useful with a relatively small expansion RAM (such as the Intel® 8155) or for addressing external peripherals.

Single-chip controller programs are finalized during the project design cycle, and are not modified after production. Intel's single-chip microcomputers are not "von Neumann" architectures common among main-frame and mini-computer systems: the MCS-51™ processor *data* memory—on-chip and external—may *not* be used for program code. Just as there is no write-control signal for program memory, there is no way for the CPU to execute instructions out of RAM. In return, this concession allows an architecture optimized for efficient controller applications: a large, fixed program located in ROM, a hundred or so variables in RAM, and different methods for efficiently addressing each.

(Von Neumann machines are helpful for software development and debug. An 8051 system could be modified to have a single off-chip memory space by gating together the two memory-read controls ($\overline{\text{PSEN}}$ and $\overline{\text{RD}}$) with a two-input AND gate (Figure 5). The CPU could then write data into the common memory array using $\overline{\text{WR}}$ and

AFN-01502A-11

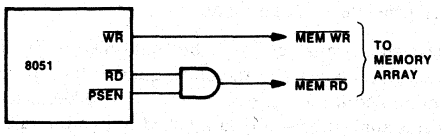


Figure 5. Combining External Program and Data Memory Arrays

external data transfer instructions, and read instructions or data with the AND gate output and data transfer or program memory look-up instructions.)

In addition to the memory arrays, there is (yet) another (albeit sparsely populated) physical address space. Connected to the internal data bus are a score of special-purpose eight-bit registers scattered throughout the chip. Some of these—B, SP, PSW, DPH, and DPL—have been discussed above. Others—I/O ports and peripheral function registers—will be introduced in the following sections. Collectively, these registers are designated as the "special-function register" address space. Even the accumulator is assigned a spot in the special-function register address space for additional flexibility and uniformity.

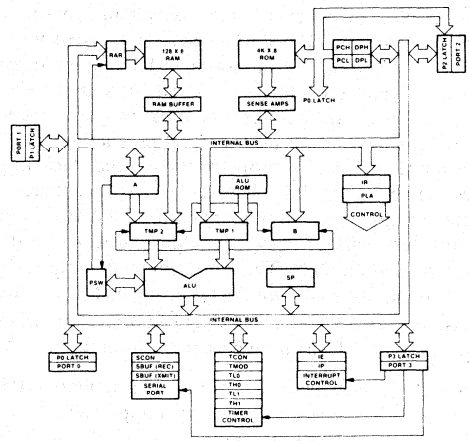
Thus, the MCS-51™ architecture supports several distinct "physical" address spaces, functionally separated at the hardware level by different addressing mechanisms, read and write control signals, or both:

- On-chip program memory;
- On-chip data memory;
- Off-chip program memory;
- Off-chip data memory;
- On-chip special-function registers.

What the *programmer sees,* though, are "logical" address spaces. For example, as far as the programmer is concerned, there is only one type of program memory, 64K bytes in length. The fact that it is formed by combining on- and off-chip arrays (split 4K/60K on the 8051 and 8751) is "invisible" to the programmer; the CPU automatically fetches each byte from the appropriate array, based on its address.

(Presumably, future microcomputers based on the MCS-51™ architecture may have a different physical split, with more or less of the 64K total implemented on-chip. Using the MCS-48™ family as a precedent, the 8048's 4K potential program address space was split 1K/3K between on- and off-chip arrays; the 8049's was split 2K/2K.)

Why go into such tedious details about address spaces? The logical addressing modes are described in the Instruction Set chapter in terms of physical address spaces. Understanding their differences now will pay off in understanding and using the chips later.



Input/Output Ports

The MCS-51™ I/O port structure is extremely versatile. The 8051 and 8751 each have 32 I/O pins configured as four eight-bit parallel ports (P0, P1, P2, and P3). Each pin will input or output data (or both) under software control, and each may be referenced by a wide repertoire of byte and bit operations.

In various operating or expansion modes, some of these I/O pins are also used for special input or output functions. Instructions which access external memory use Port 0 as a multiplexed address/data bus: at the beginning of an external memory cycle eight bits of the address are output on P0; later data is transferred on the same eight pins. External data transfer instructions which supply a sixteen-bit address, and any instruction accessing external program memory, output the high-order eight bits on P2 during the access cycle. (The 8031 *always* uses the pins of P0 and P2 for external addressing, but P1 and P3 are available for standard I/O.)

The eight pins of Port 3 (P3) each have a special function. Two external interrupts, two counter inputs, two serial data lines, and two timing control strobes use pins of P3 as described in Figure 6. Port 3 pins corresponding to functions not used are available for conventional I/O.

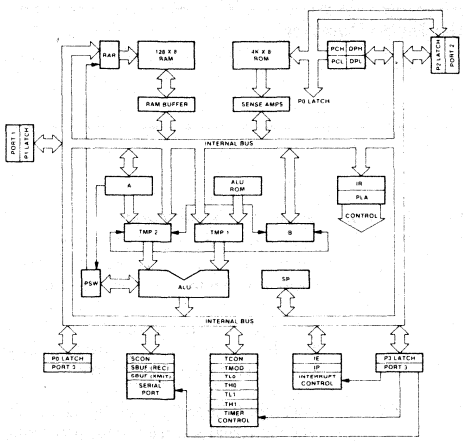
Even within a single port, I/O functions may be combined in many ways: input and output may be performed using different pins at the same time, or the same pins at different times; in parallel in some cases, and in serial in others; as test pins, or (in the case of Port 3) as additional special functions.

AFN-01502A-12

(MSB)							(LSB)
RD	WR	T1	T0	INT1	INT0	TXD	RXD

Symbol	Position	Name and Significance
RD	P3.7	Read data control output. Active low pulse generated by hardware when external data memory is read.
WR	P3.6	Write data control output. Active low pulse generated by hardware when external data memory is written.
T1	P3.5	Timer/counter 1 external input or test pin.
T0	P3.4	Timer/counter 0 external input or test pin.
INT1	P3.3	Interrupt 1 input pin. Low-level or falling-edge triggered.
INT0	P3.2	Interrupt 0 input pin. Low-level or falling-edge triggered.
TXD	P3.1	Transmit Data pin for serial port in UART mode. Clock output in shift register mode.
RXD	P3.0	Receive Data pin for serial port in UART mode. Data I/O pin in shift register mode.

Figure 6. P3—Alternate Special Functions of Port 3



Special Peripheral Functions

There are a few special needs common among control-oriented computer systems:

- keeping track of elapsed real-time;
- maintaining a count of signal transitions;
- measuring the precise width of input pulses;
- communicating with other systems or people;
- closely monitoring asynchronous external events.

Until now, microprocessor systems needed peripheral chips such as timer/counters, USARTs, or interrupt controllers to meet these needs. The 8051 integrates all of these capabilities on-chip!

Timer/Counters

There are two sixteen-bit multiple-mode Timer/Counters on the 8051, each consisting of a "High" byte (corresponding to the 8048 "T" register) and a low byte (similar to the 8048 prescaler, with the additional flexibility of being software-accessible). These registers are called, naturally enough, TH0, TL0, TH1, and TL1. Each pair may be independently software programmed to any of a dozen modes with a mode register designated TMOD (Figure 7), and controlled with register TCON (Figure 8).

The timer modes can be used to measure time intervals, determine pulse widths, or initiate events, with one-microsecond resolution, up to a maximum interval of 65,536 instruction cycles (over 65 milliseconds). Longer delays may easily be accumulated through software. Configured as a counter, the same hardware will accumulate external events at frequencies from D.C. to 500 KHz, with up to sixteen bits of precision.

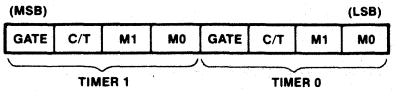
Serial Port Interface

Each microcomputer contains a high-speed, full-duplex, serial port which is software programmable to function in four basic modes: shift-register I/O expander, 8-bit UART, 9-bit UART, or interprocessor communications link. The UART modes will interface with standard I/O devices (e.g. CRTs, teletypewriters, or modems) at data rates from 122 baud to 31 kilobaud. Replacing the standard 12 MHz crystal with a 10.7 MHz crystal allows 110 baud. Even or odd parity (if desired) can be included with simple bit-handling software routines. Inter-processor communications in distributed systems takes place at 187 kilobaud with hardware for automatic address/data message recognition. Simple TTL or CMOS shift registers provide low-cost I/O expansion at a super-fast 1 Megabaud. The serial port operating modes are controlled by the contents of register SCON (Figure 9).

Interrupt Capability and Control

(Interrupt capability is generally considered a CPU function. It is being introduced here since, from an applications point of view, interrupts relate more closely to peripheral and system interfacing.)

AFN-01502A-13

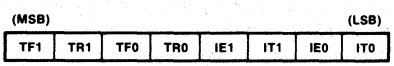


GATE — Gating control. When set, Timer/counter "x" is enabled only while "INTx" pin is high and "TRx" control bit is set. When cleared, timer/counter is enabled whenever "TRx" control bit is set.

C/T — Timer or Counter Selector. Cleared for Timer operation (input from internal system clock). Set for Counter operation (input from "Tx" input pin).

M1	M0	Operating Mode
0	0	MCS-48 Timer. "TLx" serves as five-bit prescaler.
0	1	16-bit timer/counter. "THx" and "TLx" are cascaded; there is no prescaler.
1	0	8-bit auto-reload timer/counter. "THx" holds a value which is to be reloaded into "TLx" each time it overflows.
1	1	(Timer 0) TL0 is an eight-bit timer/counter controlled by the standard Timer 0 control bits. TH0 is an eight-bit timer only controlled by Timer 1 control bits.
1	1	(Timer 1) Timer/counter 1 stopped.

Figure 7. TMOD—Timer/Counter Mode Register



Symbol	Position	Name and Significance
TF1	TCON.7	Timer 1 overflow Flag. Set by hardware on timer/counter overflow. Cleared when interrupt processed.
TR1	TCON.6	Timer 1 Run control bit. Set/cleared by software to turn timer/counter on/off.
TF0	TCON.5	Timer 0 overflow Flag. Set by hardware on timer/counter overflow. Cleared when interrupt processed.
TR0	TCON.4	Timer 0 Run control bit. Set/cleared by software to turn timer/counter on/off.
IE1	TCON.3	Interrupt 1 Edge flag. Set by hardware when external interrupt edge detected. Cleared when interrupt processed.
IT1	TCON.2	Interrupt 1 Type control bit. Set/cleared by software to specify falling edge/low level triggered external interrupts.
IE0	TCON.1	Interrupt 0 Edge flag. Set by hardware when external interrupt edge detected. Cleared when interrupt processed.
IT0	TCON.0	Interrupt 0 Type control bit. Set/cleared by software to specify falling edge/low level triggered external interrupts.

Figure 8. TCON—Timer/Counter Control/Status Register

AFN-01502A-14

(MSB)							(LSB)
SM0	SM1	SM2	REN	TB8	RB8	TI	RI

Symbol	Position	Name and Significance
SM0	SCON.7	Serial port Mode control bit 0. Set/cleared by software (see note).
SM1	SCON.6	Serial port Mode control bit 1. Set/cleared by software (see note).
SM2	SCON.5	Serial port Mode control bit 2. Set by software to disable reception of frames for which bit 8 is zero.
REN	SCON.4	Receiver Enable control bit. Set/cleared by software to enable/disable serial data reception.
TB8	SCON.3	Transmit Bit 8. Set/cleared by hardware to determine state of ninth data bit transmitted in 9-bit UART mode.
RB8	SCON.2	Receive Bit 8. Set/cleared by hardware to indicate state of ninth data bit received.
TI	SCON.1	Transmit Interrupt flag. Set by hardware when byte transmitted. Cleared by software after servicing.
RI	SCON.0	Received Interrupt flag. Set by hardware when byte received. Cleared by software after servicing.

Note— the state of (SM0,SM1) selects:
(0,0)—Shift register I/O expansion.
(0,1)—8 bit UART, variable data rate.
(1,0)—9 bit UART, fixed data rate.
(1,1)—9 bit UART, variable data rate.

Figure 9. SCON—Serial Port Control/Status Register

These peripheral functions allow special hardware to monitor real-time signal interfacing without bothering the CPU. For example, imagine serial data is arriving from one CRT while being transmitted to another, and one timer/counter is tallying high-speed input transitions while the other measures input pulse widths. During all of this the CPU is thinking about something else.

But how does the CPU know when a reception, transmission, count, or pulse is finished? The 8051 programmer can choose from three approaches.

TCON and SCON contain status bits set by the hardware when a timer overflows or a serial port operation is completed. The first technique reads the control register into the accumulator, tests the appropriate bit, and does a conditional branch based on the result. This "polling" scheme (typically a three-instruction sequence though additional instructions to save and restore the accumulator may sometimes be needed) will surely be familiar to programmers used to multi-chip microcomputer systems and peripheral controller chips. This process is rather cumbersome, especially when monitoring multiple peripherals.

As a second approach, the 8051 can perform a conditional branch based on the state of any control or status bit or input pin in a single instruction; a four instruction sequence could poll the four simultaneous happenings mentioned above in just eight microseconds.

Unfortunately, the CPU must still drop what it's doing to test these bits. A manager cannot do his own work well if he is continuously monitoring his subordinates; they should interrupt him (or her) only when they need attention or guidance. So it is with machines: ideally, the CPU would not have to worry about the peripherals until they require servicing. At that time, it would postpone the background task long enough to handle the appropriate device, then return to the point where it left off.

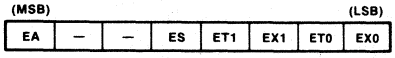
This is the basis of the third and generally optimal solution, hardware interrupts. The 8051 has five interrupt sources: one from the serial port when a transmission or reception is complete, two from the timers when overflows occur, and two from input pins INT0 and INT1. Each source may be independently enabled or disabled to allow polling on some sources or at some times, and each may be classified as high or low priority. A high priority source can interrupt a low priority service routine; the manager's boss can interrupt conferences with subordinates. These options are selected by the interrupt enable and priority control registers, IE and IP (Figures 10 and 11).

Each source has a particular program memory address associated with it (Table 3), starting at 0003H (as in the 8048) and continuing at eight-byte intervals. When an event enabled for interrupts occurs the CPU automatically executes an internal subroutine call to the corresponding address. A user subroutine starting at this location (or jumped to from this location) then performs the instructions to service that particular source. After completing the interrupt service routine, execution returns to the background program.

Table 3. 8051 Interrupt Sources and Service Vectors

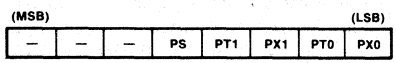
Interrupt Source	Service Routine Starting Address
(Reset)	0000H
External 0	0003H
Timer/Counter 0	000BH
External 1	0013H
Timer/Counter 1	001BH
Serial Port	0023H

AFN-01502A-15



Symbol	Position	Name and Significance
EA	IE.7	Enable All control bit. Cleared by software to disable all interrupts, independent of the state of IE.4-IE.0.
—	IE.6	(reserved)
—	IE.5	(reserved)
ES	IE.4	Enable Serial port control bit. Set/cleared by software to enable/disable interrupts from TI or RI flags.
ET1	IE.3	Enable Timer 1 control bit. Set/cleared by software to enable/disable interrupts from timer/counter 1.
EX1	IE.2	Enable External interrupt 1 control bit. Set/cleared by software to enable/disable interrupts from INT1.
ET0	IE.1	Enable Timer 0 control bit. Set/cleared by software to enable/disable interrupts from timer/counter 0
EX0	IE.0	Enable External interrupt 0 control bit. Set/cleared by software to enable/disable interrupts from INT0.

Figure 10. IE—Interrupt Enable Register



Symbol	Position	Name and Significance
—	IP.7	(reserved)
—	IP.6	(reserved)
—	IP.5	(reserved)
PS	IP.4	Serial port Priority control bit. Set/cleared by software to specify high/low priority interrupts for Serial port.
PT1	IP.3	Timer 1 Priority control bit. Set/cleared by software to specify high/low priority interrupts for timer/counter 1.
PX1	IP.2	External interrupt 1 Priority control bit. Set/cleared by software to specify high/low priority interrupts for INT1.
PT0	IP.1	Timer 0 Priority control bit. Set/cleared by software to specify high/low priority interrupts for timer/counter 0.
PX0	IP.0	External interrupt 0 Priority control bit. Set/cleared by software to specify high/low priority interrupts for INT0.

Figure 11. IP—Interrupt Priority Control Register

Table 4. MCS-51™ Instruction Set Description

ARITHMETIC OPERATIONS

Mnemonic		Description	Byte	Cyc
ADD	A,Rn	Add register to Accumulator	1	1
ADD	A,direct	Add direct byte to Accumulator	2	1
ADD	A,@Ri	Add indirect RAM to Accumulator	1	1
ADD	A,#data	Add immediate data to Accumulator	2	1
ADDC	A,Rn	Add register to Accumulator with Carry	1	1
ADDC	A,direct	Add direct byte to A with Carry flag	2	1
ADDC	A,@Ri	Add indirect RAM to A with Carry flag	1	1
ADDC	A,#data	Add immediate data to A with Carry flag	2	1
SUBB	A,Rn	Subtract register from A with Borrow	1	1
SUBB	A,direct	Subtract direct byte from A with Borrow	2	1
SUBB	A,@Ri	Subtract indirect RAM from A w/ Borrow	1	1
SUBB	A,#data	Subtract immed. data from A w/ Borrow	2	1
INC	A	Increment Accumulator	1	1
INC	Rn	Increment register	1	1
INC	direct	Increment direct byte	2	1
INC	@Ri	Increment indirect RAM	1	1
DEC	A	Decrement Accumulator	1	1
DEC	Rn	Decrement register	1	1
DEC	direct	Decrement direct byte	2	1
DEC	@Ri	Decrement indirect RAM	1	1
INC	DPTR	Increment Data Pointer	1	2
MUL	AB	Multiply A & B	1	4
DIV	AB	Divide A by B	1	4
DA	A	Decimal Adjust Accumulator	1	1

LOGICAL OPERATIONS

Mnemonic		Destination	Byte	Cyc
ANL	A,Rn	AND register to Accumulator	1	1
ANL	A,direct	AND direct byte to Accumulator	2	1
ANL	A,@Ri	AND indirect RAM to Accumulator	1	1
ANL	A,#data	AND immediate data to Accumulator	2	1
ANL	direct,A	AND Accumulator to direct byte	2	1
ANL	direct,#data	AND immediate data to direct byte	3	2
ORL	A,Rn	OR register to Accumulator	1	1
ORL	A,direct	OR direct byte to Accumulator	2	1
ORL	A,@Ri	OR indirect RAM to Accumulator	1	1
ORL	A,#data	OR immediate data to Accumulator	2	1
ORL	direct,A	OR Accumulator to direct byte	2	1
ORL	direct,#data	OR immediate data to direct byte	3	2
XRL	A,Rn	Exclusive-OR register to Accumulator	1	1
XRL	A,direct	Exclusive-OR direct byte to Accumulator	2	1
XRL	A,@Ri	Exclusive-OR indirect RAM to A	1	1
XRL	A,#data	Exclusive-OR immediate data to A	2	1
XRL	direct,A	Exclusive-OR Accumulator to direct byte	2	1
XRL	direct,#data	Exclusive-OR immediate data to direct	3	2
CLR	A	Clear Accumulator	1	1
CPL	A	Complement Accumulator	1	1
RL	A	Rotate Accumulator Left	1	1
RLC	A	Rotate A Left through the Carry flag	1	1
RR	A	Rotate Accumulator Right	1	1
RRC	A	Rotate A Right through Carry flag	1	1
SWAP	A	Swap nibbles within the Accumulator	1	1

DATA TRANSFER

Mnemonic		Description	Byte	Cyc
MOV	A,Rn	Move register to Accumulator	1	1
MOV	A,direct	Move direct byte to Accumulator	2	1
MOV	A,@Ri	Move indirect RAM to Accumulator	1	1
MOV	A,#data	Move immediate data to Accumulator	2	1
MOV	Rn,A	Move Accumulator to register	1	1
MOV	Rn,direct	Move direct byte to register	2	2
MOV	Rn,#data	Move immediate data to register	2	1
MOV	direct,A	Move Accumulator to direct byte	2	1
MOV	direct,Rn	Move register to direct byte	2	2
MOV	direct,direct	Move direct byte to direct	3	2
MOV	direct,@Ri	Move indirect RAM to direct byte	2	2
MOV	direct,#data	Move immediate data to direct byte	3	2
MOV	@Ri,A	Move Accumulator to indirect RAM	1	1
MOV	@Ri,direct	Move direct byte to indirect RAM	2	2
MOV	@Ri,#data	Move immediate data to indirect RAM	2	1
MOV	DPTR,#data16	Load Data Pointer with a 16-bit constant	3	2

DATA TRANSFER (cont.)

Mnemonic		Description	Byte	Cyc
MOVC	A,@A+DPTR	Move Code byte relative to DPTR to A	1	2
MOVC	A,@A+PC	Move Code byte relative to PC to A	1	2
MOVX	A,@Ri	Move External RAM (8-bit addr) to A	1	2
MOVX	A,@DPTR	Move External RAM (16-bit addr) to A	1	2
MOVX	@Ri,A	Move A to External RAM (8-bit addr)	1	2
MOVX	@DPTR,A	Move A to External RAM (16-bit addr)	1	2
PUSH	direct	Push direct byte onto stack	2	2
POP	direct	Pop direct byte from stack	2	2
XCH	A,Rn	Exchange register with Accumulator	1	1
XCH	A,direct	Exchange direct byte with Accumulator	2	1
XCH	A,@Ri	Exchange indirect RAM with A	1	1
XCHD	A,@Ri	Exchange low-order Digit ind. RAM w/A	1	1

BOOLEAN VARIABLE MANIPULATION

Mnemonic		Description	Byte	Cyc
CLR	C	Clear Carry flag	1	1
CLR	bit	Clear direct bit	2	1
SETB	C	Set Carry flag	1	1
SETB	bit	Set direct Bit	2	1
CPL	C	Complement Carry flag	1	1
CPL	bit	Complement direct bit	2	1
ANL	C,bit	AND direct bit to Carry flag	2	2
ANL	C, bit	AND complement of direct bit to Carry	2	2
ORL	C,bit	OR direct bit to Carry flag	2	2
ORL	C, bit	OR complement of direct bit to Carry	2	2
MOV	C,bit	Move direct bit to Carry flag	2	1
MOV	bit,C	Move Carry flag to direct bit	2	2

PROGRAM AND MACHINE CONTROL

Mnemonic		Description	Byte	Cyc
ACALL	addr11	Absolute Subroutine Call	2	2
LCALL	addr16	Long Subroutine Call	3	2
RET		Return from subroutine	1	2
RETI		Return from interrupt	1	2
AJMP	addr11	Absolute Jump	2	2
LJMP	addr16	Long Jump	3	2
SJMP	rel	Short Jump (relative addr)	2	2
JMP	@A+DPTR	Jump indirect relative to the DPTR	1	2
JZ	rel	Jump if Accumulator is Zero	2	2
JNZ	rel	Jump if Accumulator is Not Zero	2	2
JC	rel	Jump if Carry flag is set	2	2
JNC	rel	Jump if No Carry flag	2	2
JB	bit,rel	Jump if direct Bit set	3	2
JNB	bit,rel	Jump if direct Bit Not set	3	2
JBC	bit,rel	Jump if direct Bit is set & Clear bit	3	2
CJNE	A,direct,rel	Compare direct to A & Jump if Not Equal	3	2
CJNE	A,#data,rel	Comp. immed. to A & Jump if Not Equal	3	2
CJNE	Rn,#data,rel	Comp. immed. to reg. & Jump if Not Equal	3	2
CJNE	@Ri,#data,rel	Comp. immed. to ind. & Jump if Not Equal	3	2
DJNZ	Rn,rel	Decrement register & Jump if Not Zero	2	2
DJNZ	direct,rel	Decrement direct & Jump if Not Zero	3	2
NOP		No operation	1	1

Notes on data addressing modes:

Rn	Working register R0-R7
direct	128 internal RAM locations, any I/O port, control or status register
@Ri	Indirect internal RAM location addressed by register R0 or R1
#data	8-bit constant included in instruction
#data16	16-bit constant included as bytes 2 & 3 of instruction
bit	128 software flags, any I/O pin, control or status bit

Notes on program addressing modes:

addr16	Destination address for LCALL & LJMP may be anywhere within the 64-Kilobyte program memory address space.
addr11	Destination address for ACALL & AJMP will be within the same 2-Kilobyte page of program memory as the first byte of the following instruction.
rel	SJMP and all conditional jumps include an 8-bit offset byte. Range is +127 -128 bytes relative to first byte of the following instruction.

3. INSTRUCTION SET AND ADDRESSING MODES

The 8051 instruction set is extremely regular, in the sense that most instructions can operate with variables from several different physical or logical address spaces. Before getting deeply enmeshed in the instruction set proper, it is important to understand the details of the most common data addressing modes. Whereas Table 4 summarizes the instructions set broken down by functional group, this chapter starts with the addressing mode classes and builds to include the related instructions.

Data Addressing Modes

MCS-51 assembly language instructions consist of an operation mnemonic and zero to three operands separated by commas. In two operand instructions the destination is specified first, then the source. Many byte-wide data

operations (such as ADD or MOV) inherently use the accumulator as a source operand and/or to receive the result. For the sake of clarity the letter "A" is specified in the source or destination field in all such instructions. For example, the instruction,

ADD A,<source>

will add the variable<source>to the accumulator, leaving the sum in the accumulator.

The operand designated "<source>" above may use any of four common logical addressing modes:

- Register—one of the working registers in the currently enabled bank.
- Direct—an internal RAM location, I/O port, or special-function register.
- Register-indirect—an internal RAM location, pointed to by a working register.
- Immediate data—an eight-bit constant incorporated into the instruction.

The first three modes provide access to the internal RAM and Hardware Register address spaces, and may therefore be used as source or destination operands; the last mode accesses program memory and may be a source operand only.

(It is hard to show a "typical application" of any instruction without involving instructions not yet described. The following descriptions use only the self-explanatory ADD and MOV instructions to demonstrate how the four addressing modes are specified and used. Subsequent examples will become increasingly complex.)

Register Addressing

The 8051 programmer has access to eight "working registers," numbered R0-R7. The least-significant three-bits of the instruction opcode indicate one register within this logical address space. Thus, a function code and operand address can be combined to form a short (one byte) instruction (Figure 12.a).

The 8051 assembly language indicates register addressing with the symbol Rn (where n is from 0 to 7) or with a symbolic name previously defined as a register by the EQUate or SET directives. (For more information on assembler directives see the Macro Assembler Reference Manual.)

Example 1—Adding Two Registers Together

```
;REGADR ADD CONTENTS OF REGISTER 1
;       TO CONTENTS OF REGISTER 0
;
REGADR: MOV     A,R0
        ADD     A,R1
        MOV     R0,A
```

There are four such banks of working registers, only one of which is active at a time. Physically, they occupy the first 32 bytes of on-chip data RAM (addresses 0-1FH). PSW bits 4 and 3 determine which bank is active. A hardware reset enables register bank 0; to select a different bank the programmer modifies PSW bits 4 and 3 accordingly.

Example 2—Selecting Alternate Memory Banks

```
MOV     PSW.#00010000B  SELECT BANK 2
```

Register addressing in the 8051 is the same as in the 8048 family, with two enhancements: there are four banks rather than one or two, and 16 instructions (rather than 12) can access them.

Direct Byte Addressing

Direct addressing can access any on-chip variable or hardware register. An additional byte appended to the opcode specifies the location to be used (Figure 12.b).

Depending on the highest order bit of the direct address byte, one of two physical memory spaces is selected. When the direct address is between 0 and 127 (00H-7FH) one of the 128 low-order on-chip RAM locations is used. (Future microcomputers based on the MCS-51™ architecture may incorporate more than 128 bytes of on-chip RAM. Even if this is the case, only the low-order 128 bytes will be directly addressable. The remainder would be accessed indirectly or via the stack pointer.)

Example 3—Adding RAM Location Contents

```
;DIRADR ADD CONTENTS OF RAM LOCATION 41H
;       TO CONTENTS OF RAM LOCATION 40H
;
DIRADR: MOV     A,40H
        ADD     A,41H
        MOV     40H,A
```

All I/O ports and special function, control, or status registers are assigned addresses between 128 and 255 (80H-0FFH). When the direct address byte is between these limits the corresponding hardware register is accessed. For example, Ports 0 and 1 are assigned direct addresses 80H and 90H, respectively. A complete list is presented in Table 5. Don't waste your time trying to memorize the addresses in Table 5. Since programs using absolute addresses for function registers would be difficult to write or understand, ASM51 allows and understands the abbreviations listed instead.

Example 4—Adding Input Port Data to Output Port Data

```
;PRTADR ADD DATA INPUT ON PORT 1
;       TO DATA PREVIOUSLY OUTPUT
;       ON PORT 0.
;
PRTADR: MOV     A,P0
        ADD     A,P1
        MOV     P0,A
```

Direct addressing allows all special-function registers in the 8051 to be read, written, or used as instruction operands. In general, this is the *only* method used for accessing I/O ports and special-function registers. If direct addressing is used with special-function register addresses other than those listed, the result of the instruction is undefined.

AFN-01502A-18

The 8048 does not have or need any generalized direct addressing mode, since there are only five special registers (BUS, P1, P2, PSW, & T) rather than twenty. Instead, 16 special 8048 opcodes control output bits or read or write each register to the accumulator. These functions are all subsumed by four of the 27 direct addressing instructions of the 8051.

Table 5. 8051 Hardware Register Direct Addresses

Register	Address	Function
P0	80H*	Port 0
SP	81H	Stack Pointer
DPL	82H	Data Pointer (Low)
DPH	83H	Data Pointer (High)
TCON	88H*	Timer register
TMOD	89H	Timer Mode register
TL0	8AH	Timer 0 Low byte
TL1	8BH	Timer 1 Low byte
TH0	8CH	Timer 0 High byte
TH1	8DH	Timer 1 High byte
P1	90H*	Port 1
SCON	98H*	Serial Port Control register
SBUF	99H	Serial Port data Buffer
P2	0A0H*	Port 2
IE	0A8H*	Interrupt Enable register
P3	0B0H*	Port 3
IP	0B8H*	Interrupt Priority register
PSW	0D0H*	Program Status Word
ACC	0E0H*	Accumulator (direct address)
B	0F0H*	B register

* = bit addressable register.

Register-Indirect Addressing

How can you handle variables whose locations in RAM are determined, computed, or modified while the program is running? This situation arises when manipulating sequential memory locations, indexed entries within tables in RAM, and multiple precision or string operations. Register or Direct addressing cannot be used, since their operand addresses are fixed at assembly time.

The 8051 solution is "register-indirect RAM addressing." R0 and R1 of each register bank may operate as index or pointer registers, their contents indicating an address into RAM. The internal RAM location so addressed is the actual operand used. The least significant bit of the instruction opcode determines which register is used as the "pointer" (Figure 12.c).

In the 8051 assembly language, register-indirect addressing is represented by a commercial "at" sign ("@") preceding R0, R1, or a symbol defined by the user to be equal to R0 or R1.

Example 5—Indirect Addressing

```
; INDADR ADD CONTENTS OF MEMORY LOCATION
;       ADDRESSED BY REGISTER 1
;       TO CONTENTS OF RAM LOCATION
;       ADDRESSED BY REGISTER 0
;
INDADR: MOV     A,@R0
        ADD     A,@R1
        MOV     @R0,A
```

Indirect addressing on the 8051 is the same as in the 8048 family, except that all eight bits of the pointer register contents are significant; if the contents point to a non-existent memory location (i.e., an address greater than 7FH on the 8051) the result of the instruction is undefined. (Future microcomputers based on the MCS-51™ architecture could implement additional memory in the on-chip RAM logical address space at locations above 7FH.) The 8051 uses register-indirect addressing for five new instructions plus the 13 on the 8048.

Immediate Addressing

When a source operand is a constant rather than a variable (i.e.—the instruction uses a value known at assembly time), then the constant can be incorporated into the instruction. An additional instruction byte specifies the value used (Figure 12.d).

The value used is fixed at the time of ROM manufacture or EPROM programming and may not be altered during program execution. In the assembly language immediate operands are preceded by a number sign ("#"). The operand may be either a numeric string, a symbolic variable, or an arithmetic expression using constants.

Example 6—Adding Constants Using Immediate Addressing

```
; IMMADR ADD THE CONSTANT 12 (DECIMAL)
;       TO THE CONSTANT 34 (DECIMAL)
;       LEAVE SUM IN ACCUMULATOR
;
IMMADR: MOV     A,#12
        ADD     A,#34
```

The preceding example was included for consistency; it has little practical value. Instead, ASM51 could compute the sum of two constants at assembly time.

Example 7—Adding Constants Using ASM51 Capabilities

```
; ASMSUM LOAD ACC WITH THE SUM OF
;       THE CONSTANT 12 (DECIMAL) AND
;       THE CONSTANT 34 (DECIMAL).
;
ASMSUM: MOV     A,#(12+34)
```

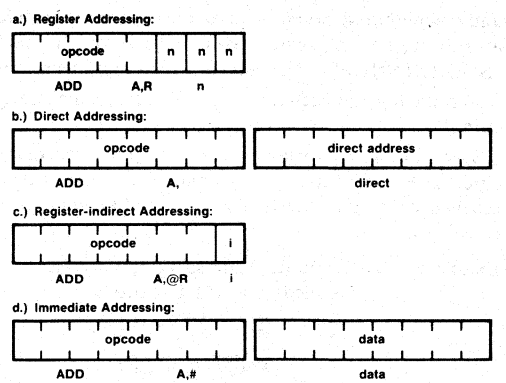


Figure 12. Data Addressing Machine Code Formats

AFN-01502A-19

Addressing Mode Combinations

The above examples all demonstrated the use of the four data-addressing modes in two-operand instructions (MOV, ADD) which use the accumulator as one operand. The operations ADDC, SUBB, ANL, ORL, and XRL (all to be discussed later) could be substituted for ADD in each example. The first three modes may be also be used for the XCH operation or, in combination with the Immediate Addressing mode (and an additional byte), loaded with a constant. The one-operand instructions INC and DEC, DJNZ, and CJNE may all operate on the accumulator, or may specify the Register, Direct, and Register-indirect addressing modes. Exception: as in the 8048, DJNZ cannot use the accumulator or indirect addressing. (The PUSH and POP operations cannot inherently address the accumulator as a special register either. However, all three can *directly* address the accumulator as one of the twenty special-function registers by putting the symbol "ACC" in the operand field.)

Advantages of Symbolic Addressing

Like most assembly or higher-level programming languages, ASM51 allows instructions or variables to be given appropriate, user-defined symbolic names. This is done for instruction lines by putting a label followed by a colon (":") before the instruction proper, as in the above examples. Such symbols must start with an alphabetic character (remember what distinguished BACH from 0BACH?), and may include any combination of letters, numbers, question marks ("?") and underscores ("_"). For **very** long names only the first 31 characters are relevant.

Assembly language programs may intermix upper- and lower-case letters arbitrarily, but ASM51 converts both to upper-case. For example, ASM51 will internally process an "I" for an "i" and, of course, "A_TOOTH" for "a_tooth."

The underscore character makes symbols easier to read and can eliminate potential ambiguity (as in the label for a subroutine to switch two entires on a stack, "S_EXCHANGE"). The underscore is significant, and would distinguish between otherwise-identical character strings.

ASM51 allows *all* variables (registers, ports, internal or external RAM addresses, constants, etc.) to be assigned labels according to these rules with the EQUate or SET directives.

Example 8—Symbolic Addressing of Variables Defined as RAM Locations

```
VAR_0   SET     20H
VAR_1   SET     21H
;
; SYMB_1 ADD CONTENTS OF VAR_1
;        TO CONTENTS OF VAR_0
;
SYMB_1: MOV     A, VAR_0
        ADD     A, VAR_1
        MOV     VAR_0, A
```

Notice from Table 4 that the MCS-51™ instruction set has relatively few instruction mnemonics (abbreviations) for the programmer to memorize. Different data types or addressing modes are determined by the operands specified, rather than variations on the mnemonic. For example, the mnemonic "MOV" is used by 18 different instructions to operate on three data types (bit, byte, and address). The fifteen versions which move byte variables between the logical address spaces are diagrammed in Figure 13. Each arrow shows the direction of transfer from source to destination.

Notice also that for most instructions allowing register addressing there is a corresponding direct addressing instruction and vice versa. This lets the programmer begin writing 8051 programs as if (s)he has access to 128 different registers. When the program has evolved to the point where the programmer has a fairly accurate idea how often each variable is used, he/she may allocate the working registers in each bank to the most "popular" variables. (The assembly cross-reference option will show exactly how often and where each symbol is referenced.) If symbolic addressing is used in writing the source program only the lines containing the symbol definition will need to be changed; the assembler will produce the appropriate instructions even though the rest of the program is left untouched. Editing only the first two lines of Example 8 will shrink the six-byte code segment produced in half.

How are instruction sets "counted"? There is no standard practice; different people assessing the same CPU using different conventions may arrive at different totals.

Each operation is then broken down according to the different addressing modes (or combinations of addressing modes) it can accommodate. The "CLR" mnemonic is used by two instructions with respect to bit variables ("CLR C" and "CLR bit") and once ("CLR A") with regards to bytes. This expansion yields the 111 separate instructions of Table 4.

The method used for the MCS-51® instruction set first breaks it down into "operations": a basic function applied to a single data type. For example, the four versions of the ADD instruction are grouped to form one operation — addition of eight-bit variables. The six forms of the ANL instruction for *byte* variables make up a different operation; the two forms of ANL which operate on *bits* are considered still another. The MOV mnemonic is used by three different operation classes, depending on whether bit, byte, or 16-bit values are affected. Using this terminology the 8051 can perform 51 different operations.

AFN-01502A-20

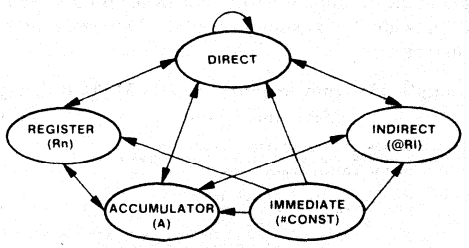


Figure 13. Road map for moving data bytes

Example 9—Redeclaring Example 8 Symbols as Registers

```
VAR_0   SET     R0
VAR_1   SET     R1
;
;SYMB_2 ADD CONTENTS OF VAR_1
;       TO CONTENTS OF VAR_0
;
SYMB_2: MOV     A, VAR_0
        ADD     A, VAR_1
        MOV     VAR_0, A
```

Arithmetic Instruction Usage — ADD, ADDC, SUBB and DA

The ADD instruction adds a byte variable with the accumulator, leaving the result in the accumulator. The carry flag is set if there is an overflow from bit 7 and cleared otherwise. The AC flag is set to the carry-out from bit 3 for use by the DA instruction described later. ADDC adds the previous contents of the carry flag with the two byte variables, but otherwise is the same as ADD.

The SUBB (subtract with borrow) instruction subtracts the byte variable indicated and the contents of the carry flag together from the accumulator, and puts the result back in the accumulator. The carry flag serves as a "Borrow Required" flag during subtraction operations; when a greater value is subtracted from a lesser value (as in subtracting 5 from 1) requiring a borrow into the highest order bit, the carry flag is set; otherwise it is cleared.

When performing signed binary arithmetic, certain combinations of input variables can produce results which seem to violate the Laws of Mathematics. For example, adding 7FH (127) to itself produces a sum of 0FEH, which is the two's complement representation of -2 (refer back to Table 2)! In "normal" arithmetic, two positive values can't have a negative sum. Similarly, it js normally impossible to subtract a positive value from a negative value and leave a positive result — but in two's complement there are instances where this too may happen. Fundamentally, such anomolies occur when the magnitude of the resulting value is too great to "fit" into the seven bits allowed for it; there is no one-byte two's complement representation for 254, the true sum of 127 and 127.

The MCS-51™ processors detect whether these situations occur and indicate such errors with the OV flag. (OV may be tested with the conditional jump instructions JB and JNB, described under the Boolean Processor chapter.)

At a hardware level, OV is set if there is a carry out of bit 6 but not out of bit 7, or a carry out of bit 7 but not out of bit 6. When adding signed integers this indicates a negative number produced as the sum of two positive operands, or a positive sum from two negative operands; on SUBB this indicates a negative result after subtracting a negative number from a positive number, or a positive result when a positive number is subtracted from a negative number.

The ADDC and SUBB instructions incorporate the previous state of the carry (borrow) flag to allow multiple precision calculations by repeating the operation with successively higher-order operand bytes. In either case, the carry must be cleared before the first iteration.

If the input data for a multiple precision operation is an unsigned string of integers, upon completion the carry flag will be set if an overflow (for ADDC) or underflow (for SUBB) occurs. With two's complement signed data (i.e., if the most significant bit of the original input data indicates the sign of the string), the overflow flag will be set if overflow or underflow occurred.

Example 10—String Subtraction with Signed Overflow Detection

```
;SUBSTR SUBTRACT STRING INDICATED BY R1
;       FROM STRING INDICATED BY R0 TO
;       PRECISION INDICATED BY R2.
;       CHECK FOR SIGNED UNDERFLOW WHEN DONE.
;
SUBSTR: CLR     C           ;BORROW= 0
SUBS1:  MOV     A, @R0
        SUBB    A, @R1      ;SUBTRACT NEXT PLACE
        MOV     @R0, A
        INC     R0          ;BUMP POINTERS
        INC     R1
        DJNZ    R2, SUBS1           ;LOOP AS NEEDED
;               WHEN DONE, TEST IF OVERFLOW OCCURED
;               ON LAST ITERATION OF LOOP
        JNB     OV, OV_OK
;                           (OVERFLOW RECOVERY ROUTINE)
OV_OK:  RET                 ;RETURN
```

Decimal addition is possible by using the DA instruction in conjunction with ADD and/or ADDC. The eight-bit binary value in the accumulator resulting from an earlier addition of two variables (each a packed BCD digit-pair) is adjusted to form two BCD digits of four bits each. If the contents of accumulator bits 3-0 are greater than nine (xxxx1010-xxxx1111), or if the AC flag had been set, six is added to the accumulator producing the proper BCD digit in the low-order nibble. (This addition might itself set — but would not clear — the carry flag.) If the carry flag is set, or if the four high-order bits now exceed nine (1010xxxx-1111xxxx), these bits are incremented by six. The carry flag is left set if originally set or if either addition of six produces a carry out of the highest-order bit, indicating the sum of the original two BCD variables is greater than or equal to decimal 100.

Example 11—Two Byte Decimal Add with Registers and Constants

```
;BCDADD ADD THE CONSTANT 1,234 (DECIMAL) TO THE
;       CONTENTS OF REGISTER PAIR <R3><R2>
;       (ALREADY A 4 BCD-DIGIT VARIABLE)
;
BCDADD: MOV     A,R2
        ADD     A,#34H
        DA      A
        MOV     R2,A
        MOV     A,R3
        ADDC    A,#12H
        DA      A
        MOV     R3,A
        RET
```

Multiplication and Division

The instruction "MUL AB" multiplies the unsigned eight-bit integer values held in the accumulator and B-registers. The low-order byte of the sixteen-bit product is left in the accumulator, the higher-order byte in B. If the high-order eight-bits of the product are all zero the overflow flag is cleared; otherwise it is set. The programmer can poll OV to determine when the B register is non-zero and must be processed.

"DIV AB" divides the unsigned eight-bit integer in the accumulator by the unsigned eight-bit integer in the B-register. The integer part of the quotient is returned in the accumulator; the remainder in the B-register. If the B-register originally contained 00H then the overflow flag will be set to indicate a division error, and the values returned will be undefined. Otherwise OV is cleared.

The divide instruction is also useful for purposes such as radix conversion or separating bit fields of the accumulator. A short subroutine can convert an eight-bit unsigned binary integer in the accumulator (between 0 & 255) to a three-digit (two byte) BCD representation. The hundred's digit is returned in one register (HUND) and the ten's and one's digits returned as packed BCD in another (TENONE).

Example 12—Use of DIV Instruction for Radix Conversion

```
;BINBCD CONVERT 8-BIT BINARY VARIABLE IN ACC
;       TO 3-DIGIT PACKED BCD FORMAT
;       HUNDREDS' PLACE LEFT IN VARIABLE 'HUND',
;       TENS' AND ONES' PLACES IN 'TENONE'.
;
HUND    EQU     21H
TENONE  EQU     22H
;
BINBCD: MOV     B,#100  ;DIVIDE BY 100 TO
        DIV     AB      ;DETERMINE NUMBER OF HUNDREDS
        MOV     HUND,A
        MOV     A,#10   ;DIVIDE REMAINDER BY 10 TO
        XCH     A,B     ;DETERMINE # OF TENS LEFT
        DIV     AB      ;TENS DIGIT IN ACC, REMAINDER IS ONES
                        ;DIGIT
        SWAP    A
        ADD     A,B     ;PACK BCD DIGITS IN ACC
        MOV     TENONE,A
        RET
```

The divide instruction can also separate eight bits of data in the accumulator into sub-fields. For example, packed BCD data may be separated into two nibbles by dividing the data by 16, leaving the high-nibble in the accumulator and the low-order nibble (remainder) in B. The two digits may then be operated on individually or in conjunction with each other. This example receives two packed BCD digits in the accumulator and returns the product of the two individual digits in packed BCD format in the accumulator.

Example 13—Implementing a BCD Multiply Using MPY and DIV

```
;MULBCD UNPACK TWO BCD DIGITS RECEIVED IN ACC,
;       FIND THEIR PRODUCT, AND RETURN PRODUCT
;       IN PACKED BCD FORMAT IN ACC
;
MULBCD: MOV     B,#10H  ;DIVIDE INPUT BY 16
        DIV     AB      ;A & B HOLD SEPARATED DIGITS
                        ;(EACH RIGHT JUSTIFIED IN REGISTER)
        MUL     AB      ;A HOLDS PRODUCT IN BINARY FORMAT (0 -
                        ;99(DECIMAL) = 0 - 63H)
        MOV     B,#10   ;DIVIDE PRODUCT BY 10
        DIV     AB      ;A HOLDS # OF TENS, B HOLDS REMAINDER
        SWAP    A
        ORL     A,B     ;PACK DIGITS
        RET
```

Logical Byte Operations — ANL, ORL, XRL

The instructions ANL, ORL, and XRL perform the logical functions AND, OR, and/or Exclusive-OR on the two byte variables indicated, leaving the results in the first. No flags are affected. (A word to the wise — do not vocalize the first two mnemonics in mixed company.)

These operations may use all the same addressing modes as the arithmetics (ADD, etc.) but unlike the arithmetics, they are not restricted to operating on the accumulator. Directly addressed bytes may be used as the destination with either the accumulator or a constant as the source. These instructions are useful for clearing (ANL), setting (ORL), or complementing (XRL) one or more bits in a RAM, output ports, or control registers. The pattern of bits to be affected is indicated by a suitable mask byte. Use immediate addressing when the pattern to be affected is known at assembly time (Figure 14); use the accumulator versions when the pattern is computed at run-time.

I/O ports are often used for parallel data in formats other than simple eight-bit bytes. For example, the low-order five bits of port 1 may output an alphabetic character code (hopefully) without disturbing bits 7-5. This can be a simple two-step process. First, clear the low-order five pins with an ANL instruction; then set those pins corresponding to ones in the accumulator. (This example assumes the three high-order bits of the accumulator are originally zero.)

Example 14—Reconfiguring Port Size with Logical Byte Instructions

```
OUT_PX: ANL   P1,#11100000B ;CLEAR BITS P1.4 - P1.0
        ORL   P1,A          ;SET P1 PINS CORRESONDING TO SET ACC
                            ; BITS.
        RET
```

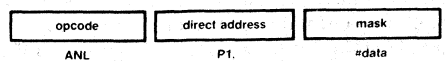


Figure 14. Instruction Pattern for Logical Operation Special Addressing Modes

AFN-01502A-22

In this example, low-order bits remaining high may "glitch" low for one machine cycle. If this is undesirable, use a slightly different approach. First, set all pins corresponding to accumulator one bits, then clear the pins corresponding to zeroes in low-order accumulator bits. Not all bits will change from original to final state at the same instant, but no bit makes an intermediate transition.

Example 15—Reconfiguring I/O Port Size without Glitching

```
ALT_PX  ORL     P1, A
        ORL     A, #11100000B
        ANL     P1, A
        RET
```

Program Control — Jumps, Calls, Returns

Whereas the 8048 only has a single form of the simple jump instruction, the 8051 has three. Each causes the program to unconditionally jump to some other address. They differ in how the machine code represents the destination address.

LJMP (Long Jump) encodes a sixteen-bit address in the second and third instruction bytes (Figure 15.a); the destination may be anywhere in the 64 Kilobyte program memory address space.

The two-byte AJMP (Absolute Jump) instruction encodes its destination using the same format as the 8048: address bits 10 through 8 form a three bit field in the opcode and address bits 7 through 0 form the second byte (Figure 15.b). Address bits 15-12 are unchanged from the (incremented) contents of the P.C., so AJMP can only be used when the destination is known to be within the same 2K memory block. (Otherwise ASM51 will point out the error.)

A different two-byte jump instruction is legal with any nearby destination, regardless of memory block boundaries or "pages." SJMP (Short Jump) encodes the destination with a program counter-relative address in the second byte (Figure 15.c). The CPU calculates the destination at run-time by adding the signed eight-bit displacement value to the incremented P.C. Negative offset values will cause jumps up to 128 bytes backwards; positive values up to 127 bytes forwards. (SJMP with 00H in the machine code offset byte will proceed with the following instruction).

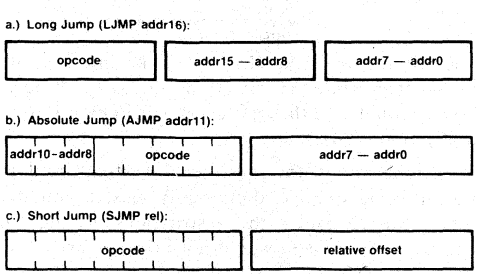


Figure 15. Jump Instruction Machine Code Formats

In keeping with the 8051 assembly language goal of minimizing the number of instruction mnemonics, there is a "generic" form of the three jump instructions. ASM51 recognizes the mnemonic JMP as a "pseudo-instruction," translating it into the machine instructions LJMP, AJMP, or SJMP, depending on the destination address.

Like SJMP, all conditional jump instructions use relative addressing. JZ (Jump if Zero) and JNZ (Jump if Not Zero) monitor the state of the accumulator as implied by their names, while JC (Jump on Carry) and JNC (Jump on No Carry) test whether or not the carry flag is set. All four are two-byte instructions, with the same format as Figure 15.c. JB (Jump on Bit), JNB (Jump on No Bit) and JBC (Jump on Bit then Clear Bit) can test any status bit or input pin with a three byte instruction; the second byte specifies which bit to test and the third gives the relative offset value.

There are two subroutine-call instructions, LCALL (Long Call) and ACALL (Absolute Call). Each increments the P.C. to the first byte of the following instruction, then pushes it onto the stack (low byte first). Saving both bytes increments the stack pointer by two. The subroutine's starting address is encoded in the same ways as LJMP and AJMP. The generic form of the call operation is the mnemonic CALL, which ASM51 will translate into LCALL or ACALL as appropriate.

The return instruction RET pops the high- and low-order bytes of the program counter successively from the stack, decrementing the stack pointer by two. Program execution continues at the address previously pushed: the first byte of the instruction immediately following the call.

When an interrupt request is recognized by the 8051 hardware, two things happen. Program control is automatically "vectored" to one of the interrupt service routine starting addresses by, in effect, forcing the CPU to process an LCALL instead of the next instruction. This automatically stores the return address on the stack. (Unlike the 8048, no status information is automatically saved.)

Secondly, the interrupt logic is disabled from accepting any other interrupts from the same or lower priority. After completing the interrupt service routine, executing an RETI (Return from Interrupt) instruction will return execution to the point where the background program was interrupted — just like RET — while restoring the interrupt logic to its previous state.

AFN-01502A-23

Operate-and-branch instructions — CJNE, DJNZ

Two groups of instructions combine a byte operation with a conditional jump based on the results.

CJNE (Compare and Jump if Not Equal) compares two byte operands and executes a jump if they disagree. The carry flag is set following the rules for subtraction: if the unsigned integer value of the first operand is less than that of the second it is set; otherwise, it is cleared. However, neither operand is modified.

The CJNE instruction provides, in effect, a one-instruction "case" statement. This instruction may be executed repeatedly, comparing the code variable to a list of "special case" value: the code segment following the instruction (up to the destination label) will be executed only if the operands match. Comparing the accumulator or a register to a series of constants is a convenient way to check for special handling or error conditions; if none of the cases match the program will continue with "normal" processing.

A typical example might be a word processing device which receives ASCII characters through the serial port and drives a thermal hard-copy printer. A standard routine translates "printing" characters to bit patterns, but control characters (<DEL>, <CR>, <LF>, <BEL>, <ESC> or <SP>) must invoke corresponding special routines. Any other character with an ASCII code less than 20H should be translated into the <NUL> value, 00H, and processed with the printing characters.

Example 16 — Case Statements Using CJNE

```
CHAR    EQU     R7          ;CHARACTER CODE VARIABLE
;
INTERP: CJNE    CHAR,#7FH,INTP_1
;                           (SPECIAL ROUTINE FOR RUBOUT CODE)
        RET
INTP_1: CJNE    CHAR,#07H,INTP_2
;                           (SPECIAL ROUTINE FOR BELL CODE)
        RET
INTP_2: CJNE    CHAR,#0AH,INTP_3
;                           (SPECIAL ROUTINE FOR LFEED CODE)
        RET
INTP_3: CJNE    CHAR,#0DH,INTP_4
;                           (SPECIAL ROUTINE FOR RETURN CODE)
        RET
INTP_4: CJNE    CHAR,#1BH,INTP_5
;                           (SPECIAL ROUTINE FOR ESCAPE CODE)
        RET
INTP_5: CJNE    CHAR,#20H,INTP_6
;                           (SPECIAL ROUTINE FOR SPACE CODE)
        RET
INTP_6: JC      PRINTC      ;JUMP IF CODE > 20H
        MOV     CHAR,#0     ;REPLACE CONTROL CHARACTERS WITH
;                           ; NULL CODE
PRINTC:                     ;PROCESS STANDARD PRINTING
;                           ; CHARACTER
        RET
```

DJNZ (Decrement and Jump if Not Zero) decrements the register or direct address indicated and jumps if the result is not zero, without affecting any flags. This provides a simple means for executing a program loop a given number of times, or for adding a moderate time delay (from 2 to 512 machine cycles) with a single instruction. For example, a 99-usec. software delay loop can be added to code forcing an I/O pin low with only two instructions.

Example 17 — Inserting a Software Delay with DJNZ

```
        CLR     WR
        MOV     R2,#49
        DJNZ    R2,$
        SETB    WR
```

The dollar sign in this example is a special character meaning "the address of this instruction." It is useful in eliminating instruction labels on the same or adjacent source lines. CJNE and DJNZ (like all conditional jumps) use program-counter relative addressing for the destination address.

Stack Operations — PUSH, POP

The PUSH instruction increments the stack pointer by one, then transfers the contents of the single byte variable indicated (direct addressing only) into the internal RAM location addressed by the stack pointer. Conversely, POP copies the contents of the internal RAM location addressed by the stack pointer to the byte variable indicated, then decrements the stack pointer by one.

(Stack Addressing follows the same rules, and addresses the same locations as Register-indirect. Future microcomputers based on the MCS-51™ CPU could have up to 256 bytes of RAM for the stack.)

Interrupt service routines must not change any variable or hardware registers modified by the main program, or else the program may not resume correctly. (Such a change might look like a spontaneous random error.) Resources used or altered by the service routine (Accumulator, PSW, etc.) must be saved and restored to their previous value before returning from the service routine. PUSH and POP provide an efficient and convenient way to save register states on the stack.

Example 18 — Use of the Stack for Status Saving on Interrupts

```
LOC_TMP EQU     $           ;REMEMBER LOCATION COUNTER
;
        ORG     0003H       ;STARTING ADDRESS FOR INTERRUPT ROUTINE
        LJMP    SERVER      ;JUMP TO ACTUAL SERVICE ROUTINE LOCATED
;                           ; ELSEWHERE
;
        ORG     LOC_TMP     ;RESTORE LOCATION COUNTER
SERVER  PUSH    PSW
        PUSH    ACC         ;SAVE ACCUMULATOR (NOTE DIRECT ADDRESSING
;                           ; NOTATION)
        PUSH    B           ;SAVE B REGISTER
        PUSH    DPL         ;SAVE DATA POINTER
        PUSH    DPH         ;
        MOV     PSW,#00001000B  ;SELECT REGISTER BANK 1
;
;
        POP     DPH         ;RESTORE REGISTERS IN REVERSE ORDER
        POP     DPL
        POP     B
        POP     ACC
        POP     PSW         ;RESTORE PSW AND RE-SELECT ORIGINAL
;                           ; REGISTER BANK
        RETI                ;RETURN TO MAIN PROGRAM AND RESTORE
;                           ; INTERRUPT LOGIC
```

If the SP register held 1FH when the interrupt was detected, then while the service routine was in progress the stack would hold the registers shown in Figure 16; SP would contain 26H.

The example shows the most general situation; if the service routine doesn't alter the B-register and data pointer, for example, the instructions saving and restoring those registers would not be necessary.

The stack may also pass parameters to and from subroutines. The subroutine can indirectly address the parameters derived from the contents of the stack pointer.

AFN-01502A-24

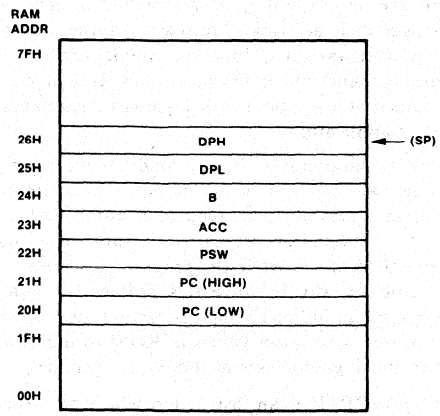


Figure 16. Stack contents during interrupt

One advantage here is simplicity. Variables need not be allocated for specific parameters, a potentially large number of parameters may be passed, and different calling programs may use different techniques for determining or handling the variables.

For example, the following subroutine reads out a parameter stored on the stack by the calling program, uses the low order bits to access a local look-up table holding bit patterns for driving the coils of a four phase stepper motor, and stores the appropriate bit pattern back in the same position on the stack before returning. The accumulator contents are left unchanged.

Example 19 — Passing Variable Parameters to Subroutines Using the Stack

```
NXTPOS: MOV     R0, SP
        DEC     R0          ;ACCESS LOCATION PARAMETER PUSHED INTO
        DEC     R0
        XCH     A, @R0      ;READ INPUT PARAMETER AND SAVE
                            ; ACCUMULATOR
        ANL     A, #03H     ;MASK ALL BUT LOW-ORDER TWO BITS
        ADD     A, #2       ;ALLOW FOR OFFSET FROM MOVC TO TABLE
        MOVC    A, @A+PC    ;READ LOOK-UP TABLE ENTRY
        XCH     A, @R0      ;PASS BACK TRANSLATED VALUE AND RESTORE
                            ; ACC
        RET                 ;RETURN TO BACKGROUND PROGRAM
STPTBL: DB      01101111B   ;POSITION 0
        DB      01011111B   ;POSITION 1
        DB      10011111B   ;POSITION 2
        DB      10101111B   ;POSITION 3
```

The background program may reach this subroutine with several different calling sequences, all of which PUSH a value before calling the routine and POP the result after. A motor on Port 1 may be initialized by placing the desired position (zero) on the stack before calling the subroutine and outputing the results directly to a port afterwards.

Example 20 — Sending and Receiving Data Parameters Via the Stack

```
        CLR     A
        PUSH    ACC
        CALL    NXTPOS
        POP     P1
```

If the position of the motor is determined by the contents of variable POSM1 (a byte in internal RAM) and the position of a second motor on Port 2 is determined by the data input to the low-order nibble of Port 2, a six-instruction sequence could update them both.

Example 21 — Loading and Unloading Stack Direct from I/O Ports

```
POSM1   EQU     51

        PUSH    POSM1
        CALL    NXTPOS
        POP     P1
        PUSH    P2
        CALL    NXTPOS
        POP     P2
```

Data Pointer and Table Look-up Instructions — MOV, INC, MOVC, JMP

The data pointer can be loaded with a 16-bit value using the instruction MOV DPTR, #data16. The data used is stored in the second and third instruction bytes, high-order byte first. The data pointer is incremented by INC DPTR. A 16-bit increment is performed; an overflow from the low byte will carry into the high-order byte. Neither instruction affects any flags.

The MOVC (Move Constant) instructions (MOVC A,@A+DPTR and MOVC A,@A+PC) read into the accumulator bytes of data from the program memory logical address space. Both use a form of indexed addressing: the former adds the unsigned eight-bit accumulator contents with the sixteen-bit data pointer register, and uses the resulting sum as the address from which the byte is fetched. A sixteen-bit addition is performed; a carry-out from the low-order eight bits may propagate through higher-order bits, but the contents of the DPTR are not altered. The latter form uses the incremented program counter as the "base" value instead of the DPTR (figure 17). Again, neither version affects the flags.

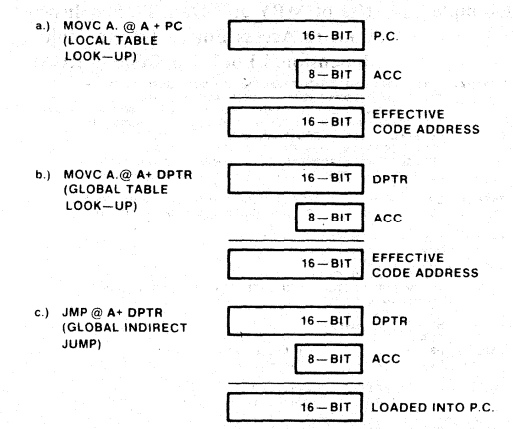


Figure 17. Operation of MOVC instructions

AFN-01502A-25

Each can be part of a three step sequence to access look-up tables in ROM. To use the DPTR-relative version, load the Data Pointer with the starting address of a look-up table; load the accumulator with (or compute) the index of the entry desired; and execute MOVC A,@A+DPTR. Unlike the similar MOVP3 instructions in the 8048, the table may be located anywhere in program memory. The data pointer may be loaded with a constant for short tables. Or to allow more complicated data structures, or tables with more than 256 entries, the values for DPH and DPL may be computed or modified with the standard arithmetic instruction set.

The PC-relative version has the advantage of not affecting the data pointer. Again, a look-up sequence takes three steps: load the accumulator with the index; compensate for the offset from the look-up instruction to the start of the table by adding the number of bytes separating them to the accumulator; then execute the MOVC A,@A+PC instruction.

Let's look at a non-trivial situation where this instruction would be used. Some applications store large multi-dimensional look-up tables of dot matrix patterns, non-linear calibration parameters, and so on in a linear (one-dimensional) vector in program memory. To retrieve data from the tables, variables representing matrix indices must be converted to the desired entry's memory address. For a matrix of dimensions (MDIMEN x NDIMEN) starting at address BASE and respective indices INDEXI and INDEXJ, the address of element (INDEXI, INDEXJ) is determined by the formula,

Entry Address = BASE + (NDIMEN x INDEXI) + INDEXJ

The code shown below can access any array with less than 255 entries (i.e., an 11x21 array with 231 elements). The table entries are defined using the Data Byte ("DB") directive, and will be contained in the assembly object code as part of the accessing subroutine itself.

Example 22—Use of MPY and Data Pointer Instructions to Access Entries from a Multi-dimensional Look-Up Table in ROM

```
;MATRX1 LOAD CONSTANT READ FROM TWO DIMENSIONAL LOOK-UP
;       TABLE IN PROGRAM MEMORY INTO ACCUMULATOR
;       USING LOCAL TABLE LOOK-UP INSTRUCTION, 'MOVC    A,@A+PC'
;       THE TOTAL NUMBER OF TABLE ENTRIES IS ASSUMED TO
;       BE SMALL, I.E. LESS THAN ABOUT 250 ENTRIES )
;       TABLE USED IN THIS EXAMPLE IS ( 11 X 21 )
;       DESIRED ENTRY ADDRESS IS GIVEN BY THE FORMULA,
;       [ (BASE ADDRESS) + (21 X INDEXI) + (INDEXJ) ]
;
INDEXI  EQU     R6      ;FIRST COORDINATE OF ENTRY (0-10)
INDEXJ  EQU     23H     ;SECOND COORDINATE OF ENTRY (0-20)
;
MATRX1: MOV     A,INDEXI
        MOV     B,#21
        MUL     AB
        ADD     A,INDEXJ
;       ALLOW FOR INSTRUCTION BYTE BETWEEN "MOVC" AND
;       ENTRY (0,0)
        INC     A
        MOVC    A,@A+PC
        RET
BASE1:  DB      1       ;(entry 0,0)
        DB      2       ;(entry 0,1)
;
        DB      21      ;(entry 0,20)
        DB      22      ;(entry 1,0)
;
        DB      42      ;(entry 1,20)
;
;
        DB      231     ;(entry 10,20)
```

There are several different means for branching to sections of code determined or selected at run time. (The single destination addresses incorporated into conditional and unconditional jumps are, of course, determined at assembly time). Each has advantages for different applications.

The most common is an N-way conditional jump based on some variable, with all of the potential destinations known at assembly time. One of a number of small routines is selected according to the value of an index variable determined while the program is running. The most efficient way to solve this problem is with the MOVC and an indirect jump instruction, using a short table of one byte offset values in ROM to indicate the relative starting addresses of the several routines.

JMP @A+DPTR is an instruction which performs an indirect jump to an address determined during program execution. The instruction adds the eight-bit unsigned accumulator contents with the contents of the sixteen-bit data pointer, just like MOVC A,@A+DPTR. The resulting sum is loaded into the program counter and is used as the address for subsequent instruction fetches. Again, a sixteen-bit addition is performed; a carry out from the low-order eight bits may propagate through the higher-order bits. In this case, neither the accumulator contents nor the data pointer is altered.

The example subroutine below reads a byte of RAM into the accumulator from one of four alternate address spaces, as selected by the contents of the variable MEMSEL. The address of the byte to be read is determined by the contents of R0 (and optionally R1). It might find use in a printing terminal application, where four different model printers all use the same ROM code but use different types and sizes of buffer memory for different speeds and options.

Example 23—N-Way Branch and Computed Jump Instructions via JMP @ ADPTR

```
MEMSEL  EQU     R3

JUMP_4: MOV     A,MEMSEL
        MOV     DPTR,#JMPTBL
        MOVC    A,@A+DPTR
        JMP     @A+DPTR
JMPTBL: DB      MEMSP0-JMPTBL
        DB      MEMSP1-JMPTBL
        DB      MEMSP2-JMPTBL
        DB      MEMSP3-JMPTBL
MEMSP0: MOV     A,@R0   ;READ FROM INTERNAL RAM
        RET
MEMSP1: MOVX    A,@R0   ;READ FROM 256 BYTES OF EXTERNAL RAM
        RET
MEMSP2: MOV     DPL,R0
        MOV DPH,R1
        MOVX    A,@DPTR ;READ FROM 64K BYTES OF EXTERNAL RAM
        RET
MEMSP3: MOV     A,R1
        ANL     A,#07H
        ANL     P1,#11111000B
        ORL     P1,A
        MOVX    A,@R0   ;READ FROM 4K BYTES OF EXTERNAL RAM
        RET
```

Note that this approach is suitable whenever the size of jump table plus the length of the alternate routines is less than 256 bytes. The jump table and routines may be located anywhere in program memory, independent of 256-byte program memory pages.

AFN-01502A-26

For applications where up to 128 destinations must be selected, all of which reside in the same 2K page of program memory which may be reached by the two-byte absolute jump instructions, the following technique may be used. In the above mentioned printing terminal example, this sequence could "parse" 128 different codes for ASCII characters arriving via the 8051 serial port.

Example 24 — N-Way Branch with 128 Optional Destinations

```
OPTION  EQU     R3
;
;
JMP128  MOV     A,OPTION
        RL      A               ;MULTIPLY BY 2 FOR 2 BYTE JUMP TABLE
        MOV     DPTR,#INSTBL    ;FIRST ENTRY IN JUMP TABLE
        JMP     @A+DPTR         ;JUMP INTO JUMP TABLE
;
INSTBL  AJMP    PROC00  ;128 CONSECUTIVE
        AJMP    PROC01  ;AJMP INSTRUCTIONS
        AJMP    PROC02
;
;
        AJMP    PROC7E
        AJMP    PROC7F
```

The destinations in the jump table (PROC00-PROC7F) are not all necessarily unique routines. A large number of special control codes could each be processed with their own unique routine, with the remaining printing characters all causing a branch to a common routine for entering the character into the output queue.

In those rare situations where even 128 options are insufficient, or where the destination routines may cross a 2K page boundary, the above approach may be modified slightly as shown below.

Example 25 — 256-Way Branch Using Address Look-Up Tables

```
RTEMP   EQU     R7

JMP256  MOV     DPTR,#ADRTBL    ;FIRST ENTRY IN TABLE OF ADDRESSES
        MOV     A,OPTION
        CLR     C
        RLC     A               ;MULTIPLY BY 2 FOR 2 BYTE JUMP TABLE
        JNC     LOW128
        INC     DPH
LOW128  MOV     RTEMP,A         ;SAVE ACC FOR HIGH BYTE READ
        MOVC    A,@A+DPTR       ;READ LOW BYTE FROM JUMP TABLE
        XCH     A,RTEMP
        INC     A
        MOVC    A,@A+DPTR       ;GET LOW-ORDER BYTE FROM TABLE
        PUSH    ACC
        MOV     A,RTEMP
        MOVC    A,@A+DPTR       ;GET HIGH-ORDER BYTE FROM TABLE
        PUSH    ACC
;       THE TWO ACC PUSHES HAVE PRODUCED
;       A "RETURN ADDRESS" ON THE STACK WHICH CORRESPONDS
;       TO THE DESIRED STARTING ADDRESS.
;       IT MAY BE REACHED BY POPPING THE STACK
;       INTO THE PC.
        RET
;
ADRTBL  DW      PROC00  ;UP TO 256 CONSECUTIVE DATA
        DW      PROC01  ;WORDS INDICATING STARTING ADDRESSES
;
        DW      PROCFF
;
;
;       DUMMY CODE ADDRESS DEFINITIONS NEEDED BY ABOVE
;       TWO EXAMPLES
;
PROC00  NOP
PROC01  NOP
PROC02  NOP
PROC7E  NOP
PROC7F  NOP
PROCFF  NOP
```

4. BOOLEAN PROCESSING INSTRUCTIONS

The commonly accepted terms for tasks at either end of the computational vs. control application spectrum are, respectively, "number-crunching" and "bit-banging". Prior to the introduction of the MCS-51™ family, nice number-crunchers made bad bit-bangers and vice versa. The 8051 is the industry's first single-chip microcomputer designed to crunch **and** bang. (In some circles, the latter technique is also referred to as "bit-twiddling". Either is correct.)

Direct Bit Addressing

A number of instructions operate on Boolean (one-bit) variables, using a direct bit addressing mode comparable to direct byte addressing. An additional byte appended to the opcode specifies the Boolean variable, I/O pin, or control bit used. The state of any of these bits may be tested for "true" or "false" with the conditional branch instructions JB (Jump on Bit) and JNB (Jump on Not Bit). The JBC (Jump on Bit and Clear) instruction combines a test-for-true with an unconditional clear.

As in direct byte addressing, bit 7 of the address byte switches between two physical address spaces. Values between 0 and 127 (00H-7FH) define bits in internal RAM locations 20H to 2FH (Figure 18a); address bytes between 128 and 255 (80H-0FFH) define bits in the 2 x "special-function" register address space (Figure 18b). If no 2 x "special-function" register corresponds to the direct bit address used the result of the instruction is undefined.

Bits so addressed have many wondrous properties. They may be set, cleared, or complemented with the two byte instructions SETB, CLR, or CPL. Bits may be moved to and from the carry flag with MOV. The logical ANL and ORL functions may be performed between the carry and either the addressed bit or its complement.

Bit Manipulation Instructions — MOV

The "MOV" mnemonic can be used to load an addressable bit into the carry flag ("MOV C, bit") or to copy the state of the carry to such a bit ("MOV bit, C"). These instructions are often used for implementing serial I/O algorithms via software or to adapt the standard I/O port structure.

It is sometimes desirable to "re-arrange" the order of I/O pins because of considerations in laying out printed circuit boards. When interfacing the 8051 to an immediately adjacent device with "weighted" input pins, such as keyboard column decoder, the corresponding pins are likely to be not aligned (Figure 19).

There is a trade-off in "scrambling" the interconnections with either interwoven circuit board traces or through software. This is extremely cumbersome (if not impossible) to do with byte-oriented computer architectures. The 8051's unique set of Boolean instructions makes it simple to move individual bits between arbitrary locations.

a.) RAM Bit Addresses.

RAM BYTE	(MSB)							(LSB)
7FH								
2FH	7F	7E	7D	7C	7B	7A	79	78
2EH	77	76	75	74	73	72	71	70
2DH	6F	6E	6D	6C	6B	6A	69	68
2CH	67	66	65	64	63	62	61	60
2BH	5F	5E	5D	5C	5B	5A	59	58
2AH	57	56	55	54	53	52	51	50
29H	4F	4E	4D	4C	4B	4A	49	48
28H	47	46	45	44	43	42	41	40
27H	3F	3E	3D	3C	3B	3A	39	38
26H	37	36	35	34	33	32	31	30
25H	2F	2E	2D	2C	2B	2A	29	28
24H	27	26	25	24	23	22	21	20
23H	1F	1E	1D	1C	1B	1A	19	18
22H	17	16	15	14	13	12	11	10
21H	0F	0E	0D	0C	0B	0A	09	08
20H	07	06	05	04	03	02	01	00
1FH	Bank 3							
18H								
17H	Bank 2							
10H								
0FH	Bank 1							
08H								
07H	Bank 0							
00H								

b.) Hardware Register Bit Addresses.

Direct Byte Address	(MSB)			Bit Addresses				(LSB)	Hardware Register Symbol
0FFH									
0F0H	F7	F6	F5	F4	F3	F2	F1	F0	B
0E0H	E7	E6	E5	E4	E3	E2	E1	E0	ACC
0D0H	D7	D6	D5	D4	D3	D2	D1	D0	PSW
0B8H	—	—	—	BC	BB	BA	B9	B8	IP
0B0H	B7	B6	B5	B4	B3	B2	B1	B0	P3
0A8H	AF	—	—	AC	AB	AA	A9	A8	IE
0A0H	A7	A6	A5	A4	A3	A2	A1	A0	P2
98H	9F	9E	9D	9C	9B	9A	99	98	SCON
90H	97	96	95	94	93	92	91	90	P1
88H	8F	8E	8D	8C	8B	8A	89	88	TCON
80H	87	86	85	84	83	82	81	80	P0

Figure 18. Bit Address Maps

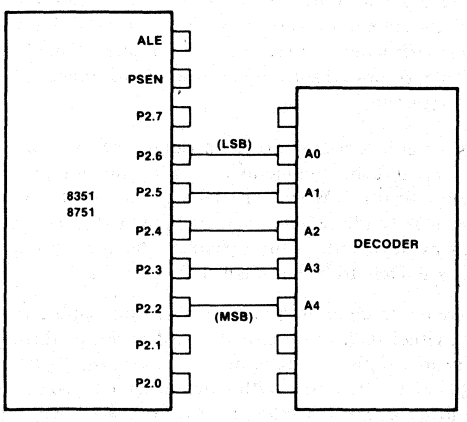


Figure 19. "Mismatch" Between I/O port and Decoder

Example 26 — Re-ordering I/O Port Configuration

```
OUT_P2: RRC     A       ;MOVE ORIGINAL ACC.0 INTO CY
        MOV     P2.6,C  ;STORE CARRY TO PIN P26
        RRC     A       ;MOVE ORIGINAL ACC.1 INTO CY
        MOV     P2.5,C  ;STORE CARRY TO PIN P25
        RRC     A       ;MOVE ORIGINAL ACC.2 INTO CY
        MOV     P2.4,C  ;STORE CARRY TO PIN P24
        RRC     A       ;MOVE ORIGINAL ACC.3 INTO CY
        MOV     P2.3,C  ;STORE CARRY TO PIN P23
        RRC     A       ;MOVE ORIGINAL ACC.4 INTO CY
        MOV     P2.2,C  ;STORE CARRY TO PIN P22
        RET
```

Solving Combinatorial Logic Equations — ANL, ORL

Virtually all hardware designers are familiar with the problem of solving complex functions using combinatorial logic. The technologies involved may vary greatly, from multiple contact relay logic, vacuum tubes, TTL, or CMOS to more esoteric approaches like fluidics, but in each case the goal is the same: a Boolean (true/false) function is computed on a number of Boolean variables.

AFN-01502A-28

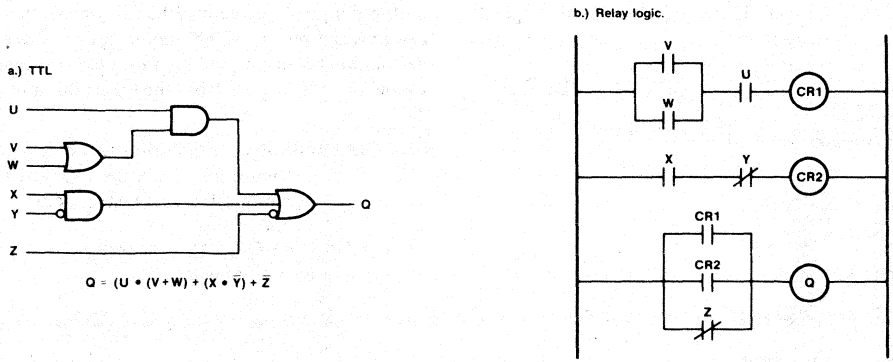


Figure 20. Implementations of Boolean functions

Figure 20 shows the logic diagram for an arbitrary function of six variables named U through Z using standard logic and relay logic symbols. Each is a solution of the equation,

$$Q = (U \cdot (V + W)) + (X \cdot \overline{Y}) + \overline{Z}$$

(While this equation could be reduced using Karnaugh Maps or algebraic techniques, that is not the purpose of this example. Even a minor change to the function equation would require re-reducing from scratch.)

Most digital computers can solve equations of this type with standard word-wide logical instructions and conditional jumps. Still, such software solutions seem somewhat sloppy because of the many paths through the program the computation can take.

Assume U and V are input pins being read by different input ports, W and X are status bits for two peripheral controllers (read as I/O ports), and Y and Z are software flags set or cleared earlier in the program. The end result must be written to an output pin on some third port.

For the sake of comparison we will implement this function with software drawn from three proper subsets of the MCS-51™ instruction set. The first two implementations follow the flow chart shown in Figure 21. Program flow would embark on a route down a test-and-branch tree and leaves either the "True" or "Not True" exit ASAP. These exits then write the output port with the data previously written to the same port with the result bit respectively one or zero.

In the first case, we assume there are no instructions for addressing individual bits other than special flags like the carry. This is typical of many older microprocessors and mainframe computers designed for number-crunching. MCS-51™ mnemonics are used here, though for most other machines the issue would be even further clouded by their use of operation-specific mnemonics like INPUT, OUTPUT, LOAD, STORE, etc., instead of the universal MOV.

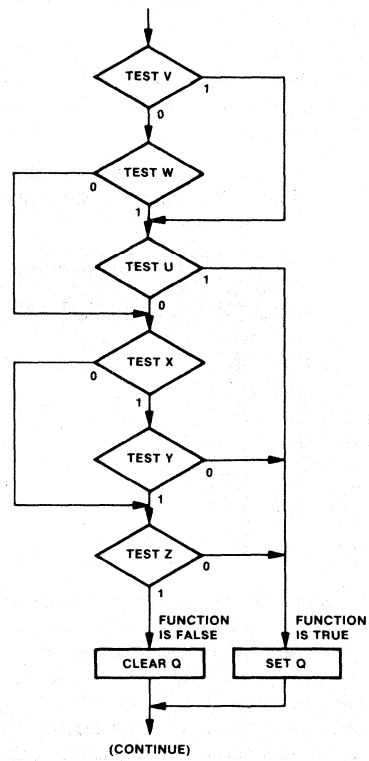


Figure 21. Flow chart for tree-branching logic implementation

AFN-01502A-29

Example 27—Software Solution to Logic Function of Figure 20, Using only Byte-Wide Logical Instructions

```
;BFUNC1 SOLVE A RANDOM LOGIC FUNCTION OF 6
;       VARIABLES BY LOADING AND MASKING THE APPROPRIATE
;       BITS IN THE ACCUMULATOR, THEN EXECUTING CONDITIONAL
;       JUMPS BASED ON ZERO CONDITION.
;       (APPROACH USED BY BYTE-ORIENTED ARCHITECTURES.)
;       BYTE AND MASK VALUES CORRESPOND TO RESPECTIVE
;       BYTE ADDRESS AND BIT POSITION.
;
OUTBUF  EQU     22H     ;OUTPUT PIN STATE MAP
;
TESTV:  MOV     A,P2
        ANL     A,#00000100B
        JNZ     TESTU
        MOV     A,TCON
        ANL     A,#00100000B
        JZ      TESTX
TESTU:  MOV     A,P1
        ANL     A,#00000010B
        JNZ     SETQ
TESTX:  MOV     A,TCON
        ANL     A,#00001000B
        JZ      TESTZ
        MOV     A,20H
        ANL     A,#00000001B
        JZ      SETQ
TESTZ:  MOV     A,21H
        ANL     A,#00000010B
        JZ      SETQ
CLRQ:   MOV     A,OUTBUF
        ANL     A,#11110111B
        JMP     OUTQ
SETQ:   MOV     A,OUTBUF
        ORL     A,#00001000B
OUTQ:   MOV     OUTBUF,A
        MOV     P3,A
```

Cumbersome, to say the least, and error prone. It would be hard to prove the above example worked in all cases without an exhaustive test.

Each move/mask/conditional jump instruction sequence may be replaced by a single bit-test instruction thanks to direct bit addressing. But the algorithm would be equally convoluted.

Example 28—Software Solution to Logic Function of Figure 20, Using only Bit-Test Instructions

```
;BFUNC2 SOLVE A RANDOM LOGIC FUNCTION OF 6
;       VARIABLES BY DIRECTLY POLLING EACH BIT.
;       (APPROACH USING MCS-51 UNIQUE BIT-TEST
;       INSTRUCTION CAPABILITY.)
;       SYMBOLS USED IN LOGIC DIAGRAM ASSIGNED TO
;       CORRESPONDING 8051 BIT ADDRESSES.
;
U       BIT     P1.1
V       BIT     P2.2
W       BIT     TF0
X       BIT     IE1
Y       BIT     20H.0
Z       BIT     21H.1
Q       BIT     P3.3
;
TEST_V: JB      V,TEST_U
        JNB     W,TEST_X
TEST_U: JB      U,SET_Q
TEST_X: JNB     X,TEST_Z
        JNB     Y,SET_Q
TEST_Z: JNB     Z,SET_Q
CLR_Q:  CLR     Q
        JMP     NXTTST
SET_Q:  SETB    Q
NXTTST:                 ;(CONTINUATION OF PROGRAM)
```

A more elegant and efficient 8051 implementation uses the Boolean ANL and ORL functions to generate the output function using straight-line code. These instructions perform the corresponding logical operations between the carry flag ("Boolean Accumulator") and the addressed bit, leaving the result in the carry. Alternate forms of each instruction (specified in the assembly language by placing a slash before the bit name) use the complement of the bit's state as the input operand.

These instructions may be "strung together" to simulate a multiple input logic gate. When finished, the carry flag contains the result, which may be moved directly to the destination or output pin. No flow chart is needed — it is simple to code directly from the logic diagrams in Figure 20.

Example 29—Software Solution to Logic Function of Figure 20, Using the MCS-51 (TM) Unique Logical Instructions on Boolean Variables

```
;BFUNC3 SOLVE A RANDOM LOGIC FUNCTION OF 6
;       VARIABLES USING STRAIGHT-LINE LOGICAL INSTRUCTIONS
;       ON MCS-51 BOOLEAN VARIABLES.
;
        MOV     C,V
        ORL     C,W     ;OUTPUT OF OR GATE
        ANL     C,U     ;OUTPUT OF TOP AND GATE
        MOV     F0,C    ;SAVE INTERMEDIATE STATE
        MOV     C,X
        ANL     C,/Y    ;OUTPUT OF BOTTOM AND GATE
        ORL     C,F0    ;INCLUDE VALUE SAVED ABOVE
        ORL     C,/Z    ;INCLUDE LAST INPUT VARIABLE
        MOV     Q,C     ;OUTPUT COMPUTED RESULT
```

Simplicity itself. Fast, flexible, reliable, easy to design, and easy to debug.

The Boolean features are useful and unique enough to warrant a complete Application Note of their own. Additional uses and ideas are presented in Application Note AP-70, **Using the Intel® MCS-51® Boolean Processing Capabilities,** publication number 121519.

5. ON-CHIP PERIPHERAL FUNCTION OPERATION AND INTERFACING

I/O Ports

The I/O port versatility results from the "quasi-bidirectional" output structure depicted in Figure 22. (This is effectively the structure of ports 1, 2, and 3 for normal I/O operations. On port 0 resistor R2 is disabled except during multiplexed bus operations, providing

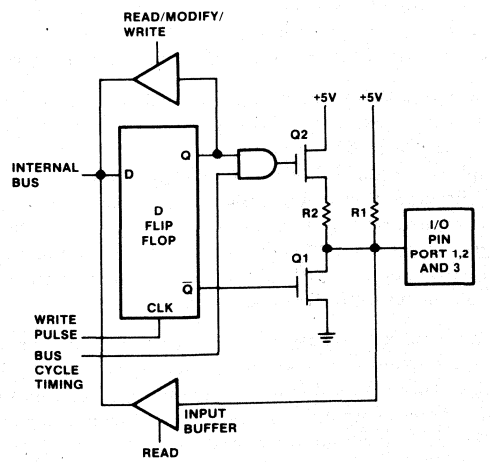


Figure 22. Pseudo-bidirectional I/O port circuitry

AFN-01502A-30

essentially open-collector outputs. For full electrical characteristics see the User's Manual.)

An output latch bit associated with each pin is updated by direct addressing instructions when that port is the destination. The latch state is buffered to the outside world by R1 and Q1, which may drive a standard TTL input. (In TTL terms, Q1 and R1 resemble an open-collector output with a pull-up resistor to Vcc.)

R2 and Q2 represent an "active pull-up" device enabled momentarily when a 0 previously output changes to a 1. This "jerks" the output pin to a 1 level more quickly than the passive pull-up, improving rise-time significantly if the pin is driving a capacitive load. Note that the active pull-up is **only** activated on 0-to-1 transitions at the output latch (unlike the 8048, in which Q2 is activated whenever a 1 is written out).

Operations using an input port or pin as the source operand use the logic level of the pin itself, rather than the output latch contents. This level is affected by both the microcomputer itself and whatever device the pin is connected to externally. The value read is essentially the "OR-tied" function of Q1 and the external device. If the external device is high-impedence, such as a logic gate input or a three state output in the third state, then reading a pin will reflect the logic level previously output. To use a pin for input, the corresponding output latch must be set. The external device may then drive the pin with either a high or low logic signal. Thus the same port may be used as both input and output by writing ones to all pins used as inputs on output operations, and ignoring all pins used as output on an input operation.

In one operand instructions (INC, DEC, DJNZ and the Boolean CPL) the output latch rather than the input pin level is used as the source data. Similarly, two operand instructions using the port as both one source and the destination (ANL, ORL, XRL) use the output latches. This ensures that latch bits corresponding to pins used as inputs will not be cleared in the process of executing these instructions.

The Boolean operation JBC tests the output latch bit, rather than the input pin, in deciding whether or not to jump. Like the byte-wise logical operations, Boolean operations which modify individual pins of a port leave the other bits of the output latch unchanged.

A good example of how these modes may play together may be taken from the host-processor interface expected by an 8243 I/O expander. Even though the 8051 does not include 8048-type instructions for interfacing with an 8243, the parts can be interconnected (Figure 23) and the protocol may be emulated with simple software.

Example 30—Mixing Parallel Output, Input, and Control Strobes on Port 2

```
;IN8243  INPUT DATA FROM AN 8243 I/O EXPANDER
;        CONNECTED TO P23-P20
;        P25 & P24 MIMIC CS*/ & PROG
;        P27-P26 USED AS INPUTS
;        PORT TO BE READ IN ACC

IN8243  ORL     A,#11010000B    ;OUTPUT INSTRUCTION CODE
        MOV     P2,A            ;FALLING EDGE OF PROG
        ORL     P2,#00001111B   ;SET FOR INPUT
        MOV     A,P2            ;READ INPUT DATA
        SETB    P2.4            ;RETURN PROG HIGH
        SETB    P2.5            ;DE-SELECT CHIP
```

Serial Port and Timer applications

Configuring the 8051's Serial Port for a given data rate and protocol requires essentially three short sections of software. On power-up or hardware reset the serial port and timer control words must be initialized to the appropriate values. Additional software is also needed in the transmit routine to load the serial port data register and in the receive routine to unload the data as it arrives.

This is best illustrated through an arbitrary example. Assume the 8051 will communicate with a CRT operating at 2400 baud (bits per second). Each character is transmitted as seven data bits, odd parity, and one stop bit. This results in a character rate of 2400/10=240 characters per second.

For the sake of clarity, the transmit and receive subroutines are driven by simple-minded software status polling code rather than interrupts. (It might help to refer back to Figures 7-9 showing the control word formats.) The serial port must be initialized to 8-bit UART mode (M0, M1=01), enabled to receive all messages (M2=0, REN=1). The flag indicating that the transmit register is free for more data will be artificially set in order to let the output software know the output register is available. This can all be set up with one instruction.

Example 31—Serial Port Mode and Control Bits

```
;SPINIT INITIALIZE SERIAL PORT
;       FOR 8-BIT UART MODE
;       & SET TRANSMIT READY FLAG
;
SPINIT: MOV     SCON,#01010010B
```

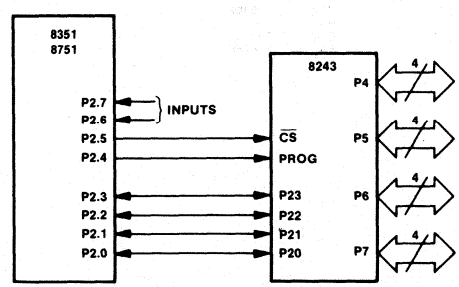


Figure 23. Connecting an 8051 with an 8243 I/O Expander

AFN-01502A-31

Timer 1 will be used in auto-reload mode as a data rate generator. To achieve a data rate of 2400 baud, the timer must divide the 1 MHz internal clock by 32 x (desired data rate):

$$\frac{1 \times 10^6}{(32)\ (2400)}$$

which equals 13.02 rounded down to 13 instruction cycles. The timer must reload the value -13, or 0F3H. (ASM51 will accept both the signed decimal or hexadecimal representations.)

Example 32—Initializing Timer Mode and Control Bits

```
;T1INIT INITIALIZE TIMER 1 FOR
;       AUTO-RELOAD AT 32*2400 HZ.
;       (TO USED AS GATED 16-BIT COUNTER )
;
T1INIT: MOV     TCON,#110100108
        MOV     TH1,#-13
        SETB    TR1
        . . .   . . . . .
```

A simple subroutine to transmit the character passed to it in the accumulator must first compute the parity bit, insert it into the data byte, wait until the transmitter is available, output the character, and return. This is nearly as easy said as done.

Example 33—Code for UART Output, Adding Parity, Transmitter Loading

```
;SP_OUT ADD ODD PARITY TO ACC AND
;       TRANSMIT WHEN SERIAL PORT READY
;
SP_OUT: MOV     C,P
        CPL     C
        MOV     ACC.7,C
        JNB     TI,$
        CLR     TI
        MOV     SBUF,A
        RET
```

A simple minded routine to wait until a character is received, set the carry flag if there is an odd-parity error, and return the masked seven-bit code in the accumulator is equally short.

Example 34—Code for UART Reception and Parity Verification

```
;SP_IN   INPUT NEXT CHARACTER FROM SERIAL PORT
;        SET CARRY IFF ODD-PARITY ERROR.
;
SP_IN:  JNB     RI,$
        CLR     RI
        MOV     A,SBUF
        MOV     C,P
        CPL     C
        ANL     A,#7FH
        RET
```

6. SUMMARY

This Application Note has described the architecture, instruction set, and on-chip peripheral features of the first three members of the MCS-51™ microcomputer family. The examples used throughout were admittedly (and necessarily) very simple. Additional examples and techniques may be found in the MCS-51™ User's Manual and other application notes written for the MCS-48™ and MCS-51™ families.

Since its introduction in 1977, the MCS-48™ family has become the industry standard single-chip microcomputer. The MCS-51™ architecture expands the addressing capabilities and instruction set of its predecessor while ensuring flexibility for the future, and maintaining basic software compatability with the past.

Designers already familiar with the 8048 or 8049 will be able to take with them the education and experience gained from past designs as ever-increasing system performance demands force them to move on to state-of-the-art products. Newcomers will find the power and regularity of the 8051 instruction set an advantage in streamlining both the learning and design processes.

Microcomputer system designers will appreciate the 8051 as basically a single-chip solution to many problems which previously required board-level computers. Designers of real-time control systems will find the high execution speed, on-chip peripherals, and interrupt capabilities vital in meeting the timing constraints of products previously requiring discrete logic designs. And designers of industrial controllers will be able to convert ladder diagrams directly from tested-and-true TTL or relay-logic designs to microcomputer software, thanks to the unique Boolean processing capabilities.

It has not been the intent of this note to gloss over the difficulty of designing microcomputer-based systems. To be sure, the hardware and software design aspects of any new computer system are nontrivial tasks. However, the system speed and level of integration of the MCS-51™ microcomputers, the power and flexibility of the instruction set, and the sophisticated assembler and other support products combine to give both the hardware and software designer as much of a head start on the problem as possible.

Using the Intel MCS-51® Boolean Processing Capabilities

Contents

1. **INTRODUCTION** 9-72

2. **BOOLEAN PROCESSOR OPERATION** 9-72
   - Processing Elements 9-73
   - Direct Bit Addressing 9-74
   - Instruction Set 9-79
   - Simple Instruction Combinations 9-80

3. **BOOLEAN PROCESSOR APPLICATIONS** 9-81
   - Design Example #1 — Bit Permutation 9-82
   - Design Example #2 — Software Serial I/O 9-85
   - Design Example #3 — Combinatorial Logic Equations 9-86
   - Design Example #4 — Automotive Dashboard Functions 9-89
   - Design Example #5 — Complex Control Functions .. 9-94
   - Additional Functions and Uses 9-99

4. **SUMMARY** 9-100

APPENDIX A 9-101

1. INTRODUCTION

The Intel microcontroller family now has three new members the Intel® 8031, 8051, and 8751 single-chip microcomputers. These devices, shown in Figure 1, will allow whole new classes of products to benefit from recent advances in Integrated Electronics. Thanks to Intel's new HMOS® technology, they provide larger program and data memory spaces, more flexible I/O and peripheral capabilities, greater speed, and lower system cost than any previous-generation single-chip microcomputer.

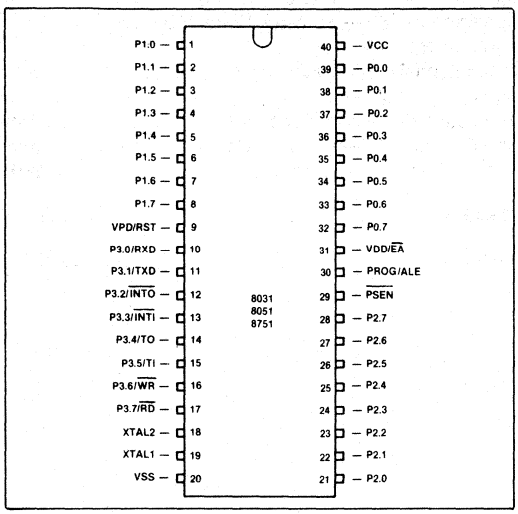


Figure 1. 8051 Family Pinout Diagram.

Table 1 summarizes the quantitative differences between the members of the MCS-48™ and 8051 families. The 8751 contains 4K bytes of EPROM program memory fabricated on-chip, while the 8051 replaces the EPROM with 4K bytes of lower-cost mask-programmed ROM. The 8031 has no program memory on-chip; instead, it accesses up to 64K bytes of program memory from external memory. Otherwise, the three new family members are identical. Throughout this Note, the term "8051" will represent all members of the 8051 Family, unless specifically stated otherwise.

The CPU in each microcomputer is one of the industry's fastest and most efficient for numerical calculations on byte operands. But controllers often deal with bits, not bytes: in the real world, switch contacts can only be open or closed, indicators should be either lit or dark, motors are either turned on or off, and so forth. For such control situations the most significant aspect of the MCS-51™ architecture is its complete hardware support for one-bit, or *Boolean* variables (named in honor of Mathematician George Boole) as a separate data type.

The 8051 incorporates a number of special features which support the direct manipulation and testing of individual bits and allow the use of single-bit variables in performing logical operations. Taken together, these features are referred to as the MCS-51™ *Boolean Processor*. While the bit-processing capabilities alone would be adequate to solve many control applications, their true power comes when they are used in conjunction with the microcomputer's byte-processing and numerical capabilities.

Many concepts embodied by the Boolean Processor will certainly be new even to experienced microcomputer system designers. The purpose of this Application Note is to explain these concepts and show how they are used. It is assumed the reader has read Application Note AP-69, **An Introduction to the Intel® MCS-51™ Single-Chip Microcomputer Family,** publication number 121518, or has been exposed to Intel's single-chip microcomputer product lines.

For detailed information on these parts refer to the **Intel MCS-51™ Family User's Manual**, publication number 121517. The instruction set, assembly language, and use of the 8051 assembler (ASM51) are further described in the **MCS-51™ Macro Assembler User's Guide**, publication number 9800937.

2. BOOLEAN PROCESSOR OPERATION

The Boolean Processing capabilities of the 8051 are based on concepts which have been around for some time. Digital computer systems of widely varying designs all have four functional elements in common (Figure 2):

Table 1. Features of Intel's Single-chip Microcomputers.

EPROM Program Memory	ROM Program Memory	External Program Memory	Program Memory (Int/Max)	Data Memory (Bytes)	Instr. Cycle Time	Input/ Output Pins	Interrupt Sources	Reg. Banks
—	8021	—	1K 1K	64	10 μSec	21	0	1
—	8022	—	2K 2K	64	10 μSec	28	2	1
8748	8048	8035	1K 4K	64	2.5 μSec	27	2	2
—	8049	8039	2K 4K	128	1.36μSec	27	2	2
8751	8051	8031	4K 64K	128	1.0 μSec	32	5	4

01489A-03

- a central processor (CPU) with the control, timing, and logic circuits needed to execute stored instructions;
- a memory to store the sequence of instructions making up a program or algorithm;
- data memory to store variables used by the program; and
- some means of communicating with the outside world.

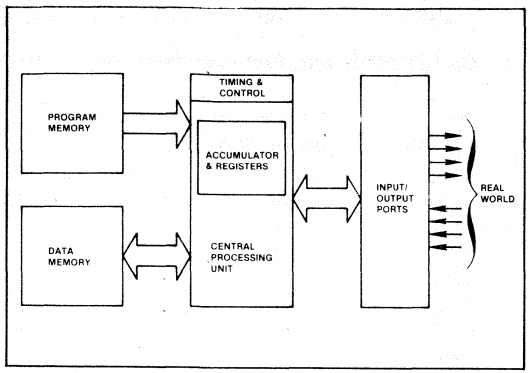


Figure 2. Block Diagram for Abstract Digital Computer.

The CPU usually includes one or more accumulators or special registers for computing or storing values during program execution. The instruction set of such a processor generally includes, at a minimum, operation classes to perform arithmetic or logical functions on program variables, move variables from one place to another, cause program execution to jump or conditionally branch based on register or variable states, and instructions to call and return from subroutines. The program and data memory functions sometimes share a single memory space, but this is not always the case. When the address spaces are separated, program and data memory need not even have the same basic word width.

A digital computer's flexibility comes in part from combining simple fast operations to produce more complex (albeit slower) ones, which in turn link together eventually solving the problem at hand. A four-bit CPU executing multiple precision subroutines can, for example, perform 64-bit addition and subtraction. The subroutines could in turn be building blocks for floating-point multiplication and division routines. Eventually, the four-bit CPU can simulate a far more complex "virtual" machine.

In fact, *any* digital computer with the above four functional elements can (given time) complete *any* algorithm (though the proverbial room full of chimpanzees at word processors might first re-create Shakespeare's classics and this Application Note)! This fact offers little consolation to product designers who want programs to run as quickly as possible. By definition, a real-time control algorithm *must* proceed quickly enough to meet the preordained speed constraints of other equipment.

One of the factors determining how long it will take a microcomputer to complete a given chore is the number of instructions it must execute. What makes a given computer architecture particularly well-or poorly-suited for a class of problems is how well its instruction set matches the tasks to be performed. The better the "primative" operations correspond to the steps taken by the control algorithm, the lower the number of instructions needed, and the quicker the program will run. All else being equal, a CPU supporting 64-bit arithmetic directly could clearly perform floating-point math faster than a machine bogged-down by multiple-precision subroutines. In the same way, direct support for bit manipulation naturally leads to more efficient programs handling the binary input and output conditions inherent in digital control problems.

Processing Elements

The introduction stated that the 8051's bit-handling capabilities alone would be sufficient to solve some control applications. Let's see how the four basic elements of a digital computer - a CPU with associated registers, program memory, addressable data RAM, and I/O capability - relate to Boolean variables.

CPU. The 8051 CPU incorporates special logic devoted to executing several bit-wide operations. All told, there are 17 such instructions, all listed in Table 2. Not shown are 94 other (mostly byte-oriented) 8051 instructions.

Program Memory. Bit-processing instructions are fetched from the same program memory as other arithmetic and logical operations. In addition to the instructions of Table 2, several sophisticated program control features like multiple addressing modes, subroutine nesting, and a two-level interrupt structure are useful in structuring Boolean Processor-based programs.

Boolean instructions are one, two, or three bytes long, depending on what function they perform. Those involving only the carry flag have either a single-byte opcode or an opcode followed by a conditional-branch destination byte (Figure 3.a). The more general instructions add a "direct address" byte after the opcode to specify the bit affected, yielding two or three byte encodings (Figure 3.b). Though this format allows potentially 256 directly addressable bit locations, not all of them are implemented in the 8051 family.

01489A-04

Table 2. MCS-51™ Boolean Processing Instruction Subset.

Mnemonic		Description	Byte	Cyc
SETB	C	Set Carry flag	1	1
SETB	bit	Set direct Bit	2	1
CLR	C	Clear Carry flag	1	1
CLR	bit	Clear direct bit	2	1
CPL	C	Complement Carry flag	1	1
CPL	bit	Complement direct bit	2	1
MOV	C,bit	Move direct bit to Carry flag	2	1
MOV	bit,C	Move Carry flag to direct bit	2	2
ANL	C,bit	AND direct bit to Carry flag	2	2
ANL	C, bit	AND complement of direct bit to Carry flag	2	2
ORL	C,bit	OR direct bit to Carry flag	2	2
ORL	C, bit	OR complement of direct bit to Carry flag	2	2
JC	rel	Jump if Carry is flag is set	2	2
JNC	rel	Jump if No Carry flag	2	2
JB	bit,rel	Jump if direct Bit set	3	2
JNB	bit,rel	Jump if direct Bit Not set	3	2
JBC	bit,rel	Jump if direct Bit is set & Clear bit	3	2

Address mode abbreviations:

C — Carry flag.

bit — 128 software flags, any I/O pin, control or status bit

rel — All conditional jumps include an 8-bit offset byte. Range is +127 -128 bytes relative to first byte of the following instruction.

Data Memory. The instructions in Figure 3.b can operate directly upon 144 general purpose bits forming the Boolean processor "RAM." These bits can be used as sofware flags or to store program variables. Two operand instructions use the CPU's carry flag ("C") as a special one-bit register; in a sense, the carry is a "Boolean accumulator" for logical operations and data transfers.

Input/Output. All 32 I/O pins can be addressed as individual inputs, outputs, or both, in any combination. Any pin can be a control strobe output, status (Test) input, or serial I/O link implemented via software. An additional 33 individually addressable bits reconfigure, control, and monitor the status of the CPU and all on-chip peripheral functions (timer/counters, serial port modes, interrupt logic, and so forth).

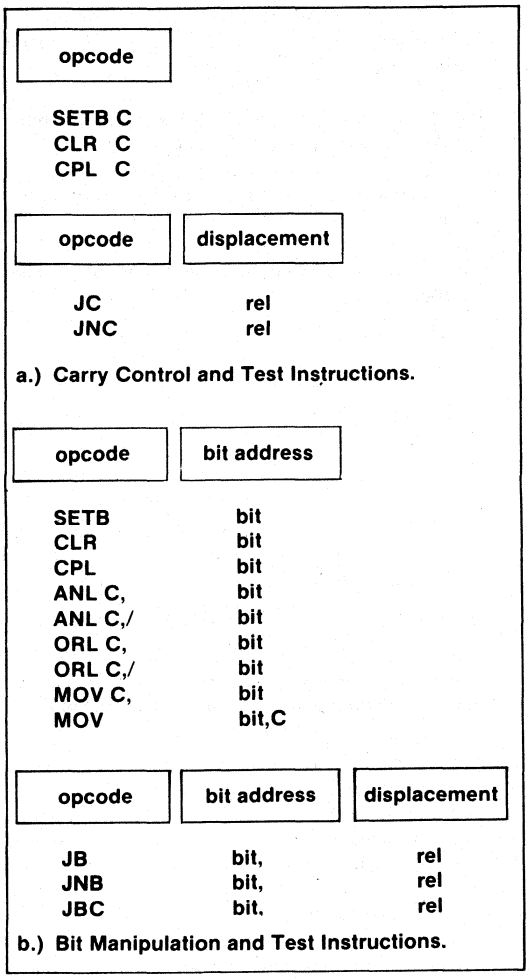


Figure 3. Bit Addressing Instruction Formats.

Direct Bit Addressing

The most significant bit of the direct address byte selects one of two groups of bits. Values between 0 and 127 (00H and 7FH) define bits in a block of 32 bytes of on-chip RAM, between RAM addresses 20H and 2FH (Figure 4.a). They are numbered consecutively from the lowest-order byte's lowest-order bit through the highest-order byte's highest-order bit.

Bit addresses between 128 and 255(80H and 0FFH) correspond to bits in a number of special registers, mostly used for I/O or peripheral control. These positions are numbered with a different scheme than RAM: the five high-order address bits match those of the register's own address, while the three low-order bits identify the bit position within that register (Figure 4.b).

01489A-05

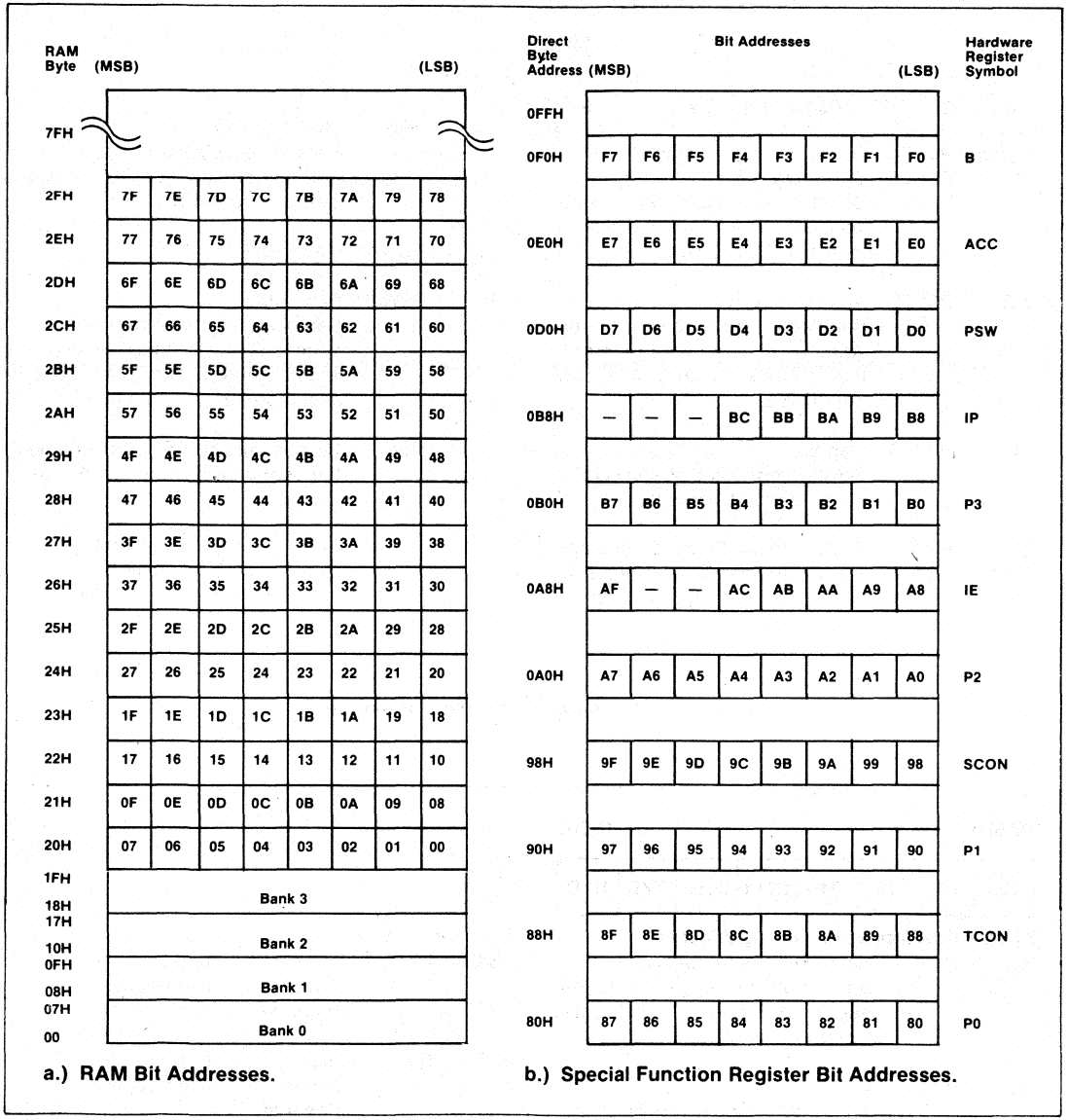


Figure 4. Bit Address Maps.

Notice the column labeled "Symbol" in Figure 5. Bits with special meanings in the PSW and other registers have corresponding symbolic names. General-purpose (as opposed to carry-specific) instructions may access the carry like any other bit by using the mnemonic CY in place of C. P0, P1, P2, and P3 are the 8051's four I/O ports; secondary functions assigned to each of the eight pins of P3 are shown in Figure 6.

Figure 7 shows the last four bit addressable registers. TCON (Timer Control) and SCON (Serial port Control) control and monitor the corresponding peripherals, while IE (Interrupt Enable) and IP (Interrupt Priority) enable and prioritize the five hardware interrupt sources. Like the reserved hardware register addresses, the five bits not implemented in IE and IP should not be accessed; they can *not* be used as software flags.

01489A-06

(MSB) (LSB)

CY	AC	F0	RS1	RS0	OV	—	P

Symbol	Position	Name and significance
CY	PSW.7	Carry flag. Set/cleared by hardware or software during certain arithmetic and logical instructions.
AC	PSW.6	Auxiliary Carry flag. Set/cleared by hardware during addition or subtraction instructions to indicate carry or borrow out of bit 3.
F0	PSW.5	Flag 0. Set/cleared/tested by software as a user-defined status flag.
RS1	PSW.4	Register bank Select control bits
RS0	PSW.3	1 & 0. Set/cleared by software to determine working register bank (see Note).
OV	PSW.2	Overflow flag. Set/cleared by hardware during arithmetic instructions to indicate overflow conditions.
—	PSW.1	(reserved)
P	PSW.0	Parity flag. Set/cleared by hardware each instruction cycle to indicate an odd/-even number of "one" bits in the accumulator, i.e., even parity.

Note - the contents of (RS1, RS0) enable the working register banks as follows:

(0,0) - Bank 0 (00H-07H)
(0,1) - Bank 1 (08H-0FH)
(1,0) - Bank 2 (10H-17H)
(1,1) - Bank 3 (18H-1FH)

Figure 5. PSW - Program Status Word organization.

(MSB) (LSB)

RD	WR	T1	T0	INT1	INT0	TXD	RXD

Symbol	Position	Name and significance
RD	P3.7	Read data control output. Active low pulse generated by hardware when external data memory is read.
WR	P3.6	Write data control output. Active low pulse generated by hardware when external data memory is written.
T1	P3.5	Timer/counter 1 external input or test pin.
T0	P3.4	Timer/counter 0 external input or test pin.
INT1	P3.3	Interrupt 1 input pin. Low-level or falling-edge triggered.
INT0	P3.2	Interrupt 0 input pin. Low-level or falling-edge triggered.
TXD	P3.1	Transmit Data pin for serial port in UART mode. Clock output in shift register mode.
RXD	P3.0	Receive Data pin for serial port in UART mode. Data I/O pin in shift register mode.

Figure 6. P3 - Alternate I/O Functions of Port 3.

01489A-07

(MSB) (LSB)

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Symbol Position Name and significance

TF1 TCON.7 Timer 1 overflow Flag. Set by hardware on timer/counter overflow. Cleared when interrupt processed.

TR1 TCON.6 Timer 1 Run control bit. Set/cleared by software to turn timer/counter on/off.

TF0 TCON.5 Timer 0 overflow Flag. Set by hardware on timer/counter overflow. Cleared when interrupt processed.

TR0 TCON.4 Timer 0 Run control bit. Set/cleared by software to turn timer/counter on/off.

IE1 TCON.3 Interrupt 1 Edge flag. Set by hardware when external interrupt edge detected. Cleared when interrupt processed.

IT1 TCON.2 Interrupt 1 Type control bit. Set/cleared by software to specify falling edge/low level triggered external interrupts.

IE0 TCON.1 Interrupt 0 Edge flag. Set by hardware when external interrupt edge detected. Cleared when interrupt processed.

IT0 TCON.0 Interrupt 0 Type control bit. Set/cleared by softrware to specify falling edge/low level triggered external interrupts.

a.) TCON - Timer/Counter Control/status register.

(MSB) (LSB)

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

Symbol Position Name and significance

SM0 SCON.7 Serial port Mode control bit 0. Set/cleared by software (see note).

SM1 SCON.6 Serial port Mode control bit 1. Set/cleared by software (see note).

SM2 SCON.5 Serial port Mode control bit 2. Set by software to disable reception of frames for which bit 8 is zero.

REN SCON.4 Receiver Enable control bit. Set/cleared by software to enable/disable serial data reception.

TB8 SCON.3 Transmit Bit 8. Set/cleared by hardware to determine state of ninth data bit transmitted in 9-bit UART mode.

RB8 SCON.2 Receive Bit 8. Set/cleared by hardware to indicate state of ninth data bit received.

TI SCON.1 Transmit Interrupt flag. Set by hardware when byte transmitted. Cleared by software after servicing.

RI SCON.0 Receive Interrupt flag. Set by hardware when byte received. Cleared by software after servicing.

Note - the state of (SM0,SM1) selects:
(0,0) - Shift register I/O expansion.
(0,1) - 8 bit UART, variable data rate.
(1,0) - 9 bit UART, fixed data rate.
(1,1) - 9 bit UART, variable data rate.

b.) SCON - Serial Port Control/status register.

Figure 7. Peripheral Configuration Registers.

01489A-08

(MSB) (LSB)

EA	—	—	ES	ET1	EX1	ET1	EX0

Symbol	Position	Name and significance
EA	IE.7	Enable All control bit. Cleared by software to disable all interrupts, independent of the state of IE.4 - IE.0.
—	IE.6	(reserved)
—	IE.5	
ES	IE.4	Enable Serial port control bit. Set/cleared by software to enable/ disable interrupts from TI or RI flags.
ET1	IE.3	Enable Timer 1 control bit. Set/cleared by software to enable/ disable interrupts from timer/counter 1.
EX1	IE.2	Enable External interrupt 1 control bit. Set/cleared by software to enable/disable interrupts from INT1.
ET0	IE.1	Enable Timer 0 control bit. Set/cleared by software to enable/ disable interrupts from timer/counter 0.
EX0	IE.0	Enable External interrupt 0 control bit. Set/cleared by software to enable/disable interrupts from INT0.

c.) IE - Interrupt Enable Register.

(MSB) (LSB)

—	—	—	PS	PT1	PX1	PT0	PX0

Symbol	Position	Name and significance
—	IP.7	(reserved)
—	IP.6	(reserved)
—	IP.5	(reserved)
PS	IP.4	Serial port Priority control bit. Set/cleared by software to specify high/low priority interrupts for Serial port.
PT1	IP.3	Timer 1 Priority control bit. Set/cleared by software to specify high/low priority interrupts for timer/counter 1.
PX1	IP.2	External interrupt 1 Priority control bit. Set/cleared by software to specify high/low priority interrupts for INT1.
PT0	IP.1	Timer 0 Priority control bit. Set/cleared by software to specify high/low priority interrupts for timer/counter 0.
PX0	IP.0	External interrupt 0 Priority control bit. Set/cleared by software to specify high/low priority interrupts for INT0.

d.) IP - Interrupt Priority Control Register.

Figure 7. (continued)

Addressable Register Set. There are 20 special function registers in the 8051, but the advantages of bit addressing only relate to the 11 described below. Five potentially bit-addressable register addresses (0C0H, 0C8H, 0D8H, 0E8H, & 0F8H) are being reserved for possible future expansion in microcomputers based on the MCS-51™ architecture. Reading or writing non-existent registers in the 8051 series is pointless, and may cause unpredictable results. Byte-wide logical operations can be used to manipulate bits in all *non*-bit addressable registers and RAM.

The accumulator and B registers (A and B) are normally involved in byte-wide arithmetic, but their individual bits can also be used as 16 general software flags. Added with the 128 flags in RAM, this gives 144 general purpose variables for bit-intensive programs. The program status word (PSW) in Figure 5 is a collection of flags and machine status bits including the carry flag itself. Byte operations acting on the PSW can therefore affect the carry.

01489A-09

Instruction Set

Having looked at the bit variables available to the Boolean Processor, we will now look at the four classes of instructions that manipulate these bits. It may be helpful to refer back to Table 2 while reading this section.

State Control. Addressable bits or flags may be set, cleared, or logically complemented in one instruction cycle with the two-byte instructions SETB, CLR, and CPL. (The "B" affixed to SETB distinguishes it from the assembler "SET" directive used for symbol definition.) SETB and CLR are analogous to loading a bit with a constant: 1 or 0. Single byte versions perform the same three operations on the carry.

The MCS-51™ assembly language specifies a bit address in any of three ways:

- by a number or expression corresponding to the direct bit address (0-255);
- by the name or address of the register containing the bit, the *dot operator* symbol (a period: "."), and the bit's position in the register (7-0);
- in the case of control and status registers, by the predefined assembler symbols listed in the first columns of Figures 5-7.

Bits may also be given user-defined names with the assembler "BIT" directive and any of the above techniques. For example, bit 5 of the PSW may be cleared by any of the four instructions.

```
USR_FLG BIT   PSW.5     ; User Symbol Definition
;         ...   .....
          CLR   0D5H    ; Absolute Addressing
          CLR   PSW.5   ; Use of Dot Operator
          CLR   F0      ; Pre-Defined Assembler
                        ;              Symbol
          CLR   USR_FLG ; User-Defined Symbol
```

Data Transfers. The two-byte MOV instructions can transport any addressable bit to the carry in one cycle, or copy the carry to the bit in two cycles. A bit can be moved between two arbitrary locations via the carry by combining the two instructions. (If necessary, push and pop the PSW to preserve the previous contents of the carry.) These instructions can replace the multi-instruction sequence of Figure 8, a program structure appearing in controller applications whenever flags or outputs are conditionally switched on or off.

Logical Operations. Four instructions perform the logical-AND and logical-OR operations between the carry and another bit, and leave the results in the carry. The instruction mnemonics are ANL and ORL; the absence or presence of a

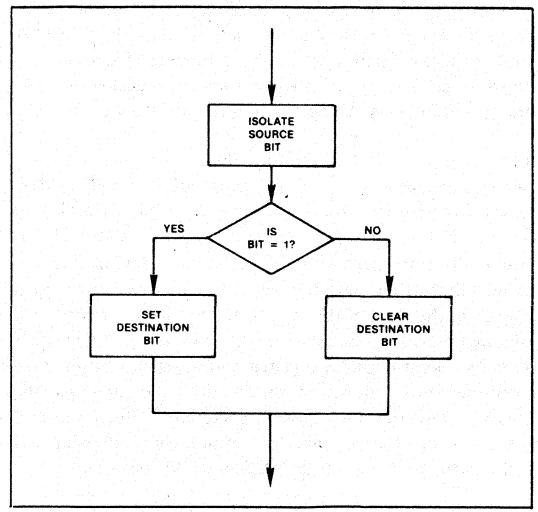


Figure 8. Bit Transfer Instruction Operation.

slash mark ("/") before the source operand indicates whether to use the positive-logic value or the logical complement of the addressed bit. (The source operand itself is never affected.)

Bit-test Instructions. The conditional jump instructions "JC rel" (Jump on Carry) and "JNC rel" (Jump on Not Carry) test the state of the carry flag, branching if it is a one or zero, respectively. (The letters "rel" denote relative code addressing.) The three-byte instructions "JB bit, rel" and "JNB bit,rel" (Jump on Bit and Jump on Not Bit) test the state of *any* addressable bit in a similar manner. A fifth instruction combines the Jump on Bit and Clear operations. "JBC bit,rel" conditionally branches to the indicated address, then clears the bit in the same two cycle instruction. This operation is the same as the MCS-48™ "JTF" instructions.

All 8051 conditional jump instructions use program counter-relative addressing, and all execute in two cycles. The last instruction byte encodes a signed displacement ranging from -128 to +127. During execution, the CPU adds this value to the incremented program counter to produce the jump destination. Put another way, a conditional jump to the immediately following instruction would encode 00H in the offset byte.

A section of program or subroutine written using only relative jumps to nearby addresses will have the same machine code independent of the code's location. An assembled routine may be repositioned anywhere in memory, even crossing memory page boundaries, without having to modify the program or recompute destination addresses. To facilitate this flexibility, there is an unconditional "Short Jump" (SJMP) which uses relative addressing as well. Since a pro-

01489A-10

grammer would have quite a chore trying to compute relative offset values from one instruction to another, ASM51 automatically computes the displacement needed given only the destination address or label. An error message will alert the programmer if the destination is "out of range."

(The so-called "Bit Test" instructions implemented on many other microprocessors simply perform the logical-AND operation between a byte variable and a constant mask, and set or clear a zero flag depending on the result. This is essentially equivalent to the 8051 "MOV C,bit" instruction. A second instruction is then needed to conditionally branch based on the state of the zero flag. This does *not* constitute abstract bit-addressing in the MCS-51™ sense. A flag exists only as a field within a register; to reference a bit the programmer must know and specify both the encompassing register and the bit's position therein. This constraint severely limits the flexibility of symbolic bit addressing and reduces the machine's code-efficiency and speed.)

Interaction with Other Instructions. The carry flag is also affected by the instructions listed in Table 3. It can be rotated through the accumulator, and altered as a side effect of arithmetic instructions. Refer to the User's Manual for details on how these instructions operate.

Simple Instruction Combinations

By combining general purpose bit operations with certain addressable bits, one can "custom build" several hundred useful instructions. All eight bits of the PSW can be tested directly with conditional jump instructions to monitor (among other things) parity and overflow status. Programmers can take advantage of 128 software flags to keep track of operating modes, resource usage, and so forth.

The Boolean instructions are also the most efficient way to control or reconfigure peripheral and I/O registers. All 32 I/O lines become "test pins," for example, tested by conditional jump instructions. Any output pin can be toggled (complemented) in a single instruction cycle. Setting or clearing the Timer Run flags (TR0 and TR1) turn the timer/counters on or off; polling the same flags elsewhere lets the program determine if a timer is running. The respective overflow flags (TF0 and TF1) can be tested to determine when the desired period or count has elapsed, then cleared in preparation for the next repetition. (For the record, these bits are all part of the TCON register, Figure 7.a. Thanks to symbolic bit addressing, the programmer only needs to remember the mnemonic associated with each function. In other words, don't bother memorizing control word layouts.)

In the MCS-48® family, instructions corresponding to some of the above functions require specific opcodes. Ten different opcodes serve to clear/complement the software flags F0 and F1, enable/disable each interrupt, and start/stop the timer. In the 8051 instruction set, just three opcodes (SETB,

Table 3. Other Instructions Affecting the Carry Flag.

Mnemonic	Description	Byte	Cyc
ADD A,Rn	Add register to Accumulator	1	1
ADD A,direct	Add direct byte to Accumulator	2	1
ADD A,@Ri	Add indirect RAM to Accumulator	1	1
ADD A,#data	Add immediate data to Accumulator	2	1
ADDC A,Rn	Add register to Accumulator with Carry flag	1	1
ADDC A, direct	Add direct byte to Accumulator with Carry flag	2	1
ADDC A,@Ri	Add indirect RAM to Accumulator with Carry flag	1	1
ADDC A,#data	Add immediate data to Acc with Carry flag	2	1
SUBB A,Rn	Subtract register from Accumulator with borrow	1	1
SUBB A,direct	Subtract direct byte from Acc with borrow	2	1
SUBB A,@Ri	Subtract indirect RAM from Acc with borrow	1	1
SUBB A,#data	Subtract immediate data from Acc with borrow	2	1
MUL AB	Multiply A & B	1	4
DIV AB	Divide A by B	1	4
DA A	Decimal Adjust Accumulator	1	1
RLC A	Rotate Accumulator Left through the Carry flag	1	1
RRC A	Rotate Accumulator Right through Carry flag	1	1
CJNE A,direct,rel	Compare direct byte to Acc & Jump if Not Equal	3	2
CJNE A,#data,rel	Compare immediate to Acc & Jump if Not Equal	3	2
CJNE Rn,#data,rel	Compare immed to register & Jump if Not Equal	3	2
CJNE @Ri,#data,rel	Compare immed to indirect & Jump if Not Equal	3	2

CLR, CPL) with a direct bit address appended perform the same functions. Two test instructions (JB and JNB) can be combined with bit addresses to test the software flags, the 8048 I/O pins T0, T1, and INT, and the eight accumulator bits, replacing 15 more different instructions.

Table 4.a shows how 8051 programs implement software flag and machine control functions associated with special using awkward sequences of other basic operations. As mentioned earlier, any CPU can solve any problem given enough time.

Quantitatively, the differences between a solution allowed by the 8051 and those required by previous architectures are numerous. What the 8051 Family buys you is a faster, cleaner, lower-cost solution to microcontroller applications.

The opcode space freed by condensing many specific 8048

Table 4.a. Contrasting 8048 and 8051 Bit Control and Testing Instructions.

8048 Instruction	Bytes	Cycles	uSec	8x51 Instruction	Bytes	Cycles & uSec
Flag Control						
CLR C	1	1	2.5	CLR C	1	1
CPL F0	1	1	2.5	CPL F0	2	1
Flag Testing						
JNC offset	2	2	5.0	JNC rel	2	2
JF0 offset	2	2	5.0	JB F0.rel	3	2
JB7 offset	2	2	5.0	JB ACC.7.rel	3	2
Peripheral Polling						
JT0 offset	2	2	5.0	JB T0.rel	3	2
JNI offset	2	2	5.0	JNB INT0.rel	3	2
JTF offset	2	2	5.0	JBC TF0.rel	3	2
Machine and Peripheral Control						
STRT T	1	1	2.5	SETB TR0	2	1
EN I	1	1	2.5	SETB EX0	2	1
DIS TCNTI	1	1	2.5	CLR ET0	2	1

Table 4.b. Replacing 8048 instruction sequences with single 8x51 instructions.

8048 Instructions	Bytes	Cycles	uSec	8051 Instructions	Bytes	Cycles & uSec
Flag Control Set carry: CLR C CPL C	= 2	2	5.0	SETB C	1	1
Set Software Flag: CLR F0 CPL F0	= 2	2	5.0	SETB F0	2	1

opcodes in the 8048. In every case the MCS-51™ solution requires the same number of machine cycles, and executes 2.5 times faster.

3. BOOLEAN PROCESSOR APPLICATIONS

So what? Then what does all this buy you?

Qualitatively, nothing. All the same capabilities *could* be (and often have been) implemented on other machines instructions into a few general operations has been used to add new functionality to the MCS-51™ architecture - both for byte and bit operations. 144 software flags replace the 8048's two. These flags (and the carry) may be directly set, not just cleared and complemented, and all can be tested for either state, not just one. Operating mode bits previously inaccessible may be read, tested, or saved. Situations where the 8051 instruction set provides new capabilities are contrasted with 8048 instruction sequences in Table 4.b. Here the 8051 speed advantage ranges from 5x to 15x!

01489A-12

Table 4b. (Continued)

8048 Instructions		Bytes	Cycles	uSec	8x51 Instructions		Bytes	Cycles & uSec
Turn Off Output Pin:								
ANL P1,#0FBH	=	2	2	5.0	CLR	P1.2	2	1
Complement Output Pin:								
IN A,P1								
XRL A,#04H								
OUTL P1,A	=	4	6	15.0	CPL	P1.2	2	1
Clear Flag in RAM:								
MOV R0,#FLGADR								
MOV A,@R0								
ANL A,#FLGMASK								
MOV @R0,A	=	6	6	15.0	CLR	USER_FLG	2	1
Flag Testing								
Jump if Software Flag is 0:								
JF0 $+4								
JMP offset	=	4	4	10.0	JNB	F0,rel	3	2
Jump if Accumulator bit is 0:								
CPL A								
JB7 offset								
CPL A	=	4	4	10.0	JNB	ACC.7,rel	3	2
Peripheral Polling								
Test if Input Pin is Grounded:								
IN A,P1								
CPL A								
JB3 offset	=	4	5	12.5	JNB	P1.3,rel	3	2
Test if Interrupt Pin is High:								
JNI $+4								
JMP offset	=	4	4	10.0	JB	INT0,rel	3	2

Combining Boolean and byte-wide instructions can produce great synergy. An MCS-51™ based application will prove to be:

- simpler to write since the architecture correlates more closely with the problems being solved;
- easier to debug because more individual instructions have no unexpected or undesirable side-effects;
- more byte efficient due to direct bit addressing and program counter relative branching;
- faster running because fewer bytes of instruction need to be fetched and fewer conditional jumps are processed;
- lower cost because of the high level of system-intergration within one component.

These rather unabashed claims of excellence shall not go unsubstantiated. The rest of this chapter examines less trivial tasks simplified by the Boolean processor. The first three compare the 8051 with other microprocessors; the last two go into 8051-based system designs in much greater depth.

Design Example #1 - Bit Permutation

First off, we'll use the bit-transfer instructions to permute a lengthy pattern of bits.

A steadily increasing number of data communication products use encoding methods to protect the security of sensitive information. By law, interstate financial transactions involving the Federal banking system must be transmitted using the Federal Information Processing *Data Encryption Standard* (DES).

Basically, the DES combines eight bytes of "plaintext" data (in binary, ASCII, or any other format) with a 56-bit "key", producing a 64-bit encrypted value for transmission. At the receiving end the same algorithm is applied to the incoming data using the same key, reproducing the original eight byte message. The algorithm used for these permutations is fixed; different user-defined keys ensure data privacy.

01489A-13

It is not the purpose of this note to describe the DES in any detail. Suffice it to say that encryption/decryption is a long, iterative process consisting of rotations, exclusive -OR operations, function table look-ups, and an extensive (and quite bizarre) sequence of bit permutation, packing, and unpacking steps. (For further details refer to the June 21, 1979 issue of **Electronics** magazine.) The bit manipulation steps are included, it is rumored, to impede a general purpose digital supercomputer trying to "break" the code. Any algorithm implementing the DES with previous generation microprocessors would spend virtually all of its time diddling bits.

The bit manipulation performed is typified by the Key Schedule Calculation represented in Figure 9. This step is repeated 16 times for each key used in the course of a transmission. In essence, a seven-byte, 56-bit "Shifted Key Buffer" is transformed into an eight-byte, "Permutation Buffer" without altering the shifted Key. The arrows in Figure 9 indicate a few of the translation steps. Only six bits of each byte of the Permutation Buffer are used; the two high-order bits of each byte are cleared. This means only 48 of the 56 Shifted Key Buffer bits are used in any one iteration.

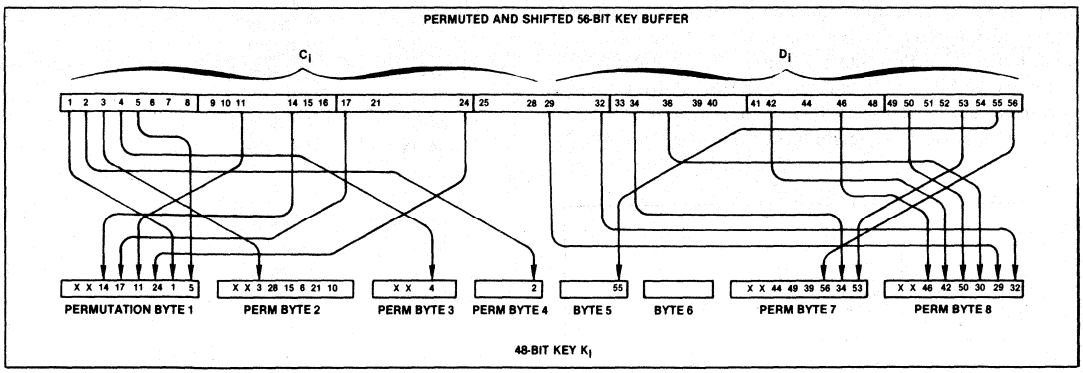


Figure 9. DES Key Schedule Transformation.

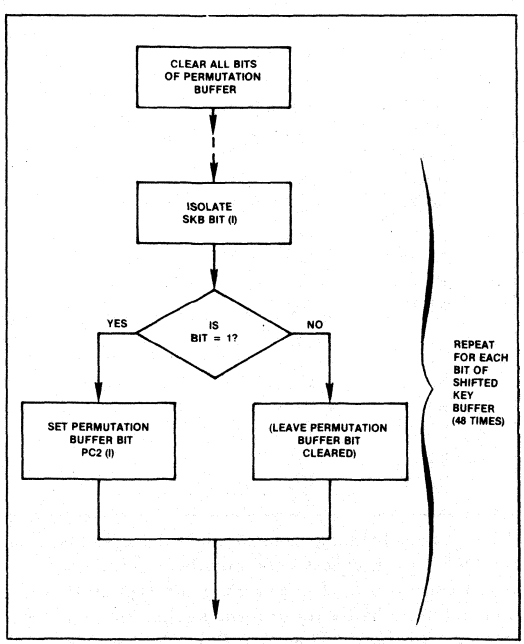


Figure 10.a. Flowchart for Key permutation attempted with a byte processor.

Different microprocessor architectures would best implement this type of permutation in different ways. Most approaches would share the steps of Figure 10.a:

- Initialize the Permutation Buffer to default state (ones or zeroes);
- Isolate the state of a bit of a byte from the Key Buffer. Depending on the CPU, this might be accomplished by rotating a word of the Key Buffer through a carry flag or testing a bit in memory or an accumulator against a mask byte;
- Perform a conditional jump based on the carry or zero flag if the Permutation Buffer default state is correct;
- Otherwise reverse the corresponding bit in the permutation buffer with logical operations and mask bytes.

Each step above may require several instructions. The last three steps must be repeated for all 48 bits. Most microprocessors would spend 300 to 3,000 microseconds on each of the 16 iterations.

Notice, though, that this flow chart looks a lot like Figure 8. The Boolean Processor can permute bits by simply moving

01489A-14

them from the source to the carry to the destination—a total of two instructions taking four bytes and three microseconds per bit. Assume the Shifted Key Buffer and Permutation Buffer both reside in bit-addressable RAM, with the bits of the former assigned symbolic names SKB_1, SKB_2, . . . SKB_56, and that the bytes of the latter are named PB_1, . . . PB_8. Then working from Figure 9, the software for the permutation algorithm would be that of Example 1.a. The total routine length would be 192 bytes, requiring 144 microseconds.

The algorithm of Figure 10.b is just slightly more efficient in this time-critical application and illustrates the synergy of an integrated byte and bit processor. The bits needed for each byte of the Permutation Buffer are assimilated by loading each bit into the carry (1 usec.) and shifting it into the accumulator (1 usec.). Each byte is stored in RAM when completed. Forty-eight bits thus need a total of 112 instructions, some of which are listed in Example 1.b.

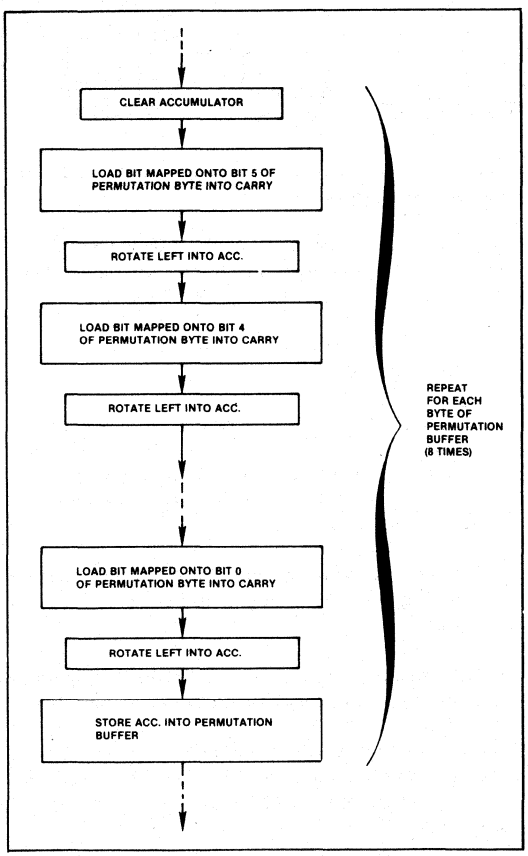


Figure 10.b. DES Key Permutation with Boolean Processor.

Worst-case execution time would be 112 microseconds, since each instruction takes a single cycle. Routine length would also decrease, to 168 bytes. (Actually, in the context of the complete encryption algorithm, each permuted byte would be processed as soon as it is assimilated—saving memory and cutting execution time by another 8 usec.)

Example 1. DES Key Permutation Software.

a.) "Brute Force" technique.

```
MOV     C,SKB_1
MOV     PB_1.1,C
MOV     C,SKB_2
MOV     PB_4.0,C
MOV     C,SKB_3
MOV     PB_2.5,C
MOV     C,SKB_4
MOV     PB_1.0,C
...     .....
...     .....
MOV     C,SKB_55
MOV     PB_5.0,C
MOV     C,SKB_56
MOV     PB_7.2,C
```

b.) Using Accumulator to Collect Bits.

```
CLR     A
MOV     C,SKB_14
RLC     A
MOV     C,SKB_17
RLC     A
MOV     C,SKB_11
RLC     A
MOV     C,SKB_24
RLC     A
MOV     C,SKB_1
RLC     A
MOV     C,SKB_5
RLC     A
MOV     PB_1,A
...     .....
...     .....
MOV     C,SKB_29
RLC     A
MOV     C,SKB_32
RLC     A
MOV     PB_8,A
```

To date, most banking terminals and other systems using the DES have needed special boards or peripheral controller chips just for the encryption decryption process, and still more hardware to form a serial bit stream for transmission (Figure 11.a). An 8051 solution could pack most of the entire system onto the one chip (Figure 11.b). The whole DES algorithm would require less than one-

01489A-15

fourth of the on-chip program memory, with the remaining bytes free for operating the banking terminal (or whatever) itself.

Moreover, since transmission and reception of data is performed through the on-board UART, the unencrypted data (plaintext) never even exists outside the microcomputer! Naturally, this would afford a high degree of security from data interception.

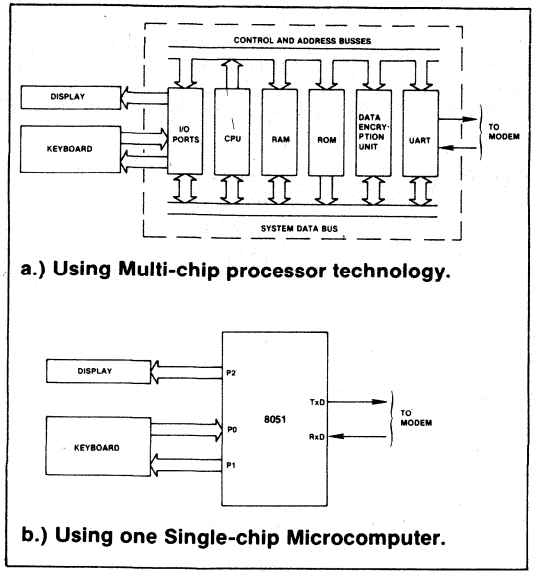


Figure 11. Secure Banking Terminal Block Diagram.

Design Example #2 - Software Serial I/O

An exercise often imposed on beginning microcomputer students is to write a program simulating a UART. (See, for example, Application Notes AP24, AP29, and AP49.) Though doing this with the 8051 Family may appear to be a moot point (given that the hardware for a full UART is on-chip), it is still instructive to see how it would be done, and maintains a product line tradition.

As it turns out, the 8051 microcomputers can receive or transmit serial data via software very efficiently using the Boolean instruction set. Since any I/O pin may be a serial input or output, several serial links could be maintained at once.

Figures 12.a and 12.b show algorithms for receiving or transmitting a byte of data. (Another section of program would invoke this algorithm eight times, synchronizing it with a start bit, clock signal, software delay, or timer interrupt.) Data is received by testing an input pin, setting the carry to the same state, shifting the carry into a data buffer, and saving the partial frame in internal RAM. Data is transmitted by shifting an output buffer through the carry, and generating each bit on an output pin.

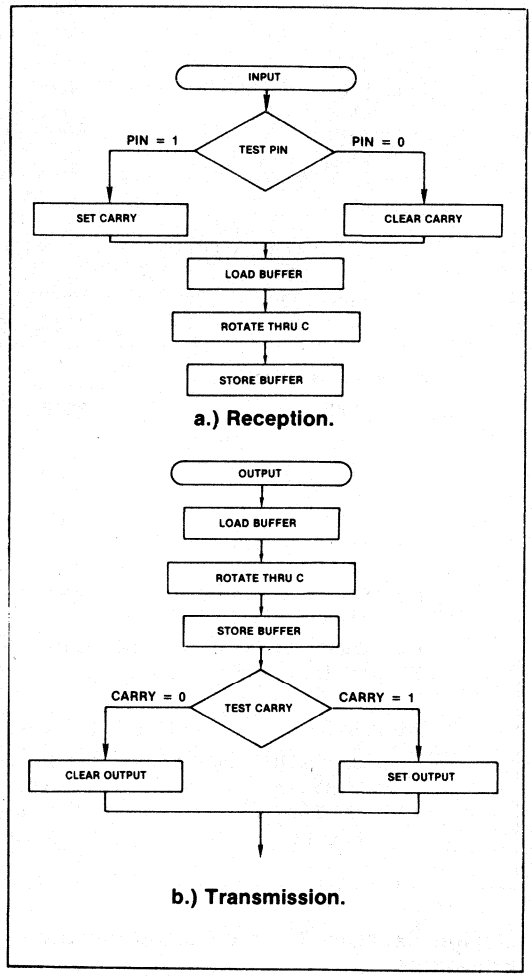


Figure 12. Serial I/O Algorithms.

A side-by-side comparison of the software for this common "bit-banging" application with three different microprocessor architectures is shown in Table 5.a and 5.b. The 8051 solution is more efficient than the others on every count!

01489A-16

Table 5. Serial I/O Programs for Various Microprocessors.

a.) Input Routine.

```
8085                    8048                    8051
IN    SERPORT                                   MOV  C,SERPIN
ANI   MASK              CLR  C
JZ    LO                JNT0 LO
CMC                     CPL  C
LO: LXI   HL,SERBUF     MOV  R0,#SERBUF
MOV   A,M               MOV  A,@R0              MOV  A,SERBUF
RR                      RRC  A                  RRC  A
MOV   M,A               MOV  @R0,A              MOV  SERBUF,A

RESULTS:
 8 INSTRUCTIONS          7 INSTRUCTIONS          4 INSTRUCTIONS
 14 BYTES                9 BYTES                 7 BYTES
 56 STATES               9 CYCLES                4 CYCLES
 19 uSEC.                22.5 uSEC.              4 uSEC.
```

b.) Output Routine.

```
8085                    8048                    8051
LXI   HL,SERBUF         MOV  R0,#SERBUF
MOV   A,M               MOV  A,@R0              MOV  A,SERBUF
RR                      RRC  A                  RRC  A
MOV   M,A               MOV  @R0,A              MOV  SERBUF,A
IN    SERPORT
JC    HI                JC   HI
LO: ANI   NOT MASK      ANL  SERPRT,#NOT MASK   MOV  SERPIN,C
JMP   CNT               JMP  CNT
HI: ORI   MASK      HI: ORL  SERPRT,#MASK
CNT: OUT  SERPORT   CNT:

RESULTS:
 10 INSTRUCTIONS         8 INSTRUCTIONS          4 INSTRUCTIONS
 20 BYTES                13 BYTES                7 BYTES
 72 STATES               11 CYCLES               5 CYCLES
 24 uSEC.                27.5 uSEC.              5 uSEC.
```

Design Example #3 - Combinatorial Logic Equations

Next we'll look at some simple uses for bit-test instructions and logical operations. (This example is also presented in Application Note AP-69.)

Virtually all hardware designers have solved complex functions using combinatorial logic. While the hardware involved may vary from relay logic, vacuum tubes, or TTL or to more esoteric technologies like fluidics, in each case the goal is the same: to solve a problem represented by a logical function of several Boolean variables.

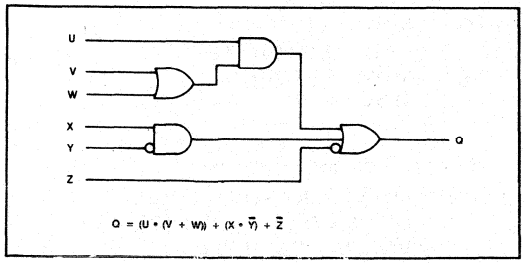
Figure 13 shows TTL and relay logic diagrams for a function of the six variables U through Z. Each is a solution of the equation.

$$Q = (U \cdot (V + W)) + (X \cdot \overline{Y}) + \overline{Z}$$

Equations of this sort might be reduced using Karnaugh Maps or algebraic techniques, but that is not the purpose of this example. As the logic complexity increases, so does the difficulty of the reduction process. Even a minor change to the function equations as the design evolves would require tedious re-reduction from scratch.

01489A-17

Figure 13. Hardware Implementations of Boolean functions.



a.) Using TTL:

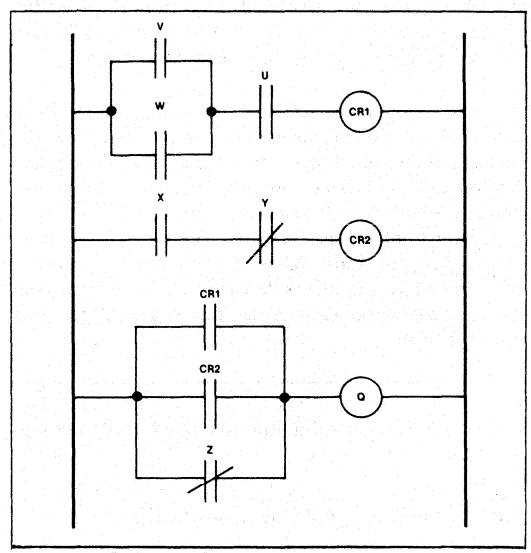
For the sake of comparison we will implement this function three ways, restricting the software to three proper subsets of the MCS-51™ instruction set. We will also assume that U and V are input pins from different input ports, W and X are status bits for two peripheral controllers, and Y and Z are software flags set up earlier in the program. The end result must be written to an output pin on some third port. The first two implementations follow the flow-chart shown in Figure 14. Program flow would embark on a route down a test-and-branch tree and leaves either the "True" or "Not True" exit ASAP — as soon as the proper result has been determined. These exits then rewrite the output port with the result bit respectively one or zero.

Other digital computers must solve equations of this type with standard word-wide logical instructions and conditional jumps. So for the first implementation, we won't use any generalized bit-addressing instructions. As we shall soon see, being constrained to such an instruction subset produces somewhat sloppy software solutions. MCS-51™ mnemonics are used in Example 2.a; other machines might further cloud the situation by requiring operation-specific mnemonics like INPUT, OUTPUT, LOAD, STORE, etc., instead of the MOV mnemonic used for all variable transfers in the 8051 instruction set.

The code which results is cumbersome and error prone. It would be difficult to prove whether the software worked for all input combinations in programs of this sort. Furthermore, execution time will vary widely with input data.

Thanks to the direct bit-test operations, a single instruction can replace each move/mask/conditional jump sequence in Example 2.a, but the algorithm would be equally convoluted (see Example 2.B). To lessen the confusion "a bit" each input variable is assigned a symbolic name.

A more elegant and efficient implementation (Example 2.c) strings together the Boolean ANL and ORL functions to generate the output function with straight-line code.



b.) Using Relay Logic:

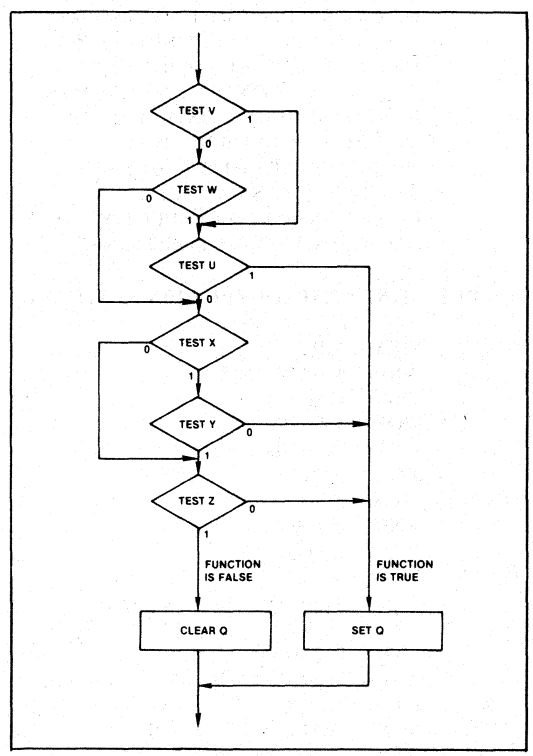


Figure 14. Flow chart for tree-branching algorithm.

01489A-18

When finished, the carry flag contains the result, which is simply copied out to the destination pin. No flow chart is needed—code can be written directly from the logic diagrams in Figure 14. The result is simplicity itself: fast, flexible, reliable, easy to design, and easy to debug.

An 8051 program can simulate an N-input AND or OR gate with at most N+1 lines of source program—one for each input and one line to store the results. To simulate NAND and NOR gates, complement the carry after computing the function. When some inputs to the gate have "inversion bubbles," perform the ANL or ORL operation on inverted operands. When the first input is inverted, either load the operand into the carry and then complement it, or use DeMorgan's Theorem to convert the gate to a different form.

Example 2. Software Solutions to Logic Function of Figure 13.

a.) Using only byte-wide logical instructions.

```
;BFUNC1 SOLVE RANDOM LOGIC FUNCTION
;       OF 6 VARIABLES BY LOADING AND
;       MASKING THE APPROPRIATE BITS
;       IN THE ACCUMULATOR, THEN
;       EXECUTING CONDITIONAL JUMPS
;       BASED ON ZERO CONDITION.
;       (APPROACH USED BY BYTE-
;       ORIENTED ARCHITECTURES.)
;       BYTE AND MASK VALUES
;       CORRESPOND TO RESPECTIVE BYTE
;       ADDRESS AND BIT POSITIONS.
;
OUTBUF  DATA 22H  ;OUTPUT PIN STATE MAP
;
TESTV:  MOV   A,P2
        ANL   A,#00000100B
        JNZ   TESTU
        MOV   A,TCON
        ANL   A,#00100000B
        JZ    TESTX
TESTU:  MOV   A,P1
        ANL   A,#00000010B
        JNZ   SETQ
TESTX:  MOV   A,TCON
        ANL   A,#00001000B
        JZ    TESTZ
        MOV   A,20H
        ANL   A,#00000001B
        JZ    SETQ
TESTZ:  MOV   A,21H
        ANL   A,#00000010B
        JZ    SETQ
CLRQ:   MOV   A,OUTBUF
        ANL   A,#11110111B
        JMP   OUTQ
SETQ:   MOV   A,OUTBUF
        ORL   A,#00001000B
OUTQ:   MOV   OUTBUF,A
        MOV   P3,A
```

b.) Using only bit-test instructions.

```
;BFUNC2 SOLVE A RANDOM LOGIC FUNCTION
;       OF 6 VARIABLES BY DIRECTLY
;       POLLING EACH BIT.
;       (APPROACH USING MCS-51 UNIQUE
;       BIT-TEST INSTRUCTION CAPABILITY.)
;       SYMBOLS USED IN LOGIC DIAGRAM
;       ASSIGNED TO CORRESPONDING 8x51
;       BIT ADDRESSES.
;
U       BIT   P1.1
V       BIT   P2.2
W       BIT   TF0
X       BIT   IE1
Y       BIT   20H.0
Z       BIT   21H.1
Q       BIT   P3.3
;       ...   ....
TEST_V: JB    V,TEST_U
        JNB   W,TEST_X
TEST_U: JB    U,SET_Q
TEST_X: JNB   X,TEST_Z
        JNB   Y,SET_Q
TEST_Z: JNB   Z,SET_Q
CLR_Q:  CLR   Q
        JMP   NXTTST
SET_Q:  SETB  Q
NXTTST:               ;(CONTINUATION OF
                      ;PROGRAM)
```

c.) Using logical operations on Boolean variables.

```
;FUNC3  SOLVE A RANDOM LOGIC FUNCTION
;       OF 6 VARIABLES USING
;       STRAIGHT-LINE LOGICAL
;       INSTRUCTIONS ON MCS-51 BOOLEAN
;       VARIABLES.
;
MOV   C,V
ORL   C,W   ;OUTPUT OF OR GATE
ANL   C,U   ;OUPUT OF TOP AND GATE
MOV   F0,C  ;SAVE INTERMEDIATE STATE
MOV   C,X
ANL   C, Y  ;OUTPUT OF BOTTOM AND GATE
ORL   C,F0  ;INCLUDE VALUE SAVED ABOVE
ORL   C, Z  ;INCLUDE LAST INPUT VARIABLE
MOV   Q,C   ;OUTPUT COMPUTED RESULT
```

01489A-19

An upper-limit can be placed on the complexity of software to simulate a large number of gates by summing the total number of inputs and outputs. The *actual* total should be somewhat shorter, since calculations can be "chained," as shown above. The output of one gate is often the first input to another, bypassing the intermediate variable to eliminate two lines of source.

Design Example #4 - Automotive Dashboard Functions

Now let's apply these techniques to designing the software for a complete controller system. This application is patterned after a familiar real-world application which isn't nearly as trivial as it might first appear: automobile turn signals.

Imagine the three position turn lever on the steering column as a single-pole, triple-throw toggle switch. In its central position all contacts are open. In the up or down positions contacts close causing corresponding lights in the rear of the car to blink. So far very simple.

Two more turn signals blink in the front of the car, and two others in the dashboard. All six bulbs flash when an emergency switch is closed. A thermo-mechanical relay (accessible under the dashboard in case it wears out) causes the blinking.

Applying the brake pedal turns the tail light filaments on constantly . . . unless a turn is in progress, in which case the blinking tail light is not affected. (Of course, the front turn signals and dashboard indicators are not affected by the brake pedal.) Table 6 summarizes these operating modes.

But we're not done yet. Each of the exterior turn signal (but not the dashboard) bulbs has a second, somewhat dimmer filament for the parking lights. Figure 15 shows TTL circuitry which could control all six bulbs. The signals labeled "High Freq." and "Low Freq." represent two square-wave inputs. Basically, when one of the turn switches is closed or the emergency switch is activated the low frequency signal (about 1 Hz) is gated through to the appropriate dashboard indicator(s) and turn signal(s). The rear signals are also activated when the brake pedal is depressed provided a turn is not being made in the same direction. When the parking light switch is closed the higher frequency oscillator is gated to each front and rear turn signal, sustaining a low-intensity background level. (This is to eliminate the need for additional parking light filaments.)

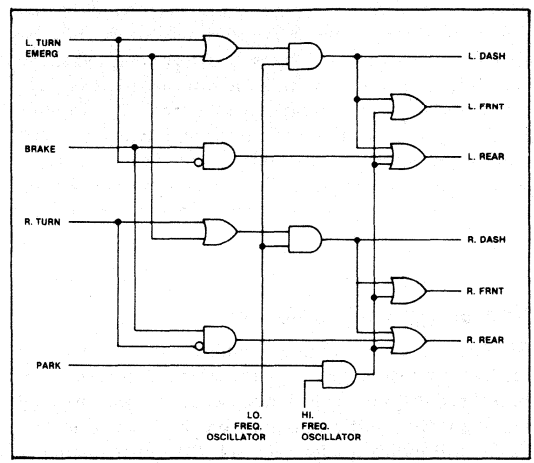


Figure 15. TTL logic implementation of automotive turn signals.

Table 6. Truth table for turn-signal operation.

INPUT SIGNALS				OUTPUT SIGNALS			
BRAKE SWITCH	EMERG. SWITCH	LEFT TURN SWITCH	RIGHT TURN SWITCH	LEFT FRONT & DASH	RIGHT FRONT & DASH	LEFT REAR	RIGHT REAR
0	0	0	0	OFF	OFF	OFF	OFF
0	0	0	1	OFF	BLINK	OFF	BLINK
0	0	1	0	BLINK	OFF	BLINK	OFF
0	1	0	0	BLINK	BLINK	BLINK	BLINK
0	1	0	1	BLINK	BLINK	BLINK	BLINK
0	1	1	0	BLINK	BLINK	BLINK	BLINK
1	0	0	0	OFF	OFF	ON	ON
1	0	0	1	OFF	BLINK	ON	BLINK
1	0	1	0	BLINK	OFF	BLINK	ON
1	1	0	0	BLINK	BLINK	ON	ON
1	1	0	1	BLINK	BLINK	ON	BLINK
1	1	1	0	BLINK	BLINK	BLINK	ON

01489A-20

In most cars, the switching logic to generate these functions requires a number of multiple-throw contacts. As many as 18 conductors thread the steering column of some automobiles solely for turn-signal and emergency blinker functions. (The author discovered this recently to his astonishment and dismay when replacing the whole assembly because of one burned contact.)

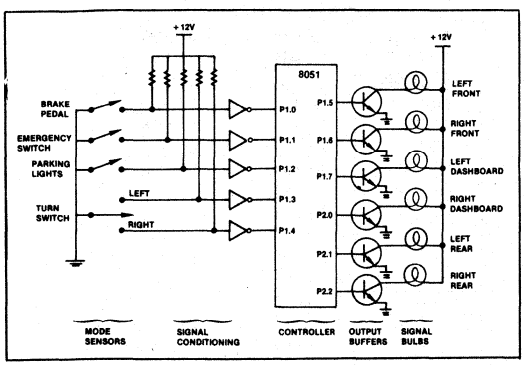
A multiple-conductor wiring harness runs to each corner of the car, behind the dash, up the steering column, and down to the blinker relay below. Connectors at each termination for each filament lead to extra cost and labor during construction, lower reliability and safety, and more costly repairs. And considering the system's present complexity, increasing its reliability or detecting failures would be quite difficult.

There are two reasons for going into such painful detail describing this example. First, to show that the messiest part of many system designs is determining what the controller should do. Writing the software to solve these functions will be comparatively easy. Secondly, to show the many potential failure points in the system. Later we'll see how the peripheral functions and intelligence built into a microcomputer (with a little creativity) can greatly reduce external interconnections and mechanical part count.

The Single-chip Solution

The circuit shown in Figure 16 indicates five input pins to the five input variables—left-turn select, right-turn select, brake pedal down, emergency switch on, and parking lights on. Six output pins turn on the front, rear, and dashboard indicators for each side. The microcomputer implements all logical functions through software, which periodically updates the output signals as time elapses and input conditions change.

Figure 16. Microcomputer Turn-signal Connections.



Design Example #3 demonstrated that symbolic addressing with user-defined bit names makes code and documentation easier to write and maintain. Accordingly, we'll assign these I/O pins names for use throughout the program. (The format of this example will differ somewhat from the others. Segments of the overall program will be presented in sequence as each is described.)

```
;
;          INPUT PIN DECLARATIONS:
;     (ALL INPUTS ARE POSITIVE-TRUE LOGIC)
;
BRAKE    BIT P1.0   ; BRAKE PEDAL DEPRESSED
EMERG    BIT P1.1   ; EMERGENCY BLINKER
                      ACTIVATED
PARK     BIT P1.2   ; PARKING LIGHTS ON
L_TURN   BIT P1.3   ; TURN LEVER DOWN
R_TURN   BIT P1.4   ; TURN LEVER UP
;
;          OUTPUT PIN DECLARATIONS:
;
L_FRNT   BIT P1.5   ; FRONT LEFT-TURN
                      INDICATOR
R_FRNT   BIT P1.6   ; FRONT RIGHT-TURN
                      INDICATOR
L_DASH   BIT P1.7   ; DASHBOARD LEFT-TURN
                      INDICATOR
R_DASH   BIT P2.0   ; DASHBOARD RIGHT-TURN
                      INDICATOR
L_REAR   BIT P2.1   ; REAR LEFT-TURN
                      INDICATOR
R_REAR   BIT P2.2   ; REAR RIGHT-TURN
                      INDICATOR
;
```

Another key advantage of symbolic addressing will appear further on in the design cycle. The locations of cable connectors, signal conditioning circuitry, voltage regulators, heat sinks, and the like all affect P.C. board layout. It's quite likely that the somewhat arbitrary pin assignment defined early in the software design cycle will prove to be less than optimum; rearranging the I/O pin assignment could well allow a more compact module, or eliminate costly jumpers on a single-sided board. (These considerations apply especially to automotive and other cost-sensitive applications needing single-chip controllers.) Since other architectures mask bytes or use "clever" algorithms to isolate bits by rotating them into the carry, re-routing an input signal (from bit 1 of port 1, for example, to bit 4 of port 3) could require extensive modifications throughout the software.

The Boolean Processor's direct bit addressing makes such changes absolutely trivial. The number of the port containing the pin is irrelevent, and masks and complex program structures are not needed. Only the initial Boolean varia-

01489A-21

```
;                   ...         .....
; INTERRUPT RATE SUBDIVIDER
SUB_DIV         DATA    20H
; HIGH-FREQUENCY OSCILLATOR BIT
HI_FREQ         BIT     SUB_DIV.0
; LOW-FREQUENCY OSCILLATOR BIT
LO_FREQ         BIT     SUB_DIV.7
;
                ...
                ORG     0000H
JMP             INIT
;
                ...     .....
                ORG     100H
; PUT TIMER 0 IN MODE 1
INIT:           MOV     TMOD,#00000001B
; INITIALIZE TIMER REGISTERS
                MOV     TL0,#0
                MOV     TH0,#-16
; SUBDIVIDE INTERRUPT RATE BY 244
                MOV     SUB_DIV,#244
; ENABLE TIMER INTERRUPTS
                SETB    ET0
; GLOBALLY ENABLE ALL INTERRUPTS
                SETB    EA
; START TIMER
                SETB    TR0
;
; (CONTINUE WITH BACKGROUND PROGRAM)
;

; PUT TIMER 0 IN MODE 1
; INITIALIZE TIMER REGISTERS

; SUBDIVIDE INTERRUPT RATE BY 244
; ENABLE TIMER INTERRUPTS
; GLOBALLY ENABLE ALL INTERRUPTS
; START TIMER
```

ble declarations need to be changed; ASM51 automatically adjusts all addresses and symbolic references to the reassigned variables. The user is assured that no additional debugging or software verification will be required.

Timer 0 (one of the two on-chip timer/counters) replaces the thermo-mechanical blinker relay in the dashboard controller. During system initialization it is configured as a timer in mode 1 by setting the least significant bit of the timer mode register (TMOD). In this configuration the low-order byte (TL0) is incremented every machine cycle, overflowing and incrementing the high-order byte (TH0) every 256 μSec. Timer interrupt 0 is enabled so that a hardware interrupt will occur each time TH0 overflows. (For details of the numerous timer operating modes see the MCS-51™ User's Manual.)

An eight-bit variable in the bit-addressable RAM array will be needed to further subdivide the interrupts via software. The lowest-order bit of this counter toggles very fast to modulate the parking lights; bit 7 will be "tuned" to approximately 1 Hz for the turn- and emergency-indicator blinking rate.

Loading TH0 with -16 will cause an interrupt after 4.096 msec. The interrupt service routine reloads the high-order byte of timer 0 for the next interval, saves the CPU registers likely to be affected on the stack, and then decrements SUB_DIV. Loading SUB_DIV. with 244 initially and each time it decrements to zero will produce a 0.999 second period for the highest-order bit.

```
ORG     000BH       ; TIMER 0 SERVICE VECTOR
MOV     TH0,#-16
PUSH    PSW
PUSH    ACC
PUSH    B
DJNZ    SUB_DIV,T0SERV
MOV     SUB_DIV,#244
```

The code to sample inputs, perform calculations, and update outputs—the real "meat" of the signal controller algorithm—may be performed either as part of the interrupt service routine or as part of a background program loop. The only concern is that it must be executed at least several dozen times per second to prevent parking light flickering. We will assume the former case, and insert the code into the timer 0 service routine.

First, notice from the logic diagram (Figure 15) that the subterm (PARK · H_FREQ), asserted when the parking lights are to be on dimly, figures into four of the six output functions. Accordingly, we will first compute that term and save it in a temporary location named "DIM". The PSW contains two general purpose flags: F0, which corresponds to the 8048 flag of the same name, and PSW.1. Since The PSW has been saved and will be restored to its previous state after servicing the interrupt, we can use either bit for temporary storage.

```
DIM     BIT         PSW.1   ; DECLARE TEMP.
                            STORAGE FLAG
;   ...     .....
    MOV     C,PARK          ; GATE PARKING
                            LIGHT SWITCH
    ANL     HI_FREQ         ; WITH HIGH
                            FREQUENCY
                            SIGNAL
    MOV     DIM,C           ; AND SAVE IN
                            TEMP. VARIABLE.
```

This simple three-line section of code illustrates a remarkable point. The software indicates in very abstract terms exactly what function is being performed, independent of

01489A-22

the hardware configuration. The fact that these three bits include an input pin, a bit within a program variable, and a software flag in the PSW is totally invisible to the programmer.

Now generate and output the dashboard left turn signal.

```
:
   MOV  C,L_TURN        ; SET CARRY IF
                          TURN
   ORL  C,EMERG         ; OR EMERGENCY
                          SELECTED.
   ANL  C,LO_FREQ       ; GATE IN 1 HZ
                          SIGNAL
   MOV  L_DASH,C        ; AND OUTPUT TO
                          DASHBOARD.
```

To generate the left front turn signal we only need to add the parking light function in F0. But notice that the function in the carry will also be needed for the rear signal. We can save effort later by saving its current state in F0.

```
:
   MOV  F0,C            ; SAVE FUNCTION
                          SO FAR.
   ORL  C,DIM           ; ADD IN PARKING
                          LIGHT FUNCTION
   MOV  L_FRNT,C        ; AND OUTPUT TO
                          TURN SIGNAL.
```

Finally, the rear left turn signal should also be on when the brake pedal is depressed, provided a left turn is not in progress.

```
   MOV  C,BRAKE         ; GATE BRAKE
                          PEDAL SWITCH
   ANL  C,/L_TURN       ; WITH TURN
                          LEVER.
   ORL  C,F0            ; INCLUDE TEMP.
                          VARIABLE FROM
                          DASH
   ORL  C,DIM           ; AND PARKING
                          LIGHT FUNCTION
   MOV  L_REAR,C        ; AND OUTPUT TO
                          TURN SIGNAL.
```

Now we have to go through a similar sequence for the right-hand equivalents to all the left-turn lights. This also gives us a chance to see how the code segments above look when combined.

```
   MOV  C,R_TURN        ; SET CARRY IF
                          TURN
   ORL  C,EMERG         ; OR EMERGENCY
                          SELECTED.
   ANL  C,LO_FREQ       ; IF SO, GATE IN 1
                          HZ SIGNAL
   MOV  R_DASH,C        ; AND OUTPUT TO
                          DASHBOARD.
   MOV  F0,C            ; SAVE FUNCTION
                          SO FAR.
   ORL  C,DIM           ; ADD IN PARKING
                          LIGHT FUNCTION
   MOV  R_FRNT,C        ; AND OUTPUT TO
                          TURN SIGNAL.
   MOV  C,BRAKE         ; GATE BRAKE
                          PEDAL SWITCH
   ANL  C,/R_TURN       ; WITH TURN
                          LEVER.
   ORL  C,F0            ; INCLUDE TEMP.
                          VARIABLE FROM
                          DASH
   ORL  C,DIM           ; AND PARKING
                          LIGHT FUNCTION
   MOV  R_REAR,C        ; AND OUTPUT TO
                          TURN SIGNAL.
```

(The perceptive reader may notice that simply rearranging the steps could eliminate one instruction from each sequence.)

Now that all six bulbs are in the proper states, we can return from the interrupt routine, and the program is finished. This code essentially needs to reverse the status saving steps at the beginning of the interrupt.

```
   POP  B               ; RESTORE CPU
                          REGISTERS.
   POP  ACC
   POP  PSW
   RETI
```

Program Refinements. The luminescence of an incandescent light bulb filament is generally non-linear; the 50% duty cycle of HI_FREQ may not produce the desired intensity. If the application requires, duty cycles of 25%, 75%, etc. are easily achieved by ANDing and ORing in additional low-order bits of SUB_DIV. For example, 30 Hz signals of seven different duty cycles could be produced by considering bits 2—0 as shown in Table 7. The only software change required would be to the code which sets-up variable DIM:

```
   MOV  C,SUB_DIV.1     ; START WITH 50
                          PERCENT
   ANL  C,SUB_DIV.0     ; MASK DOWN TO 25
                          PERCENT
   ORL  C,SUB_DIV.2     ; AND BUILD BACK TO
                          62 PERCENT
   MOV  DIM,C           ; DUTY CYCLE FOR
                          PARKING LIGHTS.
```

01489A-23

Table 7. Non-trivial Duty Cycles.

SUB_DIV BITS								DUTY CYCLES						
7	6	5	4	3	2	1	0	12.5%	25.0%	37.5%	50.0%	62.5%	75.0%	87.5%
X	X	X	X	X	0	0	0	OFF	OFF	OFF	OFF	OFF	OFF	OFF
X	X	X	X	X	0	0	1	OFF	OFF	OFF	OFF	OFF	OFF	ON
X	X	X	X	X	0	1	0	OFF	OFF	OFF	OFF	OFF	ON	ON
X	X	X	X	X	0	1	1	OFF	OFF	OFF	OFF	ON	ON	ON
X	X	X	X	X	1	0	0	OFF	OFF	OFF	ON	ON	ON	ON
X	X	X	X	X	1	0	1	OFF	OFF	ON	ON	ON	ON	ON
X	X	X	X	X	1	1	0	OFF	ON	ON	ON	ON	ON	ON
X	X	X	X	X	1	1	1	ON	ON	ON	ON	ON	ON	ON

Interconnections increase cost and decrease reliability. The simple buffered pin-per-function circuit in Figure 16 is insufficient when many outputs require higher-than-TTL drive levels. A lower-cost solution uses the 8051 serial port in the shift-register mode to augment I/O. In mode 0, writing a byte to the serial port data buffer (SBUF) causes the data to be output sequentially through the "RXD" pin while a burst of eight clock pulses is generated on the "TXD" pin. A shift register connected to these pins (Figure 17) will load the data byte as it is shifted out. A number of special peripheral driver circuits combining shift-register inputs with high drive level outputs have been introduced recently.

Cascading multiple shift registers end-to-end will expand the number of outputs even further. The data rate in the I/O expansion mode is one megabaud, or 8 usec. per byte. This is the mode which the serial port defaults to following a reset, so no initialization is required.

The software for this technique uses the B register as a "map" corresponding to the different output functions. The program manipulates these bits instead of the output pins. After all functions have been calculated the B register is shifted by the serial port to the shift-register driver. (While some outputs may glitch as data is shifted through them, at 1 Megabaud most people wouldn't notice. Some shift registers provide an "enable" bit to hold the output states while new data is being shifted in.)

This is where the earlier decision to address bits symbolically throughout the program is going to pay off. This major I/O restructuring is nearly as simple to implement as rearranging the input pins. Again, only the bit declarations need to be changed.

```
L_FRNT  BIT  B.0  ; FRONT LEFT-TURN
                  ; INDICATOR
R_FRNT  BIT  B.1  ; FRONT RIGHT-TURN
                  ; INDICATOR
L_DASH  BIT  B.2  ; DASHBOARD LEFT-TURN
                  ; INDICATOR
R_DASH  BIT  B.3  ; DASHBOARD RIGHT-TURN
                  ; INDICATOR
```

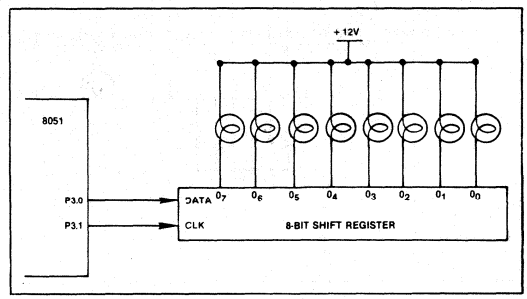


Figure 17. Output expansion using serial port.

```
L_REAR  BIT  B.4  ; REAR LEFT-TURN
                  ; INDICATOR
R_REAR  BIT  B.5  ; REAR RIGHT-TURN
                  ; INDICATOR
```

The original program to compute the functions need not change. After computing the output variables, the control map is transmitted to the buffered shift register through the serial port:

```
MOV  SBUF,B  ;LOAD BUFFER AND TRANSMIT
```

The Boolean Processor solution holds a number of advantages over older methods. Fewer switches are required. Each is simpler, requiring fewer poles and lower current contacts. The flasher relay is eliminated entirely. Only six filaments are driven, rather than 10. The wiring harness is therefore simpler and less expensive—one conductor for each of the six lamps and each of the five sensor switches. The fewer conductors use far fewer connectors. The whole system is more reliable.

And since the system is much simpler it would be feasible to implement redundancy and or fault detection on the four main turn indicators. Each could **still** be a standard double filament bulb, but with the filaments driven in parallel to tolerate single-element failures.

Even with redundancy, the lights will eventually fail. To handle this inescapable fact current or voltage sensing

01489A-24

circuits on each main drive wire can verify that each bulb and its high-current driver is functioning properly. Figure 18 shows one such circuit.

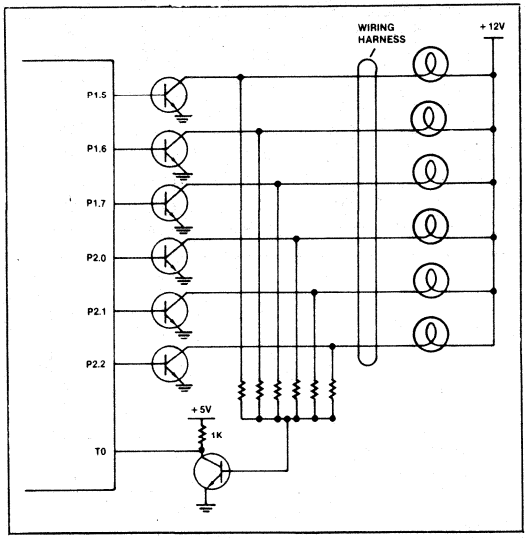


Figure 18.

Assume all of the lights are turned on except one; i.e., all but one of the collectors are grounded. For the bulb which is turned off, if there is continuity from +12 V through the bulb base and filament, the control wire, all connectors, and the P.C. board traces, and if the transistor is indeed not shorted to ground, then the collector will be pulled to +12 V. This turns on the base of Q8 through the corresponding resistor, and grounds the input pin, verifying that the bulb circuit is operational. The continuity of each circuit can be checked by software in this way.

Now turn *all* the bulbs on, grounding all the collectors. Q7 should be turned off, and the Test pin should be high. However, a control wire shorted to +12 V or an open-circuited drive transistor would leave one of the collectors at the higher voltage even now. This too would turn on Q7, indicating a different type of failure. Software could perform these checks once per second by executing the routine every time the software counter SUB_DIV is reloaded by the interrupt routine.

```
DJNZ SUB_DIV,T0SERV
MOV  SUB_DIV,#244        ; RELOAD COUNTER
ORL  P1,#11100000B       ; SET CONTROL
                           OUTPUTS HIGH
ORL  P2,#00000111B
CLR  L_FRNT              ; FLOAT DRIVE
                           COLLECTOR
JB   T0,FAULT            ; T0 SHOULD BE
                           PULLED LOW
SETB L_FRNT              ; PULL COLLECTOR
                           BACK DOWN
CLR  L_DASH
JB   T0,FAULT
SETB L_DASH
CLR  L_REAR
JB   T0,FAULT
SETB L_REAR
CLR  R_FRNT
JB   T0,FAULT
SETB R_FRNT
CLR  R_DASH
JB   T0,FAULT
SETB R_DASH
CLR  R_REAR
JB   T0,FAULT
SETB R_REAR
;
; WITH ALL COLLECTORS GROUNDED, T0
  SHOULD BE HIGH
; IF SO, CONTINUE WITH INTERRUPT ROUTINE.
  JB   T0,T0SERV
FAULT:                   ; ELECTRICAL FAILURE
                         ;PROCESSING ROUTINE
                         ; (LEFT TO READER'S
                         ; IMAGINATION)
T0SERV:                  ; CONTINUE WITH
                         ;INTERRUPT PROCESSING
;
;
```

The complete assembled program listing is printed in Appendix A. The resulting code consists of 67 program statements, not counting declarations and comments, which assemble into 150 bytes of object code. Each pass through the service routine requires (coincidently) 67 usec, plus 32 usec once per second for the electrical test. If executed every 4 msec as suggested this software would typically reduce the throughput of the background program by less than 2%.

Once a microcomputer has been designed into a system, new features suddenly become virtually free. Software could make the emergency blinkers flash alternately or at a rate faster than the turn signals. Turn signals could override the emergency blinkers. Adding more bulbs would allow multiple tail light sequencing and syncopation — true flash factor, so to speak.

Design Example #5 - Complex Control Functions

Finally, we'll mix byte and bit operations to extend the use of 8051 into extremely complex applications.

Programmers can arbitrarily assign I/O pins to input and output functions only if the total does not exceed 32, which is insufficient for applications with a very large number of input variables. One way to expand the number of inputs is with a technique similar to multiplexed-keyboard scanning.

Figure 19 shows a block diagram for a moderately complex programmable industrial controller with the following characteristics:

- 64 input variable sensors;
- 12 output signals;
- Combinational and sequential logic computations;
- Remote operation with communications to a host processor via a high-speed full-duplex serial link;
- Two prioritized external interrupts;
- Internal real-time and time-of-day clocks.

While many microprocessors could be programmed to provide these capabilities with assorted peripheral support chips, an 8051 microcomputer needs **no** other integrated circuits!

The 64 input sensors are logically arranged as an 8x8 matrix. The pins of Port 1 sequentially enable each column of the sensor matrix; as each is enabled Port 0 reads in the state of each sensor in that column. An eight-byte block in bit-addressable RAM remembers the data as it is read in so that after each complete scan cycle there is an internal map of the current state of all sensors. Logic functions can then directly address the elements of the bit map.

The computer's serial port is configured as a nine-bit UART, transferring data at 17,000 bytes-per-second. The ninth bit may distinguish between address and data bytes.

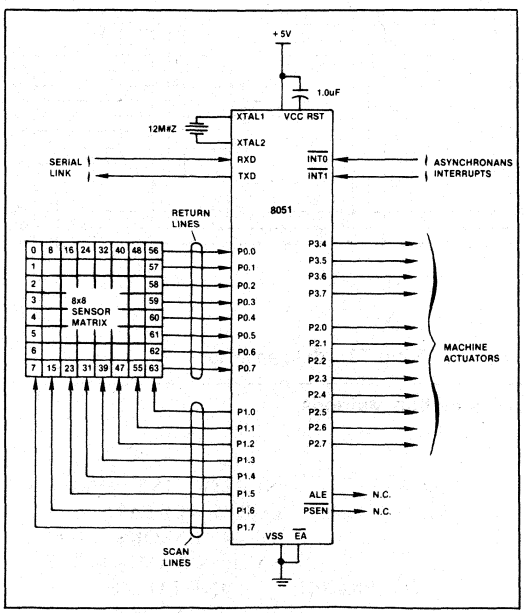


Figure 19. Block diagram of 64-input machine controller.

The 8051 serial port can be configured to detect bytes with the address bit set, automatically ignoring all others. Pins INT0 and INT1 are interrupts configured respectively as high-priority, falling-edge triggered and low-priority, low-level triggered. The remaining 12 I/O pins output TTL-level control signals to 12 actuators.

There are several ways to implement the sensor matrix circuitry, all logically similar. Figure 20.a shows one possibility. Each of the 64 sensors consists of a pair of simple switch contacts in series with a diode to permit multiple contact closures throughout the matrix.

The scan lines from Port 1 provide eight un-encoded active-high scan signals for enabling columns of the matrix. The return lines on rows where a contact is closed are pulled high and read as logic ones. Open return lines are pulled to ground by one of the 40 kohm resistors and are read as zeroes. (The resistor values must be chosen to ensure all return lines are pulled above the 2.0 V logic threshold, even in the worst-case, where all contacts in an enabled column are closed.) Since P0 is provided open-collector outputs and high-impedance MOS inputs its input loading may be considered negligible.

The circuits in Figures 20.b—20.d are variations on this theme. When input signals must be electrically isolated from the computer circuitry as in noisy industrial environments, phototransistors can replace the switch diode pairs **and** provide optical isolation as in Figure 20.b. Additional opto-isolators could also be used on the control output and special signal lines.

The other circuits assume that input signals are already at TTL levels. Figure 20.c uses octal three-state buffers enabled by active-low scan signals to gate eight signals onto Port 0. Port 0 is available for memory expansion or peripheral chip interfacing between sensor matrix scans. Eight-to-one multiplexers in Figure 20.d select one of eight inputs for each return line as determined by encoded address bits output on three pins of Port 1. (Five more output pins are thus freed for more control functions.) Each output can drive at least one standard TTL or up to 10 low-power TTL loads without additional buffering.

Going back to the original matrix circuit, Figure 21 shows the method used to scan the sensor matrix. Two complete bit maps are maintained in the bit-addressable region of the RAM: one for the current state and one for the previous state read for each sensor. If the need arises, the program could then sense input transitions and/or debounce contact closures by comparing each bit with its earlier value.

01489A-26

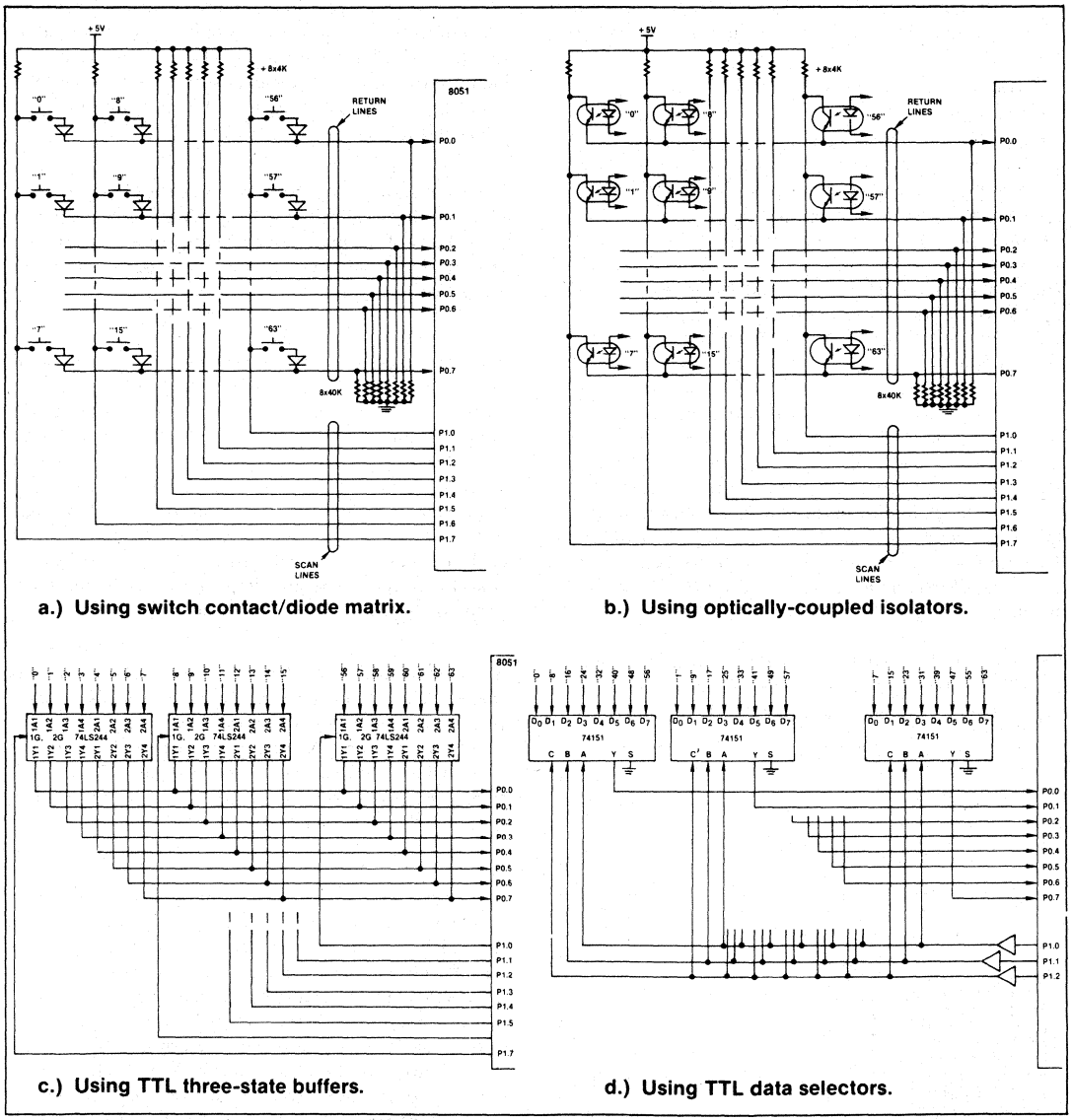


Figure 20. Sensor Matrix Implementation Methods.

The code in Example 3 implements the scanning algorithm for the circuits in Figure 20.a. Each column is enabled by setting a single bit in a field of zeroes. The bit maps are positive logic; ones represent contacts that are closed or isolators turned on.

Example 3.

```
INPUT_SCAN:          ; SUBROUTINE TO READ
                     ; CURRENT STATE
                     ; OF 64 SENSORS AND
                     ; SAVE IN RAM 20H-27H.
        MOV  R0,#20H     ; INITIALIZE
                         ; POINTERS
        MOV  R1,#28H     ; FOR BIT MAP
                         ; BASES.
```

01489A-27

```
        MOV  A,#80H        ; SET FIRST BIT IN
                             ACC.
SCAN:   MOV  P1,A          ; OUTPUT TO SCAN
                             LINES.
        RR   A             ; SHIFT TO ENABLE
                             NEXT COLUMN
                             NEXT.
        MOV  R2,A          ; REMEMBER CUR-
                             RENT SCAN
                             POSITION.
        MOV  A,P0          ; READ RETURN
                             LINES.
        XCH  A,@R0         ; SWITCH WITH
                             PREVIOUS MAP
                             BITS.
        MOV  @R1,A         ; SAVE PREVIOUS
                             STATE AS WELL.
        INC  R0            ; BUMP POINTERS.
        INC  R1
        MOV  A,R2          ; RELOAD SCAN LINE
                             MASK
        JNB  ACC.7,SCAN    ; LOOP UNTIL ALL
                             EIGHT COLUMNS
                             READ.
        RET
```

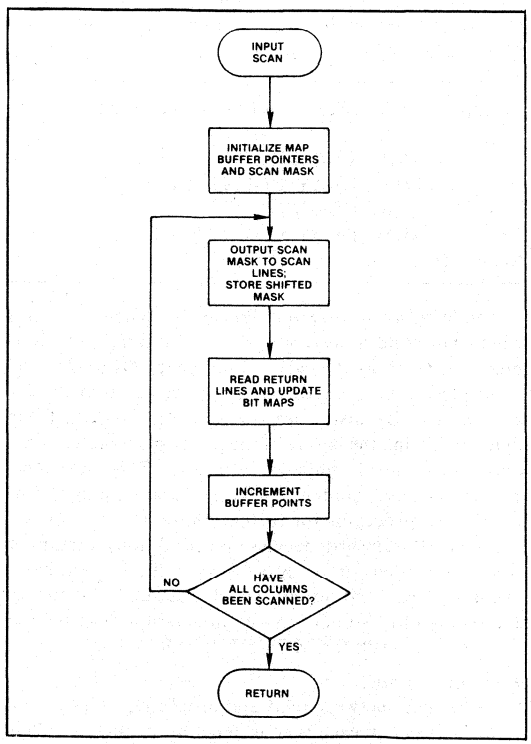


Figure 21. Flowchart for reading in sensor matrix.

What happens after the sensors have been scanned depends on the individual application. Rather than inventing some artificial design problem, software corresponding to commonplace logic elements will be discussed.

Combinatorial Output Variables. An output variable which is a simple (or not so simple) combinational function of several input variables is computed in the spirit of Design Example 3. All 64 inputs are represented in the bit maps; in fact, the sensor numbers in Figure 20 correspond to the absolute bit addresses in RAM! The code in Example 4 activates an actuator connected to P2.2 when sensors 12, 23, and 34 are closed and sensors 45 and 56 are open.

Example 4.

Simple Combinatorial Output Variables.

```
; SET P2.2 = (12) (23) (34) ( 45) ( 56)
        MOV  C,12
        ANL  C,23
        ANL  C,34
        ANL  C, 45
        ANL  C, 56
        MOV  P2.2,C
```

Intermediate Variables. The examination of a typical relay-logic ladder diagram will show that many of the rungs control *not* outputs but rather relays whose contacts figure into the computation of other functions. In effect, these relays indicate the state of intermediate variables of a computation.

The MCS-51™ solution can use any directly addressable bit for the storage of such intermediate variables. Even when all 128 bits of the RAM array are dedicated (to input bit maps in this example), the accumulator, PSW, and B register provide 18 additional flags for intermediate variables.

For example, suppose switches 0 through 3 control a safety interlock system. Closing any of them should deactivate certain outputs. Figure 22 is a ladder diagram for this situation. The interlock function could be recomputed for every output affected, or it may be computed once and saved (as implied by the diagram). As the program proceeds this bit can qualify each output.

Example 5. Incorporating Override signal into actuator outputs.

```
        CALL INPUT_SCAN
        MOV  C,0
        ORL  C,1
        ORL  C,2
        ORL  C,3
        MOV  F0,C
;       ....    .....
```

01489A-28

```
;       COMPUTE FUNCTION 0
;
        ANL   C, F0
        MOV   P1.0,C
;       ....    .....
;       COMPUTE FUNCTION 1
;
        ANL   C, F0
        MOV   P1.1,C
;       ....    .....
;       COMPUTE FUNCTION 2
;
        ANL   C, F0
        MOV   P1.2,C
;       ....    .....
```

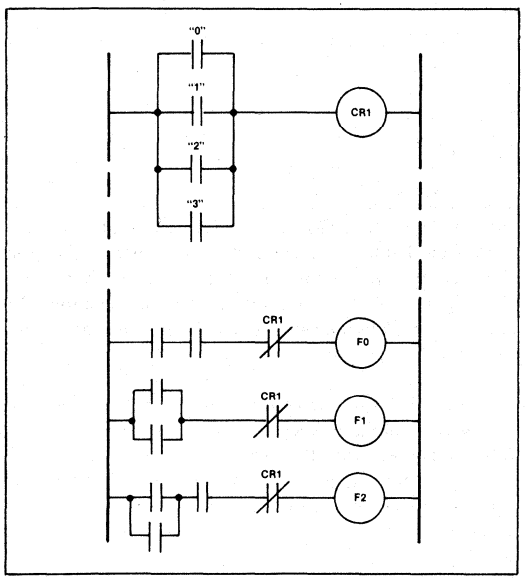


Figure 22. Ladder diagram for output override circuitry.

Latching Relays. A latching relay can be forced into either the ON or OFF state by two corresponding input signals, where it will remain until forced onto the opposite state—analogous to a TTL Set/Reset flip-flop. The relay is used as an intermediate variable for other calculations. In the previous example, the emergency condition could be remembered and remain active until an "emergency cleared" button is pressed.

Any flag or addressable bit may represent a latching relay with a few lines of code (see Example 6).

Example 6. Simulating a latching relay.

```
:I_SET   SET FLAG 0 IF C=1
I_SET:    ORL   C,F0
          MOV   F0,C
;

:I_RSET RESET FLAG 0 IF C=1
I_RSET:  CPS   C
          ANL   C,F0
          MOV   F0,C
;
```

Time Delay Relays. A time delay relay does not respond to an input signal until it has been present (or absent) for some predefined time. For example, a ballast or load resistor may be switched in series with a D.C. motor when it is first turned on, and shunted from the circuit after one second. This sort of time delay may be simulated by an interrupt routine driven by one of the two 8051 timer/counters. The procedure followed by the routine depends heavily on the details of the exact function needed; time-outs or time delays with resettable or non-resettable inputs are possible. If the interrupt routine is executed every 10 milliseconds the code in Example 7 will clear an intermediate variable set by the background program after it has been active for two seconds.

Example 7. Code to clear USRFLG after a fixed time delay.

```
          JNB   USR_FLG,NXTTST
          DJNZ DLAY_COUNT,NXTTST
          CLR   USR_FLG
          MOV   DLAY_COUNT,#200
NXTTST:  ;..     .....
```

Serial Interface to Remote Processor. When it detects emergency conditions represented by certain input combinations (such as the earlier Emergency Override), the controller could shut down the machine immediately and/or alert the host processor via the serial port. Code bytes indicating the nature of the problem could be transmitted to a central computer. In fact, at 17,000 bytes-per-second, the entire contents of both bit maps could be sent to the host processor for further analysis in less than a millisecond! If the host decides that conditions warrant, it could alert other remote processors in the system that a problem exists and specify which shut-down sequence each should initiate. For more information on using the serial port, consult the MCS-51™ User's Manual.

Response Timing.
One difference between relay and programmed industrial controllers (when each is considered as a "black box") is their respective reaction times to input changes. As reflected by a ladder diagram, relay systems contain a

large number of "rungs" operating in parallel. A change in input conditions will begin propagating through the system immediately, possibly affecting the output state within milliseconds.

Software, on the other hand, operates sequentially. A change in input states will not be detected until the next time an input scan is performed, and will not affect the outputs until that section of the program is reached. For that reason the raw speed of computing the logical functions is of extreme importance.

Here the Boolean processor pays off. *Every instruction mentioned in this Note* completes in one or two microseconds—the *minimum* instruction execution time for many other microcontrollers! A ladder diagram containing a hundred rungs, with an average of four contacts per rung can be replaced by approximately five hundred lines of software. A complete pass through the entire matrix scanning routine and all computations would require about a millisecond; less than the time it takes for most relays to change state.

A programmed controller which simulates each Boolean function with a subroutine would be less efficient by at least an order of magnitude. Extra software is needed for the simulation routines, and each step takes longer to execute for three reasons: several byte-wide logical instructions are executed per user program step (rather than one Boolean operation); most of those instructions take longer to execute with microprocessors performing multiple off-chip accesses; and calling and returning from the various subroutines requires overhead for stack operations.

In fact, the speed of the Boolean Processor solution is likely to be much faster than the system requires. The CPU might use the time left over to compute feedback parameters, collect and analyze execution statistics, perform system diagnostics, and so forth.

Additional functions and uses.

With the building-block basics mentioned above many more operations may be synthesized by short instruction sequences.

Exclusive-OR. There are no common mechanical devices or relays analogous to the Exclusive-OR operation, so this instruction was omitted from the Boolean Processor. However, the Exclusive-OR or Exclusive-NOR operation may be performed in two instructions by conditionally complementing the carry or a Boolean variable based on the state of any other testable bit.

```
; EXCLUSIVE-OR FUNCTION IMPOSED ON CARRY
; USING F0 IS INPUT VARIABLE.
XOR_F0:  JNB   F0,XORCNT  ; ("JB" FOR X-NOR)
         CPL   C
XORCNT:...     .....
```

XCH. The contents of the carry and some other bit may be exchanged (switched) by using the accumulator as temporary storage. Bits can be moved into and out of the accumulator simultaneously using the Rotate-through-carry instructions, though this would alter the accumulator data.

```
; EXCHANGE CARRY WITH USRFLG
XCHBIT:  RLC   A
         MOV   C,USR_FLG
         RRC   A
         MOV   USR_FLG,C
         RLC   A
```

Extended Bit Addressing. The 8051 can directly address 144 general-purpose bits for all instructions in Figure 3.b. Similar operations may be extended to any bit anywhere on the chip with some loss of efficiency.

The logical operations AND, OR, and Exclusive-OR are performed on byte variables using six different addressing modes, one of which lets the source be an immediate mask, and the destination any directly addressable byte. Any bit may thus be set, cleared, or complemented with a three-byte, two-cycle instruction if the mask has all bits but one set or cleared.

Byte variables, registers, and indirectly addressed RAM may be moved to a bit addressable register (usually the accumulator) in one instruction. Once transferred, the bits may be tested with a conditional jump, allowing any bit to be polled in 3 microseconds—still much faster than most architectures—or used for logical calculations. (This technique can also simulate additional bit addressing modes with byte operations.)

Parity of bytes or bits. The parity of the current accumulator contents is always available in the PSW, from whence it may be moved to the carry and further processed. Error-correcting Hamming codes and similar applications require computing parity on groups of isolated bits. This can be done by conditionally complementing the carry flag based on those bits or by gathering the bits into the accumulator (as shown in the DES example) and then testing the parallel parity flag.

Multiple byte shift and CRC codes.

Though the 8051 serial port can accommodate eight- or nine-bit data transmissions, some protocols involve much

longer bit streams. The algorithms presented in Design Example 2 can be extended quite readily to 16 or more bits by using multi-byte input and output buffers.

Many mass data storage peripherals and serial communications protocols include Cyclic Redundancy (CRC) codes to verify data integrity. The function is generally computed serially by hardware using shift registers and Exclusive-OR gates, but it can be done with software. As each bit is received into the carry, appropriate bits in the multi-byte data buffer are conditionally complemented based on the incoming data bit. When finished, the CRC register contents may be checked for zero by ORing the two bytes in the accumulator.

4. SUMMARY

A truly unique facet of the Intel MCS-51™ microcomputer family design is the collection of features optimized for the one-bit operations so often desired in real-world, real-time control applications. Included are 17 special instructions, a Boolean accumulator, implicit and direct addressing modes, program and mass data storage, and many I O options. These are the world's first single-chip microcomputers able to efficiently manipulate, operate on, and transfer either bytes or individual bits as data.

This Application Note has detailed the information needed by a microcomputer system designer to make full use of these capabilities. Five design examples were used to contrast the solutions allowed by the 8051 and those required by previous architectures. Depending on the individual application, the 8051 solution will be easier to design, more reliable to implement, debug, and verify, use less program memory, and run up to an order of magnitude faster than the same function implemented on previous digital computer architectures.

Combining byte- and bit-handling capabilities in a single microcomputer has a strong synergistic effect: the power of the result exceeds the power of byte- and bit-processors laboring individually. Virtually all user applications will benefit in some ways from this duality. Data intensive applications will use bit addressing for test pin monitoring or program control flags; control applications will use byte manipulation for parallel I O expansion or arithmetic calculations.

It is hoped that these design examples give the reader an appreciation of these unique features and suggest ways to exploit them in his or her own application.

```
ISIS-II MCS-51 MACRO ASSEMBLER V1.0
OBJECT MODULE PLACED IN :F0:AP70.HEX
ASSEMBLER INVOKED BY:  :f1:asm51 ap70 src date(328)
LOC  OBJ             LINE     SOURCE

                        1     $XREF TITLE(AP-70 APPENDIX)
                        2     ;*******************************************************
                        3     ;
                        4     ;       THE FOLLOWING PROGRAM USES THE BOOLEAN INSTRUCTION SET
                        5     ;       OF THE INTEL 8051 MICROCOMPUTER TO PERFORM A NUMBER OF
                        6     ;       AUTOMOTIVE DASHBOARD CONTROL FUNCTIONS RELATING TO
                        7     ;       TURN SIGNAL CONTROL, EMERGENCY BLINKERS, BRAKE LIGHT
                        8     ;       CONTROL, AND PARKING LIGHT OPERATION.
                        9     ;       THE ALGORITHMS AND HARDWARE ARE DESCRIBED IN DESIGN
                       10     ;       EXAMPLE #4 OF INTEL APPLICATION NOTE AP-70,
                       11     ;               "USING THE INTEL MCS-51(TM)
                       12     ;             BOOLEAN PROCESSING CAPABILITIES"
                       13     ;
                       14     ;*******************************************************
                       15     ;
                       16     ;       INPUT PIN DECLARATIONS:
                       17     ;        (ALL INPUTS ARE POSITIVE-TRUE LOGIC.
                       18     ;       INPUTS ARE HIGH WHEN RESPECTIVE SWITCH CONTACT IS CLOSED.)
                       19     ;
0090                   20     BRAKE   BIT     P1.0    ; BRAKE PEDAL DEPRESSED
0091                   21     EMERG   BIT     P1.1    ; EMERGENCY BLINKER ACTIVATED
0092                   22     PARK    BIT     P1.2    ; PARKING LIGHTS ON
0093                   23     L_TURN  BIT     P1.3    ; TURN LEVER DOWN
0094                   24     R_TURN  BIT     P1.4    ; TURN LEVER UP
                       25     ;
                       26     ;       OUTPUT PIN DECLARATIONS:
                       27     ;       (ALL OUTPUTS ARE POSITIVE TRUE LOGIC.
                       28     ;       BULB IS TURNED ON WHEN OUTPUT PIN IS HIGH.)
                       29     ;
0095                   30     L_FRNT  BIT     P1.5    ; FRONT LEFT-TURN INDICATOR
0096                   31     R_FRNT  BIT     P1.6    ; FRONT RIGHT-TURN INDICATOR
0097                   32     L_DASH  BIT     P1.7    ; DASHBOARD LEFT-TURN INDICATOR
00A0                   33     R_DASH  BIT     P2.0    ; DASHBOARD RIGHT-TURN INDICATOR
00A1                   34     L_REAR  BIT     P2.1    ; REAR LEFT-TURN INDICATOR
00A2                   35     R_REAR  BIT     P2.2    ; REAR RIGHT-TURN INDICATOR
                       36     ;
00A3                   37     S_FAIL  BIT     P2.3    ; ELECTRICAL SYSTEM FAULT INDICATOR
                       38     ;
                       39     ;       INTERNAL VARIABLE DEFINITIONS:
                       40     ;
0020                   41     SUB_DIV DATA    20H             ; INTERRUPT RATE SUBDIVIDER
0000                   42     HI_FREQ BIT     SUB_DIV.0       ; HIGH-FREQUENCY OSCILLATOR BIT
0007                   43     LO_FREQ BIT     SUB_DIV.7       ; LOW-FREQUENCY OSCILLATOR BIT
                       44     ;       .       .
00D1                   45     DIM     BIT     PSW.1           ; PARKING LIGHTS ON FLAG
                       46     ;
                       47     ;=======================================================
                       48 +1  $EJECT
```

```
LOC  OBJ            LINE     SOURCE

                      49             ORG     0000H           ; RESET VECTOR
0000 020040           50             LJMP    INIT
                      51     ;
000B                  52             ORG     000BH           ; TIMER 0 SERVICE VECTOR
000B 758CF0           53             MOV     TH0,#-16        ; HIGH TIMER BYTE ADJUSTED TO CONTROL INT. RATE
000E C0D0             54             PUSH    PSW             ; EXECUTE CODE TO SAVE ANY REGISTERS USED BELOW
0010 0154             55             AJMP    UPDATE          ;      (CONTINUE WITH REST OF ROUTINE)
                      56     ;
0040                  57             ORG     0040H
0040 758A00           58     INIT:   MOV     TL0,#0          ; ZERO LOADED INTO LOW-ORDER BYTE AND
0043 758CF0           59             MOV     TH0,#-16        ; -16 IN HIGH-ORDER BYTE GIVES 4 MSEC PERIOD
0046 758961           60             MOV     TMOD,#01100001B ; 8-BIT AUTO RELOAD COUNTER MODE FOR TIMER 1,
                      61                                     ;  16-BIT TIMER MODE FOR TIMER 0 SELECTED
0049 7520F4           62             MOV     SUB_DIV,#244    ; SUBDIVIDE INTERRUPT RATE BY 244 FOR 1 HZ
004C D2A9             63             SETB    ET0             ; USE TIMER 0 OVERFLOWS TO INTERRUPT PROGRAM
004E D2AF             64             SETB    EA              ; CONFIGURE IE TO GLOBALLY ENABLE INTERRUPTS
0050 D28C             65             SETB    TR0             ; KEEP INSTRUCTION CYCLE COUNT UNTIL OVERFLOW
0052 80FE             66             SJMP    $               ; START BACKGROUND PROGRAM EXECUTION
                      67     ;
                      68     ;
0054 D52038           69     UPDATE: DJNZ    SUB_DIV,TOSERV  ; EXECUTE SYSTEM TEST ONLY ONCE PER SECOND
0057 7520F4           70             MOV     SUB_DIV,#244    ; GET VALUE FOR NEXT ONE SECOND DELAY AND
                      71     ;                               ; GO THROUGH ELECTRICAL SYSTEM TEST CODE:
005A 4390E0           72             ORL     P1,#11100000B   ; SET CONTROL OUTPUTS HIGH
005D 43A007           73             ORL     P2,#00000111B
0060 C295             74             CLR     L_FRNT          ; FLOAT DRIVE COLLECTOR
0062 20B428           75             JB      T0,FAULT        ; TO SHOULD BE PULLED LOW
0065 D295             76             SETB    L_FRNT          ; PULL COLLECTOR BACK DOWN
0067 C297             77             CLR     L_DASH          ; REPEAT SEQUENCE FOR L_DASH,
0069 20B421           78             JB      T0,FAULT
006C D297             79             SETB    L_DASH
006E C2A1             80             CLR     L_REAR          ;                 L_REAR,
0070 20B41A           81             JB      T0,FAULT
0073 D2A1             82             SETB    L_REAR
0075 C296             83             CLR     R_FRNT          ;                 R_FRNT,
0077 20B413           84             JB      T0,FAULT
007A D296             85             SETB    R_FRNT
007C C2A0             86             CLR     R_DASH          ;                 R_DASH,
007E 20B40C           87             JB      T0,FAULT
0081 D2A0             88             SETB    R_DASH
0083 C2A2             89             CLR     R_REAR          ;                 AND R_REAR.
0085 20B405           90             JB      T0,FAULT
0088 D2A2             91             SETB    R_REAR
                      92     ;
                      93     ;       WITH ALL COLLECTORS GROUNDED, T0 SHOULD BE HIGH
                      94     ;       IF SO, CONTINUE WITH INTERRUPT ROUTINE.
                      95     ;
008A 20B402           96             JB      T0,TOSERV
008D B2A3             97     FAULT:  CPL     S_FAIL          ; ELECTRICAL FAILURE PROCESSING ROUTINE
                      98                                     ; (TOGGLE INDICATOR ONCE PER SECOND)
                      99 +1  $EJECT
```

```
LOC  OBJ            LINE     SOURCE

                     100     ;       CONTINUE WITH INTERRUPT PROCESSING:
                     101     ;
                     102     ;   1)  COMPUTE LOW BULB INTENSITY WHEN PARKING LIGHTS ARE ON.
                     103     ;
008F A201            104     TOSERV: MOV     C,SUB_DIV.1     ; START WITH 50 PERCENT,
0091 8200            105             ANL     C,SUB_DIV.0     ; MASK DOWN TO 25 PERCENT,
0093 7202            106             ORL     C,SUB_DIV.2     ; BUILD BACK TO 62.5 PERCENT,
0095 8292            107             ANL     C,PARK          ; GATE WITH PARKING LIGHT SWITCH,
0097 92D1            108             MOV     DIM,C           ; AND SAVE IN TEMP. VARIABLE.
                     109     ;
                     110     ;   2)  COMPUTE AND OUTPUT LEFT-HAND DASHBOARD INDICATOR.
                     111     ;
0099 A293            112             MOV     C,L_TURN        ; SET CARRY IF TURN
009B 7291            113             ORL     C,EMERG         ; OR EMERGENCY SELECTED.
009D 8207            114             ANL     C,LO_FREQ       ; IF SO, GATE IN 1 HZ SIGNAL
009F 9297            115             MOV     L_DASH,C        ; AND OUTPUT TO DASHBOARD.
                     116     ;
                     117     ;   3)  COMPUTE AND OUTPUT LEFT-HAND FRONT TURN SIGNAL.
                     118     ;
00A1 92D5            119             MOV     F0,C            ; SAVE FUNCTION SO FAR.
00A3 72D1            120             ORL     C,DIM           ; ADD IN PARKING LIGHT FUNCTION
00A5 9295            121             MOV     L_FRNT,C        ; AND OUTPUT TO TURN SIGNAL.
                     122     ;
                     123     ;   4)  COMPUTE AND OUTPUT LEFT-HAND REAR TURN SIGNAL.
                     124     ;
00A7 A290            125             MOV     C,BRAKE         ; GATE BRAKE PEDAL SWITCH
00A9 B093            126             ANL     C,/L_TURN       ; WITH TURN LEVER.
00AB 72D5            127             ORL     C,F0            ; INCLUDE TEMP. VARIABLE FROM DASH
00AD 72D1            128             ORL     C,DIM           ; AND PARKING LIGHT FUNCTION
00AF 92A1            129             MOV     L_REAR,C        ; AND OUTPUT TO TURN SIGNAL.
                     130     ;
                     131     ;   5)  REPEAT ALL OF ABOVE FOR RIGHT-HAND COUNTERPARTS.
                     132     ;
00B1 A294            133             MOV     C,R_TURN        ; SET CARRY IF TURN
00B3 7291            134             ORL     C,EMERG         ; OR EMERGENCY SELECTED.
00B5 8207            135             ANL     C,LO_FREQ       ; IF SO, GATE IN 1 HZ SIGNAL
00B7 92A0            136             MOV     R_DASH,C        ; AND OUTPUT TO DASHBOARD.
00B9 92D5            137             MOV     F0,C            ; SAVE FUNCTION SO FAR.
00BB 72D1            138             ORL     C,DIM           ; ADD IN PARKING LIGHT FUNCTION
00BD 9296            139             MOV     R_FRNT,C        ; AND OUTPUT TO TURN SIGNAL.
00BF A290            140             MOV     C,BRAKE         ; GATE BRAKE PEDAL SWITCH
00C1 B094            141             ANL     C,/R_TURN       ; WITH TURN LEVER.
00C3 72D5            142             ORL     C,F0            ; INCLUDE TEMP. VARIABLE FROM DASH
00C5 72D1            143             ORL     C,DIM           ; AND PARKING LIGHT FUNCTION
00C7 92A2            144             MOV     R_REAR,C        ; AND OUTPUT TO TURN SIGNAL.
                     145     ;
                     146     ;       RESTORE STATUS REGISTER AND RETURN.
                     147     ;
00C9 D0D0            148             POP     PSW             ; RESTORE PSW
00CB 32              149             RETI                    ; AND RETURN FROM INTERRUPT ROUTINE
                     150     ;
                     151             END
```

```
XREF SYMBOL TABLE LISTING
---- ------ ----- -------


NAME      TYPE      VALUE AND REFERENCES

BRAKE .   N BSEG    0090H  20# 125 140
DIM . .   N BSEG    00D1H  45# 108 120 128 138 143
EA. . .   N BSEG    00AFH  64
EMERG .   N BSEG    0091H  21# 113 134
ETO . .   N BSEG    00A9H  63
FO. . .   N BSEG    00D5H  119 127 137 142
FAULT .   L CSEG    008DH  75 78 81 84 87 90 97#
HI_FREQ   N BSEG    0000H  42#
INIT. .   L CSEG    0040H  50 58#
L_DASH.   N BSEG    0097H  32# 77 79 115
L_FRNT.   N BSEG    0095H  30# 74 76 121
L_REAR.   N BSEG    00A1H  34# 80 82 129
L_TURN.   N BSEG    0093H  23# 112 126
LO_FREQ   N BSEG    0007H  43# 114 135
P1. . .   N DSEG    0090H  20 21 22 23 24 30 31 32 72
P2. . .   N DSEG    00A0H  33 34 35 37 73
PARK. .   N BSEG    0092H  22# 107
PSW . .   N DSEG    00D0H  45 54 148
R_DASH.   N BSEG    00A0H  33# 86 88 136
R_FRNT.   N BSEG    0096H  31# 83 85 139
R_REAR.   N BSEG    00A2H  35# 89 91 144
R_TURN.   N BSEG    0094H  24# 133 141
S_FAIL.   N BSEG    00A3H  37# 97
SUB_DIV   N DSEG    0020H  41# 42 43 62 69 70 104 105 106
TO. . .   N BSEG    00B4H  75 78 81 84 87 90 96
TOSERV.   L CSEG    008FH  69 96 104#
THO . .   N DSEG    008CH  53 59
TLO . .   N DSEG    008AH  58
TMOD. .   N DSEG    0089H  60
TRO . .   N BSEG    008CH  65
UPDATE.   L CSEG    0054H  55 69#

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

Designing Microcontroller Systems for Electrically Noisy Environments

Contents

SYMPTOMS OF NOISE PROBLEMS 9-106

TYPES AND SOURCES OF ELECTRICAL NOISE 9-106
- Supply Line Transients 9-106
- EMP and RFI 9-106
- ESD 9-107
- Ground Noise 9-107

"RADIATED" AND "CONDUCTED" NOISE 9-107

SIMULATING THE ENVIRONMENT 9-108

TYPES OF FAILURES AND FAILURE MECHANISMS 9-108

THE GAME PLAN 9-109

CURRENT LOOPS 9-109

SHIELDING 9-110
- Shielding Against Capacitive Coupling 9-110
- Shielding Against Inductive Coupling 9-110
- RF Shielding 9-113

GROUNDS 9-114
- Safety Ground 9-114
- Signal Ground 9-115
- Practical Grounding 9-116
- Braided Cable 9-116

POWER SUPPLY DISTRIBUTION AND DECOUPLING 9-118
- Selecting the Value of the Decoupling Cap .. 9-119
- The Case for On-Board Voltage Regulation . 9-120

RECOVERING GRACEFULLY FROM A SOFTWARE UPSET 9-120

SPECIAL PROBLEM AREAS 9-122
- ESD 9-122
- The Automotive Environment 9-123

PARTING THOUGHTS 9-125

REFERENCES 9-126

Figures 23 and 26 reprinted with the permission of General Semiconductor Industries from *TranZorb® Quick Reference Guide.* TranZorb is a registered trademark of General Semiconductor Industries.

Figure 2A reprinted with permission from T.J. Tucker, "Spark Initiation Requirements of a Secondary Explosive," *Annals of the New York Academy of Sciences,* vol. 152, article I, 1968.

Figure 24 reprinted with permission from "Recommended Environmental Practices for Electronic Equipment Design," *1980 SAE Handbook.* Figure 25 reprinted with permission from Kearney, Shreve and Vincent, "Microprocessor Based Systems in the Automobile," Electronic Engine Management and Driveline Control Systems, *SAE Publication SP-481.* ©1980 and 1981, respectively, Society of Automotive Engineers, Inc.

Figures 8 and 20 reprinted with permission from H. Ott, "Digital Circuit Grounding and Interconnection," and Figure 2B from King and Reynolds, "Personnel Electrostatic Discharge: Impulse Waveforms Resulting from ESD of Humans Directly and through Small Hand-Held Metallic Objects Intervening in the Discharge Path," *Proceedings of the IEEE Symposium on Electromagnetic Compatibility,* August 1981. ©1981 IEEE.

Figures 9, 10, 11, 12, 16, and 18 reprinted with permission from H. Ott, *Noise Reduction Techniques in Electronic Systems,* John Wiley & Sons, Inc., 1976.

Digital circuits are often thought of as being immune to noise problems, but really they're not. Noises in digital systems produce software upsets: program jumps to apparently random locations in memory. Noise-induced glitches in the signal lines can cause such problems, but the supply voltage is more sensitive to glitches than the signal lines.

Severe noise conditions, those involving electrostatic discharges, or as found in automotive environments, can do permanent damage to the hardware. Electrostatic discharges can blow a crater in the silicon. In the automotive environment, in ordinary operation, the "12V" power line can show + and -400V transients.

This Application Note describes some electrical noises and noise environments. Design considerations, along the lines of PCB layout, power supply distribution and decoupling, and shielding and grounding techniques, that may help minimize noise susceptibility are reviewed. Special attention is given to the automotive and ESD environments.

Symptoms of Noise Problems

Noise problems are not usually encountered during the development phase of a microcontroller system. This is because benches rarely simulate the system's intended environment. Noise problems tend not to show up until the system is installed and operating in its intended environment. Then, after a few minutes or hours of normal operation the system finds itself someplace out in left field. Inputs are ignored and outputs are gibberish. The system may respond to a reset, or it may have to be turned off physically and then back on again, at which point it commences operating as though nothing had happened. There may be an obvious cause, such as an electrostatic discharge from somebody's finger to a keyboard or the upset occurs every time a copier machine is turned on or off. Or there may be no obvious cause, and nothing the operator can do will make the upset repeat itself. But a few minutes, or a few hours, or a few days later it happens again.

One symptom of electrical noise problems is randomness, both in the occurrence of the problem and in what the system does in its failure. All operational upsets that occur at seemingly random intervals are not necessarily caused by noise in the system. Marginal VCC, inadequate decoupling, rarely encountered software conditions, or timing coincidences can produce upsets that seem to occur randomly. On the other hand, some noise sources can produce upsets downright periodically. Nevertheless, the more difficult it is to characterize an upset as to cause and effect, the more likely it is to be a noise problem.

Types and Sources of Electrical Noise

The name given to electrical noises other than those that are inherent in the circuit components (such as thermal noise) is EMI: electromagnetic interference. Motors, power switches, fluorescent lights, electrostatic discharges, etc., are sources of EMI. There is a veritable alphabet soup of EMI types, and these are briefly described below.

SUPPLY LINE TRANSIENTS

Anything that switches heavy current loads onto or off of AC or DC power lines will cause large transients in these power lines. Switching an electric typewriter on or off, for example, can put a 1000V spike onto the AC power lines.

The basic mechanism behind supply line transients is shown in Figure 1. The battery represents any power source, AC or DC. The coils represent the line inductance between the power source and the switchable loads R1 and R2. If both loads are drawing current, the line current flowing through the line inductance establishes a magnetic field of some value. Then, when one of the loads is switched off, the field due to that component of the line current collapses, generating transient voltages, $v = L(di/dt)$, which try to maintain the current at its original level. That's called an "inductive kick." Because of contact bounce, transients are generated whether the switch is being opened or closed, but they're worse when the switch is being opened.

An inductive kick of one type or another is involved in most line transients, including those found in the automotive environment. Other mechanisms for line transients exist, involving noise pickup on the lines. The noise voltages are then conducted to a susceptible circuit right along with the power.

EMP AND RFI

Anything that produces arcs or sparks will radiate electromagnetic pulses (EMP) or radio-frequency interference (RFI).

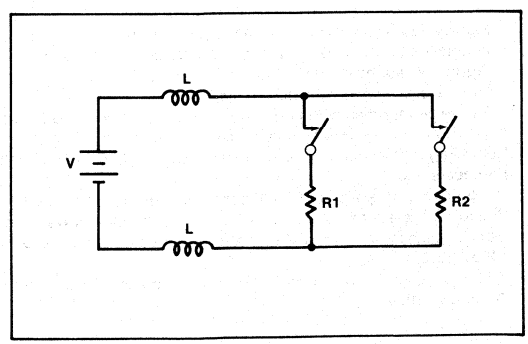


Figure 1. Supply Line Transients

Spark discharges have probably caused more software upsets in digital equipment than any other single noise source. The upsetting mechanism is the EMP produced by the spark. The EMP induces transients in the circuit, which are what actually cause the upset.

Arcs and sparks occur in automotive ignition systems, electric motors, switches, static discharges, etc. Electric motors that have commutator bars produce an arc as the brushes pass from one bar to the next. DC motors and the "universal" (AC/DC) motors that are used to power hand tools are the kinds that have commutator bars. In switches, the same inductive kick that puts transients on the supply lines will cause an opening or closing switch to throw a spark.

ESD

Electrostatic discharge (ESD) is the spark that occurs when a person picks up a static charge from walking across a carpet, and then discharges it into a keyboard, or whatever else can be touched. Walking across a carpet in a dry climate, a person can accumulate a static voltage of 35kV. The current pulse from an electrostatic discharge has an extremely fast risetime — typically, 4A/nsec. Figure 2 shows ESD waveforms that have been observed by some investigators of ESD phenomena.

It is enlightening to calculate the L(di/dt) voltage required to drive an ESD current pulse through a couple of inches of straight wire. Two inches of straight wire has about 50nH of inductance. That's not very much, but using 50nH for L and 4A/nsec for di/dt gives an L(di/dt) drop of about 200V. Recent observations by W.M. King suggest even faster risetimes (Figure 2B) and the occurrence of multiple discharges during a single discharge event.

Obviously, ESD-sensitivity needs to be considered in the design of equipment that is going to be subjected to it, such as office equipment.

GROUND NOISE

Currents in ground lines are another source of noise. These can be 60Hz currents from the power lines, or RF hash, or crosstalk from other signals that are sharing this particular wire as a signal return line. Noise in the ground lines is often referred to as a "ground loop" problem. The basic concept of the ground loop is shown in Figure 3. The problem is that true earth-ground is not really at the same potential in all locations. If the two ends of a wire are earth-grounded at different locations, the voltage difference between the two "ground" points can drive significant currents (several amperes) through the wire. Consider the wire to be part of a loop which contains, in addition to the wire, a voltage source that represents the difference in potential between the two ground points, and you have the classical "ground loop." By extension, the term is used to refer to any unwanted (and often unexpected) currents in a ground line.

"Radiated" and "Conducted" Noise

Radiated noise is noise that arrives at the victim circuit in the form of electromagnetic radiation, such as EMP and RFI. It causes trouble by inducing extraneous voltages in the circuit. Conducted noise is noise that arrives at the victim circuit already in the form of an extraneous voltage, typically via the AC or DC power lines.

One defends against radiated noise by care in designing layouts and the use of effective shielding techniques. One defends against conducted noise with filters and suppres-

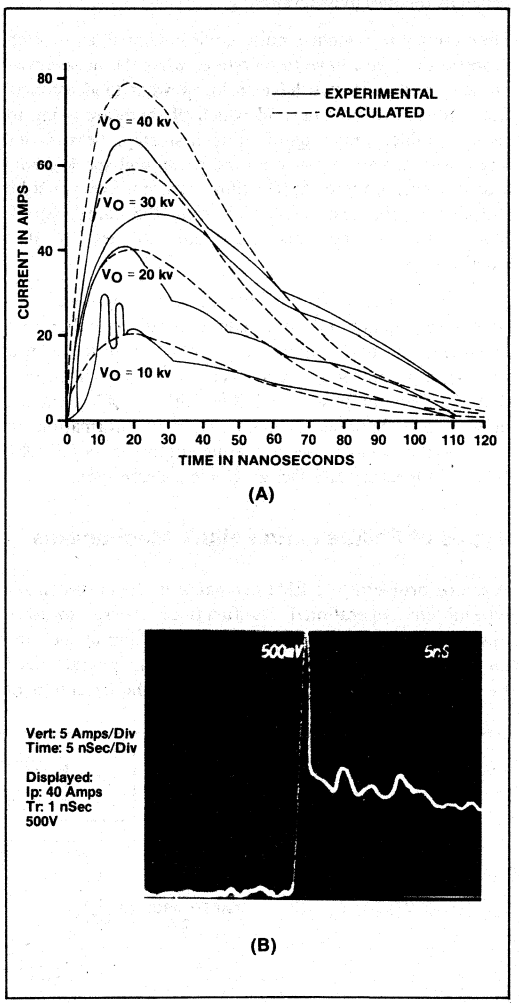


Figure 2. Waveforms of Electrostatic Discharge Currents From a Hand-Held Metallic Object

sors, although layouts and grounding techniques are important here, too.

Simulating the Environment

Addressing noise problems after the design of a system has been completed is an expensive proposition. The ill will generated by failures in the field is not cheap either. It's cheaper in the long run to invest a little time and money in learning about noise and noise simulation equipment, so that controlled tests can be made on the bench as the design is developing.

Simulating the intended noise environment is a two-step process: First you have to recognize what the noise environment is, that is, you have to know what kinds of electrical noises are present, and which of them are going to cause trouble. Don't ignore this first step, because it's important. If you invest in an induction coil spark generator just because your application is automotive, you'll be straining at the gnat and swallowing the camel. Spark plug noise is the least of your worries in that environment.

The second step is to generate the electrical noise in a controlled manner. This is usually more difficult than first imagined; one first imagines the simulation in terms of a waveform generator and a few spare parts, and then finds that a wideband power amplifier with a 200V dynamic range is also required. A good source of information on who supplies what noise-simulating equipment is the 1981 "ITEM" Directory and Design Guide (reference 6).

Types of Failures and Failure Mechanisms

A major problem that EMI can cause in digital systems is intermittent operational malfunction. These software upsets occur when the system is in operation at the time an EMI source is activated, and are usually characterized by a loss of information or a jump in the execution of the program to some random location in memory. The person who has to iron out such problems is tempted to say the program counter went crazy. There is usually no damage to the hardware, and normal operation can resume as soon as the EMI has passed or the source is de-activated. Resuming normal operation usually requires manual or automatic reset, and possibly re-entering of lost information.

Electrostatic discharges from operating personnel can cause not only software upsets, but also permanent ("hard") damage to the system. For this to happen the system doesn't even have to be in operation. Sometimes the permanent damage is latent, meaning the initial damage may be marginal and require further aggravation through operating stress and time before permanent failure takes place. Sometimes too the damage is hidden.

One ESD-related failure mechanism that has been identified has to do with the bias voltage on the substrate of the chip. On some CPU chips the substrate is held at -2.5V by a phase-shift oscillator working into a capacitor/diode clamping circuit. This is called a "charge pump" in chip-design circles. If the substrate wanders too far in either direction, program read errors are noted. Some designs have been known to allow electrostatic discharge currents to flow directly into port pins of an 8048. The resulting damage to the oxide causes an increase in leakage current, which loads down the charge pump, reducing the substrate voltage to a marginal or unacceptable level. The system is then unreliable or completely inoperative until the CPU chip is replaced. But if the CPU chip was subjected to a discharge spark once, it will eventually happen again.

Chips that have a grounded substrate, such as the 8748, can sometimes sustain some oxide damage without actually becoming inoperative. In this case the damage is present, and the increased leakage current is noted; however, since the substrate voltage retains its design value, the damage is largely hidden.

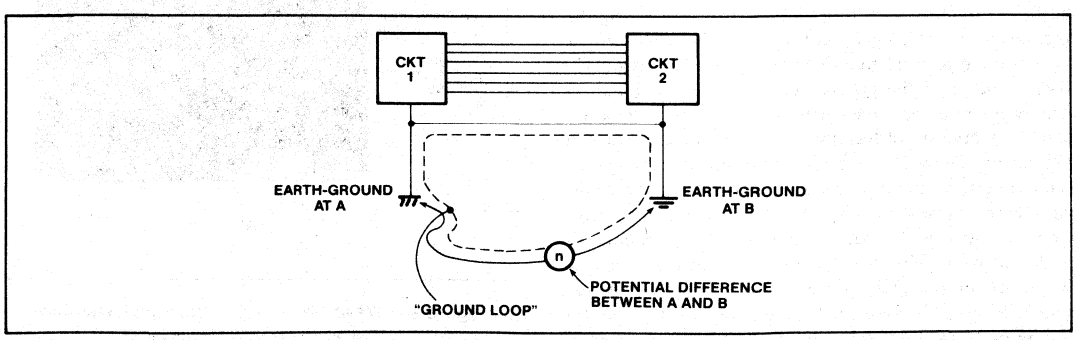


Figure 3. What a Ground Loop Is

It must therefore be recognized that connecting port pins unprotected to a keyboard or to anything else that is subject to electrostatic discharges, makes an extremely dangerous configuration. It doesn't make any difference what CPU chip is being used, or who makes it. If it connects unprotected to a keyboard, it will eventually be destroyed. Designing for an ESD-environment will be discussed further on.

We might note here that MOS chips are not the only components that are susceptible to permanent ESD damage. Bipolar and linear chips can also be damaged in this way. PN junctions are subject to a hard failure mechanism called thermal secondary breakdown, in which a current spike, such as from an electrostatic discharge, causes microscopically localized spots in the junction to approach melt temperatures. Low power TTL chips are subject to this type of damage, as are op-amps. Op-amps, in addition, often carry on-chip MOS capacitors which are directly across an external pin combination, and these are susceptible to dielectric breakdown.

We return now to the subject of software upsets. Noise transients can upset the chip through any pin, even an output pin, because every pin on the chip connects to the substrate through a pn junction. However, the most vulnerable pin is probably the VCC line, since it has direct access to all parts of the chip: every register, gate, flip-flop and buffer.

The menu of possible upset mechanisms is quite lengthy. A transient on the substrate at the wrong time will generally cause a program read error. A false level at a control input can cause an extraneous or misdirected opcode fetch. A disturbance on the supply line can flip a bit in the program counter or instruction register. A short interruption or reversal of polarity on the supply line can actually turn the processor off, but not long enough for the power-up reset capacitor to discharge. Thus when the transient ends, the chip starts up again without a reset.

A common failure mode is for the processor to lock itself into a tight loop. Here it may be executing the data in a table, or the program counter may have jumped a notch, so that the processor is now executing operands instead of opcodes, or it may be trying to fetch opcodes from a nonexistent external program memory.

It should be emphasized that mechanisms for upsets have to do with the arrival of noise-induced transients at the pins of the chips, rather than with the generation of noise pulses within the chip itself, that is, it's not the chip that is picking up noise, it's the circuit.

The Game Plan

Prevention is usually cheaper than suppression, so first we'll consider some preventive methods that might help to minimize the generation of noise voltages in the circuit. These methods involve grounding, shielding, and wiring techniques that are directed toward the mechanisms by which noise voltages are generated in the circuit. We'll also discuss methods of decoupling. Then we'll look at some schemes for making a graceful recovery from upsets that occur in spite of preventive measures. Lastly, we'll take another look at two special problem areas: electrostatic discharges and the automotive environment.

Current Loops

The first thing most people learn about electricity is that current won't flow unless it can flow in a closed loop. This simple fact is sometimes temporarily forgotten by the overworked engineer who has spent the past several years mastering the intricacies of the DO loop, the timing loop, the feedback loop, and maybe even the ground loop. The simple current loop probably owes its apparent demise to the invention of the ground symbol. By a stroke of the pen one avoids having to draw the return paths of most of the current loops in the circuit. Then "ground" turns into an infinite current sink, so that any current that flows into it is gone and forgotten. Forgotten it may be, but it's not gone. It must return to its source, so that its path will by all the laws of nature form a closed loop.

The physical geometry of a given current loop is the key to why it generates EMI, why it's susceptible to EMI, and how to shield it. Specifically, it's the area of the loop that matters.

Any flow of current generates a magnetic field whose intensity varies inversely to the distance from the wire that carries the current. Two parallel wires conducting currents +I and –I (as in signal feed and return lines) would generate a nonzero magnetic field near the wires, where the distance from a given point to one wire is noticeably different from the distance to the other wire, but farther away (relative to the wire spacing), where the distances from a given point to either wire are about the same, the fields from both wires tend to cancel out. Thus, maintaining proximity between feed and return paths is an important way to minimize their interference with other signals. The way to maintain their proximity is essentially to minimize their loop area. And, because the mutual inductance from current loop A to current loop B is the same as the mutual inductance from current loop B to current loop A, a circuit that doesn't radiate interference doesn't receive it either.

Thus, from the standpoint of reducing both generation of EMI and susceptibility to EMI, the hard rule is to keep loop areas small. To say that loop areas should be minimized is the same as saying the circuit inductance should

be minimized. Inductance is by definition the constant of proportionality between current and the magnetic field it produces: $\phi = LI$. Holding the feed and return wires close together so as to promote field cancellation can be described either as minimizing the loop area or as minimizing L. It's the same thing.

Shielding

There are three basic kinds of shields: shielding against capacitive coupling, shielding against inductive coupling, and RF shielding. Capacitive coupling is electric field coupling, so shielding against it amounts to shielding against electric fields. As will be seen, this is relatively easy. Inductive coupling is magnetic field coupling, so shielding against it is shielding against magnetic fields. This is a little more difficult. Strangely enough, this type of shielding does not in general involve the use of magnetic materials. RF shielding, the classical "metallic barrier" against all sorts of electromagnetic fields, is what most people picture when they think about shielding. Its effectiveness depends partly on the selection of the shielding material, but mostly, as it turns out, on the treatment of its seams and the geometry of its openings.

SHIELDING AGAINST CAPACITIVE COUPLING

Capacitive coupling involves the passage of interfering signals through mutual or stray capacitances that aren't shown on the circuit diagram, but which the experienced engineer knows are there. Capacitive coupling to one's body is what would cause an unstable oscillator to change its frequency when the person reaches his hand over the circuit, for example. More importantly, in a digital system it causes crosstalk in multi-wire cables.

The way to block capacitive coupling is to enclose the circuit or conductor you want to protect in a metal shield. That's called an electrostatic or Faraday shield. If coverage is 100%, the shield does not have to be grounded, but it usually is, to ensure that circuit-to-shield capacitances go to signal reference ground rather than act as feedback and crosstalk elements. Besides, from a mechanical point of view, grounding it is almost inevitable.

A grounded Faraday shield can be used to break capacitive coupling between a noisy circuit and a victim circuit, as shown in Figure 4. Figure 4A shows two circuits capacitively coupled through the stray capacitance between them. In Figure 4B the stray capacitance is intercepted by a grounded Faraday shield, so that interference currents are shunted to ground. For example, a grounded plane can be inserted between PCBs (printed circuit boards) to eliminate most of the capacitive coupling between them.

Another application of the Faraday shield is in the electrostatically shielded transformer. Here, a conducting foil is laid between the primary and secondary coils so as to intercept the capacitive coupling between them. If a system is being upset by AC line transients, this type of transformer may provide the fix. To be effective in this application, the shield must be connected to the green-wire ground.

SHIELDING AGAINST INDUCTIVE COUPLING

With inductive coupling, the physical mechanism involved is a magnetic flux density B from some external interference source that links with a current loop in the victim circuit, and generates a voltage in the loop in accordance with Lenz's law: $v = -NA(dB/dt)$, where in this case N = 1 and A is the area of the current loop in the victim circuit.

There are two aspects to defending a circuit against inductive pickup. One aspect is to try to minimize the offensive fields at their source. This is done by minimizing the area of the current loop at the source so as to promote field cancellation, as described in the section on current loops. The other aspect is to minimize the inductive pickup in the victim circuit by minimizing the area of that current loop, since, from Lenz's law, the induced voltage is proportional to this area. So the two aspects really involve the same corrective action: minimize the areas of the current loops. In other words, minimizing the offensiveness of a circuit inherently minimizes its susceptibility.

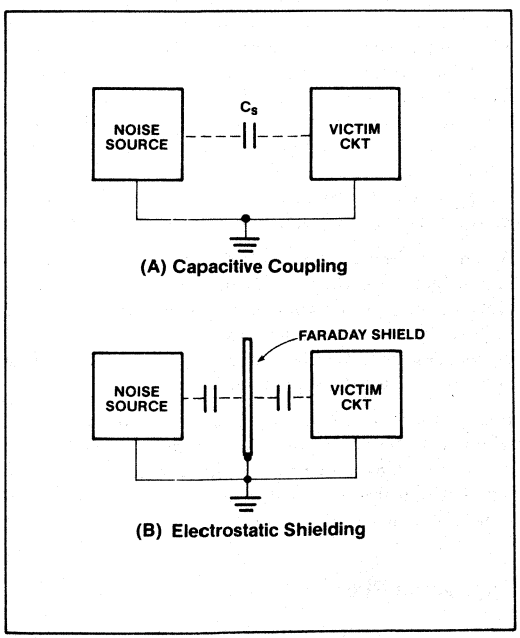


Figure 4. Use of Faraday Shield

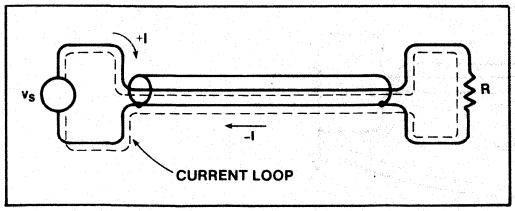


Figure 5. External to the Shield, ϕ=0

Shielding against inductive coupling means nothing more nor less than controlling the dimensions of the current loops in the circuit. We must look at four examples of this type of "shielding": the coaxial cable, the twisted pair, the ground plane, and the gridded-ground PCB layout.

The Coaxial Cable — Figure 5 shows a coaxial cable carrying a current I from a signal source to a receiving load. The shield carries the same current as the center conductor. Outside the shield, the magnetic field produced by +I flowing in the center conductor is cancelled by the field produced by -I flowing in the shield. To the extent that the cable is ideal in producing zero external magnetic field, it is immune to inductive pickup from external sources. The cable adds effectively zero area to the loop. This is true only if the shield carries the same current as the center conductor.

In the real world, both the signal source and the receiving load are likely to have one end connected to a common signal ground. In that case, should the cable be grounded at one end, both ends, or neither end? The answer is that it should be grounded at both ends. Figure 6A shows the situation when the cable shield is grounded at only one end. In that case the current loop runs down the center conductor of the cable, then back through the common ground connection. The loop area is not well defined. The shield not only does not carry the same current as the center conductor, but it doesn't carry any current at all. There is no field cancellation at all. The shield has no effect whatsoever on either the generation of EMI or susceptibility to EMI. (It is, however, still effective as an electrostatic shield, or at least it would be if the shield coverage were 100%.)

Figure 6B shows the situation when the cable is grounded at both ends. Does the shield carry all of the return current, or only a portion of it on account of the shunting effect of the common ground connection? The answer to that question depends on the frequency content of the signal. In general, the current loop will follow the path of least impedance. At low frequencies, 0Hz to several kHz, where the inductive reactance is insignificant, the current will follow the path of least resistance. Above a few kHz, where inductive reactance predominates, the current will follow the path of least inductance. The path of least

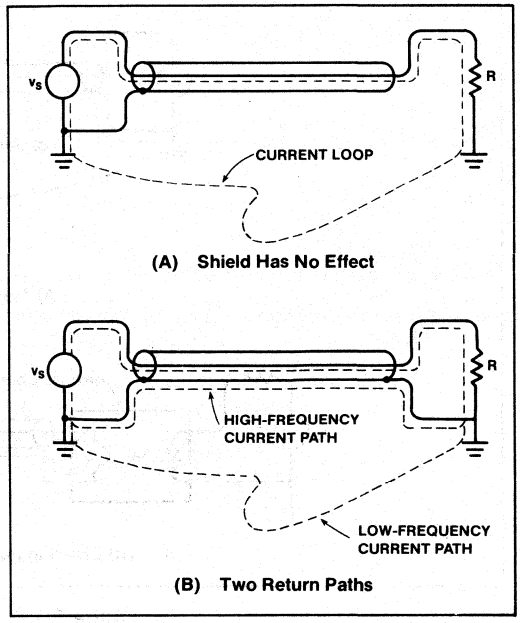


Figure 6. Use of Coaxial Cable

inductance is the path of minimum loop area. Hence, for higher frequencies the shield carries virtually the same current as the center conductor, and is therefore effective against both generation and reception of EMI.

Note that we have now introduced the famous "ground loop" problem, as shown in Figure 7A. Fortunately, a digital system has some built-in immunity to moderate ground loop noise. In a noisy environment, however, one can break the ground loop, and still maintain the shielding effectiveness of the coaxial cable, by inserting an optical coupler, as shown in Figure 7B. What the optical coupler does, basically, is allow us to re-define the signal source as being ungrounded, so that that end of the cable need not be grounded, and still lets the shield carry the same current as the center conductor. Obviously, if the signal source weren't grounded in the first place, the optical coupler wouldn't be needed.

The Twisted Pair — A cheaper way to minimize loop area is to run the feed and return wires right next to each other. This isn't as effective as a coaxial cable in minimizing loop area. An ideal coaxial cable adds zero area to the loop, whereas merely keeping the feed and return wires next to each other is bound to add a finite area.

However, two things work to make this cheaper method almost as good as a coaxial cable. First, real coaxial cables are not ideal. If the shield current isn't evenly distributed around the center conductor at every cross-

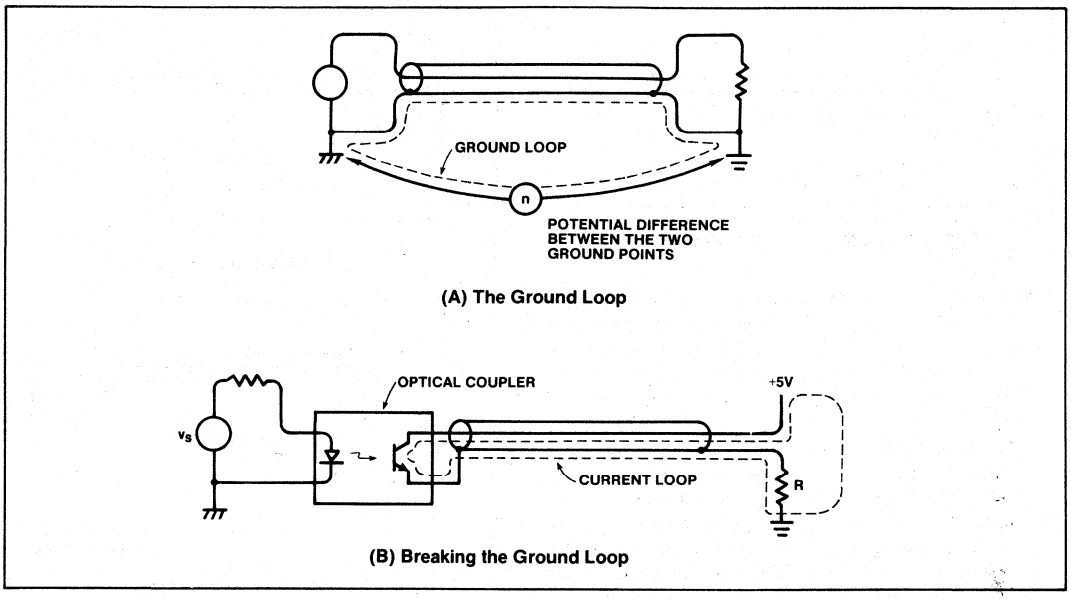


Figure 7. Use of Optical Coupler

section of the cable (it isn't), then field cancellation external to the shield is incomplete. If field cancellation is incomplete, then the effective area added to the loop by the cable isn't zero. Second, in the cheaper method the feed and return wires can be twisted together. This not only maintains their proximity, but the noise picked up in one twist tends to cancel out the noise picked up in the next twist down the line. Thus the "twisted pair" turns out to be about as good a shield against inductive coupling as coaxial cable is.

The twisted pair does not, however, provide electrostatic shielding (i.e., shielding against capacitive coupling). Another operational difference between them is that the coaxial cable works better at higher frequencies. This is primarily because the twisted pair adds more capacitive loading to the signal source than the coaxial cable does. The twisted pair is normally considered useful up to only about 1MHz, as opposed to near a GHz for the coaxial cable.

The Ground Plane — The best way to minimize loop areas when many current loops are involved is to use a ground plane. A ground plane is a conducting surface that is to serve as a return conductor for all the current loops in the circuit. Normally, it would be one or more layers of a multilayer PCB. All ground points in the circuit go not to a grounded trace on the PCB, but directly to the ground plane. This leaves each current loop in the circuit free to complete itself in whatever configuration yields minimum loop area (for frequencies wherein the ground path impedance is primarily inductive).

Thus, if the feed path for a given signal zigzags its way across the PCB, the return path for this signal is free to zigzag right along beneath it on the ground plane, in such a configuration as to minimize the energy stored in the magnetic field produced by this current loop. Minimal magnetic flux means minimal effective loop area and minimal susceptibility to inductive coupling.

The Gridded-Ground PCB Layout — The next best thing to a ground plane is to lay out the ground traces on a PCB in the form of a grid structure, as shown in Figure 8. Laying horizontal traces on one side of the board and vertical traces on the other side allows the passage of signal and power traces. Wherever vertical and horizontal ground traces cross, they must be connected by a feed-through.

Have we not created here a network of "ground loops"? Yes, in the literal sense of the word, but loops in the ground layout on a PCB are not to be feared. Such inoffensive little loops have never caused as much noise pickup as their avoidance has. Trying to avoid innocent little loops in the ground layout, PCB designers have forced current loops into geometries that could swallow a whale. That is exactly the wrong thing to do.

The gridded ground structure works almost as well as the ground plane, as far as minimizing loop area is concerned. For a given current loop, the primary return path may have to zig once in a while where its feed path zags,

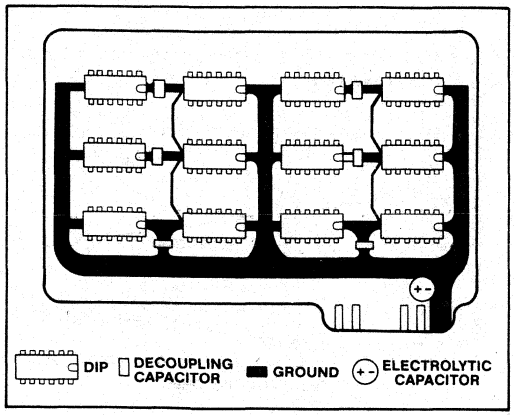


Figure 8. PCB with Gridded Ground

but you still get a mathematically optimal distribution of currents in the grid structure, such that the current loop produces less magnetic flux than if the return path were restrained to follow any single given ground trace. The key to attaining minimum loop areas for all the current loops together is to let the ground currents distribute themselves around the entire area of the board as freely as possible. They want to minimize their own magnetic field. Just let them.

RF SHIELDING

A time-varying electric field generates a time-varying magnetic field, and vice versa. Far from the source of a time-varying EM field, the ratio of the amplitudes of the electric and magnetic fields is always 377 ohms. Up close to the source of the fields, however, this ratio can be quite different, and dependent on the nature of the source. Where the ratio is near 377 ohms is called the far field, and where the ratio is significantly different from 377 ohms is called the near field. The ratio itself is called the wave impedance, E/H.

The near field goes out about 1/6 of a wavelength from the source. At 1MHz this is about 150 feet, and at 10MHz it's about 15 feet. That means if an EMI source is in the same room with the victim circuit, it's likely to be a near field problem. The reason this matters is that in the near field an RF interference problem could be almost entirely due to E-field coupling or H-field coupling, and that could influence the choice of an RF shield or whether an RF shield will help at all.

In the near field of a whip antenna, the E/H ratio is higher than 377 ohms, which means it's mainly an E-field generator. A wire-wrap post can be a whip antenna. Interference from a whip antenna would be by electric field coupling, which is basically capacitive coupling. Methods to protect a circuit from capacitive coupling, such as a Faraday shield, would be effective against RF interference from a whip antenna. A gridded-ground structure would be less effective.

In the near field of a loop antenna, the E/H ratio is lower than 377 ohms, which means it's mainly an H-field generator. Any current loop is a loop antenna. Interference from a loop antenna would be by magnetic field coupling, which is basically the same as inductive coupling. Methods to protect a circuit from inductive coupling, such as a gridded-ground structure, would be effective against RF interference from a loop antenna. A Faraday shield would be less effective.

A more difficult case of RF interference, near field or far field, may require a genuine metallic RF shield. The idea behind RF shielding is that time-varying EMI fields induce currents in the shielding material. The induced currents dissipate energy in two ways: I^2R losses in the shielding material and radiation losses as they re-radiate their own EM fields. The energy for both of these mechanisms is drawn from the impinging EMI fields. Hence the EMI is weakened as it penetrates the shield.

More formally, the I^2R losses are referred to as absorption loss, and the re-radiation is called reflection loss. As it turns out, absorption loss is the primary shielding mechanism for H-fields, and reflection loss is the primary shielding mechanism for E-fields. Reflection loss, being a surface phenomenon, is pretty much independent of the thickness of the shielding material. Both loss mechanisms, however, are dependent on the frequency (ω) of the impinging EMI field, and on the permeability (μ) and conductivity (σ) of the shielding material. These loss mechanisms vary approximately as follows:

$$\text{reflection loss to an E-field (in dB)} \sim \log \frac{\sigma}{\omega\mu}$$

$$\text{absorption loss to an H-field (in dB)} \sim t\sqrt{\omega\sigma\mu}$$

where t is the thickness of the shielding material.

The first expression indicates that E-field shielding is more effective if the shield material is highly conductive, and less effective if the shield is ferromagnetic, and that low-frequency fields are easier to block than high-frequency fields. This is shown in Figure 9.

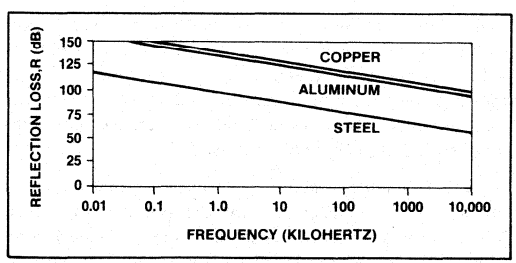


Figure 9. E-Field Shielding

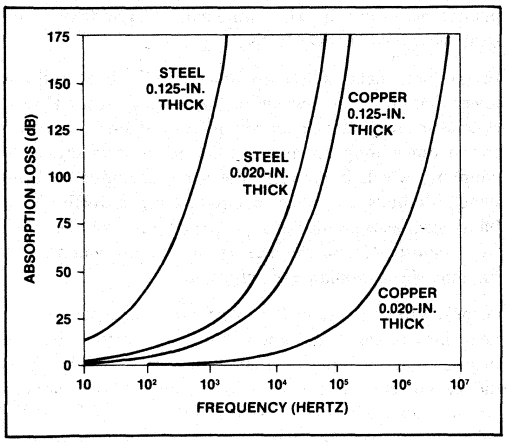


Figure 10. H-Field Shielding

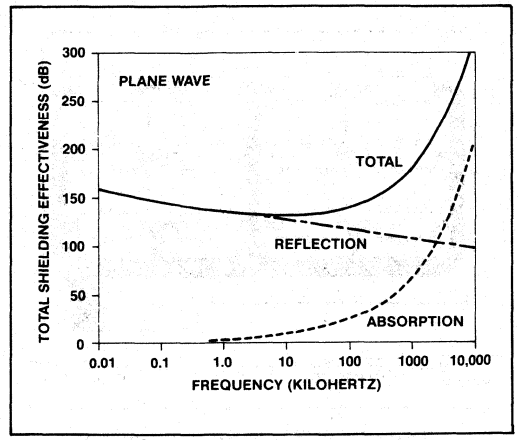


Figure 11. E- and H-Field Shielding

Copper and aluminum both have the same permeability, but copper is slightly more conductive, and so provides slightly greater reflection loss to an E-field. Steel is less effective for two reasons. First, it has a somewhat elevated permeability due to its iron content, and, second, as tends to be the case with magnetic materials, it is less conductive.

On the other hand, according to the expression for absorption loss to an H-field, H-field shielding is more effective at higher frequencies and with shield material that has both high conductivity and high permeability. In practice, however, selecting steel for its high permeability involves some compromise in conductivity. But the increase in permeability more than makes up for the decrease in conductivity, as can be seen in Figure 10. This figure also shows the effect of shield thickness.

A composite of E-field and H-field shielding is shown in Figure 11. However, this type of data is meaningful only in the far field. In the near field the EMI could be 90% H-field, in which case the reflection loss is irrelevant. It would be advisable then to beef up the absorption loss, at the expense of reflection loss, by choosing steel. A better conductor than steel might be less expensive, but quite ineffective.

A different shielding mechanism that can be taken advantage of for low frequency magnetic fields is the ability of a high permeability material such as mumetal to divert the field by presenting a very low reluctance path to the magnetic flux. Above a few kHz, however, the permeability of such materials is the same as steel.

In actual fact the selection of a shielding material turns out to be less important than the presence of seams, joints and holes in the physical structure of the enclosure. The shielding mechanisms are related to the induction of currents in the shield material, but the currents must be allowed to flow freely. If they have to detour around slots and holes, as shown in Figure 12, the shield loses much of its effectiveness.

As can be seen in Figure 12, the severity of the detour has less to do with the area of the hole than it does with the geometry of the hole. Comparing Figure 12C with 12D shows that a long narrow discontinuity such as a seam can cause more RF leakage than a line of holes with larger total area. A person who is responsible for designing or selecting rack or chassis enclosures for an EMI environment needs to be familiar with the techniques that are available for maintaining electrical continuity across seams. Information on these techniques is available in the references.

Grounds

There are two kinds of grounds: earth-ground and signal ground. The earth is not an equipotential surface, so earth ground potential varies. That and its other electrical properties are not conducive to its use as a return conductor in a circuit. However, circuits are often connected to earth ground for protection against shock hazards. The other kind of ground, signal ground, is an arbitrarily selected reference node in a circuit—the node with respect to which other node voltages in the circuit are measured.

SAFETY GROUND

The standard 3-wire single-phase AC power distribution system is represented in Figure 13. The white wire is earth-grounded at the service entrance. If a load circuit has a metal enclosure or chassis, and if the black wire develops a short to the enclosure, there will be a shock hazard to operating personnel, unless the enclosure itself is earth-grounded. If the enclosure is earth-grounded, a

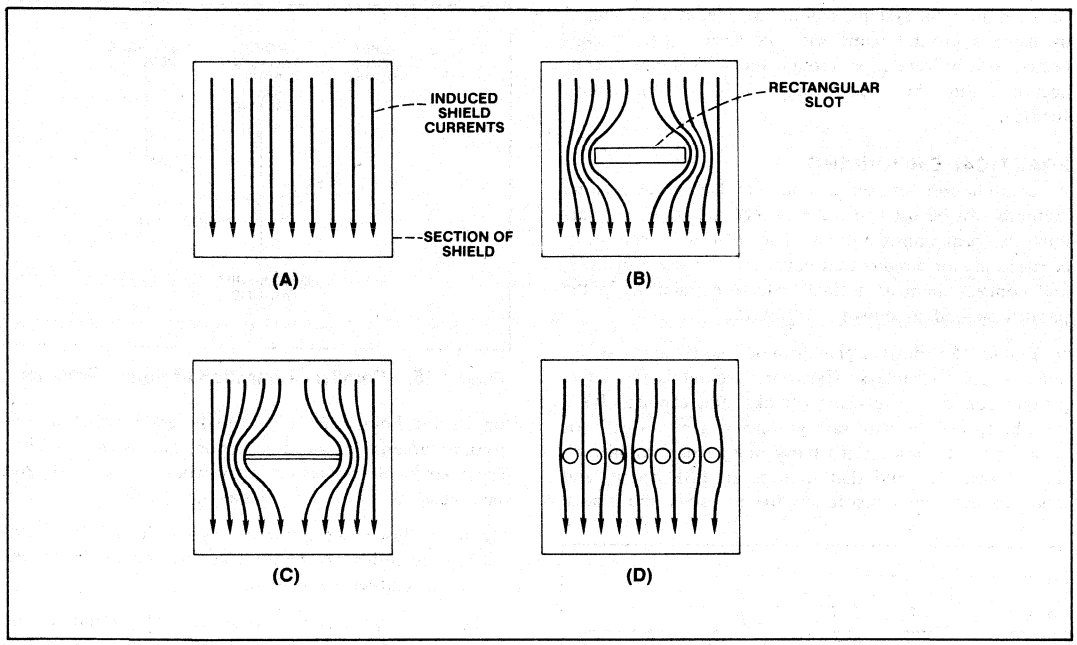


Figure 12. Effect of Shield Discontinuity on Magnetically Induced Shield Current

short results in a blown fuse rather than a "hot" enclosure. The earth-ground connection to the enclosure is called a safety ground. The advantage of the 3-wire power system is that it distributes a safety ground along with the power.

Note that the safety-ground wire carries no current, except in case of a fault, so that at least for low frequencies it's at earth-ground potential along its entire length. The white wire, on the other hand, may be several volts off ground, due to the IR drop along its length.

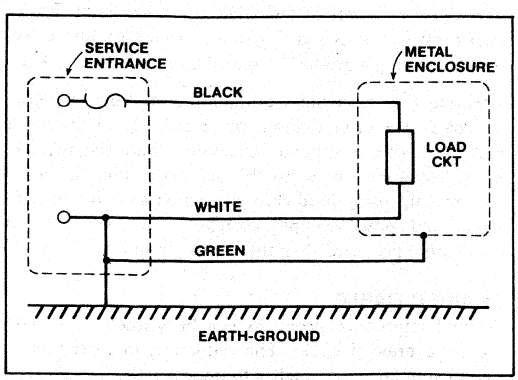


Figure 13. Single-Phase Power Distribution

SIGNAL GROUND

Signal ground is a single point in a circuit that is designated to be the reference node for the circuit. Commonly, wires that connect to this single point are also referred to as "signal ground." In some circles "power supply common" or PSC is the preferred terminology for these conductors. In any case, the manner in which these wires connect to the actual reference point is the basis of distinction among three kinds of signal-ground wiring methods: series, parallel, and multipoint. These methods are shown in Figure 14.

The series connection is pretty common because it's simple and economical. It's the noisiest of the three, however, due to common ground impedance coupling between the circuits. When several circuits share a ground wire, currents from one circuit, flowing through the finite impedance of the common ground line, cause variations in the ground potential of the other circuits. Given that the currents in a digital system tend to be spiked, and that the common impedance is mainly inductive reactance, the variations could be bad enough to cause bit errors in high current or particularly noisy situations.

The parallel connection eliminates common ground impedance problems, but uses a lot of wire. Other disadvantages are that the impedance of the individual ground lines can be very high, and the ground lines themselves can become sources of EMI.

In the multipoint system, ground impedance is minimized by using a ground plane with the various circuits connected to it by very short ground leads. This type of connection would be used mainly in RF circuits above 10MHz.

PRACTICAL GROUNDING

A combination of series and parallel ground-wiring methods can be used to trade off economic and the various electrical considerations. The idea is to run series connections for circuits that have similar noise properties, and connect them at a single reference point, as in the parallel method, as shown in Figure 15.

In Figure 15, "noisy signal ground" connects to things like motors and relays. Hardware ground is the safety ground connection to chassis, racks, and cabinets. It's a mistake to use the hardware ground as a return path for signal currents because it's fairly noisy (for example, it's the hardware ground that receives an ESD spark) and tends to have high resistance due to joints and seams.

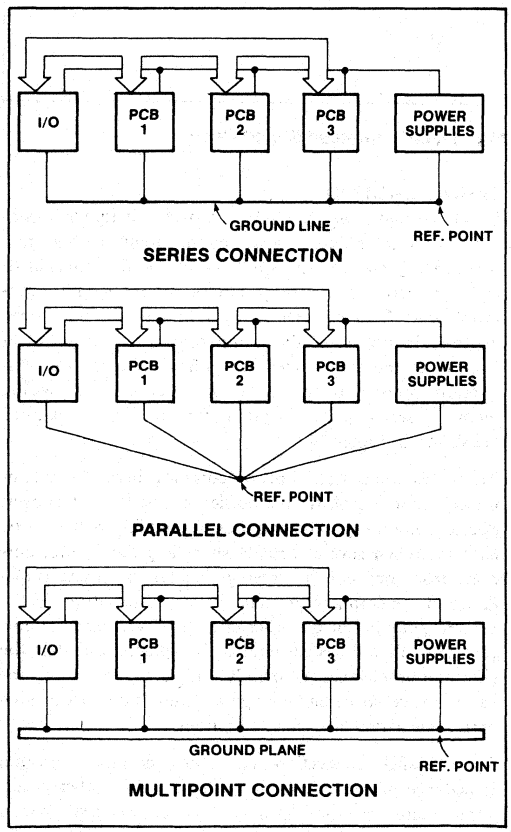


Figure 14. Three Ways to Wire the Grounds

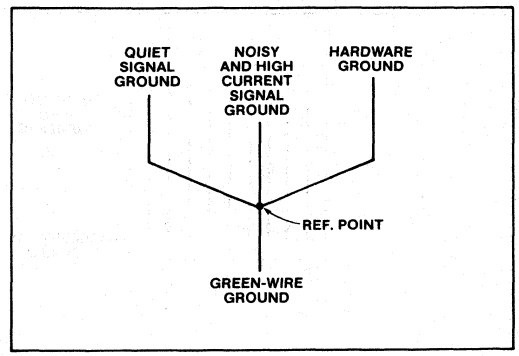


Figure 15. Parallel Connection of Series Grounds

Screws and bolts don't always make good electrical connections because of galvanic action, corrosion, and dirt. These kinds of connections may work well at first, and then cause mysterious maladies as the system ages.

Figure 16 illustrates a grounding system for a 9-track digital tape recorder, showing an application of the series/parallel ground-wiring method.

Figure 17 shows a similar separation of grounds at the PCB level. Currents in multiplexed LED displays tend to put a lot of noise on the ground and supply lines because of the constant switching and changing involved in the scanning process. The segment driver ground is relatively quiet, since it doesn't conduct the LED currents. The digit driver ground is noisier, and should be provided with a separate path to the PCB ground terminal, even if the PCB ground layout is gridded. The LED feed and return current paths should be laid out on opposite sides of the board like parallel flat conductors.

Figure 18 shows right and wrong ways to make ground connections in racks. Note that the safety ground connections from panel to rack are made through ground straps, not panel screws. Rack 1 correctly connects signal ground to rack ground only at the single reference point. Rack 2 incorrectly connects signal ground to rack ground at two points, creating a ground loop around points 1, 2, 3, 4, 1.

Breaking the "electronics ground" connection to point 1 eliminates the ground loop, but leaves signal ground in rack 2 sharing a ground impedance with the relatively noisy hardware ground to the reference point; in fact, it may end up using hardware ground as a return path for signal and power supply currents. This will probably cause more problems than the ground loop.

BRAIDED CABLE

Ground impedance problems can be virtually eliminated by using braided cable. The reduction in impedance is due to skin effect: At higher frequencies the current tends to flow along the surface of a conductor rather than uni-

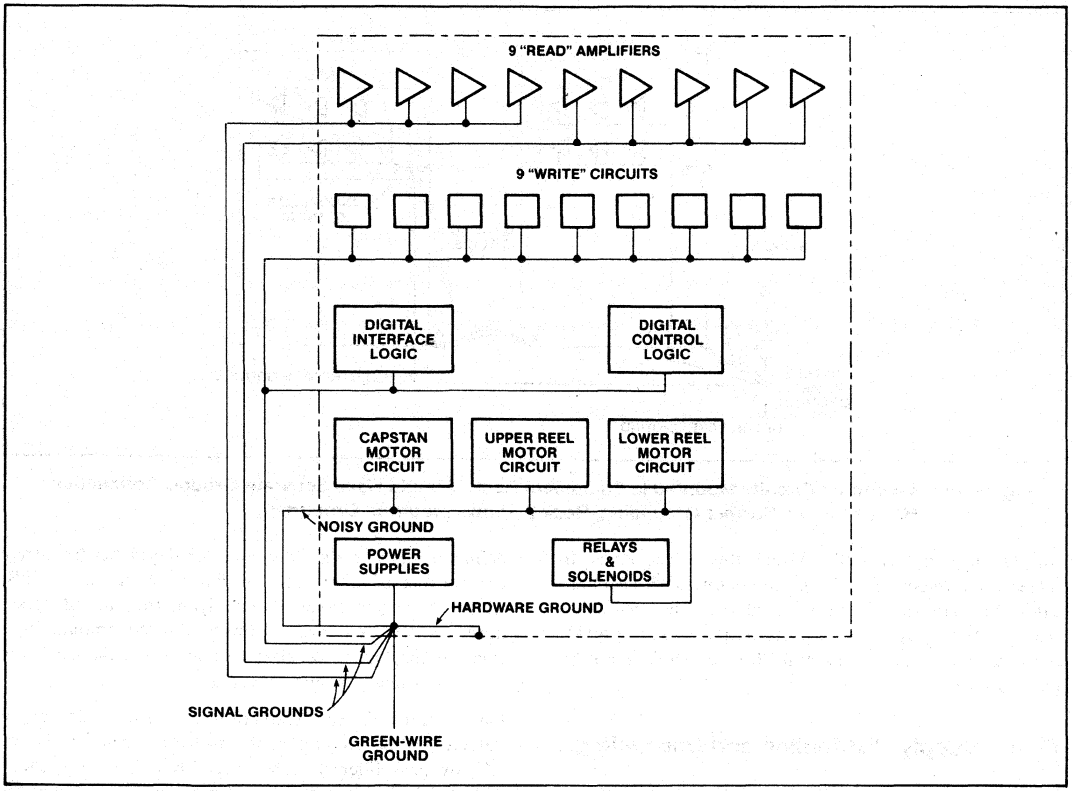


Figure 16. Ground System in a 9-Track Digital Recorder

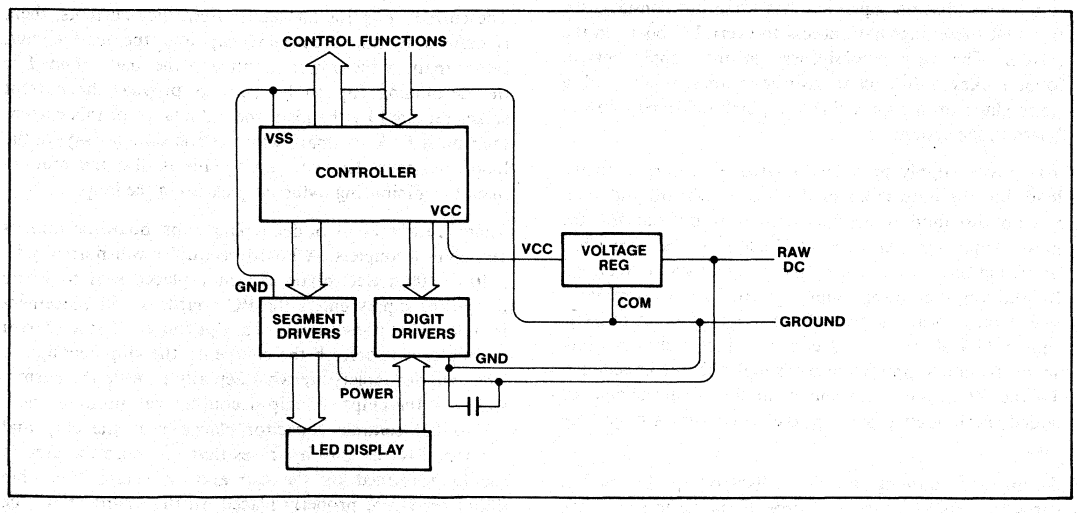


Figure 17. Separate Ground for Multiplexed LED Display

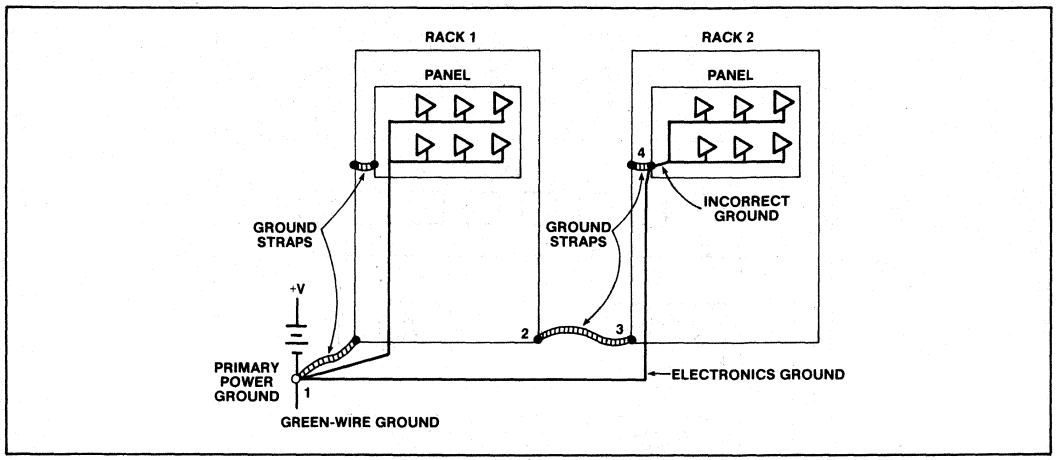


Figure 18. Electronic Circuits Mounted in Equipment Racks Should Have Separate Ground Connections. Rack 1 Shows Correct Grounding, Rack 2 Shows Incorrect Grounding

formly through its bulk. While this effect tends to increase the impedance of a given conductor, it also indicates the way to minimize impedance, and that is to manipulate the shape of the cross-section so as to provide more surface area. For its bulk, braided cable is almost pure surface.

Power Supply Distribution and Decoupling

The main consideration for power supply distribution lines is, as for signal lines, to minimize the areas of the current loops. But the power supply lines take on an importance that no signal line has when one considers the fact that these lines have access to every PC board in the system. The very extensiveness of the supply current loops makes it difficult to keep loop areas small. And, a noise glitch on a supply line is a glitch delivered to every board in the system.

The power supply provides low-frequency current to the load, but the inductance of the board-to-board and chip-to-chip distribution network makes it difficult for the power supply to maintain VCC specs on the chip while providing the current spikes that a digital system requires. In addition, the power supply current loop is a very large one, which means there will be a lot of noise pick-up. Figure 19A shows a load circuit trying to draw current spikes from a supply voltage through the line impedance. To the VCC waveform shown in that figure should be added the inductive pick-up associated with a large loop area.

Adding a decoupling capacitor solves two problems: The capacitor acts as a nearby source of charge to supply the current spikes through a smaller line impedance, and it defines a much smaller loop area for the higher frequency components of EMI. This is illustrated in Figure 19B, which shows the capacitor supplying the current spike, during which VCC drops from 5V by the amount indicated in the figure. Between current spikes the capacitor recovers through the line impedance.

One should resist the temptation to add a resistor or an inductor to the decoupler so as to form a genuine RC or LC low-pass filter because that slows down the speed with which the decoupler cap can be refreshed. Good filtering and good decoupling are not necessarily the same thing.

The current loop for the higher frequency currents, then, is defined by the decoupling cap and the load circuit, rather than by the power supply and the load circuit. For the decoupling cap to be able to provide the current spikes required by the load, the inductance of this current loop must be kept small, which is the same as saying the loop area must be kept small. This is also the requirement for minimizing inductive pick-up in the loop.

There are two kinds of decoupling caps: board decouplers and chip decouplers. A board decoupler will normally be a 10 to 100μf electrolytic capacitor placed near to where the power supply enters the PC board, but its placement is relatively non-critical. The purpose of the board decoupler is to refresh the charge on the chip decouplers. The chip decouplers are what actually provide the current spikes to the chips. A chip decoupler will normally be a 0.1 to 1μf ceramic capacitor placed near the chip and connected to the chip by traces that minimize the area of the loop formed by the cap and the chip. If a chip decoupler is not properly placed on the board, it will be ineffective as a decoupler and will serve only to increase

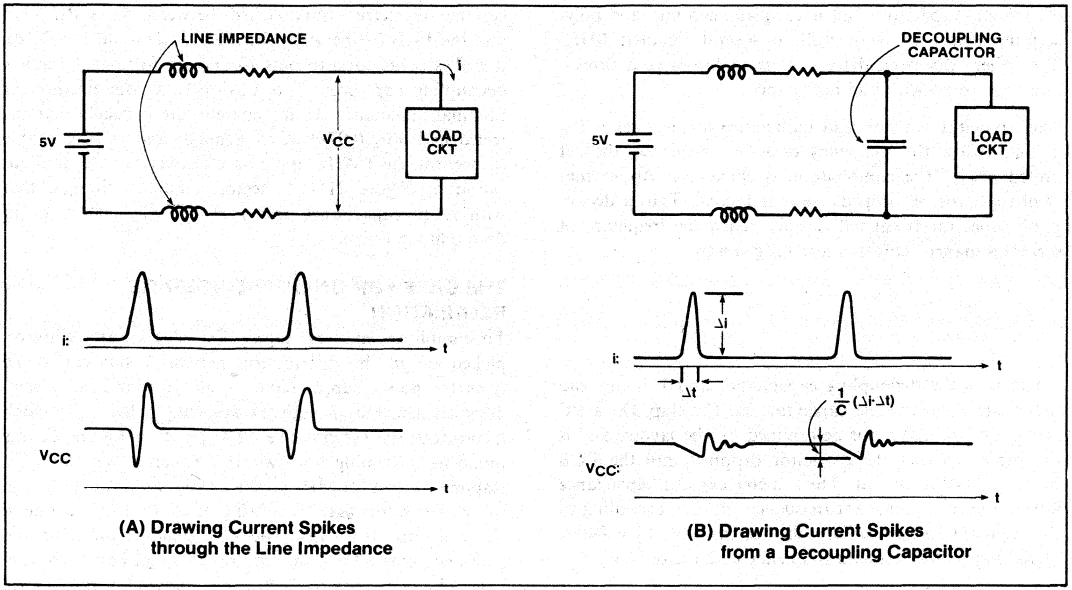


Figure 19. What a Decoupling Capacitor Does

the cost of the board. Good and bad placement of decoupling capacitors are illustrated in Figure 20.

Power distribution traces on the PC board need to be laid out so as to obtain minimal area (minimal inductance) in the loops formed by each chip and its decoupler, and by the chip decouplers and the board decoupler. One way to accomplish this goal is to use a power plane. A power plane is the same as a ground plane, but at VCC potential. More economically, a power grid similar to the ground grid previously discussed (Figure 8) can be used. Actually, if the chip decoupling loops are small, other aspects of the power layout are less critical. In other words, power planes and power gridding aren't needed, but power traces *should* be laid in the closest possible proximity to ground traces, preferably so that each power trace is on the direct opposite side of the board from a ground trace.

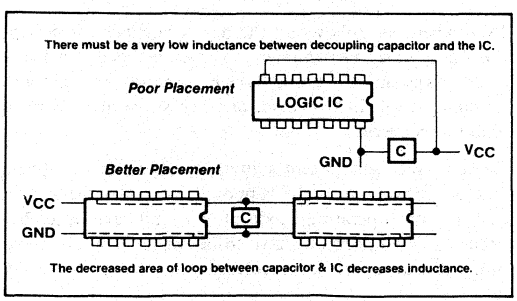


Figure 20. Placement of Decoupling Capacitors

Special-purpose power supply distribution buses which mount on the PCB are available. The buses use a parallel flat conductor configuration, one conductor being a VCC line and the other a ground line. Used in conjunction with a gridded ground layout, they not only provide a low-inductance distribution system, but can themselves form part of the ground grid, thus facilitating the PCB layout. The buses are available with and without enhanced bus capacitance, under the names Mini/Bus® and Q/PAC® from Rogers Corp. (5750 E. McKellips, Mesa, AZ 85205).

SELECTING THE VALUE OF THE DECOUPLING CAP

The effectiveness of the decoupling cap has a lot to do with the way the power and ground traces connect this capacitor to the chip. In fact, the area formed by this loop is more important than the value of the capacitance. Then, given that the area of this loop is indeed minimal, it can generally be said that the larger the value of the decoupling cap, the more effective it is, if the cap has a mica, ceramic, glass, or polystyrene dielectric.

It's often said, and not altogether accurately, that the chip decoupler shouldn't have too large a value. There are two reasons for this statement. One is that some capacitors, because of the nature of their dielectrics, tend to become inductive or lossy at higher frequencies. This is true of

electrolytic capacitors, but mica, glass, ceramic, and polystyrene dielectrics work well to several hundred MHz. The other reason cited for not using too large a capacitance has to do with lead inductance.

The capacitor with its lead inductance forms a series LC circuit. Below the frequency of series resonance, the net impedance of the combination is capacitive. Above that frequency, the net impedance is inductive. Thus a decoupling capacitor is capacitive only below the frequency of series resonance. This frequency is given by

$$f_0 = \frac{1}{2\pi\sqrt{LC}}$$

where C is the decoupling capacitance and L is the lead inductance between the capacitor and the chip. On a PC board this inductance is determined by the layout, and is the same whether the capacitor dropped into the PCB holes is 0.001μf or 1μf. Thus, increasing the capacitance lowers the series resonant frequency. In fact, according to the resonant frequency formula, increasing C by a factor of 100 lowers the resonant frequency by a factor of 10.

Figures quoted on the series resonant frequency of a 0.01μf capacitor run from 10 to 15MHz, depending on the lead length. If these numbers were accurate, a 1μf capacitor in the same position on the board would have a resonant frequency of 1.0 to 1.5MHz, and as a decoupler would do more harm than good. However, the numbers are based on a presumed inductance of a given length of wire (the lead length). It should be noted that a "length of wire" has no inductance at all, strictly speaking. Only a complete current loop has inductance, and the inductance depends on the geometry of the loop. Figures quoted on the inductance of a length of wire are based on a presumably "very large" loop area, such that the magnetic field produced by the return current has no cancellation effect on the field produced by the current in the given length of wire. Such a loop geometry is not and should not be the case with the decoupling loop.

Figure 21 shows VCC waveforms, measured between pins 40 and 20 (VCC and VSS) of an 8751 CPU, for several conditions of decoupling on a PC board that has a decoupling loop area slightly larger than necessary. These photographs show the effects of increasing the decoupling capacitance and decreasing the area of the decoupling loop. The indications are that a 1μf capacitor is better than a 0.1μf capacitor, which in turn is better than nothing, and that the board should have been laid out with more attention paid to the area of the decoupling loop.

Figure 21E was obtained using a special-purpose experimental capacitor designed by Rogers Corp. (Q-Pac Division, Mesa, AZ) for use as a decoupler. It consists of two parallel plates, the length of a 40-pin DIP, separated by a ceramic dielectric. Sandwiched between the CPU chip and the PCB (or between the CPU socket and the PCB), it makes connection to pins 40 and 20, forming a leadless decoupling capacitor. It is obviously a configuration of minimal inductance. Unfortunately, the particular sample tested had only 0.07μf of capacitance and so was unable to prevent the 1MHz ripple as effectively as the configuration of Figure 21D. It seems apparent, though, that with more capacitance this part will alleviate a lot of decoupling problems.

THE CASE FOR ON-BOARD VOLTAGE REGULATION

To complicate matters, supply line glitches aren't always picked up in the distribution networks, but can come from the power supply circuit itself. In that case, a well-designed distribution network faithfully delivers the glitch throughout the system. The VCC glitch in Figure 22 was found to be coming from within a bench power supply in response to the EMP produced by an induction coil spark generator that was being used at Intel during a study of noise sensitivity. The VCC glitch is about 400mV high and some 20μsec in duration. Normal board decoupling techniques were ineffective in removing it, but adding an on-board voltage regulator chip did the job.

Thus, a good case can be made in favor of using a voltage regulator chip on each PCB, instead of doing all the voltage regulation at the supply circuit. This eases requirements on the heat-sinking at the supply circuit, and alleviates much of the distribution and board decoupling headaches. However, it also brings in the possibility that different boards would be operating at slightly different VCC levels due to tolerance in the regulator chips; this then leads to slightly different logic levels from board to board. The implications of that may vary from nothing to latch-up, depending on what kinds of chips are on the boards, and how they react to an input "high" that is perhaps 0.4V higher than local VCC.

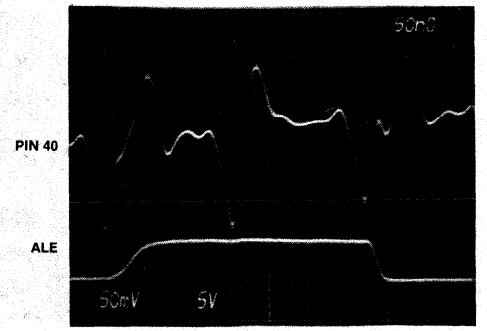
Recovering Gracefully from a Software Upset

Even when one follows all the best guidelines for designing for a noisy environment, it's always possible for a noise transient to occur which exceeds the circuit's immunity level. In that case, one can strive at least for a graceful recovery.

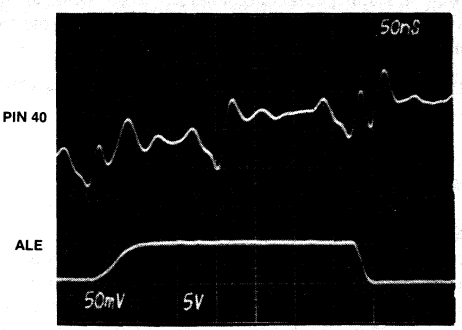
Graceful recovery schemes involve additional hardware and/or software which is supposed to return the system to a normal operating mode after a software upset has occurred. Two decisions have to be made: How to recognize when an upset has occurred, and what to do about it.

If the designer knows what kinds and combinations of

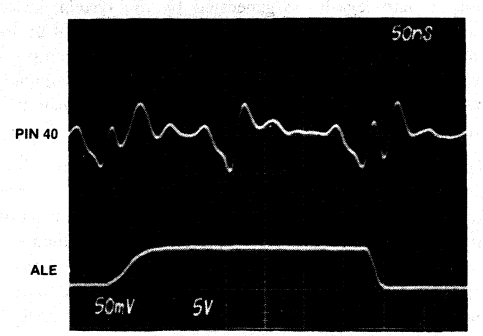
(A) No Decoupling Cap



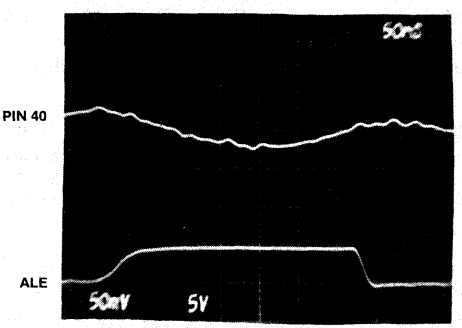
(B) 0.1μf Decoupler in Place on the PCB



(C) 0.1μf Decoupler Stretched Directly from Pin 40 to Pin 20, under the Socket (The difference between this and 21B is due only to the change in loop geometry. Also shown is the upward slope of a ripple in V_{CC}. The ripple frequency is 1MHz, the same as ALE.)



(D) 1μf Decoupler Stretched Directly from Pin 40 to Pin 20, under the Socket. (This prevents the 1MHz ripple, but there's no reduction in higher frequency components. Further increases in capacitance effected no further improvement.)



(E) Special-Purpose Decoupling Cap under Development by Rogers Corp. (Further discussion in text.)

Figure 21. Noise on V_{CC} Line

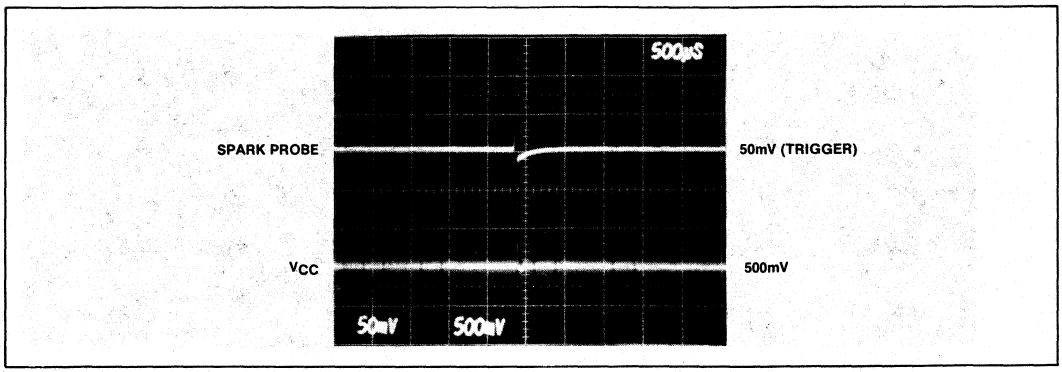


Figure 22. EMP-Induced Glitch

outputs can legally be generated by the system, he can use gates to recognize and flag the occurrence of an illegal state of affairs. The flag can then trigger a jump to a recovery routine which then may check or re-initialize data, perhaps output an error message, or generate a simple reset.

The most reliable scheme is to use a so-called watchdog circuit. Here the CPU is programmed to generate a periodic signal as long as the system is executing instructions in an expected manner. The periodic signal is then used to hold off a circuit that will trigger a jump to a recovery routine. The periodic signal needs to be AC-coupled to the trigger circuit so that a "stuck-at" fault won't continue to hold off the trigger. Then, if the processor locks up someplace, the periodic signal is lost and the watchdog triggers a reset.

In practice, it may be convenient to drive the watchdog circuit with a signal which is being generated anyway by the system. One needs to be careful, however, that an upset does in fact discontinue that signal. Specifically, for example, one could use one of the digit drive signals going to a multiplexed display. But display scanning is often handled in response to a timer-interrupt, which may continue operating even though the main program is in a failure mode. Even so, with a little extra software, the signal can be used to control the watchdog (see reference 8 on this).

Simpler schemes can work well for simpler systems. For example, if a CPU isn't doing anything but scanning and decoding a keyboard, there's little to lose and much to gain by simply resetting it periodically with an astable multivibrator. It only takes about 13μsec (at 6MHz) to reset an 8048 if the clock oscillator is already running.

A zero-cost measure is simply to fill all unused program memory with NOPs and JMPs to a recovery routine. The effectiveness of this method is increased by writing the program in segments that are separated by NOPs and JMPs. It's still possible, of course, to get hung up in a data table or something. But you get a lot of protection, for the cost.

Further discussion of graceful recovery schemes can be found in reference 13.

Special Problem Areas

ESD

MOS chips have some built-in protection against a static charge build-up on the pins, as would occur during normal handling, but there's no protection against the kinds of current levels and rise times that occur in a genuine electrostatic spark. These kinds of discharges can blow a crater in the silicon.

It must be recognized that connecting CPU pins unprotected to a keyboard or to anything else that is subject to electrostatic discharges makes an extremely fragile configuration. Buffering them is the very least one can do. But buffering doesn't completely solve the problem, because then the buffer chips will sustain the damage (even TTL); therefore, one might consider mounting the buffer chips in sockets for ease of replacement.

Transient suppressors, such as the TranZorbs® made by General Semiconductor Industries (Tempe, AZ), may in the long run provide the cheapest protection if their "zero inductance" structure is used. The structure and circuit application are shown in Figure 23.

The suppressor element is a pn junction that operates like a Zener diode. Back-to-back units are available for AC operation. The element is more or less an open circuit at normal system voltage (the standoff voltage rating for the device), and conducts like a Zener diode at the clamping voltage.

The lead inductance in the conventional transient suppressor package makes the conventional package essen-

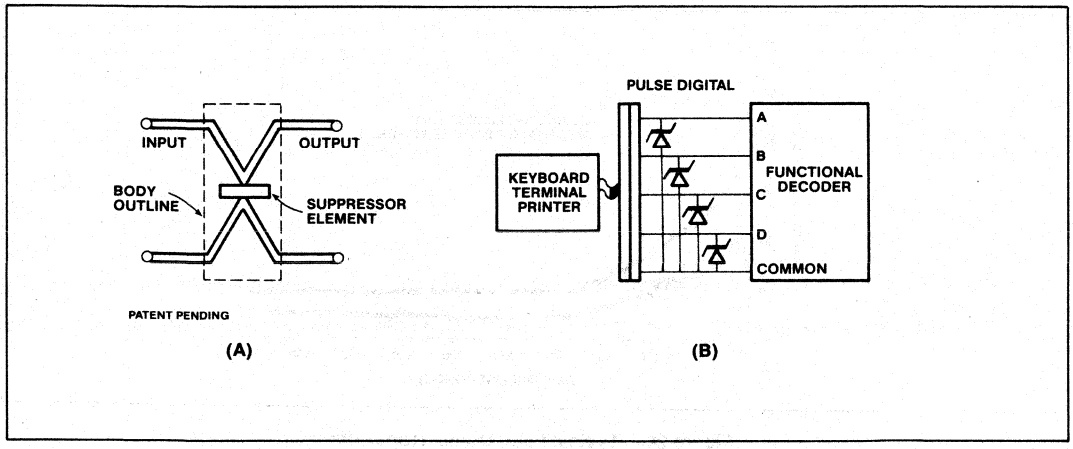


Figure 23. "Zero-inductance" Structure and Use in Circuit

tially useless for protection against ESD pulses, owing to the fast rise of these pulses. The "zero inductance" units are available singly in a 4-pin DIP, and in arrays of four to a 16-pin DIP for PCB level protection. In that application they should be mounted in close proximity to the chips they protect.

In addition, metal enclosures or frames or parts that can receive an ESD spark should be connected by braided cable to the green-wire ground. Because of the ground impedance, ESD current shouldn't be allowed to flow through any signal ground, even if the chips are protected by transient suppressors. A 35kV ESD spark can always spare a few hundred volts to drive a fast current pulse down a signal ground line if it can't find a braided cable to follow. Think how delighted your 8048 will be to find its VSS pin about 250V higher than VCC for a few 10s of nanoseconds.

THE AUTOMOTIVE ENVIRONMENT

The automobile presents an extremely hostile environment for electronic systems. There are several parts to it:

1. Temperature extremes from -40°C to +125°C (under the hood) or +85°C (in the passenger compartment)
2. Electromagnetic pulses from the ignition system
3. Supply line transients that will knock your socks off

One needs to take a long, careful look at the temperature extremes. The allowable storage temperature range for most Intel MOS chips is -65°C to +150°C, although some chips have a maximum storage temperature rating of +125°C. In operation (or "under bias," as the data sheets say) the allowable ambient temperature range depends on the product grade, as follows:

Grade	Ambient Temperature min.	Ambient Temperature max.
Commercial	0	70
Industrial	-40	+85
Automotive	-40	+110
Military	-55	+125

The different product grades are actually the same chip, but tested according to different standards. Thus, a given commercial-grade chip might actually pass military temperature requirements, but not have been tested for it. (Of course, there are other differences in grading requirements having to do with packaging, burn-in, traceability, etc.)

In any case, it's apparent that commercial-grade chips can't be used safely in automotive applications, not even in the passenger compartment. Industrial-grade chips can be used in the passenger compartment, and automotive or military chips are required in under-the-hood applications.

Ignition noise, CB radios, and that sort of thing are probably the least of your worries. In a poorly designed system, or in one that has not been adequately tested for the automotive environment, this type of EMI might cause a few software upsets, but not destroy chips.

The major problem, and the one that seems to come as the biggest surprise to most people, is the line transients. Regrettably, the 12V battery is not actually the source of power when the car is running. The charging system is, and it's not very clean. The only time the battery is the real source of power is when the car is first being started, and in that condition the battery terminals may be delivering about 5 or 6V. Below is a brief description of the major idiosyncracies of the "12V" automotive power line.

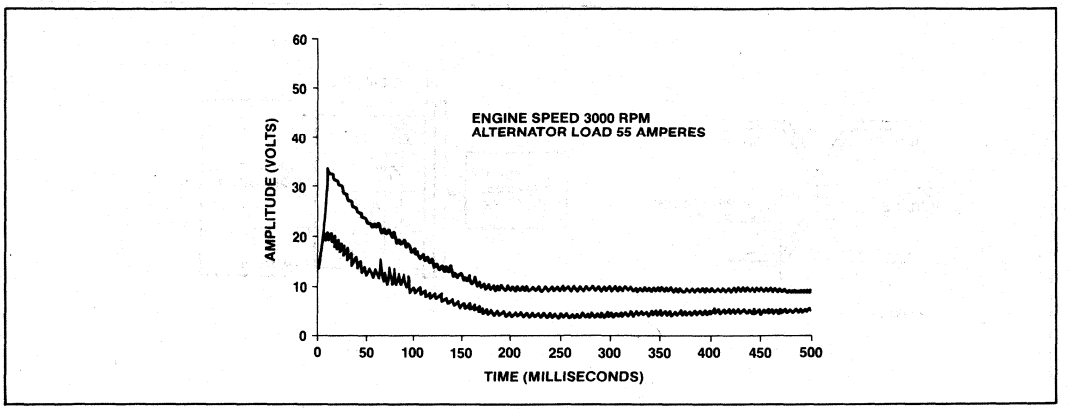


Figure 24. Typical Load Dump Transients

- An abrupt reduction in the alternator load causes a positive voltage transient called "load dump." In a load dump transient the line voltage rises to 20 or 30V in a few msec, then decays exponentially with a time constant of about 100msec, as shown in Figure 24. Much higher peak voltages and longer decay times have also been reported. The worst case load dump is caused by disconnecting a low battery from the alternator circuit while the alternator is running. Normally this would happen intermittently when the battery terminal connections are defective.
- When the ignition is turned off, as the field excitation decays, the line voltage can go to between -40 and -100V for 100 msec or more.
- Miscellaneous solenoid switching transients, such as the one shown in Figure 25, can drive the line to + or -200 to 400V for several μsec.
- Mutual coupling between unshielded wires in long harnesses can induce 100 and 200V transients in unprotected circuits.

What all this adds up to is that people in the business of building systems for automotive applications need a comprehensive testing program. An SAE guideline which describes the automotive environment is available to designers: SAE J1211, "Recommended Environmental Practices for Electronic Equipment Design," *1980 SAE Handbook,* Part 1, pp. 22.80–22.96.

Some suggestions for protecting circuitry are shown in Figure 26. A transient suppressor is placed in front of the regulator chip to protect it. Since the rise times in these transients are not like those in ESD pulses, lead inductance is less critical and conventional devices can be used. The regulator itself is pretty much of a necessity, since a load dump transient is simply not going to be removed

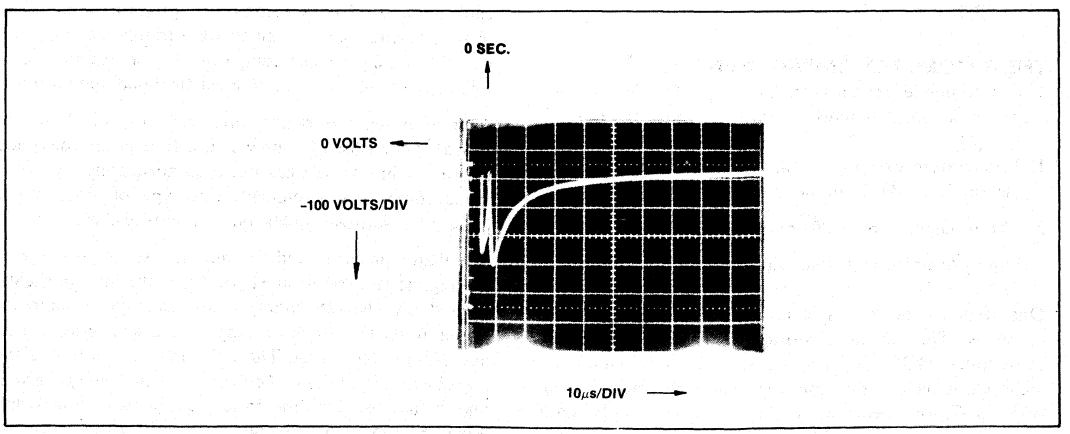


Figure 25. Transient Created by De-energizing an Air Conditioning Clutch Solenoid

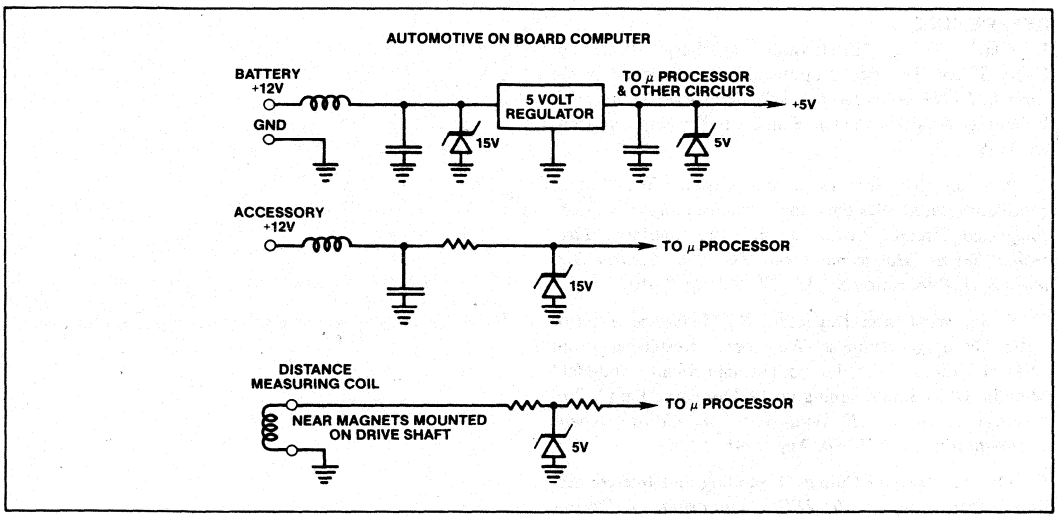


Figure 26. Use of Transient Suppressors in Automotive Applications

by any conventional LC or RC filter.

Special I/O interfacing is also required, because of the need for high tolerance to voltage transients, input noise, input/output isolation, etc. In addition, switches that are being monitored or driven by these buffers are usually referenced to chassis ground instead of signal ground, and in a car there can be many volts difference between the two. I/O interfacing is discussed in reference 2.

Parting Thoughts

The main sources of information for this Application Note were the references by Ott and by White. Reference 5 is probably the finest treatment currently available on the subject. The other references provided specific information as cited in the text.

Courses and seminars on the subject of electromagnetic interference are given regularly throughout the year. Information on these can be obtained from:

IEEE Electromagnetic Compatibility Society
EMC Education Committee
345 East 47th Street
New York, NY 10017
Phone: (212) 752-6800

Don White Consultants, Inc.
International Training Centre
P.O. Box D
Gainesville, VA 22065
Phone: (703) 347-0030

The EMC Education committee has available a video tape: "Introduction to EMC — A Video Training Tape," by Henry Ott. Don White Consultants offers a series of training courses on many different aspects of electromagnetic compatibility. Most organizations that sponsor EMC courses also offer in-plant presentations.

REFERENCES

1. Clark, O.M., "Electrostatic Discharge Protection Using Silicon Transient Suppressors," *Proceedings of the Electrical Overstress/Electrostatic Discharge Symposium.* Reliability Analysis Center, Rome Air Development Center, 1979.

2. Kearney, M; Shreve, J.; and Vincent, W., "Microprocessor Based Systems in the Automobile: Custom Integrated Circuits Provide an Effective Interface," *Electronic Engine Management and Driveline Control Systems,* SAE Publication SP-481, 810160, pp. 93-102.

3. King, W.M. and Reynolds, D., "Personnel Electrostatic Discharge: Impulse Waveforms Resultsing From ESD of Humans Directly and Through Small Hand-Held Metallic Objects Intervening in the Discharge Path," *Proceedings of the IEEE Symposium on Electromagnetic Compatibility,* pp. 577-590, Aug. 1981.

4. Ott, H., "Digital Circuit Grounding and Interconnection," *Proceedings of the IEEE Symposium on Electromagnetic Compatibility,* pp. 292-297, Aug. 1981.

5. Ott, H., *Noise Reduction Techniques in Electronic Systems.* New York: Wiley, 1976.

6. *1981 Interference Technology Engineers' Master (ITEM) Directory and Design Guide.* R. and B. Enterprises, P.O. Box 328, Plymouth Meeting, PA 19426.

7. SAE J1211, "Recommended Environmental Practices for Electronic Equipment Design," *1980 SAE Handbook,* Part 1, pp. 22.80–22.96.

8. Smith, L., "A Watchdog Circuit for Microcomputer Based Systems," *Digital Design,* pp. 78, 79, Nov. 1979.

9. *TranZorb Quick Reference Guide.* General Seminconductor Industries, P.O. Box 3078, Tempe, AZ 85281.

10. Tucker, T.J., "Spark Initiation Requirements of a Secondary Explosive," *Annals of the New York Academy of Sciences,* Vol 152, Article I, pp. 643-653, 1968.

11. White, D., *Electromagnetic Interference and Compatibility, Vol. 3: EMI Control Methods and Techniques.* Don White Consultants, 1973.

12. White, D., *EMI Control in the Design of Printed Circuit boards and Backplanes.* Don White Consultants, 1981.

13. Yarkoni, B. and Wharton, J., "Designing Reliable Software for Automotive Applications," *SAE Transactions,* 790237, July 1979.

Putting a new wrinkle in a CMOS process has led to a pair of high-caliber microcontrollers that limit power dissipation but not speed.

Microcontrollers go CMOS without slowing down

Long considered a low-speed, complex technology compared with NMOS, CMOS is on the rise as it prepares to compete successfully with NMOS for new, high-speed microprocessor applications. Using its HMOS-II technology, Intel has developed high-performance CMOS (C-HMOS) logic and applied it to two standard NMOS microcontrollers, the 8049 and 8051.

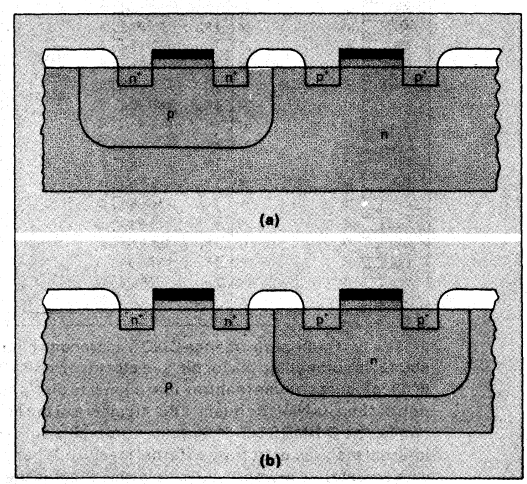
The two new members of the MCS-48 and MCS-51 families—the 80C49 and 80C51, respectively—are out to erase the low-performance image of previous CMOS devices while minimizing price differences and chip sizes compared with their n-channel counterparts. C-HMOS technology borrows many of the design and performance characteristics of high-performance MOS (HMOS). However, developing C-HMOS called for the creation of an n-well CMOS process (Fig. 1). That had not been done previously because CMOS technology grew naturally out of the older p-channel MOS technology.

N-well CMOS offers two important processing advantages: First, the substrate remains the same as in HMOS. Thus n-channel transistors in C-HMOS require no new process characterization. Second, the only new processing necessary are the n well and a p-channel transistor.

Because they retain the design rules of HMOS-II, C-HMOS devices realize the high speed and superior packing density of that process. In addition, C-HMOS, like CMOS, dissipates less power than any other semiconductor technology. One of the best uses of the combined characteristics is in a microcontroller that must operate at low input power levels, and the number of such applications is growing. They include battery-powered equipment and systems that must keep working when normal ac power is removed. In fact, this class of applications would not exist if not for CMOS.

Microcontrollers are an ideal vehicle for the first use of C-HMOS technology for several reasons. For one, since they are generally stand-alone devices, designers can construct a complete one-chip CMOS microcontroller system. For another, they are widely used in battery-powered systems, in which CMOS devices are mandatory. Also, since they contain RAM, ROM, and random logic on a single chip, designers gain experience with the major forms of CMOS hardware contained on a single device. As for the 80C49 and 80C51 in particular, designers already familiar with the 8049 and 8051 can gain the benefits of CMOS with minor software modifications.

In developing the 80C49 and 80C51, the architectural differences between the NMOS and C-HMOS



1. In the standard CMOS process, a p well resides in an n-type substrate (a). C-HMOS technology reverses this process, placing an n well in a p-type substrate (b).

John Katausky, Technical Marketing Manager
George Leach, Design Engineering Manager
Intel Corp., Microcontroller Operation
5000 W. Williams Field Rd., Chandler, AZ 85224

Reprinted from ELECTRONIC DESIGN - February 18, 1982

versions were minimized. As a result, designers need not relearn instruction sets, development tools, and existing hardware designs. Existing software can even be patched with idle instructions to reduce significantly the part's power consumption in a given application.

The 80C49 leads the way

The 80C49 is pin– and instruction-set–compatible with the industry-standard 8049 microcontroller (Fig. 2a). It contains 2 kbytes of ROM, 128 bytes of RAM, and an 8-bit timer-counter. Other features include three 8-bit I/O ports, two test inputs, and an output structure geared to easy expansion of the I/O. To cover a range of controller applications, two companion devices are in the offing. One is the 80C48 (1 kbyte of ROM, 64 bytes of RAM) and the other is the 80C50 (4 kbytes of ROM, 256 bytes of RAM). Both will use the same CPU as the 80C49 and include the same power-down and idle features.

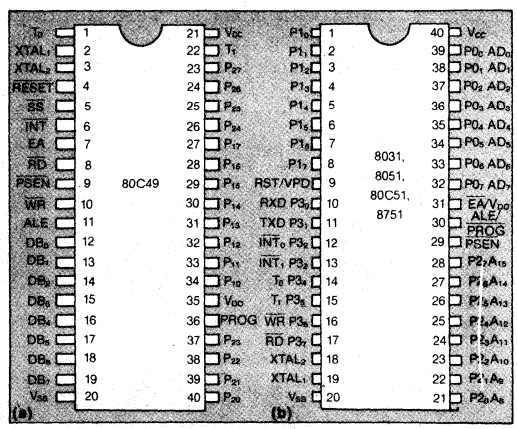
The 80C49 operates off 4.0 to 6.0 V. Between 4.5 to 5.5 V, its inputs and outputs match those of TTL circuits. When interfaced with CMOS circuitry, both inputs and outputs reach CMOS levels for the logic 1 and 0 states.

The operating frequency in CMOS devices is highly dependent on supply voltage. In the case of 80C49, above 4.5 V it can run at 11 MHz, but below that the clock rate must be reduced. For example, at the maximum supply voltage of 6 V and an operating frequency of 11 MHz, the chip draws a maximum current of 15 mA; with a 4-V power supply and a frequency of 1 MHz, however, the current drain is just 1 mA (see the table).

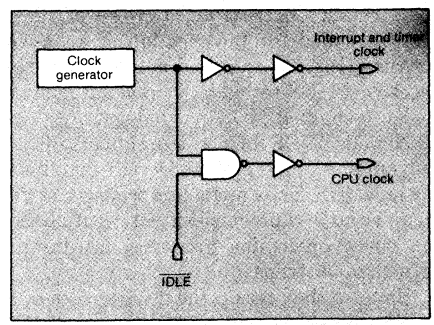
To reduce power consumption further, the 80C49 incorporates a very flexible idle mode. "Idle" is a new instruction added to the 80C49's architecture. Its function is to stop all CPU activity and allow only the timer-counter and the interrupt circuitry to remain active. In hardware, that is accomplished using two sets of clocks, one for the CPU and the other for the timer-counter and interrupt circuitry. Figure 3 shows the logical implementation of the dual clock. With this setup, the CPU is easily disabled by selectively stopping its clock.

During idle, the counter-timer clock continues to run and the entire 80C49 draws but a fraction of its normal active current. The user can monitor events or real-time functions while holding the current drain to just 500 μA at 6 V and 11 MHz.

Either the timer-counter interrupt or an external interrupt terminates the idle mode. Interrupts are then handled as though they were received under normal operating conditions. Moreover, a simple flag, implemented by the user, can indicate if the interrupt occurred during the idle condition or during normal operation. A hardware reset also terminates the idle mode.



2. The first high-performance CMOS microcontroller, the 80C49, is compatible pin for pin and electrically with the NMOS 8049 (a), but contains features that allow it to dissipate less power than its NMOS cousin. Like members of the MCS-48 family, the C-HMOS 80C51 executes programs out of its internal memory or from an external memory. When operating with an external memory, the chip's pins assume the functions shown in the outside columi (b).



3. The 80C49's low-power idle mode is implemented using dual clocks, allowing the timer-counter and interrupts to remain active while the controller draws just 500 μA operating from a 6-V power supply at an 11-MHz clock rate.

The timer-counter remains active during the idle mode to allow the idle instruction to be as useful as possible. For example, the absence of an active timer-counter forces the user to implement an external hardware interrupt to end the idle mode, a less than ideal solution in almost all applications. With an active timer-counter, the user can operate the 80C49 in a programmable duty-cycle mode, which allows the ratio of CPU active time to idle time to be determined by software.

Holding down power

Power consumption is reduced to an absolute minimum by placing the 80C49 in a hardware power-down mode (Fig. 4). During power-down, only the on-board RAM is kept active. Power-down is activated by turning off the V_{CC} supply and reducing the V_{DD} supply to its minimum value of 2 V. That cuts the current drain to only 10 μA.

For some applications, it is useful for the designer to know whether a power-up condition occurred from a cold start or from the power-down mode. That is made possible by software and is accomplished by inserting a 1- or 2-byte "key" in the RAM before entering the power-down mode. When the device resets and comes out of power-down, the key bytes are examined, enabling the CPU to determine from what condition the reset occurred.

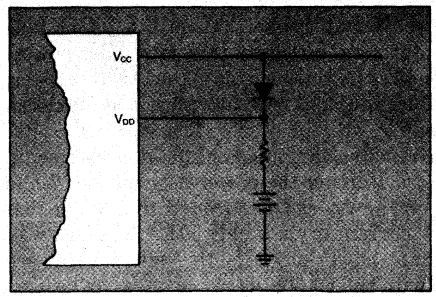
Higher performance with the 80C51

Specifically targeted at high-end 8-bit microprocessor applications requiring low power consumption, the 80C51 has all of the 8051's architectural features, including its enhanced CPU and I/O functions. The on-chip program memory size remains 4 kbytes, with a full 64-kbyte external range. The 128-byte on-chip RAM also is externally expandable—to 64 kbytes. Also, on board are two 16-bit timer-counters, a full duplex serial port, and a 1-bit Boolean processor for control functions. Figure 2b shows the 80C51's pinout for both the expanded and the port mode.

Electrically similar to the 80C49, the 80C51 has a V_{CC} range of 4 to 6 V and is TTL-compatible between 4.5 and 5.5 V. Above 4.5 V, the chip operates at clock rates of up to 15 MHz (see the table).

Like the 80C49, the 80C51 offers an idle mode for lower power dissipation. This mode is initiated by setting a bit in a new on-chip register, called the PCON register, that resides in previously unused chip area. When the bit is set, the CPU is disabled, but the timer-counter, interrupts, and serial port continue to function normally. A reset or any enabled interrupt terminates the idle mode.

If an interrupt ends the idle mode, all register data remain unchanged and the interrupt is serviced. Idle



4. Absolute minimum power is drawn during the 80C49's power-down mode. A fail-safe power-down mode circuit can be designed with just a germanium diode, a resistor, and two or three nickel-cadmium cells.

The current drain for the 80C49/80C51 (mA)

V_{CC}	At 1 MHz	At 6 MHz	At 11/12 MHz*	At 15 MHz
4 V	1/1.3	5.5/8	—/16	—/—
5 V	1.2/1.6	7/10	12.5/20	—/26
6 V	1.5/2.0	8.5/12.5	15/25	—/31

*11 MHz for the 80C49, 12 MHz for the 80C51.

is implemented in much the same way as on the 80C49, that is, by splitting the on-chip clocks into two signals, one for the CPU and the other for the active idle circuitry. Using this technique, current drain in the idle mode is about 1 mA at a supply voltage of 6 V and an operating frequency of 15 MHz.

Other bits of the PCON registers can be set as flags to determine whether an interrupt has been used for its normal function or to terminate the idle mode. These flags permit an interrupt to do double duty, instead of having to dedicate a particular interrupt specifically to this mode.

How the idle mode works

Connecting a number of slave 80C51s to a common serial link serves as an example of how to use the 80C51's idle mode. These slave processors would be controlled by a master computer or a master 80C51. Each slave processor would contain a unique code byte that would be stored in on-chip memory. To conserve power, each of the slave 80C51s could be in the idle mode. When the master transmitted a code byte, all of the slaves would receive a serial-port interrupt terminating the mode. Then each slave would compare the transmitted code byte with its internal code byte. The 80C51 whose internal code matched the transmitted code could then carry on the task it was programmed for, while the other slaves would return to the idle mode.

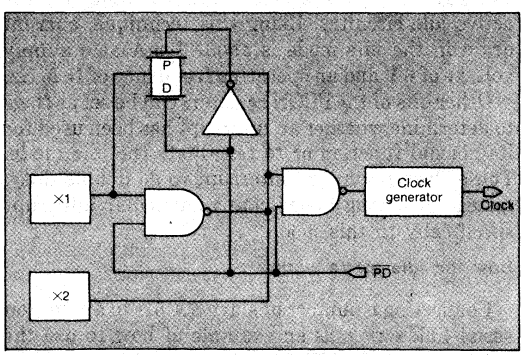
A new power-down mode is incorporated on the

80C51 (Fig. 5), but unlike that on the 80C49, it is achieved through software. As with idle, power-down is entered by setting a bit in the PCON register. That disables the clock oscillator, freezing the clocks and stopping all functions. Therefore all vital hardware-register data must be stored in the on-chip RAM before starting the power-down mode. While powered down, V_{CC} can be lowered to 2 V—at that level, total current drain is only 50 μA. The only way to terminate power-down is through a hardware reset. As with the 80C49, software flags can be used to determine the state the processor was in when an interrupt or reset occurred.

C-HMOS for other kinds of circuits

Although the first products are single-chip microcontrollers, C-HMOS technology can be used for any product fabricated in HMOS and for some made with other technologies.

In building CMOS and C-HMOS circuits, p-channel and n-channel devices are created. Thus n-type and p-type impurities are routinely doped into the substrate. It is therefore possible to fabricate npn and pnp bipolar transistors on the same substrate. When high-drive capability is needed, a designer can create an npn device. In fact, in an experimental ECL-compatible RAM memory, designers have used an npn transistor to provide the ECL output levels.



5. The 80C51's power-down mode is activated by a bit in the new on-board PCON register. Setting the register's $\overline{PD}$ bit turns off the chip's oscillator. This bit actually enables a gate that turns off the oscillator.

The ability to fabricate n-channel, p-channel, and bipolar devices on the same substrate gives designers considerable flexibility. It should, for example, be easier in the future to combine analog and digital functions on the same chip.

Experiments with memories

C-HMOS has been successfully demonstrated in laboratory memory devices. One is a 4-k static RAM that is functionally identical to the 2147. The chip exhibits the same address and data access times as its HMOS counterpart but consumes one-fifth as much power.

Another experimental 4-k static RAM is compatible with 10K series ECL memories. Here, the memory array was fabricated with six-transistor CMOS devices, and special conversion circuitry was fitted to the front and back ends. These converters change the ECL inputs to MOS inputs and the MOS outputs to ECL outputs. The resulting performance rivals that of the chip's bipolar equivalents. More importantly, the circuit may pave the way for 16-k, and larger, ECL-compatible memories. At those levels, working with bipolar technology becomes very difficult, if not impossible. Thus it may be that higher-capacity ECL-compatible memories will have to be fabricated in an MOS technology.

Lots of room to shrink

Both HMOS and C-HMOS will no doubt undergo scaling down in the future. Interestingly, C-HMOS may have some unique advantages as sizes shrink. For example, p-channel transistors are typically used as load devices. At channel lengths of 2 μm or more, these devices have, typically, one-fourth the gain of n-channel transistors, because of the dominance of carrier mobility. However, as device dimensions shrink, saturation velocity becomes more important than carrier mobility. Therefore, as lengths are scaled down below 2 μm, the gain disparity should improve to about a 2:1 ratio, and the p-channel transistors will be even better load devices.

With scaling down and 5-V operation, it is likely that hot-electron trapping will occur and cause a shift in reference levels. To reduce that effect, it will probably be necessary to lower the power-supply levels commensurately. That is no problem, because with CMOS or C-HMOS technology the reference point will naturally assume the half-voltage level between the supply rails.□

CHAPTER 10
DESIGN CONSIDERATIONS WHEN USING CHMOS

10.0 INTRODUCTION

This chapter lists considerations that must be taken into account when designing with CHMOS. As always, if good design techniques are used, quality and reliability will be built in. The topics that are discussed are:

- Power Supply Operating Range
- Power Supply Rules
- Latch-up (SCR condition)
- CHMOS Input Rules
- CHMOS I/O Port Structure
- Interfacing Between Logic Families

10.1 POWER SUPPLY CONSIDERATIONS

Intel CHMOS microcontrollers operate over a wide voltage range from 4 volts to 6 volts. Within this 5 volt 20% tolerance range, the 80C51, 80C48, 80C49, 80C50 are CMOS compatible. The CHMOS products are also pin compatible with their respective HMOS counterparts and are electrically compatible with the TTL logic when V_{CC}/V_{DD} is held within 10% of 5 volts.

The power supply, as viewed by the microcontroller, should be extremely low in inductance because of the peak currents associated with CHMOS switching characteristics. Note that these peak currents increase with frequency. Bypass capacitors of approximately 0.1 uF should be used in the proper orientation to ensure a low inductance, high peak current power source. Low inductance capacitors are available that fit under the DIP. These capacitors are also advantages for HMOS in noisy environments. (See Application Note AP125 "designing Microcontroller Systems For Electrically Noisy Environments" for detailed discussion.)

Power supply glitches must be filtered to ensure the maximum voltage rating of the device is not violated. Violation may induce an SCR effect between V_{CC}/V_{DD} and V_{SS} (latch-up).

The polarity of the power supply must never be reversed. The V_{CC} or V_{DD} pins must never be more negative than –0.5 volts with respect to V_{SS}. If this condition occurs, the input protection diode would be forward-biased and short V_{CC}/V_{DD} and V_{SS}.

When the microcontroller is powered separately from the surrounding circuitry, the microcontroller should always be powered up before any input signal is applied. When Power Down is used, the input signals must be turned off before powering down the microcontroller to ensure that the "$V_{SS} < Vin < V_{CC}$" rule is not violated otherwise latch-up may occur.

For the 80C48, 80C49, 80C50, V_{DD} and V_{CC} must track each other within 1.5 volts (except during power down) and also maintain the 5V 20% spec. This ensures that the CPU, powered by V_{CC}, and the RAM, powered by V_{DD}, have proper voltage levels to communicate.

10.2 INPUT CONSIDERATIONS

CHMOS inputs must never be allowed to float. Floating inputs may cause high internal currents in the input buffer which will increase circuit current. Power dissipation problems could result. Unused inputs should be tied to V_{CC} or V_{SS}.

The inputs to the microcontroller must be kept within the spec range of $V_{SS} < Vin < V_{CC}$ as stated earlier. If it is possible that an input will ever swing outside the supply rails, the input current must be limited to 10 mA maximum to protect from possible latch-up. Remember the absolute maximum conditions spec'd in the data sheets must never be violated.

Intel's CHMOS is not spec'd in a no clock or D.C. condition. A clock must be applied at all times when powered up. Stopping the clock as a form of power management is unacceptable except in the case of power down.

10.3 CHMOS I/O PORT STRUCTURE

The CHMOS I/O ports have similar drive capability to their HMOS counterparts but the differences must be noted.

10.3.1 As An Output Pin

The I/O structure is implemented as shown in Figure 10-1. As an output pin latched to a low (0), pullups P1, P2, and P3 are in an off state while the pulldown N1 is on. This configuration uses little current due to no path between V_{CC} and ground exists in the output buffer stage and allows the output pin to be pulled low by N1.

When the output pin is to be latched in the high state, the data line turns off N1 and turns on the weak (5 μA) pullup P2. At the same time P1 (a strong pullup) turns on for one state time pulling the output pin up very quickly. Once the output voltage is above approximately 2 volts, P3 turns on to supply the source current. P3 and the inverter form a latch. This latch along with the support of P3 keeps the output high.

10.3.2 As An Input Pin

To use the I/O as an input, a one must be written to the pin first leaving the pin in the state just discussed. The input device may have a low source current when in the "1" state due to internal pullups P2 and P3 are on.

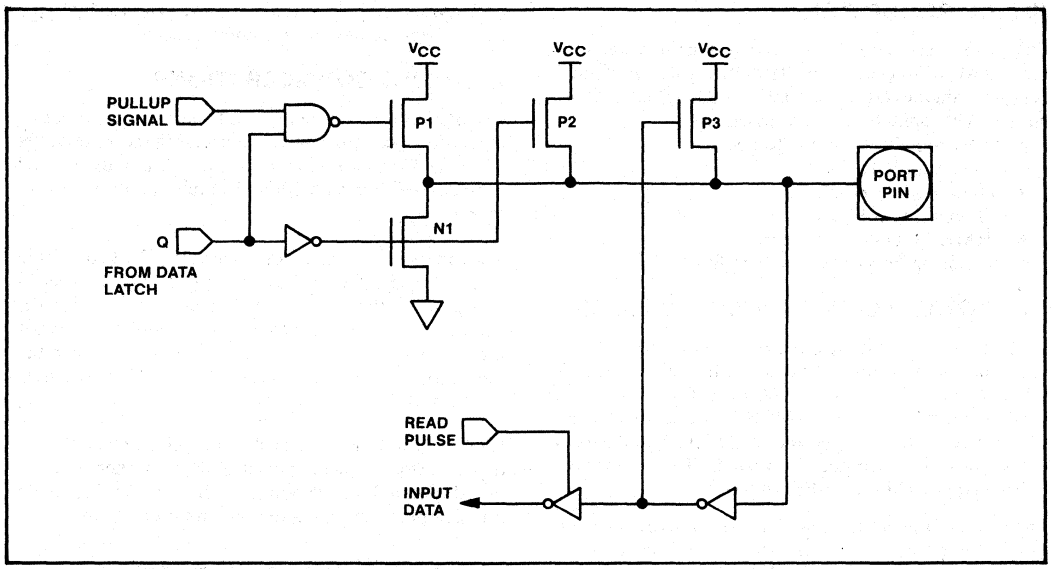


Figure 10-1

When the input goes low past 2 volts, P3 turns off to avoid any high sink currents from being presented to the input device. Note when returning back to a one, P2 is the only internal pullup that is on. This will result in a long rise time if the input device is depending on P2 to pullup the I/O pin.

10.4 INTERFACING BETWEEN CHMOS AND OTHER LOGIC FAMILIES

CHMOS TO TTL, TTL TO CHMOS

CHMOS, when kept within 10% of 5 volts is TTL compatible. No special interfacing considerations other than those in this chapter are needed.

CHMOS TO NMOS, NMOS TO CHMOS

Same as above, when V_{CC} is held to within 10% of 5 volts, no special interface is needed.

CHMOS TO CMOS, CMOS TO CHMOS

CHMOS is CMOS compatible over the supply range of 4 to 6 volts. No special interface is needed.

CHMOS TO TRANSISTOR

CHMOS is capable of sourcing milliamps of short circuit current. When driving a transistor's base with an output, a series resistor should be used to limit the current to the Ioh at the spec'd Voh of the pin.

CHAPTER 11
THE RUPI™-44 FAMILY

11.0 INTRODUCTION

The RUPI-44 family is designed for applications requiring local intelligence at remote nodes, and communication capability among these distributed nodes. The RUPI-44 integrates onto a single chip Intel's highest performance microcontroller, the 8051-core, with an intelligent and high performance HDLC/SDLC communication controller, called the Serial Interface Unit, or SIU. See Figure 11-1. This dual controller architecture allows complex control and high speed data communication functions to be realized cost effectively.

The RUPI-44 family consists of three pin compatible parts:

- 8344—8051 Microcontroller with SIU
- 8044—An 8344 with 4K bytes of on-chip ROM program memory.
- 8744—An 8344 with 4K bytes of on-chip EPROM program memory.

11.1 ARCHITECTURE OVERVIEW

The 8044's dual controller architecture enables the RUPI to perform complex control tasks and high speed communication in a distributed network environment.

The 8044 microcontroller is the 8051-core, and maintains complete software compatibility with it. The microcontroller contains a powerful CPU with on-chip peripherals, making it capable of serving sophisticated real-time control applications such as instrumentation, industrial control, and intelligent computer peripherals. The microcontroller features on-chip peripherals such as two 16-bit timer/counters and 5 source interrupt capability with programmable priority levels. The microcontroller's high performance CPU executes most instructions in 1 microsecond, and can perform an 8 × 8 multiply in 4 microseconds. The CPU features a Boolean processor that can perform operations on 256 directly addressable bits. 192 bytes of on-chip data RAM can be extended to 64K bytes externally. 4K bytes of on-chip program ROM can be extended to 64K bytes externally. The CPU and SIU run concurrently. See Figure 11-2.

The SIU is designed to perform serial communications with little or no CPU involvement. The SIU supports data rates up to 2.4Mbps, externally clocked, and 375K bps self clocked (i.e., the data clock is recovered by an on-chip digital phase locked loop). SIU hardware supports the HDLC/SDLC protocol: zero bit insertion/deletion, address recognition, cyclic redundancy check, and frame number sequence check are automatically performed.

The SIU's Auto mode greatly reduces communication software overhead. The AUTO mode supports the SDLC Normal Response Mode, by performing secondary station responses in hardware without any CPU involvement. The Auto mode's interrupt control and frame sequence numbering capability eliminates software overhead normally required in conventional systems. By using the Auto mode, the CPU is free to concentrate on real time control of the application.

11.2 THE HDLC/SDLC PROTOCOLS

11.2.1 HDLC/SDLC Advantages

HDLC and SDLC are both well recognized standard serial protocols. The Synchronous Data Link Control, SDLC, is an IBM standard communication protocol.

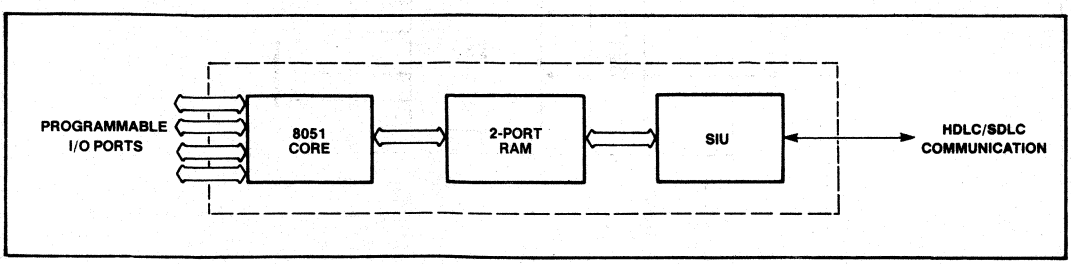


Figure 11-1. RUPI™-44 Dual Controller Architecture

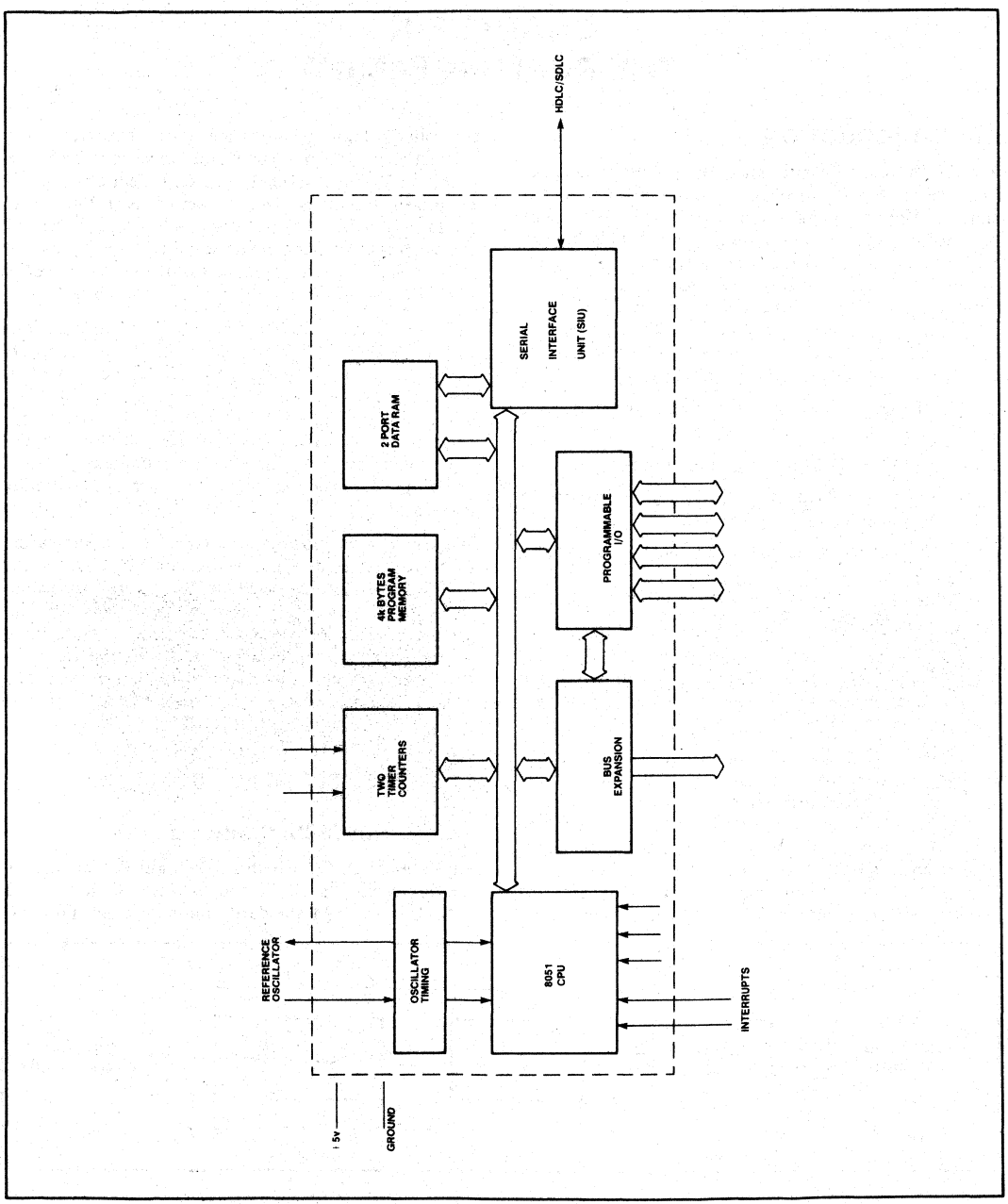


Figure 11-2. Simplified 8044 Block Diagram

The High Level Data Link Control, HDLC, is a standard communication link control established by the International Standards Organization (ISO). SDLC is a subset of HDLC.

The HDLC/SDLC protocols can be used to realize low cost and reliable data links. Because these are serial protocols, expensive cabling and interconnect hardware can be replaced by a low cost two wire link. The corresponding reduced physical connections ensures increased reliability over multi-wire links. Data transmission reliability is ensured at the bit level by sending a frame check sequence using cyclic redundancy with the frame. Reliable frame transmisssion is ensured by sending a frame number identification with each frame.

HDLC/SDLC are "data transparent" protocols. Data transparency means that an arbitrary data stream can be sent without concern that some of the data could be mistaken for a protocol control character. Data transparency relieves the communication controller of having to detect special characters, as in Bisync protocol. Thus, system complexity is reduced.

11.2.2 HDLC/SDLC Networks

In both the HDLC and SDLC line protocols a (Master) primary station controls the overall network (data link) and issues commands to the secondary (Slave) stations. The latter complies with instructions and responds by sending appropriate responses. Whenever a transmitting station must end transmission prematurely, it sends an abort character. Upon detecting an abort character, a receiving station ignores the transmission block called a frame.

RUPI-44 supported HDLC/SDLC network configurations are point to point (half duplex) multipoint (half duplex), and loop. In the loop configuration the stations themselves act as repeaters, so that long links can be easily realized, see Figure 11-3.

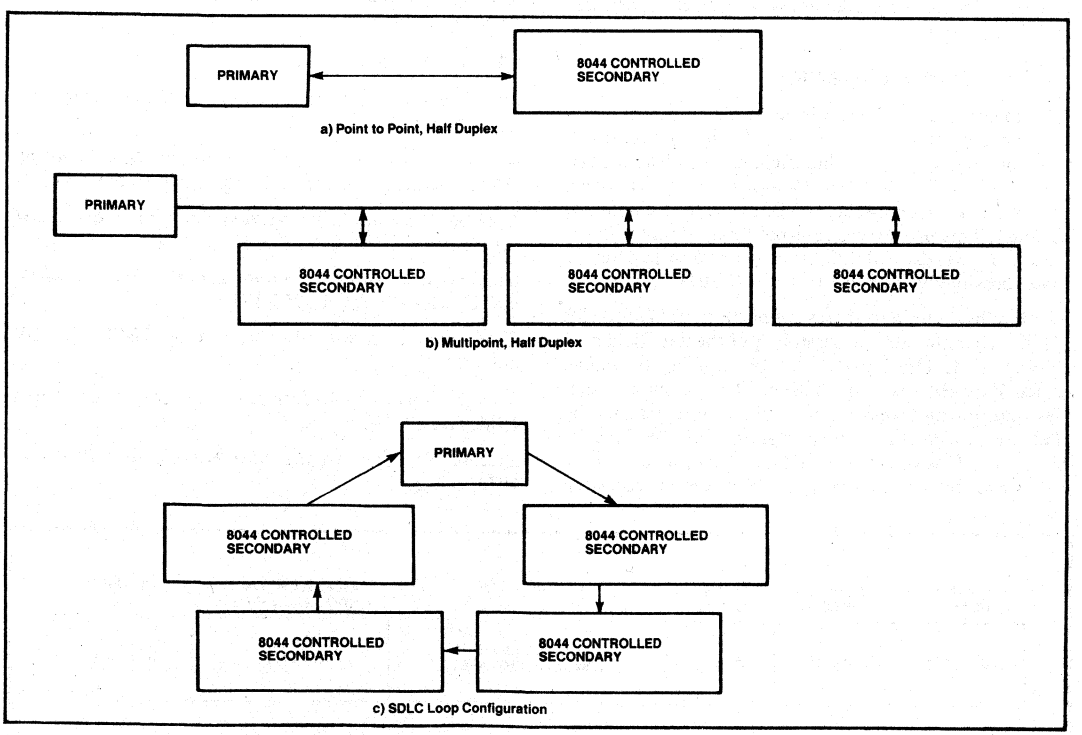


Figure 11-3. RUPI™-44 Supported Network Configurations

11.2.3 Frames

An HDLC/SDLC frame consists of five basic fields: Flag, Address, Control, Data and Error Detection. A frame is bounded by flags—opening and closing flags. An address field is 8 bits wide in SDLC, extendable to 2 or more bytes in HDLC. The control field is also 8 bits wide, extendable to two bytes in HDLC. The SDLC data field or information field may be any number of bytes. The HDLC data field may or may not be on an 8 bit boundary. A powerful error detection code called Frame Check Sequence contains the calculated CRC (Cycle Redundancy Code) for all the bits between the flags. See Figure 11-4.

In HDLC and SDLC are three types of frames; an Information Frame is used to transfer data, a Supervisory Frame is used for control purposes, and a Non-sequenced Frame is used for initialization and control of the secondary stations.

For a more detailed discussion of higher level protocol functions interested readers may refer to the references listed in Section 11.1.6.

11.2.4 Zero Bit Insertion

In data communications, it is desirable to transmit data which can be of arbitrary content. Arbitrary data transmission requires that the data field cannot contain characters which are defined to assist the transmission protocol (like opening flag in HDLC/SDLC communications). This property is referred to as "data transparency". In HDLC/SDLC, this code transparency is made possible by Zero Bit Insertion (ZBI).

The flag has a unique binary bit pattern: 01111110 (7E HEX). To eliminate the possibility of the data field containing a 7E HEX pattern, a bit stuffing technique called Zero Bit Insertion is used. This technique specifies that during transmission, a binary 0 be inserted by the transmitter after any succession of five contiguous binary 1's. This will ensure that no pattern of 0 1 1 1 1 1 0 is ever transmitted between flags. On the receiving side, after receiving the flag, the receiver hardware automatically deletes any 0 following five consecutive 1's. The 8044 performs zero bit insertion and deletion automatically.

11.2.5 Non-return to Zero Inverted (NRZI)

NRZI is a method of clock and data encoding that is well suited to the HDLC/SDLC protocol. It allows HDLC/SDLC protocols to be used with low cost asynchronous modems. NRZI coding is done at the transmitter to enable clock recovery from the data at the receiver terminal by using standard digital phase locked loop (DPLL) techniques. NRZI coding specifies that the signal condition does not change for transmitting a 1, while an 0 causes a change of state. NRZI coding ensures that an active data line will have a transition at least every 5-bit times (recall Zero Bit Insertion), while contiguous 0's will cause a change of state. Thus, ZBI and NRZI encoding makes it possible for the 8044's on-chip DPLL to recover a receive clock (from received data) synchronized to the received data and at the same time ensure data transparency.

11.2.6 References

IBM Synchronous Data Link Control General Information, IBM, GA 27-3093-1.

Standard Network Access Protocol Specification, DATAPAC Trans-Canada Telephone System CCG111.

IBM 3650 Retail Store System Loop Interface OEM Information, IBM, GA 27-3098-0.

Guidebook to Data Communications, Training Manual, Hewlett-Packard 5955-1715.

IBM Introduction to Teleprocessing, IBM, GC 20-8095-02.

System Network Architecture, Technical Overview, IBM, GA 27-3102.

System Network Architecture Format and Protocol, IBM, GA 27-3112.

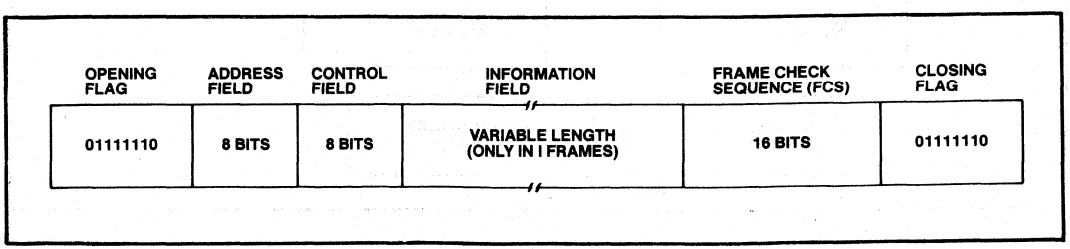


Figure 11-4. Frame Format

11.3 RUPI™-44 DESIGN SUPPORT

11.3.1 Design Tool Support

A critical design consideration is time to market. Intel provides a sophisticated set of design tools to speed hardware and software development time of 8044 based products. These include ICE-44, ASM-51, and PL/M-51.

A primary tool is the 8044 In Circuit Emulator, called ICE-44. See Figure 11-5. In conjunction with Intel's Intellec® Microprocessor Development System, the ICE-44 emulator allows hardware and software development to proceed interactively. This approach is more effective than the traditional method of independent hardware and software development followed by system integration. With the ICE-44 module, prototype hardware can be added to the system as it is designed. Software and hardware integration occurs while the product is being developed.

The ICE-44 emulator assists four stages of development:

1) Software Debugging

   It can be operated without being connected to the user's system before any of the user's hardware is available. In this stage ICE-44 debugging capabilities can be used in conjunction with the Intellec text editor and 8044 macroassembler to facilitate program development.

2) Hardware Development

   The ICE-44 module's precise emulation characteristics and full-speed program RAM make it a valuable tool for debugging hardware, including the time-critical SDLC serial port, parallel port, and timer interfaces.

3) System Integration

   Integration of software and hardware can begin when any functional element of the user system hardware is connected to the 8044 socket. As each section of the user's hardware is completed, it is added to the prototype. Thus, each section of the hardware and software is system tested in real-time operation as it becomes available.

4) System Test

   When the user's prototype is complete, it is tested with the final version of the user system software. The ICE-44 module is then used for real-time emulation of the 8044 to debug the system as a completed unit.

The final product verification test may be performed using the 8744 EPROM version of the 8044 microcomputer. Thus, the ICE-44 module provides the user with the ability to debug a prototype or production system at any stage in its development.

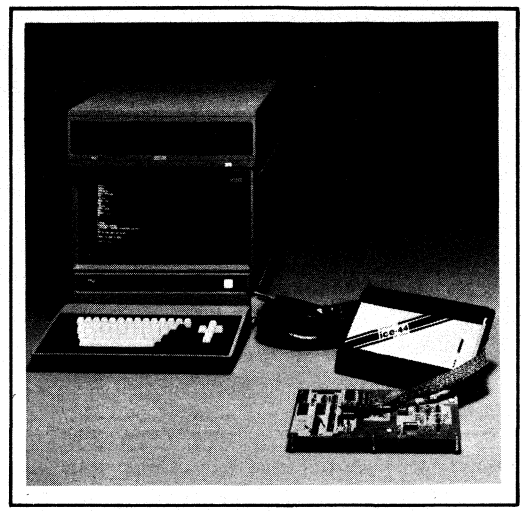


Figure 11-5. RUPI™-44 Development Support Configuration Intellec® System, ICE™-44 Buffer Box, and ICE-44 Module Plugged into a User Prototype Board.

A conversion kit, ICE-44 CON, is available to upgrade an ICE-51 module to ICE-44.

Intel's ASM-51 Assembler supports the 8044 special function registers and assembly program development. PL/M-51 provides designers with a high level language for the 8044. Programming in PL/M can greatly reduce development time, and ensure quick time to market.

11.3.2 8051 Workshop

Intel provides 8051 training to its customers through the 5-day 8051 workshop. Familiarity with the 8051 and 8044 is achieved through a combination of lecture and laboratory exercises.

For designers not familiar with the 8051, the workshop is an effective way to become proficient with the 8051 architecture and capabilities.

CHAPTER 12
8044 ARCHITECTURE

12.0 GENERAL

The 8044 is based on the 8051 core. The 8044 replaces the 8051's serial port with an intelligent HDLC/SDLC controller called the Serial Interface or SIU. Thus the differences between the two result from the 8044's increased on-chip RAM (192 bytes) and additional special function registers necessary to control the SIU. Aside from the increased memory, the SIU itself, and differences in 5 pins (for the serial port), the 8044 and 8051 are compatible.

This chapter describes the differences between the 8044 and 8051. Information pertaining to the 8051 core, eg. instruction set, port operation, EPROM programming, etc. is located in the 8051 sections of this manual.

A block diagram of the 8044 is shown in figure 12-1. The pinout is shown on the inside front cover.

12.1 MEMORY ORGANIZATION OVERVIEW

The 8044 maintains separate address spaces for Program Memory and Data Memory. The Program Memory can be up to 64K bytes long, of which the lowest 4K bytes are in the on-chip ROM.

If the $\overline{EA}$ pin is held high, the 8044 executes out of internal ROM unless the Program Counter exceeds 0FFFH. Fetches from locations 1000H through FFFFH are directed to external Program Memory.

If the $\overline{EA}$ pin is held low, the 8044 fetches all instructions from external Program Memory.

The Data Memory consists of 192 bytes of on-chip RAM, plus 35 Special Function Registers, in addition to which the device is capable of accessing up to 64K bytes of external data memory.

The Program Memory uses 16-bit addresses. The external Data Memory can use either 8-bit or 16-bit addresses. The internal Data Memory uses 8-bit addresses, which provide a 256-location address space. The lower 192 addresses access the on-chip RAM. The Special Function Registers occupy various locations in the upper 128 bytes of the same address space.

The lowest 32 bytes in the internal RAM (locations 00 through 1FH) are divided into 4 banks of registers, each bank consisting of 8 bytes. Any one of these banks can be selected to be the "working registers" of the CPU, and can be accessed by a 3-bit address in the same byte as the opcode of an instruction. Thus, a large number of instructions are one-byte instructions.

The next higher 16 bytes of the internal RAM (locations 20H through 2FH) have individually addressable bits. These are provided for use as software flags or for one-bit (Boolean) processing. This bit-addressing capability is an important feature of the 8044. In addition to the 128 individually addressable bits in RAM, twelve of the Special Function Registers also have individually addressable bits.

A memory map is shown in Figure 12-2.

12.1.1 Special Function Registers

The Special Function Registers are as follows:

*	ACC	Accumulator (A Register)
*	B	B Register
*	PSW	Program Status Word
	SP	Stack Pointer
	DPTR	Data Pointer (consisting of DPH AND DPL)
*	P0	Port 0
*	P1	Port 1
*	P2	Port 2
*	P3	Port 3
*	IP	Interrupt Priority
*	IE	Interrupt Enable
	TMOD	Timer/Counter Mode
*	TCON	Timer/Counter Control
	TH0	Timer/Counter 0 (high byte)
	TL0	Timer/Counter 0 (low byte)
	THI	Timer/Counter 1 (high byte)
	TL1	Timer/Counter 1 (low byte)
	SMD	Serial Mode
*	STS	Status/Command
*	NSNR	Send/Receive Count
	STAD	Station Address
	TBS	Transmit Buffer Start Address
	TBL	Transmit Buffer Length
	TCB	Transmit Control Byte
	RBS	Receive Buffer Start Address
	RBL	Receive Buffer Length
	RFL	Received Field Length
	RCB	Received Control Byte
	DMS CNT	DMA Count
	FIFO	FIFO (three bytes)
	SIUST	SIU State Counter
	PCON	Power Control

The registers marked with * are both byte- and bit-addressable.

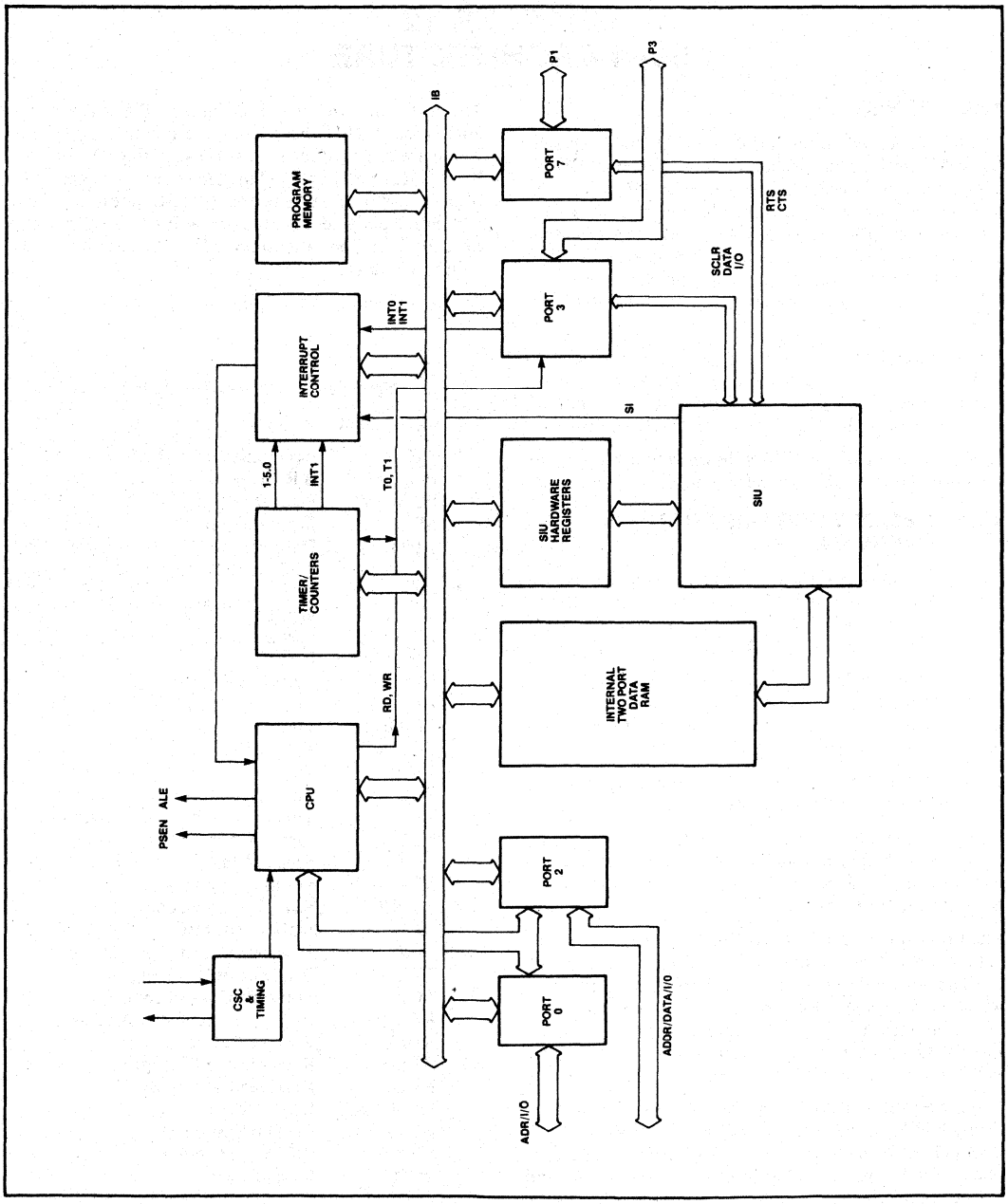


Figure 12-1 RUPI™ Block Diagram

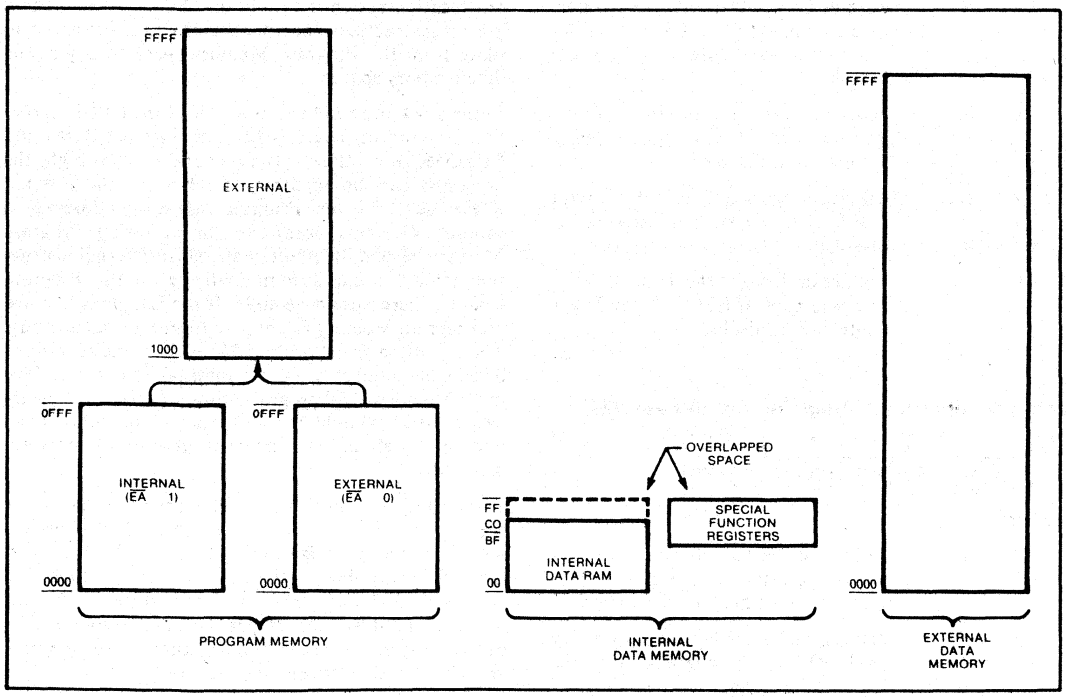


Figure 12-2. RUPI™-44 Memory Map

Stack Pointer

The Stack Pointer is 8 bits wide. The stack can reside anywhere in the 192 bytes of on-chip RAM. When the 8044 is reset, the stack pointer is initialized to 07H. When executing a PUSH or a CALL, the stack pointer is incremented before data is stored, so the stack would begin at location 08H.

12.1.2 Interrupt Control Registers

The Interrupt Request Flags are as listed below:

Source	*Request Flag*	*Location*
External Interrupt 0	$\overline{INT0}$, if IT0 = 0	P3.2
	IE0, if IT0 = 1	TCON.1
Timer 0 Overflow	TF0	TCON.5
External Interrupt 1	$\overline{INT1}$, if IT1 = 0	P3.3
	IE1, if IT1 = 1	TCON.3
Timer 1 Overflow	TF1	TCON.7
Serial Interface Unit	SI	STS.4

External Interrupt control bits IT0 and IT1 are in TCON.0 and TCON.2, respectively. Reset leaves all flags inactive, with IT0 and IT1 cleared.

All the interrupt flags can be set or cleared by software, with the same effect as by hardware.

The Enable and Priority Control Registers are shown below. All of these control bits are set or cleared by software. All are cleared by reset.

IE: Interrupt Enable Register (bit-addressable)

Bit: 7 6 5 4 3 2 1 0

EA	X	X	ES	ET1	EX1	ET0	EX0

where:

- EA disables all interrupts. If EA = 0, no interrupt will be acknowledged. If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.

- ES enables or disables the Serial Interface Unit interrupt. If ES = 0, the Serial Interface Unit interrupt is disabled.
- ET1 enables or disables the Timer 1 Overflow interrupt. If ET1 = 0, the Timer 1 interrupt is disabled.
- EX1 enables or disables External Interrupt 1. If EX1 = 0, External Interrupt 1 is disabled.
- ET0 enables or disables the Timer 0 Overflow interrupt. If ET0 = 0, the Timer 0 interrupt is disabled.

IP: Interrupt Priority Register (bit-addressable)

Bit:	7	6	5	4	3	2	1	0
	X	X	X	PS	PT1	PX1	PT0	PX0

where:

- PS defines the Serial Interface Unit interrupt priority level. PS = 1 programs it to the higher priority level.
- PT1 defines the Timer 1 interrupt priority level. PT1 = 1 programs it to the higher priority level.
- PX1 defines the External Interrupt 1 priority level. PX1 = 1 programs it to the higher priority level.
- PT0 defines the Timer 0 interrupt priority level. PT0 = 1 programs it to the higher priority level.
- PX0 defines the External Interrupt 0 priority level. PX0 = 1 programs it to the higher priority level.

12.2 Memory Organization Details

In the 8044 family the memory is organized over three address spaces and the program counter. The memory spaces shown in Figure 12-2 are the:

- 64K-byte Program Memory address space
- 64K-byte External Data Memory address space
- 384-byte Internal Data Memory address space

The 16-bit Program Counter register provides the 8044 with its 64K addressing capabilities. The Program Counter allows the user to execute calls and branches to any location within the Program Memory space. There are no instructions that permit program execution to move from the Program Memory space to any of the data memory spaces.

In the 8044 and 8744 the lower 4K of the 64K Program Memory address space is filled by internal ROM and EPROM, respectively. By tying the $\overline{EA}$ pin high, the processor can be forced to fetch from the internal ROM/EPROM for Program Memory addresses 0 through 4K. Bus expansion for accessing Program Memory beyond 4K is automatic since external instruction fetches occur automatically when the Program Counter increases above 4095. If the $\overline{EA}$ pin is tied low all Program Memory fetches are from external memory. The execution speed of the 8044 is the same regardless of whether fetches are from internal or external Program Memory. If all program storage is on-chip, byte location 4095 should be left vacant to prevent an undesired prefetch from external Program Memory address 4096.

Certain locations in Program Memory are reserved for specific programs. Locations 0000 through 0002 are reserved for the initialization program. Following reset, the CPU always begins execution at location 0000. Locations 0003 through 0042 are reserved for the five interrupt-request service programs. Each resource that can request an interrupt requires that its service program be stored at its reserved location.

The 64K-byte External Data Memory address space is automatically accessed when the MOVX instruction is executed.

Functionally the Internal Data Memory is the most flexible of the address spaces. The Internal Data Memory space is subdivided into a 256-byte Internal Data RAM address space and a 128-byte Special Function Register address space as shown in Figure 12-3.

The Internal Data RAM address space is 0 to 255. Four 8-Register Banks occupy locations 0 through 31. The stack can be located anywhere in the Internal Data RAM address space. In addition, 128 bit locations of the on-chip RAM are accessible through Direct Addressing. These bits reside in Internal Data RAM at byte locations 32 through 47. Currently locations 0 through 191 of the Internal Data RAM address space are filled with on-chip RAM.

The stack depth is limited only by the available Internal Data RAM, thanks to an 8-bit reloadable Stack Pointer. The stack is used for storing the Program Counter during subroutine calls and may be used for passing parameters. Any byte of Internal Data RAM or Special Function Register accessible through Direct Addressing can be pushed/popped.

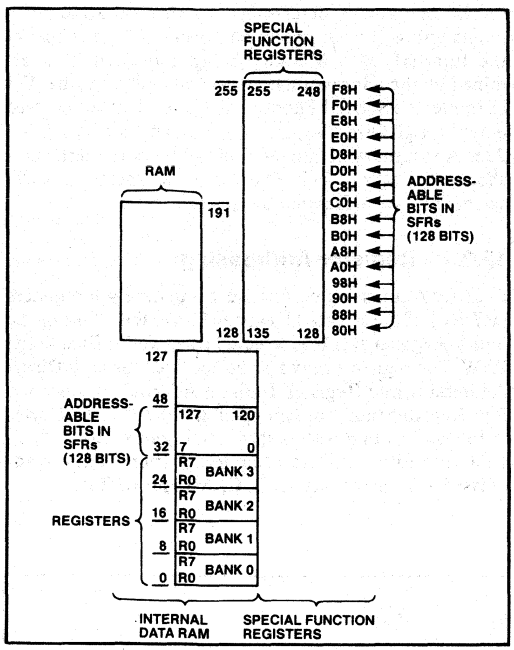


Figure 12-3. Internal Data Memory Address Space

ARITHMETIC REGISTERS:
- Accumulator*, B register*,
- Progam Status Word*

POINTERS:
- Stack Pointer, Data Pointer (high & low)

PARALLEL I/O PORTS:
- Port 3*, Port 2*, Port 1*, Port 0*

INTERRUPT SYSTEM:
- Interrupt Priority Control*,
- Interrupt Enable Control*

TIMERS:
- Timer MODe, Timer CONtrol*, Timer 1 (high & low), Timer 0 (high & low)

SERIAL INTERFACE UNIT:
- Transmit Buffer Start,
- Transmit Buffer Length,
- Transmit Control Byte,
- Send Count Receive Count*,
- DMA Count,
- Station Address
- Receive Field Length
- Receive Buffer Start
- Receive Buffer Length
- Receive Control Byte,
- Serial Mode,
- Status Register.*

*** Bits in these registers are bit addressable.**

Figure 12-4. Special Function Registers

The Special Function Register address space is 128 to 255. All registers except the Program Counter and the four 8-Register Banks reside here. Memory mapping the Special Function Registers allows them to be accessed as easily as internal RAM. As such, they can be operated on by most instructions. In the overlapping memory space (address 128–191), indirect addressing is used to access RAM, and direct addressing is used to access the SFR's. The SFR's at addresses 192–255 are also accessed using direct addressing. The Special Function Registers are listed in Figure 12-4. Their mapping in the Special Function Register address space is shown in Figures 12-5 and 12-6.

Performing a read from a location of the Internal Data memory where neither a byte of Internal Data RAM (i.e., RAM addresses 192–255) nor a Special Function Register exists will access data of indeterminable value.

Architecturally, each memory space is a linear sequence of 8-bit wide bytes. By Intel convention the storage of multi-byte address and data operands in program and data memories is the least significant byte at the low-order address and the most significant byte at the high-order address. Within byte X, the most significant bit is represented by X.7 while the least significant bit is X.0. Any deviation from these conventions will be explicitly stated in the text.

12.2.1 Operand Addressing

There are five methods of addressing source operands. They are Register Addressing, Direct Addressing, Register-Indirect Addressing, Immediate Addressing and Base-Register-plus Index-Register-Indirect Addressing. The first three of these methods can also be used to address a destination operand. Since operations in the 8044 require 0 (NOP only), 1, 2, 3 or 4 operands, these five addressing methods are used in combinations to provide the 8044 with its 21 addressing modes.

Most instructions have a "destination, source" field that specifies the data type, addressing methods and operands involved. For operations other than moves, the destination operand is also a source operand. For example, in "subtract-with-borrow A, #5" the A register receives the result of the value in register A minus 5, minus C.

Most operations involve operands that are located in Internal Data Memory. The selection of the Program Memory space or External Data Memory space for a

ARITHMETIC REGISTERS:
Accumulator*, B register*,
Progam Status Word*
POINTERS:
Stack Pointer, Data Pointer (high & low)
PARALLEL I/O PORTS:
Port 3*, Port 2*, Port 1*, Port 0*
INTERRUPT SYSTEM:
Interrupt Priority Control*,
Interrupt Enable Control*
TIMERS:
Timer Mode, Timer Control*, Timer 1 (high & low), Timer 0 (high & low)
SERIAL INTERFACE UNIT:
Serial Mode, Status/Command*,
Send/Receive Count*, Station Address,
Transmit Buffer Start Address,
Transmit Buffer Length,
Transmit Control Byte,
Receive Buffer Start Address,
Receive Buffer Length,
Receive Field Length,
Receive Control Byte,
DMA Count,
FIFO (three bytes),
SIU Controller State Counter

* Bits in these registers are bit-addressable

Figure 12-5. Mapping of Special Function Registers

second operand is determined by the operation mnemonic unless it is an immediate operand. The subset of the Internal Data Memory being addressed is determined by the addressing method and address value. For example, the Special Function Registers can be accessed only through Direct Addressing with an address of 128–255. A summary of the operand addressing methods is shown in Figure 12-6. The following paragraphs describe the five addressing methods.

12.2.2 Register Addressing

Register Addressing permits access to the eight registers (R7-R0) of the selected Register Bank (RB). One of the four 8-Register Banks is selected by a two-bit field in the PSW. The registers may also be accessed through Direct Addressing and Register-Indirect Addressing, since the four Register Banks are mapped into the lowest 32 bytes of Internal Data RAM as shown in Figures 12-9 and 12-10. Other Internal Data Memory locations that are addressed as registers are A, B, C, AB and DPTR.

REGISTER NAMES	SYMBOLIC ADDRESS	BIT ADDRESS	BYTE ADDRESS
B REGISTER	B	247 through 240	240 (F0H)
ACCUMULATOR	ACC	231 through 224	224 (E0H)
*THREE BYTE FIFO	FIFO		223 (DFH)
	FIFO		222 (DEH)
	FIFO		221 (DDH)
TRANSMIT BUFFER START	TBS		220 (DCH)
TRANSMIT BUFFER LENGTH	TBL		219 (DBH)
TRANSMIT CONTROL BYTE	TCB		218 (DAH)
*SIU STATE COUNTER	SIUST		217 (D9H)
SEND COUNT RECEIVE COUNT	NSNR	223 through 216	216 (D8H)
PROGRAM STATUS WORD	PSW	215 through 208	208 (D0H)
*DMA COUNT	DMA CNT		207 (CFH)
STATION ADDRESS	STAD		206 (CEH)
RECEIVE FIELD LENGTH	RFL		205 (CDH)
RECEIVE BUFFER START	RBS		204 (CCH)
RECEIVE BUFFER LENGTH	RBL		203 (CBH)
RECEIVE CONTROL BYTE	RCB		202 (CAH)
SERIAL MODE	SMD		201 (C9H)
STATUS REGISTER	STS	207 through 200	200 (C8H)
INTERRUPT PRIORITY CONTROL	IP	191 through 184	184 (B8H)
PORT 3	P3	183 through 176	176 (B0H)
INTERRUPT ENABLE CONTROL	IE	175 through 168	168 (A8H)
PORT 2	P2	167 through 160	160 (A0H)
PORT 1	P1	151 through 144	144 (90H)
TIMER HIGH 1	TH1		141 (8DH)
TIMER HIGH 0	TH0		140 (8CH)
TIMER LOW 1	TL1		139 (8BH)
TIMER LOW 0	TL0		138 (8AH)
TIMER MODE	TMOD		137 (89H)
TIMER CONTROL	TCON	143 through 136	136 (88H)
DATA POINTER HIGH	DPH		131 (83H)
DATA POINTER LOW	DPL		130 (82H)
STACK POINTER	SP		129 (81H)
PORT 0	P0	135 through 128	128 (80H)

SFR's CONTAINING DIRECT ADDRESSABLE BITS

*ICE Support Hardware registers. Under normal operating conditions there is no need for the CPU to access these registers.

12-6. Mapping of Special Function Registers

Direct Byte Address	Bit Address (MSB)							(LSB)	Hardware Register Symbol
240	F7	F6	F5	F4	F3	F2	F1	F0	B
224	E7	E6	E5	E4	E3	E2	E1	E0	ACC
	NS2	NS1	NS0	SES	NR2	NR1	NR0	SER	
216	DF	DE	DD	DC	DB	DA	D9	D8	NSNR
	CY	AC	FO	RS1	RS0	OV		P	
204	D7	D6	D5	D4	D3	D2	D1	D0	PSW
	TBF	RE	RTS	SI	BV	CPB	AM	RBP	
200	CF	CE	CD	CC	CB	CA	C9	C8	STS
				PS	PT1	PX1	PT0	PX0	
184	—	—	—	BC	BB	BA	B9	B8	IP
176	B7	B6	B5	B4	B3	B2	B1	B0	P3
	EA			ES	ET1	EX1	ET0	EX0	
168	AF	—	—	AC	AB	AA	A9	A8	IE
160	A7	A6	A5	A4	A3	A2	A1	A0	P2
144	97	96	95	94	93	92	91	90	P1
	TF1	TR1	TF0	TR0	IE1	IT1	IE	IT0	
136	8F	8E	8D	8C	8B	8A	89	88	TCON
128	87	86	85	84	83	82	81	80	P0

Figure 12-7. Special Function Register Bit Address

- **Register Addressing**
  - R7-R0
  - A,B,C (bit), AB (two bytes), DPTR (double byte)
- **Direct Addressing**
  - Lower 128 bytes of Internal Data RAM
  - Special Function Registers
  - 128 bits in subset of Special Function Register address space
- **Register-Indirect Addressing**
  - Internal Data RAM [@R1, @R0, @SP (PUSH and POP only)]
  - Least Significant Nibbles in Internal Data RAM (@R1, @R0)
  - External Data Memory (@R1, @R0, @DPTR)
- **Immediate Addressing**
  - Program Memory (in-code constant)
- **Base-Register-plus Index-Register-Indirect Addressing**
  - Program Memory (@ DPTR + A, @ PC+A)

Figure 12-8. Operand Addressing Methods

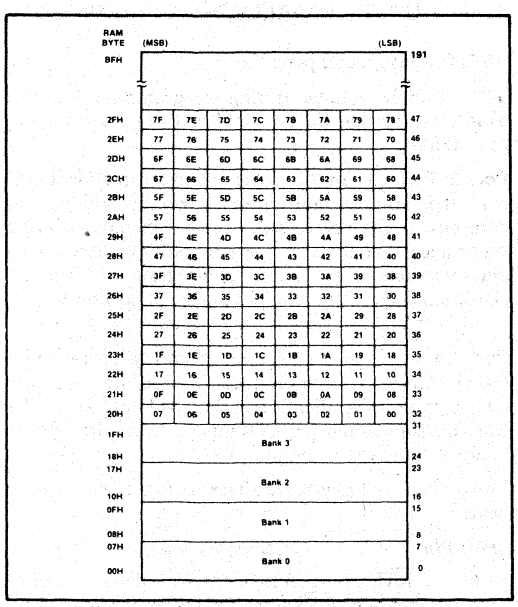


Figure 12-9. RAM Bit Addresses

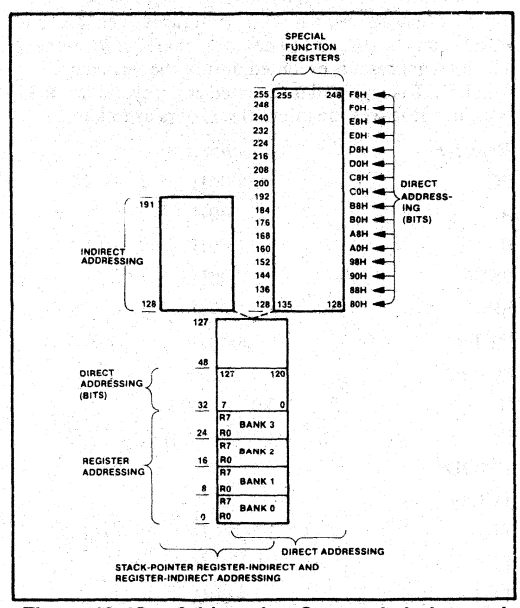


Figure 12-10. Addressing Operands in Internal Data Memory

12.2.3 Direct Addressing

Direct Addressing provides the only means of accessing the memory-mapped byte-wide Special Function Registers and memory mapped bits within the Special Function Registers and Internal Data RAM. Direct Addressing of bytes may also be used to access the lower 128 bytes of Internal Data RAM. Direct Addressing of bits gains access to a 128 bit subset of the Internal Data RAM and 128 bit subset of the Special Function Registers as shown in Figures 12-5, 12-6, 12-9, and 12-10.

Register-Indirect Addressing using the content of R1 or R0 in the selected Register Bank, or using the content of the Stack Pointer (PUSH and POP only), addresses the Internal Data RAM. Register-Indirect Addressing is also used for accessing the External Data Memory. In this case, either R1 or R0 in the selected Register Bank may be used for accessing locations within a 256-byte block. The block number can be preselected by the contents of a port. The 16-bit Data Pointer may be used for accessing any location within the full 64K external address space.

12.3 RESET

Reset is accomplished by holding the RST pin high for at least two machine cycles (24 oscillator periods) *while the oscillator is running*. The CPU responds by executing an internal reset. It also configures the ALE and PSEN pins as inputs. (They are quasi-bidirectional.) The internal reset is executed during the second cycle in which RST is high and is repeated every cycle until RST goes low. It leaves the internal registers as follows:

Register	*Content*
PC	0000H
A	00H
B	00H
PSW	00H
SP	07H
DPTR	0000H
P0 - P3	0FFH
IP	(XXX00000)
IE	(0XX00000)
TMOD	00H
TCON	00H
TH0	00H
TL0	00H
TH1	00H
TL1	00H
SMD	00H
STS	00H
NSNR	00H
STAD	00H
TBS	00H
TBL	00H
TCB	00H
RBS	00H
RBL	00H
RFL	00H
RCB	00H
DMA CNT	00H
FIFO1	00H
FIFO2	00H
FIFO3	00H
SIUST	01H
PCON	(0XXXXXXX)

The internal RAM is not affected by reset. When VCC is turned on, the RAM content is indeterminate unless VPD was applied prior to VCC being turned off (see Power Down Operation.)

12.4 RUPI™-44 FAMILY PIN DESCRIPTION

VSS: Circuit ground potential.

VCC: Supply voltage during programming (of the 8744), verification (of the 8044 or 8744), and normal operation.

Port 0: Port 0 is an 8-bit open drain bidirectional I/O port. It is also the multiplexed low-order address and data bus during accesses to external memory (during which accesses it activates internal pullups). It also outputs instruction bytes during program verification. (External pullups are required during program verification.) Port 0 can sink eight LS TTL inputs.

Port 1: Port 1 is an 8-bit bidirectional I/O port with internal pullups. It receives the low-order address byte during program verification in the 8044 or 8744. Port 1 can sink/source four LS TTL inputs. It can drive MOS inputs without external pullups.

Two of the Port 1 pins serve alternate functions, as listed below:

Port Pin	*Alternate Function*
P1.6	$\overline{\text{RTS}}$ (Request to Send). In a non-loop configuration, $\overline{\text{RTS}}$ signals that the 8044 is ready to transmit data.

P1.7 $\overline{CTS}$ (Clear to Send). In a non-loop configuration, $\overline{CTS}$ signals to the 8044 that the receiving station is ready to accept data.

Port 2: Port 2 is an 8-bit bidirectional I/O port with internal pullups. It emits the high-order address byte during accesses to external memory. It also receives the high-order address bits and control signals during program verification in the 8044 or 8744. Port 2 can sink/source four LS TTL inputs. It can drive MOS inputs without external pullups.

Port 3: Port 3 is an 8-bit bidirectional I/O port with internal pullups. Port 3 can sink/source four LS TTL inputs. It can drive MOS inputs without external pullups.

Port 3 pins also serve alternate functions, as listed below:

Port Pin	*Alternate Function*
P3.0	RXD (serial input port in loop configuration). $\overline{I/O}$ (data direction control in non-loop configuration).
P3.1	TXD (serial output port in loop configuration). DATA input/output pin in non-loop configuration.
P3.2	$\overline{INT0}$ (external interrupt)
P3.3	$\overline{INT1}$ (external interrupt)
P3.4	T0 (Timer 0 external input)
P3.5	T1 (Timer 1 external input) SCLK (Serial Data Clock Input)
P3.6	$\overline{WR}$ (external Data Memory write strobe)
P3.7	$\overline{RD}$ (external Data Memory read strobe)

RST/VPD: A high level on this pin for two machine cycles while the oscillator is running resets the device. An internal pulldown permits Power-On reset using only a capacitor connected to VCC.

ALE/$\overline{PROG}$: Address Latch Enable output for latching the low byte of the address during accesses to external memory. ALE is activated though for this purpose at a constant rate of 1/6 the oscillator frequency even when external memory is not being accessed. Consequently it can be used for external clocking or timing purposes. (However, one ALE pulse is skipped during each access to external *Data* Memory.) This pin is also the program pulse input ($\overline{PROG}$) during EPROM programming.

$\overline{PSEN}$: Program Store Enable output is the read strobe to external Program Memory. $\overline{PSEN}$ is activated twice each machine cycle during fetches from external Program Memory. (However, when executing out of external Program Memory two activations of $\overline{PSEN}$ are skipped during each access to external *Data* Memory.) $\overline{PSEN}$ is not activated during fetches from internal Program Memory.

$\overline{EA}$/VPP: When EA is held high the CPU executes out of internal Program Memory (unless the Program Counter exceeds 0FFFH). When $\overline{EA}$ is held low the CPU executes only out of external Program Memory. In the 8344, $\overline{EA}$ must be externally wired low. In the 8744, this pin also receives the 21V programming supply voltage (VPP) during EPROM programming.

XTAL1: Input to the inverting amplifier that forms the oscillator. Should be grounded when an external oscillator is used.

XTAL2: Output of the inverting amplifier that forms the oscillator, and input to the internal clock generator. Receives the external oscillator signal when an external oscillator is used.

CHAPTER 13
THE 8044 SERIAL INTERFACE UNIT

13.0 SERIAL INTERFACE

The serial interface provides a high-performance communication link. The protocol used for this communication is based on the IBM Synchronous Data Link Control (SDLC). The serial interface also supports a subset of the ISO HDLC (International Standards Organization High-Level Data Link Control) protocol.

The SDLC/HDLC protocols have been accepted as standard protocols for many high-level teleprocessing systems. The serial interface performs many of the functions required to service the data link without intervention from the 8044's own CPU. The programmer is free to concentrate on the 8044's function as a peripheral controller, rather than having to deal with the details of the communication process.

Five pins on the 8044 are involved with the serial interface (refer to Section 12.4, Family Pin Description, for details):

Pin 7	$\overline{RTS}$/P16
Pin 8	$\overline{CTS}$/P17
Pin 10	I/$\overline{O}$/RXD/P30
Pin 11	DATA/TXD/P31
Pin 15	SCLK/T1/P35

Figure 13-1 is a functional block diagram of the serial interface unit (SIU). More details on the SIU hardware are given in Section 13.9.

13.1 DATA LINK CONFIGURATIONS

The serial interface is capable of operating in three serial data link configurations:

1) Half-Duplex, point-to-point
2) Half-Duplex, multipoint (with a half-duplex or full-duplex primary)
3) Loop

Figure 13-2 shows these three configurations. The RTS (Request to Send) and CTS (Clear to Send) handshaking signals are available in the point-to-point and multipoint configurations.

13.2 DATA CLOCKING OPTIONS

The serial interface can operate in an externally clocked mode or in a self clocked mode.

Externally Clocked Mode

In the externally clocked mode, a common Serial Data Clock (SCLK on pin 15) synchronizes the serial bit stream. This clock signal may come from the master CPU or primary station, or from an external phase-locked loop local to the 8044. Figure 13-3 illustrates the timing relationships for the serial interface signals when the externally clocked mode is used in point-to-point and multipoint data link configurations.

Incoming data is sampled at the rising edge of SCLK, and outgoing data is shifted out at the falling edge of SCLK. More detailed timing information is given in the 8044 data sheet.

Self Clocked (Asynchronous) Mode

The self clocked mode allows data transfer without a common system data clock. Using an on-chip DPLL (digital phase locked loop) the serial interface recovers the data clock from the data stream itself. The DPLL requires a reference clock equal to either 16 times or 32 times the data rate. This reference clock may be externally supplied or internally generated. When the serial interface generates this clock internally, it uses either the 8044's internal logic clock (half the crystal frequency's PH2) or the "timer 1" overflow. Figure 13-4 shows the serial interface signal timing relationships for the loop configuration, when the unclocked mode is used.

The DPLL monitors the received data in order to derive a data clock that is centered on the received bits. Centering is achieved by detecting all transitions of the received data, and then adjusting the clock transition (in increments of 1/16 bit period) toward the center of the received bit. The DPLL converges to the nominal bit center within eight bit transitions, worst case.

To aid in the phase locked loop capture process, the 8044 has a NRZI (non-return-to-zero inverted) data encoding and decoding option. NRZI coding specifies that a signal does not change state for a transmitted binary 1, but does change state for a binary 0. Using the NRZI coding with zero-bit insertion, it can be guaranteed that an active signal line undergoes a transition at least every six bit times.

13.3 DATA RATES

The maximum data rate in the externally clocked mode is 2.4M bits per second (bps) a half-duplex configuration, and 1.0M in a loop configuration.

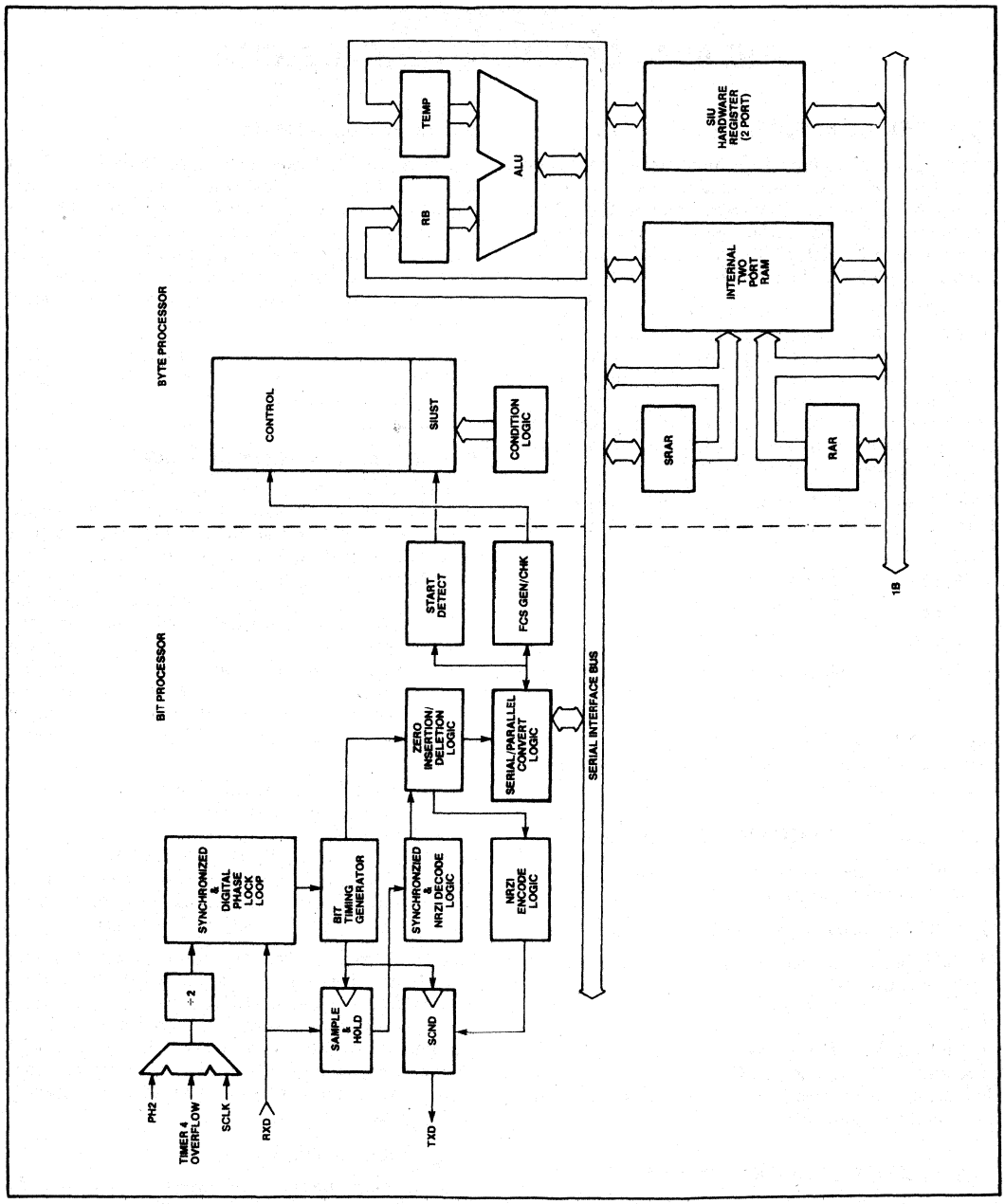


Figure 13-1. SIU Block Diagram

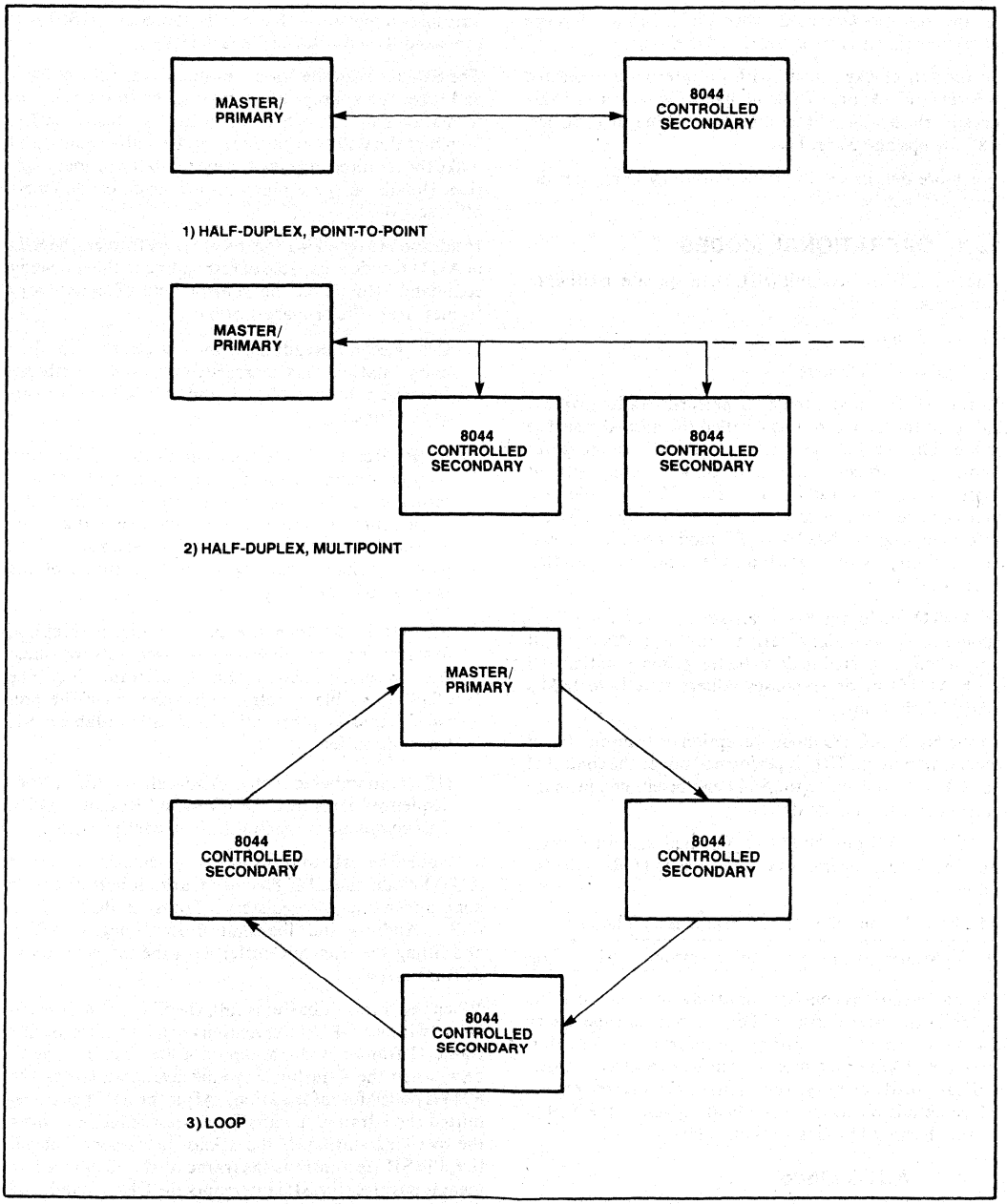


Figure 13-2. RUPI-44 Data Link Configurations

In the self clocked mode with an external reference clock, the maximum data rate is 375K bps.

In the self clocked mode with an internally generated reference clock, and the 8044 operating with a 12 MHz crystal, the available data rates are 244 bps to 62.5k bps, 187.5K bps and 375K bps.

For more details see the table in the SMD register description, below.

13.4 OPERATIONAL MODES

The Serial Interface Unit (SIU) can operate in either of two response modes:

1) AUTO mode

2) NON-AUTO mode

In the AUTO mode, the SIU performs in hardware a subset of the SDLC protocol called the normal response mode. The AUTO mode enables the SIU to recognize and respond to certain kinds of SDLC frames without intervention from the 8044's CPU. AUTO mode provides a faster turnaround time and a simplified software interface, whereas NON-AUTO mode provides a greater flexibility with regard to the kinds of operation permitted.

In AUTO mode, the 8044 can act only as a normal response mode secondary station—that is, it can transmit only when instructed to do so by the primary station. All such AUTO mode responses adhere strictly to IBM's SDLC definitions.

In the NON-AUTO mode, reception or transmission of each frame by the SIU is performed under the control of the CPU. In this mode the 8044 can be either a primary station or a secondary station.

In both AUTO and NON-AUTO modes, short frames, aborted frames, or frames which have bad CRC's are ignored by the SIU.

The basic format of an SDLC frame is as follows:

Flag	Address	Control	Information	FCS	Flag

Format variations consist of omitting one or more of the fields in the SDLC frame. For example, a supervisory frame is formed by omitting the information field. Supervisory frames are used to confirm received frames, indicate ready or busy conditions, and to report errors. More details on frame formats are given in the SDLC Frame Format Options section, below.

13.4.1 AUTO Mode

To enable the SIU to receive a frame in AUTO mode, the 8044 CPU sets up a receive buffer. This is done by writing two registers—Receive Buffer Start (RBS) Address and Receive Buffer Length (RBL).

The SIU receives the frame, examines the control byte, and takes the appropriate action. If the frame is an information frame, the SIU will load the receive buffer, interrupt the CPU (to have the receive buffer read), and make the required acknowledgement to the primary station. Details on these processes are given in the Operation section, below.

In addition to receiving the information frames, the SIU in AUTO mode is capable of responding to the following commands (found in the control field of supervisory frames) from the primary station:

RR (Receive Ready): Acknowledges that the Primary station has correctly received numbered frames up through $N_R - 1$, and that it is ready to receive frame N_R.

RNR (Receive Not Ready): Indicates a temporary busy condition (at the primary station) due to buffering or other internal constraints. The quantity N_R in the control field indicates the number of the frame expected after the busy condition ends, and may be used to acknowledge the correct reception of the frames up through $N_R - 1$.

REJ (Reject): Acknowledges the correct reception of frames up through $N_R - 1$, and requests transmission or retransmission starting at frame N_R. The 8044 is capable of retransmitting at most the previous frame, and then only if it is still available in the transmit buffer.

UP (Unnumbered Poll): Also called NSP (Non-Sequenced Poll) or ORP (Optional Response Poll). This command is used in the loop configuration.

To enable the SIU to transmit an information frame in AUTO mode, the CPU sets up a transmit buffer. This is done by writing two registers—Transmit Buffer Start (TBS) Address and Transmit Buffer Length (TBL), and filling the transmit buffer with the information to be transmitted.

When the transmit buffer is full, the SIU can automatically (without CPU intervention) send an information frame (I-frame) with the appropriate sequence numbers, when the data link becomes available (when the 8044 is polled for information). After the SIU has transmitted the I-frame, it waits for acknowledgement from the receiving station. If the acknowledgement is negative, the SIU retransmits the frame. If the acknowledgement is positive, the SIU interrupts the CPU, to indicate that the transmit buffer may be reloaded with new information.

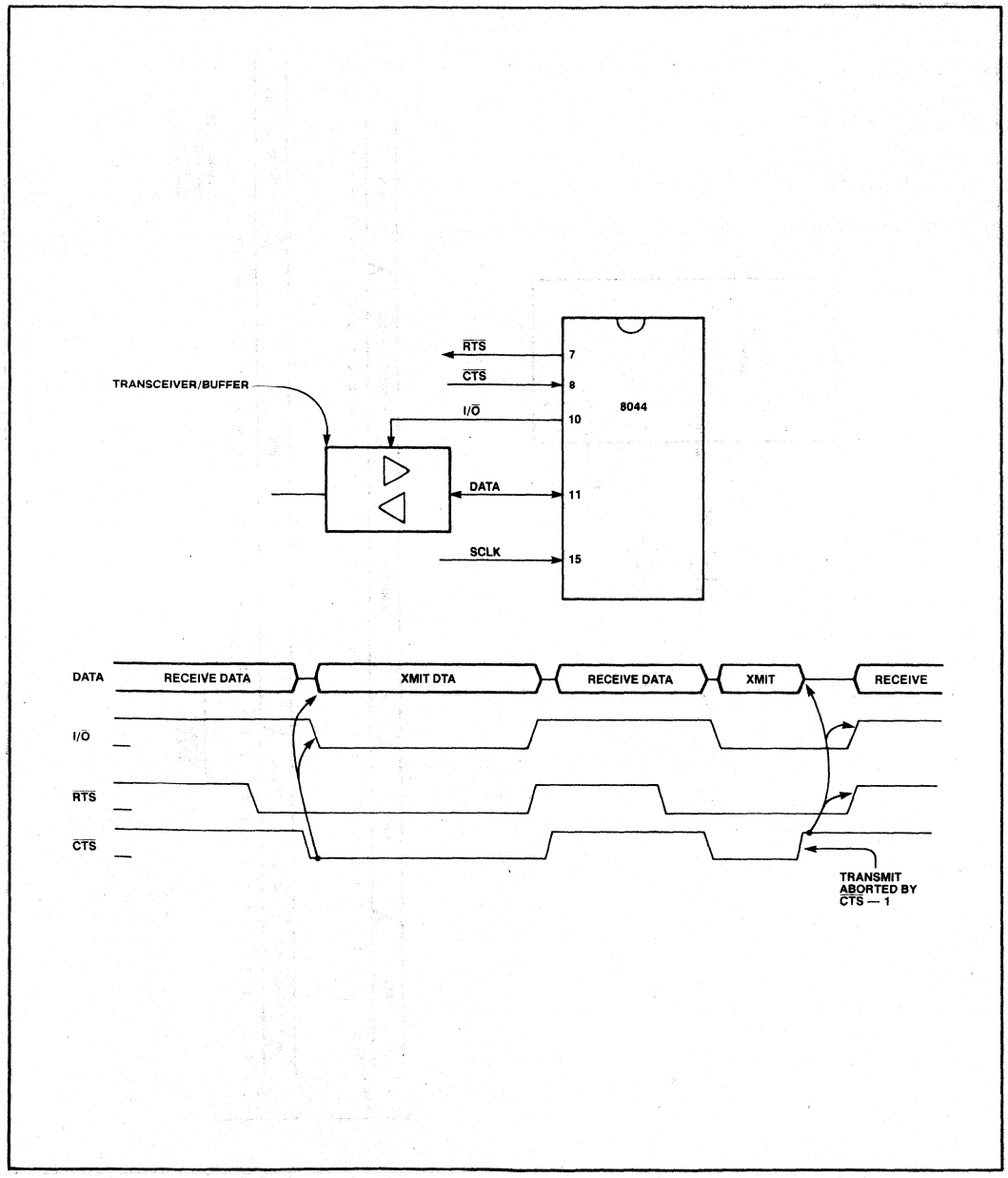


Figure 13-3. Serial Interface Timing—Clocked Mode

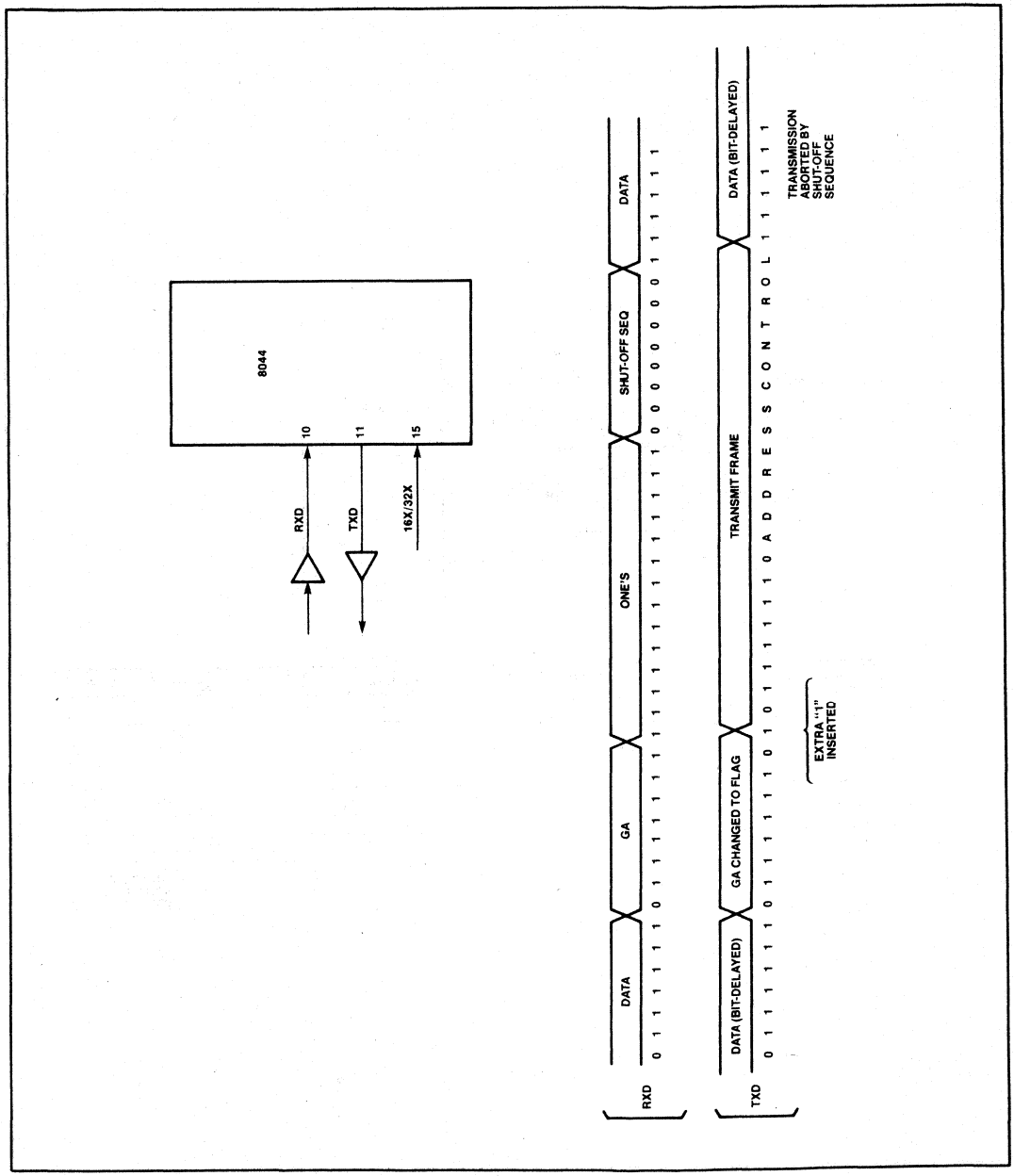


Figure 13-4. Serial Interface Timing—Self Clocked Mode

In addition to transmitting the information frames, the SIU in AUTO mode is capable of sending the following responses to the primary station:

RR (Receive Ready): Acknowledges that the 8044 has correctly received numbered frames up through $N_R - 1$, and that it is ready to receive frame N_R.

RNR (Receive Not Ready): Indicates a temporary busy condition (at the 8044) due to buffering or other internal constraints. The quantity N_R in the control field indicates the number of the frame expected after the busy condition ends, and acknowledges the correct reception of the frames up through $N_R - 1$.

13.4.2 NON-AUTO Mode

In the NON-AUTO (or flexible) mode, all reception and transmission is under the control of the CPU. The full SDLC and HDLC protocols can be implemented, as well as any bit-synchronous variants of these protocols.

NON-AUTO mode provides more flexibility than AUTO mode, but it requires more CPU overhead, and much longer recognition and response times. This is especially true when the CPU is servicing an interrupt that has higher priority than the interrupts from the SIU.

In NON-AUTO mode, when the SIU receives a frame, it interrupts the CPU. The CPU then reads the control byte from the Receive Control Byte (RCB) register. If the received frame is an information frame, the CPU also reads the information from the receive buffer, according to the values in the Receive Buffer Start (RBS) address register and the Received Field Length (RFL) register.

In NON-AUTO mode, the 8044 can initiate transmissions without being polled, and thus it can act as the primary station. To initiate transmission or to generate a response, the CPU sets up and enables the SIU. The SIU then formats and transmits the desired frame. Upon completion of the transmission, without waiting for a positive acknowledgement from the receiving station, the SIU interrupts the CPU.

13.5 8044 FRAME FORMAT OPTIONS

As mentioned above, variations on the basic SDLC frame consist of omitting one or more of the fields. The choice of which fields to omit, as well as the selection of AUTO mode versus NON-AUTO mode, is specified by the settings of the following three bits in the Serial Mode Register (SMD) and the Status/Control Register (STS):

SMD Bit 0: NFCS (No Frame Check Sequence)

SMD Bit 1: NB (Non-Buffered Mode—No Control Field)

STS Bit 1: AM (AUTO Mode or Addressed Mode)

Figure 13-5 shows how these three bits control the frame format.

The following paragraphs discuss some properties of the standard SDLC format, and the significance of omitting some of the fields.

13.5.1 Standard SDLC Format

The standard SDLC format consists of an opening flag, an 8-bit address field, and 8-bit control field, an n-byte information field, a 16-bit Frame Check Sequence (FCS), and a closing flag. The FCS is based on the CCITT-CRC polynominal ($X^{16} + X^{12} + X^5 + 1$). The address and control fields may not be extended. Within the 8044, the address field is held in the Station Address (STAD) register, and the control field is held in the Receive Control Byte (RCB) or Transmit Control Byte (TCB) register. The standard SDLC format may be used in either AUTO mode or NON-AUTO mode.

13.5.2 No Control Field (Non-Buffered Mode)

When the control field is not present, the RCB and TCB registers are not used. The information field begins immediately after the address field, or, if the address field is also absent, immediately after the opening flag. The entire information field is stored in the 8044's on-chip RAM. If there is no control field, NON-AUTO mode must be used. Control information may, of course, be present in the information field, and in this manner the No Control Field option may be used for implementing extended control fields.

13.5.3 No Control Field and No Address Field

The No Address Field option is available only in conjunction with the No Control Field option. The STAD, RCB, and TCB registers are not used. When both these fields are absent, the information field begins immediately after the opening flag. The entire information field is stored in on-chip RAM. NON-AUTO mode must be used. Formats without an address field have the following applications:

Point-to-point data links (where no addressing is necessary)

Monitoring line activity (receiving all messages regardless of the address field)

Extended addressing

FRAME OPTION	NFCS	NB	AM	FRAME FORMAT
Standard SDLC NON-AUTO Mode	0	0	0	F A C I FCS F
Standard SDLC AUTO Mode	0	0	1	F A C I FCS F
No Control Field NON-AUTO Mode	0	1	1	F A I FCS F
No Control Field No Address Field NON-AUTO Mode	0	1	0	F I FCS F
No FCS Field NON-AUTO Mode	1	0	0	F A C I F
No FCS Field AUTO Mode	1	0	1	F A C I F
No FCS Field No Control Field Non-AUTO Mode	1	1	1	F A I F
No FCS Field No Control Field No Address Field NON-AUTO Mode	1	1	0	F I F

Key to Abbreviations:

F = Flag (01111110)
A = Address Field
C = Control Field
I = Information Field
FCS = Frame Check Sequence

Note: The AM bit is AUTO mode control bit when NB = 0, and Address Mode control bit when NB = 1.

Figure 13-5. Frame Format Options

13.5.4 No FCS Field

In the normal case (NFCS=0), the last 16 bits before the closing flag are the Frame Check Sequence (FCS) field. These bits are not stored in the 8044's RAM. Rather, they are used to compute a cyclic redundancy check (CRC) on the data in the rest of the frame. A received frame with a CRC error (incorrect FCS) is ignored. In transmission, the FCS field is automatically computed by the SIU, and placed in the transmitted frame just prior to the closing flag.

The NFCS bit (SMD Bit 0) gives the user the capability of overriding this automatic feature. When this bit is set (NFCS=1), all bits from the beginning of the information field to the beginning of the closing flag are treated as part of the information field, and are stored in the on-chip RAM. No FCS checking is done on the received frames, and no FCS is generated for the transmitted frames. The No FCS Field option may be used in conjunction with any of the other options. It is typically used in NON-AUTO mode, although it does not strictly exclude AUTO mode. Use of the No FCS Field option in

AUTO Mode may, however, result in SDLC protocol violations, since the data integrity is not checked by the SIU.

Formats without an FCS field have the following applications:

- Receiving and transmitting frames without verifying data integrity
- Using an alternate data verification algorithm
- Using an alternate CRC-16 polynomial (such as $X^{16} + X^{15} + X^2 + 1$), or a 32-bit CRC
- Performing data link diagnosis by forcing false CRCs to test error detection mechanisms

In addition to the applications mentioned above, all of the format variations are useful in the support of non-standard bit-synchronous protocols.

13.6 HDLC

In addition to its support of SDLC communications, the 8044 also supports some of the capabilities of HDLC. The following remarks indicate the principal differences between SDLC and HDLC.

- HDLC permits any number of bits in the information field, whereas SDLC requires a byte structure (multiple of 8 bits). The 8044 itself operates on byte boundaries, and thus it restricts fields to multiples of 8 bits.
- HDLC provides functional extensions to SDLC: an unlimited address field is allowed, and extended frame number sequencing.
- HDLC does not support operation in loop configurations.

13.7 SIU SPECIAL FUNCTION REGISTERS

The 8044 CPU communicates with and controls the SIU through hardware registers. These registers are accessed using direct addressing. The SIU special function registers (SIU SFRs) are of three types:

- Control and Status Registers
- Parameter Registers
- ICE Support Registers

13.7.1 Control and Status Registers

There are three SIU Control and Status Registers:

- Serial Mode Register (SMD)
- Status/Command Register (STS)
- Send/Receive Count Register (NSNR)

The SMD, STS, and NSNR registers are all cleared by system reset. This assures that the SIU will power up in an idle state (neither receiving nor transmitting).

These registers and their bit assignments are described below (see also the More Details on Registers section).

SMD: Serial Mode Register (byte-addressable)

Bit:	7	6	5	4	3	2	1	0
	SCM2	SCM1	SCM0	NRZI	LOOP	PFS	NB	NFCS

The Serial Mode Register (Address C9H) selects the operational modes of the SIU. The 8044 CPU can both read and write SMD. The SIU can read SMD but cannot write to it. To prevent conflict between CPU and SIU access to SMD, the CPU should write SMD only when the Request To Send (RTS) and Receive Buffer Empty (RBE) bits (in the STS register) are both false (0). Normally, SMD is accessed only during initialization.

The individual bits of the Serial Mode Register are as follows:

Bit #	Name	Description
SMD.0	NFCS	No FCS field in the SDLC frame.
SMD.1	NB	Non-Buffered mode. No control field in the SDLC frame.
SMD.2	PFS	Pre-Frame Sync mode. In this mode, the 8044 transmits two bytes before the first flag of a frame, for DPLL synchronization. If NRZI is enabled, 00H is sent; otherwise, 55H is sent. In either case, 16 pre-frame transitions are guaranteed.
SMD.3	LOOP	Loop configuration.
SMD.4	NRZI	NRZI coding option.
SMD.5	SCM0	Select Clock Mode — Bit 0
SMD.6	SCM1	Select Clock Mode — Bit 1
SMD.7	SCM2	Select Clock Mode — Bit 2

The SCM bits decode as follows:

SCM 2	SCM 1	SCM 0	Clock Mode	Data Rate (Bits/sec)*
0	0	0	Externally clocked	0-2.4M**
0	0	1	Undefined	
0	1	0	Self clocked, timer overflow	244-62.5K
0	1	1	Undefined	
1	0	0	Self clocked, external 16x	0-375K

SCM 2	1	0	Clock Mode	Data Rate (Bits/sec)*
1	0	1	Self clocked, external 32x	0-187.5K
1	1	0	Self clocked, internal fixed	375K
1	1	1	Self clocked, internal fixed	187.5k

*Based on a 12 Mhz crystal frequency
**0-1M bps in loop configuration

STS: Status/Command Register (bit-addressable)

Bit:	7	6	5	4	3	2	1	0
	TBF	RBE	RTS	SI	BOV	OPB	AM	RBP

The Status/Command Register (Address C8H) provides operational control of the SIU by the 8044 CPU, and enables the SIU to post status information for the CPU's access. The SIU can read STS, and can alter certain bits, as indicated below. The CPU can both read and write STS asynchronously. However, 2-cycle instructions that access STS during both cycles ('JBC/B, REL' and 'MOV /B,C.') should not be used, since the SIU may write to STS between the two CPU accesses.

The individual bits of the Status/Command Register are as follows:

Bit #	Name	Description
STS.0	RBP	Receive Buffer Protect. Inhibits writing of data into the receive buffer. In AUTO mode, RBP forces an RNR response instead of an RR.
STS.1	AM	AUTO Mode/Addressed Mode. Selects AUTO mode where AUTO mode is allowed. If NB is true, (=1), the AM bit selects the addressed mode. AM may be cleared by the SIU.
STS.2	OPB	Optional Poll Bit. Determines whether the SIU will generate an AUTO response to an optional poll (UP with P=0). OPB may be set or cleared by the SIU.
STS.3	BOV	Receive Buffer Overrun. BOV may be set or cleared by the SIU.
STS.4	SI	SIU Interrupt. This is one of the five interrupt sources to the CPU. The vector location = 23H. SI may be set by the SIU. It should be cleared by the CPU before returning from an interrupt routine.
STS.5	RTS	Request To Send. Indicates that the 8044 is ready to transmit or is transmitting. RTS may be read or written by the CPU. RTS may be read by the SIU, and in AUTO mode may be written by the SIU.
STS.6	RBE	Receive Buffer Empty. RBE can be thought of as Receive Enable. RBE is set to one by the CPU when it is ready to receive a frame, or has just read the buffer, and to zero by the SIU when a frame has been received.
STS.7	TBF	Transmit Buffer Full. Written by the CPU to indicate that it has filled the transmit buffer. TBF may be cleared by the SIU.

NSNR: Send/Receive Count Register (bit-addressable)

Bit:	7	6	5	4	3	2	1	0
	NS2	NS1	NS0	SES	NR2	NR1	NR0	SER

The Send/Receive Count Register (Address D8H) contains the transmit and receive sequence numbers, plus tally error indications. The SIU can both read and write NSNR. The 8044 CPU can both read and write NSNR asynchronously. However, 2-cycle instructions that access NSNR during both cycles ('JBC /B, REL', and 'MOV /B,C') should not be used, since the SIU may write to NSNR between the two 8044 CPU accesses.

The individual bits of the Send/Receive Count Register are as follows:

Bit #	Name	Description
NSNR.0	SER	Receive Sequence Error: NS (P) ≠ NR (S)
NSNR.1	NR0	Receive Sequence Counter—Bit 0
NSNR.2	NR1	Receive Sequence Counter—Bit 1
NSNR.3	NR2	Receive Sequence Counter—Bit 2
NSNR.4	SES	Send Sequence Error: NP (P) ≠ NS (S) and NP (P) ≠ NS (S) + 1
NSNR.5	NS0	Send Sequence Counter — Bit 0
NSNR.6	NS1	Send Sequence Counter — Bit 1
NSNR.7	NS2	Send Sequence Counter — Bit 2

13.7.2 Parameter Registers

There are eight parameter registers that are used in connection with SIU operation. All eight registers may be read or written by the 8044 CPU. RFL and RCB are normally loaded by the SIU.

The eight parameter registers are as follows:

STAD: Station Address Register (byte-addressable)

The Station Address register (Address CEH) contains the station address. To prevent access conflict, the CPU should access STAD only when the SIU is idle (RTS=0 and RBE=0). Normally, STAD is accessed only during initialization.

TBS: Transmit Buffer Start Address Register (byte-addressable)

The Transmit Buffer Start address register (Address DCH) points to the location in on-chip RAM for the beginning of the I-field of the frame to be transmitted. The CPU should access TBS only when the SIU is not transmitting a frame (when TBF=0).

TBL: Transmit Buffer Length Register (byte-addressable)

The Transmit Buffer Length register (Address DBH) contains the length (in bytes) of the I-field to be transmitted. A blank I-field (TBL=0) is valid. The CPU should access TBL only when the SIU is not transmitting a frame (when TBF=0).

NOTE: The transmit and recieve buffers are not allowed to "wrap around" in the on-chip RAM. A "buffer end" is automatically generated if address 191 (BFH) is reached.

TCB: Transmit Control Byte Register (byte-addressable)

The Transmit Control Byte register (Address DAH) contains the byte which is to be placed in the control field of the transmitted frame, during NON-AUTO mode transmission. The CPU should access TCB only when the SIU is not transmitting a frame (when TBF=0). The N_S and N_R counters are not used in the NON-AUTO mode.

RBS: Receive Buffer Start Address Register (byte-addressable)

The Receive Buffer Start address register (Address CCH) points to the location in on-chip RAM where the beginning of the I-field of the frame being received is to be stored. The CPU should write RBS only when the SIU is not receiving a frame (when RBE=0).

RBL: Receive Buffer Length Register (byte-addressable)

The Receive Buffer Length register (Address CBH) contains the length (in bytes) of the area in on-chip RAM allocated for the received I-field. RBL=0 is valid. The CPU should write RBL only when RBE=0.

RFL: Receive Field Length Register (byte-addressable)

The Received Field Length register (Address CDH) contains the length (in bytes) of the received I-field that has just been loaded into on-chip RAM. RFL is loaded by the SIU. RFL=0 is valid. RFL should be accesssed by the CPU only when RBE=0.

RCB: Receive Control Byte Register (byte-addressable)

The Received Control Byte register (Address CAH) contains the control field of the frame that has just been received. RCB is loaded by the SIU. The CPU can only read RCB, and should only access RCB when RBE=0.

13.7.3 ICE Support Registers

The 8044 In-Circuit Emulator (ICE-44) allows the user to exercise the 8044 application system and monitor the execution of instructions in real time.

The emulator operates with Intel's Intellec® development system. The development system interfaces with the user's 8044 system through an in-cable buffer box. The cable terminates in a 8044 pin-compatible plug, which fits into the 8044 socket in the user's system. With the emulator plug in place, the user can excercise his system in real time while collecting up to 255 instruction cycles of real-time data. In addition, he can single-step the program.

Static RAM is available (in the in-cable buffer box) to emulate the 8044 internal and external program memory and external data memory. The designer can display and alter the contents of the replacement memory in the buffer box, the internal data memory, and the internal 8044 registers, including the SFRs.

Among the SIU SFRs are the following registers that support the operation of the ICE:

DMA CNT: DMA Count Register (byte-addressable)

The DMA Count register (Address CFH) indicates the number of bytes remaining in the information block that is currently being used.

FIFO: Three-Byte (byte-addressable)

The Three-Byte FIFO (Address DDH, DEH, and DFH) is used between the eight-bit shift register and the information buffer when an information block is received.

SIUST: SIU State Counter (byte-addressable)

The SIU State Counter (Address D9H) reflects the state of the internal logic which is under SIU control. Therefore, care must be taken not to write into this register.

The SIUST register can serve as a helpful aid to determine which field of a receive frame that the SIU expects next. The table below will help in debugging 8044 reception problems.

SIUST VALUE	FUNCTION
01H	Waiting for opening flag.
08H	Waiting for address field.
10H	Waiting for control field.
18H	Waiting for first byte of I field. This state is only entered if a FCS is expected. It pushes the received byte onto the top of the FIFO.
20H	Wating for second byte of I field. This state always follows state 18H.
28H	Waiting for I field byte. This sate can be entered from state 20H or from states 01H, 08H, or 10H depending upon the SIU's mode configuration. (Each time a byte is received, it is pushed onto the top of the FIFO and the byte at the bottom is put into memory. For no FCS formatted frames, the FIFO is collapsed into a single register).
30H	Waiting for the closing flag after having overflowed the receive buffer. Note that even if the receive frame overflows the assigned receive buffer length, the FCS is still checked.

Examples of SIUST status sequences for different frame formats are shown below. Note that status changes after acceptance of the received field byte.

13.8 OPERATION

The SIU is initialized by a reset signal (on pin 9), followed by write operations to the SIU SFRs. Once initialized, the SIU can function in AUTO mode or NON-AUTO mode. Details are given below.

Table 13-1. SIUST Status Sequences

									Frame Option NFCS	NB	AM
Example 1:											
Frame Format	(Idle)	F	A	C	I			FCS	F	0	0	1
SIUST Value	01	01	08	10	18	20	28	28	01			
Example 2:											
Frame Format	(Idle)	F	A	I				FCS	F	0	1	1
SIUST Value	01	01	08	18	20	28		28	01			
Example 3:											
Frame Format	(Idle)	F	I					FCS	F	0	1	0
SIUST Value	01	01	18	20	28			28	01			
Example 4:											
Frame Format	(Idle)	F	A	I	F					1	1	1
SIUST Value	01	01	08	28	01							
Example 5:											
Frame Format	(Idle)	F	I		F					1	1	0
SIUST Value	01	01	28		01							
Example 6:											
Frame Format	(Idle)	F	I				I OVERFLOW	FCS	F	0	1	0
SIUST Value	01	01	18	20	28		30	30	01			

13.8.1 Initialization

Figure 13-6 is the SIU. Registers SMD, STS, and NSNR are cleared by reset. This puts the 8044 into an idle state—neither receiving nor transmitting. The following registers must be initialized before the 8044 leaves the idle state:

STAD—to establish the 8044's SDLC station address.

SMD—to configure the 8044 for the proper operating mode.

RBS, RBL—to define the area in RAM allocated for the Receive Buffer.

TBS, TBL—to define the area in RAM allocated for the Transmit Buffer.

Once these registers have been initialized, the user may write to the STS register to enable the SIU to leave the idle state, and to begin transmits and/or receives.

Setting RBE to 1 enables the SIU for receive. When RBE = 1, the SIU monitors the received data stream for a flag pattern. When a flag pattern is found, the SIU enters Receive mode and receives the frame.

Setting RTS to 1 enables the SIU for transmit. When RTS = 1, the SIU monitors the received data stream for a GA pattern (loop configuration) or waits for a CTS

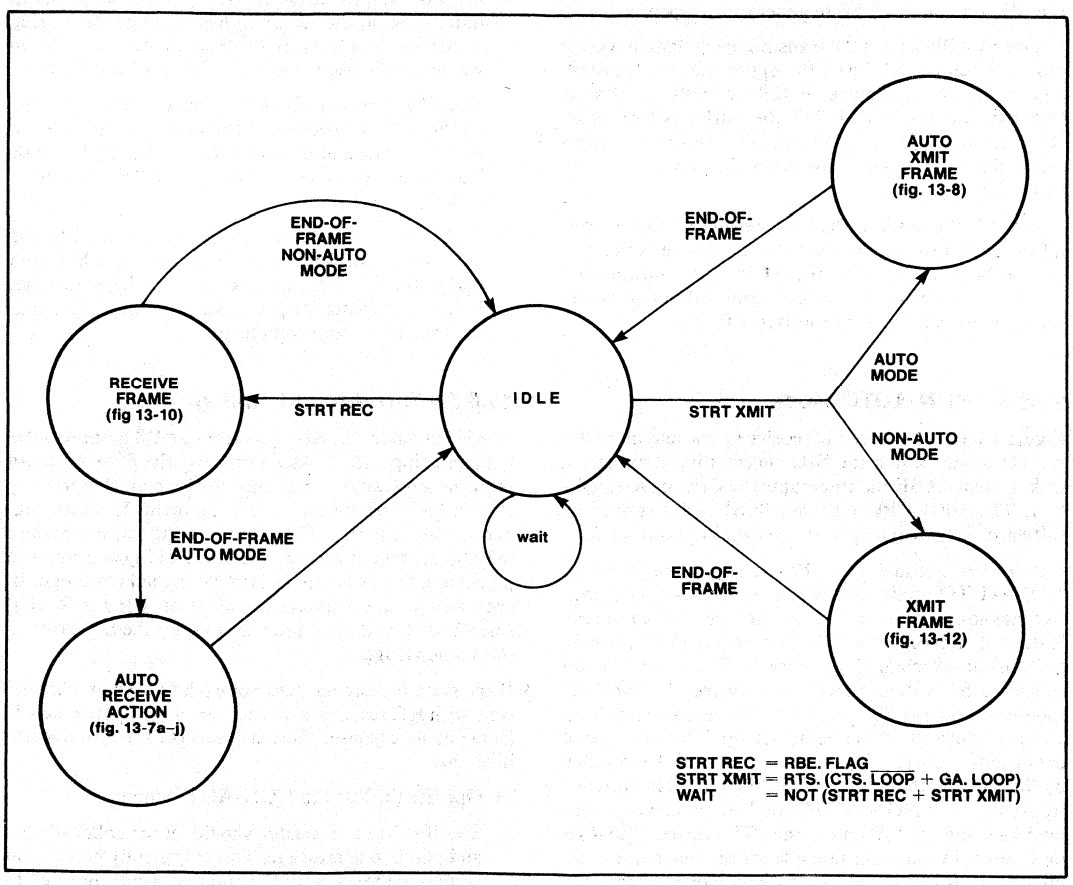


Figure 13-6. SIU State Diagram

(non-loop configuration). When the GA or CTS arrives, the SIU enters Transmit mode and transmits a frame.

In AUTO mode, the SIU sets RTS to enable automatic transmissions of appropriate responses.

13.8.2 AUTO Mode

Figure 13-7 illustrates the receive operations in AUTO mode. The overall operation is shown in Figure 13-7a. Particular cases are illustrated in Figures 13-7b through 13-7j. If any Unnumbered Command other than UP is received, the AM bit is cleared and the SIU responds as if in the NON-AUTO mode, by interrupting the CPU for supervision. This will also happen if a BOV or SES condition occurs. If the received frame contains a poll, the SIU sets the RTS bit to generate a response.

Figure 13-8 illustrates the transmit operations in AUTO mode. When the SIU gets the opportunity to transmit, and if the transmit buffer is full, it sends an I-frame. Otherwise, it sends an RR if the buffer is free, or an RNR if the buffer is protected. The sequence counters NS and NR are used to construct the appropriate control fields.

Figure 13-9 shows how the CPU responds to an SI (serial interrupt) in AUTO mode. The CPU tests the AM bit (in the STS register). If AM = 1, it indicates that the SIU has received either an I-frame, or a positive response to a previously transmitted I-frame.

13.8.3 NON-AUTO Mode

Figure 13-10 illustrates the receive operations in NON-AUTO mode. When the SIU successfully completes a task, it clears RBF and interrupts the CPU by setting SI to 1. The exact CPU response to SI is determined by software. A typical response is shown in Figure 13-11.

Figure 13-12 illustrates the transmit operations in NON-AUTO mode. The SIU does not wait for a positive acknowledge response to the transmitted frame. Rather, it interrupts the CPU (by setting SI to 1) as soon as it finishes transmitting the frame. The exact CPU response to SI is determined by software. A typical response is shown in Figure 13-13. This response results in another transmit frame being set up. The sequence of operations shown in Figure 13-13 can also be initiated by the CPU, without an SI. Thus the CPU can initiate a transmission in NON-AUTO mode without a poll, simply by setting the RTS bit in the STS register. The RTS bit is always used to initiate a transmission, but it is applied to the RTS pin only when a non-loop configuration is used.

13.8.4 8044 Data Link Particulars

The following facts should be noted:

1) In a non-loop configuration, one or two bits are transmitted before the opening flag. This is necessary for NRZI synchronization.

2) In a non-loop configuration, one to eight extra dribble bits are transmitted after the closing flag. These bits are a zero followed by ones.

3) In a loop configuration, when a GA is received and the 8044 begins transmitting, the sequence is 01111110101111110 . . . (FLAG, 1, FLAG, ADDRESS, etc.). The first flag is created from the GA. The second flag begins the message.

4) CTS is sampled after the rising edge of the serial data, at about the center of the bit cell, except during a non-loop, externally clocked mode transmit, in which case it is sampled just after the falling edge.

5) The SIU does not check for illegal I-fields. In particular, if a supervisory command is received in AUTO mode, and if there is also an I-field, it will be loaded into the receive buffer (if RBP=0), but it cannot cause a BOV.

6) In relation to the Receive Buffer Protect facility, the user should set RFL to 0 when clearing RBP, such that, if the SIU is in the process of receiving a frame, RFL will indicate the proper value when reception of the frame has been completed.

13.8.5 Turn Around Timing

In AUTO mode, the SIU generates an RTS immediately upon being polled. Assuming that the 8044 sends an information frame in response to the poll, the primary station sends back an acknowledgement. If, in this acknowledgement, the 8044 is polled again, a response may be generated even before the CPU gets around to processing the interrupt caused by the acknowledge. In such a case, the response would be an RR (or RNR), since TBF would have been set to 0 by the SIU, due to the acknowledge.

If the system designer does not wish to take up channel time with RR responses, but prefers to generate a new I-frame as a response, there are several ways to accomplish this:

1) Operate the 8044 in NON-AUTO mode.

2) Specify that the master should never acknowledge and poll in one message. This is typically how a loop system operates, with the poll operation confined to the UP command. This leaves plenty of time for the

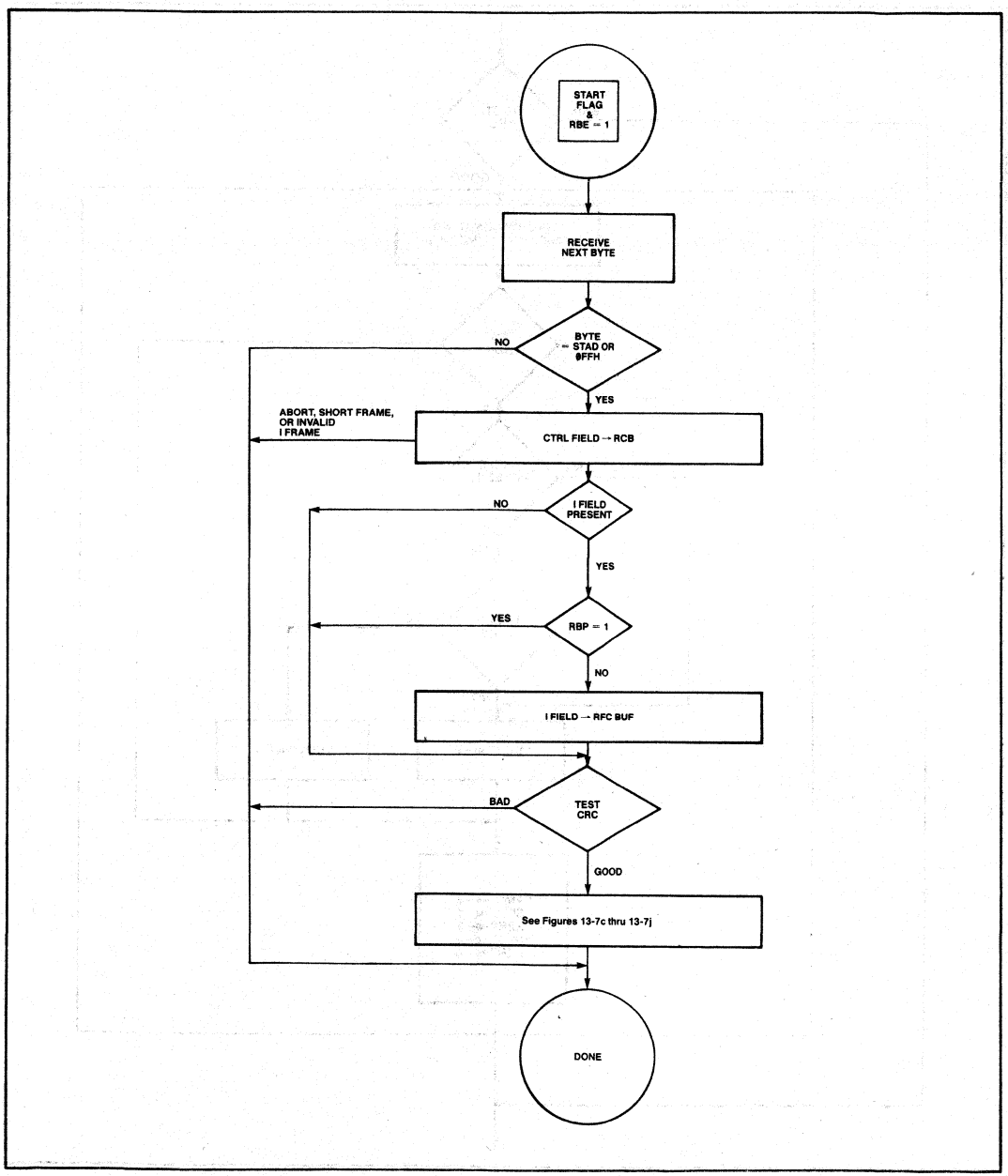


Figure 13-7a. SIU AUTO Mode Receive Flowchart—General

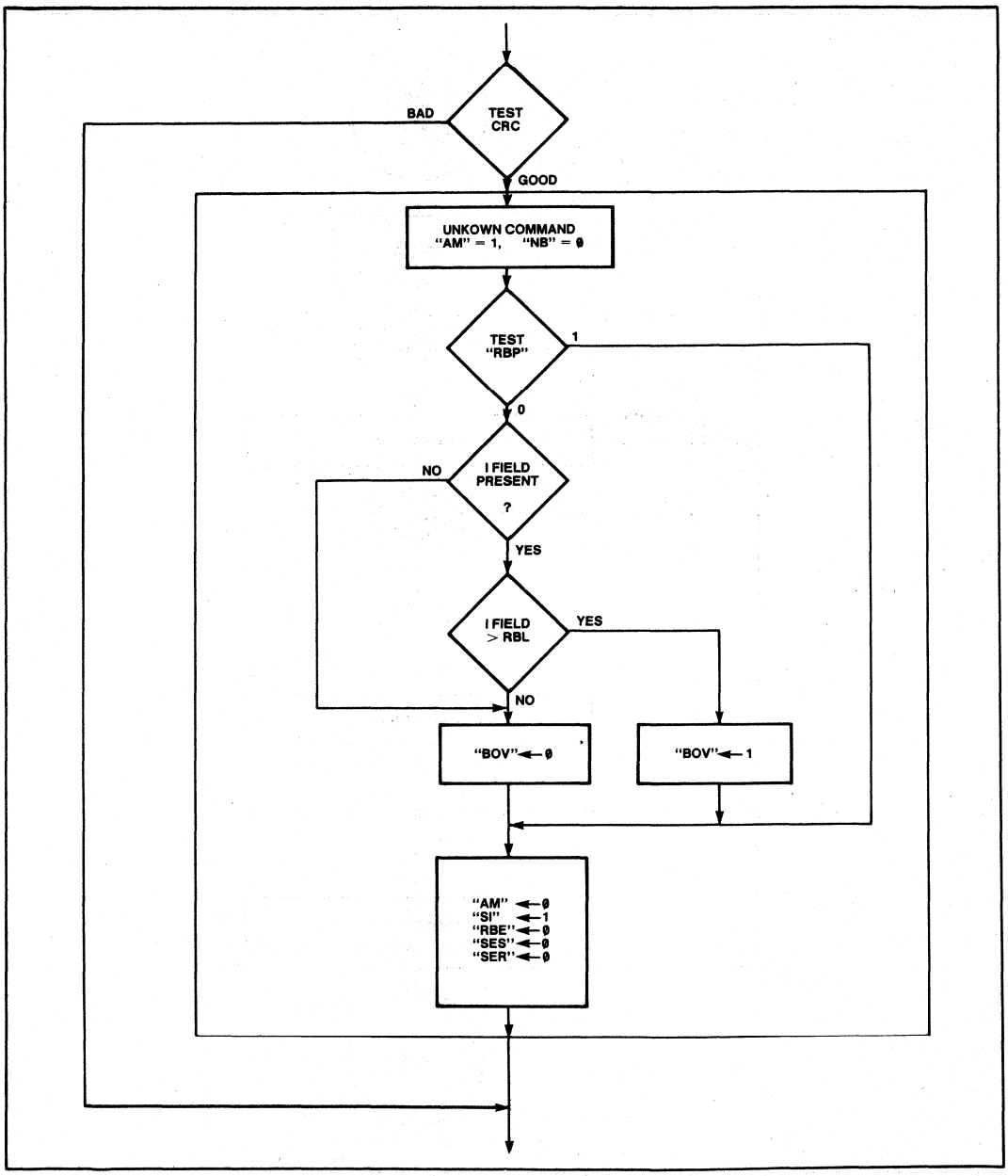


Figure 13-7b. SIU AUTO Mode Receive Flowchart—Unknown Command

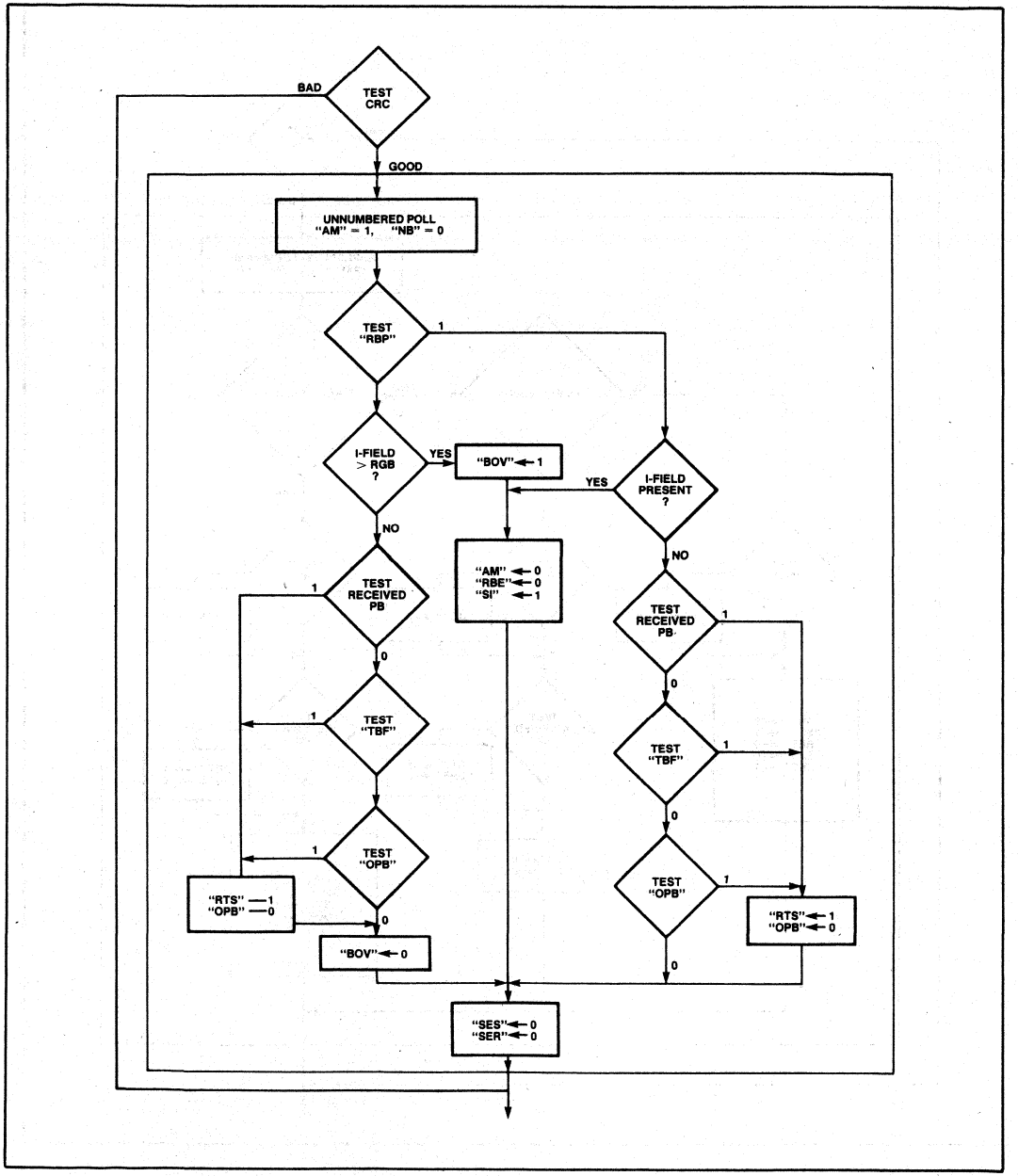


Figure 13-7c. SIU AUTO Mode Receive Flowchart—Unnumbered Poll

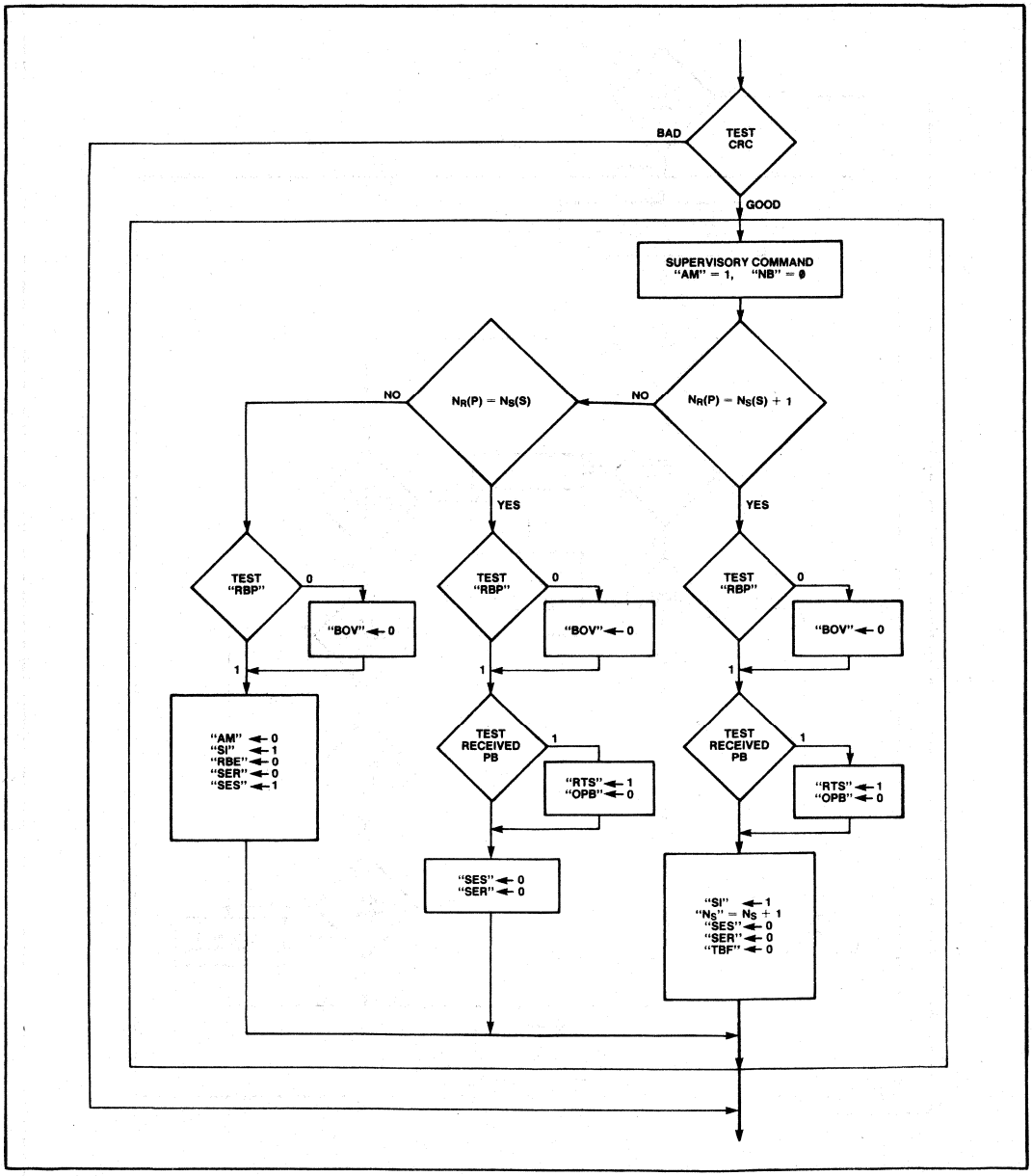


Figure 13-7d. SIU AUTO Mode Receive Flowchart—Supervisory Command

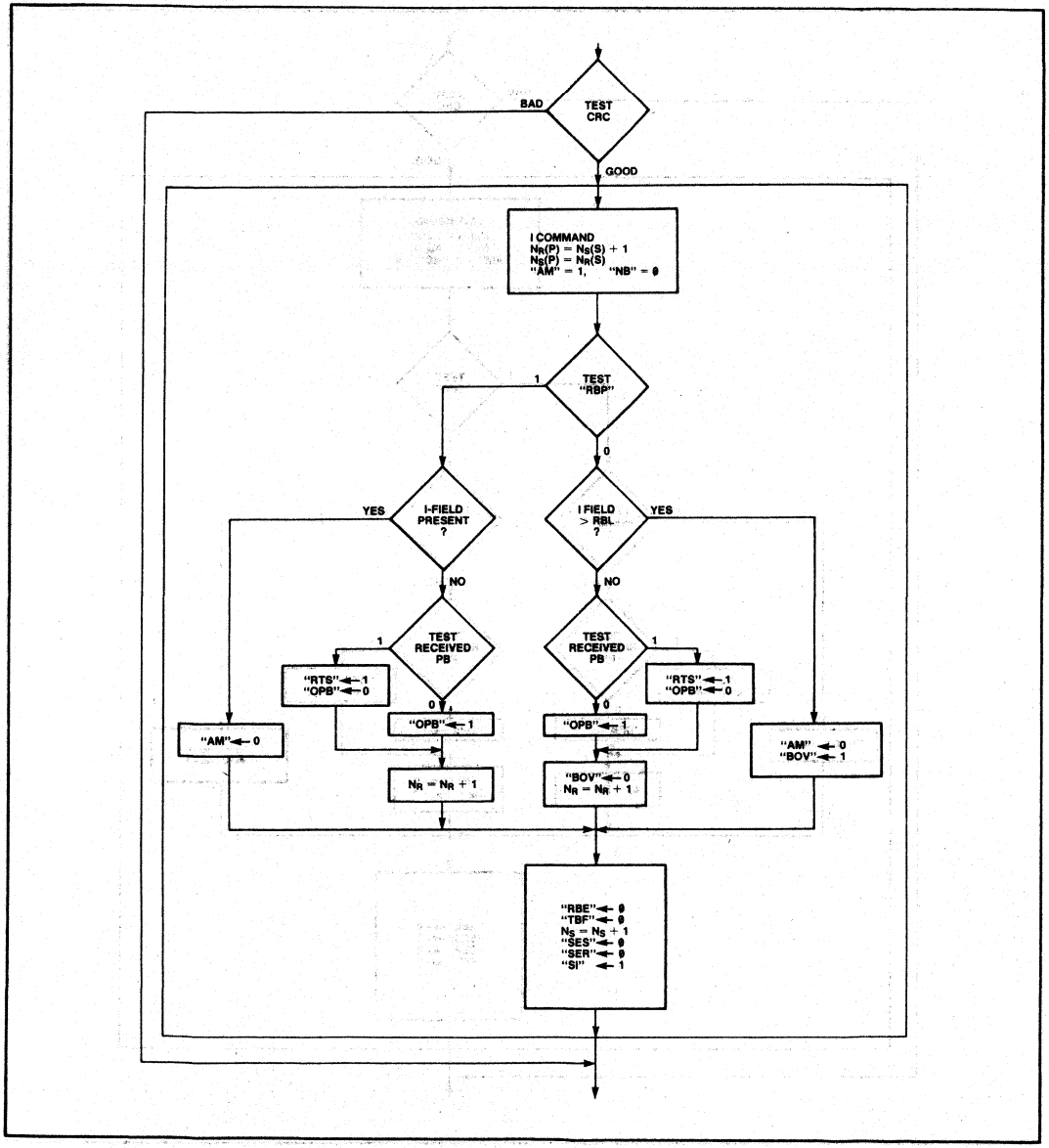


Figure 13-7e. SIU AUTO Mode Receive Flowchart—I Command: Prior Transmitted I-Field Confirmed, Current Received I-Field in Sequence

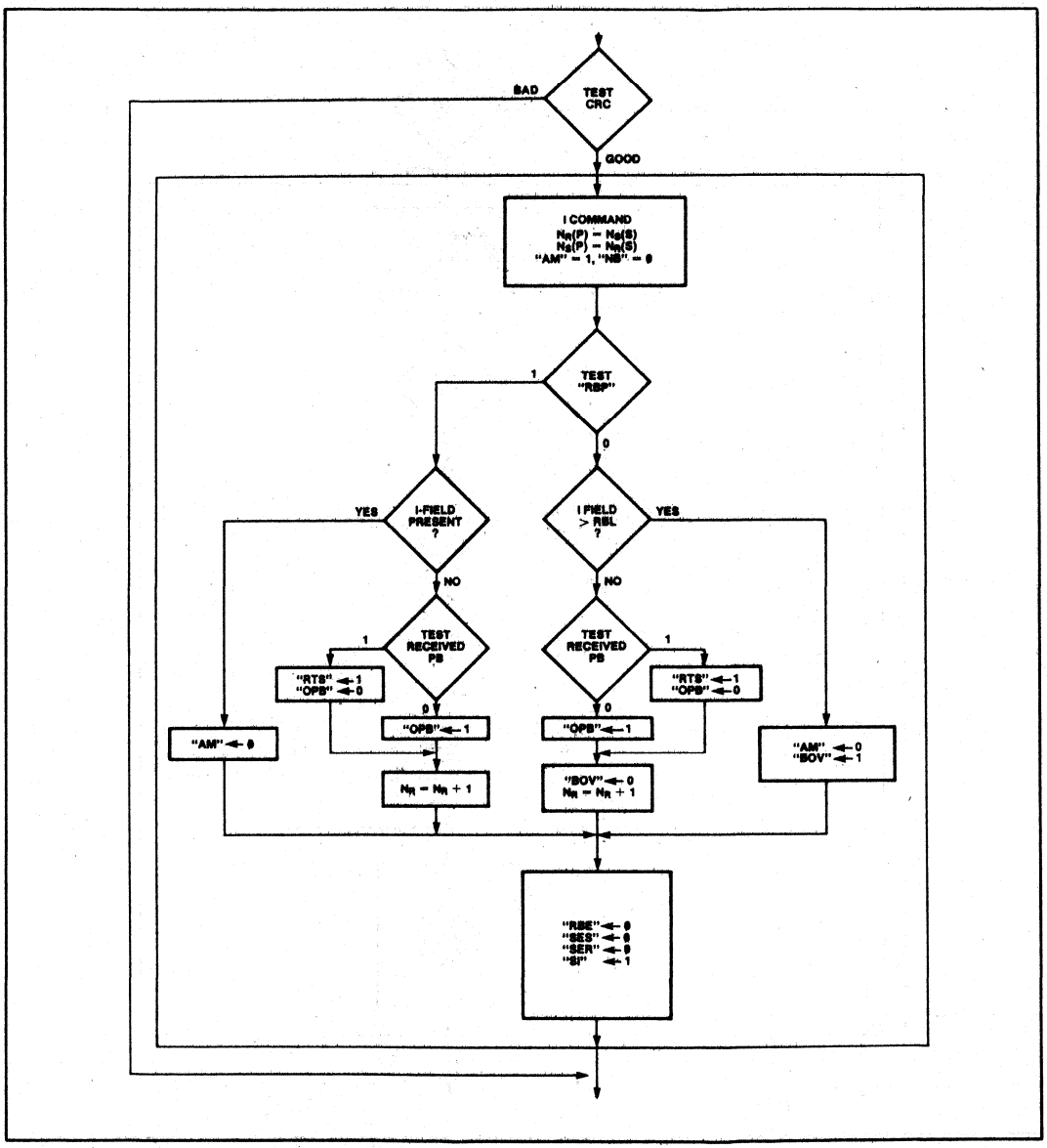


Figure 13-7f. SIU AUTO Mode Receive Flowchart—I Command: Prior Transmitted I-Field Not Confirmed, Current Received I-Field in Sequence

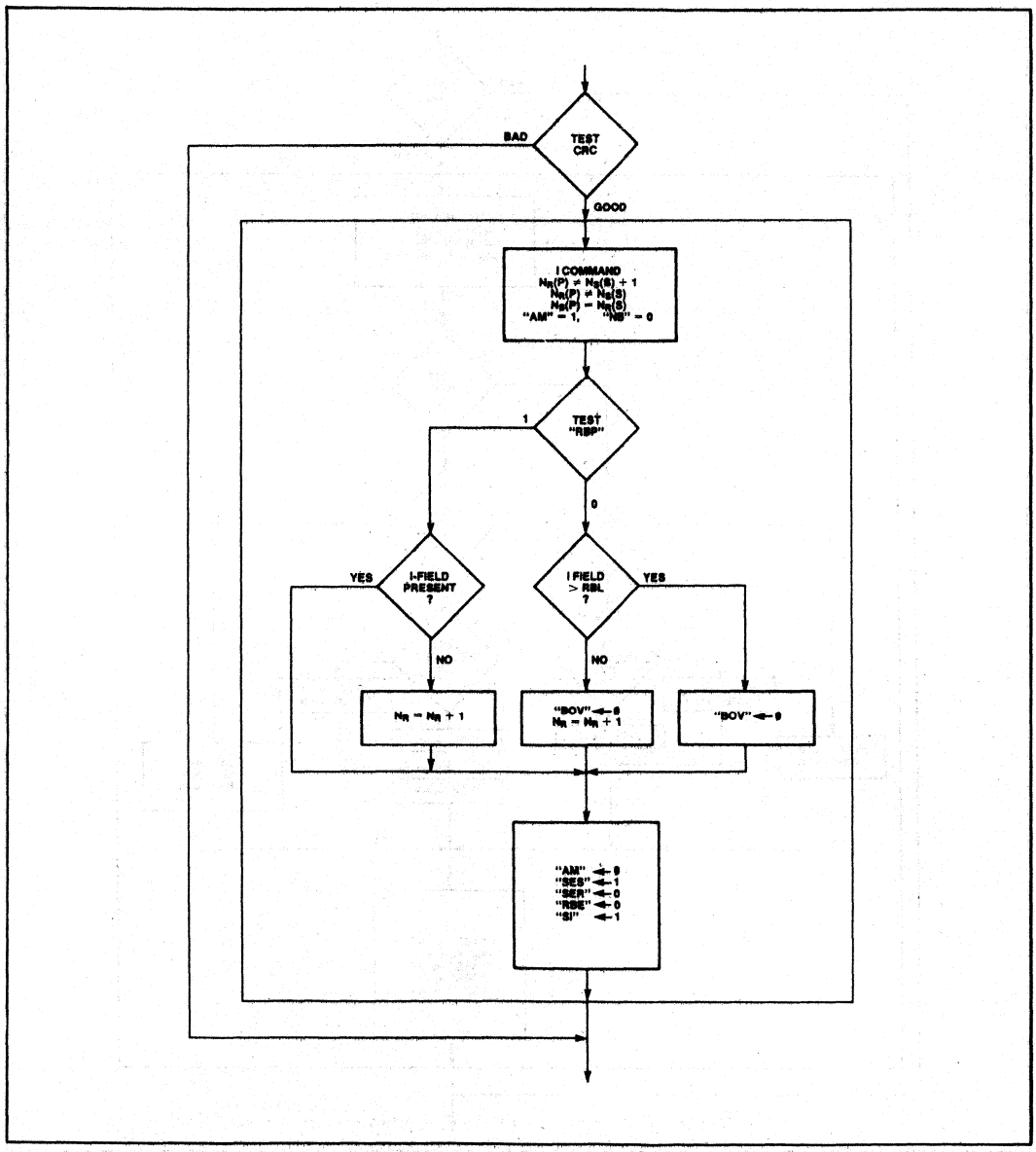


Figure 13-7g. SIU AUTO Mode Receive Flowchart—I Command: Sequence Error Send, Current Received I-Field in Sequence

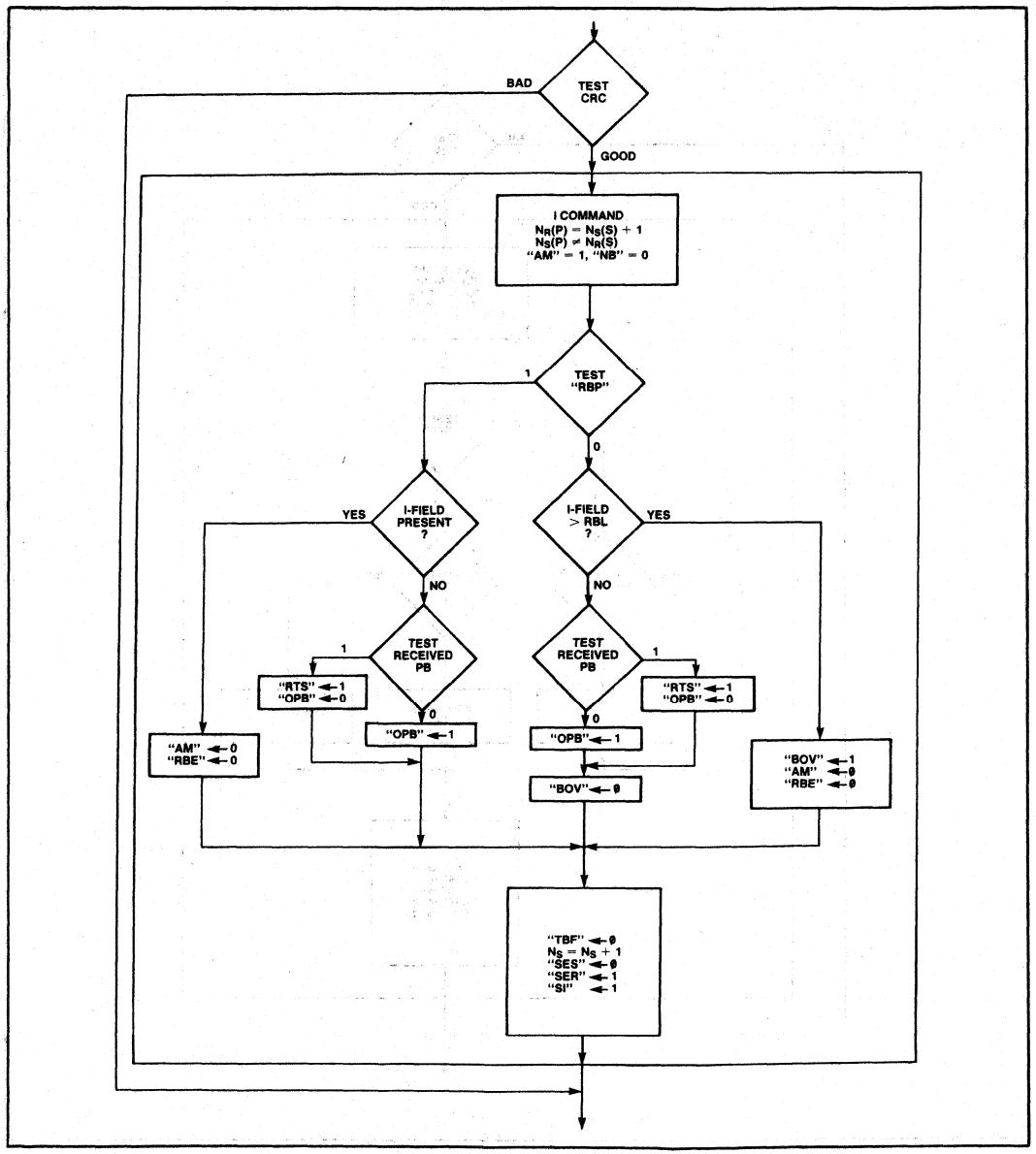


Figure 13-7h. SIU AUTO Mode Receive Flowchart—I Command: Prior Transmitted I-Field Confirmed, Sequence Error Receive

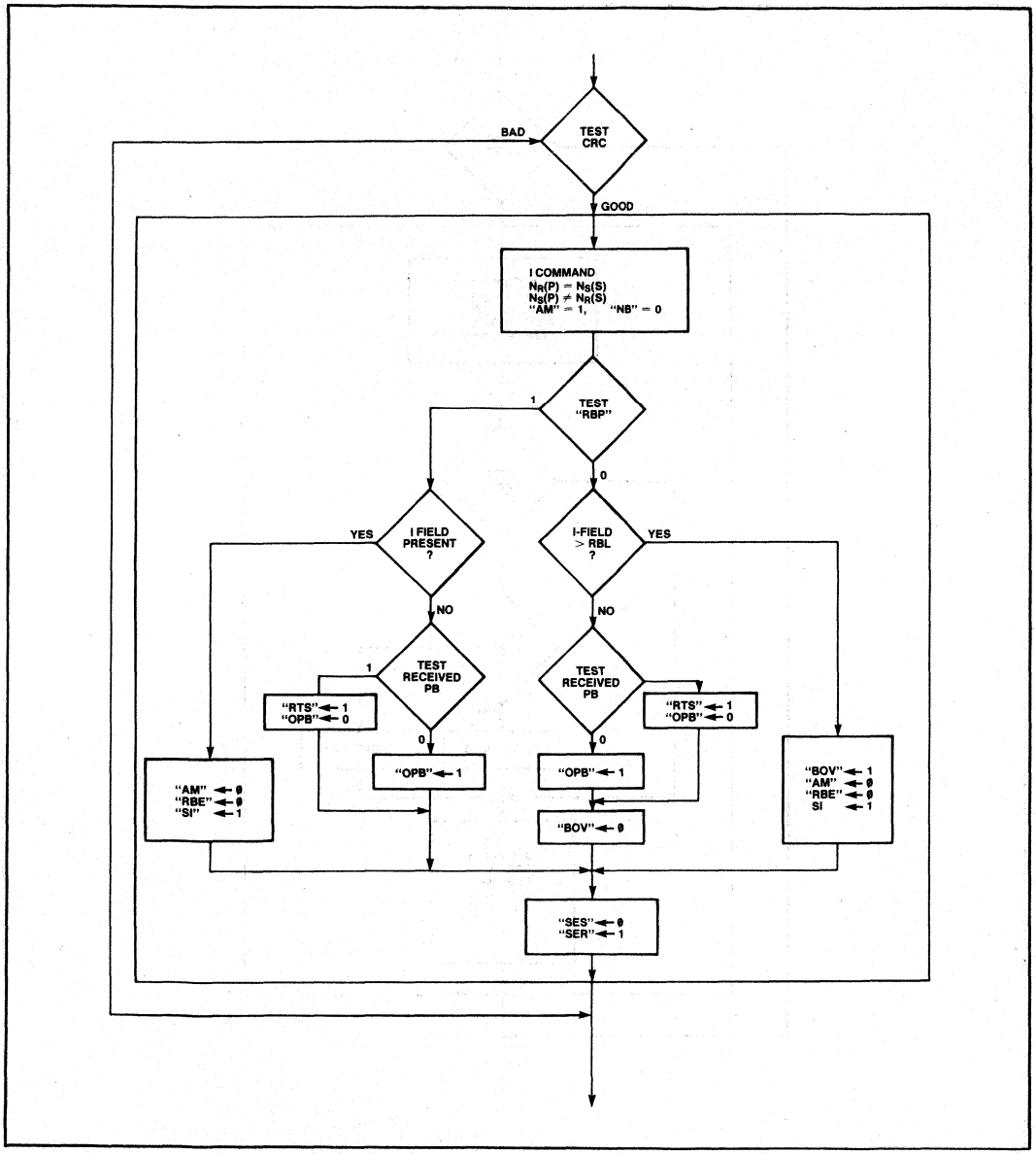


Figure 13-7i. SIU AUTO Mode Receive Flowchart—I Command: Prior Transmitted I-Field Not Confirmed, Sequence Error Receive

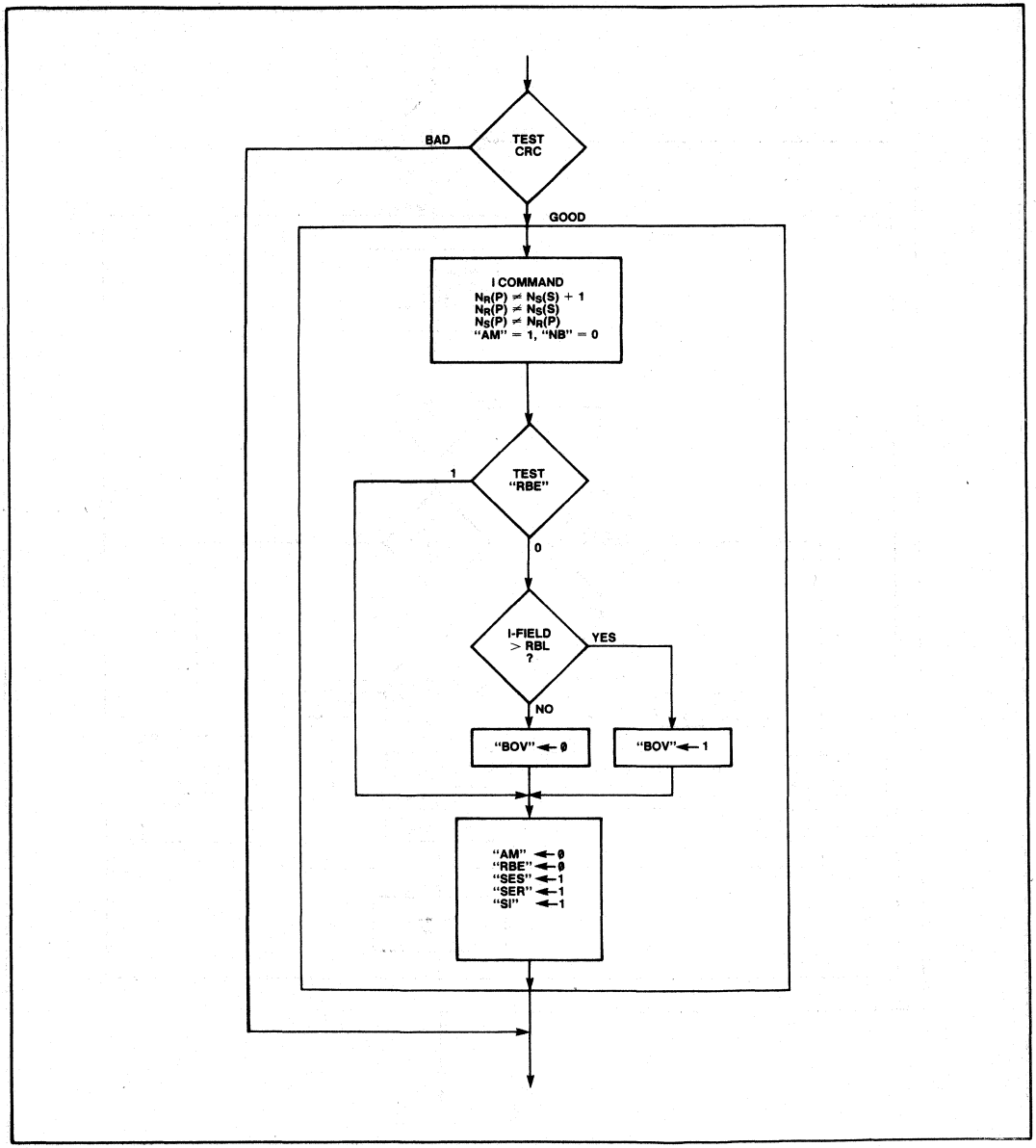


Figure 13-7j. SIU AUTO Mode Receive Flowchart—I Command: Sequence Error Send and Sequence Error Receive

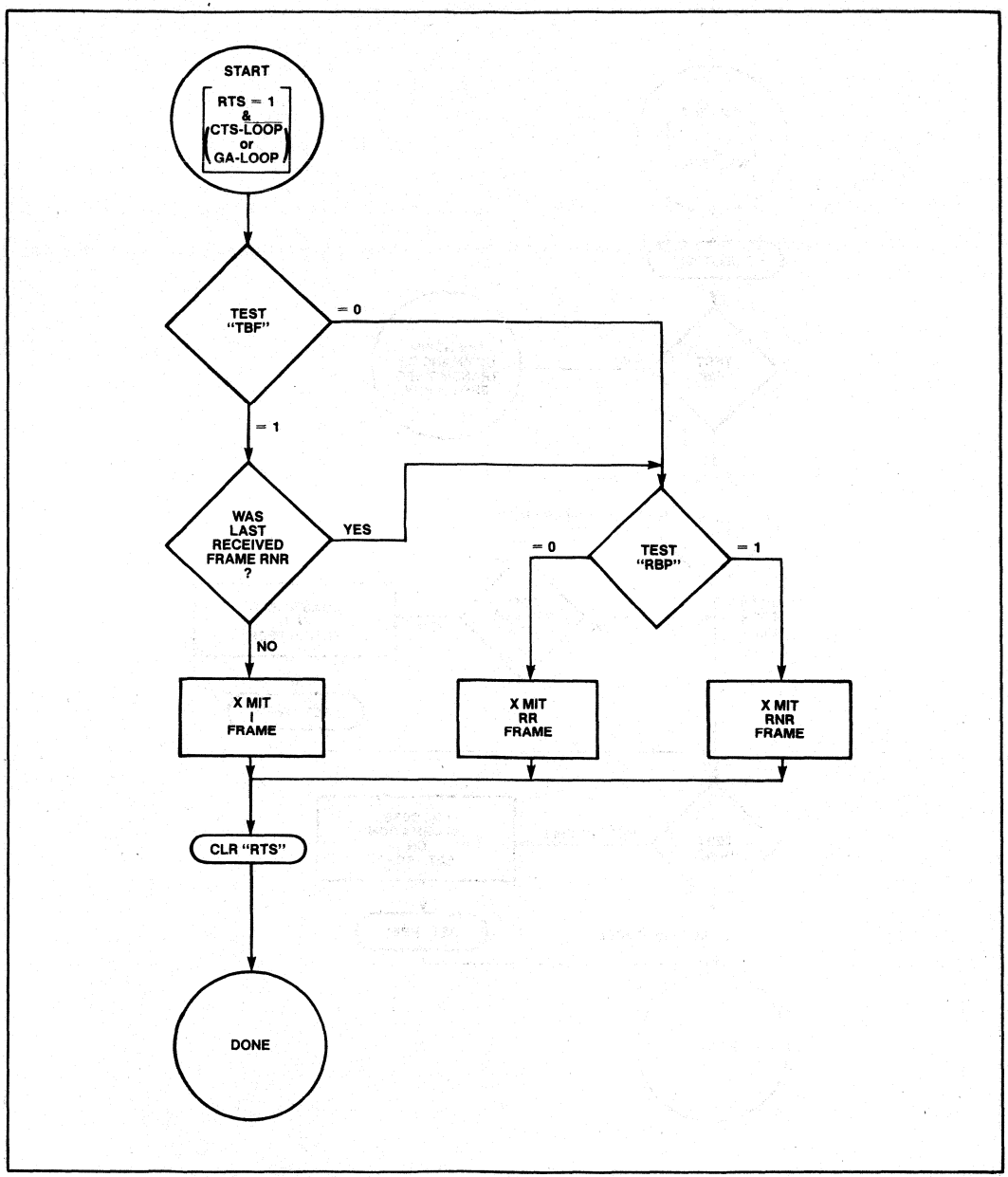


Figure 13-8. SIU AUTO Mode Transmit Flowchart

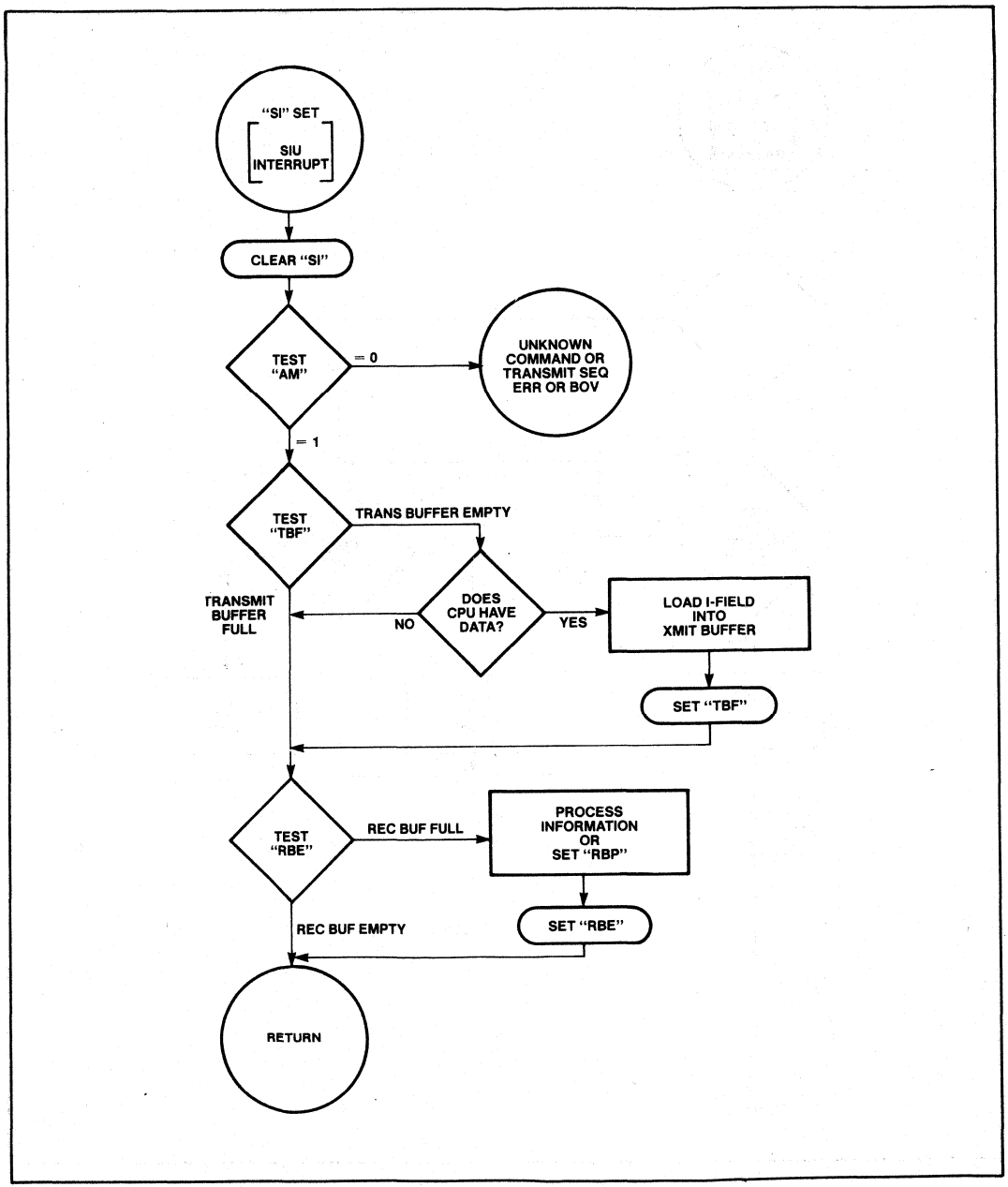


Figure 13-9. AUTO Mode Response to "SI"

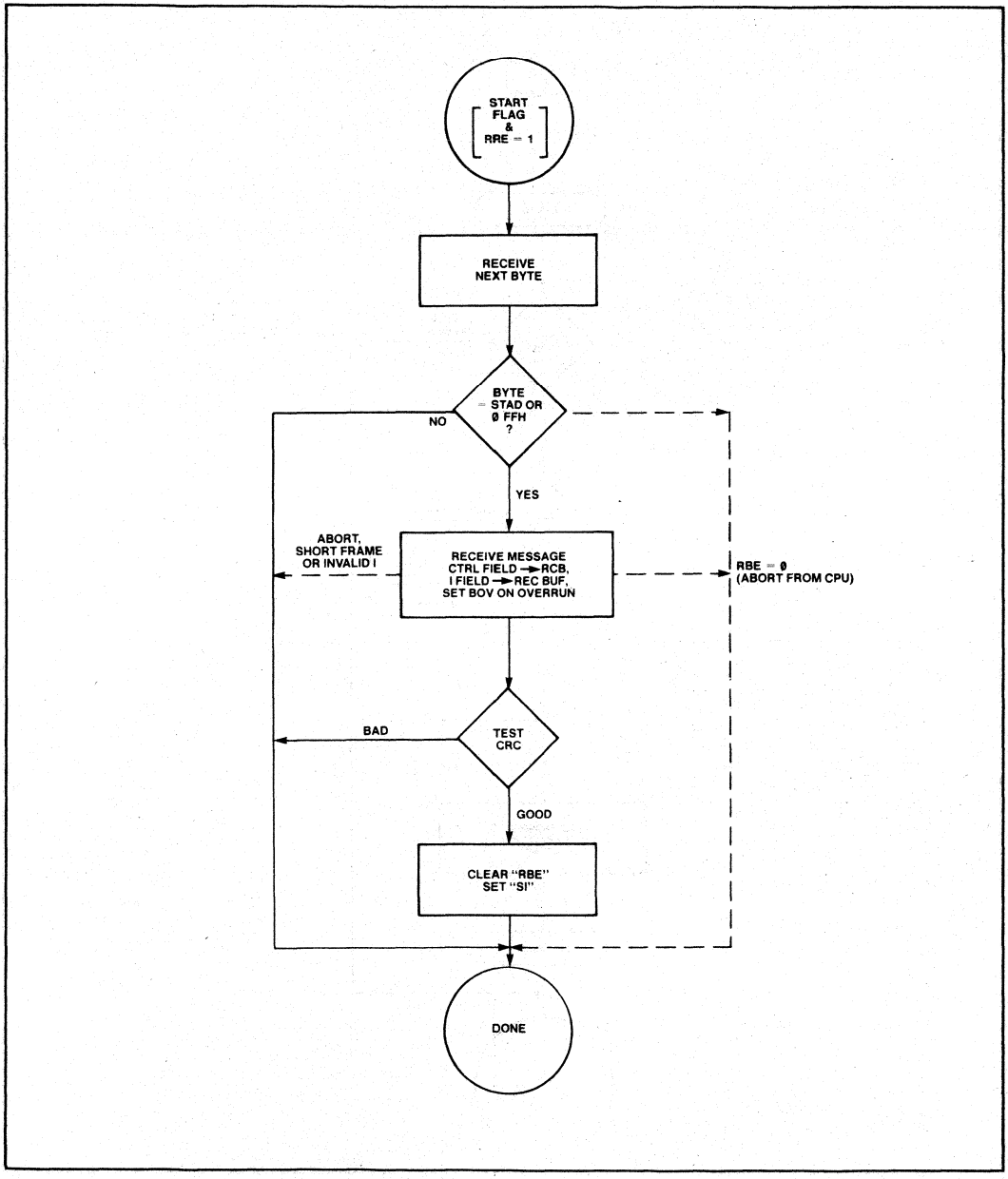


Figure 13-10. SIU NON-AUTO Mode Receive Flowchart

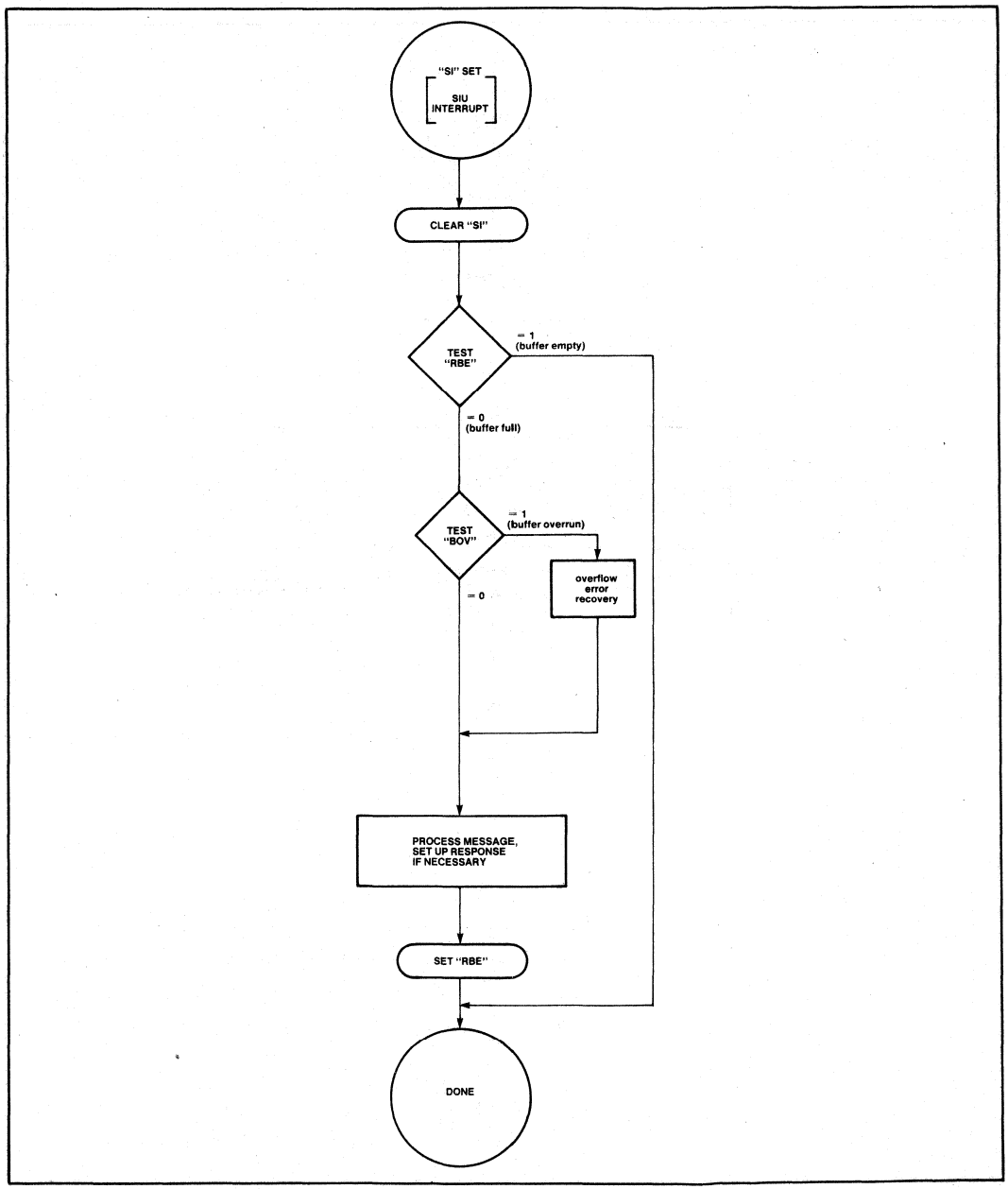


Figure 13-11. NON-AUTO Mode Response to Receive "SI"

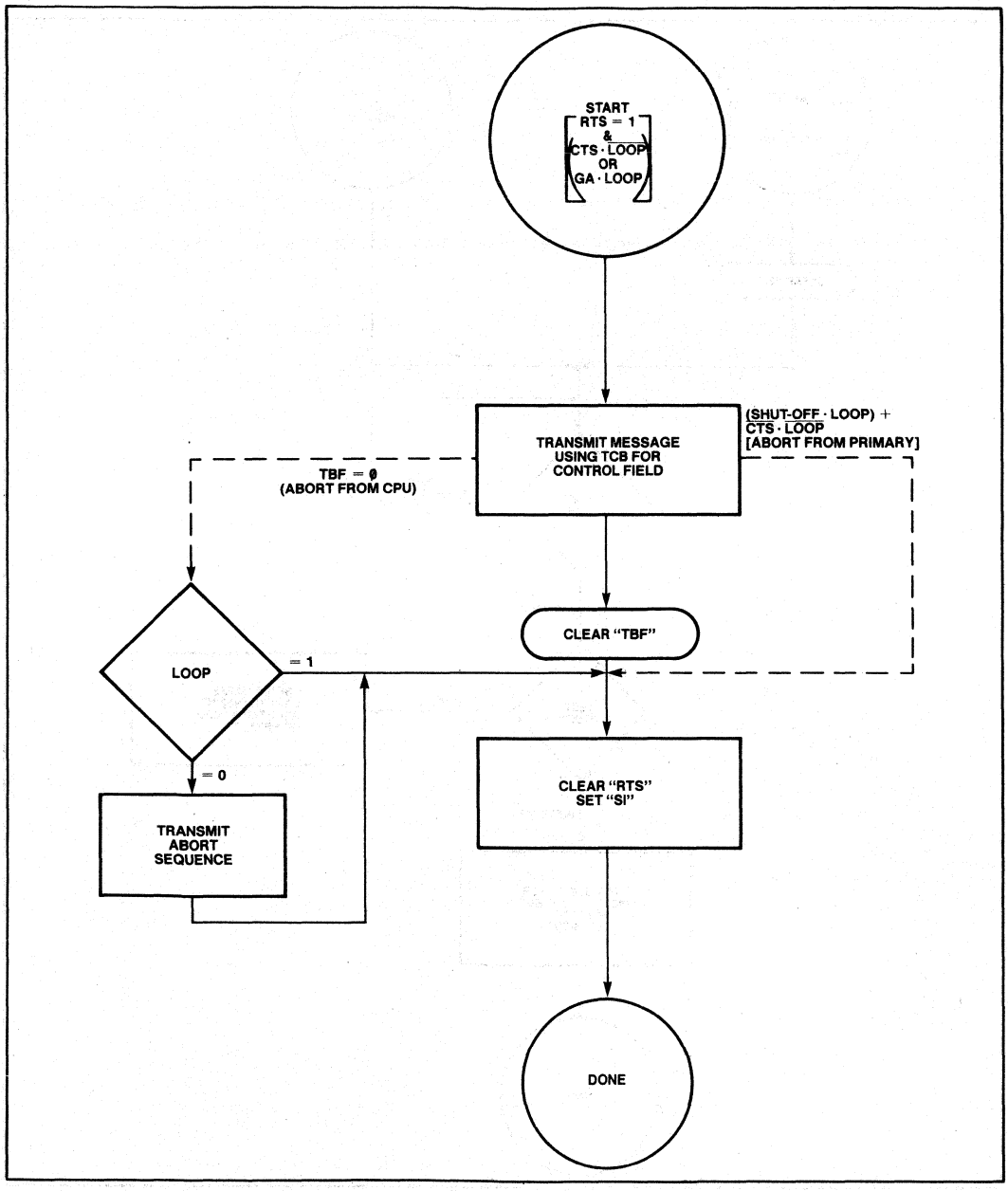


Figure 13-12. SIU NON-AUTO Mode Transmit Flowchart

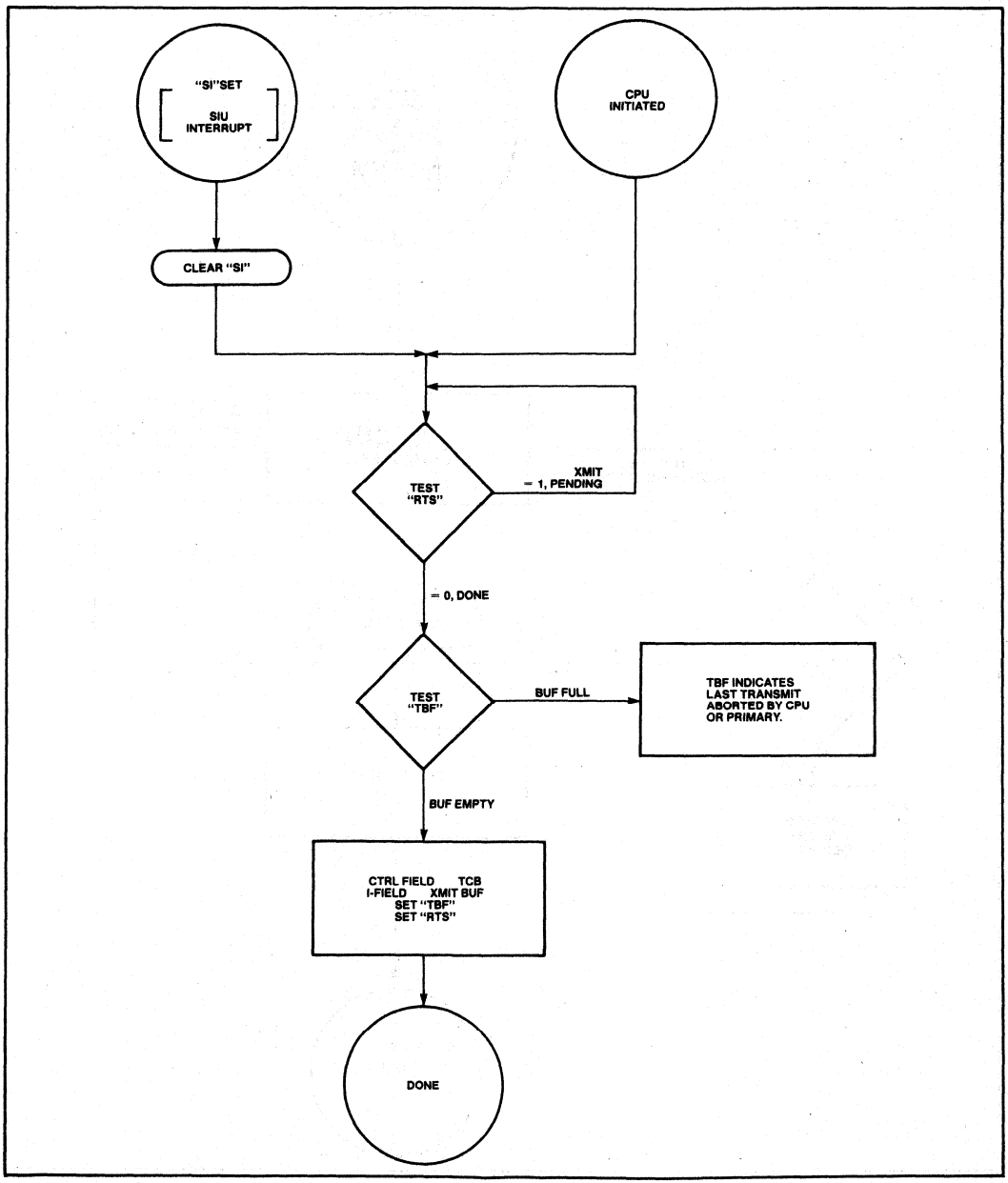


Figure 13-13. NON-AUTO Mode Response to Transmit "SI"

8044 to get its transmit buffer loaded with new information after an acknowledge.

3) The 8044 CPU can clear RTS. This will prevent a response from being sent, or abort it if it is already in progress. A system using external RTS/CTS handshaking could use a one-shot to delay RTS or CTS, thereby giving the CPU more time to disable the response.

13.9 MORE DETAILS ON SIU HARDWARE

The SIU divides functionally into two sections—a bit processor (BIP) and a byte processor (BYP)—sharing some common timing and control logic. As shown in Figure 13-14, the BIP operates between the serial port pins and the SIU bus, and performs all functions necessary to transmit/receive a byte of data to/from the serial data stream. These operations include shifting, NRZI encoding/decoding, zero insertion/deletion, and FCS generation/checking. The BYP manipulates bytes of data to perform message formatting, and other transmitting and receiving functions. It operates between the SIU bus (SIB) and the 8044's internal bus (IB). The interface between the SIU and the CPU involves an interrupt and some locations in on-chip RAM space which are managed by the BYP.

The maximum possible data rate for the serial port is limited to 1/2 the internal clock rate. This limit is imposed by both the maximum rate of DMA to the on-chip RAM, and by the requirements of synchronizing to an external clock. The internal clock rate for an 8044 running on a 12 MHz crystal is 6 MHz. Thus the maximum 8044 serial data rate is 3 MHz. This data rate drops down to 2.4 MHz when time is allowed for external clock synchronization.

13.9.1 The Bit Processor

In the asynchronous (self clocked) modes the clock is extracted from the data stream using the on-chip digital phase-locked-loop (DPLL). The DPLL requires a clock input at 16 times the data rate. This 16× clock may originate from SCLK, Timer 1 Overflow, or PH2 (one half the oscillator frequency). The extra divide-by-two described above allows these sources to be treated alternatively as 32× clocks.

The DPLL is a free-running four-bit counter running off the 16× clock. When a transition is detected in the receive data stream, a count is dropped (by suppressing the carry-in) if the current count value is greater than 8. A count is added (by injecting a carry into the second stage rather than the first) if the count is less than 8. No adjustment is made if the transition occurs at the count of 8. In this manner the counter locks in on the point at which transitions in the data stream occur at the count of 8, and a clock pulse is generated when the count overflows to 0.

In order to perform NRZI decoding, the NRZI decoder compares each bit of input data to the previous bit. There are no clock delays in going through the NRZI decoder.

The zero insert/delete circuitry (ZID) performs zero insertion/deletion, and also detects flags, GA's (Go-Ahead's), and aborts (same as GA's) in the data stream. The pattern 1111110 is detected as an early GA, so that the GA may be turned into a flag for loop mode transmission.

The shut-off detector monitors the receive data stream for a sequence of eight zeros, which is a shut-off command for loop mode transmissions. The shut-off detector is a three-bit counter which is cleared whenever a one is found in the receive data stream. Note that the ZID logic could not be used for this purpose, because the receive data must be monitored even when the ZID is being used for transmission.

As an example of the operation of the bit processor, the following sequence occurs in relation to the receive data:

1) RXD is sampled by SCLK, and then synchronized to the internal processor clock (IPC).

2) If the NRZI mode is selected, the incoming data is NRZI decoded.

3) When receiving other than the flag pattern, the ZID deletes the '0' after 5 consecutive '1's (during transmission this zero is inserted). The ZID locates the byte boundary for the rest of the circuitry. The ZID deletes the '0's by preventing the SR (shift register) from receiving a clocking pulse.

4) The FCS (which is a function of the data between the flags—not including the flags) is initialized and started at the detection of the byte boundary at the end of the opening flag. The FCS is computed each bit boundary until the closing flag is detected. Note that the received FCS has gone through the ZID during transmission.

13.9.2 The Byte Processor

Figure 13-16 is a block diagram of the byte processor (BYP). The BYP contains the registers and controllers necessary to perform the data manipulations associated with SDLC communications. The BYP registers may be read or written by the CPU over the 8044's internal bus

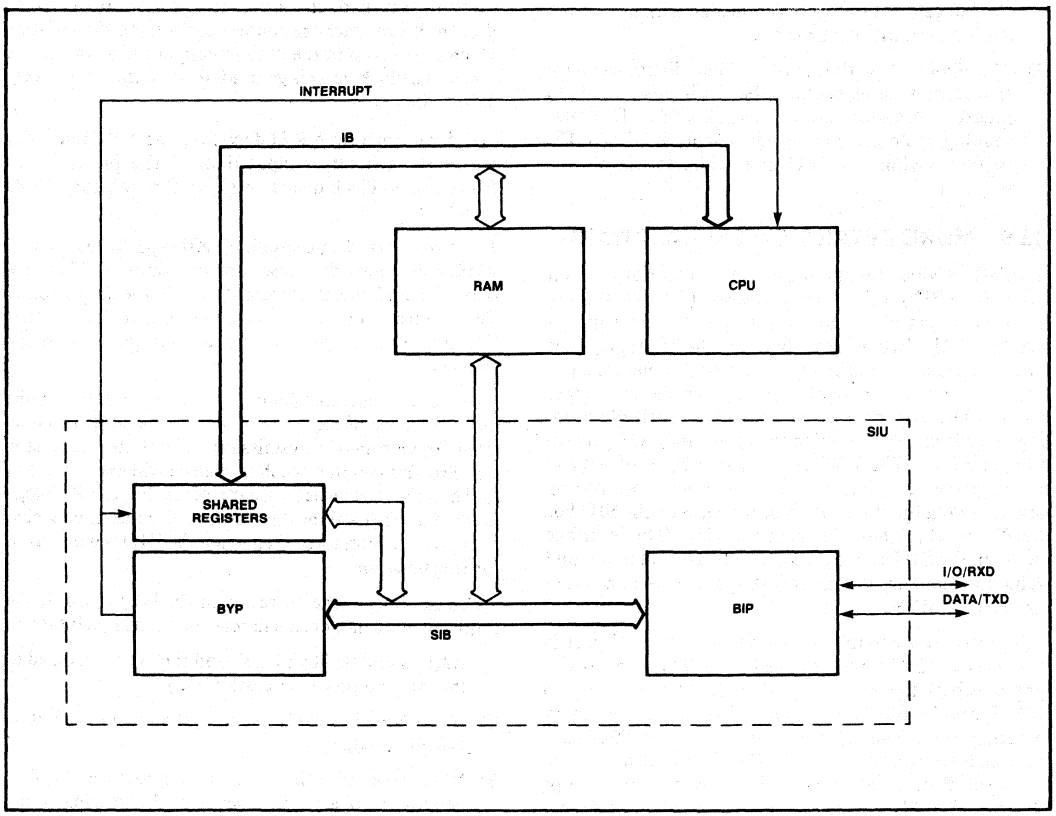


Figure 13-14. The Bit and Byte Processors

(IB), using standard 8044 hardware register operations. The 8044 register select PLA controls these operations. Three of the BYP registers connect to the IB through the IBS, a sub-bus which also connects to the CPU interrupt control registers.

Simultaneous access of a register by both the IB and the SIB is prevented by timing. In particular, RAM access is restricted to alternate internal processor cycles for the CPU and the SIU, in such a way that collisions do not occur.

As an example of the operation of the byte processor, the following sequence occurs in relation to the receive data:

1) Assuming that there is an address field in the frame, the BYP takes the station address from the register file into temporary storage. After the opening flag, the next field (the address field) is compared to the station address in the temporary storage. If a match occurs, the operation continues.
2) Assuming that there is a control field in the frame, the BYP takes the next byte and loads it into the RCB register. The RCB register has the logic to update the NSNR register (increment receive count, set SES and SER flags, etc.).
3) Assuming that there is an information field, the next byte is dumped into RAM at the RBS location. The DMA CNT (RBL at the opening flag) is loaded from the DMA CNT register into the RB register and decremented. The RFL is then loaded into the RB register, incremented, and stored back into the register file.

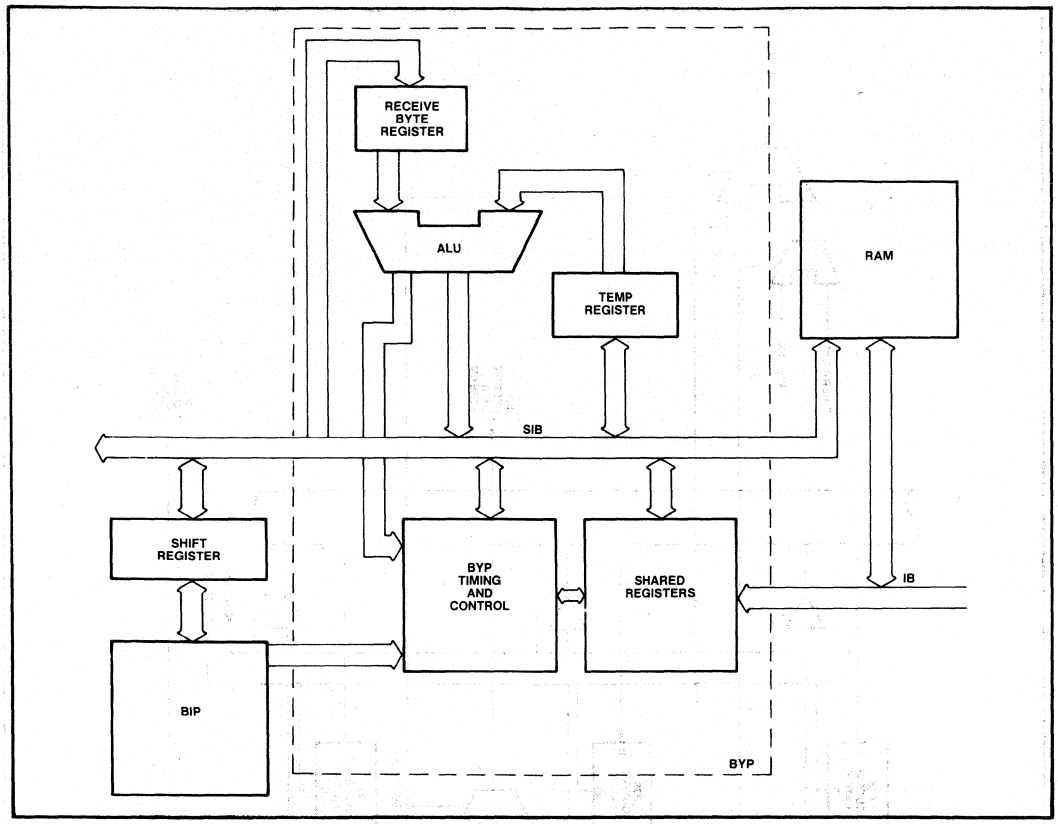


Figure 13-16. The Byte Processor

4) This process continues until the DMA CNT reaches zero, or until a closing flag is received. Upon either event, the BYP updates the status, and, if the CRC is good, the NSNR register.

13.10 DIAGNOSTICS

An SIU test mode has been provided, so that the on-chip CPU can perform limited diagnostics on the SIU. The test mode utilizes the output latches for P3.0 and P3.1 (pins 10 and 11). These port 3 pins are not useful as output ports, since the pins are taken up by the serial port functions. Figure 13-17 shows the signal routing associated with the SIU test mode.

Writing a 0 to P3.1 enables the serial test mode (P3.1 is set to 1 by reset). In test mode the P3.0 bit is mapped into the received data stream, and the 'write port 3' control signal is mapped into the SCLK path in place of T1. Thus, in test mode, the CPU can send a serial data stream to the SIU by writing to P3.0. The transmit data stream can be monitored by reading P3.1. Each sucessive bit is transmitted from the SIU by writing to any bit in Port 3, which generates SCLK.

In test mode, the P3.0 and P3.1 pins are placed in a high voltage, high impedance state. When the CPU reads P3.0 and P3.1 the logic level applied to the pin will be returned. In the test mode, when the CPU reads 3.1, the transmit data value will be returned, not the voltage on the pin. The transmit data remains constant for a bit time. Writing to P3.0 will result in the signal being outputted for a short period of time. However, since the signal is not latched, P3.0 will quickly return to a high voltage, high impedance state.

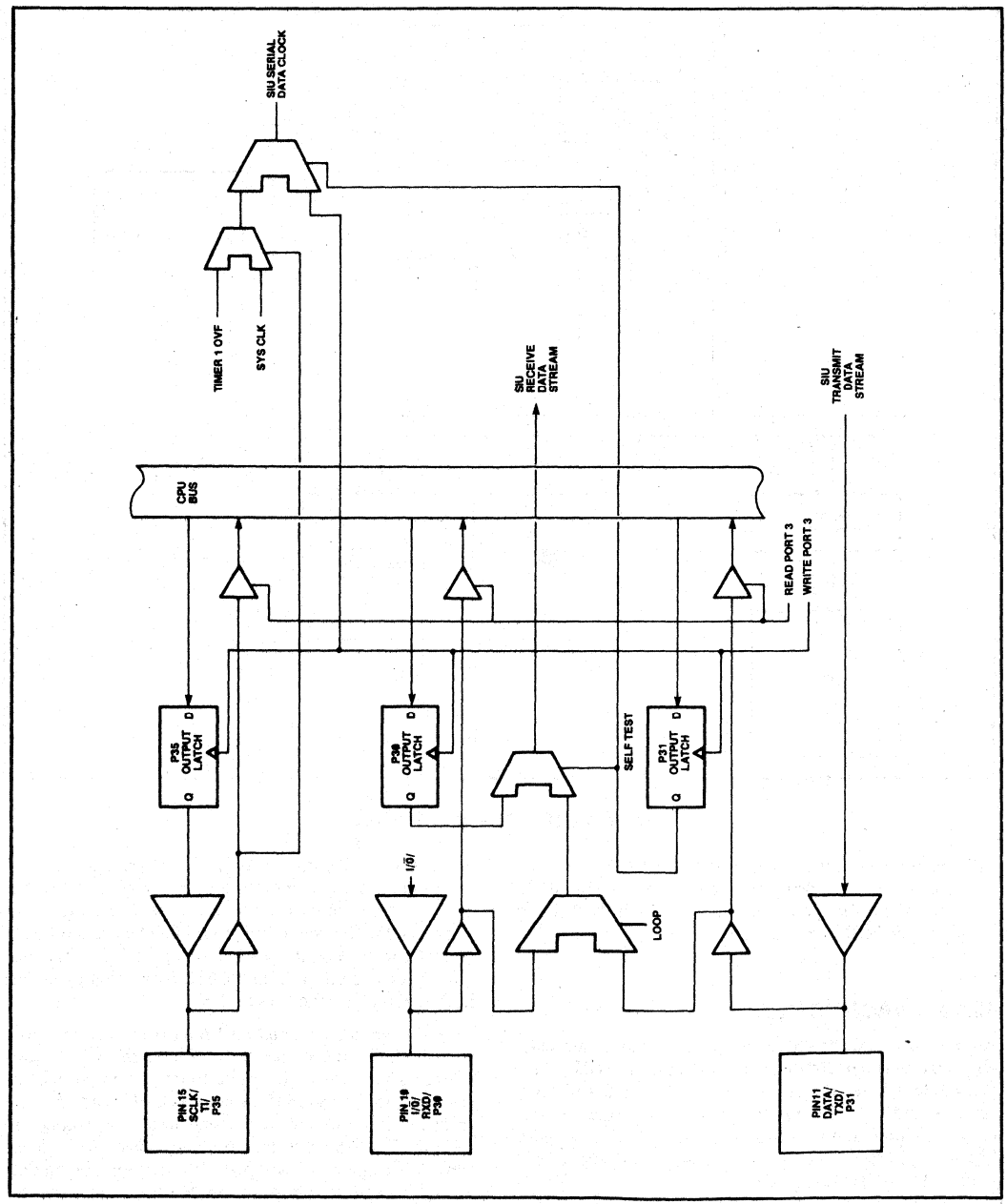


Figure 13-17. SIU Test Mode

The serial test mode is disabled by writing a 1 to P3.1. Care must be taken that a 0 is never written to P3.1 in the course of normal operation, since this causes the test mode to be entered.

Figure 13-18 is an example of a simple program segment that can be imbedded into the user's diagnostic program. The example shows how to put the 8044 into "Loop-back mode" to test the basic transmitting and receiving functions of the SIU.

Loop-back mode is functionally equivalent to a hardwire connection between pins 10 and 11 on the 8044.

In this example, the 8044 CPU plays the role of the primary station. The SIU is in the AUTO mode. The CPU sends the SIU a supervisory frame with the poll bit set and an RNR command. The SIU responds with a supervisory frame with the poll bit set and an RR command.

The operation proceeds as follows:

Interrupts are disabled, and the self test mode is enabled by writing a zero to P3.1. This establishes P3.0 as the data path from the CPU to the SIU. CTS (clear-to-send) is enabled by writing a zero to P1.7. The station address is initialized by writing 08AH into the STAD (station address register).

The SIU is configured for receive operation in the clocked mode and in AUTO mode. The CPU then transmits a supervisory frame. This frame consists of an opening flag, followed by the station address, a control field indicating that this is a supervisory frame with an RNR command, and then a closing flag.

Each byte of the frame is transmitted by writing that byte into the A register and then calling the subroutine XMIT8. Two additional SCLKs are generated to guarantee that the last bits in the frame have been clocked into the SIU. Finally the CPU reads the status register (STS). If the operation has proceeded correctly, the status will be 072H. If it is not, the program jumps to the ERROR loop and terminates.

The SIU generates an SI (SIU interrupt) to indicate that it has received a frame. The CPU clears this interrupt, and then begins to monitor the data stream that is being generated by the SIU in response to what it has received. As each bit arrives (via P3.1), it is moved into the accumulator, and the CPU compares the byte in the accumulator with 07EH, which is the opening flag. When a match occurs, the CPU identifies this as byte boundary, and thereafter processes the information byte-by-byte.

The CPU calls the RCV8 subroutine to get each byte into the accumulator. The CPU performs compare operations on (successively) the station address, the control field (which contains the RR response), and the closing flag. If any of these do not compare, the program jumps to the ERROR loop. If no error is found, the program jumps to the DONE loop.

```
MCS-51 MACRO ASSEMBLER    DATA

ISIS-II MCS-51 MACRO ASSEMBLER V2.0
OBJECT MODULE PLACED IN :F1:DATA.OBJ
ASSEMBLER INVOKED BY:  asm51 :f1:data.man device(44)

LOC  OBJ            LINE     SOURCE

                       1
                       2
0000 75C800            3     INIT:   MOV     STS,#00H
0003 C2B1              4             CLR     P3.1                ; Enable self test mode
0005 C297              5             CLR     P1.7                ; Enable CTS
0007 75CE8A            6             MOV     STAD, #8AH          ; Initialize address
                       7
                       8       ; CONFIGURE RECEIVE OPERATION
                       9
000A 75D86A           10             MOV     NSNR, #6AH          ; NS(S)=3, SES=0, NR(S)=5, SER=0
000D 75C901           11             MOV     SMD, #01H           ; NFCS=1
0010 75C8C2           12             MOV     STS, #0C2H          ; TBF=1, RBE=1, AM=1
                      13
                      14     ;   TRANSMIT A SUPERVISORY FRAME FROM THE PRIMARY STATION WITH THE POLL
                      15     ;   BIT SET AND A RNR COMMAND
                      16
0013 747E             17     SEND:   MOV     A, #7EH             ; The SIU receives a flag first
0015 120066           18             CALL    XMITB
0018 748A             19             MOV     A, #8AH             ; The address is next
001A 120066           20             CALL    XMITB
001D 7495             21             MOV     A, #095H            ; RNR SUP FRAME with P/F=1, NR(P)=4
001F 120066           22             CALL    XMITB
0022 747E             23             MOV     A, #7EH             ; Receive closing flag
0024 120066           24             CALL    XMITB
0027 D2B0             25             SETB    P3.0                ; Generate extra SCLK's to
0029 D2B0             26             SETB    P3.0                ; Initiate receive action
                      27
002B E5C8             28             MOV     A, STS              ; Check for appropriate status
002D B4722A           29             CJNE    A, #72H, ERROR
                      30
                      31     ; PREPARE TO RECEIVE RUPI'S RESPONCE TO PRIMARY'S RNR
                      32
                      33
                      34
0030 C2CC             35     RECV:   CLR     SI                  ; Clear SI
0032 7400             36             MOV     A, #00H             ; Clear ACC
0034 780C             37             MOV     R3, #12             ; Try 12 times
                      38
                      39     ;  LOOK FOR THE OPENING FLAG
                      40
0036 D2B0             41     WFLAG1: SETB    P3.0                ; SCLK
0038 A2B1             42             MOV     C, P3.1             ; Transmitted data
003A 13               43             RRC     A
003B B47E03           44             CJNE    A, #07EH, WFLG1
003E 020046           45             JMP     CNTINU
0041 DBF3             46     WFLG1:  DJNZ    R3, WFLAG1
0043 02005A           47             JMP     ERROR
                      48
                      49
0046 12005C           50     CNTINU: CALL    RCVB                ; Get SIU's Transmitted address field
0049 B48A0E           51             CJNE    A, #08AH, ERROR
004C 12005C           52             CALL    RCVB                ; Primary expects to receive RR from SIU
004F B4B108           53             CJNE    A, #0B1H, ERROR
0052 12005C           54             CALL    RCVB                ; Receive closing flag
0055 B47E02           55             CJNE    A, #07EH, ERROR
                      56
0058 80FE             57     DONE:   JMP     DONE
                      58
005A 80FE             59     ERROR:  JMP     ERROR
                      60
                      61
005C 7808             62       RCVB: MOV     R0, #08             ; Initialize the bit counter
005E D2B0             63     GETBIT: SETB    P3.0                ; SCLK
0060 A2B1             64             MOV     C, P3.1             ; Transmitted data
0062 13               65             RRC     A
0063 DBF9             66             DJNZ    R0, GETBIT
0065 22               67             RET
                      68
                      69
                      70
0066 7809             71     XMITB:  MOV     R0, #9              ; Initialize the bit counter
0068 13               72        L3:  RRC     A                   ; Put the bit to be transmitted
                      73                                         ; in the Carry
0069 DB01             74             DJNZ    R0, L1              ; When all bits have been sent
006B 22               75             RET                         ; return
                      76
006C 4004             77        L1:  JC      L2                  ; If the carry bit is set, set
                      78                                         ; port P3.0 else
006E C2B0             79             CLR     P3.0                ; clear port P3.0
0070 80F6             80             JMP     L3
                      81
0072 D2B0             82        L2:  SETB    P3.0
0074 80F2             83             JMP     L3
                      84     end
```

Figure 13.8. Loop-Back Mode Software

CHAPTER 14
8044 APPLICATION EXAMPLES

14.0 8044 APPLICATION EXAMPLES

14.1 INTERFACING THE 8044 TO A MICROPROCESSOR

The 8044 is designed to serve as an intelligent controller for remote peripherals. However, it can also be used as an intelligent HDLC/SDLC front end for a microprocessor, capapble of extensively off-loading link control functions for the CPU. In some applications, the 8044 can even be used for communications preprocessing, in addition to data link control.

This section describes a sample hardware interface for attaching the 8044 to an 8088. It is general enough to be extended to other microprocessors such as the 8086 or the 80186.

OVERVIEW

A sample interface is shown in Figure 14-1. Transmission occurs when the 8088 loads a 64 byte block of memory with some known data. The 8088 then enables the 8237A to DMA this data to the 8044. When the 8044 has received all of the data from the 8237A, it sends the data in a SDLC frame. The frame is captured by the Spectron Datascope®* which displays it on a CRT in hex format.

In reception, the Datascope sends an SDLC information frame to the 8044. The 8044 receives the SDLC frame, buffers it, and sends it to the 8088's memory. In this example the 8044 is being operated in the NON-AUTO mode; therefore, it does not need to be polled by a primary station in order to transmit.

THE INTERFACE

The 8044 does not have a parallel slave port. The 8044's 32 I/O lines can be configured as a local microprocessor bus master. In this configuration, the 8044 can expand the ROM and RAM memory, control peripherals, and communicate with a microprocessor.

The 8044, like the 8051, does not have a Ready line, so there is no way to put the 8044 in wait state. The clock on the 8044 cannot be stopped. Dual port RAM could still be used, however, software arbitration would be the only way to prevent collisions. Another way to interface the 8044 with another CPU is to put a FIFO or queue between the two processors, and this was the method chosen for this design.

Figure 14-2 shows the schematic of the 8044/8088 interface. It involves two 8 bit tri-state latches, two SR flip-flops, and some logic gates (6 TTL packs). The circuitry implements a one byte FIFO. RS422 transceivers are used, which can be connected to a multidrop link. Figure 14-3 shows the 8088 and support circuitry; the memory and decoders are not shown. It is a basic 8088 Min Mode system with an 8237A DMA controller and an 8259A interrupt controller.

DMA Channel One transfers a block of memory to the tri-state latch, while Channel Zero transfers a block of data from the latch to 8088's memory. The 8044's Interrupt 0 signal vectors the CPU into a routine which reads from the internal RAM and writes to the latch. The 8044's Interrupt 1 signal causes the chip to read from the latch and write to its on-chip data RAM. Both DMA requests and acknowledges are active low.

Initially, when the power is applied, a reset pulse coming from the 8284A initializes the SR flip-flops. In this initialization state, the 8044's transmit interrupt and the 8088's transmit DMA request are active; however, the software keeps these signals disabled until either of the two processors are ready to transmit. The software leaves the receive signals enabled, unless the receive buffers are full. In this way either the 8088 or the 8044 are always ready to receive, but they must enable the transmit signal when they have prepared a block to transmit. After a block has been transmitted or received, the DMA and interrupt signals return to the initial state.

The receive and transmit buffer sizes for the blocks of data sent between the 8044 and the 8088 have a maximum fixed length. In this case the buffer size was 64 bytes. The buffer size must be less than 192 bytes to enable 8044 to buffer the data in its on-chip RAM. This design allows blocks of data that are less than 64 bytes, and accommodates networks that allow frames of varying size. The first byte transferred between the 8088 and the 8044 is the byte count to follow; thus the 8044 knows how many bytes to receive before it transmits the SDLC frame. However, when the 8044 sends data to the 8088's memory, the 8237A will not know if the 8044 will send less than the count the 8237A was programmed for. To solve this problem, the 8237A is operated in the single mode. The 8044 uses an I/O bit to generate an interrupt request to the 8259A. In the 8088's interrupt routine, the 8237A's receive DMA channel is disabled, thus allowing blocks of data less than 64 bytes to be received.

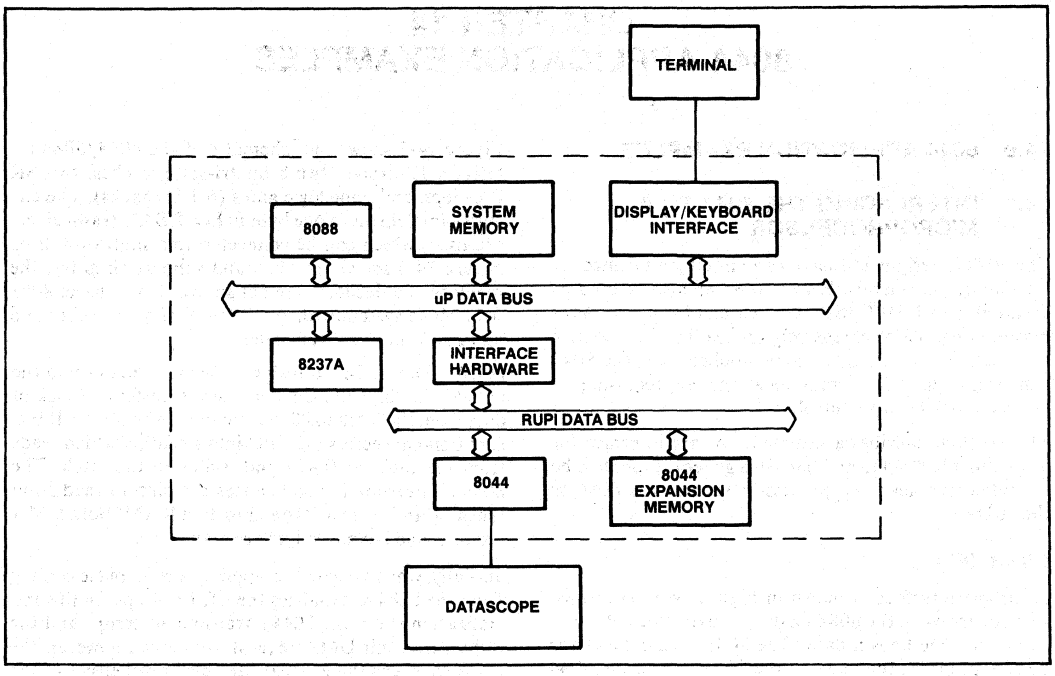


Figure 14-1. Block Diagram of 8088/8044 Interface Test

THE SOFTWARE

The software for the 8044 and the 8088 is shown in Table 14-1. The 8088 software was written in PL/M86, and the 8044 software was written in assembly language.

The 8044 software begins by initializing the stack, interrupt priorities, and triggering types for the interrupts. At this point, the SIU parameter registers are initialized. The receive and transmit buffer starting addresses and lengths are loaded for the on-chip DMA. This DMA is for the serial port. The serial station address and the transmit control bytes are loaded too.

Once the initialization has taken place, the SIU interrupt is enabled, and the external interrupt which receives bytes from the 8088 is enabled. Setting the 8044's Receive Buffer Empty (RBE) bit enables the receiver. If this bit is reset, no serial data can be received. The 8044 then waits in a loop for either RECEIVE DMA interrupt or the SERIAL INT interrupt.

The RECEIVE DMA interrupt occurs when the 8237A is transferring a block of data to the 8044. The first time this interrupt occurs, the 8044 reads the latch and loads the count value into the R2 register. On subsequent interrupts, the 8044 reads the latch, loads the data into the transmit buffer, and decrements R2. When R2 reaches zero, the interrupt routine sends the data in an SDLC frame, and disables the RECEIVE DMA interrupt. After the frame has been transmitted, a serial interrupt is generated. The SERIAL INT routine detects that a frame has been transmitted and re-enables the RECEIVE DMA interrupt. Thus, while the frame is being transmitted through the SIU, the 8237A is inhibited from sending data to the 8044's transmit buffer.

The TRANSMIT DMA routine sends a block of data from the 8044's receive buffer to the 8088's memory. Normally this interrupt remains disabled. However, if a serial interrupt occurs, and the SERIAL INT routine detects that a frame has been received, it calls the SEND subroutine. The SEND subroutine loads the number of bytes which were received in the frame into the receive buffer. Register R1 points to the receive buffer and R2 is loaded with the count. The TRANSMIT DMA interrupt is enabled, and immediately upon returning from the SERIAL INT routine, the interrupt is

acknowledged. Each time the TRANSMIT DMA interrupt occurs, a byte is read from the receive buffer, written to the latch, and R2 is decremented. When R2 reaches 0, the TRANSMIT DMA interrupt is disabled, the SIU receiver is re-enabled, and the 8044 interrupts the 8088.

The 8088 software simply transmits a block of data and receives a block of data, then stops. The software begins by initializing the 8237A, and the 8259A. It then loads a block of memory with some data and enables the 8237A to transmit the data. In the meantime the 8088 waits in a loop. After a block of data is received from the 8044, the 8088 is interrupted, and it shuts off the 8237A receive DMA.

CONCLUSION

For the software shown in Table 14-1, the transfer rate from the 8088's memory to the 8044 was measured at 75K bytes/sec. This transfer rate largely depends upon the number of instructions in the 8044's interrupt service routine. Fewer instructions result in a higher transfer rate.

There are many ways of interfacing the 8044 locally to another microprocessor: FIFO's, dual port RAM with software arbitration, and 8255's are just a few. Alternative approaches, which may be more optimal for certain applications, are certainly possible.

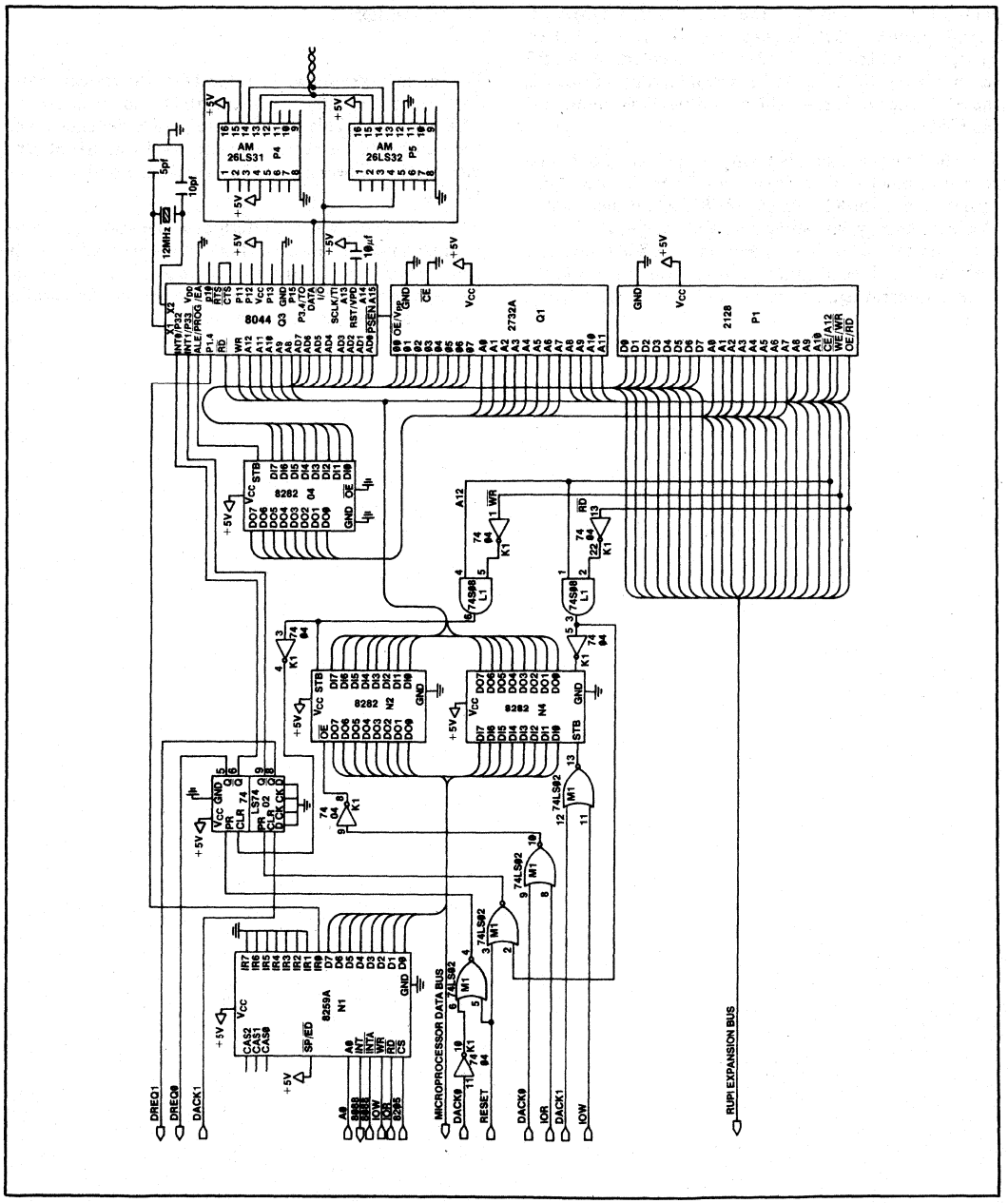


Figure 14-2. 8044 Interface to the 8088

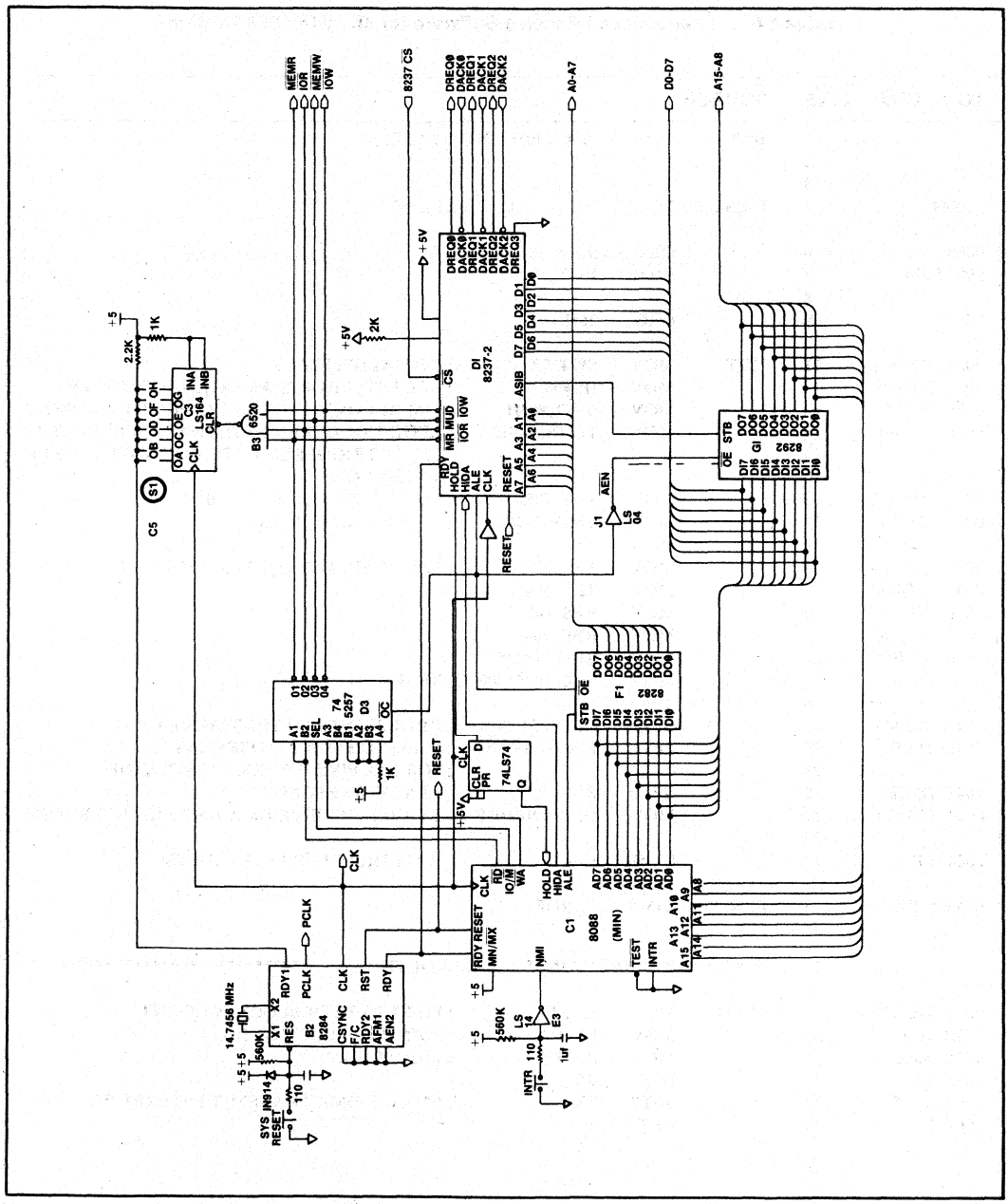


Figure 14-3. 8088 Min Mode System

Table 14-1. Transmit and Receive Software for an 8044/8088 System

```
LOC  OBJ       LINE  SOURCE

                1    $debug    title    (8044/8088 INTERFACE)
                2
                3
 0000           4    FIRST_BYTE        BIT      0       ; FLAG
                5
0000            6              ORG      0
0000 8024       7              SJMP     INIT
                8
0026            9              ORG      26H
               10
0026 7581AA    11    INIT:     MOV      SP, #170        ; INITIALIZE STACK
0029 75B800    12              MOV      IP, #00         ; ALL INTERRUPTS ARE EQUAL PRIORITY
002C 75C954    13              MOV      SMD, #54H       ; TIMER 1 OVERFLOW, NRZI, PRE-FRAME SYNC
002F 758844    14              MOV      TCON, #44H      ; EDGE TRIGGERED EXTERNAL INTERRUPT 1
               15                                       ; LEVEL TRIGGERED EXTERNAL INTERRUPT 0
               16                                       ; TIMER 1 ON
0032 758DEC    17              MOV      TH1, #0ECH      ; INITIALIZE TIMER, 3125 BPS
0035 758920    18              MOV      TMOD, #20H      ; TIMER 1 AUTO RELOAD
               19
0038 75DC6A    20              MOV      TBS, #106       ; SET UP SIU PARAMETER REGISTERS
003B 75DB40    21              MOV      TBL, #64
003E 75CC2A    22              MOV      RBS, #42
0041 75CB40    23              MOV      RBL, #64
0044 75CE55    24              MOV      STAD, #55H
0047 75DA11    25              MOV      TCB, #00010001B ; RR, P/F=1
               26
004A 901000    27              MOV      DPTR, #1000H    ; DPTR POINTS TO TRI-STATE LATCH
004D D200      28              SETB     FIRST_BYTE      ; FLAG TO INDICATE FIRST BYTE
               29                                       ; FOR RECEIVE INTERRUPT ROUTINE
004F D2CE      30              SETB     RBE             ; READY TO RECEIVE
0051 75A894    31              MOV      IE, #10010100B  ; ENABLE RECEIVE DMA AND SIU INTERRUPT
               32
0054 80FE      33              SJMP     $               ; WAIT HERE FOR INTERRUPTS
               34
0056 80FE      35    ERROR:    SJMP     ERROR
               36 +1 $EJ
               37    ;*************************    SUBROUTINES    ********************************
               38
0058 85CD29    39    SEND:     MOV      41, RFL         ; FIRST BYTE IN BLOCK IS COUNT
005B 7929      40              MOV      R1, #41         ; POINT TO BLOCK OF DATA
005D AACD      41              MOV      R2, RFL         ; LOAD COUNT
005F 0A        42              INC      R2
0060 D2A8      43              SETB     EX0             ; ENABLE DMA TRANSMIT INTERRUPT
0062 22        44              RET
               45
               46
               47
               48    ;*******************    INTERRUPT SERVICE ROUTINES    **********************
```

```
                        49
0063                    50  LOC_TMP SET     $               ; SET UP INTERRUPT TABLE JUMP
0013                    51          ORG     0013H
0013 020063             52          LJMP    RECEIVE_DMA
0063                    53          ORG     LOC_TMP
                        54
                        55  RECEIVE_DMA:
                        56
0063 10000E             57          JBC     FIRST_BYTE, L1 ; THE FIRST BYTE TRANSFERRED IS THE COUNT
                        58
0066 E0                 59          MOVX    A, @DPTR        ; READ THE LATCH
0067 F6                 60          MOV     @R0, A          ; PUT IT IN TRANSMIT BUFFER
0068 08                 61          INC     R0
0069 DA08               62          DJNZ    R2, L2          ; AFTER READING BYTES,
                        63
006B D2CF               64          SETB    TBF             ; SEND DATA
006D D2CD               65          SETB    RTS
006F D200               66          SETB    FIRST_BYTE
0071 C2AA               67          CLR     EX1
                        68
0073 32                 69  L2:     RETI
                        70
0074 786A               71  L1:     MOV     R0, #106        ; R0 IS A POINTER TO THE TRANSMIT
                        72                                  ; BUFFER STARTING ADDRESS
0076 E0                 73          MOVX    A, @DPTR        ; PUT THE FIRST BYTE INTO
0077 FA                 74          MOV     R2, A           ; R2 FOR THE COUNT
0078 32                 75          RETI
                        76
  0079                  77  LOC_TMP SET     $
0003                    78          ORG     0003H
0003 020079             79          LJMP    TRANSMIT_DMA
0079                    80          ORG     LOC_TMP
                        81
                        82  TRANSMIT_DMA
                        83
0079 E7                 84          MOV     A, @R1          ; READ BYTE OUT OF THE RECEIVE BUFFER
007A F0                 85          MOVX    @DPTR, A        ; WRITE IT TO THE LATCH
007B 09                 86          INC     R1
007C DA08               87          DJNZ    R2, L3          ; WHEN ALL BYTES HAVE BEEN SENT
                        88
007E C2A8               89          CLR     IE. 0           ; DISABLE INTERRUPT
0080 C294               90          CLR     P1. 4           ; CAUSE 8088 INTERRUPT TO TERMINATE DMA
0082 D294               91          SETB    P1. 4
0084 D2CE               92          SETB    RBE             ; ENABLE RECEIVER AGAIN
                        93
0086 32                 94  L3:     RETI
                        95
                        96
                        97
  0087                  98  LOC_TMP SET     $
0023                    99          ORG     0023H
0023 020087            100          LJMP    SERIAL_INT
0087                   101          ORG     LOC_TMP
                       102
```

```
                    103   SERIAL_INT:
                    104
0087 30CE06         105         JNB     RBE, RCV        ; WAS A FRAME RECEIVED
008A 30CF0B         106         JNB     TBF, XMIT       ; WAS A FRAME TRANSMITTED
008D 020056         107         LJMP    ERROR           ; IF NEITHER ERROR
                    108
0090 20CBC3         109   RCV:  JB      BOV, ERROR      ; IF BUFFER OVERRUN THEN ERROR
0093 1158           110         CALL    SEND            ; SEND THE FRAME TO THE 8088
0095 C2CC           111         CLR     SI
0097 32             112         RETI
                    113
0098 C2CC           114   XMIT: CLR     SI
009A D2AA           115         SETB    EX1
009C 32             116         RETI
                    117
                    118         END
```

SYMBOL TABLE LISTING

NAME	TYPE	VALUE	ATTRIBUTES
BOV	B ADDR	00C8H.3	A
ERROR	C ADDR	0056H	A
EX0	B ADDR	00A8H.0	A
EX1	B ADDR	00A8H.2	A
FIRST_BYTE	B ADDR	0020H.0	A
IE	D ADDR	00A8H	A
INIT	C ADDR	0026H	A
IP	D ADDR	00B8H	A
L1	C ADDR	0074H	A
L2	C ADDR	0073H	A
L3	C ADDR	0086H	A
LOC_TMP	C ADDR	0087H	A
P1	D ADDR	0090H	A
RBE	B ADDR	00C8H.6	A
RBL	D ADDR	00CBH	A
RBS	D ADDR	00CCH	A
RCV	C ADDR	0090H	A
RECEIVE_DMA	C ADDR	0063H	A
RFL	D ADDR	00CDH	A
RTS	B ADDR	00C8H.5	A
SEND	C ADDR	0058H	A
SERIAL_INT	C ADDR	0087H	A
SI	B ADDR	00C8H.4	A
SMD	D ADDR	00C9H	A
SP	D ADDR	0081H	A
STAD	D ADDR	00CEH	A
TBF	B ADDR	00C8H.7	A
TBL	D ADDR	00DBH	A
TBS	D ADDR	00DCH	A
TCB	D ADDR	00DAH	A
TCON	D ADDR	0088H	A
TH1	D ADDR	008DH	A
TMOD	D ADDR	0089H	A

```
TRANSMIT_DMA  .  C ADDR     0079H   A
XMIT  . . . . .  C ADDR     0098H   A

  REGISTER BANK(S) USED: 0, TARGET MACHINE(S): 8044

  ASSEMBLY COMPLETE, NO ERRORS FOUND
```

Table 14-2. PL/M-86 Compiler Rupi/8088 Interface Example

```
SERIES-III PL/M-86 V1.0 COMPILATION OF MODULE RUPI_88
OBJECT MODULE PLACED IN :F1:R88.OBJ
COMPILER INVOKED BY:  PLM86.86 :F1:R88.SRC


            $DEBUG
            $TITLE  ('RUPI/8088 INTERFACE EXAMPLE')

  1         RUPI_88:DO;

  2  1        DECLARE

              LIT                 LITERALLY   'LITERALLY',
              TRUE                LIT         '01H',
              FALSE               LIT         '00H',

              RECV_BUFFER(64)     BYTE,
              XMIT_BUFFER(64)     BYTE,
              I                   BYTE,
              WAIT                BYTE,

                              /* 8237 PORTS*/

              MASTER_CLEAR_37     LIT         'OFFDDH',
              COMMAND_37          LIT         'OFFD8H',
              ALL_MASK_37         LIT         'OFFDFH',
              SINGLE_MASK_37      LIT         'OFFDAH',
              STATUS_37           LIT         'OFFD8H',
              REQUEST_REG_37      LIT         'OFFD9H',
              MODE_REG_37         LIT         'OFFDBH',
              CLEAR_BYTE_PTR_37   LIT         'OFFDCH',

              CH0_ADDR            LIT         'OFFD0H',
              CH0_COUNT           LIT         'OFFD1H',
              CH1_ADDR            LIT         'OFFD2H',
              CH1_COUNT           LIT         'OFFD3H',
              CH2_ADDR            LIT         'OFFD4H',
              CH2_COUNT           LIT         'OFFD5H',
              CH3_ADDR            LIT         'OFFD6H',
              CH3_COUNT           LIT         'OFFD7H',

                        /* 8237 BIT ASSIGNMENTS */

              CH0_SEL             LIT         '00H',
              CH1_SEL             LIT         '01H',
              CH2_SEL             LIT         '02H',
              CH3_SEL             LIT         '03H',
              WRITE_XFER          LIT         '04H',
              READ_XFER           LIT         '08H',
              DEMAND_MODE         LIT         '00H',
              SINGLE_MODE         LIT         '40H',
              BLOCK_MODE          LIT         '80H',
              SET_MASK            LIT         '04H',
```

```
            $EJECT
                                /* 8259 PORTS */

                STATUS_POLL_59      LIT         'OFFE0H',
                ICW1_59             LIT         'OFFE0H',
                OCW1_59             LIT         'OFFE1H',
                OCW2_59             LIT         'OFFE0H',
                OCW3_59             LIT         'OFFE0H',
                ICW2_59             LIT         'OFFE1H',
                ICW3_59             LIT         'OFFE1H',
                ICW4_59             LIT         'OFFE1H';

                            /* INTERRUPT SERVICE ROUTINE */

 3   1      OFF_RECV_DMA:   PROCEDURE   INTERRUPT 32;

 4   2          OUTPUT(SINGLE_MASK_37)=40H;
 5   2          WAIT=FALSE;
 6   2          END;

 7   1          DISABLE;

                            /* INITIALIZE 8237 */

 8   1          OUTPUT(MASTER_CLEAR_37)     =0;
 9   1          OUTPUT(COMMAND_37)          =040H;
10   1          OUTPUT(ALL_MASK_37)         =0FH;
11   1          OUTPUT(MODE_REG_37)         =(SINGLE_MODE OR WRITE_XFER OR CH0_SEL);
12   1          OUTPUT(MODE_REG_37)         =(SINGLE_MODE OR READ_XFER OR CH1_SEL);
13   1          OUTPUT(CLEAR_BYTE_PTR_37)   =0;
14   1          OUTPUT(CH0_ADDR)            =00H;
15   1          OUTPUT(CH0_ADDR)            =40H;
16   1          OUTPUT(CH0_COUNT)           =64;
17   1          OUTPUT(CH0_COUNT)           =00;
18   1          OUTPUT(CH1_ADDR)            =40H;
19   1          OUTPUT(CH1_ADDR)            =40H;
20   1          OUTPUT(CH1_COUNT)           =64;
21   1          OUTPUT(CH1_COUNT)           =00;

                            /* INITIALIZE 8259 */

22   1          OUTPUT(ICW1_59)             =13H; /*SINGLE MODE, EDGE TRIGGERED
                                                    INPUT, 8086 INTERRUPT TYPE*/
23   1          OUTPUT(ICW2_59)             =20H; /*INTERRUPT TYPE 32*/
24   1          OUTPUT(ICW4_59)             =03H; /*AUTO-EOI*/
25   1          OUTPUT(OCW1_59)             =0FEH; /*ENABLE INTERRUPT LEVEL 0*/
```

```
                $EJECT
  26    1           CALL SET$INTERRUPT  (32,OFF_RECV_DMA); /*LOAD INTERRUPT VECTOR LOCATION*/

  27    1           XMIT_BUFFER(0)=64; /*THE FIRST BYTE IN THE BLOCK OF DATA IS THE NUMBER
                                         OF BYTES TO BE TRANSFERED; NOT INCLUDING THE FIRST BYTE*/

  28    1           DO  I= 1 TO 64;  /* FILL UP THE XMIT_BUFFER WITH DATA */
  29    2           XMIT_BUFFER(I)=I;
  30    2           END;

  31    1           OUTPUT(ALL_MASK_37)=0FCH;   /*ENABLE CHANNEL 1 AND 2 */

  32    1           ENABLE;

  33    1           WAIT=TRUE;
  34    1           DO WHILE WAIT;
  35    2           END;                  /* A BLOCK OF DATA WILL BE TRANSFERRED TO THE RUPI.
                                             WHEN THE RUPI RECEIVES A BLOCK OF DATA IT WILL
                                             SEND IT TO THE 8088 MEMORY AND INTERRUPT THE 8088.
                                             THE INTERRUPT SERVICE ROUTINE WILL SHUT OFF THE DMA
                                             CONTROLLER AND SET 'WAIT' FALSE */

  36    1           DO WHILE 1;
  37    2           END;


  38    1       END;



MODULE INFORMATION:

     CODE AREA SIZE     = 00D7H    215D
     CONSTANT AREA SIZE = 0000H      0D
     VARIABLE AREA SIZE = 0082H    130D
     MAXIMUM STACK SIZE = 001EH     30D
     124 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

END OF PL/M-86 COMPILATION
```

INDEX

Acknowledgment
  See AUTO Mode
Acknowledgment Time 13.8.4
AUTO Mode 11.1, 13.4.1, 13.8.2
Bit Rate 13.3
Bit Processor 13.9.1
Buffer Size
  See Transmit Buffer Length 13.7.2
Byte Processor 13.9.2
Clock
  Modes 13.2, 13.7.1
  Serial 13.2, 13.7.1
  Recovery, see also DPLL 13.2, 13.7
Control Field, HDLC/SDLC 11.1.7
Count, Send/Receive 13.7.1
Cyclic Redundancy Code (CRC) 13.5.1, 13.5.4
Data
  Clock Recovery 13.2, 13.7
  Rates 13.3
  Transparency 11.12, 11.2.4
Debugging, Serial Communication
  Data Link Interface 13.8.3
  Diagnostics 13.10
  ICE-44 11.3.1
  SIUST Register 13.7.3
Diagnostics 13.10
DPLL 11.2.5, 13.2
Dribble Bits 13.8.3
Duplex 13.1
Frame Formats 13.5
Frame Number Sequence 13.7.1
Half Duplex 13.1
HDLC/SDLC 11.2
  Definitions 11.2.1
  Differences between 11.2.3, 13.6
  Features 11.2.1
  Frame Formats 11.2.3, 13.5
  Networks 11.2.2, 13.1
  References 11.2.6
ICE-44 11.3.1, 13.7.3
Initialization 13.8.1
Link Address Field 11.1.3
Loop 11.2.2, 13.1, 13.7.1, 13.8.3
Modems, Operating link 12.10, 13.7.1
Multipoint 11.2.2, 13.1
Non-AUTO Mode 13.4, 13.8.3
Non-Buffered Mode 13.5, 13.7.1
Normal Response
  See AUTO Mode
NRZI Coding 11.1.5
Packet
  See Frame
Phase Locked Loop 11.2.5, 13.2
Pin description 12.4
Point to point 11.2.2, 13.1
Polling 13.7.1, 13.8.4
Receive
  Operation 13.8.1, 13.8.2
  Debug, See Debugging
Receive Not Ready
  See AUTO Mode
Receive Ready
  See AUTO Mode
Registers
  SIU 3.7, 12.1.1
Response, HDLC/SDLC
  See AUTO Mode
Reset 12.3
SDLC
  See HDLC/SDLC
Serial Interface Unit 3.0
Software, Sample Transmit/Receive 14.0
Transmit Operation 13.8.1, 13.8.2
Unnumbered Poll
  See AUTO Mode
Workshop 11.3.2
Zero bit insertion 11.2.4

March 1983

8096 16-Bit Microcontroller Architectural Specification and Functional Description

8096/8396
16-BIT MICROCONTROLLER

- **8396: an 8096 with 8K Bytes of On-chip ROM**

- **High Speed Pulse I/O**
- **10-bit A/D Converter**
- **8 Interrupt Sources**
- **Pulse-Width Modulated Output**
- **Four 16-bit Software Timers**
- **232 Byte Register File**
- **Memory-to-Memory Architecture**
- **Full Duplex Serial Port**
- **Five 8-bit I/O Ports**
- **Watchdog Timer**

The iACX-96 family of 16-bit microcontrollers consists of the 8096 and the 8396. The 8096 is a 16-bit microcomputer circuit designed for high-speed control functions. The 8396 is an 8096 with 8K bytes of on-chip ROM.

The CPU supports bit, byte, and word operations. 32-bit double-words are supported for a subset of the instruction set. With a 12 MHz input frequency the 8096 can do a 16-bit addition in 1.0 μsec and a 16 x 16-bit multiply or 32/16-bit divide in 6.5 μsec Instruction execution times average 1 to 2 μsec in typical applications.

Four high-speed trigger inputs are provided to record the times at which external events occur. Six high-speed pulse generator outputs are provided to trigger external events at preset times. The high-speed output unit can simultaneously perform timer functions. Up to four such 16-bit Software Timers can be in operation at once.

An optional on-chip A/D Converter converts up to 4 (in the 48-pin version) or 8 (in the 68-pin version) analog input channels to 10-bit digital values.

Also provided on-chip are a serial port, a watchdog timer, and a pulse-width modulated output signal.

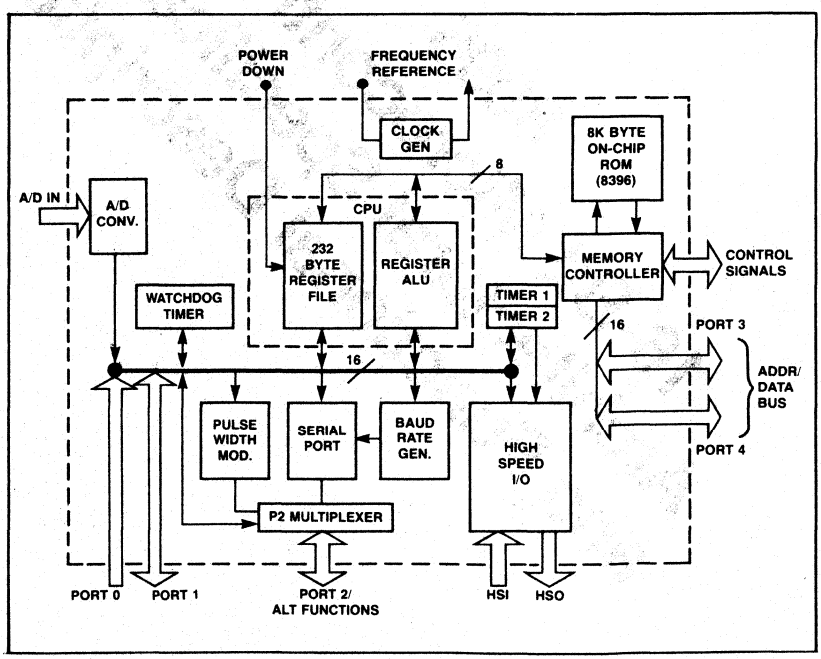


Figure 1. Block Diagram (For simplicity, lines connecting port registers to port buffers are not shown.)

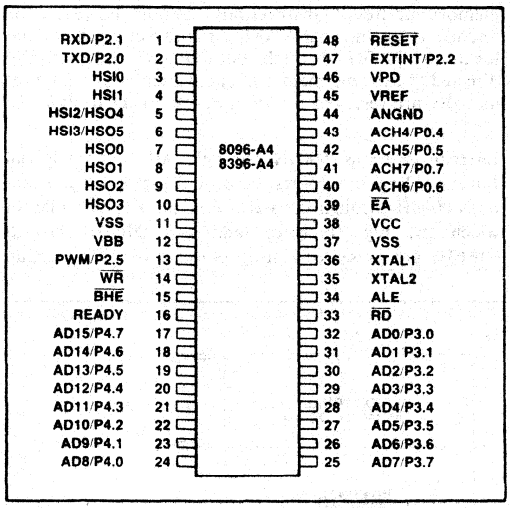


Figure 2. 48-Pin Package

Figure 1 shows the block diagram of the 8396. The 8096 differs only in not having on-chip ROM. Both devices are available in both 48-pin and 68-pin versions. Both devices are available with and without the on-chip A/D Converter. The iACX-96 numbering system is as shown below:

48-Pin	68-Pin	Features	
8096-D4	8096-D6	ROMless	No A/D
8096-A4	8096-A6	ROMless	With A/D
8396-D4	8396-D6	8K-Byte ROM	No A/D
8396-A4	8396-A6	8K-Byte ROM	With A/D

Figures 2 and 3 show the pinouts for the 48- and 68-pin packages.

The microcomputer is organized internally in a double bus structure. Internal data transfers are handled by a 16-bit data bus (D Bus) and an 8-bit address bus (A Bus). The internal registers occupy addresses 0000H through 00FFH. Communication with memory locations above

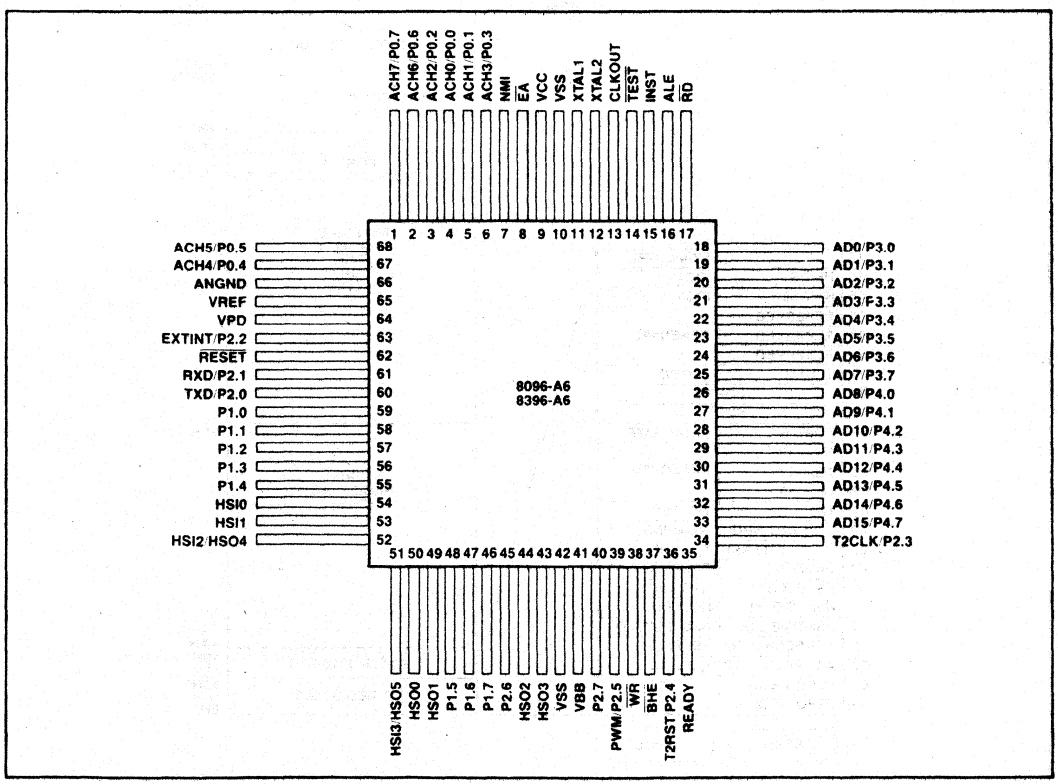


Figure 3. 68-Pin Package

00FFH is through the 8-bit bus and the Memory Controller. Thus the A Bus also conducts instruction and data bytes.

MEMORY SPACE

The 8096 uses the same address space for both program and data memory. It can execute instructions from any memory address. (Instructions cannot, however, be fetched from internal RAM. Instruction fetches from locations 0000H through 00FFH are automatically directed to external memory. These locations in external memory are reserved for Intel development tools.)

Internal locations 0000H through 0017H are Special Function Registers (ports, control registers, etc.). Locations 0018H through 00FFH access the Register File, of which the top 16 bytes (addresses 00F0H through 00FFH) are preserved during power down if a backup

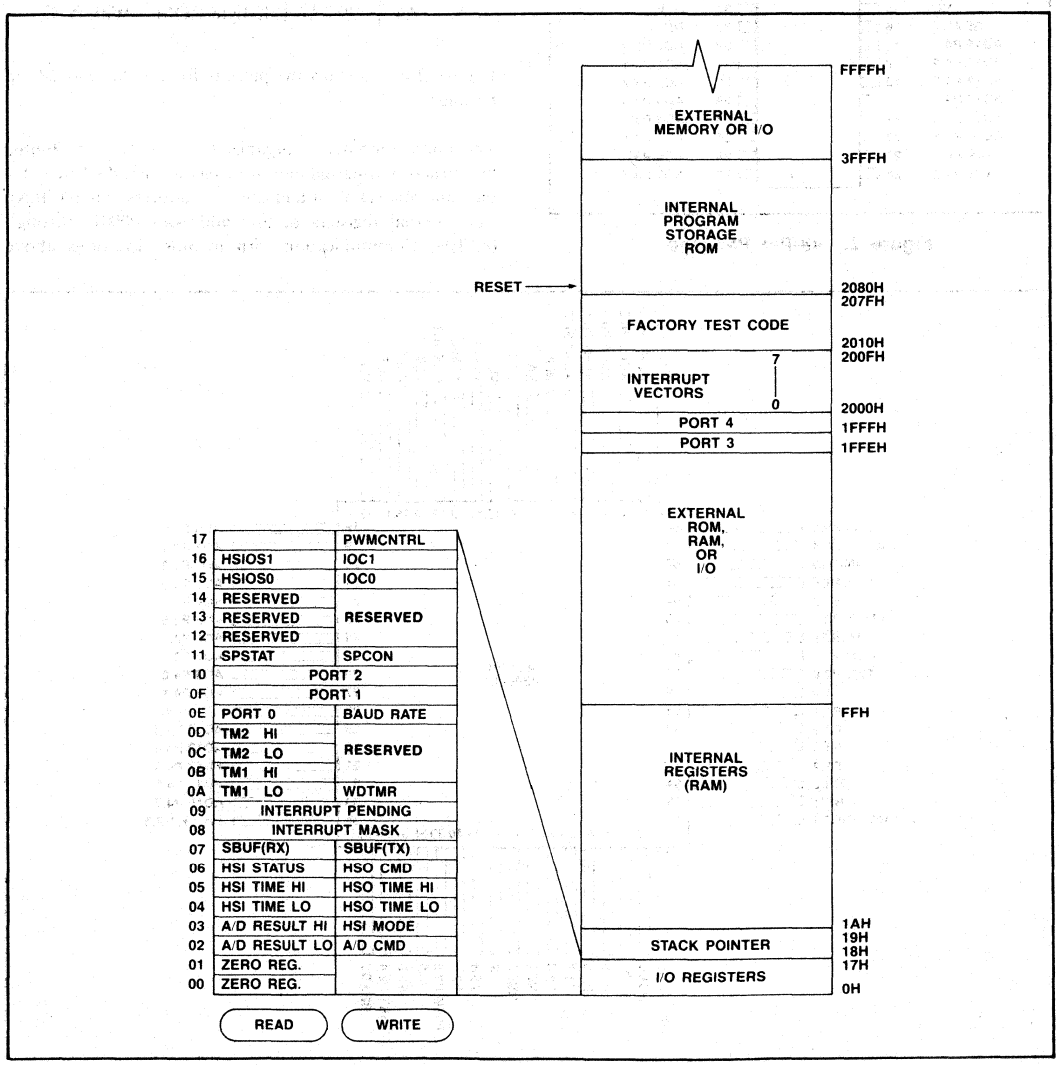


Figure 4. Memory Map

supply is applied to the VPD pin. Locations 0018H and 0019H contain the Stack Pointer.

The 8396 carries 8K bytes of on-chip ROM, occupying addresses 2000H through 3FFFH. These addresses access the on-chip ROM if the $\overline{EA}$ pin is externally held at a logical 1. Otherwise, the same addresses access off-chip memory. Addresses 2000H through 200FH contain the interrupt vectors. Addresses 2010H through 207FH contain a factory test code. Reset commences execution from location 2080H.

A memory map for the iACX-96 products is shown in Figure 4.

MEMORY CONTROLLER

Accesses to memory locations above 00FFH are through the Memory Controller.

Addresses for instruction fetches are maintained in the Slave Program Counter within the Memory controller. The Slave PC is automatically incremented after each fetch, and is updated by the CPU when program jumps are executed.

The Memory controller maintains a 3-byte queue of instruction bytes ready to feed to the Instruction Register as needed. Manipulations of the queue are automatic, and are taken into account in the state time listings.

Addresses for data accesses are assembled from the 8-bit bus into a 16-bit Data Address Register within the Memory Controller.

Accesses to off-chip memory are through the multiplexed address/data bus, which uses the output drivers and input buffers of Ports 3 and 4. (Accesses to Ports 3 and 4 themselves are also through the Memory Controller, at addresses 1FFEH and 1FFFH.)

CPU

The CPU consists of the Register File (with two 8-bit address registers for internal data transfers), the RALU (Register/ALU), and the Control Unit, as shown in Figure 1.

Register File

The Register File contains 232 bytes of RAM, which can be referenced as bytes, words, or double-words. This register space allows the user to keep the most frequently-used variables in on-chip RAM, which can be accessed more quickly than external memory.

RALU

The RALU (Register/ALU) section consists of a 17-bit ALU, the Program Status Word, the Program Counter, and several temporary registers, as shown in Figure 5.

A key feature of the 8096 is that it does not use an accumulator. Rather, it operates directly on any register in the Register File. Being able to operate directly on data in the Register File without having to move it into and out of an accumulator results in significant improvements in code length and execution speed.

The RALU provides hardware for multiply and divide.

Control Section

The Control Section consists of the Instruction Register and the PLA and associated circuitry which decodes the instructions and generates the correct sequence of control signals to execute instructions.

CLOCK GENERATOR

The Clock Generator contains an oscillator circuit to work with an external crystal (7.5 MHz to 12 MHz) connected between XTAL1 and XTAL2. The oscillator frequency is internally divided by 3 to provide an internal clock signal with a 33% duty cycle. In the 68-pin iACX-96 products, the internal clock signal is available for external use at CLKOUT.

The period of CLKOUT is referred to herein as the *state time*. One state time equals one CLKOUT period equals three oscillator periods.

HSIO UNIT

Timers

The HSIO Unit contains two 16-bit timer registers. The Timer 1 register free-runs, that is, it is automatically incremented every 8 state times (every 2.0 μsec, if the oscillator frequency is 12 MHz). It is cleared only by RESET.

The Timer 2 register, also 16 bits, can be programmed to

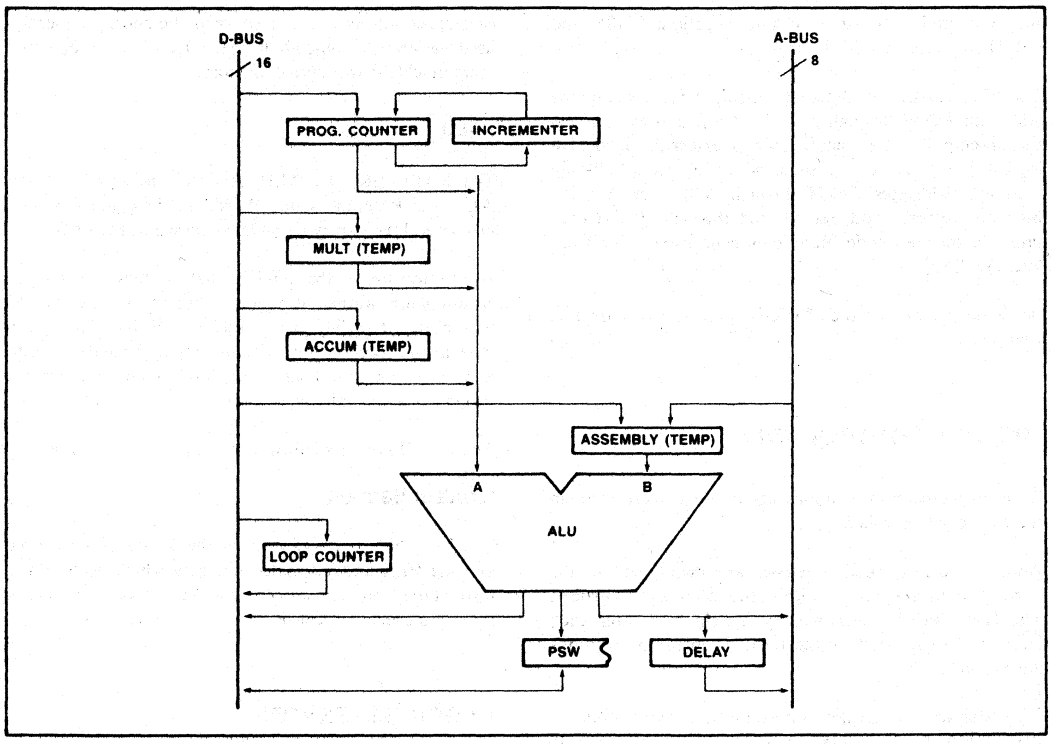


Figure 5. RALU

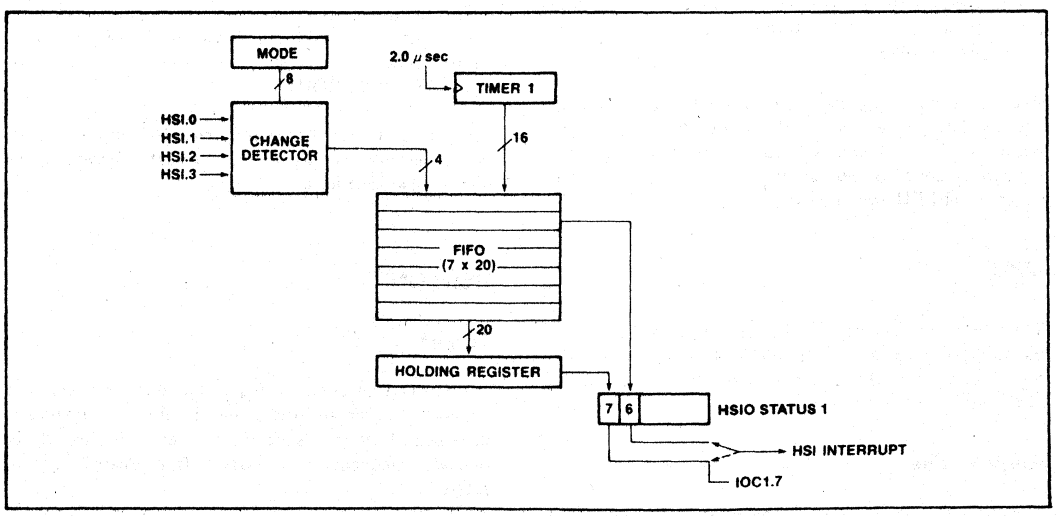


Figure 6. HSI Unit

count transitions on either T2CLK or HSI.1. The selected input to Timer 2 is sampled every 8 state times, and the count is incremented when the sample train shows a transition (*any* transition: 0-to-1 or 1-to-0) to have occurred. Timer 2 is cleared by RESET, and also by software, and can be programmed to be cleared by signals from either T2RST or HSI.0.

Either timer register can be programmed to generate an interrupt on overflow.

High-Speed Inputs

The HSI unit (Figure 6) looks for transitions in any of its input lines, and when it sees one it records the time (from Timer 1) and which input line made the transition. This information is stored in an 8-deep FIFO. The unit can be programmed to look for either positive or negative transitions, and can be programmed to activate the HSI Data Available interrupt either when the first entry to the FIFO has been made or when the 7th entry has been made.

The HSI unit has 4 input lines, HSI.0 through HSI.3. Two of these lines are shared with the High Speed Output unit. Each input line can be individually enabled or disabled to the HSI unit by software.

High Speed Outputs

The HSO unit (Figure 7) can be programmed to cause transitions on any of its output lines at specified times. The HSO unit has 6 output lines, HSO.0 through HSO.5. HSO.4 and HSO.5 are shared with the HSI unit as HSI.2 and HSI.3. These two outputs can be individually enabled or disabled by software.

The HSO unit can be programmed to set or clear any of its output lines, or reset Timer 2, or trigger an A/D conversion at the preset time. Either Timer 1 or Timer 2 can be selected to be the reference timer for this purpose. Up to 8 such preset actions can be in storage at any one time. As each action is carried out at its preset time, its command line is deleted from storage.

Storage of HSO commands is in an 8-deep CAM file (CAM = content addressable memory). Each CAM register is 23 bits wide. 16 bits specify the time at which the action is to be carried out, and 7 bits specify the nature of the action, and whether Timer 1 or Timer 2 is the reference.

To enter a command into the CAM file, one writes the 7-bit "Command Tag" into location 0006H and the time at which the action is to be carried out into word address 0004H. Writing to these addresses accesses the HSO Holding Register. The command does not actually enter the CAM file until an empty CAM register rotates around to the Holding Register (see Figure 7).

The CAM file rotates one register position per state time. Thus it takes 8 state times for the Holding Register to have had access to all 8 CAM registers. Similarly, it takes

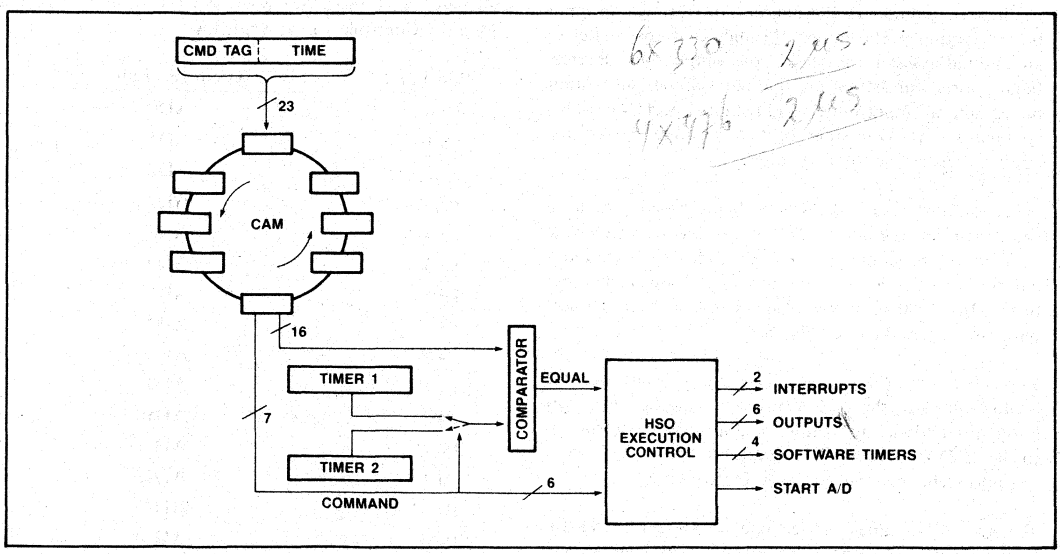


Figure 7. HSO Unit

8 state times for the comparator to have had access to all 8 CAM registers. This defines the time-resolution of the HSO unit to be 8 state times (2.0 μsec, if the oscillator frequency is 12 MHz).

Timer 1 is incremented only once every 8 state times, so if Timer 1 is being used as the reference timer for an HSO action, the comparator has a chance to look at all 8 CAM registers before Timer 1 changes its value. If Timer 2 is being used as the reference timer, the user must ensure that it is not cleared before all references to it in the CAM have been recognized. If this occurs, the unrecognized references will remain in the CAM, blocking further entries to those CAM registers.

Software Timers

The HSO can be programmed to generate interrupts at preset times. Up to four such "Software Timers" can be in operation at a time. As each preset time is reached, the HSO unit generates a Software Timer interrupt. The interrupt service routine can then examine an HSIO status register (HSIOS1) to determine which software timer expired and caused the interrupt.

The effect is essentially the same as having four additional 16-bit timers.

I/O PORTS

Input ports connect to the internal bus through an input buffer. Output ports connect through an output buffer to an internal register that holds the output bits. Bidirectional ports consist of an internal register, an output buffer, and an input buffer. Bidirectional ports are shown in Figure 1 as consisting of two separate blocks: the internal register and the I/O buffer.

When an instruction accesses a bidirectional port as a source register, the question often arises as to whether the value that is brought into the CPU comes from the internal port register or from the port pins through the input buffer. In the 8096, the value always comes from the port pins, never from the internal register.

Port 0 is an input-only port which shares its pins with the analog inputs to the A/D Converter. One can read Port 0 digitally and/or, by writing the appropriate control bits to the A/D Command Register, select one of the lines of this port to be the input to the A/D Converter.

Port 1 is a quasi-bidirectional I/O port. "Quasi-bidirectional" means the port pin has a weak internal pullup that is always active and an internal pulldown which can either be on (to output a 0) or off (to output a 1). If the internal pulldown is left off (by writing a 1 to the pin), the pin's logic level can be controlled by an external pulldown which can either be on (to input a 0) or off (to input a 1). From the user's point of view the main distinction is that a quasi-bidirectional input will source current while being externally held low.

In parallel with the weak internal pullup is a much stronger internal pullup that is activated for one state time when the pin is internally driven from 0 to 1. This is done to speed up the 0-to-1 transition time.

Port 2 is a multi-functional port. Six of its pins are shared with other functions in the 8096, as shown below:

Port Function		Alternate Function
P2.0	output	TXD (serial port transmit)
P2.1	input	RXD (serial port receive)
P2.2	input	EXTINT (external interrupt)
P2.3	input	T2CLK (Timer 2 input)
P2.4	input	T2RST (Timer 2 reset)
P2.5	output	PWM (pulse-width modulated signal)
P2.6	quasi-bidirectional	
P2.7	quasi-bidirectional	

Ports 3 and 4 are bidirectional with open-drain outputs in all pins. Also, these two ports share their pins with the Memory Controller, as shown below:

Port Pin	System Bus Function
P3.0	AD0
P3.1	AD1
P3.2	AD2
P3.3	AD3
P3.4	AD4
P3.5	AD5
P3.6	AD6
P3.7	AD7
P4.0	AD8
P4.1	AD9
P4.2	AD10
P4.3	AD11
P4.4	AD12
P4.5	AD13
P4.6	AD14
P4.7	AD15

When these pins are being used by the Memory Controller, they do have strong internal pullups. The internal pullups are only used during external memory read or write cycles when the pins are outputting address or data bits. At any other time, the internal pullups are disabled.

SERIAL PORT

The serial port is compatible with the MCS-51 serial port. It is full duplex, meaning it can transmit and receive simultaneously. It is also receive-buffered, meaning it can commence reception of a second byte before a previously received byte has been read from the receive register. The serial port registers are both accessed at the same address. A write to this location accesses the transmit register, and a read accesses a physically separate receive register.

The serial port can operate in 4 modes:

Mode 0: Mode 0 is a shift register mode. The 8096 outputs a train of 8 shift pulses to an external shift register to clock 8 bits of data into or out of the register from or to the 8096. Serial data enters and exits the 8096 through RXD. TXD outputs the shift signal. 8 bits are transmitted or received, LSB first. This mode is useful as an I/O expander, in which application external shift registers can be used as additional parallel I/O ports.

Mode 1: 10 bits are transmitted (through TXD) or received (through RXD): a start bit (0), 8 data bits (LSB first), and a stop bit (1).

Mode 2: 11 bits are transmitted (through TXD) or received (through RXD): a start bit (0), 8 data bits (LSB first), a programmable 9th data bit, and a stop bit (1). On transmit, the 9th data bit can be assigned the value of 0 or 1. On receive, the serial port interrupt is not activated unless the received 9th data bit is 1.

Mode 3: 11 bits are transmitted (through TXD) or received (through RXD): a start bit (0), 8 data bits (LSB first), a programmable 9th data bit, and a stop bit (1). On transmit, the 9th data bit can be assigned the value of 0 or 1. On receive, the received 9th data bit is stored, and the serial port interrupt is activated regardless of its value.

Mode 2 is provided for multi-processor communications. In this mode if the received 9th data bit is not 1, the serial port interrupt is not activated. The way to use this feature in multi-processor systems is as follows.

When the master processor wants to transmit a block of data to one of several slaves, it first sends out an address frame which identifies the target slave. An address frame will differ from a data frame in that the 9th data bit is 1 in an address frame and 0 in a data frame. No slave in mode 2 will be interrupted by a data frame. An address frame, however, will interrupt all slaves, so that each slave can examine the received byte and see if it is being addressed. The addressed slave switches to mode 3 to receive the coming data frames, while the slaves that weren't addressed stay in mode 2 and go on about their business.

Baud rates in all modes are determined by the contents of a 16-bit register, which is loaded sequentially as 2 bytes. The MSB of this register selects one of two sources for the input frequency to the baud rate generator: the oscillator frequency or an external frequency from the T2CLK pin. The unsigned integer represented by the remaining 15 bits of the baud rate register, plus 1, defines a number B, where B can thus take any value from 1 to 32768. The baud rate for the 4 serial modes is then given by

Mode 0: baud rate = input frequency/(4*B)

Others: baud rate = input frequency/(64*B)

PULSE WIDTH MODULATOR

The PWM output shares a pin with port bit P2.5. A bit in control register IOC1 selects the function of this pin to be either PWM or P2.5. When PWM is selected, this pin outputs a pulse train having a fixed period of 256 state times, and a programmable width of 0 to 255 state times. The width is programmed by loading the desired value, in state times, to the PWM Control Register.

A/D CONVERTER

The analog-to-digital converter is a ratiometric, 10-bit, successive approximation converter. This type of converter has a fixed conversion time, in this case 168 state times, independent of the value of the analog input. The analog input must be in the range of 0 to VREF (VREF = 5 ± .1V). The A/D result is related to VREF as follows:

A/D result = 1024 * Vin/VREF

This input can be selected from 8 analog input lines, which connect to the same pins as Port 0. A conversion can be initiated either by setting a control bit in the A/D Command register, or by programming the HSO unit to trigger the conversion at some specified time.

INTERRUPTS

The interrupt structure for the 8096 is shown in Figure 8.

The 8096 has 8 interrupt sources. A 0-to-1 transition from any of the sources sets a corresponding bit in the Interrupt Pending register. The content of the Interrupt Mask register determines if a pending interrupt will be responded to or not. If it is to be responded to, the CPU pushes the current Program Counter onto the Stack and reloads it with the vector corresponding to the interrupt being responded to. The interrupt vectors are located in addresses 2000H through 200FH, as shown below:

Interrupt Source	Vector Location	
	high byte	*low byte*
EXTINT	200FH	200EH
Serial Port	200DH	200CH
Software Timers	200BH	200AH
HSI.0	2009H	2008H
High Speed Outputs	2007H	2006H
HSI Data Available	2005H	2004H
A/D Conversion Complete	2003H	2002H
Timer Overflow	2001H	2000H

Execution then vectors to the ISR (interrupt service routine). At least one instruction in the ISR will always be executed before a second interrupt can be responded to. Any ISR can itself be interrupted, but not before the first instruction of the current ISR has been executed. Normally, this first instruction is PUSHF (push flags). PUSHF does two things: First it pushes the current PSW onto the Stack, and then it clears the PSW. Clearing the PSW clears the Interrupt Mask register, in effect blocking further interrupts. In the next instruction, if the programmer so desires, interrupts from selected sources can be re-enabled by writing 1's to the Mask register. In this manner, the software determines the priority structure.

The ISR then would normally terminate with POPF (pop flags), which restores the original PSW, and RET, which

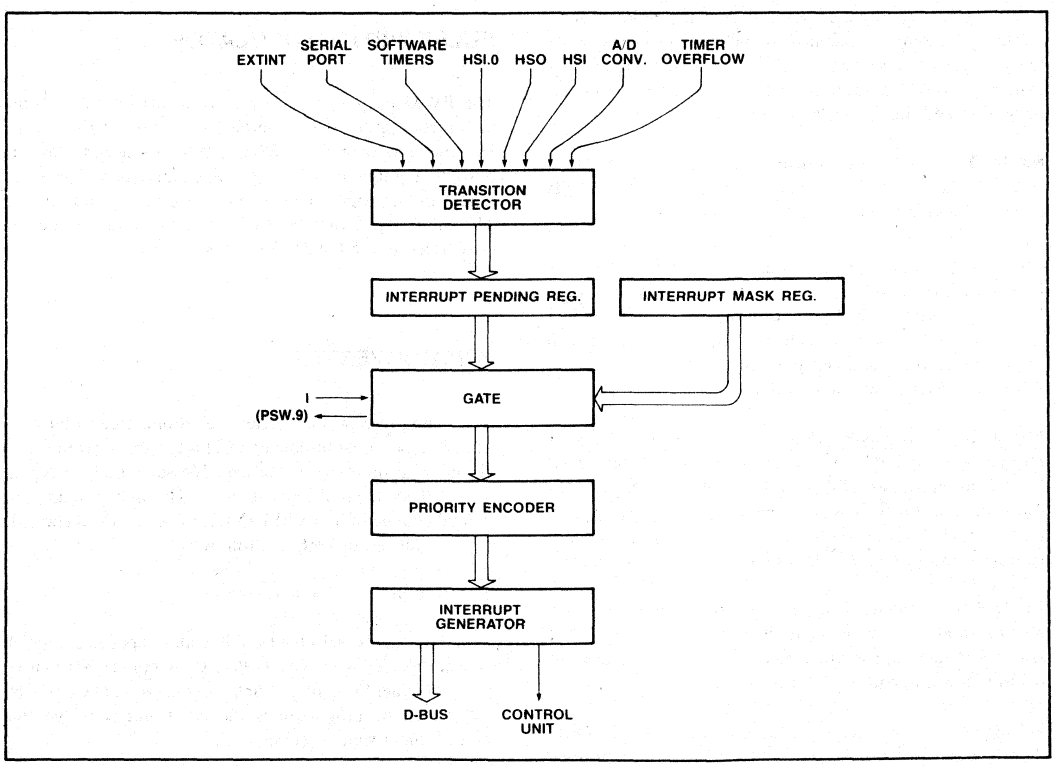


Figure 8. Interrupt Structure

restores the original Program Counter.

While the 8096 has only a single priority level in the sense that any interrupt may be itself interruptible, it does have a priority structure for resolving *simultaneous* interrupts, as follows:

Source	Priority
EXTINT	7 (highest)
Serial Port	6
Software Timers	5
HSI.0	4
High Speed Outputs	3
HSI Data Available	2
A/D Conversion Complete	1
Timer Overflow	0 (lowest)

WATCHDOG TIMER

The watchdog timer is a 16-bit counter which is incremented every state time. When it overflows it pulls down the $\overline{\text{RESET}}$ pin for two state times, which action reinitializes the system.

This feature is provided as a means of graceful recovery from a software upset. The way it's used is that the counter must be cleared by the software before it overflows, or else the system assumes an upset has occurred and activates $\overline{\text{RESET}}$.

To clear the watchdog timer, one writes 1EH to the WDT address (000AH), and then 0E1H to the same address.

The watchdog function can be disabled during program development by externally holding the $\overline{\text{RESET}}$ pin high. Using this technique for other than debugging purposes is not recommended, as it may affect long-term reliability.

RESET

To complete a reset, the $\overline{\text{RESET}}$ pin must be held active (low) for at least 2 state times. Then when $\overline{\text{RESET}}$ is brought high again the 8096 executes a reset sequence that takes 10 state times. (It clears the PSW and jumps to address 2080H.) Reset leaves the Special Function Registers as follows:

SFR	Reset Value
Port 1	0FFH
Port 2	110XXXX1B
Port 3	0FFH
Port 4	0FFH
PWM Control	00H
Serial Port (Transmit)	undefined
Serial Port (Receive)	undefined
Baud Rate Register	undefined
Serial Control/Status	undefined
A/D Command	undefined
A/D Result	undefined
Interrupt Pending	undefined
Interrupt Mask	00H
Timer 1	0000H
Timer 2	0000H
Watchdog Timer	0000H
HSI Mode	00H
HSI Status	undefined
HSIOS0	00H
HSIOS1	00H
IOC0	00H
IOC1	00H

Other conditions following reset are as follows:

HSO CAM	empty
HSI FIFO	empty
PSW	0000H
Stack Pointer	undefined
Program Counter	2080H

A low-to-high transition in $\overline{\text{RESET}}$ causes the 8096 to synchronize itself such that the rising edge of CLKOUT occurs 4 oscillator periods after $\overline{\text{RESET}}$ goes high.

POWER DOWN

During normal operation the VPD pin must be held within the same specifications as VCC. In a Power Down condition (i.e. VCC drops to zero), if $\overline{\text{RESET}}$ is activated before VCC drops below spec and VPD continues to be held within spec by a backup supply, the top 16 bytes in the Register File will retain their contents, drawing power from VPD. $\overline{\text{RESET}}$ must be held low during the Power Down and should not be brought high until VCC is back up and the oscillator has stabilized.

SPECIAL FUNCTION REGISTERS

PWM Control Register

The width of the PWM output pulse is programmed by writing the desired value, in state times (0 to 0FFH) into this register.

Serial Port Control/Status Register

Bit:	7	6	5	4	3	2	1	0
	RB8/RPE	RI	TI	TB8	REN	PEN	M2	M1

where: M1,M2 specifies the Mode;
PEN enables the parity function (even parity);
REN enables the receive function;
TB8 programs the 9th data bit (if not parity) on transmission;
TI is the transmit interrupt flag;
RI is the receive interrupt flag;
RB8 is the 9th data bit received (if not parity);
RPE is the parity error indicator (if parity active).

A/D Command Register

Bit:	7	6	5	4	3	2	1	0
	X	X	X	X	GO	.... channel #		

where: channel # selects which of the 8 analog input channels is to be converted to digital form;
GO indicates when the conversion is to be initiated (GO = 1 means start now, GO = 0 means the conversion is to be initiated by the HSO unit at a specified time).

A/D Result Register

Bit:	15	6	5	4	3	2	1	0
	MSB.....LSB		X	X	S	... channel # ...		
	(10-bit A/D Result)							

where: channel # tells which of the 8 analog input channels this was converted from;
S is a STATUS bit (S = 1 means an A/D conversion is currently under way, S = 0 means the A/D unit is currently idle).

Because of the way the A/D Result Register is implemented on the chip, it must be read as two separate bytes.

Interrupt Registers

The Interrupt Pending and Mask Registers each contain one bit for each of the 8 interrupt sources: Bit 7 is for the highest priority source (EXTINT) and bit 0 is for the lowest priority source (Timer Overflow). The Interrupt Mask Register is the low byte of the PSW.

HSI Mode Register

As previously described, the HSI unit looks for transitions of a desired type in its input lines, and when it sees one it records the time (from Timer 1) and in which input line the transition occurred.

The HSI Mode register defines what kinds of input transitions cause the contents of Timer 1 to be captured into the HSI FIFO, as follows:

Bit:	7	6	5	4	3	2	1	0
	/ HSI.3 Mode /		HSI.2 Mode /		HSI.1 Mode /		HSI.0 Mode /	

Each 2-bit mode control field defines one of 4 possible modes:

0 0	8 positive transitions
0 1	positive transitions
1 0	negative transitions
1 1	any transition

High and low levels need to hold for at least 1 state time or they may be missed.

HSI Status Register

The HSI Status register tells which input made a targeted transition and what its current state is (0 or 1), as follows:

Bit:	7	6	5	4	3	2	1	0
	/ HSI.3	/	HSI.2	/	HSI.1	/	HSI.0	/

For each 2-bit field, the lower bit indicates whether or not the transition has been made at this input, and the upper bit gives the current state at that pin.

HSO Command Tag

The HSO unit can be programmed to cause transitions in any of its output lines at specified times. The specified time can be taken from either Timer 1 or Timer 2. The unit has 6 output lines, but 2 of them (HSO.4 and

HSO.5) are shared by the HSI unit. The unit can initiate a number of other actions at specified times: initiate an A/D conversion, reset Timer 2, or indicate a software timer timeout. It can initiate these actions with or without an accompanying interrupt.

The HSO Command register is as follows:

Bit:	7	6	5	4	3	2	1	0
	X	T	1/0	I	. . . . . channel #			

where: channel # defines the recipient of the action;
I says whether or not the action is to be accompanied by an interrupt;
1/0 says whether the action on an output line is to be set to 1 or to 0;
T selects whether the "specified time" is to be from Timer 1 (T = 0) or Timer 2 (T = 1).

The channel # codes are as follows:

Channel #	Recipient of Action
0-5H	Output lines HSO.0-HSO.5
6H	Output lines HSO.0 and HSO.1 combined
7H	Output lines HSO.2 and HSO.3 combined
8H-0BH	Software Timers ST0-ST3 (the action being to establish it)
0EH	Timer 2 (the action being to reset it)
0FH	A/D Converter (the action being to initiate a conversion)

I/O Control Registers

There are two I/O Control registers, IOC0 and IOC1. IOC0 conditions Timer 2 and the HSI lines. IOC1 selects some pin functions and enables or disables some interrupt sources.

Timer 2 can count one of two selectable clock sources, T2CLK or HSI.1. It can be reset by one of two selectable hardware sources (T2RST or HSI.0) or by timed software (through the HSO unit) or by immediate software (through IOC0 bit 1). The selected hardware reset source pin can be enabled or disabled to this function.

The four HSI lines can be enabled or disabled to the HSI unit by setting or clearing bits in IOC0.

The bits of IOC0 are as follows:

Bit	Control Function
0	Set to 1 to enable HSI.0 to the HSI function. Clear to 0 to disable HSI.0 to the HSI function. (It can still be used to reset Timer 2, however.)
1	Writing a 1 to this bit resets Timer 2 every time.
2	Set to 1 to enable HSI.1 to the HSI function. Clear to 0 to disable HSI.1 to the HSI function. (It can still be used to clock Timer 2, however.)
3	Set to 1 to enable the external Timer 2 reset source (T2RST or HSI.0). Clear to disable same.
4	Set to 1 to enable HSI.2 to the HSI function.
5	Select external Timer 2 reset source: 0 selects T2RST; 1 selects HSI.0.
6	Set to 1 to enable HSI.3 to the HSI function.
7	Select Timer 2 clock source: 0 selects T2CLK; 1 selects HSI.1

IOC1 selects some pin functions and enables or disables some interrupt sources. Port pin P2.5 can be selected to be the PWM output. The external interrupt source can be selected to be either EXTINT (same pin as P2.2) or Analog Channel 7 (ACH7, same pin as P0.7). Timer 1 and Timer 2 overflow interrupts can be individually enabled or disabled. The HSI interrupt can be selected to activate either when there is 1 FIFO entry or 7. Port pin P2.0 can be selected to be the TXD output. HSO.4 and HSO.5 can be enabled or disabled to the HSO unit.

The bits of IOC1 are as follows:

Bit	Control Function
0	Select pin function: If this bit is 1, P2.5 becomes PWM.
1	Select external interrupt source: 0 selects EXTINT; 1 selects ACH7 (P0.7).
2	Set to 1 to enable Timer 1 overflow interrupt. Clear to disable same.
3	Set to 1 to enable Timer 2 overflow interrupt. Clear to disable same.
4	Set to 1 to enable HSO.4 to the HSO function.
5	Select pin function: If this bit is 1, P2.0 becomes TXD.
6	Set to 1 to enable HSO.5 to the HSO function.
7	HSI interrupt option: 0 selects to activate interrupt at first entry to FIFO. 1 selects to activate on 7th entry.

HSIO Status Registers

There are two HSIO Status registers. One gives the current status of the HSO lines and CAM. The other gives the overflow status of the 4 software timers, and of Timers 1 and 2, and indicates the status of the HSI FIFO.

The bits of HSIOS0 are as follows:

Bit	Function
0	Current state of HSO.0
1	Current state of HSO.1
2	Current state of HSO.2
3	Current state of HSO.3
4	Current state of HSO.4
5	Current state of HSO.5
6	A 0 indicates that the HSO Holding Register is empty and there is at least one empty register in the CAM.
7	A 0 indicates that the HSO Holding Register is empty.

The bits of HSIOS1 are as follows:

Bit	Function
0	A value 1 indicates Software Timer 0 has expired.
1	A value 1 indicates Software Timer 1 has expired.
2	A value 1 indicates Software Timer 2 has expired.
3	A value 1 indicates Software Timer 3 has expired.
4	A value 1 indicates Timer 2 has overflowed.
5	A value 1 indicates Timer 1 has overflowed.
6	A value 1 indicates the HSI FIFO contains at least seven entries.
7	A value 1 indicates the HSI FIFO contains at least one entry.

PIN DESCRIPTION

VCC

Main supply voltage (5V).

VSS

Digital circuit ground (0V).

VPD

RAM standby supply voltage (5V). This voltage must be present during normal operation.

VREF

Reference voltage for the A/D converter (5V). VREF is also the supply voltage to the analog portion of the A/D converter.

ANGND

Reference ground for the A/D converter. Should be held at nominally the same potential as VSS.

VBB

Substrate voltage from the on-chip back-bias generator. This pin should be connected to ANGND through a 0.01 μf capacitor (and not connected to anything else).

XTAL1

Input of the oscillator inverter.

XTAL2

Output of the oscillator inverter, and input to the internal clock generator.

CLKOUT

Output of the internal clock generator. The frequency of CLKOUT is 1/3 the oscillator frequency. It has a 33% duty cycle. CLKOUT can drive one S TTL input.

$\overline{\text{RESET}}$

Reset input to the chip. Input low for at least 2 state times to reset the chip. The subsequent low-to-high transition resynchronizes CLKOUT and commences a 12-state-time sequence in which the PSW is cleared and a jump to address 2080H is executed. Input high for normal operation. $\overline{\text{RESET}}$ has a strong internal pullup. This pin can also be used as an active low output with the RST instruction.

$\overline{\text{TEST}}$

Input low enables a factory test mode. The user should tie this pin to VCC for normal operation.

NMI

Input high disables internal interrupts, resets the watchdog timer, and causes a vector to *external* memory location 0000H. External memory locations 0000H through 00FFH are reserved for Intel development tools.

INST

Output high during an external memory read indicates the read is an instruction fetch.

$\overline{EA}$

Input for memory select (External Access). $\overline{EA}$ = 1 causes memory accesses to locations 2000H through 3FFFH to be directed to on-chip ROM. $\overline{EA}$ = 0 causes accesses to these locations to be directed to off-chip memory. $\overline{EA}$ has an internal pulldown, so it goes to 0 unless driven to 1.

ALE

Address Latch Enable output. ALE is activated only during external memory accesses. It is used to latch the address from the multiplexed address/data bus. ALE can drive one S TTL input.

$\overline{RD}$

Read signal output to external memory. $\overline{RD}$ is activated only during external memory reads. $\overline{RD}$ can drive one S TTL input.

$\overline{WR}$

Write signal output to external memory. $\overline{WR}$ is activated only during external memory writes. $\overline{WR}$ can drive one S TTL input.

$\overline{BHE}$

Bus High Enable signal output to external memory. $\overline{BHE}$ = 0 selects the bank of memory that is connected to the high byte of the data bus. A0 = 0 selects the bank of memory that is connected to the low byte of the data bus. Thus accesses to a 16-bit-wide memory can be to the low byte only (A0 = 0, $\overline{BHE}$ = 1), to the high byte only (A0 = 1, $\overline{BHE}$ = 0), or to both bytes (A0 = 0, $\overline{BHE}$ = 0). $\overline{BHE}$ is activated by the Bus Interface Unit when required during accesses to external memory. $\overline{BHE}$ can drive one S TTL input.

READY

The READY input is used to lengthen external memory bus cycles, for interfacing to slow or dynamic memory, or for bus sharing. READY is sampled by the Memory Controller at the negative transition in CLKOUT while $\overline{RD}$ or $\overline{WR}$ is active during external memory cycles. If the pin is high (READY = 1), CPU operation continues in a normal manner. If the pin is low (READY = 0) when sampled, the Memory Controller goes into a wait mode until the next negative transition in CLKOUT, at which time READY is sampled again. The bus cycle can be lengthened by up to 1 μsec. When the external memory bus is not being used, READY has no effect. READY has a weak internal pullup, so it goes to 1 unless externally pulled down.

HSI

High impedance inputs to HSI Unit. Four HSI pins are available: HSI.0, HSI.1, HSI.2, and HSI.3. Two of them (HSI.2 and HSI.3) are shared with the HSO Unit.

HSO

Outputs from HSO Unit. Six HSO pins are available: HSO.0, HSO.1, HSO.2, HSO.3, HSO.4, and HSO.5. Two of them (HSO.4 and HSO.5) are shared with the HSI Unit. All HSO pins are capable of driving one S TTL input.

Port 0

High impedance input-only port. These pins can be used as digital inputs and/or as analog inputs to the on-chip A/D converter.

Port 1

Quasi-bidirectional I/O port. All pins of P1 are capable of driving one LS TTL input.

Port 2

Multi-functional port. Six of its pins are shared with other functions in the 8096, as shown below:

Port Function		Alternate Function
P2.0	output	TXD (serial port transmit)
P2.1	input	RXD (serial port receive)
P2.2	input	EXTINT (external interrupt)
P2.3	input	T2CLK (Timer 2 input)

Port Function		Alternate Function
P2.4	input	T2RST (Timer 2 reset)
P2.5	output	PWM (pulse-width modulated signal)
P2.6	quasi-bidirectional	
P2.7	quasi-bidirectional	

The quasi-bidirectional pins can drive one LS TTL input. The multi-functional outputs can drive one S TTL input. The multi-functional inputs are high impedance.

Ports 3 and 4

8-bit bidirectional I/O ports with open drain outputs. These pins are also the multiplexed address/data bus when accessing external memory, as shown below:

Port Pin	System Bus Function
P3.0	AD0
P3.1	AD1
P3.2	AD2
P3.3	AD3
P3.4	AD4
P3.5	AD5
P3.6	AD6
P3.7	AD7
P4.0	AD8
P4.1	AD9
P4.2	AD10
P4.3	AD11
P4.4	AD12
P4.5	AD13
P4.6	AD14
P4.7	AD15

When these pins are being used by the Memory Controller, they do have strong internal pullups. The internal pullups are only used during external memory read or write cycles when the pins are outputting address or data bits, in which application they can drive one S TTL input. At any other time, the internal pullups are disabled.

When these pins are being used as output ports, they can drive one LS TTL input. As input ports, they are high impedance.

INSTRUCTION SET

Program Status Word (PSW)

The low byte of the PSW is the Interrupt Mask register. The high byte is as follows:

Bit:	15	14	13	12	11	10	9	8
	Z	N	V	VT	C	(X)	I	ST

- Z — Zero bit. Z = 1 means the result of the previous operation was zero. (ADDC and SUBC can clear Z, but they can't set it.)
- N — Negative bit. N = 1 means the algebraically correct result is negative. (The actual result as stored may have an incorrect sign due to an overflow, but N reflects the algebraically correct sign.)
- V — Overflow bit. V = 1 means the magnitude of the algebraically correct result is too large to be correctly represented in the available space.
- VT — Overflow Trap bit. Any time V gets set, VT gets set too. Subsequent operations may clear V, but VT can only be cleared by software.
- C — Carry bit. This is the carry out of the MSB of the result of any arithmetic operation. During left-shifts, the MSB moves into C and a 0 moves into the LSB. During right-shifts, the LSB moves into C and a 0 moves into the MSB.
- (X) — Unused. (Reads 0.)
- I — Interrupt enable bit. I = 0 disables all interrupts.
- ST — Sticky Bit. During right-shifts the LSB moves into C. If the previous value of C was 1, ST gets set (see SHR and SHRA instructions.).

Word Operations

In the following descriptions, the symbols A, B, and D represent word operands except where noted. Word addresses must be even numbers.

B and D must be located in the Register File. A can be located anywhere in memory.

A and B are source registers. D is the destination register.

ARITHMETIC WORD OPERATIONS

Mnemonic	*Meaning*	*Operation*	*Explanation/ Rules*
ADD	Add 2 words.	D = D + A D = B + A	
ADDC	Add with carry.	D = D + A + C	C is the carry bit.
SUB	Subtract one word from another.	D = D - A D = B - A	
SUBC	Subtract with carry.	D = D - A - 1 + C	C is the carry bit.
MUL	Multiply 2 words to form a double-word product.	D, D + 2 = D*A D, D + 2 = B*A	The address of the product must be a legal double-word address; that is, it must be evenly divisible by 4.
MULU	Multiplying unsigned.	D, D + 2 = D*A D, D + 2 = B*A	Same as MUL, except operands are treated as unsigned integers.
DIV	Divide a double-word integer by a single-word integer.	D = (D, D + 2) / A D + 2 = remainder	The address of D must be a valid double-word address; that is, it must be evenly divisible by 4.
DIVU	Divide unsigned.	D = (D, D + 2) / A D + 2 = remainder	Same as DIV, except operands are treated as unsigned integers.
INC	Increment word.	D = D + 1	
DEC	Decrement word.	D = D - 1	
CMP	Compare two words.	B - A	The result of the subtraction (B - A) is not stored. Only the PSW is affected.
NEG	Negate word.	D = - D	Changes the sign of the addressed word.
EXTB	Extend byte to word.	D = D D + 1 = SIGN(D)	Sign-extends the addressed byte to word-length. (Hence the address of D must be a valid word-address.)
EXT	Extend word to double-word.	D = D D + 2 = SIGN(D)	Sign-extends the addressed word to double-word length. (Hence the address of D must be a valid double-word address.)

LOGICAL WORD OPERATIONS

Mnemonic	*Meaning*	*Operation*
AND	Logical AND.	D = D AND A D = B AND A
OR	Logical OR.	D = D OR A
XOR	Exclusive OR.	D = D XOR A
NOT	Logical NOT.	D = NOT(D)
CLR	Clear word.	D = 0000H

DATA TRANSFER

Mnemonic	*Meaning*	*Operation*	*Explanation/ Rules*
LD	Load word.	D = A	Both operands are words.
ST	Store word.	A = B	Both operands are words. B must be in the Register File. In this case A, which can be anywhere in memory, is the destination register.
LDBSE	Load byte sign-extended to word.	D = A D + 1 = SIGN(A)	D is a word. A is a byte.
LDBZE	Load byte zero-extended to word.	D = A D + 1 = 00H	D is a word. A is a byte.
PUSH	Push word.	SP = SP - 2 (SP) = A	Notice the stack grows *down,* not up.
POP	Pop word.	A = (SP) SP = SP + 2	
PUSHF	Push PSW.	SP = SP - 2 (SP) = PSW PSW = 0000H	Note that PUSHF clears the PSW after copying it into the stack. The effect is to disable all interrupts.
POPF	Pop PSW.	PSW = (SP) SP = SP + 2	

Shift Word Operations

Shift instructions have an 8-bit field, N, which indicates the number of places the word is to be shifted. If (N) $<$16, (N) is the shift count. If (N) $\geq$ 16, (N) is the address of the shift count. All shift operands must be in the Register File.

Mnemonic	*Meaning*	*Explanation/ Rules*
SHR	Shift right N places.	Each shift brings a 0 in from the left (into the MSB) and moves the LSB into C. The previous value of C is lost. If C loses any 1s during this series of shifts (except for the first shift), the ST bit (PSW.8) is set. Otherwise, ST = 0.
SHL	Shift left N places.	Each shift moves the MSB into C and a 0 in from the right (into the LSB). The previous value of C is lost. ST is unaffected.
SHRA	Shift right N places, with arithmetic sign.	Each shift brings a 0 (if the MSB is 0) or a 1 (if the MSB is 1) in from the left and moves the LSB into C. The previous value of C is lost. If C loses any 1s during this series of shifts (except for the first shift), the ST bit (PSW.8) is set. Otherwise, ST = 0. The shifted word has the same arithmetic sign as it did before the shift.

Byte and Double-Word Operations

A number of the previously described word instructions actually involve some byte and double-word variables, as well. For example, EXTB involves a byte operand, and MUL, MULU, DIV, DIVU, and EXT involve double-word operands. In addition, most of the arithmetic, logical, and data transfer instructions previously described for word operands have specifically byte versions, as listed below.

ARITHMETIC BYTE OPERATIONS

Mnemonic	*Meaning*	*Operation*	*Explanation/ Rules*
ADDB	Add 2 bytes.	D = D + A D = B + A	
ADDCB	Add with carry.	D = D + A + C	C is the carry bit (PSW.11).
SUBB	Subtract one byte from another.	D = D - A D = B - A	
SUBCB	Subtract with carry.	D = D - A - 1 + C	C is the carry bit (PSW. 11).
MULB	Multiply 2 bytes to form a double-byte (word) product.	D, D + 1 = D*A D, D + 1 = B*A	The address of the product must be a valid word address; that is, it must be even.
MULUB	Multiply unsigned.	D, D + 1 = D*A D, D + 1 = B*A	Same as MULB, except operands are treated as unsigned integers.
DIVB	Divide a word-integer by a byte-integer.	D = (D, D + 1)/A D + 1 = remainder	The address of D must be a valid word address; that is, it must be even.
DIVUB	Divide unsigned.	D = (D, D + 1)/A D + 1 = remainder	Same as DIVB, except operands are treated as unsigned integers.
INCB	Increment byte.	D = D + 1	
DECB	Decrement byte.	D = D - 1	
CMPB	Compare two bytes.	B - A	The result of the subtraction (B-A) is not stored. Only the PSW is affected.
NEGB	Negate byte.	D = -D	Changes the sign of the addressed byte.
EXTB	Extend byte to word.	D = D D + 1 = SIGN(D)	Sign-extends the addressed byte to word-length. (Hence the address of D must be a valid word address.)

LOGICAL BYTE OPERATIONS

Mnemonic	*Meaning*	*Operation*
ANDB	Logical AND.	D = D AND A D = B AND A
ORB	Logical OR.	D = D OR A
XORB	Exclusive OR.	D = D XOR A
NOTB	Logical NOT.	D = NOT(D)
CLRB	Clear byte.	D = 00H

DATA TRANSFER

Mnemonic	*Meaning*	*Operation*	*Explanation/ Rules*
LDB	Load byte.	D = A	Both operands are bytes.
STB	Store byte.	A = B	Both operands are bytes. In this case A, which can be anywhere in memory, is the destination register.
LDBSE	Load byte sign-extended to word.	D = A D + 1 = SIGN(A)	D is a word. A is a byte.

Mnemonic	*Meaning*	*Operation*	*Explanation/Rules*
LDBZE	Load byte zero-extended to word.	D = A D + 1 = 00H	D is a word. A is a byte.

Conspicuously absent from the byte-wide data transfer operations are PUSH and POP instructions. PUSH and POP are exclusively word operations. (Hence the Stack Pointer should *always* contain a valid word address, that is, it should be an even number.)

Byte and Double-Word Shift Operations

All of the word shift instructions have byte and double-word versions, as listed below. Their meanings and explanation are exactly the same as for the corresponding word shift instruction.

SHIFT OPERATIONS

Word	*Byte*	*Double-Word*
SHR	SHRB	SHRL
SHL	SHLB	SHLL
SHRA	SHRAB	SHRAL

There is one type of shift instruction that is exclusively for double-word operands:

Mnemonic	*Meaning*	*Explanation/Rules*
NORML	Normalize.	The (double-word) operand is left-shifted until the MSB is 1. The number of shifts required to do that is recorded in the Register File at a specified address. If the operand doesn't contain any 1s, the shifting will cease after 31 counts and the Z flag will be set.

Program Branching Instructions

UNCONDITIONAL JUMPS AND CALLS

Mnemonic	*Meaning*	*Explanation/Rules*
SJMP	Short jump.	Jump to an address that is within (+1023, -1024) of the current PC.
LJMP	Long jump.	Jump to an address that is within (+32767, -32768) of the current PC.
INDJMP	Indirect jump.	The instruction specifies a word address in the Register File. The PC is reloaded with the contents of that word.
SCALL	Short call.	Pushes the PC, then executes an SJMP to the subroutine. (Used when the subroutine is within (+1023, -1024) of the current PC.)
LCALL	Long call.	Pushes the PC, then executes an LJMP to the subroutine.
RET	Return.	Pops the PC, so that execution will continue from where it left off when the subroutine or interrupt service routine was branched to.

The 8096 assembler recognizes these mnemonics (except INDJMP), but also recognizes a number of generic jump and call instructions. For example, the assembler mnemonic BR (branch) translates to SJMP, LJMP, or INDJMP as required.

LOOP CONTROL

Mnemonic	*Meaning*	*Explanation/Rules*
DJNZ	Decrement and jump if not zero.	Instruction specifies a byte in the Register File which is to be decremented. If the result is not 00H, execution jumps to a specified address. The range is (+127, –128).

The normal application for DJNZ is as a loop counter, to control the number of times a program loop is executed.

CONDITIONAL JUMP INSTRUCTIONS

Conditional jump instructions test bits or combinations of bits to see if certain conditions are met. If they're met, the jump is executed. Otherwise execution continues in the normal sequence.

Most of these instructions test flag bits in the PSW. The flag bits involved are:

PSW.15	Z	Zero bit
PSW.14	N	Negative bit
PSW.13	V	Overflow bit
PSW.12	VT	Overflow trap bit
PSW.11	C	Carry bit
PSW.8	ST	Sticky bit

Mnemonic	*Meaning*	*Bit(s) Tested*	*Explanation*
JBS	Jump if bit set.	B.x	B is any byte in the Register File or SFR space. x is any bit in B. The jump is executed if B.x = 1.
JBC	Jump if bit clear.	B.x	Same as JBS, except the jump is executed if B.x = 0.
JC	Jump on carry.	C	Jump if C = 1.
JNC	Jump on no carry.	C	Jump if C = 0.
JE	Jump if equal.	Z	Jump if Z = 1.
JNE	Jump if not equal.	Z	Jump if Z = 0.
JGE	Jump if greater than or equal to.	N	Jump if N = 0.
JLT	Jump if less than.	N	Jump if N = 1.
JGT	Jump if greater than.	N, Z	Jump if N and Z are *both* 0.
JLE	Jump if less than or equal to.	N, Z	Jump if N = 1 or Z = 1.
JH	Jump if higher.	C, Z	An unsigned version of JGT. Jump if C = 1 and Z = 0.
JNH	Jump if not higher.	C, Z	An unsigned version of JLE. Jump if C = 0 or Z = 1.
JV	Jump on overflow.	V	Jump if V = 1.
JNV	Jump on no overflow.	V	Jump if V = 0.
JVT	Jump on overflow trap.	VT	Jump if VT = 1. Clear VT.
JNVT	Jump on no overflow trap.	VT	Jump if VT = 0. Clear VT.
JST	Jump on sticky bit.	ST	Jump if ST = 1.
JNST	Jump on no sticky bit.	ST	Jump if ST = 0.

The range of these jumps is limited to (+127, –128). The 8096 assembler, in addition to recognizing these mnemonics, also recognizes generic conditional jumps (such as BBS for JBS), in which the jump destination can range over the entire 64K.

SPECIAL CONTROL INSTRUCTIONS

Mnemonic	*Meaning*	*Explanation/ Rules*
SETC	Set carry bit.	
CLRC	Clear carry bit.	
CLRVT	Clear VT bit.	
DI	Disable interrupts.	Clears I (Interrupt bit, PSW.9). I = 0 disables all interrupts.
EI	Enable interrupts.	Sets I.
RST	Reset.	Pulls the $\overline{\text{RESET}}$ pin low, triggering a reset.
NOP	No operation.	
SKIP	Skip.	Skips the next instruction byte. In effect, a two-byte NOP. The opcode for SKIP is 00H.

Notice the RST instruction. This instruction not only resets the CPU, but also activates the $\overline{\text{RST}}$ pin, so that external logic can also be reset. The opcode for the instruction was selected to be 0FFH, so that if a particularly harsh electrical transient (or a software error) causes the CPU to start executing nonimplemented memory, 0FFH will be read and the CPU will reset.

Addressing Modes

The addressing modes that are supported by the 8096 are as follows:

DIRECT

The operand is specified by an 8-bit address field in the instruction. The operand must be in the Register File or SFR space (locations 0000H through 00FFH).

IMMEDIATE

The operand itself follows the opcode in the instruction stream as immediate data. The immediate data can be either 8-bits or 16-bits, as required by the opcode.

INDIRECT

An 8-bit address field in the instruction gives the word address of a word register in the Register File which contains the 16-bit address of the operand. The operand can be anywhere in memory.

INDIRECT WITH AUTO-INCREMENT

Same as Indirect, except that, after the operand is referenced, the word register that contains the operand's address is incremented (by 1 if the operand is a byte, or by 2 if the operand is a word).

INDEXED

The instruction contains an 8-bit address field and either an 8-bit or a 16-bit displacement field. The 8-bit address field gives the word address of a word register in the Register File which contains a 16-bit base address. The 8- or 16-bit displacement field contains a signed displacement that will be added to the base address to produce the address of the operand. The operand can be anywhere in memory.

If the displacement field is 8 bits wide, the displacement value is sign-extended to 16 bits before being added to the base address.

The 8096 contains a Zero Register at word address 0000H (and which contains 0000H). This register is available for use as a base register in indexed addressing. This in effect provides direct addressing to all 64K of memory.

In the 8096, the Stack Pointer is at word address 0018H in the Register File. If the 8-bit address field contains 18H, the Stack Pointer becomes the base register. This allows direct accessing of variables in the stack.

ELECTRICAL CHARACTERISTICS

ABSOLUTE MAXIMUM RATINGS

Ambient Temperature Under Bias . . . . . 0°C to +70°C
Storage Temperature . . . . . -40°C to +150°C
Voltage from Any Pin to VSS or ANGND . . . -0.3V to +7.0V
Average Output Current from Any Pin . . . . . 10 mA
Power Dissipation . . . . . 1.5 Watts

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

OPERATING CONDITIONS

Symbol	Parameter	Min	Max	Units
TA	Ambient Temperature Under Bias	0	+70	C
VCC	Digital Supply Voltage	4.50	5.50	V
VREF	Analog Supply Voltage	4.9	5.1	V
fOSC	Oscillator Frequency	7.5	12	MHz
VPD	Power-Down Supply Voltage	4.50	5.50	V

VBB should be connected to ANGND through a 0.01 μF capacitor. ANGND and VSS should be nominally at the same potential.

DC CHARACTERISTICS

Symbol	Parameter	Min	Max	Units	Test Conditions
VIL	Input Low Voltage	-0.3	+0.8	V	
VIH	Input High Voltage	2.0	VCC+0.5	V	
VOL	Output Low Voltage		0.45	V	See Note 1.
VOH	Output High Voltage	2.4		V	See Note 2.
ICC	VCC Supply Current		200	mA	All outputs disconnected.
IPD	VPD Supply Current		1	mA	Normal operation and Power-Down.
IREF	VREF Supply Current		15	mA	
ILI	Input Leakage Current to all pins of HSI, P0, P3, P4, and to P2.1, P2.2, P2.3, and P2.4		±10	μA	Vin = 0 to VCC See Note 3
IIH	Input High Current to $\overline{EA}$		100	μA	VIH = 2.4V
IIL	Input Low Current to all pins of P1, and to P2.6, P2.7, and READY		-100	μA	VIL = 0.45V
IIL1	Input Low Current to $\overline{RESET}$		-2	mA	VIL = 0.45V
Cs	Pin Capacitance (Any Pin to VSS)		10	pF	fTEST = 1MHz

NOTES:

1. IOL = 0.36 mA for all pins of P1, for P2.6 and P2.7, and for all pins of P3 and P4 when used as ports.

   IOL = 2.0 mA for TXD, RXD (in serial port mode 0), PWM, CLKOUT, ALE, $\overline{BHE}$, $\overline{RD}$, $\overline{WR}$, and all pins of HSO and P3 and P4 when used as external memory bus (AD0-AD15).

2. IOH = -20 μA for all pins of P1, for P2.6 and P2.7.

   IOH = -50 μA for TXD, RXD (in serial port mode 0), PWM, CLKOUT, ALE, $\overline{BHE}$, $\overline{RD}$, $\overline{WR}$, and all pins of HSO and P3 and P4 when used as external memory bus (AD0-AD15).

   P3 and P4, when used as ports, have open-drain outputs.

3. Analog Conversion not in process.

A/D CONVERTER SPECIFICATIONS

A/D Converter operation is verified only in devices having the "-A4" or "-A6" suffix.

The conversion accuracy is dependent on the accuracy of VREF. The specifications given below assume adherence to the Operating Conditions section of these data sheets.

Resolution ..... ±0.001 VREF
Accuracy ..... ±0.004 VREF
Differential nonlinearity ..... ±0.002 VREF max
Integral nonlinearity ..... ±0.004 VREF max
Channel-to-channel matching ..... ±1 LSB
Crosstalk (DC to 100kHz) ..... -60dB max

AC CHARACTERISTICS

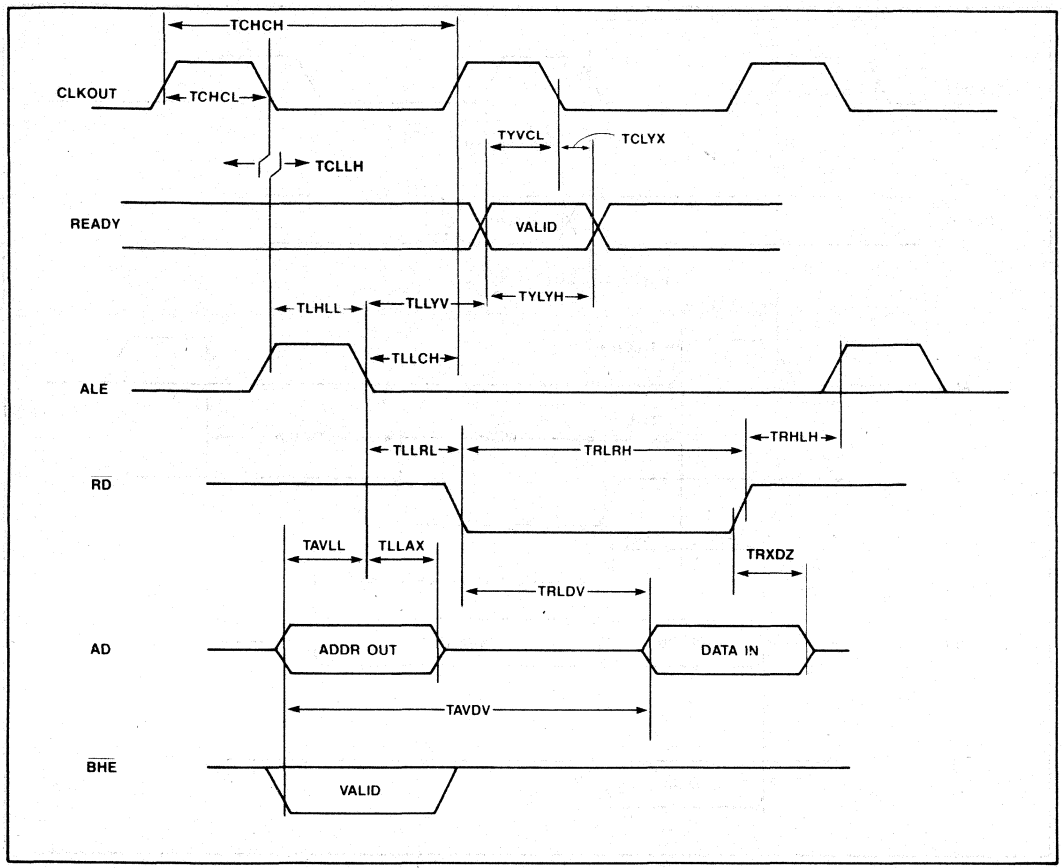
Test Conditions: Oscillator Frequency = 12MHz
Load Capacitance on Output Pins = 100pF
Other Operating Conditions as previously described.
Tosc = 1/(Oscillator Frequency)

Timing Requirements (Other system components must meet these specs.)

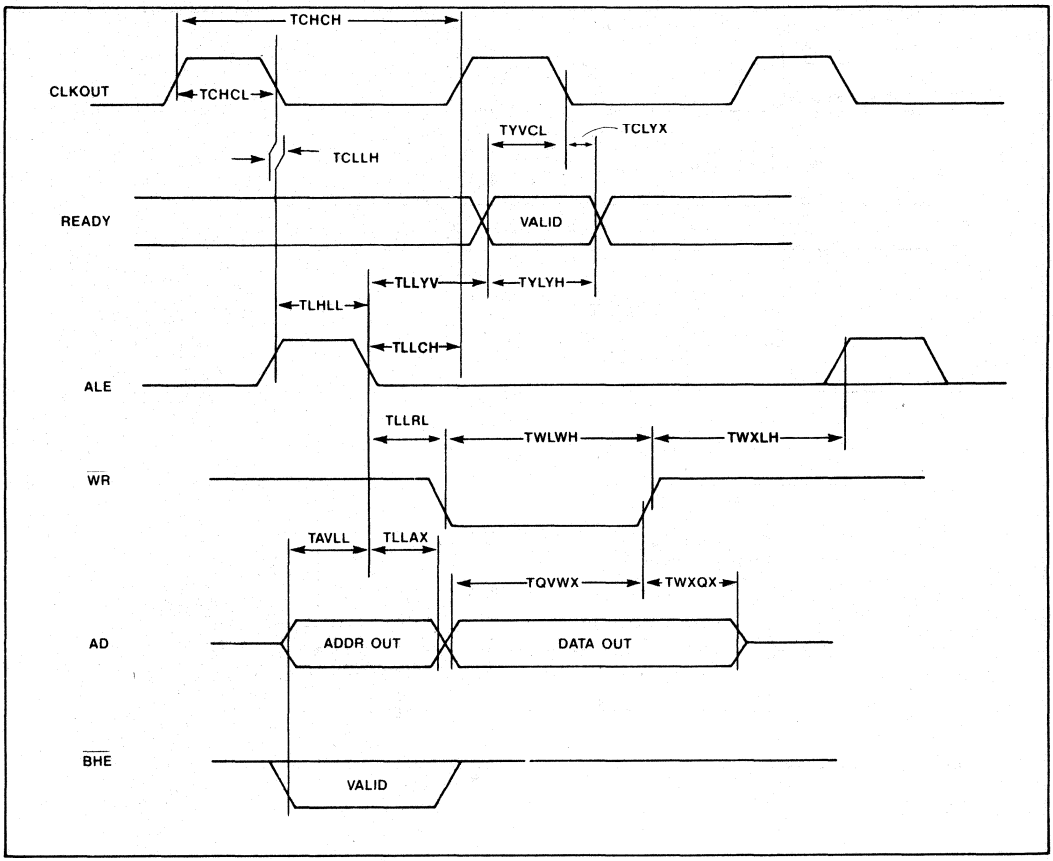
Symbol	Parameter	Min	Max	Units
TYVCL	READY Setup to CLKOUT Edge	40		nsec
TCLYX	READY Hold after CLKOUT Edge	0		nsec
TLLYV	End of ALE to READY Setup		2Tosc-50	nsec
TYLYH	Non-ready Time		1000	nsec
TAVDV	Address Valid to Input Data Valid		5Tosc-30	nsec
TRLDV	$\overline{RD}$ Active to Input Data Valid		3Tosc-50	nsec
TRXDZ	End of $\overline{RD}$ to Input Data Float	0	Tosc-5	nsec

Timing Responses (iACX-96 meets these specs.)

Symbol	Parameter	Min	Max	Units
TCHCH	CLKOUT Period	3Tosc	3Tosc	nsec
TCHCL	CLKOUT High Time	Tosc-20	Tosc	nsec
TCLLH	CLKOUT Low to ALE High	-5	+10	nsec
TLLCH	ALE Low to CLKOUT High	Tosc-20	Tosc+10	nsec
TLHLL	ALE Pulse Width	Tosc-10	Tosc	nsec
TAVLL	Address Valid to End of ALE	Tosc-20	Tosc+10	nsec
TLLRL	End of ALE to $\overline{RD}$ or $\overline{WR}$ Active	Tosc-20		nsec
TLLAX	End of ALE to Address Invalid	Tosc-20		nsec
TWLWH	$\overline{WR}$ Pulse Width	2Tosc-15		nsec
TQVWX	Output Data Valid to End of $\overline{WR}$	2Tosc-30		nsec
TWXQX	Output Data Hold after $\overline{WR}$	Tosc-25		nsec
TWXLH	End of $\overline{WR}$ to Next ALE	2Tosc-30		nsec
TRLRH	$\overline{RD}$ Pulse Width	3Tosc-30		nsec
TRHLH	End of $\overline{RD}$ to Next ALE	Tosc-10		nsec



External Memory Read Cycle



External Memory Write Cycle

Opcode and State Time Listing

MNEMONIC	OPERANDS	DIRECT			IMMEDIATE			INDIRECT* NORMAL			INDIRECT* AUTO-INC.			INDEXED* SHORT			INDEXED* LONG		
		OPCODE	BYTES	STATE① TIMES	OPCODE	BYTES	STATE① TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES	BYTES	STATE① TIMES
ARITHMETIC INSTRUCTIONS																			
ADD	2	64	3	4	65	4	5	66	3	6/11	3	7/12	67	4	6/11	5	7/12		
ADD	3	44	4	5	45	5	6	46	4	7/12	4	8/13	47	5	7/12	6	8/13		
ADDB	2	74	3	4	75	3	4	76	3	6/11	3	7/12	77	4	6/11	5	7/12		
ADDB	3	54	4	5	55	4	5	56	4	7/12	4	8/13	57	5	7/12	6	8/13		
ADDC	2	A4	3	4	A5	4	5	A6	3	6/11	3	7/12	A7	4	6/11	5	7/12		
ADDCB	2	B4	3	4	B5	3	4	B6	3	6/11	3	7/12	B7	4	6/11	5	7/12		
SUB	2	68	3	4	69	4	5	6A	3	6/11	3	7/12	6B	4	6/11	5	7/12		
SUB	3	48	4	5	49	5	6	4A	4	7/12	4	8/13	4B	5	7/12	6	8/13		
SUBB	2	78	3	4	79	3	4	7A	3	6/11	3	7/12	7B	4	6/11	5	7/12		
SUBB	3	58	4	5	59	4	5	5A	4	7/12	4	8/13	5B	5	7/12	6	8/13		
SUBC	2	A8	3	4	A9	4	5	AA	3	6/11	3	7/12	AB	4	6/11	5	7/12		
SUBCB	2	B8	3	4	B9	3	4	BA	3	6/11	3	7/12	BB	4	6/11	5	7/12		
CMP	2	88	3	4	89	4	5	8A	3	6/11	3	7/12	8B	4	6/11	5	7/12		
CMPB	2	98	3	4	99	3	4	9A	3	6/11	3	7/12	9B	4	6/11	5	7/12		
MULU	2	6C	3	26	6D	4	27	6E	3	28/33	3	29/34	6F	4	28/33	5	29/34		
MULU	3	4C	4	27	4D	5	28	4E	4	29/34	4	30/35	4F	5	29/34	6	30/35		
MULUB	2	7C	3	17	7D	3	17	7E	3	19/24	3	20/25	7F	4	19/24	5	20/25		
MULUB	3	5C	4	18	5D	4	18	5E	4	20/25	4	21/26	5F	5	20/25	6	21/26		
MUL	2	②	4	30	②	5	31	②	4	32/37	4	33/38	②	5	32/37	6	33/38		
MUL	3	②	5	31	②	6	32	②	5	33/38	5	34/39	②	6	33/38	7	34/39		
MULB	2	②	4	21	②	4	21	②	4	23/28	4	24/29	②	5	23/28	6	24/29		
MULB	3	②	5	22	②	5	22	②	5	24/29	5	25/30	②	6	24/29	7	25/30		
DIVU	2	8C	3	26	8D	4	27	8E	3	28/33	3	29/34	8F	4	28/33	5	29/34		
DIVUB	2	9C	3	18	9D	3	18	9E	3	20/25	3	21/26	9F	4	20/25	5	21/26		
DIV	2	②	4	30	②	5	31	②	4	32/37	4	33/38	②	5	32/37	6	33/38		
DIVB	2	②	4	22	②	4	22	②	4	24/29	4	25/30	②	5	24/29	6	25/30		

Notes:

* Long indexed and Indirect+instructions have identical opocodes with Short indexed and Indirect modes, respectively.The second byte of instructions using any indirect or indexed addressing mode specifies the exact mode used. If the second byte is even, use Indirect or Short indexed. If it is odd, use Indirect+or Long indexed. In all cases the second byte of the instruction always specifies an even (word) location for the address referenced.

① Number of state times shown for internal/external operands.

② The opcodes for signed multiply and divide are the opcodes for the unsigned functions with an "FE" appended as a prefix.

MNEMONIC	OPERANDS	DIRECT			IMMEDIATE			INDIRECT* NORMAL			INDIRECT* AUTO-INC.			INDEXED* SHORT			INDEXED* LONG		
		OPCODE	BYTES	STATE① TIMES	OPCODE	BYTES	STATE① TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES
LOGICAL INSTRUCTIONS																						
AND	2	60	3	4	61	4	5	62	3	6/11	3	7/12	63	4	6/11	5	7/12					
AND	3	40	4	5	41	5	6	42	4	7/12	4	8/13	43	5	7/12	6	8/13					
ANDB	2	70	3	4	71	3	4	72	3	6/11	3	7/12	73	4	6/11	5	7/12					
ANDB	3	50	4	5	51	4	5	52	4	7/12	4	8/13	53	5	7/12	6	8/13					
OR	2	80	3	4	81	4	5	82	3	6/11	3	7/12	83	4	6/11	5	7/12					
ORB	2	90	3	4	91	3	4	92	3	6/11	3	7/12	93	4	6/11	5	7/12					
XOR	2	84	3	4	85	4	5	86	3	6/11	3	7/12	87	4	6/11	5	7/12					
XORB	2	94	3	4	95	3	4	96	3	6/11	3	7/12	97	4	6/11	5	7/12					
DATA TRANSFER INSTRUCTIONS																						
LD	2	A0	3	4	A1	4	5	A2	3	6/11	3	7/12	A3	4	6/11	5	7/12					
LDB	2	B0	3	4	B1	3	4	B2	3	6/11	3	7/12	B3	4	6/11	5	7/12					
ST	2	C0	3	4	—	—	——	C2	3	7/13	3	8/14	C3	4	7/13	5	8/14					
STB	2	C4	3	4	—	—	——	C6	3	7/13	3	8/14	C7	4	7/13	5	8/14					
LDBSE	2	BC	3	4	BD	3	4	BE	3	6/11	3	7/12	BF	4	6/11	5	7/12					
LDBZE	2	AC	3	4	AD	3	4	AE	3	6/11	3	7/12	AF	4	6/11	5	7/12					
STACK OPERATIONS (internal stack)																						
PUSH	1	C8	2	8/12	C9	3	9/13	CA	2	10/15	2	11/16	CB	3	10/15	4	11/16					
POP	1	CC	2	12/16	—	—	——	CE	2	15/20	2	16/21	CF	3	15/20	4	16/21					
PUSHF	0	F2	1	8																		
POPF	0	F3	1	9																		
STACK OPERATIONS (external stack)																						
PUSH	1	C8	2	13/17	C9	3	14/18	CA	2	15/20	2	16/21	CB	3	15/20	4	16/21					
POP	1	CC	2	16/20	—	—	——	CE	2	19/24	2	20/25	CF	3	19/24	4	20/25					
PUSHF	0	F2	1	12																		
POPF	0	F3	1	13																		

JUMPS AND CALLS							
MNEMONIC	**OPCODE**	**BYTES**	**STATES**	**MNEMONIC**	**OPCODE**	**BYTES**	**STATES**
LJMP	E7	3	8	LCALL	EF	3	12/16⑤
SJMP	20-27④	2	8	SCALL	28-2F④	2	12/16⑤
INDJMP③	E3	2	10	RET	F0	1	12/16⑤

Notes:

① Number of state times shown for internal/external operands.

③ This instruction is generated by the assembler when a branch indirect (BR [Rx]) instruction is reached.

④ The least significant 3 bits of the opcode are concatenated with the following 8 bits to form an 11-bit, 2's complement, offset for the relative call or jump.

⑤ State times for stack located internal/external.

CONDITIONAL JUMPS

All conditional jumps are 2 byte instructions. They require 8 state times if the jump is taken, 4 if it is not.

MNEMONIC	OPCODE	MNEMONIC	OPCODE	MNEMONIC	OPCODE	MNEMONIC	OPCODE
JC	DB	JE	DF	JGE	D6	JGT	D2
JNC	D3	JNE	D7	JLT	DE	JLE	DA
JH	D9	JV	DD	JVT	DC	JST	D8
JNH	D1	JNV	D5	JNVT	D4	JNST	D0

JUMP ON BIT CLEAR OR BIT SET

These instructions are 3-byte instructions. They require 9 state times if the jump is taken, 5 if it is not.

	BIT NUMBER							
MNEMONIC	0	1	2	3	4	5	6	7
JBC	30	31	32	33	34	35	36	37
JBS	38	39	3A	3B	3C	3D	3E	3F

LOOP CONTROL

DJNZ	OPCODE E0;	3 BYTES;	5/9 STATE TIMES (NOT TAKEN/TAKEN)

SINGLE REGISTER INSTRUCTIONS

MNEMONIC	OPCODE	BYTES	STATES	MNEMONIC	OPCODE	BYTES	STATES
DEC	05	2	4	EXT	06	2	4
DECB	15	2	4	EXTB	16	2	4
NEG	03	2	4	NOT	02	2	4
NEGB	13	2	4	NOTB	12	2	4
INC	07	2	4	CLR	01	2	4
INCB	17	2	4	CLRB	11	2	4

SHIFT INSTRUCTIONS

INSTR	WORD		INSTR	BYTE		INSTR	DBL WD		
MNEMONIC	OP	B	MNEMONIC	OP	B	MNEMONIC	OP	B	STATE TIMES
SHL	09	3	SHLB	19	3	SHLL	0D	3	8 + 1 PER SHIFT
SHR	08	3	SHRB	18	3	SHRL	0C	3	8 + 1 PER SHIFT
SHRA	0A	3	SHRAB	1A	3	SHRAL	0E	3	8 + 1 PER SHIFT

SPECIAL CONTROL INSTRUCTIONS

MNEMONIC	OPCODE	BYTES	STATES	MNEMONIC	OPCODE	BYTES	STATES
SETC	F9	1	4	DI	FA	1	4
CLRC	F8	1	4	EI	FB	1	4
CLRVT	FC	1	4	NOP	FD	1	4
RST	FF	1	16⑥	SKIP	00	2	4

NORMALIZE

NORML	0F	3	11 + 1 PER SHIFT

Notes:

⑥ This instruction takes 4 states to pull RST low, then holds it low for 2 states to initiate a reset. The reset takes 10 states, at which time the program restarts at location 2080H.

Instruction Summary

MNEMONIC	OPER-ANDS	OPERATION (Note 1)	FLAGS						NOTES
			Z	N	C	V	VT	ST	
ADD/ADDB	2	D← D+A	√	√	√	√	√	—	
ADD/ADDB	3	D← B+A	√	√	√	√	√	—	
ADDC/ADDCB	2	D← D+A+C	√	√	√	√	√	—	
SUB/SUBB	2	D← D − A	√	√	√	√	√	—	
SUB/SUBB	3	D← B − A	√	√	√	√	√	—	
SUBC/SUBCB	2	D← D − A + C − 1	√	√	√	√	√	—	
CMP/CMPB	2	D − A	√	√	√	√	√	—	
MUL/MULU	2	D, D + 2← D * A	—	—	—	—	—	√	2
MUL/MULU	3	D, D + 2← B * A	—	—	—	—	—	√	2
MULB/MULUB	2	D, D + 1← D * A	—	—	—	—	—	√	3
MULB/MULUB	3	D, D + 1← B * A	—	—	—	—	—	√	3
DIV/DIVU	2	D← (D, D + 2)/A D + 2← remainder	—	—	—	√	√	—	2
DIVB/DIVUB	3	D← (D, D + 1)/A D + 1← remainder	—	—	—	√	√	—	3
AND/ANDB	2	D← D and A	√	√	0	0	—	—	
AND/ANDB	3	D← B and A	√	√	0	0	—	—	
OR/ORB	2	D← D or A	√	√	0	0	—	—	
XOR/XORB	2	D← D (excl. or) A	√	√	0	0	—	—	
LD/LDB	2	D← A	—	—	—	—	—	—	
ST/STB	2	A← D	—	—	—	—	—	—	
LDBSE	2	D← A; D + 1← SIGN(A)	—	—	—	—	—	—	3,4
LDBZE	2	D← A; D + 1← 0	—	—	—	—	—	—	3,4
PUSH	1	SP← SP − 2; (SP)← A	—	—	—	—	—	—	
POP	1	A← (SP); SP← SP + 2	—	—	—	—	—	—	
PUSHF	0	SP← SP − 2; (SP)← PSW; PSW← 0000H	0	0	0	0	0	0	
POPF	0	PSW← (SP); SP← SP + 2	√	√	√	√	√	√	
SJMP	1	PC← PC + 11-bit offset	—	—	—	—	—	—	5
LJMP	1	PC← PC + 16-bit offset	—	—	—	—	—	—	5
INDJMP	1	PC← (A)	—	—	—	—	—	—	
SCALL	1	SP← SP − 2; (SP)← PC; PC← PC + 11-bit offset	—	—	—	—	—	—	5
LCALL	1	SP← SP − 2; (SP)← PC; PC← PC + 16-bit offset	—	—	—	—	—	—	5
RET	0	PC← (SP); SP← SP + 2	—	—	—	—	—	—	
J(conditional)	1	PC← PC + 8-bit offset	—	—	—	—	—	—	5
JC		Jump if C = 1	—	—	—	—	—	—	5
JNC		Jump if C = 0	—	—	—	—	—	—	5

Note

1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B, and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.
2. D, D + 2 are consecutive WORDS in memory; D is DOUBLE-WORD aligned.
3. D, D + 1 are consecutive BYTES in memory; D is WORD aligned.
4. Changes a byte to a word.
5. Offset is a 2's complement number.

Instruction Summary

MNEMONIC	OPERANDS	OPERATION (Note 1)	FLAGS						NOTES
			Z	N	C	V	VT	ST	
JE		Jump if Z = 1	—	—	—	—	—	—	5
JNE		Jump if Z = 0	—	—	—	—	—	—	5
JGE		Jump if N = 1	—	—	—	—	—	—	5
JLT		Jump if N = 0	—	—	—	—	—	—	5
JGT		Jump if N = C and Z = 0	—	—	—	—	—	—	5
JLE		Jump if N = 1 or Z = 1	—	—	—	—	—	—	5
JH		Jump if C = 1 and Z = 0	—	—	—	—	—	—	5
JNH		Jump if C = 0 or Z = 1	—	—	—	—	—	—	5
JV		Jump if V = 1	—	—	—	—	—	—	5
JNV		Jump if V = 0	—	—	—	—	—	—	5
JVT		Jump if VT = 1; Clear VT	—	—	—	—	0	—	5
JNVT		Jump if VT = 0; Clear VT	—	—	—	—	0	—	5
JST		Jump if ST = 1	—	—	—	—	—	—	5
JNST		Jump if ST = 0	—	—	—	—	—	—	5
JBS		Jump if Specified Bit = 1	—	—	—	—	—	—	5, 6
JBC		Jump if Specified Bit = 0	—	—	—	—	—	—	5, 6
DJNZ	1	D← D − 1; if D ≠ 0 then PC← PC + 8-bit offset	—	—	—	—	—	—	5
DEC/DECB	1	D← D − 1	√	√	√	√	√	—	
NEG/NEGB	1	D← 0 − D	√	√	√	√	√	—	
INC/INCB	1	D← D + 1	√	√	√	√	√	—	
EXT	1	D← D; D + 2← Sign (D)	√	√	0	0	—	—	2
EXTB	1	D← D; D + 1← Sign (D)	√	√	0	0	—	—	3
NOT/NOTB	1	D← Logical Not (D)	√	√	0	0	—	—	
CLR/CLRB	1	D← 0	1	0	0	0	—	—	
SHL/SHLB/SHLL	1	C ← [msb — — — — — lsb] ← 0	√	√	√	√	√	—	7
SHR/SHRB/SHRL	1	0 → [msb — — — — — lsb] →C	√	√	√	0	—	√	7
SHRA/SHRAB/SHRAL	1	msb → [msb — — — — — lsb] →C	√	√	√	0	—	√	7
SETC	0	C← 1	—	—	1	—	—	—	
CLRC	0	C← 0	—	—	0	—	—	—	
CLRVT	0	VT← 0	—	—	—	—	0	—	
RST	0	PC← 2080H	0	0	0	0	0	0	8
DI	0	Disable All Interrupts	—	—	—	—	—	—	
EI	0	Enable All Interrupts	—	—	—	—	—	—	
NOP	0	PC← PC + 1	—	—	—	—	—	—	
SKIP	0	PC← PC + 2	—	—	—	—	—	—	
NORML	2	Normalize (See Text)	√	1	—	—	—	—	7

Note

1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B, and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.
5. Offset is a 2's complement number.
6. Specified bit is one of the 2048 bits in the register file.
7. The "L" (Long) suffix indicates double-word operation.
8. Initiates a Reset by pulling RESET low. Software should re-initialize all the necessary registers with code starting at 2080H.

Extensive I/O subsystems and a tailored instruction set allow a 16-bit microcontroller to set its sights on a widening range of industrial and computer (and telecomm and consumer) applications.

Controller chip takes on many industrial, computer uses

With industrial and computer control applications increasing all the time—and telecommunications and consumer applications emerging—designers increasingly need microcontrollers whose performance extends beyond that of the conventional 8-bit architectures. Normally, control system designers must depend on expensive and complex multiple-chip microprocessors to achieve high performance. But now, a 16-bit single-chip controller offers a much better solution. Not only does the 8096 offer perhaps the most extensive input/output "services" of any microcontroller, it also provides an instruction set and addressing modes tuned for both fast control operations and high-speed arithmetic.

In industrial applications, the 8096 can be used for process control, robotics, numerical and motor control, and instrumentation. Figure 1 shows the chip in a typical closed-loop servo system of the type used in industrial applications. In computer applications performance is the key feature, and here the 8096 provides greater throughput in systems in which simple data structures—a single I/O bit—and relatively small memories are required. Typical applications are computer peripherals such as printers, plotters, Winchesters, and other hard-disk systems.

In the consumer end, moreover, the 8096 is ideally suited for automotive engine and other controls (see "Stopping a Car") and sophisticated video games. Both applications need the speed, calculating power, and addressability of a 16-bit microcomputer. For telecommunications, the controller is intended for high-speed modems, PABXs, and central office switching systems.

In addition to the full 16-bit CPU, the 8096's basic architecture includes an 8-kbyte ROM and a 232-byte RAM, which serves as a register file. To meet the wide needs of controller environments, the chip contains an eight-channel, 10-bit analog-to-digital converter, a full-duplex UART (universal asynchronous receiver-transmitter), two 16-bit timers, and a programmable pulse-width-modulated output.

Since a microcontroller must be able to interface with various types of transducers and sensors, the 8096 features built-in, extensive I/O facilities. These include an eight-level priority interrupt structure, full-duplex serial I/O, parallel I/O, a watchdog timer, analog inputs for the a-d converter, a pulse-width modulated output and a high-resolution pulse output. Each of these facilities is integrated not only physically but logically into the chip's structure by being tightly coupled to the CPU.

The inherently high performance of a CPU suffers if the controller spends too much time administering complex real-time I/O operations. The 8096's on-board I/O facilities solve this problem by permitting the CPU to devote more time to executing mathematics and control algorithms and less on I/O.

Table 1. Memory allocations of the 8096

Locations	Uses
0000–0017	On-chip I/O
0018–0019	Data register/stack pointer
001A–00FF	Data registers (230 bytes)
0100–1FFD	Off-chip expansion RAM/ROM/I/O
1FFE–1FFF	On-chip I/O
2000–200F	Internal ROM interrupt vectors
2010–207F	Reserved
2080–3FFF	Internal ROM user program space
4000–FFFF	Off-chip expansion RAM/ROM/I/O

Steve Wiseman, Product Marketing Manager
Steve Burton, Senior Engineer
John Katausky, Technical Marketing Manager
Intel Corp.
5000 W. Williams Field Rd., Chandler, Ariz. 85224

The instruction set handles signed and unsigned 16-bit multiplications and divisions. Both 8-bit bytes and 16-bit double words are supported, and even 32-bit double words are supported for a subset of the main instruction set. A full 64 kbytes of memory address space is usable.

A flexible register structure

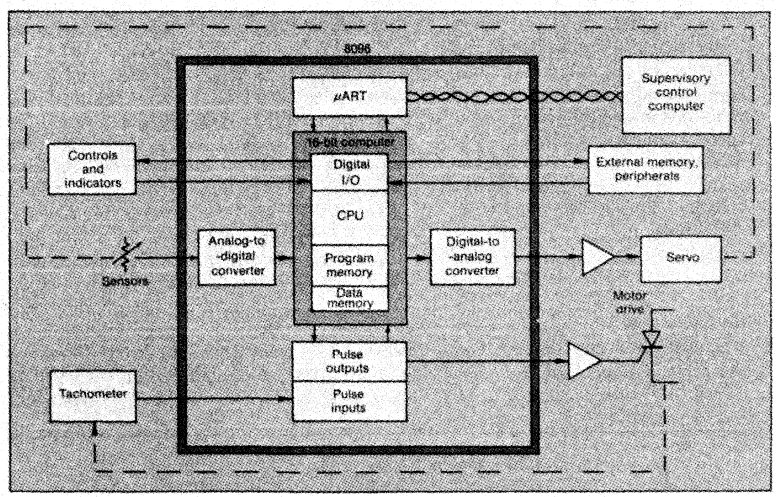
The 8096 instruction set directly supports 256 bytes of registers, which can be referenced as 128-word registers or as 64 double-word registers. These registers also appear—for memory reference instructions—as the first 256 bytes of the 64-kbyte RAM address space. This permits, for example, the use of a portion of the register space as the subroutine stack on smaller systems that do not have external expansion memory (Table 1).

The first 24 bytes of this register space are reserved for on-chip I/O addresses. I/O locations are memory-mapped and can be referenced directly as registers. The word register located at address 18H serves as the stack pointer. Such a large register space allows a programmer to keep his most frequently referenced scalar variables in registers.

Table 2. Address modes of the 8096

Name	Time (μs)[1]	Time (μs)[2]	Written form		Action taken
Direct	0	N.A.	ADD	B,A	(B) <= (B) + (A)
Immediate	0.2 (word)	N.A.	ADDB	B,#A	(B) <= (B) + A
Immediate	0 (byte)	N.A.	ADD	B,#A	(B) <= (B) + A
Indirect	0.4	1.4	ADD	B,[A]	(B) <= (B) + ((A))
Autoincrement	0.6	1.6	ADD	B,[A]+	(B) <= (B) + ((A)); (A) <= (A) + <length of A>
Short indexed	0.4	1.4	ADD	B,C[A]	(B) <= (B) + ((A) + C); where −128<C<127
Short indexed	0.6	1.8	ST	C[A],B	((A) + C) <= (B) ST, Pop only
Long indexed	0.6	1.6	ADD	B[A],C	(B) <= (B) + ((A) + C)
Long indexed	0.8	1.8	ST	[A]C,B	((A) + C) = (B) ST, Pop only

1. Addressed operand located in register space.
2. Addressed operand located in ROM space or external memory-expansion space.
Note: ST and Pop are the only instructions with an address-moded destination.



1. Sitting at the center of a closed-loop servo system, the 8096 microcontroller's I/O facilities interface with both digital and analog input signals. The 16-bit chip can handle virtually any type of computer or industrial control-system application.

Since the number of instructions required is reduced, fewer external memory references are needed. As a result, program execution is accelerated.

The 8096 uses separate internal instruction and data buses. With this architecture, ROM and external memory references are slightly slower than register references. Instructions cannot be executed out of the internal RAM register space, but external-expansion RAM instructions can be executed.

Both memory space and register space are fully byte-addressable. A 16-bit word begins on an even byte address, and the odd byte is the most significant byte of the word. A 32-bit double word begins on an even-word address—both bit 0 and bit 1 of the address are zeros. Double words are produced by Multiply Words, Shift, and Normalize instructions and are used by Divide Words, Shift, and Normalize instructions. Double words are added and subtracted using Add with Carry and Subtract with Borrow instructions.

In most computers, the most commonly used instruction is Move. Because of the many registers in the 8096, a programmer can get away with fewer Move instructions—it is not necessary to switch

Stopping a car

The 8096 can be very useful in an automotive antiskid braking system that allows a driver to decelerate his vehicle safely when one or more wheels start slipping. Slip conditions can be detected either by excessive wheel-speed differential or by excessive apparent deceleration, or both. For example, if one wheel hits an ice patch during heavy braking, the rotation of that wheel will slow down significantly, indicating a skid. On the other hand, normal tires begin to slip at or below 1 g of acceleration. If the apparent deceleration as measured by wheel speed is less than 1 g, the wheel is assumed to be skidding. Both skid-detection techniques depend on measuring a wheel's rotational speed.

Wheels are monitored by reluctance (magnetic) pickups, which generate pulse trains whose frequencies are proportional to wheel speed. Usually, a simple numerical relationship relates frequency to speed—13.3 Hz per miles per hour is a typical value. Four pickup outputs are easily handled by the 8096's high-speed input unit. At each transition of any of the pickups, the current value of timer 0 is saved in the input FIFO. The programmable edge detector in the high-speed input unit provides a convenient device for handling the wide dynamic range of the period measurement. At slow speeds, the edge detector can be programmed to respond to both edges of the input signal; at medium speeds, to recognize only positive-going edges; and at high speeds, to respond to just one of eight positive-going edges. This technique not only extends the dynamic range of the measurement, but also reduces the interrupt overhead at high speeds.

Three successive time samples —T_x, T_y, and T_z—allow the speed and acceleration of a wheel to be determined from the following equations:

$$V_{xy} = C/(T_y - T_x) \quad (1)$$

$$V_{yz} = C/(T_z - T_y) \quad (2)$$

$$A_{zx} = (V_{yz} - V_{xy})/[0.5\,(T_z - T_x)] \quad (3)$$

where C is the reciprocal of the proportionality constant (e.g., C = 1/13.3), V_{xy}, V_{yz}, and V_{xy} are velocities, and A_{zx} is acceleration.

The period of the incoming frequency is in units of 1.6 μs since timer 0 is incremented at this rate. If I_x represents the value in the timer at time x and I_x the value at time y, then Eq. 1 and 2 are written as

$$V_{xy} = 46992.48/(I_y - I_x) \quad (4)$$

$$V_{yz} = 46992.48/(I_z - I_y) \quad (5)$$

In practice, the constants in Eq. 4 and 5 should be multiplied by a scaling factor to allow calculations to be performed in integer arithmetic. A factor of 100, for example, gives speed measured in units of 1/100 of a mile per hour. Then the value 46992.48 becomes 4699248, which can easily be represented within 32 bits, and a 32-by-16-division instruction is used to perform the division.

Each wheel requires two such divisions—one for speed, one for acceleration—during each loop of the calculation. A typical loop takes about 10 ms. A typical 8-bit microprocessor takes 500 to 750 μs per division, which means eight such divisions would require 4 to 6 ms. But the 8096 does all eight divisions in about 50 μs. This speed improvement translates into a higher-performance module in response time or adaptability.

The watchdog timer of the 8096 helps enhance the reliability of the braking module. At a 15-MHz clock rate, the timer is incremented every 200 ns. During operation, the system software executes diagnostics periodically to ensure that the overall system—including hardware and software—is operating properly. If the operation is correct, the software will issue commands to reset the watchdog. But if a system failure prevents a diagnostic from running within a prescribed period, the watchdog timer will reset the entire system. The software cannot reset properly on an erroneous operation, such as a counter overflow, except by writing to the watchdog timer twice within its counting cycle.

operands in and out of memory locations. In addition, the chip's powerful three-operand instructions—Add, Subtract, Multiply and Logical And—often eliminate them. Since programmer productivity (measured in lines of code written per day) is reasonably constant, writing fewer Move instructions can lead to reduced development expense. Register Load and Store instructions, with a full set of addressing modes, handle moves that cannot be eliminated.

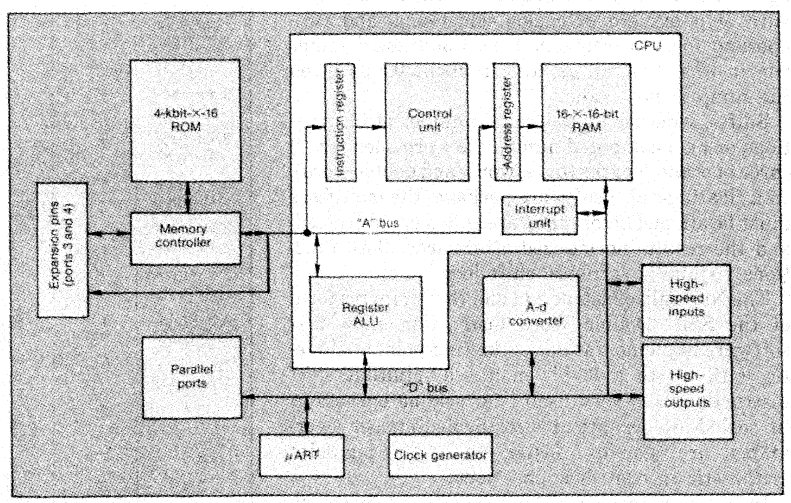
Keeping addressing simple

Because a study of the ways in which addressing is used on the 8086 microprocessor indicates that programmers use complex addressing modes less than 0.7% of the time, the 8096's instruction set bypasses those in favor of the more commonly used addressing modes. But should complex addressing be needed, programmers can build such modes through macros. Addressing modes in the 8096 include direct, register-indirect, immediate, autoincrement, and both short (8-bit) and long (16-bit) indexed-address. Table 2 lists the address modes and the operations that occur when each is activated.

Indexed-address modes, by adding an 8- or 16-bit displacement to the contents of any 16-bit register to form the effective address of an operand, allow fast access to arrays stored anywhere in memory. Indexed modes are also useful for referencing elements of based structures, as in the PL/M language. However, preliminary calculations are needed to reference a based array.

The stack pointer is fully addressable, as are all other 16-bit registers. As a result, it can be the base register for moded references. Stack-relative addressing, which is easy to program, is often used for recursive-subroutine parameter passing and dynamically allocated variables. While this technique does not make the best use of a large register space, it adds flexibility to the system. The stack need not be confined to internal RAM, but can fill any available RAM space in the system. The stack can flow across the boundary into register space at will, allowing recursion to very great subroutine depths.

Of the instruction set's 71 instructions, 25 take on both word and byte form, which increases the total to 96 instructions. The set includes 16 varieties of conditional jump, allowing for both signed and unsigned comparisons. All of the 2048 bits in register space can be tested individually by a Jump on Bit/Not Bit instruction. A Decrement and Jump on Not Zero instruction provides for loop control.



2. Centered around the CPU and memory, the 8096's extensive I/O subsystems include an analog-to-digital converter, a universal asynchronous receiver-transmitter (UART), high-speed input and output circuitry, and a pulse-width modulation output circuit. Such intelligent I/O allows the CPU to concentrate not on real-time housekeeping but on high-speed arithmetic and control operations.

Most of the instructions execute in about 0.8 μs; the longest, to normalize a zero, takes 8 μs. All data-reference instructions except Pop, Push, and Normalize are available in byte form, and all such instructions except Jump on Bit and Normalize are available in word form. Table 3 lists some typical 8096 instructions and their run times.

A survey of code frequency usage shows that although most multiplications and divisions are unsigned, a signed form is still necessary. When unsigned multiplication and division instructions are preceded by a 0.8-μs SIGND prefix, they are converted into full two's-complement signed multiplication and division. Either type of operation executes in less than 6 μs. Word multiplications result in a double-word product, and byte multiplications produce a word product. With an instruction called Word Divide, a double-word dividend is divided by a word divisor to produce a word quotient and remainders.

Because jumps and calls are PC-relative, code is easy to relocate. Both Jump and Call instructions are available in a short 2-byte form with an 11-bit displacement. Jump on Bit is a 3-byte instruction with an 8-bit displacement. An indirect jump for the "do-case" is also provided.

In addition to the usual sign-extending (EXT) instructions for byte-to-word and word-to-double-word conversions, the set includes instructions LDBSA and LDBZE, which move a byte into a word with sign or zero extension. Most one- and two-operand forms execute in 1 μs. Conditional jumps run in less than 1.8 μs, and in about 0.8 μs when the jump is not taken.

Shifts, whether by a specific number of bit positions or by a computed number, are provided for all three operand lengths (byte, word, and double word). In a floating-point software package, the mantissas must be aligned before they are added or subtracted, and the results normalized afterwards. Both functions require a software shift loop.

The Normalize instruction and the computer form of the Shift Double Word instruction allow fast software implementations of floating-point arithmetic with up to a 32-bit mantissa. Multibit shift instructions are very useful for scaling operations in scaled-integer arithmetic. Scaled-integer operations are usually faster than floating-point arithmetic in control applications.

In addition to an overflow flag, which is set by each arithmetic instruction, there is an overflow-trap flag. It can be checked at the end of a sequence of instructions to determine whether an overflow has occurred anywhere in the sequence.

The instruction set is complemented by a variety of I/O subsystems for handling virtually any com-

Table 3. Typical instruction times

Microseconds	Operands	Mnemonics	Descriptions
0.8	0	CLRC,SETC,DI,EI, CLRVT,SIGND	Flag manipulations
0.8	1	INC,DEC,CLR, NOT,NEG,SEX	One-operand instructions
0.8	2*	XOR,ADDC,SUB, AND,ADD,SUBC	Two-operand arithmetics
0.8	2*	OR,CMP	Two-op arithmetics
0.8	2*	LD,LDBSE,LDBZE, ST	Load and store registers
0.8 (not taken)	1	JC,JNC,ETC.	Conditional jumps
1.0	3*	AND,SUB,ADD	Three-op arithmetics
1.0 (not taken)	2	JBS,JBC	Jump on bit/ jump on not bit
1.6 (taken)	1	JC,JNC,ETC.	Conditional jumps
1.6	1	SJMP,IJMP, LJMP	Unconditional jumps
1.6 (stack register)	0	PUSHF	Push PSW
1.6 (stack register)	1*	PUSH	Stack push
1.8 (taken)	2	JBS,JBC,DJNZ	Jump on bit/ decrement and jump
1.6 + 0.2/shift	2	SHL,SHR,SHRA	Shift instructions
1.8	0	POPF	Pop PSW
2.2 + 0.2/shift	2	NORML	Normalize
2.4	1*	POP	Stack pop
2.4 (stack register)	1	LCALL,SCALL, RET	Subroutines
2.4 (stack external)	0	PUSHF	Push PSW
2.4 (stack external)	1*	PUSH	Stack push
2.6 (stack external)	0	POPF	Pop PSW
2.8 (stack external)	1*	POP	Stack pop
3.0 (stack external)	1	SCALL,CALL	Subroutines
3.2 (stack external)	0	RET	Subroutine
3.4	2*	MULB	Byte multiplication
3.6	2*	DIVB	Byte division
3.6	3*	MULB	Byte multiplication
5.2	2*	MUL	Word multiplication
5.2	2*	DIV	Word division
5.4	3*	MUL	Word multiplication

*One of these operands may have full address modes.

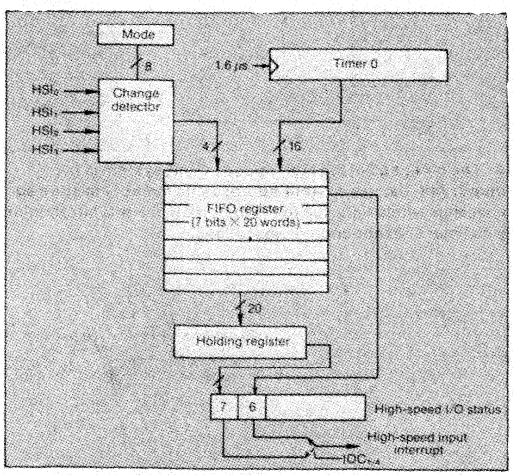
puter peripheral or industrial application (Fig. 2). They include an a-d converter, a UART, timer-counters, and a programmable pulse-width–modulated output.

I/O resources include a-d

The controller contains a complete eight-channel, 10-bit a-d converter. Using successive approximation to achieve high speed—33.6 μs at a 15-MHz clock rate—it handles analog input voltages in the range of 0 to 5 V. An external reference is required and must be connected between the reference voltage and analog ground terminals. The converter generates a vectored interrupt when it completes a conversion cycle, allowing the CPU to have rapid access to the a-d input handler when operating in a multitask environment.

Conversion is initiated by writing to an 8-bit a-d command register. The results of a conversion are read from two 8-bit output data registers. One 8-bit register contains the eight most significant bits, and the other holds the two least significant bits, a 3-bit channel indicator, two unused bits, and a status bit. The status bit, which indicates whether the a-d conversion is still in progress, is typically used in a noninterrupt-driven environment.

Just four bits of the a-d command register are used. Three of the bits specify the channel to be converted, and the fourth specifies the method of initiating an a-d conversion cycle. For example, if the fourth bit is a 1, the cycle begins immediately after writing to the command register. If it is a 0, the high-speed output logic subsystem initiates the conversion. The reason for the option is that many data acquisition algorithms require that conversions occur at specific intervals. This requirement is often difficult to manage through software because of interrupt latency and other conditions. Thus, the high-speed output subsystem provides the proper timing for periodic a-d conversions.



3. One half of the 8096's high-speed I/O subsystem is an input unit, which contains a user-programmable change detector that defines input transitions for the high-speed inputs. Each of the four inputs (HSI_0 – HSI_3) can be programmed to respond to a different input transition.

The 8096's UART is virtually a carbon copy of the one on the 8051 microcontroller. One of its 8-bit registers receives data, another transmits data, and another indicates the UART status plus bits to configure it for a specific operating mode. By setting the appropriate bits in the third, or control-status, register, a user can select one of four modes:

- Mode 0 (shift register) is a simple, synchronous mode in which the 8096 provides a clock to synchronize incoming or outgoing data. Mode 0 can also be used to expand the I/O.
- Mode 1 is an 8-bit UART mode in which the eighth bit is used for parity when it is enabled.
- Mode 2 is a 9-bit UART mode in which the ninth bit is used for parity when it is enabled.
- Mode 3 is a 9-bit data/address mode in which the UART transmits and receives nine bits of data. This is useful for implementing a simple multiprocessor intercommunications link in which the ninth bit distinguishes address from data.

The remaining six bits of the control-status register are used for six operations: enabling the receiver section of the UART, enabling parity for both transmission and reception (even parity); storing the ninth bit when in the 9-bit transmitting mode; storing the ninth bit when in the 9-bit receiving mode, indicating that the receiver is ready and indicating that the transmitter is ready. Also on board are a dedicated 15-bit baud-rate generator and a baud-rate clock that can be driven by either the 8096's crystal oscillator or an input at pin T2CLK. This gives maximum flexibility in setting baud rates.

The pulse-width modulated output can produce a pulse train of variable duty cycle, which can be integrated and clamped to provide an accurate digital-to-analog output function. The PWM circuit operates as follows: The 8096 crystal frequency is divided by three and clocks an 8-bit free-running counter. The counter output connects to one side of an 8-bit comparator; the other side of the comparator is tied to a user-addressable register. When the free-running counter value is the same as the one stored in the addressable register, an R-S flip-flop is set. The flip-flop is also reset when the counter rolls over from a count of 255 to 0. This produces a simple yet accurate variable duty-cycle oscillator, which can be programmed for a variable duty cycle from 0 to 255 in increments of X/256.

The watchdog timer offers a simple way to recover from a software or hardware error. Essentially a 16-bit free-running counter that is clocked by the CPU clock generator circuitry, the timer is reset by writing a $01E_H$ followed by a $0E1_H$ to byte location 000AH. If a resetting does not occur at least once every 13.107 ms, the timer will overflow, causing the 8096 to be reset—resetting reinitializes the 8096. This feature makes it virtually impossible for the 8096 to become lost in a program for too long. For development purposes, the reset terminal can be connected to V_{CC} to disable the watchdog timer.

More I/O—and faster

Correlating events in real time is one of the most important considerations in computer-based control system design. Another common requirement is generating pulses and pulse trains to drive actuators. Most single-chip microcontrollers support such operations by having one or more timer/event counters under software control. The 8096, on the other hand, offers a complete integrated subsystem to perform these functions. Called the high-speed I/O unit, it is intended to be an integrated subsystem, but it can be viewed as separate units for input and output.

Figure 3 shows the block diagram of the high-speed input unit. Its major components are a 16-bit timer, a programmable change detector and a first-in, first-out (FIFO) memory. Also included are several registers used by the software to control the high-speed input unit.

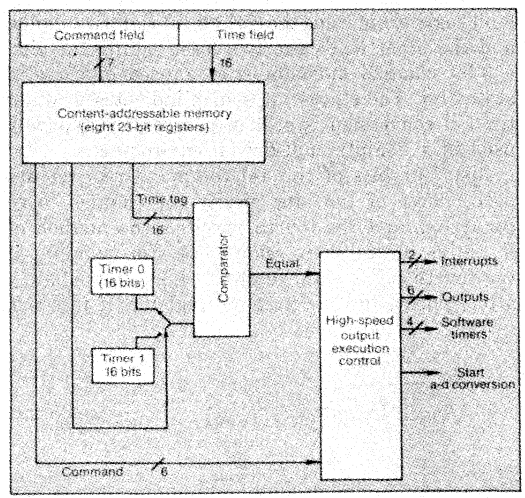
The read-only timer is cleared by the system reset and incremented once every eight CPU cycles (every 1.6 μs with a 15-MHz crystal). When the timer overflows—rolls over from $FFFF_H$ to 0000_H—a status bit is set and an interrupt is generated. The change detector monitors four pins on the 8096 and looks for predefined changes. Change definitions are controlled by the high-speed input unit's mode register, which is set by the software. This register contains a 2-bit field for each of the four high-speed inputs. Using the fields, a programmer can select the type of change for each input. Fields are encoded in one of four ways:

- 00 defines positive transitions divided by 8.
- 01 defines positive transitions.
- 10 defines negative transitions.
- 11 defines positive and negative transitions.

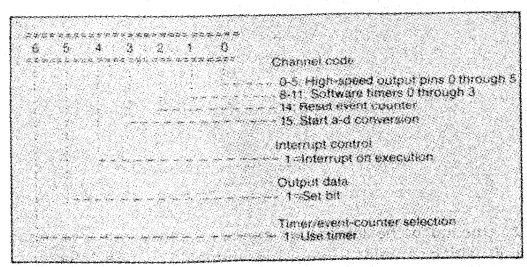
Each high-speed input can be disabled through a second control register. When this is done, inputs of the high-speed input unit become available as digital input pins or, if required, two of the pins can be connected to the high-speed output unit.

As the block diagram in Fig. 4 shows, the high-speed output unit uses the same timer as the input unit and also has a 16-bit event counter. The read-only event counter is similar to the timer in that it can be read at any time, generates an overflow-interrupt or status indication, and cannot be written into. It differs from the timer, in that its reset and clock sources, instead of being fixed by hardware, can be selected under software control.

Two of the 8096's pins are dedicated to the event counter. A positive-going pulse on ECRST (Event Counter Reset) clears the counter, and either edge of a pulse applied to ECCLK (Event Counter Clock) increments the counter. A programmer has the option of using HSI_0 instead of ECRST or HSI_1 instead of ECCLK. These options are available by setting the appropriate bits in the I/O control register. The event counter can also be cleared under software control either directly, by setting a bit in the I/O control register, or indirectly, using the high-



4. The other half of the high-speed I/O subsystem is the output unit. Using a content-addressable memory to store so-called time-field data, the unit's logic matches this information with timer or event-counter operations.



5. In the content-addressable memory of the high-speed output unit, 23-bit words are broken down into a 16-bit time field and a 7-bit command field. Command-field encoding defines the output unit's operating mode.

speed output unit itself.

The FIFO register of the high-speed input unit is replaced by a content-addressable memory in the output unit. The memory contains a file of eight 23-bit registers. The 23 bits are divided into a 16-bit time field and a 7-bit command field. Control logic continually scans each location in the memory to determine whether its time field matches either the timer or the event counter as selected by one of the seven bits in the command field. When a match is found, the remaining six bits in the command field are executed.

What the bits do

The encoding of the command field bits and their functions are shown in Fig. 5. Four-bit channel code selects the output unit's operation. For example, the event counter can be reset or an a-d conversion can be initiated. If one of the high-speed output pins is to be changed, bit 5 of the command field will determine its state. For all of the high-speed output unit's operations, bit 4 determines whether an interrupt is generated upon execution of a command.

The high-speed output unit uses two interrupt vectors, one for the software timers and one for all other functions. When a software timer interrupt occurs, the interrupt service routine can interrogate an I/O status register to determine which of the four timers caused the interrupt. The ability of the command field to trigger an a-d conversion allows measurements to be made at precise moments, an absolute necessity in digital signal processing. Also, the ability to reset a count when it reaches a preset limit allows the simple implementation of a modulo-N counter. This is useful, for example, in a crankshaft position-sensor application that generates 214 pulses per revolution.

The eight locations in the content-addressable memory's file are scanned at the rate of one CPU cycle per location. At a 15-MHz clock rate, all eight locations will be scanned within 1.6 μs. A high-speed output unit's command is executed as soon as it finds a time match. As each command is executed, it is removed from the content-addressable memory to make room for a new command, which is sent in from the input holding register.

Because of the extensive functions built into the 8096, a standard 40-lead DIP is far too small; the 8096 is housed in a 68-pin JEDEC package. Alternatively, it is supplied in a 48-pin DIP.□

8020H

HMOS SINGLE-COMPONENT 8-BIT MICROCOMPUTER

- 8-bit CPU, ROM, RAM, I/O in Single 20-pin Dip
- Single +5V Supply (+4.5V to 6.5V)
- 8.38 μsec Cycle with 3.58 MHz XTAL. All instructions 1 or 2 cycles.
- Instructions — 8048 Subset
- Zero-Cross Detection Capability
- 1K x 8 ROM
  64 x 8 RAM
  13 I/O Lines
- Internal Timer/Counter
- 30mA Operation @ 25°C
- Clock Generated with Single Inductor, Resistor or Crystal

The Intel 8020H is a 20-pin DIP version of the Intel 8021H. This totally self-sufficient 8-bit parallel computer is fabricated on a single silicon chip using Intel's advanced N-channel silicon gate HMOS process.

The 8020H is designed for high-volume cost-sensitive applications such as home entertainment products, appliances and simple control tasks. In addition to its significant board real estate savings, the 8020H has a subset of the industry standard 8048 instruction set optimized for byte efficiency and control.

The 8020H also features a 1K x 8 program memory, 64 x 8 data memory, 13 I/O lines and an 8-bit timer/counter with on-board oscillator and clock circuits. Standard low cost TTL can be used to expand the number of I/O lines.

To minimize development problems and maximize flexibility, an 8020H system can be easily designed using the EM-2 emulator board and an 8020H pin scrambler interface.

For a complete summary of 8020H operating characteristics, refer to the 8021H data sheet.

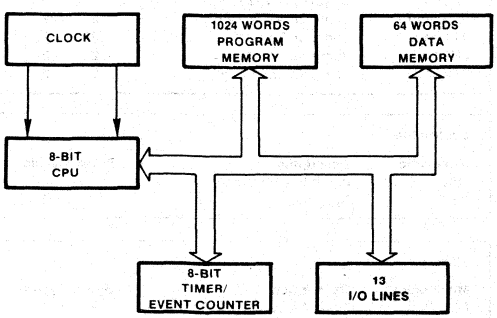


Figure 1.
Block Diagram

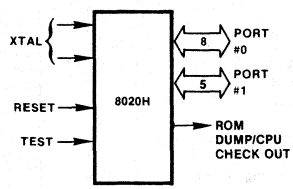


Figure 2.
Logic Symbol

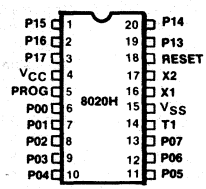


Figure 3. Pin
Configuration

Table 1. Pin Description

Symbol	Pin No.	Function
V_{SS}	15	Circuit GND potential
V_{CC}	4	+5V power supply
PROG	5	Output pin used to dump internal ROM and to activate CPU checkout.
P00-P07 Port 0	6-13	8-bit quasi-bidirectional port
P3-P17 Port 1	19-20 1-3	5-bit quasi-bidirectional port
T1	14	Input pin testable using the JT1 and JNT1 instructions. Can be designated the timer/event counter input using the STRT CNT instruction. Also allows zero-crossover sensing of slowly moving AC inputs
RESET	18	Input used to initialize the processor by clearing status flip-flops and setting program counters to zero. (Active High).
XTAL1	16	One side of crystal or inductor input for internal oscillator. Also input for external source. (Not TTL compatible.)
XTAL2	17	Other side of timing control element.

Table 2. Instruction Set

Accumulator

Mnemonic	Description	Bytes	Cycles
ADD A, R	Add register to A	1	1
ADD A, @R	Add data memory to A	1	1
ADD A, # data	Add immediate to A	2	2
ADDC A, R	Add with carry	1	1
ADDC A, @R	Add with carry	1	1
ADDC A, # data	Add with carry	2	2
ANL A, R	And register to A	1	1
ANL A, @R	And data memory to A	1	1
ANL A, # data	And immediate to A	2	2
ORL A, R	Or register to A	1	1
ORL A, @R	Or data memory to A	1	1
ORL A, # data	Or immediate to A	2	2
XRL A, R	Exclusive or register to A	1	1
XRL A, @R	Exclusive or data memory to A	1	1
XRL A, # data	Exclusive or immediate to A	2	2
INC A	Increment A	1	1
DEC A	Decrement A	1	1
CLR A	Clear A	1	1
CPL A	Complement A	1	1
DA A	Decimal Adjust A	1	1
SWAP A	Swap nibles of A	1	1
RL A	Rotate A left	1	1
RLC A	Rotate A left through carry	1	1
RR A	Rotate A right	1	1
RRC A	Rotate A right through carry	1	1

Input/Output

Mnemonic	Description	Bytes	Cycles
IN A, P	Input port to A	1	2
OUTL P, A	Output A to port	1	2

Registers

Mnemonic	Description	Bytes	Cycles
INC R	Increment register	1	1
INC @R	Increment data memory	1	1

Branch

Mnemonic	Description	Bytes	Cycles
JMP addr	Jump unconditional	2	2
JMPP @A	Jump indirect	1	2
DJNZ R, addr	Decrement register and Jump on R not zero	2	2
JC addr	Jump on Carry = 1	2	2
JNC addr	Jump on Carry = 0	2	2
JZ addr	Jump on A Zero	2	2
JNZ addr	Jump on A not Zero	2	2
JT1 addr	Jump on T1 = 1	2	2
JNT1 addr	Jump on T1 = 0	2	2
JTF addr	Jump on timer flag	2	2

Subroutine

Mnemonic	Description	Bytes	Cycles
CALL	Jump to subroutine	2	2
RET	Return	1	2

Flags

Mnemonics	Description	Bytes	Cycles
CLR C	Clear Carry	1	1
CPL C	Complement Carry	1	1

Table 2. Instruction Set Summary (cont.)

Data Moves

Mnemonics	Description	Bytes	Cycles
MOV A, R	Move register to A	1	1
MOV A, @R	Move data memory to A	1	1
MOV A, # data	Move immediate to A	2	2
MOV R, A	Move A to register	1	1
MOV @R, A	Move A to data memory	1	1
MOV R, # data	Move immediate to register	2	2
MOV @R, #data	Move immediate to data memory	2	2
XCH A, R	Exchange A and register	1	1
XCH A, @R	Exchange A and data memory	1	1
XCHD A, @R	Exchange nibble of A and register	1	1
MOVP A, @A	Move to A from current page	1	2

Timer/Counter

Mnemonic	Description	Bytes	Cycles
MOV A, T	Read Timer/Counter	1	1
MOV T, A	Load Timer/Counter	1	1
STRT T	Start Timer	1	1
STRT CNT	Start Counter	1	1
STOP TCNT	Stop Timer/Counter	1	1

Mnemonics	Description	Bytes	Cycles
NOP	No Operation	1	1

PRELIMINARY

8021H

HMOS SINGLE-COMPONENT 8-BIT MICROCOMPUTER

- 8-Bit CPU, ROM, RAM, I/O in Single 28-Pin Package
- Single 5V Supply (+4.5V to 6.5V)
- 8.38 μsec Cycle with 3.58 MHz XTAL; All instructions 1 or 2 Cycles
- 30mA Operation @ 25°C
- Instructions—8048 Subset
- High Current Drive Capability—2 Pins
- 1K x 8 ROM
  64 x 8 RAM
  21 I/O Lines
- Interval Timer/Event Counter
- Clock Generated with Single Inductor, Resistor or Crystal
- Zero-Cross Detection Capability
- Easily Expandable I/O

The Intel® 8021H is a totally self-sufficient 8-bit parallel computer fabricated on a single silicon chip using Intel's advanced N-channel silicon gate HMOS process. The features of the 8021H include a subset of the 8048 features optimized for low cost, high volume applications, plus additional I/O flexibility and power.

The 8021H contains 1K x 8 program memory, 64 x 8 data memory, 21 I/O lines, and an 8-bit timer/event counter, in addition to on-board oscillator and clock circuits. For systems that require extra I/O capability, the 8021H can be expanded using the 8243 or discrete logic.

This microcomputer is designed to be an efficient controller as well as an arithmetic processor. The 8021H has bit handling capability as well as facilities for both binary and BCD arithmetic. Efficient use of program memory results from an instruction set consisting mostly of single-byte instructions and no instructions over two bytes in length.

To minimize development problems and maximize flexibility, an 8021H system can be easily designed using the 8022 emulation board, the EM-2.

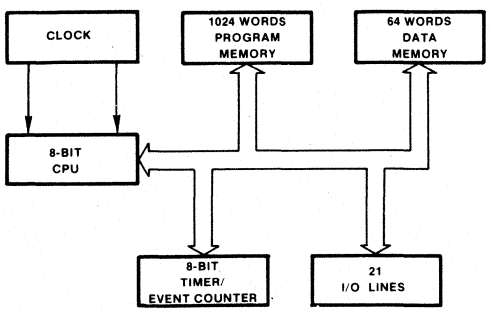


Figure 1.
Block Diagram

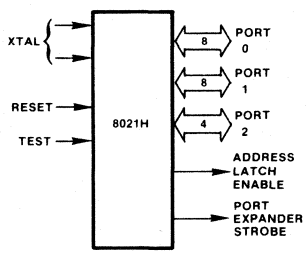


Figure 2.
Logic Symbol

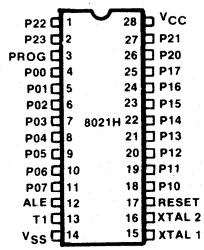


Figure 3. Pin
Configuration

Table 1. Pin Description

Symbol	Pin No.	Function
V_{SS}	14	Circuit GND potential
V_{CC}	28	+5V power supply
PROG	3	Output Strobe for 8243 I/O Expander
P00-P07 Port 0	4-11	8-bit quasi-bidirectional port
P10-P17 Port 1	18-25	8-bit quasi-bidirectional port
P20-P23 Port 2	26-27 1-2	4-bit quasi-bidirectional port P20-P23 also serve as a 4-bit I/O expander bus for 8243
T1	13	Input pin testable using the JT1 and JNT1 instructions. Can be designated the timer/event counter input using the STRT CNT instructions. Also allows zero-crossover sensing of slowly moving AC inputs.
RESET	17	Input used to initialize the processor by clearing status flip-flops and setting program counters to zero. (Active high).
ALE	12	Address Latch Enable. Signal occurring once every 30 input clocks, used as an output clock.
XTAL1	15	One side of crystal resistor or inductor input for internal oscillator. Also input for external source. (Not TTL compatible.)
XTAL2	16	Other side of timing control element.

Table 2. Instruction Set Summary

Accumulator

Mnemonic	Description	Bytes	Cycles
ADD A, R	Add register to A	1	1
ADD A, @R	Add data memory to A	1	1
ADD A, # data	Add immediate to A	2	2
ADDC A, R	Add with carry	1	1
ADDC A, @R	Add with carry	1	1
ADDC A, # data	Add with carry	2	2
ANL A, R	And register to A	1	1
ANL A, @R	And data memory to A	1	1
ANL A, # data	And immediate to A	2	2
ORL A, R	Or register to A	1	1
ORL A, @R	Or data memory to A	1	1
ORL A, # data	Or immediate to A	2	2
XRL A, R	Exclusive or register to A	1	1
XRL A, @R	Exclusive or data memory to A	1	1
XRL A, # data	Exclusive or immediate to A	2	2
INC A	Increment A	1	1
DEC A	Decrement A	1	1
CLR A	Clear A	1	1
CPL A	Complement A	1	1
DA A	Decimal Adjust A	1	1
SWAP A	Swap nibbles of A	1	1
RL A	Rotate A left	1	1
RLC A	Rotate A left through carry	1	1
RR A	Rotate A right	1	1
RRC A	Rotate A right through carry	1	1

Input/Output

Mnemonic	Description	Bytes	Cycles
IN A, P	Input port to A	1	2
OUTL P, A	Output A to port	1	2
MOVD A, P	Input Expander port to A	1	2
MOVD P, A	Output A to Expander port	1	2
ANLD P, A	And A to Expander port	1	2
ORLD P, A	Or A to Expander port	1	2

Registers

Mnemonic	Description	Bytes	Cycles
INC R	Increment register	1	1
INC @R	Increment data memory	1	1

Branch

Mnemonic	Description	Bytes	Cycles
JMP addr	Jump unconditional	2	2
JMPP @A	Jump indirect	1	2
DJNZ R, addr	Decrement register and Jump on R not zero	2	2

Branch

Mnemonic	Description	Bytes	Cycles
JC addr	Jump on Carry = 1	2	2
JNC addr	Jump on Carry = 0	2	2
JZ addr	Jump on A Zero	2	2
JNZ addr	Jump on A not Zero	2	2
JT1 addr	Jump on T1 = 1	2	2
JNT1 addr	Jump on T1 = 0	2	2
JTF addr	Jump on timer flag	2	2

Subroutine

Mnemonic	Description	Bytes	Cycles
CALL	Jump to subroutine	2	2
RET	Return	1	2

Flags

Mnemonic	Description	Bytes	Cycles
CLR C	Clear Carry	1	1
CPL C	Complement Carry	1	1

Table 2. Instruction Set Summary (cont.)

Data Moves

Mnemonic	Description	Bytes	Cycles
MOV A, R	Move register to A	1	1
MOV A, @R	Move data memory to A	1	1
MOV A, # data	Move immediate to A	2	2
MOV R, A	Move A to register	1	1
MOV @R, A	Move A to data memory	1	1
MOV R, # data	Move immediate to register	2	2
MOV @R, # data	Move immediate to data memory	2	2
XCH A, R	Exchange A and register	1	1
XCH A, @R	Exchange A and data memory	1	1
XCHD A, @R	Exchange nibble of A and register	1	1
MOVP A, @A	Move to A from current page	1	2

Timer/Counter

Mnemonic	Description	Bytes	Cycles
MOV A, T	Read Timer/Counter	1	1
MOV T, A	Load Timer/Counter	1	1
STRT T	Start Timer	1	1
STRT CNT	Start Counter	1	1
STOP TCNT	Stop Timer/Counter	1	1

Mnemonic	Description	Bytes	Cycles
NOP	No Operation	1	1

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias ..... 0° C to 70° C
Storage Temperature ..... -65° C to +150° C
Voltage on Any Pin with
Respect to Ground ..... -0.5V to +7V
Power Dissipation 1W

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (T_A = 0° C to 70° C, V_{CC} = 5.5V ± 1V, V_{SS} = 0V)

Symbol	Parameter	Limits			Unit	Test Conditions
		Min	Typ	Max		
V_{IL}	Input Low Voltage	-0.5		0.8	V	
V_{IH}	Input High Voltage (All except XTAL 1 & 2, T1, RESET)	3.0		V_{CC}	V	
V_{IH1}	Input High Voltage (XTAL 1 & 2, T1, RESET)	3.8		V_{CC}	V	
$V_{IH(10\%)}$	Input High Voltage (All except XTAL 1 & 2, T1, RESET)	2.0		V_{CC}	V	V_{CC} = 5.0V ± 10%
$V_{IH1(10\%)}$	Input High Voltage (XTAL 1 & 2, T1, RESET)	3.5		V_{CC}	V	V_{CC} = 5.0V ± 10%
V_{OL}	Output Low Voltage			0.45	V	I_{OL} + 1.6 mA
V_{OL1}	Output Low Voltage (P10, P11)			2.5	V	I_{OL} = 7 mA
V_{OH}	Output High Voltage (All unless Open Drain)	2.4			V	I_{OH} = 40 μA
I_{LO}	Output Leakage Current (Open Drain Option—Port 0)			±10	μA	V_{SS} + 0.4 ≤ V_{IN} ≤ V_{CC}
I_{CC}	V_{CC} Supply Current		30	60	mA	

T1 ZERO CROSS CHARACTERISTICS (T_A = 0° C to 70° C, V_{CC} = 5.5V ± 1V, V_{SS} = 0V, C_L = 80 pF)

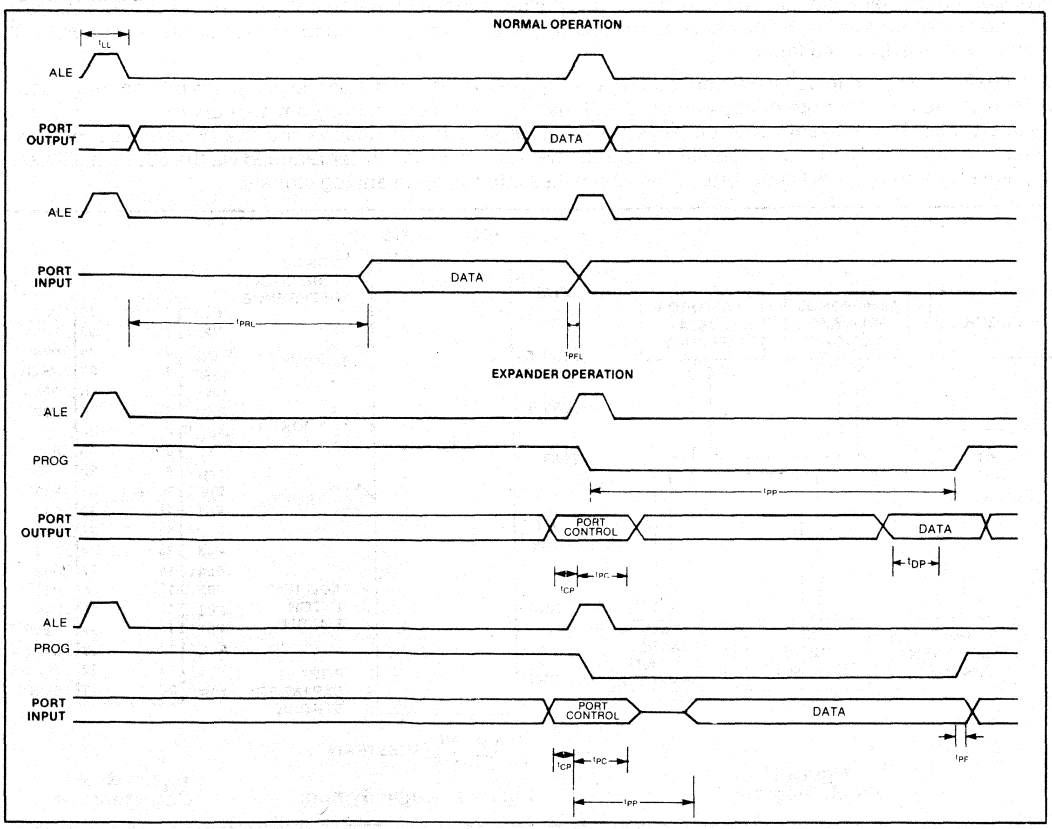
Symbol	Parameter	Min	Max	Unit	Test Conditions
V_{ZX}	Zero-Cross Detection Input (T1)	1	3	V_{PP}	AC Coupled, C = .2μF
A_{ZX}	Zero-Cross Accuracy		±135	mV	60 Hz Sine Wave
F_{ZX}	Zero-Cross Detection Input Frequency (T1)	0.05	1	kHz	
t_{CY}	Cycle Time	8.38	50.0		3.58 MHz XTAL = 8.38 μs t_{CY}

A.C. CHARACTERISTICS (T_A = 0° C to 70° C, V_{CC} = 5.5V ± 1V, V_{SS} = 0V)

Test Conditions: C_L = 80 pF, t_{CY} = 8.38 µs

	Symbol	Parameter	Min.	Max.	Unit	Test Conditions
Normal Operation	t_{CY}	Cycle Time	8.38	50.0	µs	3.58 MHz XTAL
	t_{PRL}	ALE to Time P ± Input Must Be Valid (input setup)		6.5	µs	
	t_{PFL}	Input Data Hold Time	0		µs	
	t_{LL}	ALE Pulse Width	0.8		µs	
	t_R	Reset High	3		t_{CY}	
	R_{XTAL}	Resistor Across XTAL	.5	1	MΩ	
Expander Operation	t_{CP}	Port Control Setup Before Falling Edgle of PROG	0.3		µs	
	t_{PC}	Port Control Hold After Falling Edge of PROG	0.8		µs	
	t_{PR}	PROG to Time P ± Input Must Be Valid	2.0	4.0	µs	
	t_{DP}	Output Data Setup Time	1.0		µs	
	t_{PF}	Input Data Hold Time	0	.15	µs	
	t_{PP}	PROG Pulse Width	6.0		µs	

PORT 2 TIMING



8022
SINGLE COMPONENT 8-BIT MICROCOMPUTER WITH ON-CHIP A/D CONVERTER

- **8-Bit CPU, ROM, RAM, I/O in Single 40-Pin Package**
- **On-Chip 8-Bit A/D Converter; Two Input Channels**
- **8 Comparator Inputs (Port 0)**
- **Zero-Cross Detection Capability**
- **Single 5V Supply (4.5V to 6.5V)**
- **High Current Drive Capability—2 Pins**
- **Two Interrupts—External and Timer**
- **2K x 8 ROM, 64 x 8 RAM, 28 I/O Lines**
- **8.38 μsec Cycle; All Instructions 1 or 2 Cycles**
- **Instructions—8048AH Subset**
- **Interval Timer/Event Counter**
- **Clock Generated with Single Inductor or Crystal**
- **Easily Expanded I/O**

The Intel 8022 is a member of the MCS®-48 family of single chip 8-bit microcomputers. It is designed to satisfy the requirements of low cost, high volume applications which involve analog signals, capacitive touchpanel keyboards, and/or large ROM space. The 8022 addresses these applications by integrating many new functions on-chip, such as A/D conversion, comparator inputs and zero-cross detection.

The features of the 8022 include 2K bytes of program memory (ROM), 64 bytes of data memory (RAM), 28 I/O lines, an on-chip A/D converter with two input channels, an 8-bit port with comparator inputs for interfacing to low voltage capacitive touchpanels or other non-TTL interfaces, external and timer interrupts, and zero-cross detection capability. In addition, it contains the 8-bit interval timer/event counter, on-board oscillator and clock circuitry, single 5V power supply requirement, and easily expandable I/O structure common to all members of the MCS-48 family.

The 8022 is designed to be an efficient controller as well as an arithmetic processor. It has bit handling capability plus facilities for both binary and BCD arithmetic. Efficient use of program memory results from using the MCS-48 instruction set which consists mostly of single byte instructions and has extensive conditional jump and direct table lookup capability. Program memory usage is further reduced via the 8022's hardware implementation of the A/D converter which simplifies interfacing to analog signals.

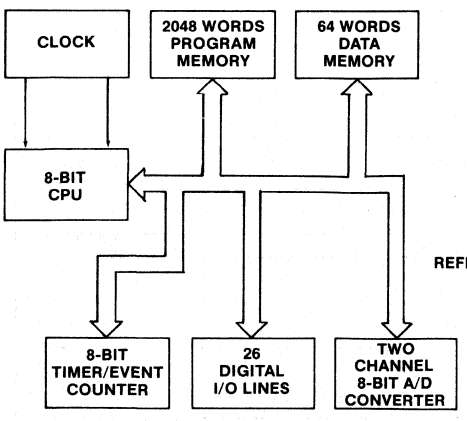


Figure 1. Block Diagram

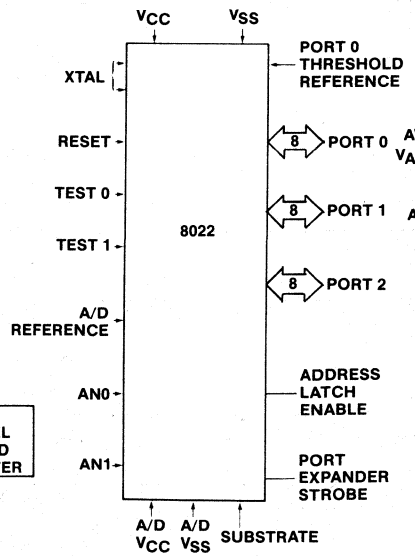


Figure 2. Logic Symbol

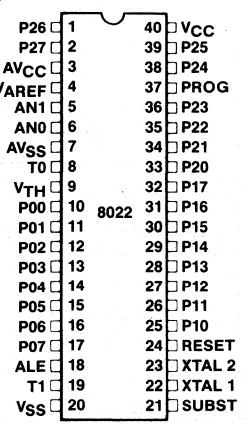
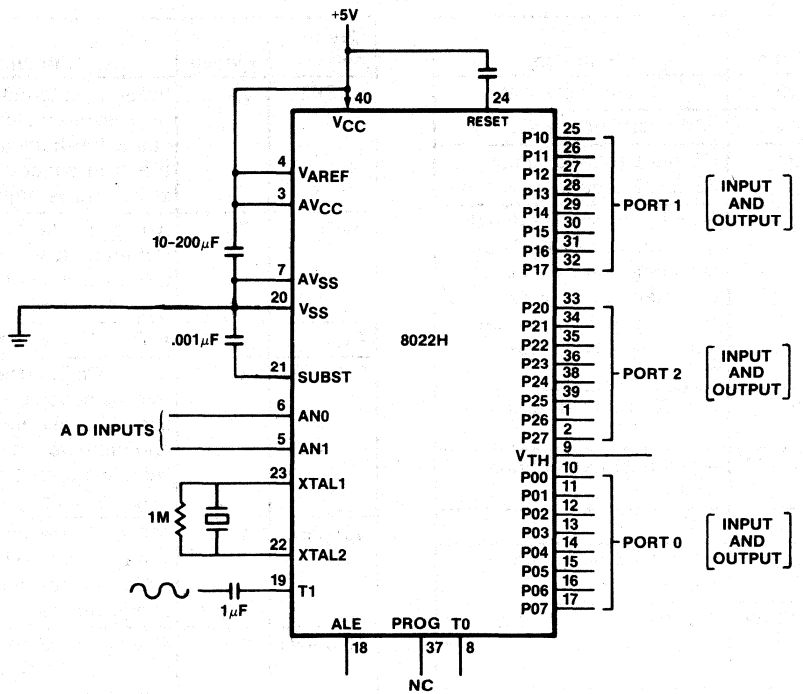


Figure 3. Pin Configuration

Table 1. Pin Description

Designation	Pin No.	Function
V_{SS}	20	Circuit GND potential
V_{CC}	40	+5V circuit power supply.
PROG	37	Output strobe for Intel® 8243 I/O expander.
P00–P07 Port 0	10–17	8-bit open-drain port with comparator inputs. The switching threshold is set externally by V_{TH}. Optional pull-up resistors may be added via ROM mask selection.
V_{TH}	9	Port 0 threshold reference pin.
P10–P17	25–32	8-bit quasi-bidirectional port.
Port 1		
P20–P27	33–36	8-bit quasi-bidirectional port.
Port 2	38–39 1–2	P20–23 also serve as a 4-bit I/O expander for Intel® 8243.
T0	8	Interrupt input and input pin testable using the conditional transfer instructions JT0 and JNT0. Initiates an interrupt following a low level input if interrupt is enabled. Interrupt is disabled after a reset.
T1	19	Input pin testable using the JT1 and JNT1 conditional transfer instructions. Can be designated the timer/event counter input using the STRT CNT instruction. Also serves as the zero-cross detection input to allow zero-crossover sensing of slowly moving AC inputs. Optional pull-up resistor may be added via ROM mask selection.

Designation	Pin No.	Function
RESET	24	Input used to initialize the processor by clearing status flip-flops and setting the program counter to zero. (Active high)
AV_{SS}	7	A/D converter GND Potential. Also establishes the lower limit of the conversion range.
AV_{CC}	3	A/D + 5V power supply.
SUBST	21	Substrate pin used with a bypass capacitor to stabilize the substrate voltage and improve A/D accuracy. Negative potential with respect to V_{SS}.
V_{AREF}	4	A/D converter reference voltage. Establishes the upper limit of the conversion range.
AN0,AN1	6,5	Analog inputs to A/D converter. Software selectable on-chip via SEL AN0 and SEL AN1 instructions.
ALE	18	Address Latch Enable. Signal occurring once every 30 input clocks (once every cycle), used as an output clock.
XTAL1	22	One side of crystal or inductor input for internal oscillator. Also input for external frequency source. (Not TTL compatible.)
XTAL2	23	Other side of timing control element. This pin is not connected when an external frequency source is used.



NOTE: All capacitors should be placed as close as possible to the 8022 chip.

Figure 4. The Stand Alone 8022

Symbols and Abbreviations Used

A	Accumulator	P	Mnemonic for "in-page" Operation
addr	11-Bit Program Memory Address	P_p	Port Designator (P = 0, 1, 2 or 4-7)
AN0, AN1	Analog Input 0, Analog Input 1	R_r	Register Designator (r = 0-7)
CNT	Event Counter	T	Timer
data	8-Bit Number or Expression	T0, T1	Test 0, Test 1
I	Interrupt	#	Immediate Data Prefix
		@	Indirect Address Prefix

Table 2. Instruction Set Summary

Mnemonic	Description	Bytes	Cycle	Hexadecimal Opcode
Accumulator				
ADD A,R_r	Add register to A	1	1	68-6F
ADD A,@R	Add data memory to A	1	1	60-61
ADD A,#data	Add immediate to A	2	2	03
ADDC A,R_r	Add register with carry	1	1	78-7F
ADDC A,@R	Add data memory with carry	1	1	70-71
ADDC A,#data	Data immediate with carry	2	2	13
ANL A,R_r	And register to A	1	1	58-5F
ANL A,@R	Add data memory to A	1	1	50-51
ANL A,#data	And immediate to A	2	2	53
ORL A,R_r	Or register to A	1	1	48-4F
ORL A,@R	Or data memory to A	1	1	40-41
ORL A,#data	Or immediate to A	2	2	43
XRL A,R_r	Exclusive Or register to A	1	1	D8-DF
XRL A,@R	Exclusive Or data memory to A	1	1	D0-D1
XRL A,#data	Exclusive Or immediate to A	2	2	D3
INC A	Increment A	1	1	17
DEC A	Decrement A	1	1	07
CLR A	Clear A	1	1	27
CPL A	Complement A	1	1	37
DA A	Decimal adjust A	1	1	57
SWAP A	Swap nibbles of A	1	1	47
RL A	Rotate A left	1	1	E7
RLC A	Rotate A left through carry	1	1	F7
RR A	Rotate A right	1	1	77
RRC A	Rotate A right through carry	1	1	67
Input/Output				
IN A, P_p	Input port to A	1	2	08,09,0A
OUTL P_pA	Output A to port	1	2	90,39,3A
MOVD A,P_p	Input expander port to A	1	2	0C-0F
MOVD P_p,A	Output A to expander port	1	2	3C-3F
ANLD P_p,A	And A to expander port	1	2	9C-9F
ORLD P_p,A	Or A to expander port	1	2	8C-8F
Registers				
INCR R_r	Increment register	1	1	18-1F
INC @R	Increment data memory	1	1	10-11
Branch				
JMP addr	Jump unconditional	2	2	04,24,44,64 84,A4,C4,E4
JMPP @ A	Jump indirect	1	2	B3
DJNZ R,addr	Decrement register and jump on R not zero	2	2	E8-EF
JC addr	Jump on carry = 1	2	2	F6
JNC addr	Jump on carry = 0	2	2	E6
JZ addr	Jump on A zero	2	2	C6
JNZ addr	Jump on A not zero	2	2	96
JT0	Jump on T0 = 1	2	2	36
JNT0	Jump on T0 = 0	2	2	26
JT1 addr	Jump on T1 = 1	2	2	56
JNT1 addr	Jump on T1 = 0	2	2	46
JTF addr	Jump on timer flag	2	2	16

Mnemonic	Description	Bytes	Cycle	Hexadecimal Opcode
Subroutine				
CALL addr	Jump to subroutine	1	2	14,34,54,74, 94,B4,D4,F4
RET	Return	1	2	83
Flags				
CLR C	Clear carry	1	1k	97
CPL C	Complement carry	1	1	A7
Data Moves				
MOV A,R_r	Move register to A	1	1	F8-FF
MOV A,@R	Move data memory to A	1	1	F0-F1
MOV A, #data	Move immediate to A	2	2	23
MOV R_r,A	Move A to register	1	1	A8-AF
MOV @R,A	Move A to data memory	1	1	A0-A1
MOV R_r, #data	Move immediate to register	2	2	B8-BF
MOV@R, #data	Move immediate to data memory	2	2	B0-B1
XCH A,R_r	Exchange A and register	1	1	28-2F
XCH A,@R	Exchange A and data memory	1	1	20-21
XCHD a,@R	Exchange nibble of A and register	1	1	30-31
MOVP A,@A	Move to A from current page	1	2	A3
Timer/Counter				
MOV A,T	Read timer/counter	1	1	42
MOV T,A	Load timer/counter	1	1	62
STRT T	Start timer	1	1	55
STRT CNT	Start counter	1	1	45
STOP TCNT	Stop timer/counter	1	1	65
A/D Converter				
RAD	Move conversion result register to A	1	2	80
SEL AN0	Select analog input zero	1	1	85
SEL AN1	Select analog input one	1	1	95
Interrupts				
EN I	Enable external interrupt	1	1	05
DIS I	Disable external interrupt	1	1	15
EN TCNTI	Enable timer/counter interrupt	1	1	25
DIS TCNTI	Disable timer/counter interrupt	1	1	35
RET I	Return from interrupt	1	2	93
NOP	No operation	1	1	00

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias . . . 0°C to 70°C

Storage Temperature . . . . . -65°C to +150°C

Voltage On Any Pin With Respect to Ground . . . . . -0.5V to +7V

Power Dissipation . . . . . 1 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS: (T_A = 0°C to 70°C; V_{CC} = 5.5V ± 1V; V_{SS} = 0V; AV_{CC} = 5.5V ± 1V; AV_{SS} = 0V)

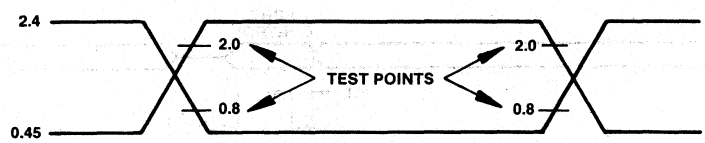
Symbol	Parameter	Limits			Unit	Test Conditions
		Min	Typ	Max		
V_{IL}	Input Low Voltage	-0.5		0.8	V	V_{TH} Floating
V_{IL1}	Input Low Voltage (Port 0)	-0.5		V_{TH}-0.1	V	
V_{IH}	High Voltage (All Except XTAL1, RESET)	2.0		V_{CC}	V	V_{CC} = 5.0V ± 10% V_{TH} Floating
V_{IH1}	Input High Voltage (All Except XTAL1, RESET)	3.0		V_{CC}	V	V_{CC} = 5.5V ± 1V V_{TH} Floating
V_{IH2}	Input High Voltage (Port 0)	V_{TH}+0.1		V_{CC}	V	
V_{IH3}	Input High Voltage (RESET, XTAL1)	3.0		V_{CC}	V	V_{CC} = 5.0V ± 10%
V_{TH}	Port 0 Threshold Reference Voltage	0		.4V_{CC}	V	
V_{OL}	Output Low Voltage			0.45	V	I_{OL} = 1.6 mA
V_{OL1}	Output Low Voltage (P10, P11)			2.5	V	I_{OL} = 7 mA
V_{OH}	Output High Voltage (all unless Open Drain Option—Port 0)	2.4			V	I_{OH} = 50 μA
I_{LI}	Input Current (T1)			±200	μA	$V_{CC} \geq V_{IN} \geq V_{SS}$+.45V
I_{L0}	Output Leakage Current (Open Drain Option—Port 0)			±10	μA	$V_{CC} \geq V_{IN} \geq V_{SS}$+0.45V
I_{CC}	V_{CC} Supply Current		40	70	mA	
I_{AREF}	V_{AREF} Supply Current		.5	1	mA	
AI_{CC}	AV_{CC} Supply Current		5	9	mA	
I_{CC} + I_{AREF}+ AI_{CC}	Total Supply Current		45	80	mA	

A.C. CHARACTERISTICS: (T_A = 0°C to 70°C; V_{CC} = 5.5V ± 1V; V_{SS} = 0V)

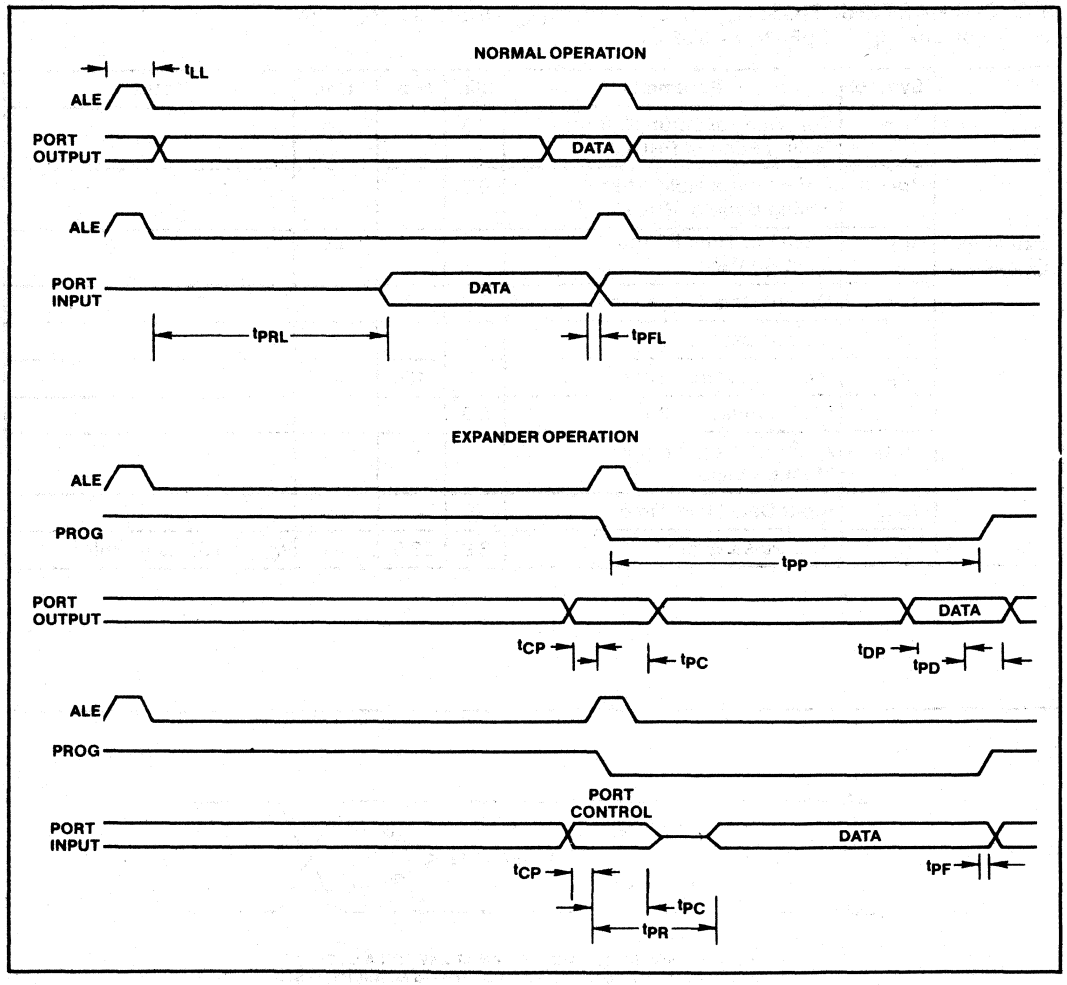
Symbol	Parameter	Min	Max	Unit	Test Conditions
t_{CY}	Cycle Time	8.38	50.0	μs	3 MHz XTAL = 10 μs t_{CY}
V_{ZX}	Zero-Cross Detection Input (T1)	1	3	VACpp	AC Coupled
A_{ZX}	Zero-Cross Accuracy		±135	mV	60 Hz Sine Wave
F_{ZX}	Zero-Cross Detection Input Frequency (T1)	0.05	1	kHz	

A.C. CHARACTERISTICS: (T_A = 0°C to 70°C; V_{CC} = 5.5V ± 1V; V_{SS} = 0V)
Test Conditions: C_L = 80 pF t_{CY} = 8.38 μs

	Symbol	Parameter	Min	Max	Unit	Notes
Expander Operation	t_{CP}	Port Control Setup Before Falling Edge of PROG	0.5		μs	
	t_{PC}	Port Control Hold After Falling Edge of PROG	0.8		μs	
	t_{PR}	PROG to Time P2 Input Must Be Valid		1.0	μs	
	t_{DP}	Output Data Setup Time	7.0		μs	
	t_{PD}	Output Data Hold Time	8.3		μs	
	t_{PF}	Input Data Hold Time	0	.150	μs	
	t_{PP}	PROG Pulse Width	8.3		μs	
	t_{PRL}	ALE to Time P2 Input Must Be Valid		3.6	μs	
	t_{PFL}	Input Data Hold Time	0		μs	
	t_{LL}	ALE Pulse Width	3.9	23.0	μs	t_{CY} = 8.38 μs for min



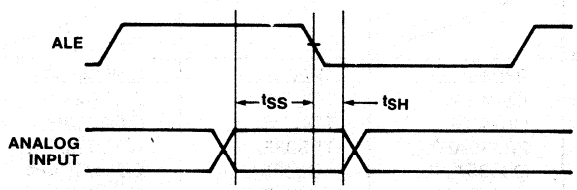
AC TESTING: INPUTS ARE DRIVEN AT 2.4V FOR A LOGIC "1" AND 0.45V FOR A LOGIC "0". OUTPUT TIMING MEASUREMENTS ARE MADE AT 2.0V FOR A LOGIC "1" AND 0.8V FOR A LOGIC "0".



Input and Output for A.C. Tests

A/D CONVERTER: (T_A = 0°C to 70°C; V_{CC} = 5.5V ± 1V; V_{SS} = 0V; AV_{CC} = 5.5V ± 1V; AV_{SS} = 0V; $AV_{CC}/2 \leq V_{AREF} \leq AV_{CC}$)

Parameter	Min	Typ	Max	Unit	Comments
Resolution	8			Bits	
Absolute Accuracy			.8% FSR ± ½ LSB	LSB	(Note 1)
Sample Setup Before Falling Edge of ALE (t_{SS})		0.20		t_{CY}	
Sample Hold After Falling Edge of ALE (t_{SH})		0.10		t_{CY}	
Input Capacitance (AN0, AN1)		1		pF	
Conversion Time	4		4	t_{CY}	



NOTE:

1. The analog input must be maintained at a constant voltage during the sample time (t_{SS} + t_{SH}).

Analog Input Timing

SINGLE-COMPONENT 8-BIT MICROCOMPUTERS

EXPRESS

- **0°C to 70°C Operation**
- **-40°C to 85°C Operation**
- **168 Hr. Burn-In**

- **8048AH/8035AHL**
- **8049AH/8039AHL**
- **8050AH/8040AHL**
- **8748H**
- **8022**
- **8749H**

The new Intel EXPRESS family of single-component 8-bit microcomputers offers enhanced processing options to the familiar 8048AH/8035AHL, 8748H, 8049AH/8039AHL/8749H, 8050AH/8040AHL and 8022 Intel components. These EXPRESS products are designed to meet the needs of those applications whose operating requirements exceed commercial standards, but fall short of military conditions.

The EXPRESS options include the commercial standard and -40°C to 85°C operation with or without 168 ± 8 hours of dynamic burn-in at 125°C per MIL-STD-883B, method 1015. Figure 1 summarizes the option marking designators and package selections.

For a complete description of 8048AH/8035AHL, 8748H, 8049AH/8039AHL, and 8022 features and operating characteristics, refer to the respective standard commercial grade data sheet. This document highlights only the electrical specifications which differ from the respective commercial part.

Temp Range C°	0-70	-40-85	0-70	-40-85
	0	0	168	168
	P8048AH	TP8048AH	QP8048AH	—
	D8048AH	TD8048AH	QD8048AH	LD8048AH
	D8748H	TD8748H	QD8748H	LD8748H
	P8035AHL	TP8035AHL	QP8035AHL	—
	D8035AHL	TD8035AHL	QD8035AHL	LD8035AHL
	P8049AH	TP8049AH	QP8049AH	—
	D8049AH	TD8049AH	QD8049AH	LD8049AH
	D8749H	TD8749H	QD8749H	LD8749H
	P8039AHL	TP8039AHL	QP8039AHL	—
	D8039AHL	TD8039AHL	QD8039AHL	LD8039AHL
	P8050AH	TP8050AH	QP8050AH	—
	D8050AH	TD8050AH	QD8050AH	LD8050AH
	P8040AHL	TP8040AHL	QP8040AHL	—
	D8040AHL	TD8040AHL	QD8040AHL	LD8040AHL
	P8022	TP8022	QP8022	—
	D8022	TD8022	QD8022	LD8022
	P8243	TP8243	QP8243	—
	D8243	TD8243	QD8243	LD8243

* Commercial Grade
P Plastic Package
D Cerdip Package

*Extended Temperature Electrical Specification Deviations**

TD8048AH/TD8035AHL/LD8048AH/LD8035AHL

D.C. CHARACTERISTICS: (T_A = -40° C to 85° C; V_{CC} = V_{DD} = 5V ± 10%; V_{SS} = 0V)

Symbol	Parameter	Limits			Unit	Test Conditions
		Min	Typ	Max		
V_{IH}	Input High Voltage (All Except XTAL1, XTAL2, $\overline{RESET}$)	2.2		V_{CC}	V	
I_{DD}	V_{DD} Supply Current		4	8	mA	
I_{DD} + I_{CC}	Total Supply Current		40	80	mA	

TD8049AH/TD8039AHL/LD8049AH/LD8039AHL

D.C. CHARACTERISTICS: (T_A = -40° C to 85° C; V_{CC} = V_{DD} = 5V ± 10%; V_{SS} = 0V)

Symbol	Parameter	Limits			Unit	Test Conditions
		Min	Typ	Max		
V_{IH}	Input High Voltage (All Except XTAL1, XTAL2, $\overline{RESET}$)	2.2		V_{CC}	V	
I_{DD}	V_{DD} Supply Current		5	10	mA	
I_{DD} + I_{CC}	Total Supply Current		50	100	mA	

TD8050AH/TD8040AHL/LD8050AH/LD8040AHL

D.C. CHARACTERISTICS: (T_A = -40° C to 85° C; V_{CC} = V_{DD} = 5V ± 10%; V_{SS} = 0V)

Symbol	Parameter	Limits			Unit	Test Conditions
		Min	Typ	Max		
V_{IH}	Input High Voltage (All Except XTAL1, XTAL2, $\overline{RESET}$)	2.2		V_{CC}	V	
I_{DD}	V_{DD} Supply Current		10	20	mA	
I_{DD} + I_{CC}	Total Supply Current		75	120	mA	

*Extended Temperature Electrical Specification Deviations**

TD8748H/LD8748H

D.C. CHARACTERISTICS: (T_A = -40° C to 85° C; V_{CC} = V_{DD} = 5V ± 10%; V_{SS} = 0V)

Symbol	Parameter	Limits			Unit	Test Conditions
		Min	Typ	Max		
V_{IH}	Input High Voltage (All Except XTAL1, XTAL2, $\overline{RESET}$)	2.2		V_{CC}	V	
I_{DD} + I_{CC}	Total Supply Current		50	130	mA	

TD8749H/LD8749H

D.C. CHARACTERISTICS: (T_A = -40° C to 85° C; V_{CC} = V_{DD} = 5V ± 10%; V_{SS} = 0V)

Symbol	Parameter	Limits			Unit	Test Conditions
		Min	Typ	Max		
V_{IH}	Input High Voltage (All Except XTAL1, XTAL2, $\overline{RESET}$)	2.2		V_{CC}	V	
I_{DD} + I_{CC}	Total Supply Current		75	150	mA	

*Extended Temperature Electrical Specification Deviations**

TD8022/LD8022

D.C. CHARACTERISTICS: (T_A = -40°C to 85°C; V_{CC} = 5.5V ± 1V; V_{SS} = 0V)

Symbol	Parameter	Limits			Unit	Test Conditions
		Min	Typ	Max		
V_{IL1}	Input Low Voltage (Port 0)	-0.5		V_{TH}-0.2	V	
V_{IH}	Input High Voltage (All Except XTAL1, RESET)	2.3		V_{CC}	V	V_{CC} = 5.0V ± 10% V_{TH} Floating
V_{IH1}	Input High Voltage (All Except XTAL1, RESET)	3.8		V_{CC}	V	V_{CC} = 5.5V ± 1V V_{TH} Floating
V_{IH2}	Input High Voltage (Port 0)	V_{TH}+0.2		V_{CC}	V	
V_{IH3}	Input High Voltage (RESET, XTAL1)	3.8		V_{CC}	V	
V_{IL}	Input Low Voltage	-0.5		0.5	V	
V_{OL}	Output Low Voltage			0.45	V	I_{OL} = 0.8 mA
V_{OL1}	Output Low Voltage (P10, P11)			2.5	V	I_{OL} = 3 mA
V_{OH}	Output High Voltage (All unless open drain option Port 0)	2.4			V	I_{OH} = 30 μA
I_{LI}	Input Current (T1)			±700	μA	$V_{CC} \geq V_{IN} \geq V_{SS}$ + 0.45V
I_{LI1}	Input Current to Ports			500	μA	V_{IN} = 0.45V
I_{CC}	V_{CC} Supply Current			120	mA	

A.C. CHARACTERISTICS: (T_A = -40°C to 85°C; V_{CC} = 5.5V ± 1V; V_{SS} = 0V)

Symbol	Parameter	Min	Max	Unit	Test Conditions
t_{CY}	Cycle Time	8.38	50.0	μs	3.58 MHz XTAL = 8.38 μs t_{CY}
V_{T1}	Zero-Cross Detection Input (T1)	1	3	VACpp	AC Coupled
A_{ZC}	Zero-Cross Accuracy		±200	mV	60 Hz Sine Wave
F_{T1}	Zero-Cross Detection Input Frequency (T1)	0.05	1	kHz	
t_{LL}	ALE Pulse Width	3.9	23.0	μs	t_{CY} = 8.38 μs for min

NOTE: Control Outputs: C_L = 80 pf; T_{CY} = 8.38 μsec.

A/D CONVERTER CHARACTERISTICS: (AVCC = 5.5V ± 1V; AVSS = 0V; AVCC/2 ≤ VAREF ≤ AVCC)

Parameter	Limits			Unit	Test Conditions
	Min	Typ	Max		
Absolute Accuracy			1.6% FSR ± ½ LSB	LSB	

NOTE: The analog input must be maintained at a constant voltage during the sample time (tss + tsh).
*Refer to individual commercial grade data sheets for complete operating characteristics.

8243
MCS®-48 INPUT/OUTPUT EXPANDER

AND EXPRESS

- **0° C to 70° C Operation**
- **-40° C to 85° C Operation**
- **168 Hr. Burn-In**

The new Intel EXPRESS 8243 family is a process enhanced version of the familiar MCS®-48 input/output expander. The EXPRESS versions are designed to meet the needs of those applications whose operating requirements exceed commercial standards, but fall short of military conditions.

The EXPRESS options include the commercial standard and -40° C to 85° C operation with or without 168 ± 8 hours of dynamic burn-in a 125° C per MIL-STD-883B, method 1015. Figure 3 summarizes the option marking designators and package selections.

All EXPRESS features and operating characteristics are identical to the standard, commercial grade part except the V_{CC} supply current. This specification is increased by 5mA on -40° C to 85° C EXPRESS options only: 15mA typical/25mA maximum.

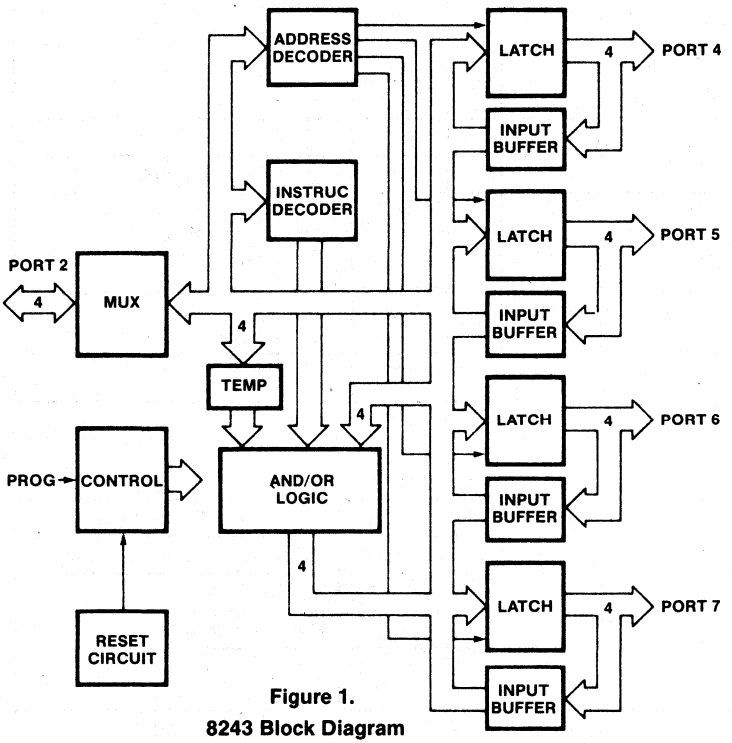


Figure 1.
8243 Block Diagram

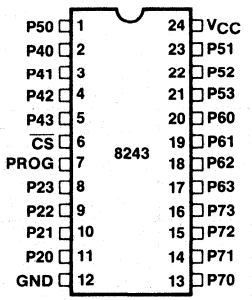


Figure 2.
8243 Pin Configuration

TEMP RANGE 0°C / BI HOURS	0-70 / 0	-40-85 / 0	0-70 / 168	-40-85 / 168
*COMMERCIAL GRADE P=PLASTIC PACKAGE D=CERDIP PACKAGE	*P8243	TP8243	QP8243	
	*D8243	TD8243	QD8243	LD8243

Figure 3. Available 8243 EXPRESS Options

Table 1. Pin Description

Symbol	Pin No.	Function
PROG	7	Clock Input. A high to low transition on PROG signifies that address and control are available on P20-P23, and a low to high transition signifies that data is available on P20-P23.
$\overline{CS}$	6	Chip Select Input. A high on CS inhibits any change of output or internal status.
P20-P23	11-8	Four (4) bit bi-directional port contains the address and control bits on a high to low transition of PROG. During a low to high transition contains the data for a selected output port if a write operation, or the data from a selected port before the low to high transition if a read operation.
GND	12	0 volt supply.
P40-P43	2-5	Four (4) bit bi-directional I/O ports.
P50-P53 P60-P63 P70-P73	1, 23-21 20-17 13-16	May be programmed to be input (during read); low impedance latched output (after write), or a tri-state (after read). Data on pins P20-P23 may be directly written, ANDed or ORed with previous data.
V_{CC}	24	+5 volt supply.

FUNCTIONAL DESCRIPTION

General Operation

The 8243 contains four 4-bit I/O ports which serve as an extension of the on-chip I/O and are addressed as port 4-7. The following operations may be performed on these ports:

- Transfer Accumulator to Port.
- Transfer Port to Accumulator.
- AND Accumulator to Port.
- OR Accumulator to Port.

All communication between the 8048AH and the 8243 occurs over Port 2 (P20-P23) with timing provided by an output pulse on the PROG pin of the processor. Each transfer consists of two 4-bit nibbles:

The first containing the "op code" and port address and the second containing the actual 4-bits of data. A high to low transition of the PROG line indicates that address is present while a low to high transition indicates the presence of data. Additional 8243's may be added to the 4-bit bus and chip selected using additional output lines from the 8048AH/8748H/8035AHL.

Power On Initialization

Initial application of power to the device forces input/output ports 4, 5, 6, and 7 to the tri-state and port 2 to the input mode. The PROG pin may be either high or low when power is applied. The first high to low transition of PROG causes device to exit power on mode. The power on sequence is initiated if V_{CC} drops below 1V.

P21	P20	Address Code	P23	P22	Instruction Code
0	0	Port 4	0	0	Read
0	1	Port 5	0	1	Write
1	0	Port 6	1	0	ORLD
1	1	Port 7	1	1	ANLD

Write Modes

The device has three write modes. MOVD Pi,A directly writes new data into the selected port and old data is lost. ORLD Pi,A takes new data, OR's it with the old data and then writes it to the port. ANLD Pi,A takes new data, AND's it with the old data and then writes it to the port. Operation code and port address are latched from the input port 2 on the high to low transition of the PROG pin. On the low to high transition of PROG data on port 2 is transferred to the logic block of the specified output port.

After the logic manipulation is performed, the data is latched and outputed. The old data remains latched until new valid outputs are entered.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias . . . 0°C to 70°C

Storage Temperature . . . . . -65°C to +150°C

Voltage On Any Pin With Respect to Ground . . . . . -0.5V to +7V

Power Dissipation . . . . . 1 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

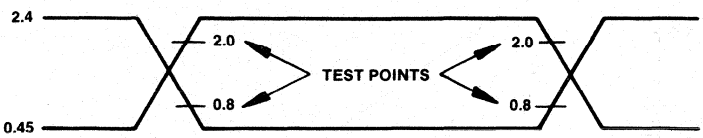
D.C. CHARACTERISTICS: (T_A = 0°C to 70°C, V_{CC} = 5V 10%)

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
V_{IL}	Input Low Voltage	-0.5		0.8	V	
V_{IH}	Input High Voltage	2.0		V_{CC}+0.5	V	
V_{OL1}	Output Low Voltage Ports 4-7			0.45	V	I_{OL} = 4.5 mA*
V_{OL2}	Output Low Voltage Port 7			1	V	I_{OL} = 20 mA
V_{OH1}	Output High Voltage Ports 4-7	2.4			V	I_{OH} = 240 μA
I_{IL1}	Input Leakage Ports 4-7	-10		20	μA	V_{in} = V_{CC} to 0V
I_{IL2}	Input Leakage Port 2, CS, PROG	-10		10	μA	V_{in} = V_{CC} to 0V
V_{OL3}	Output Low Voltage Port 2			45	V	I_{OL} = 0.6 mA
I_{CC}	V_{CC} Supply Current		10	20	mA	NOTE 1
V_{OH2}	Output Voltage Port 2	2.4				I_{OH} = 100μA
I_{OL}	Sum of all I_{OL} from 16 outputs			72	mA	4.5 mA Each Pin

*See following graph for additional sink current capability.

A.C. CHARACTERISTICS: (T_A = 0°C to 70°C, V_{CC} = 5V 10%)

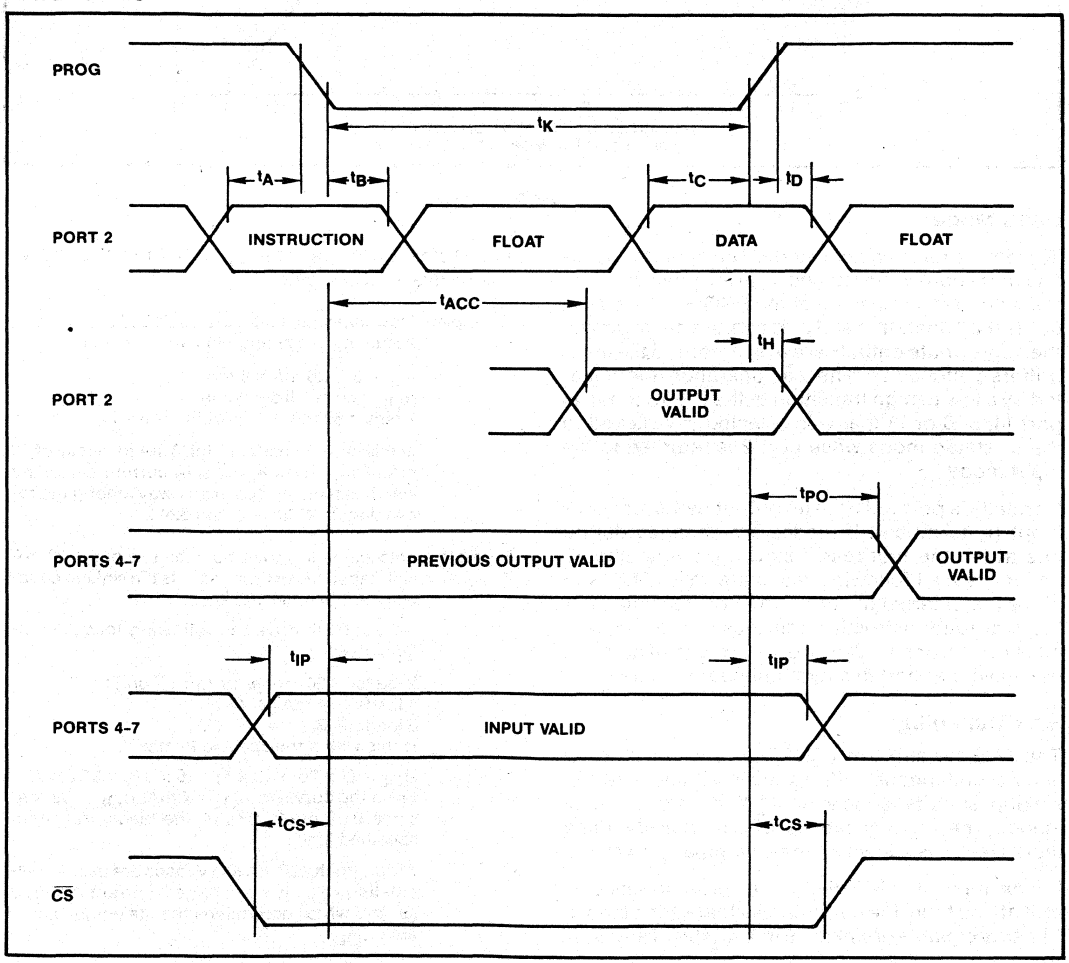
Symbol	Parameter	Min	Max	Units	Test Conditions
t_A	Code Valid Before PROG	50		ns	80 pF Load
t_B	Code Valid After PROG	60		nS	20 pF Load
t_C	Data Valid Before PROG	200		ns	80 pF Load
t_D	Data Valid After PROG	20		ns	20 pF Load
t_H	Floating After PROG	0	150	nS	20 pF Load
t_K	PROG Negative Pulse Width	700		ns	
t_{CS}	CS Valid Before/After PROG	50		ns	
t_{PO}	Ports 4-7 Valid After PROG		700	ns	100 pF Load
t_{LP1}	Ports 4-7 Valid Before/After PROG	100		ns	
t_{ACC}	Port 2 Valid After PROG		650	ns	80 pF Load



AC TESTING: INPUTS ARE DRIVEN AT 2.4V FOR A LOGIC "1" AND 0.45V FOR A LOGIC "0". OUTPUT TIMING MEASUREMENTS ARE MADE AT 2.0V FOR A LOGIC "1" AND 0.8V FOR A LOGIC "0".

NOTE 1: I_{CC} (-40° to 85° C EXPRESS options) 15mA typical/25mA maximum.

WAVEFORMS



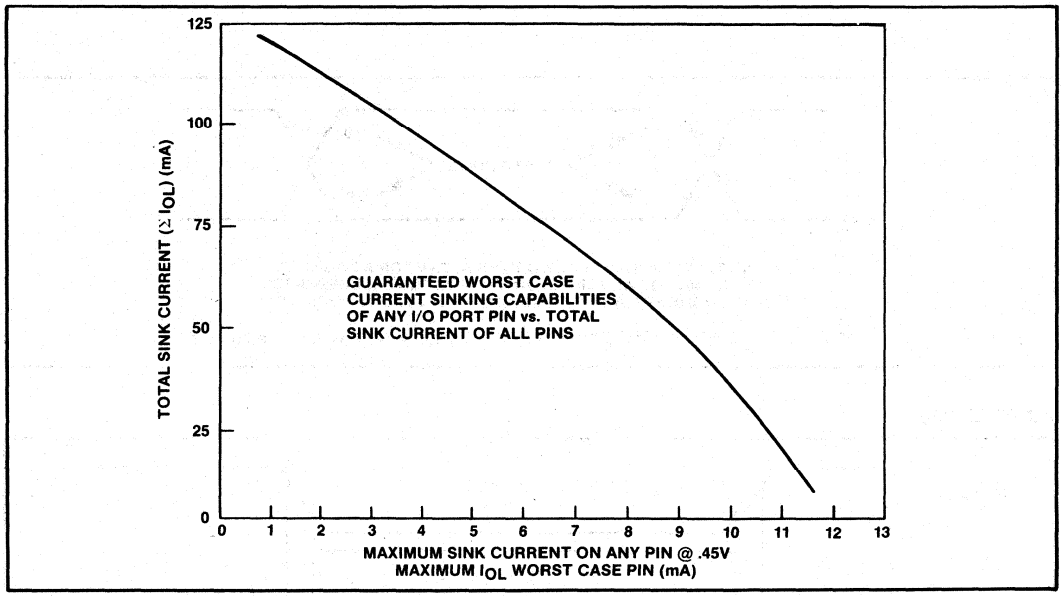


Figure 4.

Read Mode

The device has one read mode. The operation code and port address are latched from the input port 2 on the high to low transition of the PROG pin. As soon as the read operation and port address are decoded, the appropriate outputs are tri-stated, and the input buffers switched on. The read operation is terminated by a low to high transition of the PROG pin. The port (4, 5, 6 or 7) that was selected is switched to the tri-stated mode while port 2 is returned to the input mode.

Normally, a port will be in an output (write mode) or input (read mode). If modes are changed during operation, the first read following a write should be ignored; all following reads are valid. This is to allow the external driver on the port to settle after the first read instruction removes the low impedance drive from the 8243 output. A read of any port will leave that port in a high impedance state.

Sink Capability

The 8243 can sink 5 mA @ .45V on each of its 16 I/O lines simultaneously. If, however, all lines are not sinking simultaneously or all lines are not fully loaded, the drive capability of any individual line increases as is shown by the accompanying curve.

For example, if only 5 of the 16 lines are to sink current at one time, the curve shows that each of those 5 lines is capable of sinking 9 mA @ .45V (if any lines are to sink 9 mA the total I_{OL} must not exceed 45 mA or five 9 mA loads).

Example: How many pins can drive 5 TTL loads (1.6 mA) assuming remaining pins are unloaded?

I_{OL} = 5 x 1.6 mA = 8 mA
ϵI_{OL} = 60 mA from curve
pins = 60 mA ÷ 8 mA/pin = 7.5 = 7

In this case, 7 lines can sink 8 mA for a total of 56 mA. This leaves 4 mA sink current capability which can be divided in any way among the remaining 8 I/O lines of the 8243.

Example: This example shows how the use of the 20 mA sink capability of Port 7 affects the sinking capability of the other I/O lines.

An 8243 will drive the following loads simultaneously.

2 loads — 20 mA @ 1V (port 7 only)
8 loads — 4 mA @ .45V
6 loads — 3.2 mA @ .45V
Is this within the specified limits?

ϵI_{OL} = (2 x 20) + (8 x 4) + (6 x 3.2) = 91.2 mA.
From the curve: for I_{OL} = 4 mA, $\epsilon I_{OL} \approx$ 93 mA.
since 91.2 mA $<$ 93 mA the loads are within specified limits.

Although the 20 mA @ 1V loads are used in calculating ϵI_{OL}, it is the largest current required @ .45V which determines the maximum allowable ϵI_{OL}.

NOTE: A 10 to 50K Ω pullup resistor to +5V should be added to 8243 outputs when driving to 5V CMOS directly.

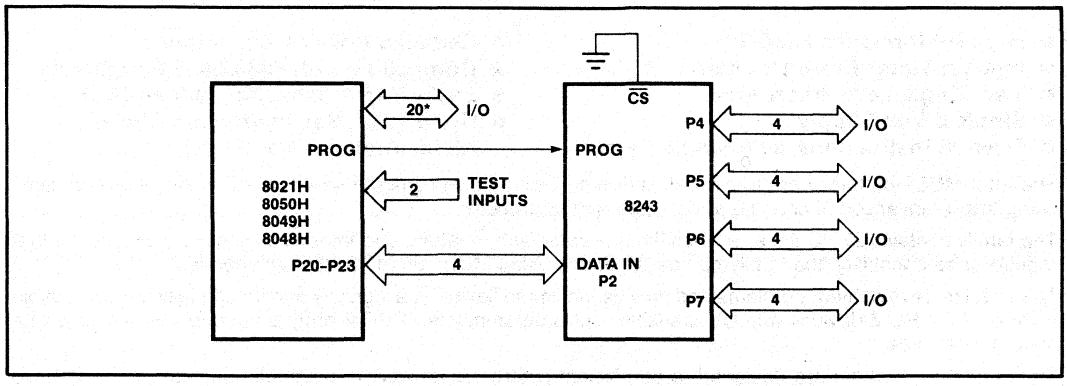


Figure 5. Expander Interface

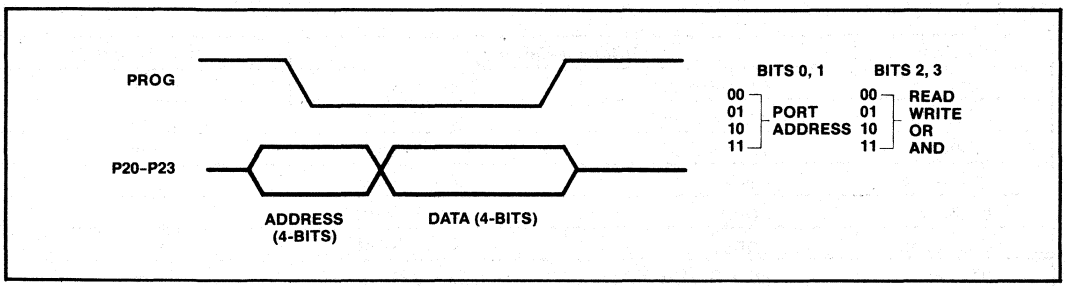


Figure 6. Output Expander Timing

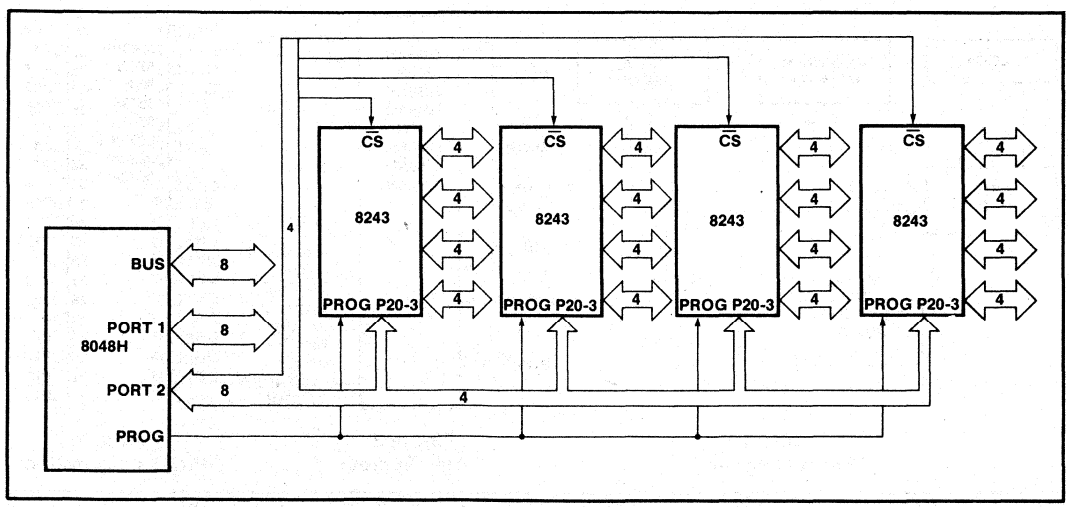


Figure 7. Using Multiple 8243's

8048AH/8748H/8035AHL/8049AH 8749H/8039AHL/8050AH/8040AHL HMOS SINGLE-COMPONENT 8-BIT MICROCOMPUTER

- **High Performance HMOS II**
- **Interval Timer/Event Counter**
- **Two Single Level Interrupts**
- **Single 5-Volt Supply**
- **Over 96 Instructions; 90% Single Byte**
- **Reduced Power Consumption**
- **Compatible with 8080/8085 Peripherals**
- **Easily Expandable Memory and I/O**
- **Up to 1.36 μSec Instruction Cycle All Instructions 1 or 2 cycles**

The Intel MCS®-48 family are totally self-sufficient, 8-bit parallel computers fabricated on single silicon chips using Intel's advanced N-channel silicon gate HMOS process.

The family contains 27 I/O lines, an 8-bit timer/counter, and on-board oscillator/clock circuits. For systems that require extra capability, the family can be expanded using MCS®-80/MCS®-85 peripherals.

To minimize development problems and provide maximum flexibility, a logically and functionally pin-compatible version of the ROM devices with UV-erasable user-programmable EPROM program memory is available with minor differences.

These microcomputers are designed to be efficient controllers as well as arithmetic processors. They have extensive bit handling capability as well as facilities for both binary and BCD arithmetic. Efficient use of program memory results from an instruction set consisting mostly of single byte instructions and no instructions over 2 bytes in length.

Device	Internal Memory		RAM Standby
8050AH	4K × 8 ROM	256 × 8 RAM	yes
8049AH	2K × 8 ROM	128 × 8 RAM	yes
8048AH	1K × 8 ROM	64 × 8 RAM	yes
8040AHL	none	256 × 8 RAM	yes
8039AHL	none	128 × 8 RAM	yes
8035AHL	none	64 × 8 RAM	yes
8749H	2K × 8 EPROM	128 × 8 RAM	no
8748H	1K × 8 EPROM	64 × 8 RAM	no

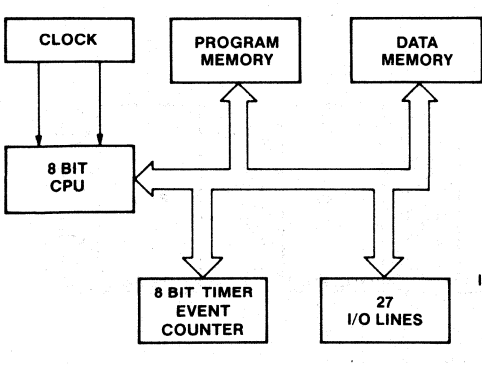


Figure 1. Block Diagram

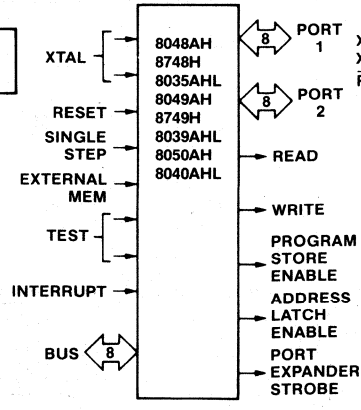


Figure 2. Logic Symbol

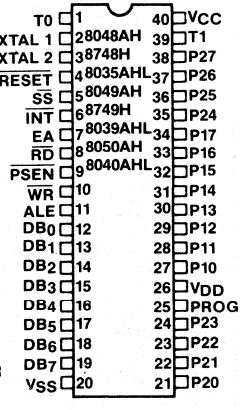


Figure 3. Pin Configuration

January 1983

Order Number 210983-001

Table 1. Pin Description

Symbol	Pin No.	Function	Device
V_{SS}	20	Circuit GND potential	All
V_{DD}	26	+5V during normal operation.	All
		Low power standby pin.	8048AH 8035AHL 8049AH 8039AHL 8050AH 8040AHL
		Programming power supply (+21V).	8748H 8749H
V_{CC}	40	Main power supply; +5V during operation and programming.	All
PROG	25	Output strobe for 8243 I/O expander.	All
		Program pulse (+18V) input pin during programming.	8748H 8749H (See Note)
P10-P17 Port 1	27-34	8-bit quasi-bidirectional port.	All
P20-P23 P24-P27 Port 2	21-24 35-38	8-bit quasi-bidirectional port. P20-P23 contain the four high order program counter bits during an external program memory fetch and serve as a 4-bit I/O expander bus for 8243.	All
DB0-DB7 BUS	12-19	True bidirectional port which can be written or read synchronously using the $\overline{RD}$, $\overline{WR}$ strobes. The port can also be statically latched.	All
(Con't)		Contains the 8 low order program counter bits during an external program memory fetch, and receives the addressed instruction under the control of $\overline{PSEN}$. Also contains the address and data during an external RAM data store instruction, under control of ALE, $\overline{RD}$, and $\overline{WR}$.	
T0	1	Input pin testable using the conditional transfer instructions JT0 and JNT0. T0 can be designated as a clock output using ENT0 CLK instruction	All
		Used during programming.	8748H 8749H
T1	39	Input pin testable using the JT1, and JNT1 instructions. Can be designated the timer/counter input using the STRT CNT instruction.	All
$\overline{INT}$	6	Interrupt input. Initiates an interrupt if interrupt is enabled. Interrupt is disabled after a reset. Also testable with conditional jump instruction. (Active low) interrupt must remain low for at least 3 machine cycles for proper operation.	All

Table 1. Pin Description (Continued)

Symbol	Pin No.	Function	Device
$\overline{RD}$	8	Output strobe activated during a BUS read. Can be used to enable data onto the bus from an external device. Used as a read strobe to external data memory. (Active low)	All
$\overline{RESET}$	4	Input which is used to initialize the processor. (Active low) (Non TTL V_{IH})	All
		Used during power down.	8048AH 8035AHL 8049AH 8039AHL 8050AH 8040AHL
		Used during programming.	8748H 8749H
		Used during ROM verification.	8048AH 8748H 8049AH 8749H 8050AH
$\overline{WR}$	10	Output strobe during a bus write. (Active low) Used as write strobe to external data memory.	All
ALE	11	Address latch enable. This signal occurs once during each cycle and is useful as a clock output. The negative edge of ALE strobes address into external data and program memory.	All
$\overline{PSEN}$	9	Program store enable. This output occurs only during a fetch to external program memory. (Active low)	ALL
$\overline{SS}$	5	Single step input can be used in conjunction with ALE to "single step" the processor through each instruction.	All
		(Active low) Used in sync mode	8048AH 8035AHL 8049AH 8039AHL 8050AH 8040AHL
EA,	7	External access input which forces all program memory fetches to reference external memory. Useful for emulation and debug. (Active high)	All
		Used during (18V) programming	8748H 8749H
		Used during ROM verification (12V)	8048AH 8049AH 8050AH
XTAL1	2	One side of crystal input for internal oscillator. Also input for external source. (Non TTL V_{IH})	All
XTAL2	3	Other side of crystal input.	All

NOTE: On the 8749H, PROG must be clamped to V_{CC} when not programming. A diode should be used when using an 8243; otherwise, a direct connection is permissible.

Table 2. Instruction Set

Accumulator

Mnemonic	Description	Bytes	Cycles
ADD A, R	Add register to A	1	1
ADD A, @R	Add data memory to A	1	1
ADD A, # data	Add immediate to A	2	2
ADDC A, R	Add register with carry	1	1
ADDC A, @R	Add data memory with carry	1	1
ADDC A, # data	Add immediate with carry	2	2
ANL A, R	And register to A	1	1
ANL A, @R	And data memory to A	1	1
ANL A, # data	And immediate to A	2	2
ORL A, R	Or register to A	1	1
ORL A @R	Or data memory to A	1	1
ORL A, # data	Or immediate to A	2	2
XRL A, R	Exclusive or register to A	1	1
XRL A, @R	Exclusive or data memory to A	1	1
XRL, A, # data	Exclusive or immediate to A	2	2
INC A	Increment A	1	1
DEC A	Decrement A	1	1
CLR A	Clear A	1	1
CPL A	Complement A	1	1
DA A	Decimal adjust A	1	1
SWAP A	Swap nibbles of A	1	1
RL A	Rotate A left	1	1
RLC A	Rotate A left through carry	1	1
RR A	Rotate A right	1	1
RRC A	Rotate A right through carry	1	1

Input/Output

Mnemonic	Description	Bytes	Cycles
IN A, P	Input port to A	1	2
OUTL P, A	Output A to port	1	2
ANL P, # data	And immediate to port	2	2
ORL P, # data	Or immediate to port	2	2
INS A, BUS	Input BUS to A	1	2
OUTL BUS, A	Output A to BUS	1	2
ANL BUS, # data	And immediate to BUS	2	2
ORL BUS, # data	Or immediate to BUS	2	2
MOVD A, P	Input expander port to A	1	2
MOVD P, A	Output A to expander port	1	2
ANLD P, A	And A to expander port	1	2
ORLD P, A	Or A to expander port	1	2

Registers

Mnemonic	Description	Bytes	Cycles
INC R	Increment register	1	1
INC @R	Increment data memory	1	1
DEC R	Decrement register	1	1

Branch

Mnemonic	Description	Bytes	Cycles
JMP addr	Jump unconditional	2	2
JMPP @A	Jump indirect	1	2
DJNZ R, addr	Decrement register and skip	2	2
JC addr	Jump on carry = 1	2	2
JNC addr	Jump on carry = 0	2	2
JZ addr	Jump on A zero	2	2
JNZ addr	Jump on A not zero	2	2
JT0 addr	Jump on T0 = 1	2	2
JNT0 addr	Jump on T0 = 0	2	2
JT1 addr	Jump on T1 = 1	2	2
JNT1 addr	Jump on T1 = 0	2	2
JF0 addr	Jump on F0 = 1	2	2
JF1 addr	Jump on F1 = 1	2	2
JTF addr	Jump on timer flag	2	2
JNI addr	Jump on INT = 0	2	2
JBb addr	Jump on accumulator bit	2	2

Subroutine

Mnemonic	Description	Bytes	Cycles
CALL addr	Jump to subroutine	2	2
RET	Return	1	2
RETR	Return and restore status	1	2

Flags

Mnemonic	Description	Bytes	Cycles
CLR C	Clear carry	1	1
CPL C	Complement carry	1	1
CLR F0	Clear flag 0	1	1
CPL F0	Complement flag 0	1	1
CLR F1	Clear flag 1	1	1
CPL F1	Complement flag 1	1	1

Table 2. Instruction Set (Continued)

Data Moves

Mnemonic	Description	Bytes	Cycles
MOV A, R	Move register to A	1	1
MOV A, @R	Move data memory to A	1	1
MOV A, # data	Move immediate to A	2	2
MOV R, A	Move A to register	1	1
MOV @R, A	Move A to data memory	1	1
MOV R, # data	Move immediate to register	2	2
MOV @R, # data	Move immediate to data memory	2	2
MOV A, PSW	Move PSW to A	1	1
MOV PSW, A	Move A to PSW	1	1
XCH A, R	Exchange A and register	1	1
XCH A, @R	Exchange A and data memory	1	1
XCHD A, @R	Exchange nibble of A and register	1	1
MOVX A, @R	Move external data memory to A	1	2
MOVX @R, A	Move A to external data memory	1	2
MOVP A, @A	Move to A from current page	1	2
MOVP3 A, @	Move to A from page 3	1	2

Timer/Counter

Mnemonic	Description	Bytes	Cycles
MOV A, T	Read timer/counter	1	1
MOV T, A	Load timer/counter	1	1
STRT T	Start timer	1	1
STRT CNT	Start timer	1	1
STOP TCNT	Stop timer/counter	1	1
EN TCNTI	Enable timer/counter interrupt	1	1
DIS TCNTI	Disable timer/counter interrupt	1	1

Control

Mnemonic	Description	Bytes	Cycles
EN I	Enable external interrupt	1	1
DIS I	Disable external interrupt	1	1
SEL RB0	Select register bank 0	1	1
SEL RB1	Select register bank 1	1	1
SEL MB0	Select memory bank 0	1	1
SEL MB1	Select memory bank 1	1	1
ENT0 CLK	Enable clock output on T0	1	1

Mnemonic	Description	Bytes	Cycles
NOP	No operation	1	1
IDL	Select Idle Operation	1	1

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias . . . 0° C to 70° C
Storage Temperature . . . . . -65° C to +150° C
Voltage On Any Pin With Respect to Ground . . . . . -0.5V to +7V
Power Dissipation . . . . . 1.5 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of device at these or any other conditions above those indicated in the operational sections of this specification is not implied.*

D.C. CHARACTERISTICS: (T_A = 0° C to 70° C; V_{CC} = V_{DD} = 5V ± 10%; V_{SS} = 0V)

Symbol	Parameter	Limits			Unit	Test Conditions	Device
		Min	Typ	Max			
V_{IL}	Input Low Voltage (All Except RESET, X1, X2)	-.5		.8	V		All
V_{IL1}	Input Low Voltage ($\overline{RESET}$, X1, X2)	-.5		.6	V		All
V_{IH}	Input High Voltage (All Except XTAL1, XTAL2, $\overline{RESET}$)	0		V_{CC}	V		All
V_{IH1}	Input High Voltage (X1, X2, $\overline{RESET}$)	3.8		V_{CC}	V		All
V_{OL}	Output Low Voltage (BUS)			.45	V	I_{OL} = 2.0 mA	All
V_{OL1}	Output Low Voltage ($\overline{RD}$, $\overline{WR}$, $\overline{PSEN}$, ALE)			.45	V	I_{OL} = 1.8 mA	All
V_{OL2}	Output Low Voltage (PROG)			.45	V	I_{OL} = 1.0 mA	All
V_{OL3}	Output Low Voltage (All Other Outputs)			.45	V	I_{OL} = 1.6 mA	All
V_{OH}	Output High Voltage (BUS)	2.4			V	I_{OH} = -400 μA	All
V_{OH1}	Output High Voltage ($\overline{RD}$, $\overline{WR}$, $\overline{PSEN}$, ALE)	2.4			V	I_{OH} = -100 μA	All
V_{OH2}	Output High Voltage (All Other Outputs)	2.4			V	I_{OH} = -40 μA	All

D.C. CHARACTERISTICS: (T_A = 0° C to 70° C; V_{CC} = V_{DD} = 5V ± 10%; V_{SS} = 0V) (Continued)

Symbol	Parameter	Limits			Unit	Test Conditions	Device
		Min	Typ	Max			
I_{L1}	Leakage Current (T1, $\overline{INT}$)			±10	μA	$V_{SS} \leq V_{IN} \leq V_{CC}$	All
I_{LI1}	Input Leakage Current (P10-P17, P20-P27, EA, $\overline{SS}$)			-500	μA	$V_{SS}+.45 \leq V_{IN} \leq V_{CC}$	All
I_{LI2}	Input Leakage Current $\overline{RESET}$	20		-300	μA	$V_{SS} \leq V_{IN} \leq V_{CC}$	All
I_{L0}	Leakage Current (BUS, T0) (High Impedance State)			±10	μA	$V_{SS} \leq V_{IN} \leq V_{CC}$	All
I_{DD}	V_{DD} Supply Current (RAM Standby)		3	5	mA		8048AH 8035AHL
			4	7	mA		8049AH 8039AHL
			5	10	mA		8050AH 8040AHL
I_{DD} + I_{CC}	Total Supply Current		30	65	mA		8048AH 8035AHL
			35	70	mA		8049AH 8039AHL
			40	80	mA		8050AH 8040AHL
			30	90	mA		8748AH
			50	110	mA		8749AH
V_{DD}	RAM Standby Voltage	2.2		5.5	V	Standby Mode Reset $\leq V_{IL1}$	8048AH 8035AH
		2.2		5.5	V		8049AH 8039AH
		2.2		5.5	V		8050AH 8040AH

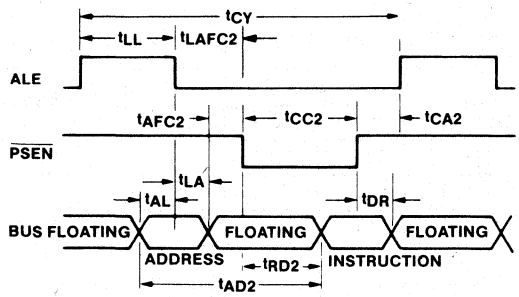
A.C. CHARACTERISTICS: (T_A = 0°C to 70°C; V_{CC} = V_{DD} = 5V ± 10%; V_{SS} = 0V)

Symbol	Parameter	f (t) (Note 3)	11 MHz		Unit	Conditions (Note 1)
			Min	Max		
t	Clock Period	1/xtal freq	90.9	1000	ns	(Note 3)
t_{LL}	ALE Pulse Width	3.5t-170	150		ns	
t_{AL}	Addr Setup to ALE	2t-110	70		ns	(Note 2)
t_{LA}	Addr Hold from ALE	t-40	50		ns	
t_{CC1}	Control Pulse Width ($\overline{RD}$, $\overline{WR}$)	7.5t-200	480		ns	
t_{CC2}	Control Pulse Width ($\overline{PSEN}$)	6t-200	350		ns	
t_{DW}	Data Setup before $\overline{WR}$	6.5t-200	390		ns	
t_{WD}	Data Hold after $\overline{WR}$	t-50	40		ns	
t_{DR}	Data Hold ($\overline{RD}$, $\overline{PSEN}$)	1.5t-30	0	110	ns	
t_{RD1}	$\overline{RD}$ to Data in	6t-170		350	ns	
t_{RD2}	$\overline{PSEN}$ to Data in	4.5t-170		190	ns	
t_{AW}	Addr Setup to $\overline{WR}$	5t-150	300		ns	
t_{AD1}	Addr Setup to Data ($\overline{RD}$)	10.5t-220		730	ns	
t_{AD2}	Addr Setup to Data ($\overline{PSEN}$)	7.5t-200		460	ns	
t_{AFC1}	Addr Float to $\overline{RD}$, $\overline{WR}$	2t-40	140		ns	(Note 2)
t_{AFC2}	Addr Float to $\overline{PSEN}$	.5t-40	10		ns	(Note 2)
t_{LAFC1}	ALE to Control ($\overline{RD}$, $\overline{WR}$)	3t-75	200		ns	
t_{LAFC2}	ALE to Control ($\overline{PSEN}$)	1.5t-75	60		ns	
t_{CA1}	Control to ALE ($\overline{RD}$, $\overline{WR}$, $\overline{PROG}$)	t-40	50		ns	
t_{CA2}	Control to ALE ($\overline{PSEN}$)	4t-40	320		ns	
t_{CP}	Port Control Setup to $\overline{PROG}$	1.5t-80	50		ns	
t_{PC}	Port Control Hold to $\overline{PROG}$	4t-260	100		ns	
t_{PR}	$\overline{PROG}$ to P2 Input Valid	8.5t-120		650	ns	
t_{PF}	Input Data Hold from $\overline{PROG}$	1.5t	0	140	ns	
t_{DP}	Output Data Setup	6t-290	250		ns	
t_{PD}	Output Data Hold	1.5t-90	40		ns	
t_{PP}	$\overline{PROG}$ Pulse Width	10.5t-250	700		ns	
t_{PL}	Port 2 I/O Setup to ALE	4t-200	160		ns	
t_{LP}	Port 2 I/O Hold to ALE	1.5t-120	15		ns	
t_{PV}	Port Output from ALE	4.5t+100		510	ns	
t_{0PRR}	T0 Rep Rate	3t	270		ns	
t_{CY}	Cycle Time	15t	1.36	15.0	μs	

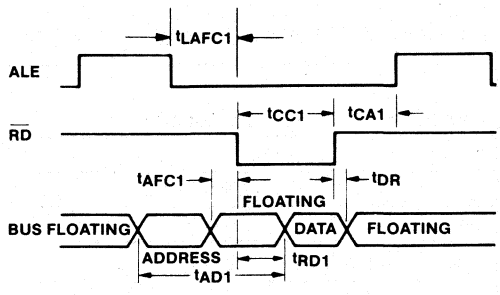
Notes:

1. Control Outputs CL = 80pF
   BUS Outputs CL = 150pF
2. BUS High Impedance
   Load 20pF
3. f(t) assumes 50% duty cycle on X1, X2. Max clock period is for a 1 MHz crystal input.

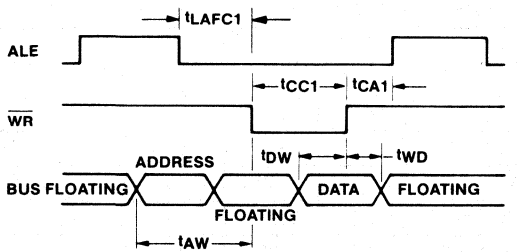
WAVEFORMS



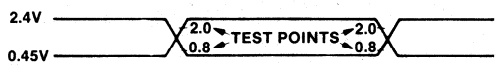
Instruction Fetch From Program Memory



Read From External Data Memory



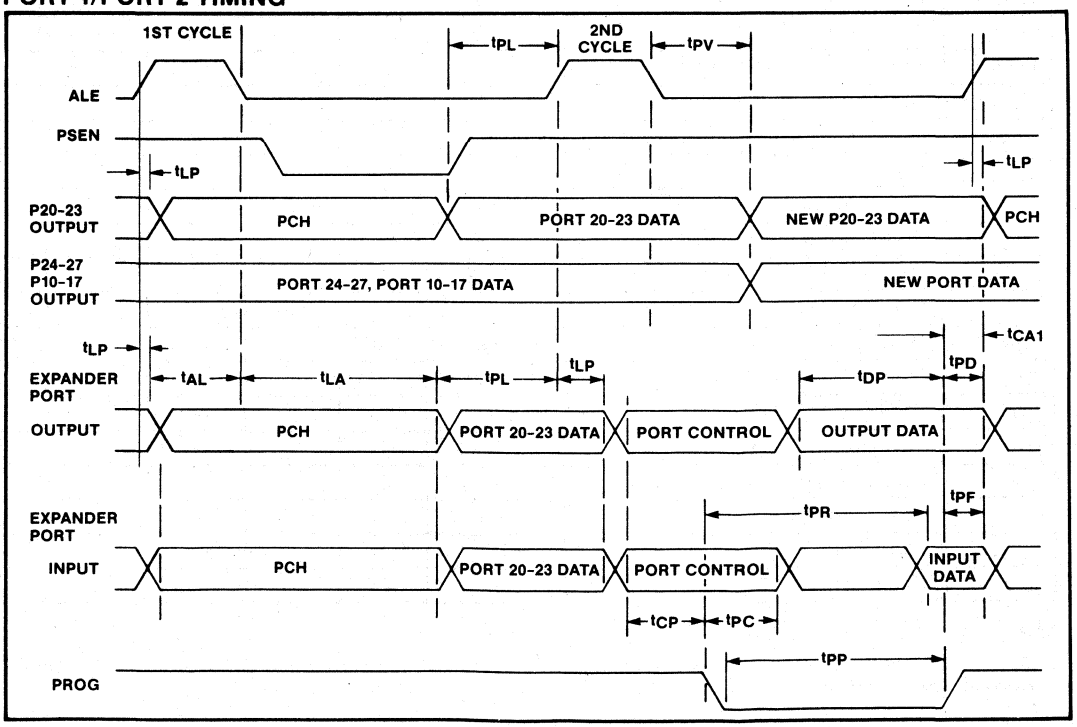
Write To External Data Memory



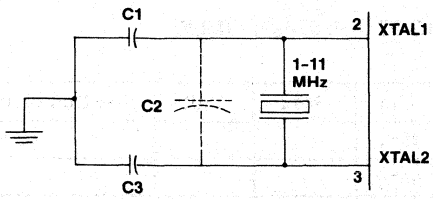
A.C. testing inputs are driven at 2.4V for a logic "1" and 0.45V for a logic "0." Output timing measurements are made at 2.0V for a logic "1" and 0.8V for a logic "0."

Input And Output For A.C. Tests

PORT 1/PORT 2 TIMING



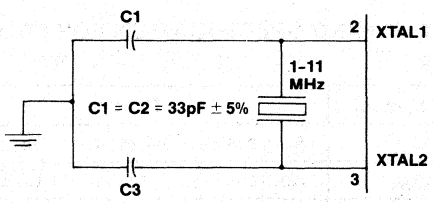
CRYSTAL OSCILLATOR MODE



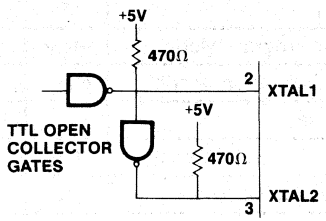
C1 = 5pF ± 1/2pF + (STRAY < 5pF)
C2 = (CRYSTAL + STRAY) < 8pF
C3 = 20pF ± 1pF + (STRAY < 5pF)

Crystal series resistance should be less than 30Ω at 11 MHz; less than 75Ω at 6 MHz; less than 180Ω at 3.6 MHz.

CERAMIC RESONATOR MODE



DRIVING FROM EXTERNAL SOURCE



For XTAL1 and XTAL2 define "high" as voltages above 1.6V and "low" as voltages below 1.6V. The duty cycle requirements for externally driving XTAL1 and XTAL2 using the circuit shown above are as follows: XTAL1 must be high 35–65% of the period and XTAL2 must be high 36–65% of the period. Rise and fall times must be faster than 20 nS.

PROGRAMMING, VERIFYING, AND ERASING THE 8749H (8748H) EPROM

Programming Verification

In brief, the programming process consists of: activating the program mode, applying an address, latching the address, applying data, and applying a programming pulse. Each word is programmed completely before moving on to the next and is followed by a verification step. The following is a list of the pins used for programming and a description of their functions:

Pin	Function
XTAL1	Clock Input (3 to 4.0 MHz)
Reset	Initialization and Address Latching
Test 0	Selection of Program or Verify Mode
EA	Activation of Program/Verify Modes
BUS	Address and Data Input Data Output During Verify
P20-P22	Address Input
V_{DD}	Programming Power Supply
PROG	Program Pulse Input

WARNING:
An attempt to program a missocketed 8749H (8748H) will result in severe damage to the part. An indication of a properly socketed part is the appearance of the ALE clock output. The lack of this clock may be used to disable the programmer.

The Program/Verify sequence is:

1. V_{DD} = 5V, Clock applied or internal oscillator operating. RESET = 0V, TEST 0 = 5V, EA = 5V, BUS and PROG floating. P10 and P11 must be tied to ground.
2. Insert 8749H (8748H) in programming socket.
3. TEST 0 = 0V (select program mode)
4. EA = 18V (activate program mode)
5. Address applied to BUS and P20-22
6. RESET = 5V (latch address)
7. Data applied to BUS
8. V_{DD} = 21V (programming power)
9. PROG = V_{CC} or float followed by one 50ms pulse to 18V
10. V_{DD} = 5V
11. TEST 0 = 5V (verify mode)
12. Read and verify data on BUS
13. TEST 0 = 0V
14. RESET = 0V and repeat from step 5
15. Programmer should be at conditions of step 1 when 8749H (8748H) is removed from socket.

A.C. TIMING SPECIFICATION FOR PROGRAMMING 8748H/8749H ONLY:

(T_A = 25°C ± 5°C; V_{CC} = 5V ± 5%; V_{DD} = 21 ± .5V)

Symbol	Parameter	Min	Max	Unit	Test Conditions
t_{AW}	Address Setup Time to $\overline{RESET}$↑*	$4t_{CY}$			
t_{WA}	Address Hold Time After $\overline{RESET}$↑*	$4t_{CY}$			
t_{DW}	Data in Setup Time to PROG↑	$4t_{CY}$			
t_{WD}	Data in Hold Time After PROG↓	$4t_{CY}$			
t_{PH}	$\overline{RESET}$ Hold Time to Verify	$4t_{CY}$			
t_{VDDW}	V_{DD} Hold Time Before PROG↑	0	1.0	ms	
t_{VDDH}	V_{DD} Hold Time After PROG↓	0	1.0	ms	
t_{PW}	Program Pulse Width	50	60	ms	
t_{TW}	Test 0 Setup Time for Program Mode	$4t_{CY}$			
t_{WT}	Test 0 Hold Time After Program Mode	$4t_{CY}$			
t_{DO}	Test 0 to Data Out Delay		$4t_{CY}$		
t_{WW}	$\overline{RESET}$ Pulse Width to Latch Address*	$4t_{CY}$			
t_r, t_f	V_{DD} and PROG Rise and Fall Times	0.5	100	μs	
t_{CY}	CPU Operation Cycle Time	3.75	5	μs	
t_{RE}	$\overline{RESET}$ Setup Time before EA↑*	$4t_{CY}$			

NOTE: If Test 0 is high, t_{DO} can be triggered by $\overline{RESET}$↑. *Valid for 8048AH/8049AH/8050AH also.

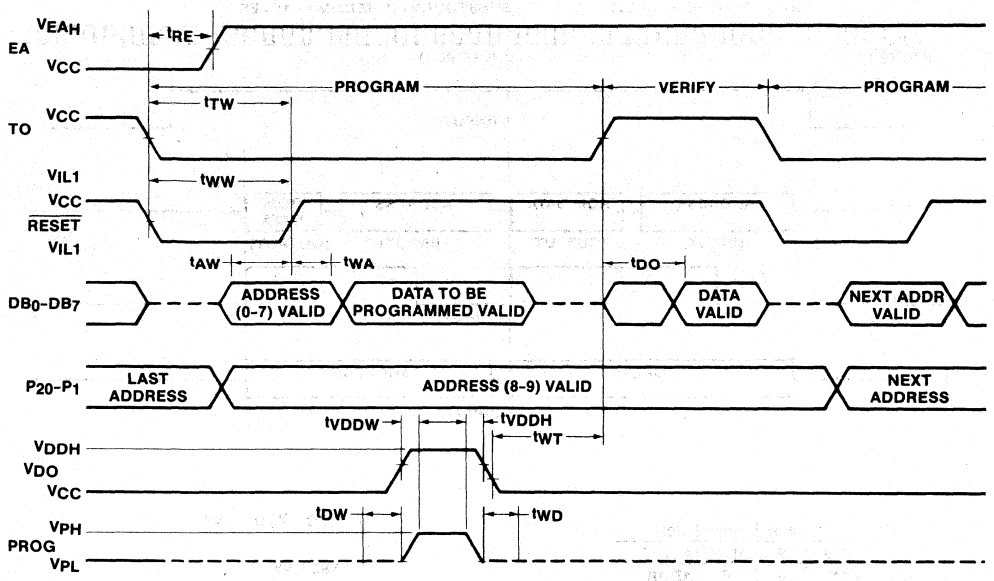
D.C. TIMING SPECIFICATION FOR PROGRAMMING 8748H/8749H ONLY:

(T_A = 25°C ± 5°C; V_{CC} = 5V ± 5%; V_{DD} = 21 ± .5V)

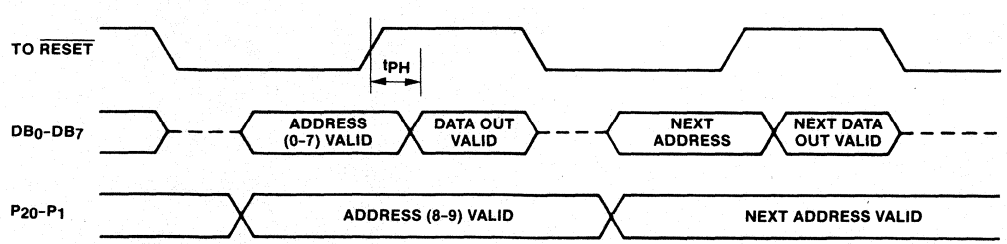
Symbol	Parameter	Min	Max	Unit	Test Conditions
V_{DDH}	V_{DD} Program Voltage High Level	20.5	21.5	V	
V_{DDL}	V_{DD} Voltage Low Level	4.75	5.25	V	
V_{PH}	PROG Program Voltage High Level	17.5	18.5	V	
V_{PL}	PROG Voltage Low Level	4.0	V_{CC}	V	
V_{EAH}	EA Program or Verify Voltage High Level	17.5	18.5	V	
I_{DD}	V_{DD} High Voltage Supply Current		20.0	mA	
I_{PROG}	PROG High Voltage Supply Current		1.0	mA	
I_{EA}	EA High Voltage Supply Current		1.0	mA	

WAVEFORMS

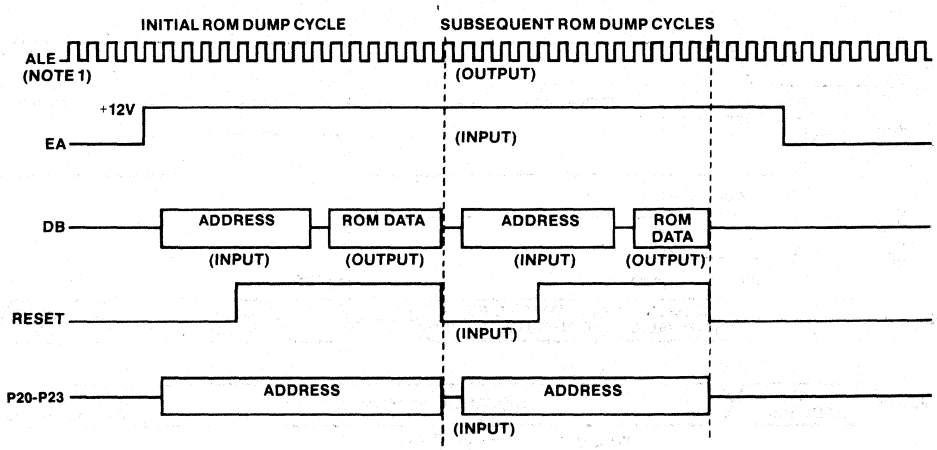
COMBINATION PROGRAM/VERIFY MODE (EPROM'S ONLY)



VERIFY MODE (ROM/EPROM)



SUGGESTED ROM VERIFICATION ALGORITHM FOR H-MOS DEVICE ONLY



	48H	49H	50H
A10	0	ADDR	ADDR
A11	0	0	ADDR

$V_{CC} = V_{DD} = +5V$

$V_{SS} = 0V$

NOTE: ALE is function of X1, X2 inputs.

80C48/80C35/80C49/80C39/80C50/80C40
CHMOS SINGLE-COMPONENT 8-BIT MICROCOMPUTER

- **80C48/80C49/80C50 Low Power Mask Programmable ROM**
- **80C35/80C39/80C40 Low Power, CPU only**

- **Pin-to-pin Compatible with Intel's 8048AH/8035AHL/8049AH/ 8039AHL/8050AH/8040AH**
- **1.36 μsec Instruction Cycle. All Instructions 1 or 2 Cycles**
- **Ability to Maintain Operation during AC Power Line Interruptions**
- **Exit Idle Mode with an External or Internal Interrupt Signal**
- **Battery Operation**
- **3 Power Consumption Selections**
  —Normal Operation: 12 mA @ 11 MHz @ 5V
  —Idle Mode: 5 mA @ 11 MHz @ 5V
  —Power Down: 2 μA @ 2.0V
- **11 MHz, TTL Compatible Operation:**
  V_{CC} = 5V ± 10%
  CMOS Compatible Operation;
  V_{CC} = 5V ± 20%

Intel's 80C48/80C35/80C49/80C39/80C50/80C40 are low power, CHMOS versions of the popular MCS®-48 HMOS family members. CHMOS is a technology built on HMOS II and features high resistivity P substrate, diffused N well, and scaled N and P channel devices. The 80C48/80C35/80C49/80C39/80C50/80C40 have been designed to provide low power consumption and high performance.

The 80C48/80C49/80C50 contains a 1K × 8/2K × 8/4K × 8 program memory, a 64 × 8/128 × 8/256 × 8 RAM data memory, 27 I/O lines, and an 8-bit timer/counter in addition to an on-board oscillator and clock circuits. For systems that require extra capability, the 80C48/80C49/80C50 can be expanded using CMOS external memories and MCS®-80 and MCS®-85 peripherals. The 80C35/80C39/80C40 is the equivalent of the 80C48/80C49/80C50 without program memory on-board.

The CHMOS design of the 80C48/80C49/80C50 opens new application areas that require battery operation, low power standby, wide voltage range, and the ability to maintain operation during AC power line interruptions. These applications include portable and hand-held instruments, telecommunications, consumer, and automotive.

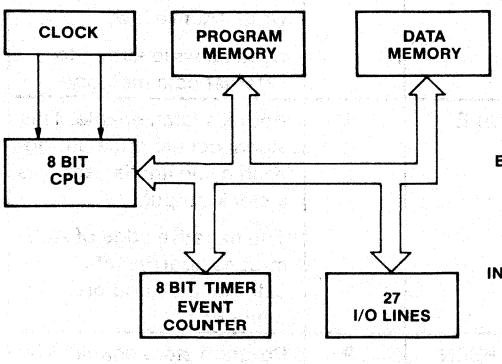


Figure 1. Block Diagram

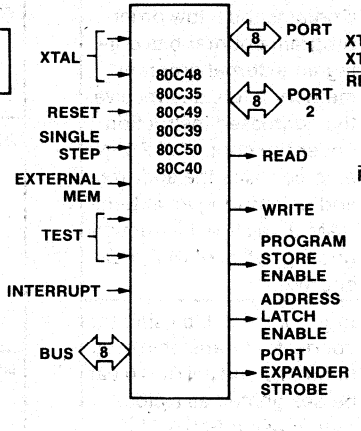


Figure 2. Logic Symbol

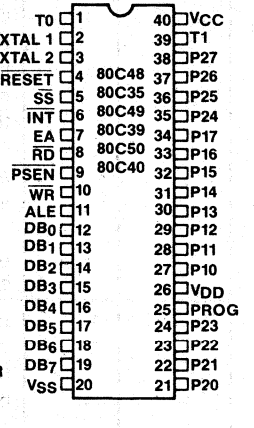


Figure 3. Pin Configuration

January 1983
Order Number 210936-001

Table 1. Pin Description

Symbol	Pin No.	Function
V_{SS}	20	Circuit GND potential
V_{DD}	26	Low Power standby pin
V_{CC}	40	Main power supply; +5V during operation.
PROG	25	Output strobe for 82C43 I/O expander.
P10–P17 Port 1	27–34	8-bit quasi-bidirectional port.
P20–P23	21–24	8-bit quasi-bidirectional port.
P24–P27 Port 2	35–38	P20–P23 contain the four high order program counter bits during an external program memory fetch and serve as a 4-bit I/O expander bus for 8243.
DB0–DB7 BUS	12–19	True bidirectional port which can be written or read synchronously using the RD, WR strobes. The port can also be statically latched. Contains the 8 low order program counter bits during an external program memory fetch, and receives the addressed instruction under the control of PSEN. Also contains the address and data during an external RAM data store instruction, under control of ALE, RD, and WR.
T0	1	Input pin testable using the conditional transfer instructions JT0 and JNTo. T0 can be designated as a clock output using ENT0 CLK instruction.
T1	39	Input pin testable using the JT1, and JNT1 instructions.

Symbol	Pin No.	Function
		Can be designated the timer/counter input using the STRT CNT instruction.
INT	6	Interrupt input. Initiates an interrupt if interrupt is enabled. Interrupt is disabled after a reset. Also testable with conditional jump instruction. (Active low) Interrupt must remain low for at least 3 machine cycles for proper operation.
RD	8	Output strobe activated during a BUS read. Can be used to enable data onto toe bus from an external device. Used as a read strobe to external data memory. (Active low)
RESET	4	Input which is used to initialize the processor. (Active low) (Non TTL V_{IH})
WR	10	Output strobe during a bus write. (Active low) Used as write strobe to external data memory.
ALE	11	Address latch enable. This signal occurs once during each cycle and is useful as a clock output. The negative edge of ALE strobes address into external data and program memory.
PSEN	9	Program store enable. This output occurs only during a fetch to external program memory. (Active low)
SS	5	Single step input can be used in conjunction with

Table 1. Pin Description (Continued)

Symbol	Pin No.	Function
SS (Con't)		ALE to "single step" the processor through each instruction (Active low)
EA	7	External access input which forces all program memory fetches to reference external memory. Useful for emulation and debug, and essential for testing
		and program verification. (Active high)
XTAL1	2	One side of crystal input for internal oscillator. Also input for external source. (Non TTL V_{IH})
XTAL2	3	Other side of crystal input.

IDLE MODE DESCRIPTION

The 80C48/80C49/80C50, when placed into Idle mode, keeps the oscillator, the internal timer and the external interrupt and counter pins functioning and maintains the internal register and RAM status.

To place the 80C48/80C49/80C50 in Idle mode, a command instruction (op code 01H) is executed. To terminate Idle mode, a reset must be performed or interrupts must be enabled and an interrupt signal generated. There are two interrupt sources that can restore normal operation. One is an external signal applied to the interrupt pin. The other is from the overflow of the timer/counter. When either interrupt is invoked, the CPU is taken out of Idle mode and vectors to the interrupt's service routine address. Along with the Idle mode, the standard MCS®-48 power-down mode is still maintained.

Table 2. Instruction Set

Accumulator

Mnemonic	Description	Bytes	Cycles
ADD A, R	Add register to A	1	1
ADD A, @R	Add data memory to A	1	1
ADD A, # data	Add immediate to A	2	2
ADDC A, R	Add register with carry	1	1
ADDC A, @R	Add data memory with carry	1	1
ADDC A, # data	Add immediate with carry	2	2
ANL A, R	And register to A	1	1
ANL A, @R	And data memory to A	1	1
ANL A, # data	And immediate to A	2	2
ORL A, R	Or register to A	1	1
ORL A @R	Or data memory to A	1	1
ORL A, # data	Or immediate to A	2	2
XRL A, R	Exclusive or register to A	1	1
XRL A, @R	Exclusive or data memory to A	1	1
XRL, A, # data	Exclusive or immediate to A	2	2
INC A	Increment A	1	1
DEC A	Decrement A	1	1
CLR A	Clear A	1	1
CPL A	Complement A	1	1
DA A	Decimal adjust A	1	1
SWAP A	Swap nibbles of A	1	1
RL A	Rotate A left	1	1
RLC A	Rotate A left through carry	1	1
RR A	Rotate A right	1	1
RRC A	Rotate A right through carry	1	1

Input/Output

Mnemonic	Description	Bytes	Cycles
IN A, P	Input port to A	1	2
OUTL P, A	Output A to port	1	2
ANL P, # data	And immediate to port	2	2
ORL P, # data	Or immediate to port	2	2
INS A, BUS	Input BUS to A	1	2
OUTL BUS, A	Output A to BUS	1	2
ANL BUS, # data	And immediate to BUS	2	2
ORL BUS, # data	Or immediate to BUS	2	2
MOVD A, P	Input expander port to A	1	2
MOVD P, A	Output A to expander port	1	2
ANLD P, A	And A to expander port	1	2
ORLD P, A	Or A to expander port	1	2

Registers

Mnemonic	Description	Bytes	Cycles
INC R	Increment register	1	1
INC @R	Increment data memory	1	1
DEC R	Decrement register	1	1

Branch

Mnemonic	Description	Bytes	Cycles
JMP addr	Jump unconditional	2	2
JMPP @A	Jump indirect	1	2
DJNZ R, addr	Decrement register and skip	2	2
JC addr	Jump on carry = 1	2	2
JNC addr	Jump on carry = 0	2	2
JZ addr	Jump on A zero	2	2
JNZ addr	Jump on A not zero	2	2
JT0 addr	Jump on T0 = 1	2	2
JNT0 addr	Jump on T0 = 0	2	2
JT1 addr	Jump on T1 = 1	2	2
JNT1 addr	Jump on T1 = 0	2	2
JF0 addr	Jump on F0 = 1	2	2
JF1 addr	Jump on F1 = 1	2	2
JTF addr	Jump on timer flag	2	2
JNI addr	Jump on INT = 0	2	2
JBb addr	Jump on accumulator bit	2	2

Subroutine

Mnemonic	Description	Bytes	Cycles
CALL addr	Jump to subroutine	2	2
RET	Return	1	2
RETR	Return and restore status	1	2

Flags

Mnemonic	Description	Bytes	Cycles
CLR C	Clear carry	1	1
CPL C	Complement carry	1	1
CLR F0	Clear flag 0	1	1
CPL F0	Complement flag 0	1	1
CLR F1	Clear flag 1	1	1
CPL F1	Complement flag 1	1	1

Table 2. Instruction Set (Continued)

Data Moves

Mnemonic	Description	Bytes	Cycles
MOV A, R	Move register to A	1	1
MOV A, @R	Move data memory to A	1	1
MOV A, # data	Move immediate to A	2	2
MOV R, A	Move A to register	1	1
MOV @R, A	Move A to data memory	1	1
MOV R, # data	Move immediate to register	2	2
MOV @R, # data	Move immediate to data memory	2	2
MOV A, PSW	Move PSW to A	1	1
MOV PSW, A	Move A to PSW	1	1
XCH A, R	Exchange A and register	1	1
XCH A, @R	Exchange A and data memory	1	1
XCHD A, @R	Exchange nibble of A and register	1	1
MOVX A, @R	Move external data memory to A	1	2
MOVX @R, A	Move A to external data memory	1	2
MOVP A, @A	Move to A from current page	1	2
MOVP3 A, @	Move to A from page 3	1	2

Timer/Counter

Mnemonic	Description	Bytes	Cycles
MOV A, T	Read timer/counter	1	1
MOV T, A	Load timer/counter	1	1
STRT T	Start timer	1	1
STRT CNT	Start timer	1	1
STOP TCNT	Stop timer/counter	1	1
EN TCNTI	Enable timer/counter interrupt	1	1
DIS TCNTI	Disable timer/counter interrupt	1	1

Control

Mnemonic	Description	Bytes	Cycles
EN I	Enable external interrupt	1	1
DIS I	Disable external interrupt	1	1
SEL RB0	Select register bank 0	1	1
SEL RB1	Select register bank 1	1	1
SEL MB0	Select memory bank 0	1	1
SEL MB1	Select memory bank 1	1	1
ENT0 CLK	Enable clock output on T0	1	1

Mnemonic	Description	Bytes	Cycles
NOP	No operation	1	1
IDL	Select Idle Operation	1	1

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias . . . 0°C to 70°C
Storage Temperature . . . . . -65°C to +150°C
Voltage On Any Pin With Respect to Ground . . . . . -0.5V to V_{CC}+1V
Maximum Voltage On Any Pin With Respect to Ground 7V
Power Dissipation . . . . . 1.5 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of device at these or any other conditions above those indicated in the operational sections of this specification is not implied.*

D.C. CHARACTERISTICS: (T_A = 0°C to 70°C; V_{CC} = V_{DD} = 5V ± 20%; $|V_{CC} - V_{DD}| \leq 1.5V$; V_{SS} = 0V)

Symbol	Parameter	Limits			Unit	Test Conditions
		Min	Typ	Max		
V_{IL}	Input Low Voltage (All Except X1, $\overline{RESET}$)	-.5		.18 V_{CC}	V	
V_{IL1}	Input Low Voltage X1, $\overline{RESET}$	-5		.13 V_{CC}	V	
V_{IH}	Input High Voltage (All Except XTAL1, $\overline{RESET}$)	.4 V_{CC}		V_{CC}	V	
V_{IH1}	Input High Voltage (X1, $\overline{RESET}$)	.7 V_{CC}		V_{CC}	V	
V_{OL}	Output Low Voltage (BUS)			.45	V	I_{OL} = 2.0 mA
V_{OL1}	Output Low Voltage ($\overline{RD}$, $\overline{WR}$, $\overline{PSEN}$, ALE)			.45	V	I_{OL} = 1.8 mA
V_{OL2}	Output Low Voltage (PROG)			.45	V	I_{OL} = 1.0 mA
V_{OL3}	Output Low Voltage (All Other Outputs)			.45	V	I_{OL} = 1.6 mA
V_{OH}	Output High Voltage (BUS)	.75 V_{CC}			V	I_{OH} = -400 μA
V_{OH1}	Output High Voltage ($\overline{RD}$, $\overline{WR}$, $\overline{PSEN}$, ALE)	.75 V_{CC}			V	I_{OH} = -100 μA
V_{OH2}	Output High Voltage (All Other Outputs)	.75 V_{CC}			V	I_{OH} = -40 μA
I_{L1}	Input Leakage Current (T1, $\overline{INT}$, EA)			±5	μA	$V_{SS} \leq V_{IN} \leq V_{CC}$
I_{LI1}	Input Leakage Current (P10-P17, P20-P27, $\overline{SS}$)			-500	μA	$V_{SS} \leq V_{IN} \leq V_{CC}$
I_{LO}	Output Leakage Current (BUS, TO) (High Impedance State)			±5	μA	$V_{SS} \leq V_{IN} \leq V_{CC}$
I_{LR}	Input Leakage Current ($\overline{RESET}$)	-20		-300	μA	$V_{SS} \leq V_{IN} \leq V_{CC}$
I_{PD}	Power Down Standby Current			2	μA	V_{DD} = 2.0V $\overline{RESET} \leq V_{IL}$

I_{CC} Active Current (mA)

V_{CC}	4V	5V	6V
1 MHz	1.5	2.3	3
6 MHz	5	6.8	8.5
11 MHz	9	12	15

I_{CC} Idle Current (mA)

V_{CC}	4V	5V	6V
1 MHz	.7	1	1.2
6 MHz	2	3	4
11 MHz	3.5	4.8	6

Absolute Maximum Unloaded Current

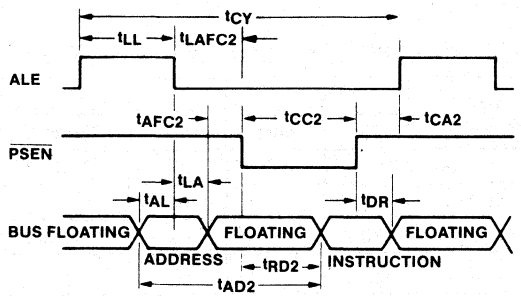
A.C. CHARACTERISTICS: (T_A = 0° C to 70° C; V_{CC} = V_{DD} = 5V ± 20%; $|V_{CC} - V_{DD}| \leq 1.5V$; V_{SS} = 0V)

Symbol	Parameter	f (t) (Note 3)	11 MHz		Unit	Conditions (Note 1)
			Min	Max		
t	Clock Period	1/xtal freq	90.9	1000	ns	(Note 3)
t_{LL}	ALE Pulse Width	3.5t-170	150		ns	
t_{AL}	Addr Setup to ALE	2t-110	70		ns	(Note 2)
t_{LA}	Addr Hold from ALE	t-40	50		ns	
t_{CC1}	Control Pulse Width ($\overline{RD}$, $\overline{WR}$)	7.5t-200	480		ns	
t_{CC2}	Control Pulse Width ($\overline{PSEN}$)	6t-200	350		ns	
t_{DW}	Data Setup before $\overline{WR}$	6.5t-200	390		ns	
t_{WD}	Data Hold after $\overline{WR}$	t-50	40		ns	
t_{DR}	Data Hold ($\overline{RD}$, $\overline{PSEN}$)	1.5t-30	0	110	ns	
t_{RD1}	$\overline{RD}$ to Data in	6t-170		350	ns	
t_{RD2}	$\overline{PSEN}$ to Data in	4.5t-170		190	ns	
t_{AW}	Addr Setup to $\overline{WR}$	5t-150	300		ns	
t_{AD1}	Addr Setup to Data ($\overline{RD}$)	10.5t-220		730	ns	
t_{AD2}	Addr Setup to Data ($\overline{PSEN}$)	7.5t-220		460	ns	
t_{AFC1}	Addr Float to $\overline{RD}$, $\overline{WR}$	2t-40	140		ns	(Note 2)
t_{AFC2}	Addr Float to $\overline{PSEN}$	.5t-40	10		ns	(Note 2)
t_{LAFC1}	ALE to Control ($\overline{RD}$, $\overline{WR}$)	3t-75	200		ns	
t_{LAFC2}	ALE to Control ($\overline{PSEN}$)	1.5t-75	60		ns	
t_{CA1}	Control to ALE ($\overline{RD}$, $\overline{WR}$, $\overline{PROG}$)	t-40	50		ns	
t_{CA2}	Control to ALE ($\overline{PSEN}$)	4t-40	320		ns	
t_{CP}	Port Control Setup to $\overline{PROG}$	1.5t-80	50		ns	
t_{PC}	Port Control Hold to $\overline{PROG}$	4t-260	100		ns	
t_{PR}	$\overline{PROG}$ to P2 Input Valid	8.5t-120		650	ns	
t_{PF}	Input Data Hold from $\overline{PROG}$	1.5t	0	140	ns	
t_{DP}	Output Data Setup	6t-290	250		ns	
t_{PD}	Output Data Hold	1.5t-90	40		ns	
t_{PP}	$\overline{PROG}$ Pulse Width	10.5t-250	700		ns	
t_{PL}	Port 2 I/O Setup to ALE	4t-200	160		ns	
t_{LP}	Port 2 I/O Hold to ALE	1.5t-120	15		ns	
t_{PV}	Port Output from ALE	4.5t+100		510	ns	
t_{0PRR}	T0 Rep Rate	3t	270		ns	
t_{CY}	Cycle Time	15t	1.36	15.0	μs	

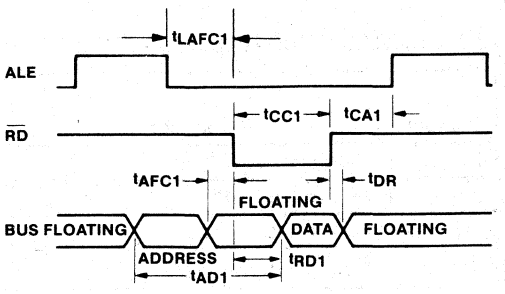
Notes:

1. Control Outputs CL = 80pF
   BUS Outputs CL = 150pF
2. BUS High Impedance Load 20pF
3. f(t) assumes 50% duty cycle on X1, X2. Max clock period is for a 1 MHz crystal input.

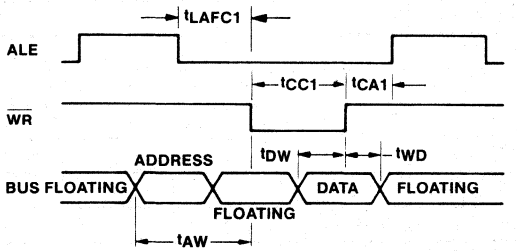
WAVEFORMS



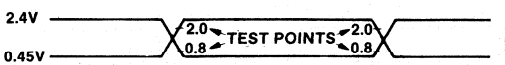
Instruction Fetch From Program Memory



Read From External Data Memory



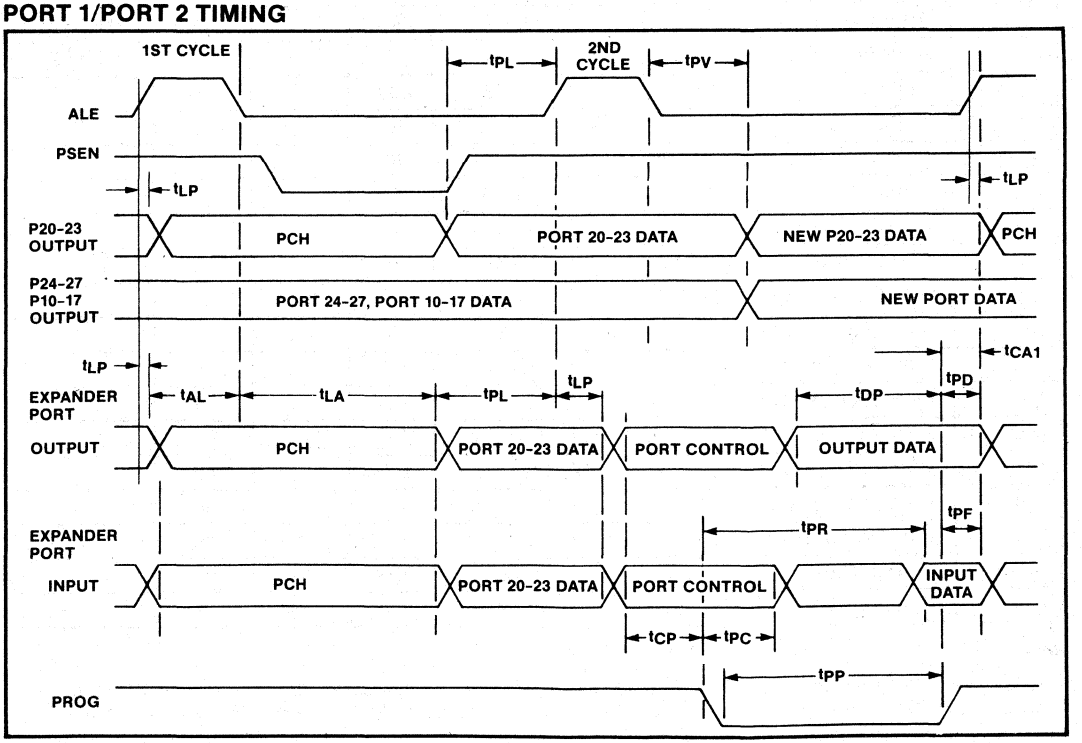
Write To External Data Memory



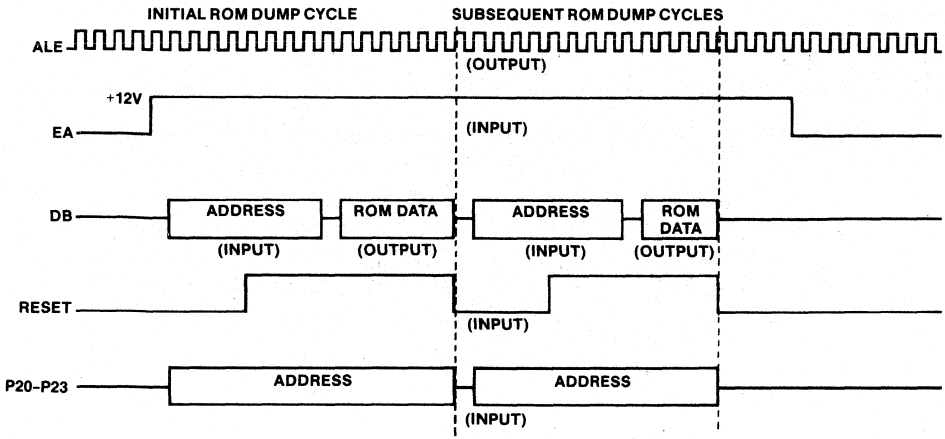
A.C. testing inputs are driven at 2.4V for a logic "1" and 0.45V for a logic "0." Output timing measurements are made at 2.0V for a logic "1" and 0.8V for a logic "0."

Input And Output For A.C. Tests

PORT 1/PORT 2 TIMING



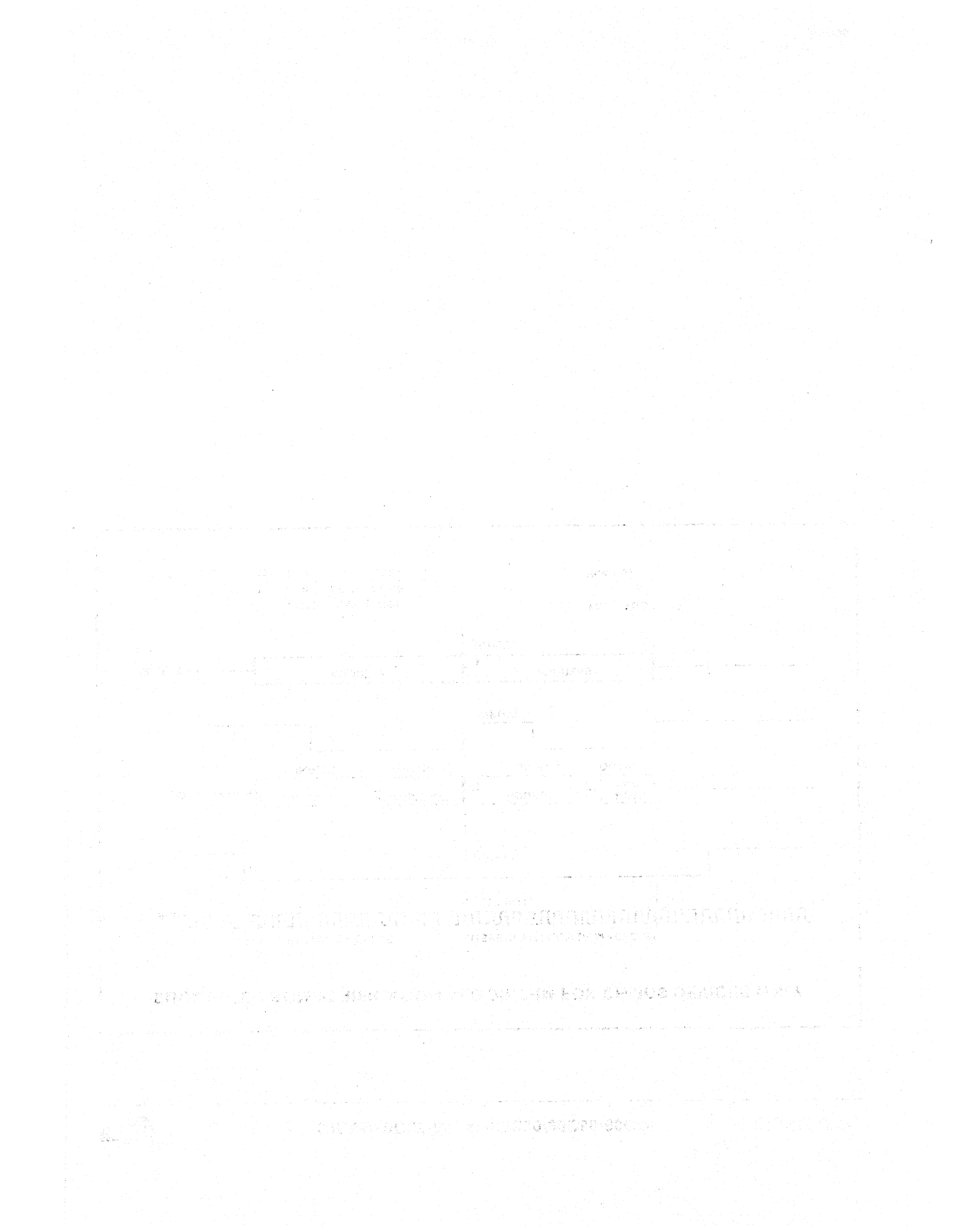
SUGGESTED ROM VERIFICATION ALGORITHM FOR CHMOS DEVICES ONLY



	C48	C49	C50
A10	0	ADDR	ADDR
A11	0	0	ADDR

$V_{CC} = V_{DD} = +5V$

$V_{SS} = 0V$



8044/8344/8744
RUPI™-44: Remote Universal Peripheral Interface

- **8044 - CPU/SIU with Factory Mask Programmable ROM**
- **8344 - AN 8044 Used with External Program Memory**
- **8744 - AN 8044 with User Programmable/Erasable EPROM**

- **8-Bit CPU Plus Independent HDLC/SDLC Protocol Controller**
- **Serial Interface Unit (SIU)**
  - **High-Performance, High-Level HDLC/SDLC Serial Communications Controller**
  - **2.4 Mbps in Clocked Mode**
  - **375 Kbps in Self Clocked Mode Using On-Chip Phase Lock Loop**
  - **SDLC Loop Mode**
- **Fully Compatible with 8051 CPU**
- **Boolean Processor**
- **4K x 8 ROM/EPROM, 192 x 8 RAM**
- **External Memory Expandable to 128 K Bytes**
- **32 I/O Lines, Configurable as a Local Bus**
- **Two 16-Bit Timer/Event Counters**

The RUPI-44 is a Remote Universal Peripheral Interface designed to serve as a programmable remote peripheral or peripheral sub-system controller. It contains its own CPU, program memory, data memory, parallel I/O, timers/counters, and an intelligent serial interface that can serve as a primary or secondary station.

The CPU and all peripheral functions except for the data memory and the serial port are based on the 8051 and maintain complete compatibility with it. The data memory is larger (192x8) to allow convenient serial data buffering. The serial port implements a subset of HDLC/SDLC in the hardware. The Serial Interface Unit (SIU) operating concurrently with the CPU, offers a high level of intelligence and performance for HDLC/SDLC based communication.

The RUPI-44 family consists of the 8044, 8744, and 8344. All three devices are identical except in respect of on-chip program memory. The 8044 contains 4K bytes of mask-programmable ROM. User-programmable EPROM replaces ROM in the 8744. The 8344 addresses all program memory externally.

The RUPI-44 devices are fabricated with Intel's reliable +5 volt, silicon-gate HMOS technology and packaged in a 40-pin DIP.

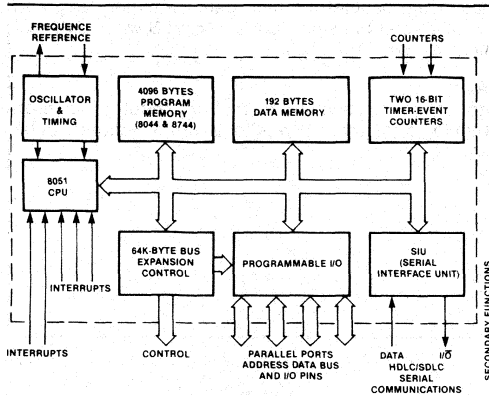


Figure 1. Block Diagram

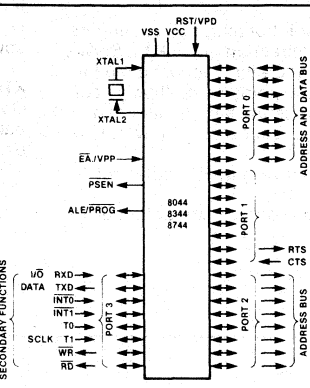


Figure 2. Logic Symbol

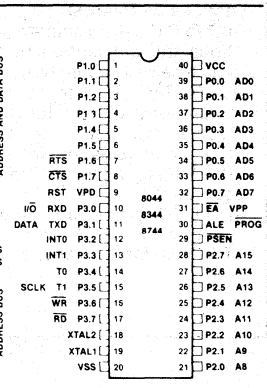


Figure 3. Pin Configuration

OCTOBER 1982
Order Number: 210735-001

Table 1. RUPI™ -44 Family Pin Description

VSS
Circuit ground potential.

VCC
+5V power supply during operation and program verification.

PORT 0
Port 0 is an 8-bit open drain bidirectional I/O port. It is also the multiplexed low-order address and data bus when using external memory. It is used for data output during program verification. Port 0 can sink/source eight LS TTL loads.

PORT 1
Port 1 is an 8-bit quasi-bidirectional I/O port. It is used for the low-order address byte during program verification. Port 1 can sink/source LS TTL loads.

In non-loop mode two of the I/O lines serve alternate functions:
—$\overline{RTS}$ (P1.6). Request-to-Send output. A low indicates that the RUPI-44 is ready to transmit.
—$\overline{CTS}$ (P1.7) Clear-to-Send input. A low indicates that a receiving station is ready to receive.

PORT 2
Port 2 is an 8-bit quasi-bidirectional I/O port. It also emits the high-order address byte when accessing external memory. It is used for the high-order address and the control signals during program verification. Port 2 can sink/source LS TTL loads.

PORT 3
Port 3 is an 8-bit quasi-bidirectional I/O port. It also contains the interrupt, timer, serial port and $\overline{RD}$ and $\overline{WR}$ pins that are used by various options. The output latch corresponding to a secondary function must be programmed to a one (1) for that function to operate. Port 3 can sink/source LS TTL loads.

In addition to I/O, some of the pins also serve alternate functions as follows:
—I/O RxD (P3.0). In point-to-point or multipoint configurations, this pin controls the direction of pin P3.1. Serves as Receive Data input in loop and diagnostic modes.
—DATA TxD (P3.1) In point-to-point or multipoint configurations, this pin functions as data input/output. In loop mode, it serves as transmit pin. A '0' written to this pin enables test mode.
—$\overline{INT0}$ (P3.2). Interrupt 0 input or gate control input for counter 0.
—$\overline{INT1}$ (P3.3). Interrupt 1 input or gate control input for counter 1.
—TO(P3.4). Input to counter 0.
—SCLK T1 (P3.5). In addition to I/O, this pin provides input to counter 1 or serves as SCLK (serial clock) input.
—$\overline{WR}$ (P3.6). The write control signal latches the data byte from Port 0 into the External Data Memory.
—$\overline{RD}$ (P3.7). The read control signal enables External Data Memory to Port 0.

RST/VPD
A high level on this pin resets the RUPI-44. A small internal pulldown resistor permits power-on reset using only a capacitor connected to VCC. If VPD is held within its spec while VCC drops below spec, VPD will provide standby power to the RAM. When VPD is low, the RAM's current is drawn from VCC.

ALE/$\overline{PROG}$
Provides Address Latch Enable output used for latching the address into external memory during normal operation. It is activitated every six oscillator periods except during an external data memory access. It also receives the program pulse input for programming the EPROM version.

$\overline{PSEN}$
The Program Store Enable output is a control signal that enables the external Program Memory to the bus during external fetch operations. It is activated every six oscillator periods, except during external data memory accesses. Remains high during internal program execution.

$\overline{EA}$/VPP
When held at a TTL high level, the RUPI-44 executes instructions from the internal ROM when the PC is less than 4096. When held at a TTL low level, the RUPI-44 fetches all instructions from external Program Memory. The pin also receives the 21V EPROM programming supply voltage on the 8744.

XTAL 1
Input to the oscillator's high gain amplifier. Required when a crystal is used. Connect to VSS when external source is used on XTAL 2.

XTAL 2
Output from the oscillator's amplifier. Input to the internal timing circuitry. A crystal or external source can be used.

Functional Description

General

The RUPI-44 integrates the powerful 8051 microcontroller with an intelligent Serial Interface Unit to provide a single-chip solution which will efficiently implement a distributed processing or distributed control system. The microcontroller is a self-sufficient unit containing ROM, RAM, ALU, and its own peripherals. The RUPI's architecture and instruction set are identical to the 8051's. The RUPI replaces the 8051's serial interface with an intelligent SDLC/HDLC Serial Interface Unit (SIU). 64 more bytes of RAM have been added to the 8051 RAM array. The SIU can communicate at bit rates up to 2.4M bps, clocked, or up to 375K bps, self clocked, using the on-chip digital phase locked loop. The SIU works concurrently with the Microcontroller so that there is no throughput loss in either unit. Since the SIU possesses its own intelligence, the CPU is off-loaded from many of the communications tasks, thus dedicating more of its computing power to controlling local peripherals or some external process.

The Microcontroller

The microcontroller is a stand-alone high-performance single-chip computer intended for use in sophisticated real-time application such as instrumentation, industrial control, and intelligent computer peripherals.

The major features of the microcontroller are:

- 8-bit CPU
- on-chip oscillator
- 4K bytes of ROM
- 192 bytes of RAM
- 32 I/O lines
- 64K address space for external Data Memory
- 64K address space for external Program Memory
- two fully programmable 16-bit timer/counters
- a five-source interrupt structure with two priority levels
- bit addressability for Boolean processing
- 1 μsec instruction cycle time for 60% of the instructions 2 μsec instruction cycle time for 40% of the instructions
- 4 μsec cycle time for 8 by 8 bit unsigned Multiply/Divide

Parallel I/O

The RUPI has 32 general-purpose I/O lines which are arranged into four groups of eight lines. Each group is called a port. Hence there are four ports; Port 0, Port 1, Port 2, and Port 3. Up to five lines from 1 and Port 2 are dedicated to supporting the serial channel when the SIU is invoked. Due to the nature of the serial port, two of Port 3's I/O lines (P3.0 and P3.1) do not have latched outputs. This is true whether or not the serial channel is used.

Port 0 and Port 2 also have an alternate dedicated function. When placed in the external access mode, Port 0 and Port 2 become the means by which the RUPI communicates with external program memory. Port 0 and Port 2 are also the means by which the RUPI communicates with external data memory. Peripherals can be memory mapped into the address space and controlled by the RUPI.

Timer/Counters

The RUPI-44 contains two 16-bit counters which can be used for measuring time intervals, measuring pulse widths, counting events, generating precise periodic interrupt requests, and clocking the serial communications. Internally the Timers are clocked at 1/12 of the crystal frequency, which is the instruction cycle time. Externally the counters can run up to 500 KHz.

Interrupt System

External events and the real-time driven on-chip peripherals require service by the CPU asynchronous to the execution of any particular section of code. To tie the asynchronous activities of these functions to normal program execution, a sophisticated multiple-source, two priority level, nested interrupt system is provided. Interrupt response latency ranges from 3 μsec to 7 μsec when using a 12 MHz clock.

All five interrupt sources can be mapped into one of the two priority levels. Each interrupt source can be enabled or disabled individually or the entire interrupt system can be enabled or disabled. The five interrupt sources are: Serial Interface Unit, Timer 1, Timer 2, and two external interrupts. The external interrupts can be either level or edge triggered.

Serial Interface Unit (SIU)

The Serial Interface Unit is used for HDLC/SDLC communications. It handles Zero Bit Insertion/Deletion, Flags, automatic address recognition, and a 16-bit cyclic redundancy check. In addition it implements in hardware a subset of the SDLC protocol such that it responds to many SDLC frames without CPU intervention. In certain applications it is advantageous to have the CPU control the reception or transmission of every single frame. For this reason the RUPI-44 has two modes of operation: "AUTO" and "NON-AUTO." It is in the AUTO mode that the RUPI responds to SDLC frames without CPU intervention; whereas, in the NON-AUTO mode the reception or transmission of every single frame will be under CPU control.

There are three control registers and eight parameter registers that are used to operate the serial interface. These registers are shown in Figure 4 and Figure 5. The control registers set the modes of operation and provide status information. The eight parameter registers buffer the station address, receive and transmit control bytes, and point to the on-chip transmit and receive buffers.

Data to be received or transmitted by the SIU must be buffered anywhere within the 192 bytes of on-chip RAM. Transmit and receive buffers are not allowed to "wrap around" in RAM; a "buffer end" is generated after address 191 is reached.

NON-AUTO Mode

In the NON-AUTO mode all communications are under control of the CPU. It is the CPU's task to encode and decode control fields, manage acknowledgements, and adhere to the requirements of the HDLC/SDLC protocols. The RUPI can be used as a primary or a secondary station in this mode.

To receive a frame in the NON-AUTO mode, the CPU must load the Receive Buffer Start register, the Receive Buffer Length register, clear the Receive Buffer Protect bit, and set the Receive Buffer Empty bit. If a valid opening flag is received and the address field matches the byte in the Station Address register or the address field contains a broadcast address, the RUPI loads the control field in the receive control byte register, and loads the I field in the receive buffer. If there is no CRC error, the SIU interrupts the CPU, indicating a frame has just been received. If there is a CRC error, the frame is ignored by the SIU and no interrupt occurs. The Receive Field Length register provides the number of bytes that were received in the information field.

To transmit a frame, the CPU must load the transmit information buffer, the Transmit Buffer Start register, the Transmit Buffer Length register, the Transmit Control Byte, and set the RTS bit. The SIU, unsolicited by an HDLC/SDLC frame, will transmit the entire information frame, and interrupt the CPU, indicating the completion of transmission. For supervisory frames or unnumbered frames, the transmit buffer length would be 0.

AUTO Mode

In the AUTO mode the SIU implements in hardware a subset of the SDLC protocol such that it responds to many SDLC frames without CPU intervention. All AUTO mode responses to the primary station will conform to IBM's SDLC definition. The advantages of the AUTO mode are that less software is required to implement a secondary station, and the hardware generated response to polls is much faster than doing it in software.

To transmit in the AUTO mode the CPU must load the Transmit Information Buffer, Transmit Buffer Start register, Transmit Buffer Length register, and set the Transmit Buffer Full bit. The RUPI automatically responds to a poll by transmitting an information frame with the P/F bit in the control field set. When the RUPI receives a positive acknowledgement from the primary station, it automatically increments the Ns field in the NSNR register and interrupts the CPU. A negative acknowledgement would cause the RUPI to retransmit the frame.

To receive in the AUTO mode, the CPU loads the Receive Buffer Start register, the Receive Buffer Length register, clears the Receive Buffer Protect bit, and sets the Receive Buffer Empty bit. If the RUPI is polled in this state, and the TBF bit indicates that the Transmit Buffer is empty, an automatic RR response will be generated. When a valid information frame is received the SIU will automatically increment Nr in the NSNR register and interrupt the CPU.

While in the AUTO mode the SIU can recognize and respond to the following commands without CPU intervention: I (Information), RR (Receive Ready), RNR (Receive Not Ready), REJ (Reject),

REGISTER NAMES	SYMBOLIC ADDRESS	BIT ADDRESS	BYTE ADDRESS	
B REGISTER	B	247 through 240	240	(F0H)
ACCUMULATOR	ACC	231 through 224	224	(E0H)
*THREE BYTE FIFO	FIFO		223	(DFH)
	FIFO		222	(DEH)
	FIFO		221	(DDH)
TRANSMIT BUFFER START	TBS		220	(DCH)
TRANSMIT BUFFER LENGTH	TBL		219	(DBH)
TRANSMIT CONTROL BYTE	TCB		218	(DAH)
* SIU STATE COUNTER	SIUST		217	(D9H)
SEND COUNT RECEIVE COUNT	NSNR	223 through 216	216	(D8H)
PROGRAM STATUS WORD	PSW	215 through 208	208	(D0H)
*DMA COUNT	DMA CNT		207	(CFH)
STATION ADDRESS	STAD		206	(CEH)
RECEIVE FIELD LENGTH	RFL		205	(CDH)
RECEIVE BUFFER START	RBS		204	(CCH)
RECEIVE BUFFER LENGTH	RBL		203	(CBH)
RECEIVE CONTROL BYTE	RCB		202	(CAH)
SERIAL MODE	SMD		201	(C9H)
STATUS REGISTER	STS	207 through 200	200	(C8H)
INTERRUPT PRIORITY CONTROL	IP	191 through 184	184	(B8H)
PORT 3	P3	183 through 176	176	(B0H)
INTERRUPT ENABLE CONTROL	IE	175 through 168	168	(A8H)
PORT 2	P2	167 through 160	160	(A0H)
PORT 1	P1	151 through 144	144	(90H)
TIMER HIGH 1	TH1		141	(8DH)
TIMER HIGH 0	TH0		140	(8CH)
TIMER LOW 1	TL1		139	(8BH)
TIMER LOW 0	TL0		138	(8AH)
TIMER MODE	TMOD		137	(89H)
TIMER CONTROL	TCON	143 through 136	136	(88H)
DATA POINTER HIGH	DPH		131	(83H)
DATA POINTER LOW	DPL		130	(82H)
STACK POINTER	SP		129	(81H)
PORT 0	P0	135 through 128	128	(80H)

SFR's CONTAINING DIRECT ADDRESSABLE BITS

*ICE Support Hardware registers. Under normal operating conditions there is no need for the CPU to access these registers.

Figure 4. Mapping of Special Function registers

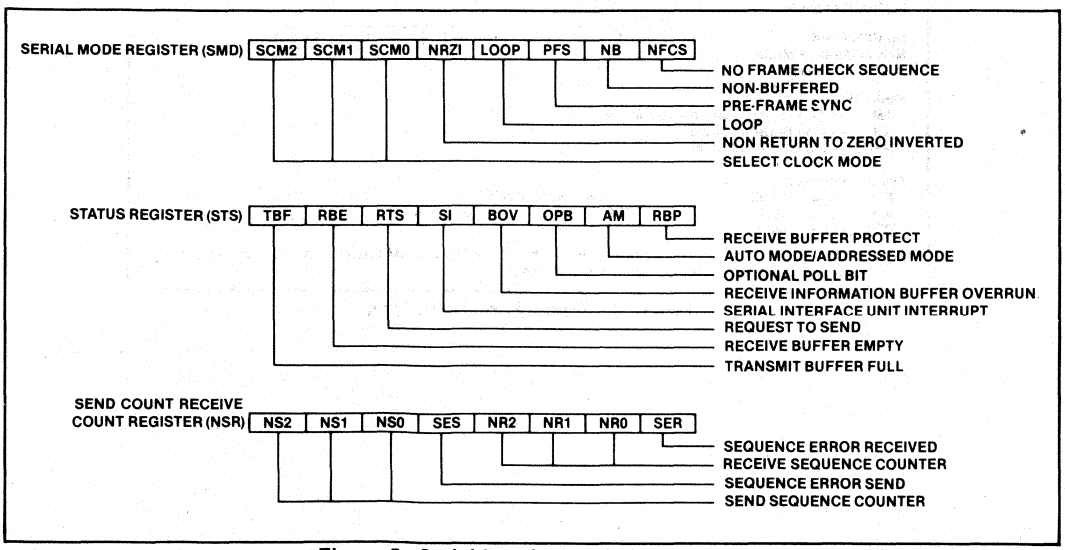


Figure 5. Serial Interface Unit Control registers

and UP (Unnumbered Poll). The SIU can generate the following responses without CPU intervention: I (Information), RR (Receive Ready), and RNR (Receive Not Ready).

When the Receive Buffer Empty bit (RBE) indicates that the Receive Buffer is empty, the receiver is enabled, and when the RBE bit indicates that the Receive Buffer is full, the receiver is disabled. Assuming that the Receive Buffer is empty, the RUPI will respond to a poll with an I frame if the Transmit Buffer is full. If the Transmit Buffer is empty, the RUPI will respond to a poll with a RR command if the Receive Buffer Protect bit (RBP) is cleared, or an RNR command if RBP is set.

Frame Format Options

In addition to the standard SDLC frame format, the RUPI will support the frames displayed in Figure 6. The standard SDLC frame is shown at the top of this figure. For the remaining frames the information field will incorporate the control or address bytes and the frame check sequences; therefore these fields will be stored in the Transmit and Receive buffers. For example, in the non-buffered mode the third byte is treated as the beginning of the information field. In the non-addressed mode, the information field begins after the opening flag. The mode bits to set the frame format options are found in the Serial Mode register and the Status register.

FRAME OPTION	NFCS	NB	AM[1]	FRAME FORMAT
Standard SDLC NON-AUTO Mode	0	0	0	F A C I FCS F
Standard SDLC AUTO Mode	0	0	1	F A C I FCS F
Non-Buffered Mode NON-AUTO Mode	0	1	1	F A I FCS F
Non-Addressed Mode NON-AUTO Mode	0	1	0	F I FCS F
No FCS Field NON-AUTO Mode	1	0	0	F A C I F
No FCS Field AUTO Mode	1	0	1	F A C I F
No FCS Field Non-Buffered Mode NON-AUTO Mode	1	1	1	F A I F
No FCS Field Non-Addressed Mode NON-AUTO Mode	1	1	0	F I F

Mode Bits:
- AM — "AUTO" Mode/Addressed Mode
- NB — Non-Buffered Mode
- NFCS — No FCS Field Mode

Key to Abbreviations:
- F = Flag (01111110)
- A = Address Field
- C = Control Field
- I = Information Field
- FCS = Frame Check Sequence

Note 1: The AM bit function is controlled by the NB bit. When NB = 0, AM becomes AUTO mode select, when NB = 1, AM becomes Address mode select.

Figure 6. Frame Format Options

HDLC

To realize an extended control field or an extended address field using the HDLC protocol, the NON-AUTO mode must be used. For an extended control field, the RUPI is programmed to be in the non-buffered mode. The extended control field will be the first and second bytes in the Receive and Transmit Buffers. For extended addressing the RUPI is placed in the non-addressed mode. In this mode the CPU must implement the address recognition for received frames. The addressing field will be the initial bytes in the Transmit and Receive buffers followed by the control field.

The RUPI can transmit and receive only frames which are multiples of 8 bits. For frames received with other than 8-bit multiples, a CRC error will cause the SIU to reject the frame.

SDLC Loop Configuration

The RUPI can be used in a SDLC loop as a secondary or primary station. When the RUPI is placed in the Loop mode it receives the data on pin 10 and transmits the data one bit time delayed on pin 11. It can also recognize the Go Ahead signal and change it into a flag when it is ready to transmit. As a secondary station the RUPI can be used in the AUTO or NON-AUTO modes. As a primary station the NON-AUTO mode is used; however, additional hardware is required for generating the Go Ahead bit pattern. In the Loop mode the maximum data rate is 1M bps clocked or 375K bps self-clocked.

SDLC Non-Loop Configuration

The RUPI can be used in a SDLC non-loop configuration as a secondary or primary station. When the RUPI is placed in the non-loop mode, data is received and transmitted on pin 11, and pin 10 drives a tri-state buffer. In non-loop mode, modem interface pins, $\overline{RTS}$ and $\overline{CTS}$, become available.

Data Clocking Option

The RUPI's serial port can operate in a clocked or self clocked system. A clocked system provides to the RUPI a clock synchronized to the data. A self-clocked system uses the RUPI's on-chip Digital Phase Locked Loop (DPLL) to recover the clock from the data, and clock this data into the Serial Receive Shift Register.

Externally Clocked Mode

In this mode, a clock synchronized with the data is externally fed into the RUPI. This clock may be generated from an External Phase Locked Loop, or possibly supplied along with the data. The RUPI can transmit and receive data in this mode at rates up to 2.4M bps.

Self Clocked Mode

This mode allows data transfer without a common system data clock. An on-chip Digital Phase Locked Loop is employed to recover the data clock which is encoded in the data stream. The DPLL will converge to the nominal bit center within eight bit transitions, worst case. The DPLL requires a reference clock of either 16 times (16x) or 32 times (32x) the data rate. This reference clock may be externally applied or internally generated. When internally generated either the RUPI's internal logic clock (crystal frequency divided by two) or the timer 1 overflow is used as the reference clock. Using the internal timer 1 clock the data rates can vary from 244 to 62.5K bps. Using the internal logic clock at a 16x sampling rate, receive data can either be 187.5K bps, or 375K bps. When the reference clock for the DPLL is externally applied the data rates can vary from 0 to 375K bps at a 16x sampling rate.

To aid in a Phase Locked Loop capture, the RUPI has a NRZI (Non Return to Zero Inverted) data encoding and decoding option. Additionally the RUPI has a pre-frame sync option that transmits two bytes of alternating 1's and 0's to ensure that the receive station DPLL will be synchronized with the data by the time it receives the opening flag.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias.....0 to 70°C
Storage Temperature.....−65°C to +150°C
Voltage on Any Pin With
Respect to Ground (V_{SS}).....−0.5V to +7V
Power Dissipation.....2 Watts

*Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

DC CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5V ± 10%, V_{SS} = 0V)

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
VIL	Input Low Voltage	−0.5		0.8	V	
VIH	Input High Voltage (Except RST/VPD and XTAL2)	2.0		VCC + 0.5	V	
VIH1	Input High Voltage To RST/VPD For Reset, XTAL2	2.5			V	XTAL1 to VSS
VPD	Power Down Voltage To RST/VPD	4.5		5.5	V	VCC = 0V
VOL	Output Low Voltage Ports 1, 2, 3 (Note 1)			0.45	V	IOL = 1.6mA
VOL1	Output Low Voltage Port 0 ALE, \PSEN (Note 1)			0.45	V	IOL = 3.2mA
VOH	Outpt High Voltage Ports 1, 2, 3	2.4			V	IOH = −60μA
VOH1	Output High Voltage Port 0, ALE, \PSEN	2.4			V	IOH = −400μA
IIL	Logical 0 Input Current Ports 1, 2, 3			−800	μA	XTAL1 at VSS VIL = 0.45V
IIH1	Input High Current TO RST/VPD For Reset			500	μA	Vin = VCC − 1.5V
ILI	Input Leakage Current To Port 0, \EA			10	μA	0 < Vin < VCC
ICC	Power Supply Current		125	200	mA	TA = 25°C
IPD	Power Down Current		15	30	mA	
CIO	Capacitance of I/O Buffer			10	pF	fc = 1MHz
IIL2	Logical 0 Input Current XTAL 2			−2.5	mA	XTAL1 = VSS VIL = 0.45V

Note 1: VOL is degraded when the RUPI-44 rapidly discharges external capacitance. This A.C. noise is most pronounced during emission of address data. When using external memory, locate the latch or buffer as close to the RUPI-44 as possible.

Datum	Emitting Ports	Time Interval	Degraded I/O Lines	VOL (peak) (max)
Address	P2, P0	T3, T9	P1, P3	.8V
Write Data	P0	T6	P1, P3, ALE	.8V

A.C. CHARACTERISTICS (T_A 0°C to 70°C, V_{CC} = 5V ± 10%, V_{SS} = 0V, C_L for Port 0, ALE and $\overline{PSEN}$ Outputs = 100pF; C_L for All Other Outputs = 80 pF)

Program Memory

Symbol	Parameter	12 MHz Clock			Variable Clock 1/TCLCL = 1.2 MHz to 12 MHz		
		Min	Max	Units	Min	Max	Units
TLHLL	ALE Pulse Width	127		ns	2TCLCL-40		ns
TAVLL	Address Setup to ALE	53		ns	TCLCL-30		ns
TLLAX[1]	Address Hold After ALE	48		ns	TCLCL-35		ns
TLLIV	ALE To Valid Instr In		233	ns		4TCLCL-100	ns
TLLPL	ALE To $\overline{PSEN}$	58		ns	TCLCL-25		ns
TPLPH	$\overline{PSEN}$ Pulse Width	215		ns	3TCLCL-35		ns
TPLIV	$\overline{PSEN}$ To Valid Instr In		125	ns		3TCLCL-125	ns
TPXIX	Input Instr Hold After $\overline{PSEN}$	0		ns	0		ns
TPXIZ[2]	Input Instr Float After $\overline{PSEN}$		63	ns		TCLCL-20	ns
TPXAV[2]	Address Valid After $\overline{PSEN}$	75		ns	TCLCL-8		ns
TAVIV	Address To Valid Instr In		302	ns		5TCLCL-115	ns
TAZPL	Address Float To $\overline{PSEN}$	0		ns	0		ns

Notes:
1. TLLAX for access to program memory is different from TLLAX for data memory.
2. Interfacing RUPI-44 devices with float times up to 75ns is permissible. This limited bus contention will not cause any damage to Port 0 drivers.

External Data Memory

Symbol	Parameter	12 MHz Clock			Variable Clock 1/TCLCL = 1.2 MHz to 12 MHz		
		Min	Max	Units	Min	Max	Units
TRLRH	$\overline{RD}$ Pulse Width	400		ns	6TCLCL-100		ns
TWLWH	$\overline{WR}$ Pulse Width	400		ns	6TCLCL-100		ns
TLLAX[1]	Address Hold After ALE	132		ns	2TCLCL-35		
TRLDV	$\overline{RD}$ To Valid Data In		250	ns		5TCLCL-165	ns
TRHDX	Data Hold After $\overline{RD}$	0		ns	0		ns
TRHDZ	Data Float After $\overline{RD}$		97	ns		2TCLCL-70	ns
TLLDV	ALE To Valid Data In		517	ns		8TCLCL-150	ns
TAVDV	Address To Valid Data In		585	ns		9TCLCL-165	ns
TLLWL	ALE To $\overline{WR}$ or $\overline{RD}$	200	300	ns	3TCLCL-50	3TCLCL + 50	ns
TAVWL	Address To $\overline{WR}$ or $\overline{RD}$	203		ns	4TCLCL-130		ns
TWHLH	$\overline{WR}$ or $\overline{RD}$ High To ALE High	43	123	ns	TCLCL-40	TCLCL + 40	ns
TDVWX	Data Valid To $\overline{WR}$ Transition	33		ns	TCLCL-50		ns
TQVWH	Data Setup Before $\overline{WR}$	433		ns	7TCLCL-150		ns
TWHQX	Data Hold After $\overline{WR}$	33		ns	TCLCL-50		ns
TRLAZ	Address Float After $\overline{RD}$		0	ns		0	ns

Note 1. TLLAX for access to program memory is different from TLLAX for access data memory.

Serial Interface

Symbol	Parameter	Min	Max	Units	
tDCY	Data Clock	420		ns	
tDCL	Data Clock Low	180		ns	
tDCH	Data Clock High	120		ns	

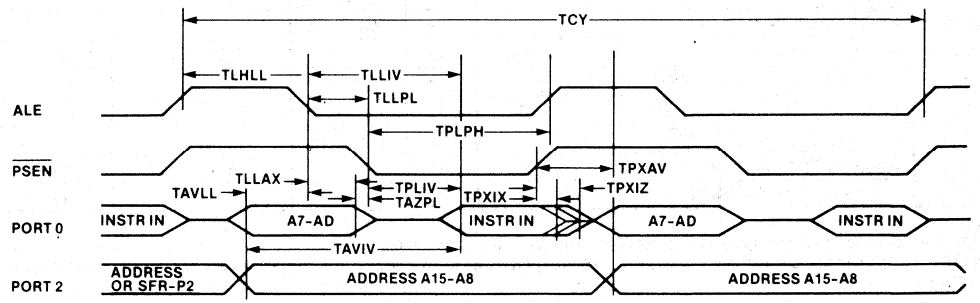
Serial Interface (Continued)

tTD	Transmit Data Delay		80	ns	
tDSS	Data Setup Time	60		ns	
tDHS	Data Hold Time	40		ns	

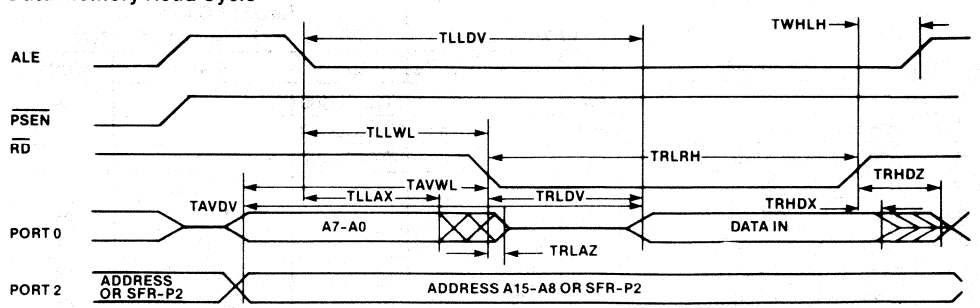
WAVEFORMS

Memory Access

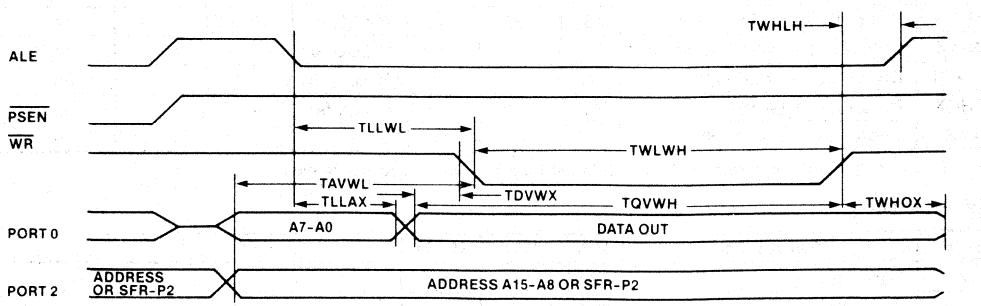
Program Memory Read Cycle



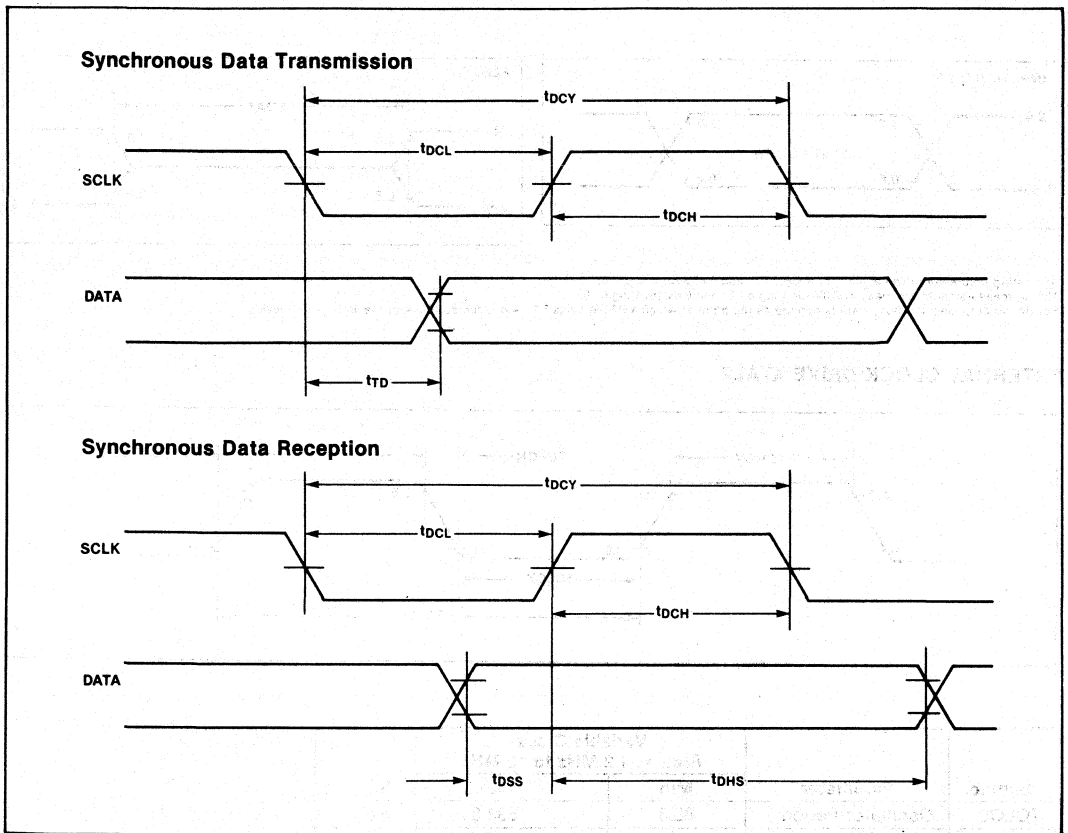
Data Memory Read Cycle



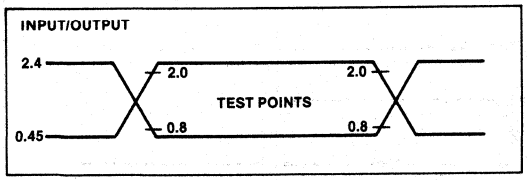
Data Memory Write Cycle

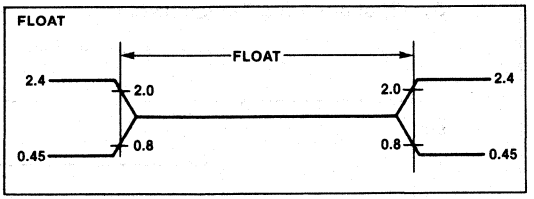


SERIAL I/O WAVEFORMS



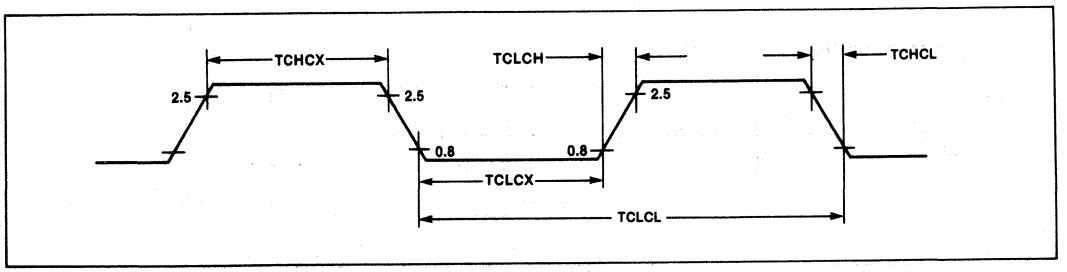
AC TESTING INPUT, OUTPUT, FLOAT WAVEFORMS





AC testing inputs are driven at 2.4V for a logic "1" and 0.45V for a logic "0."
Timing measurements are made at 2.0V for a logic "1" and 0.8V for a logic "0."
For timing purposes, the float state is defined as the point at which a P0 pin sinks 3.2mA or sources 400μA at the voltage test levels.

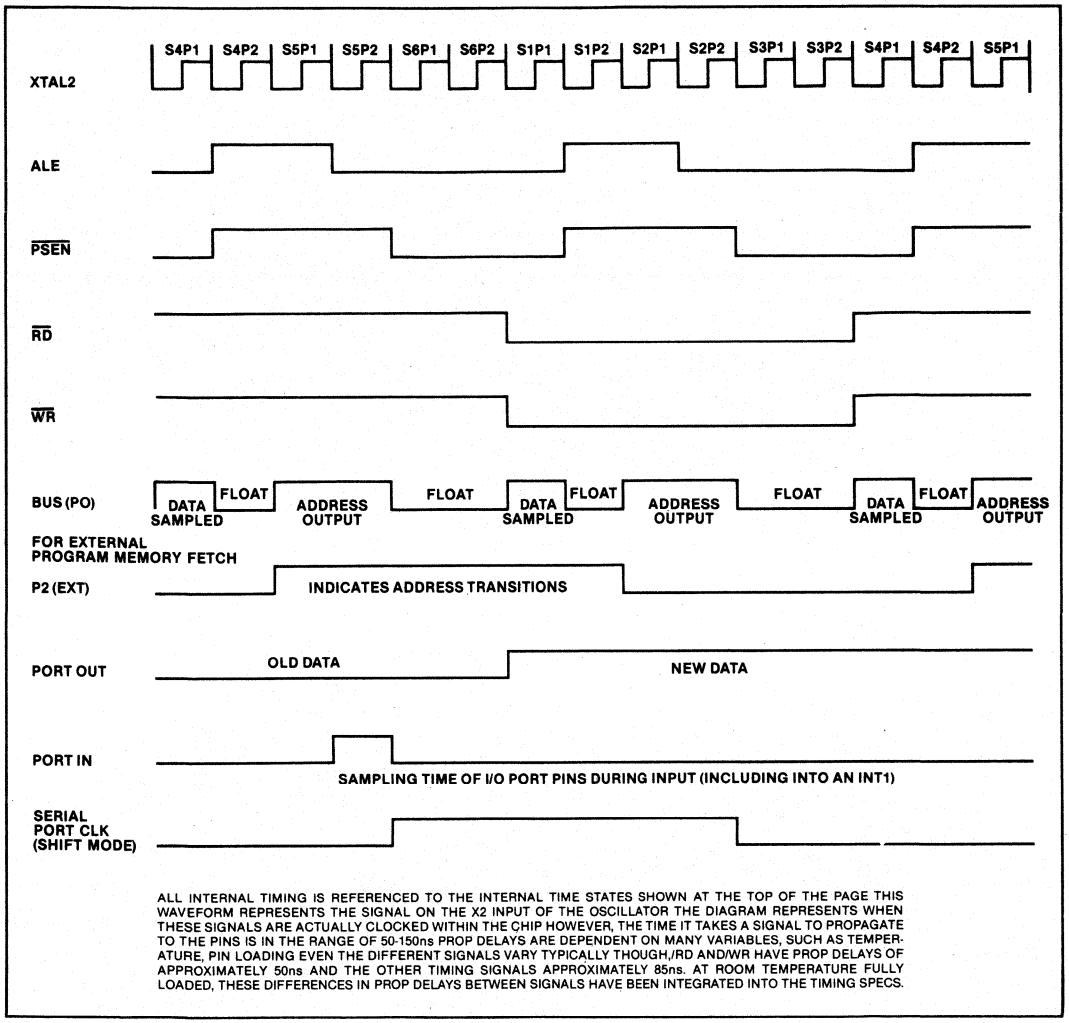
EXTERNAL CLOCK DRIVE XTAL2



Symbol	Parameter	Variable Clock Freq = 1.2 MHz to 12 MHz		Unit
		Min	Max	
TCLCL	Oscillator Period	83.3	833.3	ns
TCHCX	High Time	20	TCLCL-TCLCX	ns
TCLCX	Low Time	20	TCLCL-TCHCX	ns
TCLCH	Rise Time		20	ns
TCHCL	Fall Time		20	ns

Note 1: EPROM Programming
Refer to 8751 data sheet

WAVEFORM



ALL INTERNAL TIMING IS REFERENCED TO THE INTERNAL TIME STATES SHOWN AT THE TOP OF THE PAGE THIS WAVEFORM REPRESENTS THE SIGNAL ON THE X2 INPUT OF THE OSCILLATOR THE DIAGRAM REPRESENTS WHEN THESE SIGNALS ARE ACTUALLY CLOCKED WITHIN THE CHIP HOWEVER, THE TIME IT TAKES A SIGNAL TO PROPAGATE TO THE PINS IS IN THE RANGE OF 50-150ns PROP DELAYS ARE DEPENDENT ON MANY VARIABLES, SUCH AS TEMPERATURE, PIN LOADING EVEN THE DIFFERENT SIGNALS VARY TYPICALLY THOUGH,/RD AND/WR HAVE PROP DELAYS OF APPROXIMATELY 50ns AND THE OTHER TIMING SIGNALS APPROXIMATELY 85ns. AT ROOM TEMPERATURE FULLY LOADED, THESE DIFFERENCES IN PROP DELAYS BETWEEN SIGNALS HAVE BEEN INTEGRATED INTO THE TIMING SPECS.

PRELIMINARY

8032/8052
SINGLE-COMPONENT 8-BIT MICROCOMPUTER

- 8032—Control-Oriented CPU with RAM and I/O
- 8052—An 8032 with Factory Mask-Programmable ROM

- 8K x 8 ROM (8052 only)
- 256 x 8 RAM
- 32 I/O lines (Four 8-Bit Ports)
- Three 16-Bit Timer/Counters
- Programmable Full-Duplex Serial Channel
- Variable Transmit/Receive Baud Rate Capability
- Timer 2 Capture Capability
- 128K Accessible External Memory
- Boolean Processor
- 218 User Bit-Addressable Locations
- Upward Compatible with 8031AH/8051AH

The 8032/8052 is the highest performance member of Intel's MCS®-51 family of 8-bit microcomputers. It is fabricated with Intel's highly reliable +5 depletion-load, N-channel, silicon gate HMOS-II technology, and like the other MCS-51 members is packaged in a 40-pin DIP.

The 8032 contains 256 bytes of read/write data memory; 32 I/O lines configured as four 8-bit ports; three 16-bit timer/counters; a six-source, two-priority level, nested interrupt structure; a programmable serial I/O port; and an on-chip oscillator with clock circuitry. The 8052 has all of these features plus 8K bytes of non-volatile read-only program memory. Both microcomputers have memory expansion capabilities of up to 64K bytes of data storage and 64K bytes of program memory that may be realized with standard TTL compatible memories and/or byte-oriented MCS-80 and MCS-85 peripherals.

The 8032/8052 microcomputer, characteristic of the entire MCS-51 family, is efficient at both computational and control-oriented tasks. This results from its extensive BCD/binary arithmetic and bit-handling facilities. Efficient use of program memory is also achieved by using the familiar compact instruction set of the 8031/8051. Forty-four percent of the instructions are one-byte, 41% two byte and 15% three-byte. The majority of the instructions execute in just 1.0 μs at 12 MHz operation. The longest instructions, multiply and divide, require only 4μs at 12 MHz.

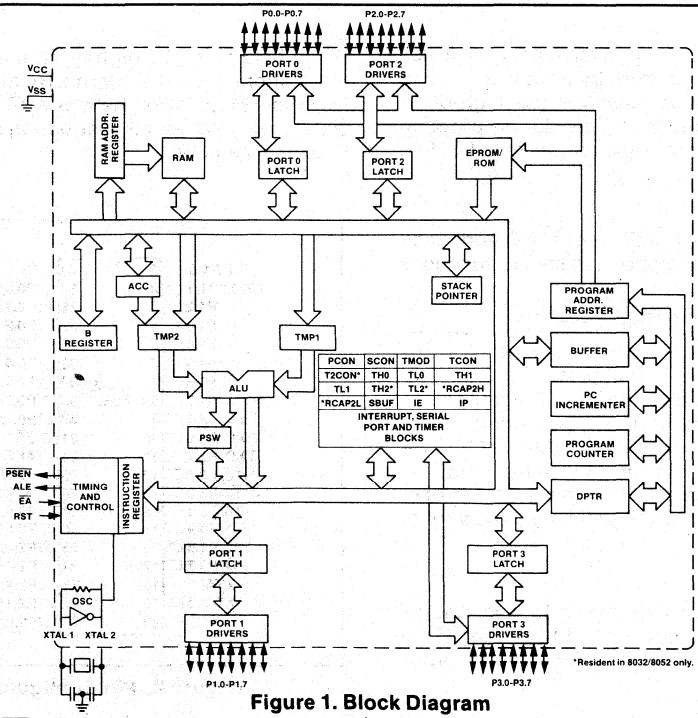


Figure 1. Block Diagram

8032/8052 PIN DESCRIPTIONS

V_{SS}
Circuit ground potential.

V_{CC}
+5V power supply during operation and program verification.

Port 0
Port 0 is an 8-bit open drain bidirectional I/O port. It is also the multiplexed low-order address and data bus when using external memory. It is used for data output during program verification. Port 0 can sink/source eight LS TTL loads.

Port 1
Port 1 is an 8-bit quasi-bidirectional I/O port. Pins P1.0 and P1.1 also correspond to the special functions T2, Timer 2 counter trigger input, and T2EX, external input to Timer 2. The output latch on these two special function pins must be programmed to a one (1) for that function to operate. Port 1 is also used for the low-order address byte during program verification. Port 1 can sink/source four LS TTL loads.

Port 2
Port 2 is an 8-bit quasi-bidirectional I/O port. It also emits the high-order address byte when accessing external memory. It is used for the high-order address and the control signals during program verification. Port 2 can sink/source four LS TTL loads.

Port 3
Port 3 is an 8-bit quasi-bidirectional I/O port. Each of the P3 pins also correspond to special functions as listed below:

Port Pin	Alternate Function
P3.0	RXD (serial input/output port)
P3.1	TXD (serial output port)
P3.2	$\overline{INT0}$ (external interrupt 0 input)
P3.3	$\overline{INT1}$ (external interrupt 1 input)
P3.4	T0 (Timer/Counter 0 external input)
P3.5	T1 (Timer/Counter 1 external input)
P3.6	$\overline{WR}$ (external Data Memory write strobe)
P3.7	$\overline{RD}$ (external Data Memory read strobe)

The output latch corresponding to a secondary function must be programmed to a one (1) for that function to operate. Port 3 can sink/source four LS TTL loads.

RST/V_{PD}
A high on this pin for two machine cycles while the oscillator is running resets the device. A small external pulldown resistor ($\approx$8.2kΩ) from RST/V_{PD} to V_{SS} permits power-on reset when a capacitor ($\approx$10 μf) is also connected from this pin to V_{CC}.

If RST/V_{PD} is held within the V_{PD} spec while V_{CC} drops below its spec, RST/V_{PD} will provide standby power to the RAM. When RST/V_{PD} is low, the RAM's current is drawn from V_{CC}.

ALE
Provides Address Latch Enable output used for latching the address into external memory during normal operation. It is activated every six oscillator periods except during an external data memory access.

$\overline{PSEN}$
The Program Store Enable output is a control signal that enables the external Program Memory to the bus during external fetch operations. It is activated every six oscillator periods, except during external data memory accesses. Remains high during internal program execution.

$\overline{EA}$
When held at a TTL high level, the 8052 executes instructions from the internal ROM when the PC is less than 8192. When held at a TTL low level, the 8032/8052 fetches all instructions from external Program Memory.

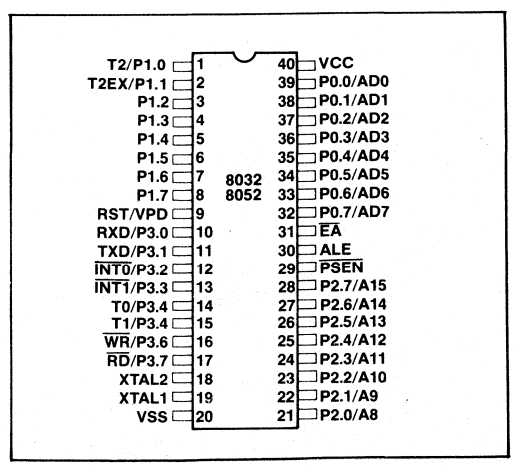


Figure 2. Pin Configuration

XTAL1
Input to the inverting amplifier that forms the oscillator.

XTAL2
Output of the inverting amplifier that forms the oscillator, and input to the internal clock generator. Receives the external oscillator signal when an external oscillator is used.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias . . . 0°C to 70°C

Storage Temperature . . . . . -65°C to +150°C

Voltage on Any Pin With Respect to Ground (V_{SS}) . . . . . -0.5V to +7V

Power Dissipation . . . . . 2 Watts

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

DC CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 4.5V to 5.5V, V_{SS} = 0V)

Symbol	Parameter	Min	Max	Unit	Test Conditions
VIL	Input Low Voltage	-0.5	0.8	V	
VIH	Input High Voltage (Except RST/VPD and XTAL2)	2.0	V_{CC} + 0.5	V	
VIH1	Input High Voltage to RST/VPD for Reset, XTAL2	2.5	V_{CC} + 0.5	V	XTAL1 to V_{SS}
VPD	Power Down Voltage to RST/VPD	4.5	5.5	V	V_{CC} = 0V
VOL	Output Low Voltage Ports 1, 2, 3 (Note 1)		0.45	V	IOL = 1.6mA
VOL1	Output Low Voltage Port 0, ALE,PSEN (Note 1)		0.45	V	IOL = 3.2mA
VOH	Output High Voltage Ports 1, 2, 3	2.4		V	IOH = -80μA
VOH1	Output High Voltage Port 0, ALE, $\overline{PSEN}$	2.4		V	IOH = -400μA
IIL	Logical 0 Input Current Ports 1, 2, 3		-800	μA	Vin = 0.45V
IIL2	Logical 0 Input Current XTAL2		-2.5	mA	XTAL1 at V_{SS}, Vin=0.45V
ILI	Input Leakage Current To Port 0, EA		±10	μA	0.45V < Vin < V_{CC}
IIH1	Input High Current to RST/VPD For Reset		500	μA	Vin = V_{CC} - 1.5V
ICC	Power Supply Current		175	mA	All outputs disconnected; $\overline{EA}$ = V_{CC}
IPD	Power Down Current		20	mA	V_C = 0V
CIO	Capacitance of I/O Buffer		10	pF	f_c = 1MHz, T_A = 25°C

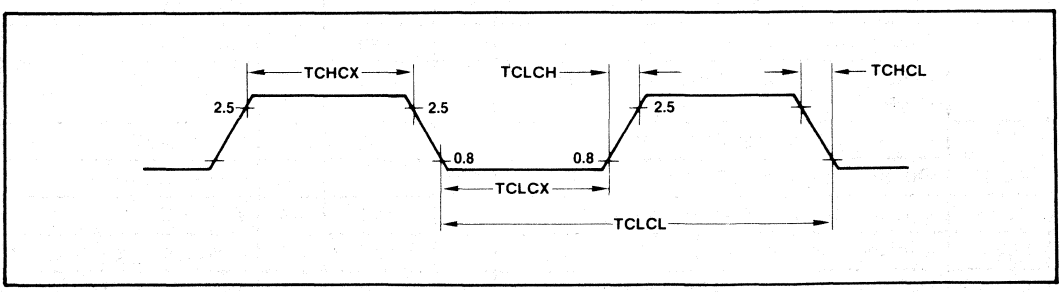
See page 4 for Notes.

Note 1: Vol is degraded when the 8032/8052 rapidly discharges external capacitance. This AC noise is most pronounced during emission of address data. When using external memory, locate the latch or buffer as close to the 8032/8052 as possible.

Datum	Emitting Ports	Degraded I/O Lines	VOL (peak) (max)
Address	P2, P0	P1, P3	0.8V
Write Data	P0	P1, P3, ALE	0.8V

EXTERNAL CLOCK DRIVE CHARACTERISTICS (XTAL2)

Symbol	Parameter	Variable Clock f = 1.2 MHz to 12 MHz		Unit
		Min	Max	
TCLCL	Oscillator Period	83.3	833.3	ns
TCHCX	High Time	20		ns
TCLCX	Low Time	20		ns
TCLCH	Rise Time		20	ns
TCHCL	Fall Time		20	ns



AC CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5V ±10%, V_{SS} = 0V, C_L for Port 0, ALE and $\overline{PSEN}$ Outputs = 100 pF, C_L for all other outputs = 80 pF)

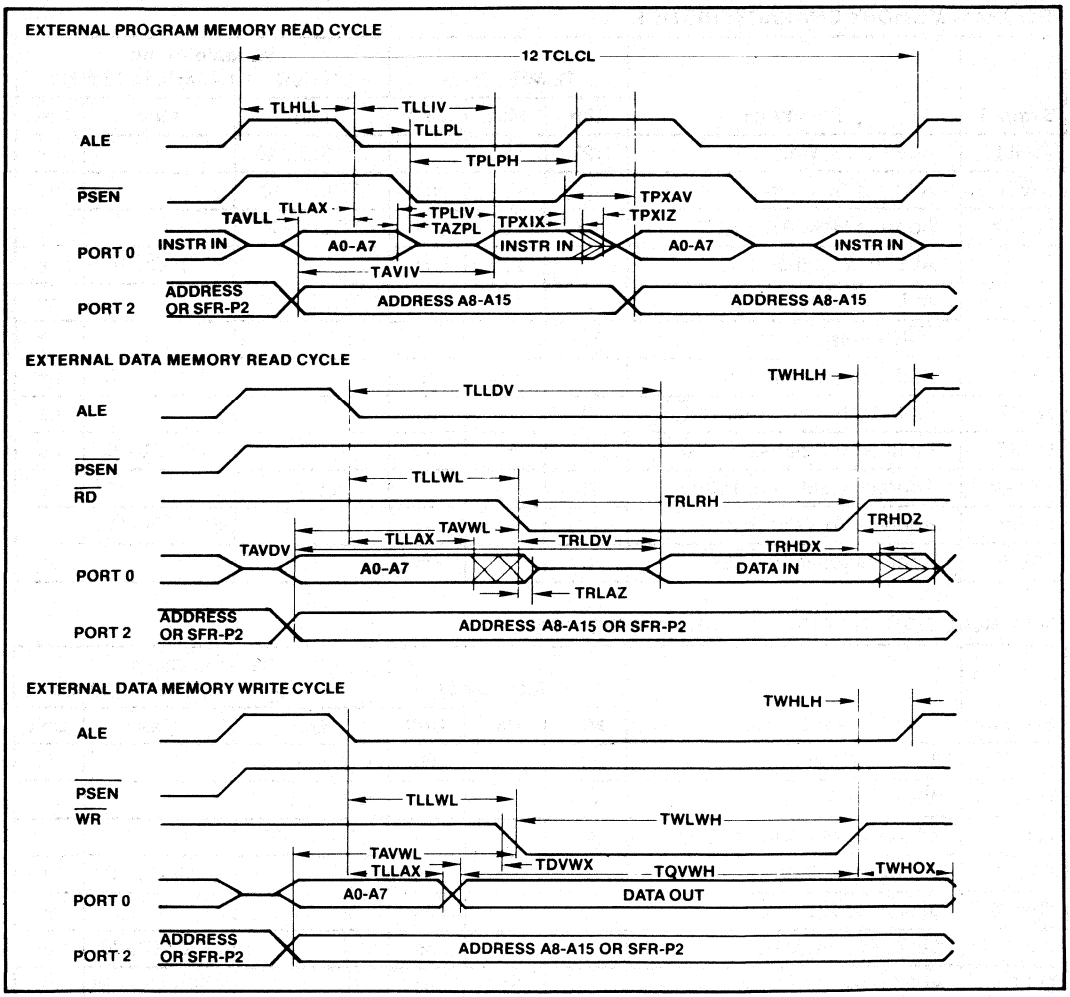
PROGRAM MEMORY CHARACTERISTICS

Symbol	Parameter	12 MHz Clock			Variable Clock 1/TCLCL = 1.2 MHz to 12 MHz		
		Min	Max	Unit	Min	Max	Unit
TLHLL	ALE Pulse Width	127		ns	2TCLCL-40		ns
TAVLL	Address Setup to ALE	43		ns	TCLCL-40		ns
TLLAX	Address Hold After ALE	48		ns	TCLCL-35		ns
TLLIV	ALE to Valid Instr In		233	ns		4TCLCL-100	ns
TLLPL	ALE To $\overline{PSEN}$	58		ns	TCLCL-25		ns
TPLPH	$\overline{PSEN}$ Pulse Width	215		ns	3TCLCL-35		ns
TPLIV	$\overline{PSEN}$ To Valid Instr In		125	ns		3TCLCL-125	ns
TPXIX	Input Instr Hold After $\overline{PSEN}$	0		ns	0		ns
TPXIZ	Input Instr Float After $\overline{PSEN}$		63	ns		TCLCL-20	ns
TPXAV	Address Valid After $\overline{PSEN}$	75		ns	TCLCL-8		ns
TAVIV	Address To Valid Instr In		302	ns		5TCLCL-115	ns
TAZPL	Address Float To $\overline{PSEN}$	0		ns	0		ns

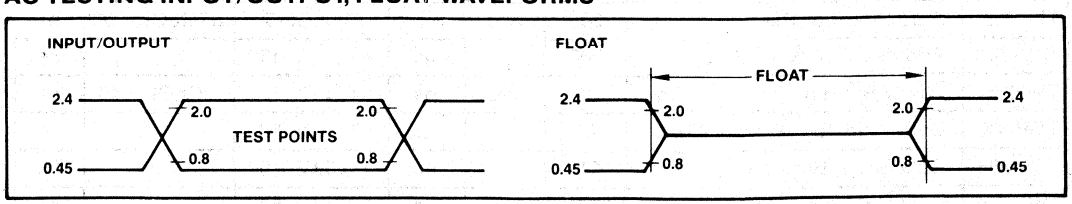
EXTERNAL DATA MEMORY CHARACTERISTICS

Symbol	Parameter	12 MHz Clock			Variable Clock 1/TCLCL = 1.2 MHz to 12 MHz		
		Min	Max	Unit	Min	Max	Unit
TRLRH	$\overline{RD}$ Pulse Width	400		ns	6TCLCL-100		ns
TWLWH	$\overline{WR}$ Pulse Width	400		ns	6TCLCL-100		ns
TLLAX	Address Hold After ALE	48		ns	TCLCL-35		
TRLDV	$\overline{RD}$ To Valid Data In		250	ns		5TCLCL-165	ns
TRHDX	Data Hold After $\overline{RD}$	0		ns	0		ns
TRHDZ	Data Float After $\overline{RD}$		97	ns		2TCLCL-70	ns
TLLDV	ALE To Valid Data In		517	ns		8TCLCL-150	ns
TAVDV	Address To Valid Data In		585	ns		9TCLCL-165	ns
TLLWL	ALE To $\overline{WR}$ or $\overline{RD}$	200	300	ns	3TCLCL-50	3TCLCL + 50	ns
TAVWL	Address To $\overline{WR}$ or $\overline{RD}$	203		ns	4TCLCL-130		ns
TWHLH	$\overline{WR}$ or $\overline{RD}$ High To ALE High	43	123	ns	TCLCL-40	TCLCL + 40	ns
TDVWX	Data Valid To $\overline{WR}$ Transition	23		ns	TCLCL-60		ns
TQVWH	Data Setup Before $\overline{WR}$	433		ns	7TCLCL-150		ns
TWHQX	Data Hold After $\overline{WR}$	33		ns	TCLCL-50		ns
TRLAZ	Address Float After $\overline{RD}$		0	ns		0	ns

AC TIMING DIAGRAMS

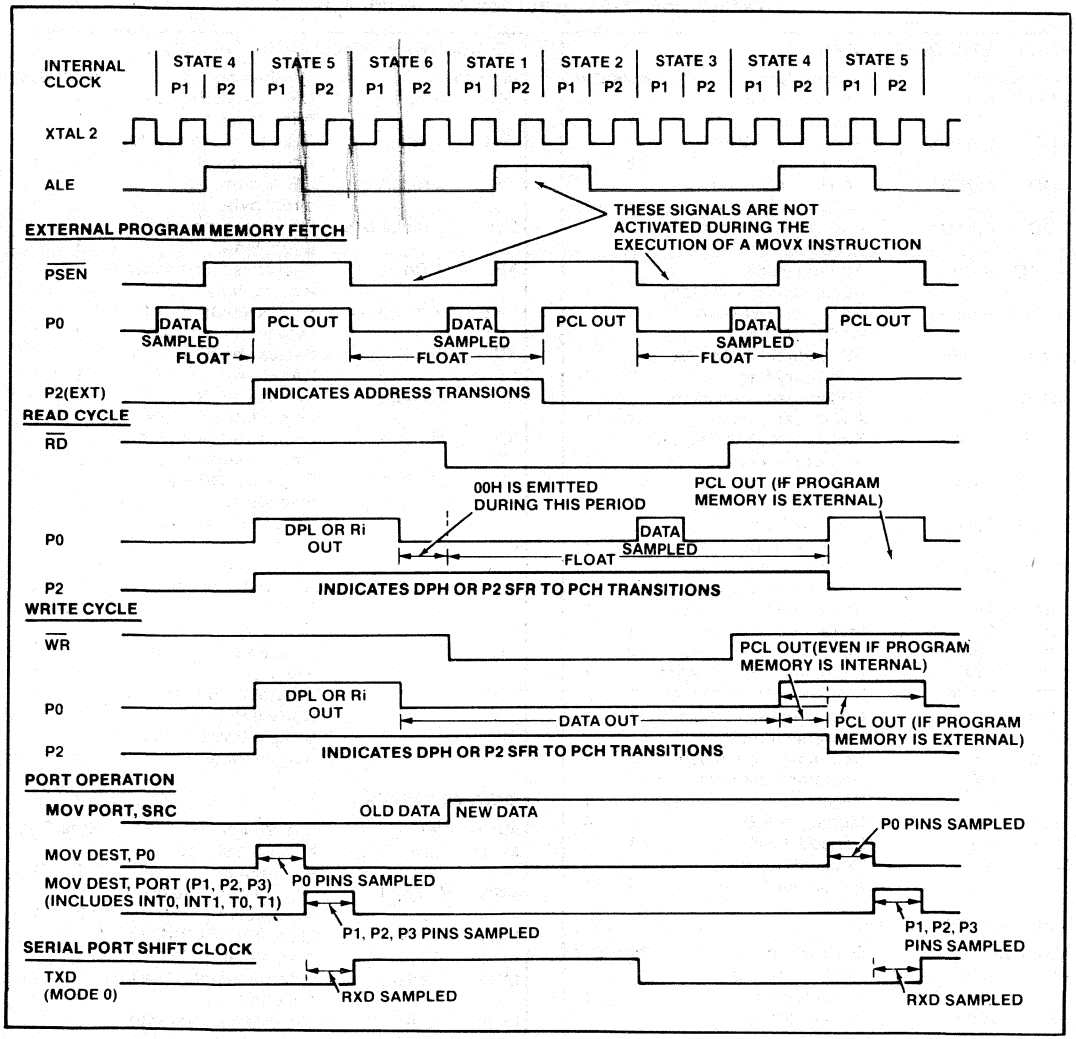


AC TESTING INPUT/OUTPUT, FLOAT WAVEFORMS



AC inputs during testing are driven at 2.4V for a logic "1" and 0.45V for a logic "0". Timing measurements are made at 2.0V for a logic "1" and 0.8V for a logic "0". For timing purposes, the float state is defined as the point at which a P0 pin sinks 2.4mA or sources 400 μA at the voltage test levels.

CLOCK WAVEFORMS



This diagram indicates when signals are clocked internally. The time it takes the signals to propagate to the pins, however, ranges from 25 to 125 ns. This propagation delay is dependent on variables such as temperature and pin loading. Propagation also varies from output to output and component to component. Typically though, (T_A = 25° C, fully loaded) RD and WR propagation delays are approximately 50 ns. The other signals are typically 85 ns. Propagation delays are incorporated in the AC specifications.

Table 1. MCS®-51 Instruction Set Description

ARITHMETIC OPERATIONS

Mnemonic		Description	Byte	Cyc
ADD	A,Rn	Add register to Accumulator	1	1
ADD	A,direct	Add direct byte to Accumulator	2	1
ADD	A,@Ri	Add indirect RAM to Accumulator	1	1
ADD	A,#data	Add immediate data to Accumulator	2	1
ADDC	A,Rn	Add register to Accumulator with Carry	1	1
ADDC	A,direct	Add direct byte to A with Carry flag	2	1
ADDC	A,@Ri	Add indirect RAM to A with Carry flag	1	1
ADDC	A,#data	Add immediate data to A with Carry flag	2	1
SUBB	A,Rn	Subtract register from A with Borrow	1	1
SUBB	A,direct	Subtract direct byte from A with Borrow	2	1
SUBB	A,@Ri	Subtract indirect RAM from A with Borrow	1	1
SUBB	A,#data	Subtract immed data from A with Borrow	2	1
INC	A	Increment Accumulator	1	1
INC	Rn	Increment register	1	1
INC	direct	Increment direct byte	2	1
INC	@Ri	Increment indirect RAM	1	1
INC	DPTR	Increment Data Pointer	1	2
DEC	A	Decrement Accumulator	1	1
DEC	Rn	Decrement register	1	1
DEC	direct	Decrement direct byte	2	1
DEC	@Ri	Decrement indirect RAM	1	1
MUL	AB	Multiply A & B	1	4
DIV	AB	Divide A by B	1	4
DA	A	Decimal Adjust Accumulator	1	1

LOGICAL OPERATIONS

Mnemonic		Destination	Byte	Cyc
ANL	A,Rn	AND register to Accumulator	1	1
ANL	A,direct	AND direct byte to Accumulator	2	1
ANL	A,@Ri	AND indirect RAM to Accumulator	1	1
ANL	A,#data	AND immediate data to Accumulator	2	1
ANL	direct,A	AND Accumulator to direct byte	2	1
ANL	direct,#data	AND immediate data to direct byte	3	2
ORL	A,Rn	OR register to Accumulator	1	1
ORL	A,direct	OR direct byte to Accumulator	2	1

LOGICAL OPERATIONS (CONTINUED)

Mnemonic		Destination	Byte	Cyc
ORL	A,@Ri	OR indirect RAM to Accumulator	1	1
ORL	A,#data	OR immediate data to Accumulator	2	1
ORL	direct,A	OR Accumulator to direct byte	2	1
ORL	direct,#data	OR immediate data to direct byte	3	2
XRL	A,Rn	Exclusive-OR register to Accumulator	1	1
XRL	A,direct	Exclusive-OR direct byte to Accumulator	2	1
XRL	A,@Ri	Exclusive-OR indirect RAM to A	1	1
XRL	A,#data	Exclusive-OR immediate data to A	2	1
XRL	direct,A	Exclusive-OR Accumulator to direct byte	2	1
XRL	direct,#data	Exclusive-OR immediate data to direct	3	2
CLR	A	Clear Accumulator	1	1
CPL	A	Complement Accumulator	1	1
RL	A	Rotate Accumulator Left	1	1
RLC	A	Rotate A Left through the Carry flag	1	1
RR	A	Rotate Accumulator Right	1	1
RRC	A	Rotate A Right through Carry flag	1	1
SWAP	A	Swap nibbles within the Accumulator	1	1

DATA TRANSFER

Mnemonic		Description	Byte	Cyc
MOV	A,Rn	Move register to Accumulator	1	1
MOV	A,direct	Move direct byte to Accumulator	2	1
MOV	A,@Ri	Move indirect RAM to Accumulator	1	1
MOV	A,#data	Mov immediate data to Accumulator	2	1
MOV	Rn,A	Move Accumulator to register	1	1
MOV	Rn,direct	Move direct byte to register	2	2
MOV	Rn,#data	Move immediate data to register	2	1
MOV	direct,A	Move Accumulator to direct byte	2	1
MOV	direct,Rn	Move register to direct byte	2	2
MOV	direct,direct	Move direct byte to direct	3	2
MOV	direct,@Ri	Move indirect RAM to direct byte	2	2

Table 1. (Cont.)

DATA TRANSFER (CONTINUED)

Mnemonic		Description	Byte	Cyc
MOV	direct,#data	Move immediate data to direct byte	3	2
MOV	@Ri,A	Move Accumulator to indirect RAM	1	1
MOV	@Ri,direct	Move direct byte to indirect RAM	2	2
MOV	@Ri,#data	Move immediate data to indirect RAM	2	1
MOV	DPTR,#data16	Load Data Pointer with a 16-bit constant	3	2
MOVC	A,@A+DPTR	Move Code byte relative to DPTR to A	1	2
MOVC	A,@A+PC	Move Code byte relative to PC to A	1	2
MOVX	A,@Ri	Move External RAM (8-bit addr) to A	1	2
MOVX	A,@DPTR	Move External RAM (16-bit addr) to A	1	2
MOVX	@Ri,A	Move A to External RAM (8-bit addr)	1	2
MOVX	@DPTR,A	Move A to External RAM (16-bit addr)	1	2
PUSH	direct	Push direct byte onto stack	2	2
POP	direct	Pop direct byte from stack	2	2
XCH	A,Rn	Exchange register with Accumulator	1	1
XCH	A,direct	Exchange direct byte with Accumulator	2	1
XCH	A,@Ri	Exchange indirect RAM with A	1	1
XCHD	A,@Ri	Exchange low-order Digit ind RAM w A	1	1

BOOLEAN VARIABLE MANIPULATION

Mnemonic		Description	Byte	Cyc
CLR	C	Clear Carry flag	1	1
CLR	bit	Clear direct bit	2	1
SETB	C	Set Carry flag	1	1
SETB	bit	Set direct Bit	2	1
CPL	C	Complement Carry flag	1	1
CPL	bit	Complement direct bit	2	1
ANL	C,bit	AND direct bit to Carry flag	2	2
ANL	C,1 bit	AND complement of direct bit to Carry	2	2
ORL	C/bit	OR direct bit to Carry flag	2	2
ORL	C,1 bit	OR complement of direct bit to Carry	2	2
MOV	C/bit	Move direct bit to Carry flag	2	1
MOV	bit,C	Move Carry flag to direct bit	2	2

PROGRAM AND MACHINE CONTROL

Mnemonic		Description	Byte	Cyc
ACALL	addr11	Absolute Subroutine Call	2	2
LCALL	addr16	Long Subroutine Call	3	2
RET		Return from subroutine	1	2
RETI		Return from interrupt	1	2
AJMP	addr11	Absolute Jump	2	2
LJMP	addr16	Long Jump	3	2
SJMP	rel	Short Jump (relative addr)	2	2
JMP	@A+DPTR	Jump indirect relative to the DPTR	1	2
JZ	rel	Jump if Accumulator is Zero	2	2
JNZ	rel	Jump if Accumulator is Not Zero	2	2
JC	rel	Jump if Carry flag is set	2	2
JNC	rel	Jump if No Carry flag	2	2
JB	bit,rel	Jump if direct Bit set	3	2
JNB	bit,rel	Jump if direct Bit Not set	3	2
JBC	bit,rel	Jump if direct Bit is set & Clear bit	3	2
CJNE	A,direct,rel	Compare direct to A & Jump if Not Equal	3	2
CJNE	A,#data,rel	Comp, immed, to A & Jump if Not Equal	3	2
CJNE	Rn,#data,rel	Comp, immed, to reg & Jump if Not Equal	3	2
CJNE	@Ri,#data,rel	Comp, immed, to ind, & Jump if Not Equal	3	2
DJNZ	Rn,rel	Decrement register & Jump if Not Zero	2	2
DJNZ	direct,rel	Decrement direct & Jump if Not Zero	3	2
NOP		No operation	1	1

Notes on data addressing modes:

Rn —Working register R0-R7
direct —128 internal RAM locations, any I/O port, control or status register
@Ri —Indirect internal RAM location addressed by register R0 or R1
#data —8-bit constant included in instruction
#data16 —16-bit constant included as bytes 2 & 3 of instruction
bit —128 software flags, any I/O pin, control or status bit

Notes on program addressing modes:

addr16 —Destination address for LCALL & LJMP may be anywhere within the 64-K program memory address space
Addr11 —Destination address for ACALL & AJMP will be within the same 2-K page of program memory as the first byte of the following instruction
rel —SJMP and all conditional jumps include an 8-bit offset byte, Range is +127-128 bytes relative to first byte of the following instruction

Table 2. Instruction Opcodes in Hexadecimal Order

Hex Code	Number of Bytes	Mnemonic	Operands
00	1	NOP	
01	2	AJMP	code addr
02	3	LJMP	code addr
03	1	RR	A
04	1	INC	A
05	2	INC	data addr
06	1	INC	@R0
07	1	INC	@R1
08	1	INC	R0
09	1	INC	R1
0A	1	INC	R2
0B	1	INC	R3
0C	1	INC	R4
0D	1	INC	R5
0E	1	INC	R6
0F	1	INC	R7
10	3	JBC	bit addr, code addr
11	2	ACALL	code addr
12	3	LCALL	code addr
13	1	RRC	A
14	1	DEC	A
15	2	DEC	data addr
16	1	DEC	@R0
17	1	DEC	@R1
18	1	DEC	R0
19	1	DEC	R1
1A	1	DEC	R2
1B	1	DEC	R3
1C	1	DEC	R4
1D	1	DEC	R5
1E	1	DEC	R6
1F	1	DEC	R7
20	3	JB	bit addr, code addr
21	2	AJMP	code addr
22	1	RET	
23	1	RL	A
24	2	ADD	A,#data
25	2	ADD	A,data addr
26	1	ADD	A,@R0
27	1	ADD	A,@R1
28	1	ADD	A,R0
29	1	ADD	A,R1
2A	1	ADD	A,R2
2B	1	ADD	A,R3
2C	1	ADD	A,R4
2D	1	ADD	A,R5
2E	1	ADD	A,R6
2F	1	ADD	A,R7
30	3	JNB	bit addr, code addr
31	2	ACALL	code addr
32	1	RETI	
33	1	RLC	A
34	2	ADDC	A,#data
35	2	ADDC	A,data addr
36	1	ADDC	A,@R0
37	1	ADDC	A,@R1
38	1	ADDC	A,R0
39	1	ADDC	A,R1
3A	1	ADDC	A,R2
3B	1	ADDC	A,R3
3C	1	ADDC	A,R4
3D	1	ADDC	A,R5
3E	1	ADDC	A,R6
3F	1	ADDC	A,R7
40	2	JC	code addr
41	2	AJMP	code addr
42	2	ORL	data addr,A
43	3	ORL	data addr,#data
44	2	ORL	A,#data
45	2	ORL	A,data addr
46	1	ORL	A,@R0
47	1	ORL	A,@R1
48	1	ORL	A,R0
49	1	ORL	A,R1
4A	1	ORL	A,R2
4B	1	ORL	A,R3
4C	1	ORL	A,R4
4D	1	ORL	A,R5
4E	1	ORL	A,R6
4F	1	ORL	A,R7
50	2	JNC	code addr
51	2	ACALL	code addr
52	2	ANL	data addr,A
53	3	ANL	data addr,#data
54	2	ANL	A,#data
55	2	ANL	A,data addr
56	1	ANL	A,@R0
57	1	ANL	A@R1
58	1	ANL	A,R0
59	1	ANL	A,R1
5A	1	ANL	A,R2
5B	1	ANL	A,R3
5C	1	ANL	A,R4
5D	1	ANL	A,R5
5E	1	ANL	A,R6
5F	1	ANL	A,R7
60	2	JZ	code addr
61	2	AJMP	code addr
62	2	XRL	data addr.A
63	3	XRL	data addr,#data
64	2	XRL	A,#data
65	2	XRL	A,data addr

Table 2. (Cont.)

Hex Code	Number of Bytes	Mnemonic	Operands
66	1	XRL	A,@R0
67	1	XRL	A,@R1
68	1	XRL	A,R0
69	1	XRL	A,R1
6A	1	XRL	A,R2
6B	1	XRL	A,R3
6C	1	XRL	A,R4
6D	1	XRL	A,R5
6E	1	XRL	A,R6
6F	1	XRL	A,R7
70	2	JNZ	code addr
71	2	ACALL	code addr
72	2	ORL	C,bit addr
73	1	JMP	@A+DPTR
74	2	MOV	A,#data
75	3	MOV	data addr,#data
76	2	MOV	@R0,#data
77	2	MOV	@R1,#data
78	2	MOV	R0,#data
79	2	MOV	R1,#data
7A	2	MOV	R2,#data
7B	2	MOV	R3,#data
7C	2	MOV	R4,#data
7D	2	MOV	R5,#data
7E	2	MOV	R6,#data
7F	2	MOV	R7,#data
80	2	SJMP	code addr
81	2	AJMP	code addr
82	2	ANL	C,bit addr
83	1	MOVC	A,@A+PC
84	1	DIV	AB
85	3	MOV	data addr, data addr
86	2	MOV	data addr,@R0
87	2	MOV	data addr,@R1
88	2	MOV	data addr,R0
89	2	MOV	data addr,R1
8A	2	MOV	data addr,R2
8B	2	MOV	data addr,R3
8C	2	MOV	data addr,R4
8D	2	MOV	data addr,R5
8E	2	MOV	data addr,R6
8F	2	MOV	data addr,R7
90	3	MOV	DPTR,#data
91	2	ACALL	code addr
92	2	MOV	bit addr,C
93	1	MOVC	A,@A+DPTR
94	2	SUBB	A,#data
95	2	SUBB	A,data addr
96	1	SUBB	A,@R0
97	1	SUBB	A,@R1
98	1	SUBB	A,R0
99	1	SUBB	A,R1
9A	1	SUBB	A,R2
9B	1	SUBB	A,R3
9C	1	SUBB	A,R4
9D	1	SUBB	A,R5
9E	1	SUBB	A,R6
9F	1	SUBB	A,R7
A0	2	ORL	C,/bit addr
A1	2	AJMP	code addr
A2	2	MOV	C,bit addr
A3	1	INC	DPTR
A4	1	MUL	AB
A5		reserved	
A6	2	MOV	@R0, data addr
A7	2	MOV	@R1,data addr
A8	2	MOV	R0,data addr
A9	2	MOV	R1,data addr
AA	2	MOV	R2,data addr
AB	2	MOV	R3,data addr
AC	2	MOV	R4,data addr
AD	2	MOV	R5,data addr
AE	2	MOV	R6,data addr
AF	2	MOV	R7,data addr
B0	2	ANL	C,/bit addr
B1	2	ACALL	code addr
B2	2	CPL	bit addr
B3	1	CPL	C
B4	3	CJNE	A,#data,code addr
B5	3	CJNE	A,data addr,code addr
B6	3	CJNE	@R0,#data,code addr
B7	3	CJNE	@R1,#data,code addr
B8	3	CJNE	R0,#data,code addr
B9	3	CJNE	R1,#data,code addr
BA	3	CJNE	R2,#data,code addr
BB	3	CJNE	R3,#data,code addr
BC	3	CJNE	R4,#data,code addr
BD	3	CJNE	R5,#data,code addr
BE	3	CJNE	R6,#data,code addr
BF	3	CJNE	R7,#data,code addr
C0	2	PUSH	data addr
C1	2	AJMP	code addr
C2	2	CLR	bit addr
C3	1	CLR	C
C4	1	SWAP	A
C5	2	XCH	A,data addr
C6	1	XCH	A,@R0
C7	1	XCH	A,@R1
C8	1	XCH	A,R0
C9	1	XCH	A,R1
CA	1	XCH	A,R2
CB	1	XCH	A,R3

Table 2. (Cont.)

Hex Code	Number of Bytes	Mnemonic	Operands
CC	1	XCH	A,R4
CD	1	XCH	A,R5
CE	1	XCH	A,R6
CF	1	XCH	A,R7
D0	2	POP	data addr
D1	2	ACALL	code addr
D2	2	SETB	bit addr
D3	1	SETB	C
D4	1	DA	A
D5	3	DJNZ	data addr,code addr
D6	1	XCHD	A,@R0
D7	1	XCHD	A,@R1
D8	2	DJNZ	R0,code addr
D9	2	DJNZ	R1,code addr
DA	2	DJNZ	R2,code addr
DB	2	DJNZ	R3,code addr
DC	2	DJNZ	R4,code addr
DD	2	DJNZ	R5,code addr
DE	2	DJNZ	R6,code addr
DF	2	DJNZ	R7,code addr
E0	1	MOVX	A,@DPTR
E1	2	AJMP	code addr
E2	1	MOVX	A,@R0
E3	1	MOVX	A,@R1
E4	1	CLR	A
E5	2	MOV	A,data addr
E6	1	MOV	A,@R0
E7	1	MOV	A,@R1
E8	1	MOV	A,R0
E9	1	MOV	A,R1
EA	1	MOV	A,R2
EB	1	MOV	A,R3
EC	1	MOV	A,R4
ED	1	MOV	A,R5
EE	1	MOV	A,R6
EF	1	MOV	A,R7
F0	1	MOVX	@DPTR,A
F1	2	ACALL	code addr
F2	1	MOVX	@R0,A
F3	1	MOVX	@R1,A
F4	1	CPL	A
F5	2	MOV	data addr,A
F6	1	MOV	@R0,A
F7	1	MOV	@R1,A
F8	1	MOV	R0,A
F9	1	MOV	R1,A
FA	1	MOV	R2,A
FB	1	MOV	R3,A
FC	1	MOV	R4,A
FD	1	MOV	R5,A
FE	1	MOV	R6,A
FF	1	MOV	R7,A

8751H
8751H-11/8751H-10/8751H-8
SINGLE-COMPONENT 8-BIT MICROCOMPUTERS

- **8751H: 12 MHz Operation**
- **8751H-11: 11.2 MHz Operation**
- **8751H-10: 10 MHz Operation**
- **8751H-8: 8 MHz Operation**

- **Program Memory Security**
- **4K × 8 EPROM**
- **128 × 8 RAM**
- **32 I/O Lines (Four 8-Bit Ports)**
- **Two 16-Bit Timer/Counters**
- **Programmable Full-Duplex Serial Channel**
- **128K Accessible External Memory**
- **Boolean Processor**
- **4 μs Multiply and Divide**
- **218 User Bit-Addressable Locations**

The 8751H is a pin compatible EPROM version of the 8051AH. Intel's advanced +5 volt, depletion load, N-channel, HMOS technology allows the 8751H to remain fully compatible with its 8751/8751-8 predecessor in addition to incorporating a new program memory security feature. This allows the 8751H to be a full-speed MCS®-51 prototyping tool and provides for an effective single component solution for highly sensitive controller applications requiring code modification flexibility.

Specifically, the 8751H features: 4K byte program memory space; 32 I/O lines; two 16-bit timer/event counters; a 5-source; 2-level interrupt structure; a full duplex serial channel; a Boolean processor; and on-chip oscillator and clock circuitry. Standard TTL and most byte-oriented MCS-80 and MCS-85 peripherals can be used for I/O and memory expansion.

The 8751H is available in a hermetically sealed, ceramic, 40-lead dual in-line package which includes a window that allows for EPROM erasure when exposed to ultraviolet light (see Erasure Characteristics). During normal operation, ambient light may adversely affect the functionality of the chip. Therefore, applications which expose the 8751H to ambient light may require an opaque label over the window.

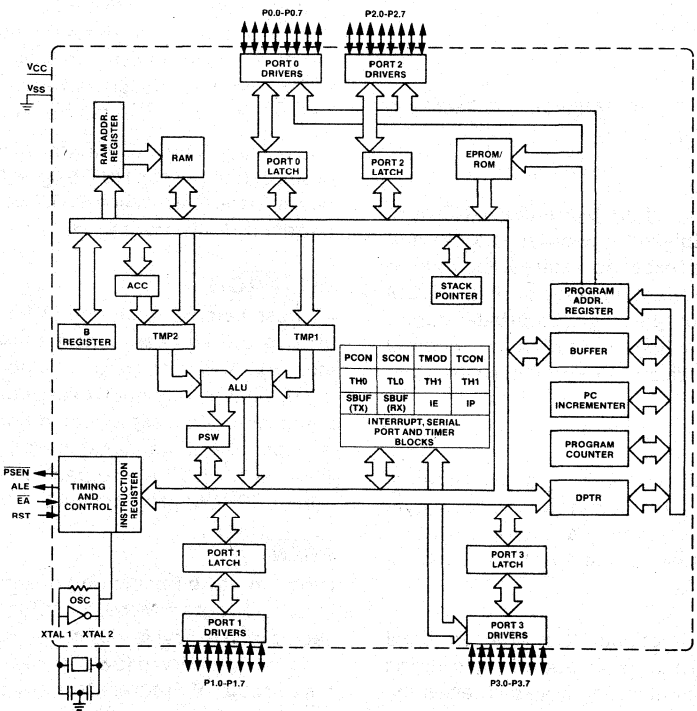


Figure 1. Block Diagram

May 1983
Order Number 230679-001

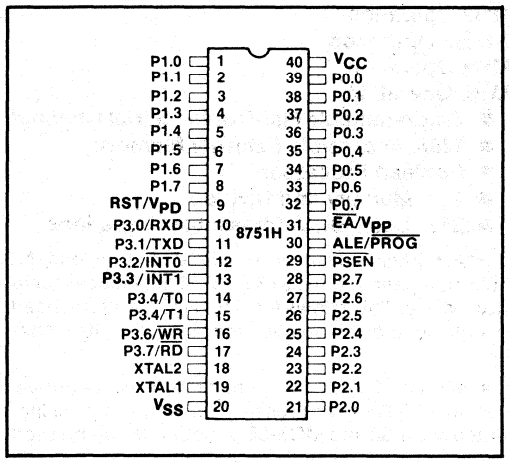


Figure 2. Pin Configuration

8751H PIN DESCRIPTIONS

V_{SS}
Circuit ground potential.

V_{CC}
Supply voltage during programming, verification, and normal operation.

Port 0
Port 0 is an 8-bit open drain bidirectional I/O port. It is also the multiplexed low-order address and data bus during accesses to external memory. It also receives the instruction bytes during EPROM programming, and outputs instruction bytes during program verification. (External pullups are required during program verification.) Port 0 can sink eight LS TTL inputs.

Port 1
Port 1 is an 8-bit bidirectional I/O port with internal pullups. It receives the low-order address byte during EPROM programming and program verification. Port 1 can sink/source four LS TTL inputs.

Port 2
Port 2 is an 8-bit bidirectional I/O port with internal pullups. It emits the high-order address byte during accesses to external memory. It also receives the high-order address bits during EPROM programming and program verification. Port 2 can sink/source four LS TTL inputs.

Port 3
Port 3 is an 8-bit bidirectional I/O port with internal pullups. It also serves the functions of various special features of the MCS®-51 Family, as listed below:

Port Pin	Alternate Function
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	$\overline{INT0}$ (external interrupt)
P3.3	$\overline{INT1}$ (external interrupt)
P3.4	T0 (Timer/counter 0 external input)
P3.5	T1 (Timer/counter 1 external input)
P3.6	$\overline{WR}$ (external Data Memory write strobe)
P3.7	$\overline{RD}$ (external Data Memory read strobe)

Port 3 can sink/source four LS TTL inputs.

RST/V_{PD}
A high on this pin for two machine cycles while the oscillator is running resets the device. A small external pulldown resistor (≈8.2kΩ) from RST/V_{PD} to V_{SS} permits power-on reset when a capacitor (≈10 μf) is also connected from this pin to V_{CC}.

If RST/V_{PD} is held within the V_{PD} spec while V_{CC} drops below its spec RST/V_{PD} will provide standby power to the RAM. When RST/V_{PD} is low, the RAMs current is drawn from V_{CC}.

ALE/$\overline{PROG}$
Address Latch Enable output for latching the low byte of the address during accesses to external memory. ALE is activated at a constant rate of 1/6 the oscillator frequency except during an external data memory access at which time one ALE pulse is skipped. ALE can sink/source 8 LS TTL inputs. This pin is also the program pulse input ($\overline{PROG}$) during EPROM programming.

$\overline{PSEN}$
Program Store Enable output is the read strobe to external Program Memory. $\overline{PSEN}$ is activated twice each machine cycle during fetches from external Program Memory. (However, even when executing out of external Program Memory two activations of $\overline{PSEN}$ are skipped during each access to external

Data Memory.) $\overline{PSEN}$ is not activated during fetches from internal Program Memory. $\overline{PSEN}$ can sink/source 8 LS TTL inputs.

$\overline{EA}$/VPP
When $\overline{EA}$ is held high, the 8751H executes out of internal Program Memory (unless the Program Counter exceeds 0FFFH). When $\overline{EA}$ is held low, the 8751H executes only out of external Program Memory. This pin also receives the 21V programming supply voltage (VPP) during EPROM programming. This pin should not be floated during normal operation.

XTAL1
Input to the inverting amplifier that forms the oscillator. XTAL1 should be grounded when an external oscillator is used.

XTAL2
Output of the inverting amplifier that forms the oscillator, and input to the internal clock generator. Receives the external oscillator signal when an external oscillator is used.

This diagram indicates when signals are clocked internally. The time it takes the signals to propagate to the pins, however, ranges from 25 to 125 ns. This propagation delay is dependent on variables such as temperature and pin loading. Propagation also varies from output to output and component to component. Typically though, (T_A = 25°C, fully loaded) $\overline{RD}$ and $\overline{WR}$ propagation delays are approximately 50 ns. The other signals are typically 85 ns. Propagation delays are incorporated in the AC specifications.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias . . . 0°C to 70°C
Storage Temperature . . . . . -65°C to +150°C
Voltage On Any Pin to V_{SS}
(Except V_{PP}) . . . . . -0.5V to +7V
Voltage from V_{PP} to V_{SS} 21.5V
Power Dissipation . . . . . 2 W

NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS: (T_A = 0° C to 70° C; V_{CC} = 4.5V to 5.5V, V_{SS} = 0V)

Symbol	Parameter	Min	Max	Unit	Test Conditions
VIL	Input Low Voltage	-0.5	0.8	V	
VIH	Input High Voltage (Except XTAL2, RST)	2.0	V_{CC} + 0.5	V	
VIH1	Input High Voltage to XTAL2, RST	2.5	V_{CC} + 0.5	V	XTAL1 = V_{SS}
VPD	Power Down Voltage to RST/V_{PD}	4.5	5.5	V	V_{CC} = 0V
VOL	Output Low Voltage Ports 1, 2, 3 (Note 1)		0.45	V	IOL = 1.6mA
VOL1	Output Low Voltage Port 0, ALE, $\overline{PSEN}$ (Note 1)		0.60 0.45	V V	IOL = 3.2mA IOL = 2.4mA
VOH	Output High Voltage Ports 1, 2, 3	2.4		V	IOH = -80μA
VOH1	Output High Voltage Port 0 (in External Bus Mode), ALE, $\overline{PSEN}$	2.4		V	IOH = -400μA
IIL	Logical 0 Input Current P1, P2, P3		500	μA	Vin = 0.45V
IIL1	Logical 0 Input Current to $\overline{EA}$/V_{PP}		-15	mA	Vin = 0.45V
IIL2	Logical 0 Input Current to XTAL2		-2.5	mA	XTAL1 = V_{SS} Vin = 0.45V
ILI	Input Leakage Current to Port 0		100	μA	0.45 < Vin < V_{CC}
IIH	Logical Input Current to $\overline{EA}$/V_{PP}		500	μA	Vin = 2.4V
IIH1	Input Current to RST/V_{PD} to activate reset		500	μA	Vin < (V_{CC} - 1.5V)
ICC	Power Supply Current		250	mA	All outputs disconnected, $\overline{EA}$ = V_{CC}
IPD	Power Down Current to RST/V_{PD}		10	mA	V_{CC} = 0V V_{PD} = 5V
CIO	Capacitance of I/O Buffers		10	pF	fc = 1 MHz T_A = 25°C

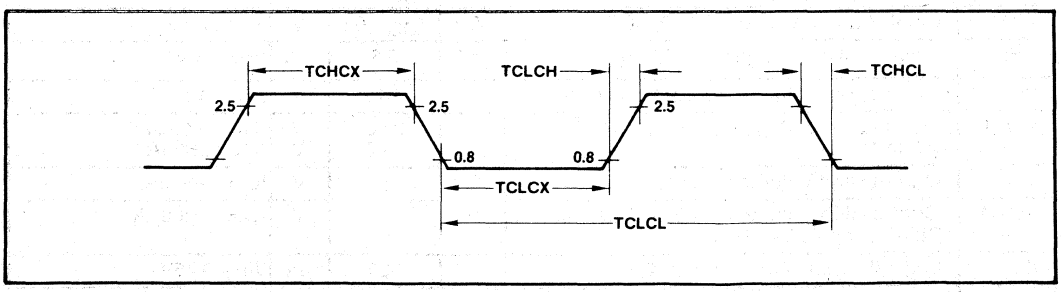
See page 4 for Notes.

NOTE: VOL is degraded when the 8751H rapidly discharges external capacitance. This AC noise is most pronounced during emission of address data. When using external memory, locate the latch or buffer as close to the 8751H as possible.

Datum	Emitting Ports	Degraded I/O Lines	VOL (peak) (max)
Address	P2, P0	P1, P3	0.8V
Write Data	P0	P1, P3, ALE	0.8V

EXTERNAL CLOCK DRIVE CHARACTERISTICS (XTAL2)

Symbol	Parameter	Min	Max	Units
TCLCL	Oscillator Period: 8751H	83.3	833.3	ns
TCHCX	High Time	20		ns
TCLCX	Low Time	20		ns
TCLCH	Rise Time		20	ns
TCHCL	Fall Time		20	ns



AC CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 4.5V to 5.5V, V_{SS} = 0V, Load Capacitance for Port 0, ALE, and $\overline{PSEN}$ = 100 pF; Load Capacitance for all other outputs = 80 pF)

EXTERNAL PROGRAM MEMORY CHARACTERISTICS

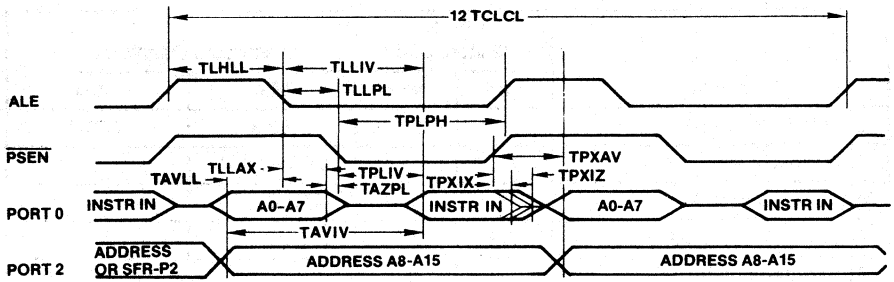
Symbol	Parameter	8751H 12 MHz Osc		Variable Oscillator		Units
		Min	Max	Min	Max	
TCLCL	Oscillator Period			83.3	833.3	ns
TLHLL	ALE Pulse Width	127		2TCLCL-40		ns
TAVLL	Address Valid to ALE	53		TCLCL-30		ns
TLLAX	Address Hold after ALE	48		TCLCL-35		ns
TLLIV	ALE to Valid Instr In		183		4TCLCL-150	ns
TLLPL	ALE to $\overline{PSEN}$	58		TCLCL-25		ns
TPLPH	$\overline{PSEN}$ Pulse Width	190		3TCLCL-60		ns
TPLIV	$\overline{PSEN}$ to Valid Instr In		100		3TCLCL-150	ns
TPXIX	Input Instr Hold after $\overline{PSEN}$	0		0		ns
TPXIZ	Input Instr Float after $\overline{PSEN}$		63		TCLCL-20	ns
TPXAV	$\overline{PSEN}$ to Address Valid	75		TCLCL-8		ns
TAVIV	Address to Valid Instr In		267		5TCLCL-150	ns
TAZPL	Address Float to $\overline{PSEN}$	0		0		ns

EXTERNAL DATA MEMORY CHARACTERISTICS

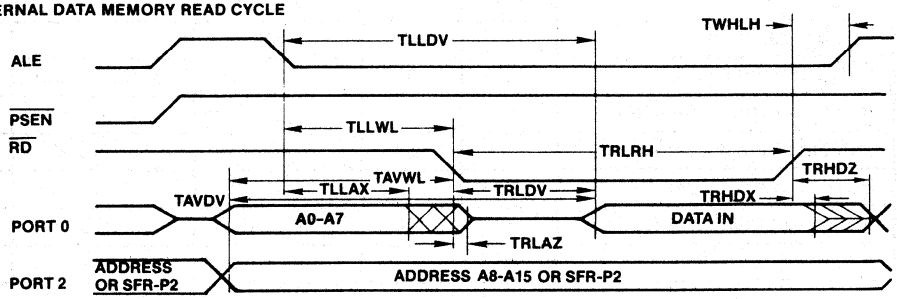
Symbol	Parameter	8751H 10 MHz Osc		Variable Oscillator		Units
		Min	Max	Min	Max	
TRLRH	$\overline{RD}$ Pulse Width	400		6TCLCL-100		ns
TWLWH	$\overline{WR}$ Pulse Width	400		6TCLCL-100		ns
TLLAX	Address Hold After ALE	48		TCLCL-35		ns
TRLDV	$\overline{RD}$ to Valid Data In		252		5TCLCL-165	ns
TRHDX	Data Hold after $\overline{RD}$	0		0		ns
TRHDZ	Data Float after $\overline{RD}$		97		2TCLCL-70	ns
TLLDV	ALE to Valid Data In		517		8TCLCL-150	ns
TAVDV	Address to Valid Data In		585		9TCLCL-165	ns
TLLWL	ALE to $\overline{WR}$ or $\overline{RD}$	200	300	3TCLCL-50	3TCLCL+50	ns
TAVWL	Address to $\overline{WR}$ or $\overline{RD}$	203		4TCLCL-130		ns
TQVWX	Data Valid to $\overline{WR}$ Transition	13		TCLCL-70		ns
TQVWH	Data Setup to $\overline{WR}$ High	433		7TCLCL-150		ns
TWHQX	Data Held after $\overline{WR}$	33		TCLCL-50		ns
TRLAZ	$\overline{RD}$ Low to Address Float		0		0	ns
TWHLH	$\overline{RD}$ or WR High to ALE High	33	133	TCLCL-50	TCLCL+50	ns

AC TIMING DIAGRAMS

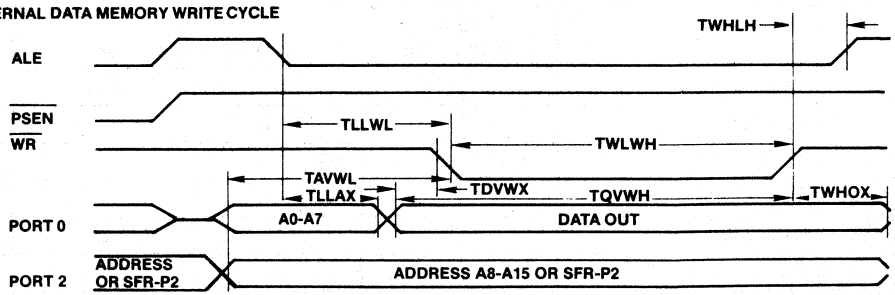
EXTERNAL PROGRAM MEMORY READ CYCLE



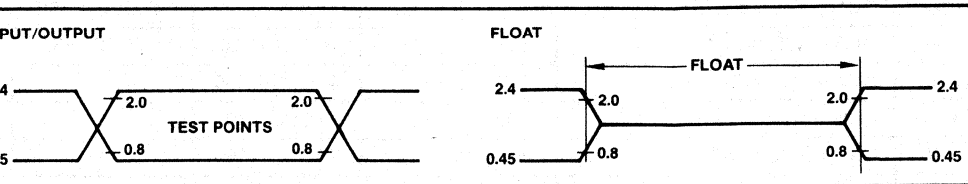
EXTERNAL DATA MEMORY READ CYCLE



EXTERNAL DATA MEMORY WRITE CYCLE

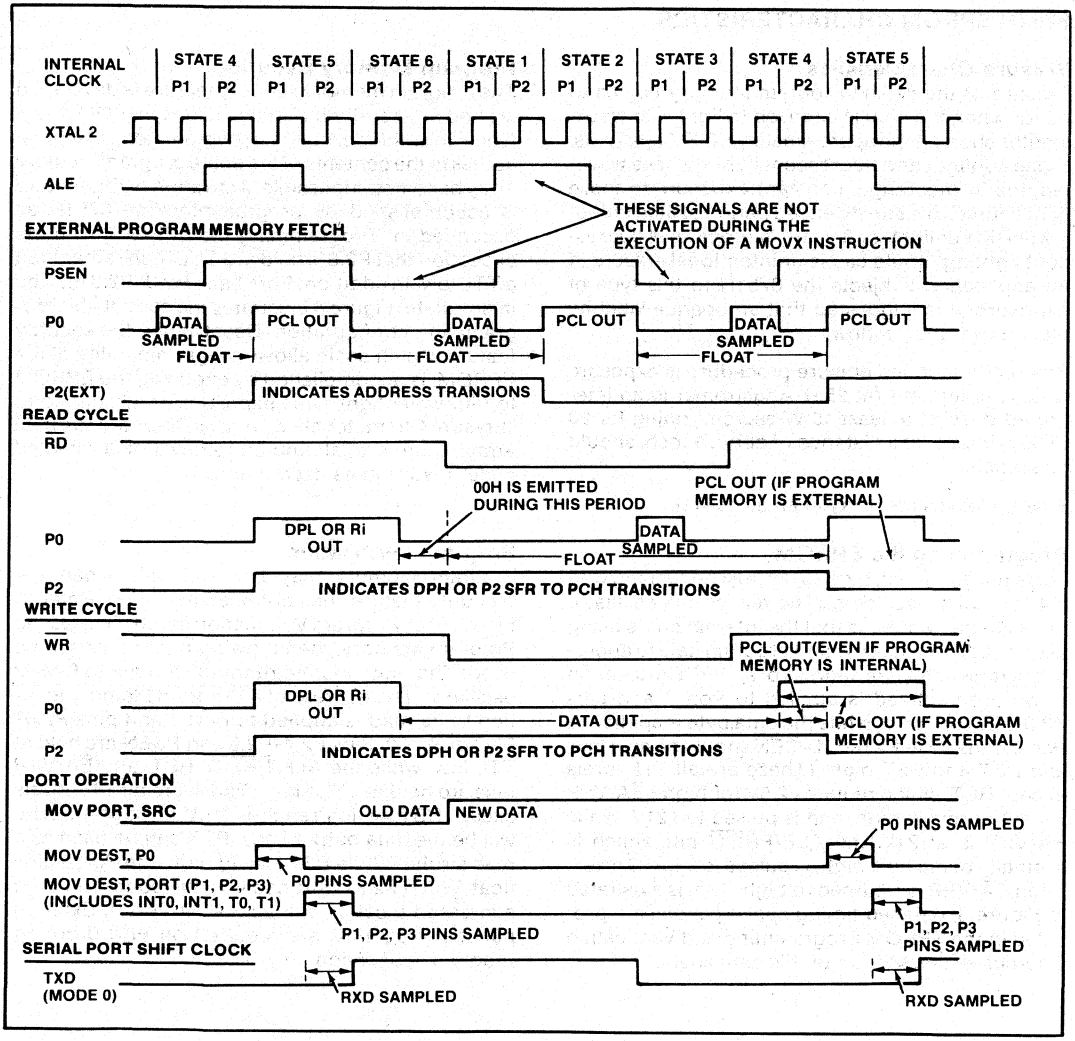


AC TESTING INPUT/OUTPUT, FLOAT WAVEFORMS



AC inputs during testing are driven at 2.4V for a logic "1" and 0.45V for a logic "0". Timing measurements are made at 2.0V for a logic "1" and 0.8V for a logic "0". For timing purposes, the float state is defined as the point at which a P0 pin sinks 2.4mA or sources 400 μA at the voltage test levels.

CLOCK WAVEFORMS



This diagram indicates when signals are clocked internally. The time it takes the signals to propagate to the pins, however, ranges from 25 to 125 ns. This propagation delay is dependent on variables such as temperature and pin loading. Propagation also varies from output to output and component to component. Typically though, (T_A = 25° C, fully loaded) RD and WR propagation delays are approximately 50 ns. The other signals are typically 85 ns. Propagation delays are incorporated in the AC specifications.

8751H EPROM CHARACTERISTICS

Erasure Characteristics

Erasure of the 8751H Program Memory begins to occur when the chip is exposed to light with wavelengths shorter than approximately 4,000 Ångstroms. Since sunlight and fluorescent lighting have wavelengths in this range, constant exposure to these light sources over an extended period of time (about 1 week in sunlight, or 3 years in room-level fluorescent lighting) could cause unintentional erasure. If an application subjects the 8751H to this type of exposure, it is suggested that an opaque label be placed over the window.

The recommended erasure procedure is exposure to ultraviolet light (at 2537 Ångstroms) to an integrated dose of at least 15 W-sec/cm^2 rating for 20 to 30 minutes, at a distance of about 1 inch, should be sufficient.

Erasure leaves the array in an all 1s state.

Programming the EPROM

To be programmed, the 8751H must be running with a 4 to 6 MHz oscillator. (The reason the oscillator needs to be running is that the internal bus is being used to transfer address and program data to appropriate registers.) The address of an EPROM location to be programmed is applied to Port 1 and pins P2.0–P2.3 of Port 2, while the data byte is applied to Port 0. Pins P2.4–P2.6 and $\overline{PSEN}$ should be held low, and P2.7 and RST high. (These are all TTL levels except RST, which requires 2.5V for high.) $\overline{EA}$/VPP is held normally high, and is pulsed to +21V. While $\overline{EA}$/VPP is at 21V, the ALE/$\overline{PROG}$ pin, which is normally being held high, is pulsed low for 50 msec. Then $\overline{EA}$/VPP is returned to high. This is illustrated in Figure 3. detailed timing specifications are provided in the EPROM Programming and Verification Characteristics section of this data sheet.

Program Memory Security

The program memory security feature is developed around a "security bit" in the 8751H EPROM array. Once this "hidden bit" is programmed, electrical access to the contents of the entire program memory array becomes impossible. Activation of this feature is accomplished by programming the 8751H as described in "Programming the EPROM" with the exception that P2.6 is held at a TTL high rather than a TTL low. In addition, Port 1 and P2.0–P2.3 may be in any state. Figure 4 illustrates the security bit programming configuration. Deactivating the security feature, which again allows programmability of the EPROM, is accomplished by exposing the EPROM to ultraviolet light. This exposure, as described in "Erasure Characteristics," erases the entire EPROM array. Therefore, attempted retrieval of "protected code" results in its destruction.

Program Verification

Program Memory may be read only when the "security feature" has not been activated. Refer to Figure 5 for Program Verification setup. To read the Program Memory, the following procedure can be used. The unit must be running with a 4 to 6 MHz oscillator. The address of a Program Memory location to be read is applied to Port 1 and pins P2.0–P2.3 of Port 2. Pins P2.4–P2.6 and $\overline{PSEN}$ are held at TTL low, while the ALE/$\overline{PROG}$, RST, and $\overline{EA}$/VPP pins are held at TTL high. (These are all TTL levels except RST, which requires 2.5V for high.) Port 0 will be the data output lines. P2.7 can be used as a read strobe. While P2.7 is held high, the Port 0 pins float. When P2.7 is strobed low, the contents of the addressed location will appear at Port 0. External pullups (e.g., 10K) are required on Port 0 during program verification.

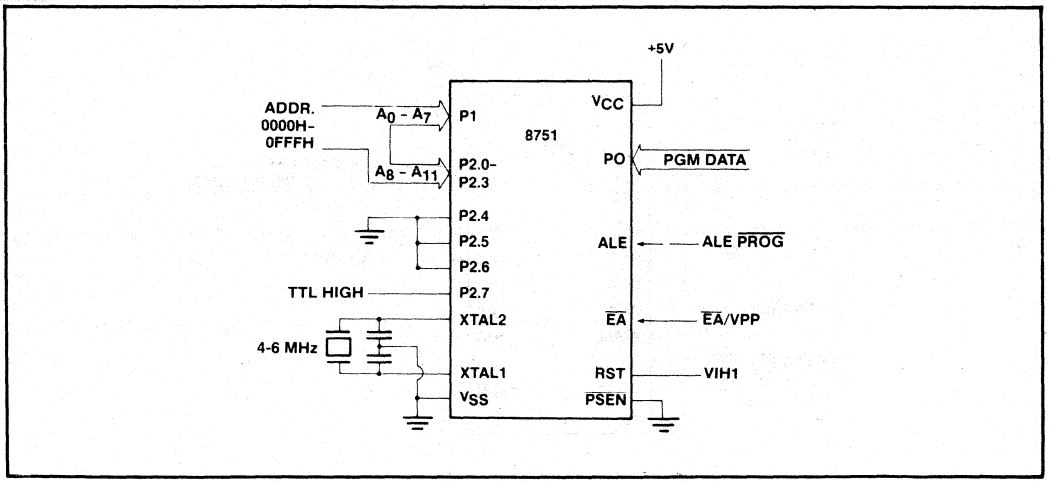


Figure 3. Programming Configuration

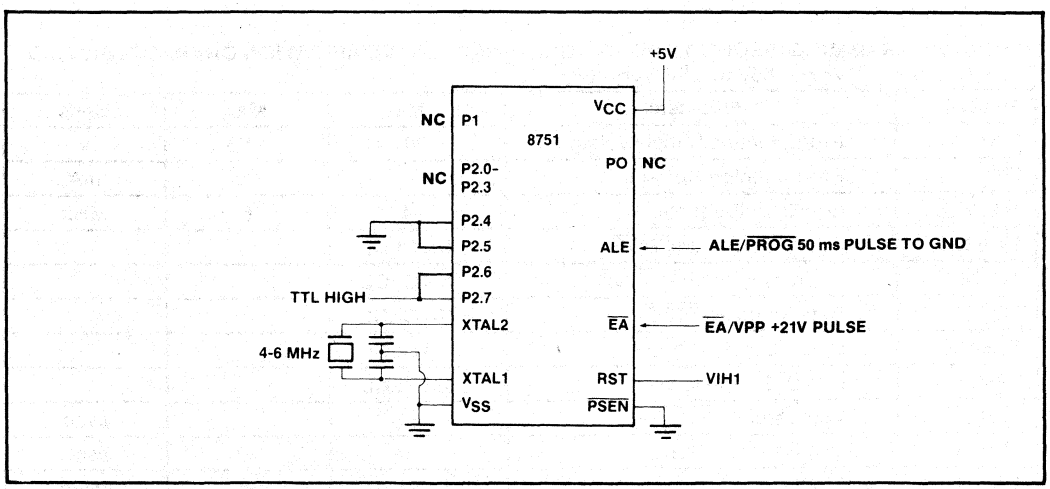


Figure 4. Security Bit Programming Configuration

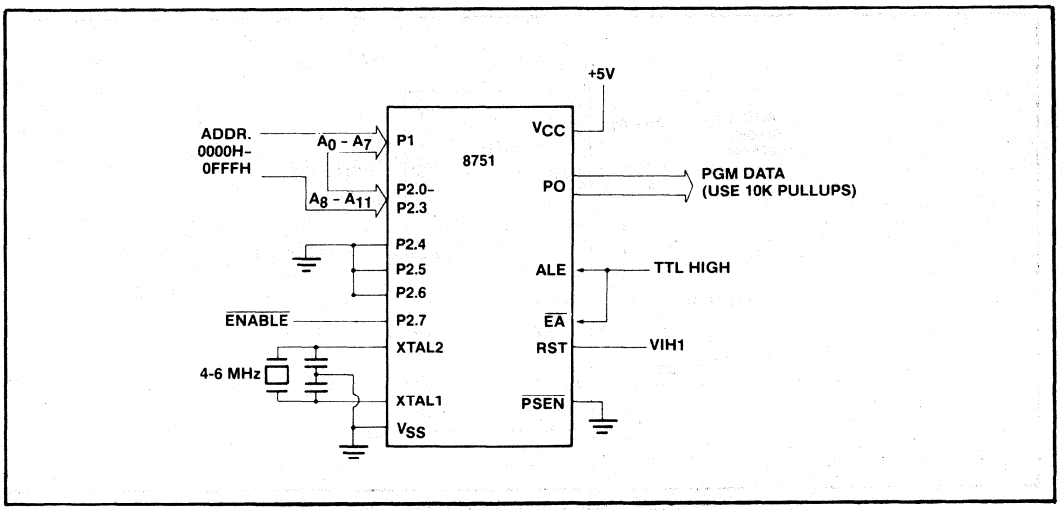
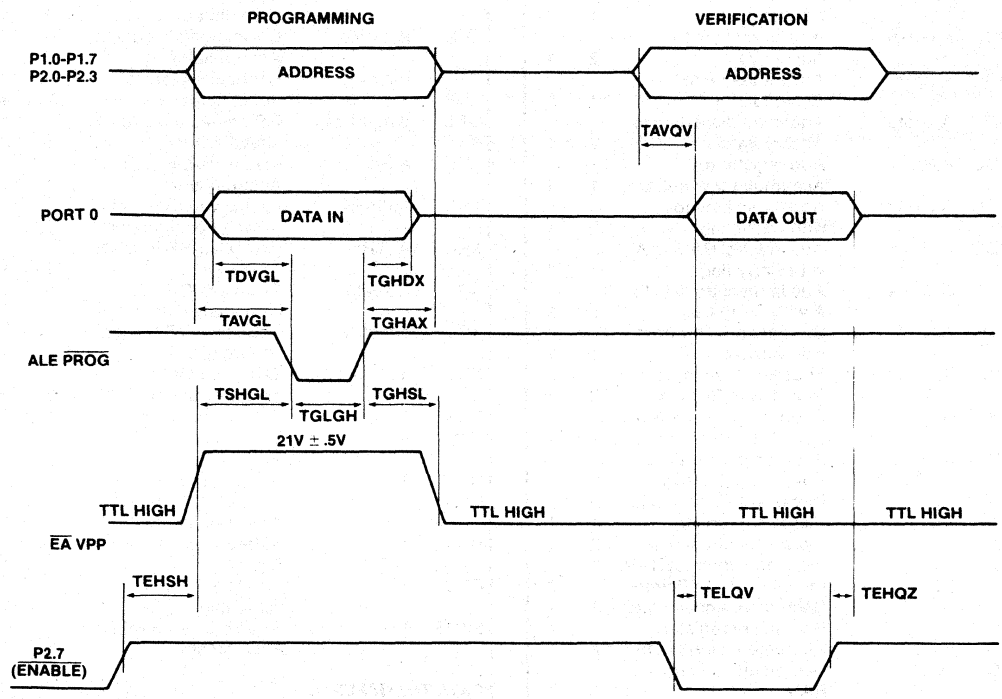


Figure 5. Program Verification Configuration

EPROM PROGRAMMING, SECURITY BIT PROGRAMMING AND VERIFICATION CHARACTERISTICS
(TA = 21°C to 27°C, V_{CC} = 4.5V to 5.5V, V_{SS} = 0V)

Symbol	Parameter	Min	Max	Units
V_{PP}	Programming Supply Voltage	20.5	21.5	V
IPP	Programming Current		30	mA
1/TCLCL	Oscillator Frequency	4	6	MHz
TAVGL	Address Setup to $\overline{PROG}$	48TCLCL		
TGHAX	Address Hold after $\overline{PROG}$	48TCLCL		
TDVGL	Data Setup to $\overline{PROG}$	48TCLCL		
TGHDX	Data Hold after $\overline{PROG}$	48TCLCL		
TEHSH	$\overline{ENABLE}$ High to V_{PP}	48TCLCL		
TSHGL	V_{PP} Setup to $\overline{PROG}$	10		µsec
TGHSL	V_{PP} Hold after $\overline{PROG}$	10		µsec
TGLGH	$\overline{PROG}$ Width	45	55	msec
TAVQV	Address to Data Valid		48TCLCL	
TELQV	$\overline{ENABLE}$ to Data Valid		48TCLCL	
TEHQZ	Data Float after $\overline{ENABLE}$	0	48TCLCL	

EPROM PROGRAMMING, SECURITY BIT PROGRAMMING AND VERIFICATION WAVEFORMS



FOR PROGRAMMING CONDITIONS SEE FIGURE 3.
FOR SECURITY BIT PROGRAMMING CONDITIONS SEE FIGURE 4.
FOR VERIFICATION CONDITIONS SEE FIGURE 5.

Table 1. MCS®-51 Instruction Set Description

ARITHMETIC OPERATIONS

Mnemonic		Description	Byte	Cyc
ADD	A,Rn	Add register to Accumulator	1	1
ADD	A,direct	Add direct byte to Accumulator	2	1
ADD	A,@Ri	Add indirect RAM to Accumulator	1	1
ADD	A,#data	Add immediate data to Accumulator	2	1
ADDC	A,Rn	Add register to Accumulator with Carry	1	1
ADDC	A,direct	Add direct byte to A with Carry flag	2	1
ADDC	A,@Ri	Add indirect RAM to A with Carry flag	1	1
ADDC	A,#data	Add immediate data to A with Carry flag	2	1
SUBB	A,Rn	Subtract register from A with Borrow	1	1
SUBB	A,direct	Subtract direct byte from A with Borrow	2	1
SUBB	A,@Ri	Subtract indirect RAM from A with Borrow	1	1
SUBB	A,#data	Subtract immed data from A with Borrow	2	1
INC	A	Increment Accumulator	1	1
INC	Rn	Increment register	1	1
INC	direct	Increment direct byte	2	1
INC	@Ri	Increment indirect RAM	1	1
INC	DPTR	Increment Data Pointer	1	2
DEC	A	Decrement Accumulator	1	1
DEC	Rn	Decrement register	1	1
DEC	direct	Decrement direct byte	2	1
DEC	@Ri	Decrement indirect RAM	1	1
MUL	AB	Multiply A & B	1	4
DIV	AB	Divide A by B	1	4
DA	A	Decimal Adjust Accumulator	1	1

LOGICAL OPERATIONS

Mnemonic		Destination	Byte	Cyc
ANL	A,Rn	AND register to Accumulator	1	1
ANL	A,direct	AND direct byte to Accumulator	2	1
ANL	A,@Ri	AND indirect RAM to Accumulator	1	1
ANL	A,#data	AND immediate data to Accumulator	2	1
ANL	direct,A	AND Accumulator to direct byte	2	1
ANL	direct,#data	AND immediate data to direct byte	3	2
ORL	A,Rn	OR register to Accumulator	1	1
ORL	A,direct	OR direct byte to Accumulator	2	1

LOGICAL OPERATIONS (CONTINUED)

Mnemonic		Destination	Byte	Cyc
ORL	A,@Ri	OR indirect RAM to Accumulator	1	1
ORL	A,#data	OR immediate data to Accumulator	2	1
ORL	direct,A	OR Accumulator to direct byte	2	1
ORL	direct,#data	OR immediate data to direct byte	3	2
XRL	A,Rn	Exclusive-OR register to Accumulator	1	1
XRL	A,direct	Exclusive-OR direct byte to Accumulator	2	1
XRL	A,@Ri	Exclusive-OR indirect RAM to A	1	1
XRL	A,#data	Exclusive-OR immediate data to A	2	1
XRL	direct,A	Exclusive-OR Accumulator to direct byte	2	1
XRL	direct,#data	Exclusive-OR immediate data to direct	3	2
CLR	A	Clear Accumulator	1	1
CPL	A	Complement Accumulator	1	1
RL	A	Rotate Accumulator Left	1	1
RLC	A	Rotate A Left through the Carry flag	1	1
RR	A	Rotate Accumulator Right	1	1
RRC	A	Rotate A Right through Carry flag	1	1
SWAP	A	Swap nibbles within the Accumulator	1	1

DATA TRANSFER

Mnemonic		Description	Byte	Cyc
MOV	A,Rn	Move register to Accumulator	1	1
MOV	A,direct	Move direct byte to Accumulator	2	1
MOV	A,@Ri	Move indirect RAM to Accumulator	1	1
MOV	A,#data	Mov immediate data to Accumulator	2	1
MOV	Rn,A	Move Accumulator to register	1	1
MOV	Rn,direct	Move direct byte to register	2	2
MOV	Rn,#data	Move immediate data to register	2	1
MOV	direct,A	Move Accumulator to direct byte	2	1
MOV	direct,Rn	Move register to direct byte	2	2
MOV	direct,direct	Move direct byte to direct	3	2
MOV	direct,@Ri	Move indirect RAM to direct byte	2	2

Table 1. (Cont.)

DATA TRANSFER (CONTINUED)

Mnemonic		Description	Byte	Cyc
MOV	direct,#data	Move immediate data to direct byte	3	2
MOV	@Ri,A	Move Accumulator to indirect RAM	1	1
MOV	@Ri,direct	Move direct byte to indirect RAM	2	2
MOV	@Ri,#data	Move immediate data to indirect RAM	2	1
MOV	DPTR,#data16	Load Data Pointer with a 16-bit constant	3	2
MOVC	A,@A+DPTR	Move Code byte relative to DPTR to A	1	2
MOVC	A,@A+PC	Move Code byte relative to PC to A	1	2
MOVX	A,@Ri	Move External RAM (8-bit addr) to A	1	2
MOVX	A,@DPTR	Move External RAM (16-bit addr) to A	1	2
MOVX	@Ri,A	Move A to External RAM (8-bit addr)	1	2
MOVX	@DPTR,A	Move A to External RAM (16-bit addr)	1	2
PUSH	direct	Push direct byte onto stack	2	2
POP	direct	Pop direct byte from stack	2	2
XCH	A,Rn	Exchange register with Accumulator	1	1
XCH	A,direct	Exchange direct byte with Accumulator	2	1
XCH	A,@Ri	Exchange indirect RAM with A	1	1
XCHD	A,@Ri	Exchange low-order Digit ind RAM w A	1	1

BOOLEAN VARIABLE MANIPULATION

Mnemonic		Description	Byte	Cyc
CLR	C	Clear Carry flag	1	1
CLR	bit	Clear direct bit	2	1
SETB	C	Set Carry flag	1	1
SETB	bit	Set direct Bit	2	1
CPL	C	Complement Carry flag	1	1
CPL	bit	Complement direct bit	2	1
ANL	C,bit	AND direct bit to Carry flag	2	2
ANL	C,/bit	AND complement of direct bit to Carry	2	2
ORL	C/bit	OR direct bit to Carry flag	2	2
ORL	C,/bit	OR complement of direct bit to Carry	2	2
MOV	C,/bit	Move direct bit to Carry flag	2	1
MOV	bit,C	Move Carry flag to direct bit	2	2

PROGRAM AND MACHINE CONTROL

Mnemonic		Description	Byte	Cyc
ACALL	addr11	Absolute Subroutine Call	2	2
LCALL	addr16	Long Subroutine Call	3	2
RET		Return from subroutine	1	2
RETI		Return from interrupt	1	2
AJMP	addr11	Absolute Jump	2	2
LJMP	addr16	Long Jump	3	2
SJMP	rel	Short Jump (relative addr)	2	2
JMP	@A+DPTR	Jump indirect relative to the DPTR	1	2
JZ	rel	Jump if Accumulator is Zero	2	2
JNZ	rel	Jump if Accumulator is Not Zero	2	2
JC	rel	Jump if Carry flag is set	2	2
JNC	rel	Jump if No Carry flag	2	2
JB	bit,rel	Jump if direct Bit set	3	2
JNB	bit,rel	Jump if direct Bit Not set	3	2
JBC	bit,rel	Jump if direct Bit is set & Clear bit	3	2
CJNE	A,direct,rel	Compare direct to A & Jump if Not Equal	3	2
CJNE	A,#data,rel	Comp, immed, to A & Jump if Not Equal	3	2
CJNE	Rn,#data,rel	Comp, immed, to reg & Jump if Not Equal	3	2
CJNE	@Ri,#data,rel	Comp, immed, to ind, & Jump if Not Equal	3	2
DJNZ	Rn,rel	Decrement register & Jump if Not Zero	2	2
DJNZ	direct,rel	Decrement direct & Jump if Not Zero	3	2
NOP		No operation	1	1

Notes on data addressing modes:

Rn —Working register R0–R7
direct —128 internal RAM locations, any I/O port, control or status register
@Ri —Indirect internal RAM location addressed by register R0 or R1
#data —8-bit constant included in instruction
#data16 —16-bit constant included as bytes 2 & 3 of instruction
bit —128 software flags, any I/O pin, control or status bit

Notes on program addressing modes:

addr16 —Destination address for LCALL & LJMP may be anywhere within the 64-K program memory address space
Addr11 —Destination address for ACALL & AJMP will be within the same 2-K page of program memory as the first byte of the following instruction
rel —SJMP and all conditional jumps include an 8-bit offset byte, Range is +127–128 bytes relative to first byte of the following instruction

Table 2. Instruction Opcodes in Hexadecimal Order

Hex Code	Number of Bytes	Mnemonic	Operands
00	1	NOP	
01	2	AJMP	code addr
02	3	LJMP	code addr
03	1	RR	A
04	1	INC	A
05	2	INC	data addr
06	1	INC	@R0
07	1	INC	@R1
08	1	INC	R0
09	1	INC	R1
0A	1	INC	R2
0B	1	INC	R3
0C	1	INC	R4
0D	1	INC	R5
0E	1	INC	R6
0F	1	INC	R7
10	3	JBC	bit addr, code addr
11	2	ACALL	code addr
12	3	LCALL	code addr
13	1	RRC	A
14	1	DEC	A
15	2	DEC	data addr
16	1	DEC	@R0
17	1	DEC	@R1
18	1	DEC	R0
19	1	DEC	R1
1A	1	DEC	R2
1B	1	DEC	R3
1C	1	DEC	R4
1D	1	DEC	R5
1E	1	DEC	R6
1F	1	DEC	R7
20	3	JB	bit addr, code addr
21	2	AJMP	code addr
22	1	RET	
23	1	RL.	A
24	2	ADD	A,#data
25	2	ADD	A,data addr
26	1	ADD	A,@R0
27	1	ADD	A,@R1
28	1	ADD	A,R0
29	1	ADD	A,R1
2A	1	ADD	A,R2
2B	1	ADD	A,R3
2C	1	ADD	A,R4
2D	1	ADD	A,R5
2E	1	ADD	A,R6
2F	1	ADD	A,R7
30	3	JNB	bit addr, code addr
31	2	ACALL	code addr
32	1	RETI	

Hex Code	Number of Bytes	Mnemonic	Operands
33	1	RLC	A
34	2	ADDC	A,#data
35	2	ADDC	A,data addr
36	1	ADDC	A,@R0
37	1	ADDC	A,@R1
38	1	ADDC	A,R0
39	1	ADDC	A,R1
3A	1	ADDC	A,R2
3B	1	ADDC	A,R3
3C	1	ADDC	A,R4
3D	1	ADDC	A,R5
3E	1	ADDC	A,R6
3F	1	ADDC	A,R7
40	2	JC	code addr
41	2	AJMP	code addr
42	2	ORL	data addr,A
43	3	ORL	data addr,#data
44	2	ORL	A,#data
45	2	ORL	A,data addr
46	1	ORL	A,@R0
47	1	ORL	A,@R1
48	1	ORL	A,R0
49	1	ORL	A,R1
4A	1	ORL	A,R2
4B	1	ORL	A,R3
4C	1	ORL	A,R4
4D	1	ORL	A,R5
4E	1	ORL	A,R6
4F	1	ORL	A,R7
50	2	JNC	code addr
51	2	ACALL	code addr
52	2	ANL	data addr,A
53	3	ANL	data addr,#data
54	2	ANL	A,#data
55	2	ANL	A,data addr
56	1	ANL	A,@R0
57	1	ANL	A@R1
58	1	ANL	A,R0
59	1	ANL	A,R1
5A	1	ANL	A,R2
5B	1	ANL	A,R3
5C	1	ANL	A,R4
5D	1	ANL	A,R5
5E	1	ANL	A,R6
5F	1	ANL	A,R7
60	2	JZ	code addr
61	2	AJMP	code addr
62	2	XRL	data addr.A
63	3	XRL	data addr,#data
64	2	XRL	A,#data
65	2	XRL	A,data addr

Table 2. (Cont.)

Hex Code	Number of Bytes	Mnemonic	Operands
66	1	XRL	A,@R0
67	1	XRL	A,@R1
68	1	XRL	A,R0
69	1	XRL	A,R1
6A	1	XRL	A,R2
6B	1	XRL	A,R3
6C	1	XRL	A,R4
6D	1	XRL	A,R5
6E	1	XRL	A,R6
6F	1	XRL	A,R7
70	2	JNZ	code addr
71	2	ACALL	code addr
72	2	ORL	C,bit addr
73	1	JMP	@A+DPTR
74	2	MOV	A,#data
75	3	MOV	data addr,#data
76	2	MOV	@R0,#data
77	2	MOV	@R1,#data
78	2	MOV	R0,#data
79	2	MOV	R1,#data
7A	2	MOV	R2,#data
7B	2	MOV	R3,#data
7C	2	MOV	R4,#data
7D	2	MOV	R5,#data
7E	2	MOV	R6,#data
7F	2	MOV	R7,#data
80	2	SJMP	code addr
81	2	AJMP	code addr
82	2	ANL	C,bit addr
83	1	MOVC	A,@A+PC
84	1	DIV	AB
85	3	MOV	data addr, data addr
86	2	MOV	data addr,@R0
87	2	MOV	data addr,@R1
88	2	MOV	data addr,R0
89	2	MOV	data addr,R1
8A	2	MOV	data addr,R2
8B	2	MOV	data addr,R3
8C	2	MOV	data addr,R4
8D	2	MOV	data addr,R5
8E	2	MOV	data addr,R6
8F	2	MOV	data addr,R7
90	3	MOV	DPTR,#data
91	2	ACALL	code addr
92	2	MOV	bit addr,C
93	1	MOVC	A,@A+DPTR
94	2	SUBB	A,#data
95	2	SUBB	A,data addr
96	1	SUBB	A,@R0
97	1	SUBB	A,@R1
98	1	SUBB	A,R0

Hex Code	Number of Bytes	Mnemonic	Operands
99	1	SUBB	A,R1
9A	1	SUBB	A,R2
9B	1	SUBB	A,R3
9C	1	SUBB	A,R4
9D	1	SUBB	A,R5
9E	1	SUBB	A,R6
9F	1	SUBB	A,R7
A0	2	ORL	C,/bit addr
A1	2	AJMP	code addr
A2	2	MOV	C,bit addr
A3	1	INC	DPTR
A4	1	MUL	AB
A5		reserved	
A6	2	MOV	@R0,data adr
A7	2	MOV	@R1,data addr
A8	2	MOV	R0,data addr
A9	2	MOV	R1,data addr
AA	2	MOV	R2,data addr
AB	2	MOV	R3,data addr
AC	2	MOV	R4,data addr
AD	2	MOV	R5,data addr
AE	2	MOV	R6,data addr
AF	2	MOV	R7,data addr
B0	2	ANL	C,/bit addr
B1	2	ACALL	code addr
B2	2	CPL	bit addr
B3	1	CPL	C
B4	3	CJNE	A,#data,code addr
B5	3	CJNE	A,data addr,code addr
B6	3	CJNE	@R0,#data,code addr
B7	3	CJNE	@R1,#data,code addr
B8	3	CJNE	R0,#data,code addr
B9	3	CJNE	R1,#data,code addr
BA	3	CJNE	R2,#data,code addr
BB	3	CJNE	R3,#data,code addr
BC	3	CJNE	R4,#data,code addr
BD	3	CJNE	R5,#data,code addr
BE	3	CJNE	R6,#data,code addr
BF	3	CJNE	R7,#data,code addr
C0	2	PUSH	data addr
C1	2	AJMP	code addr
C2	2	CLR	bit addr
C3	1	CLR	C
C4	1	SWAP	A
C5	2	XCH	A,data addr
C6	1	XCH	A,@R0
C7	1	XCH	A,@R1
C8	1	XCH	A,R0
C9	1	XCH	A,R1
CA	1	XCH	A,R2
CB	1	XCH	A,R3

Table 2. (Cont.)

Hex Code	Number of Bytes	Mnemonic	Operands
CC	1	XCH	A,R4
CD	1	XCH	A,R5
CE	1	XCH	A,R6
CF	1	XCH	A,R7
D0	2	POP	data addr
D1	2	ACALL	code addr
D2	2	SETB	bit addr
D3	1	SETB	C
D4	1	DA	A
D5	3	DJNZ	data addr,code addr
D6	1	XCHD	A,@R0
D7	1	XCHD	A,@R1
D8	2	DJNZ	R0,code addr
D9	2	DJNZ	R1,code addr
DA	2	DJNZ	R2,code addr
DB	2	DJNZ	R3,code addr
DC	2	DJNZ	R4,code addr
DD	2	DJNZ	R5,code addr
DE	2	DJNZ	R6,code addr
DF	2	DJNZ	R7,code addr
E0	1	MOVX	A,@DPTR
E1	2	AJMP	code addr
E2	1	MOVX	A,@R0
E3	1	MOVX	A,@R1
E4	1	CLR	A
E5	2	MOV	A,data addr
E6	1	MOV	A,@R0
E7	1	MOV	A,@R1
E8	1	MOV	A,R0
E9	1	MOV	A,R1
EA	1	MOV	A,R2
EB	1	MOV	A,R3
EC	1	MOV	A,R4
ED	1	MOV	A,R5
EE	1	MOV	A,R6
EF	1	MOV	A,R7
F0	1	MOVX	@DPTR,A
F1	2	ACALL	code addr
F2	1	MOVX	@R0,A
F3	1	MOVX	@R1,A
F4	1	CPL	A
F5	2	MOV	data addr,A
F6	1	MOV	@R0,A
F7	1	MOV	@R1,A
F8	1	MOV	R0,A
F9	1	MOV	R1,A
FA	1	MOV	R2,A
FB	1	MOV	R3,A
FC	1	MOV	R4,A
FD	1	MOV	R5,A
FE	1	MOV	R6,A
FF	1	MOV	R7,A

PRELIMINARY

8031AH/8051AH
SINGLE-COMPONENT 8-BIT MICROCOMPUTER

- **8031AH — Control-Oriented CPU with RAM and I/O**
- **8051AH — An 8031AH with Factory Mask-Programmable ROM**
- **Fabricated with Intel's HMOS II Process**

- **4K × 8 ROM (8051AH only)**
- **128 × 8 RAM**
- **32 I/O Lines (Four 8-Bit Ports)**
- **Two 16-Bit Timer/Counters**
- **Programmable Full-Duplex Serial Channel**
- **128K Accessible External Memory**
- **Boolean Processor**
- **4 μs Multiply and Divide**
- **218 User Bit-Addressable Locations**
- **100 mA Typical Supply Current**

The 8031AH/8051AH is Intel's HMOS II version of the high performance 8-bit 8031/8051 microcomputer. While the 8031AH/8051AH features the same powerful architecture and instruction set as its HMOS I predecessor, it offers the additional benefit of lower power supply current.

The 8031AH/8051AH provides a cost-effective solution for those controller applications requiring up to 64K bytes of program and/or 64K bytes of data storage. Specifically, the 8031AH contains 128 bytes of read/write data memory; 32 I/O lines configured as four 8-bit parallel ports; two 16-bit timer/counters; a five-source, two-priority level, nested interrupt structure; a programmable serial I/O port; and an on-chip oscillator with clock circuitry. The 8051AH has all of these 8031AH features plus 4K bytes of nonvolatile read only program memory. Both microcomputers can use standard TTL compatible memories and most byte-oriented MCS®-80 and MCS-85 peripherals for additional I/O and memory capabilities.

The 8031AH/8051AH microcomputer, characteristic of the entire MCS-51 family, is efficient in both control and computational type applications. This results from extensive BCD/binary arithmetic and bit-handling facilities. The 8031AH/8051AH also makes efficient use of its program memory space with an instruction set consisting of 44% one-byte, 41% two-byte, and 15% three-byte instructions. At 12MHz CPU operation, over half of the instructions execute in just 1.0μs, while the longest instructions, multiply and divide, require only 4μs.

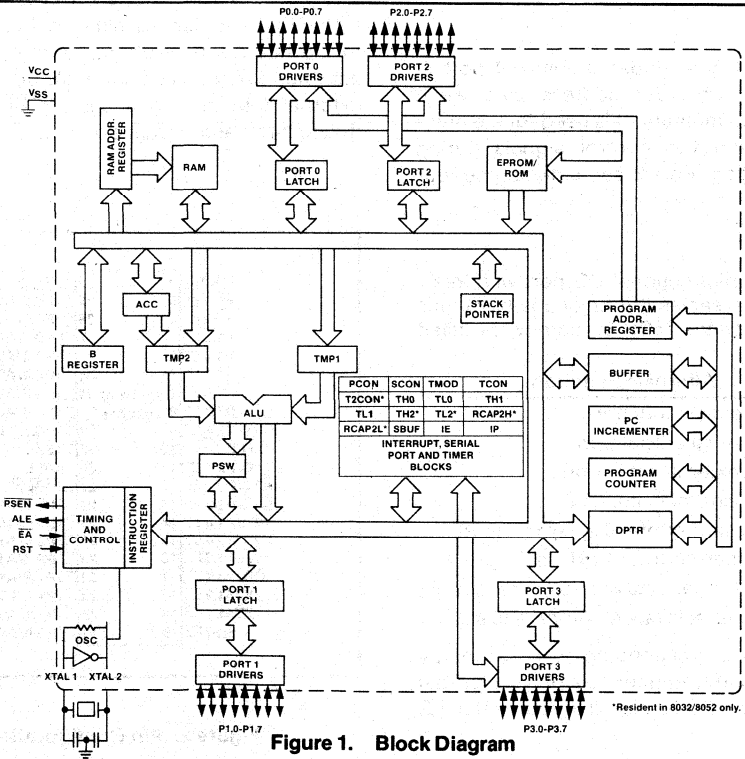


Figure 1. Block Diagram

8031AH/8051AH PIN DESCRIPTIONS

V_{SS}
Circuit ground potential.

V_{CC}
5V power supply input for normal operation and program verification.

Port 0
Port 0 is an 8-bit open drain bidirectional I/O port. It is also the multiplexed low-order address and data bus when using external memory. It is used for data output during program verification. Port 0 can sink/source eight LS TTL loads.

Port 1
Port 1 is an 8-bit quasi-bidirectional I/O port. It is also used for the low-order address byte during program verification. Port 1 can sink/source four LS TTL loads.

Port 2
Port 2 is an 8-bit quasi-bidirectional I/O port. It also emits the high-order address byte when accessing external memory. It is used for the high-order address and the control signals during program verification. Port 2 can sink/source four LS TTL loads.

Port 3
Port 3 is an 8-bit bidirectional I/O port with internal pullups. It also serves the functions of various special features of the MCS-51 Family, as listed below:

Port Pin	Alternate Function
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	INT0 (external interrupt)
P3.3	$\overline{INT1}$ (external interrupt)
P3.4	T0 (Timer/counter 0 external input)
P3.5	T1 (Timer/counter 1 external input)
P3.6	$\overline{WR}$ (external Data Memory write strobe)
P3.7	$\overline{RD}$ (external Data Memory read strobe)

The output latch corresponding to a secondary function must be programmed to a one (1) for that function to operate. Port 3 can sink/source four LS TTL loads.

RST/V_{PD}
A high on this pin for two machine cycles while the oscillator is running resets the device. A small external pulldown resistor ($\approx$8.2kΩ) from RST/VPD to V_{SS} permits power-on reset when a capacitor ($\approx$10 μf) is also connected from this pin to V_{CC}.

If RST/V_{PD} is held within the V_{PD} spec while V_{CC} drops below its spec, RST/V_{PD} will provide standby power to the RAM. When RST/V_{PD} is low, the RAM's current is drawn from V_{CC}.

ALE
Address Latch Enable output for latching the low byte of the address during accesses to external memory. ALE is activated at a constant rate of 1/6 the oscillator frequency except during an external data memory access at which time one ALE pulse is skipped. ALE can sink/source 8 LS TTL inputs.

$\overline{PSEN}$
The Program Store Enable output is a control signal that enables the external Program Memory to the bus during external fetch operations. It is activated every six oscillator periods except during external data memory access. $\overline{PSEN}$ remains high during internal program execution.

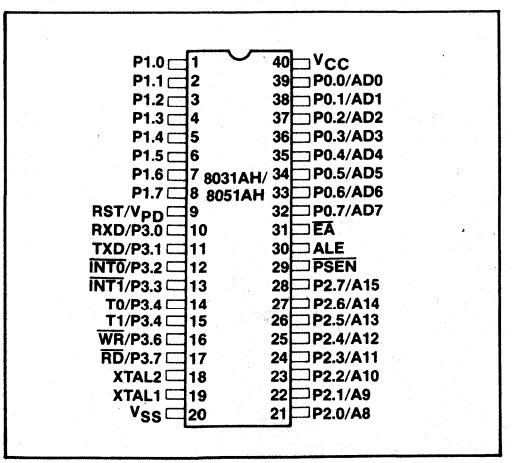


Figure 2. Pin Configuration

$\overline{EA}$

When held at a TTL high level, the 8051AH executes instructions from the internal ROM when the PC is less than 4096. When held at a TTL low level, the 8031AH/8051AH fetches all instructions from external Program Memory. Do not float $\overline{EA}$ during normal operation.

XTAL1

Input to the inverting amplifier that forms part of the oscillator. This pin should be connected to ground when an external oscillator is used.

XTAL2

Output of the inverting amplifier that forms part of the oscillator, and input to the internal clock generator. XTAL2 receives the oscillator signal when an external oscillator is used.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias . . . 0°C to 70°C

Storage Temperature . . . . . -65°C to +150°C

Voltage on Any Pin With Respect to Ground (V_{SS}) . . . . . -0.5V to +7V

Power Dissipation . . . . . 1 Watt

NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

DC CHARACTERISTICS (T_A = 0°C to 70°C; V_{CC} = 4.5V to 5.5V; V_{SS} = 0V)

Symbol	Parameter	Min	Max	Unit	Test Conditions
VIL	Input Low Voltage	-0.5	0.8	V	
VIH	Input High Voltage (Except RST/V_{PD} and XTAL2)	2.0	V_{CC} + 0.5	V	
VIH1	Input High Voltage to RST/V_{PD} For Reset, XTAL2	2.5	V_{CC} + 0.5	V	XTAL1 to V_{SS}
V_{PD}	Power Down Voltage to RST/V_{PD}	4.5	5.5	V	V_{CC} = 0V
VOL	Output Low Voltage Ports 1, 2, 3 (Note 1)		0.45	V	IOL = 1.6mA
VOL1	Output Low Voltage Port 0, ALE, $\overline{PSEN}$ (Note 1)		0.45	V	IOL = 3.2mA
VOH	Output High Voltage Ports 1, 2, 3	2.4		V	IOH = -80μA
VOH1	Output High Voltage Port 0, ALE, $\overline{PSEN}$	2.4		V	IOH = -400μA
IIL	Logical 0 Input Current Ports 1, 2, 3		-800	μA	Vin = 0.45V
IIL2	Logical 0 Input Current for XTAL 2		-2.5	mA	XTAL1 = V_{SS}, Vin = 0.45V
ILI	Input Leakage Current To Port 0, $\overline{EA}$		±10	μA	0.45V < Vin < V_{CC}
IIH1	Input High Current to RST/V_{PD} For Reset		500	μA	Vin < V_{CC} - 1.5V
ICC	Power Supply Current		125	mA	All outputs disconnected
IPD	Power Down Current		10	mA	V_{CC} = 0V
CIO	Capacitance of I/O Buffer		10	pF	f_C = 1MHz, T_A = 25°C

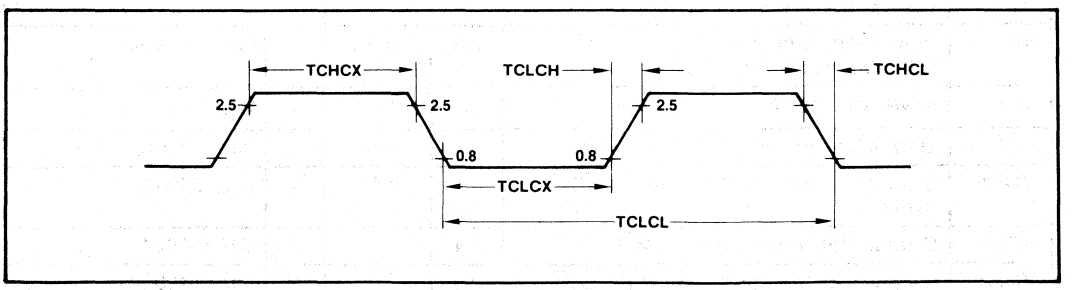
See page 4 for Notes.

Note 1: VOL is degraded when the 8031AH/8051AH rapidly discharges external capacitance. This AC noise is most pronounced during emission of address data. When using external memory, locate the latch or buffer as close to the 8031AH/8051AH as possible.

Datum	Emitting Ports	Degraded I/O Lines	VOL (peak) (max)
Address	P2, P0	P1, P3	0.8V
Write Data	P0	P1, P3, ALE	0.8V

EXTERNAL CLOCK DRIVE CHARACTERISTICS (XTAL2)

Symbol	Parameter	Variable Clock freq = 1.2 MHz to 12 MHz		Unit
		Min	Max	
TCLCL	Oscillator Period	83.3	833.3	ns
TCHCX	High Time	20		ns
TCLCX	Low Time	20		ns
TCLCH	Rise Time		20	ns
TCHCL	Fall Time		20	ns



AC CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5V ±10%, V_{SS} = 0V, C_L for Port 0, ALE and $\overline{PSEN}$ Outputs = 100 pF; C_L for all other outputs = 80 pF)

EXTERNAL PROGRAM MEMORY CHARACTERISTICS

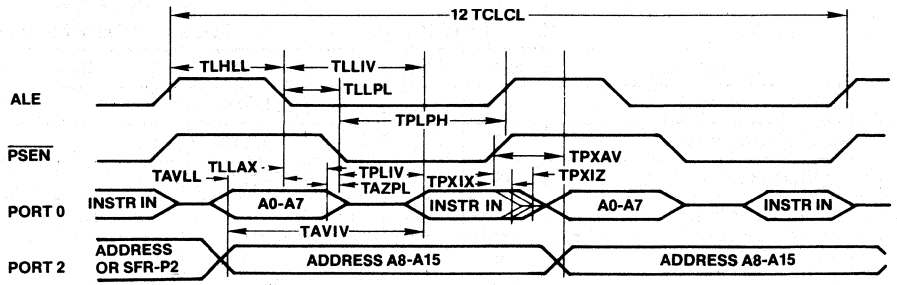
Symbol	Parameter	12 MHz Clock			Variable Clock 1/TCLCL = 1.2 MHz to 12 MHz		
		Min	Max	Unit	Min	Max	Unit
TLHLL	ALE Pulse Width	127		ns	2TCLCL-40		ns
TAVLL	Address Setup to ALE	43		ns	TCLCL-40		ns
TLLAX	Address Hold After ALE	48		ns	TCLCL-35		ns
TLLIV	ALE to Valid Instr In		233	ns		4TCLCL-100	ns
TLLPL	ALE To $\overline{PSEN}$	58		ns	TCLCL-25		ns
TPLPH	$\overline{PSEN}$ Pulse Width	215		ns	3TCLCL-35		ns
TPLIV	$\overline{PSEN}$ To Valid Instr In		125	ns		3TCLCL-125	ns
TPXIX	Input Instr Hold After $\overline{PSEN}$	0		ns	0		ns
TPXIZ	Input Instr Float After $\overline{PSEN}$		63	ns		TCLCL-20	ns
TPXAV	Address Valid After $\overline{PSEN}$	75		ns	TCLCL-8		ns
TAVIV	Address To Valid Instr In		302	ns		5TCLCL-115	ns
TAZPL	Address Float To $\overline{PSEN}$	0		ns	0		ns

EXTERNAL DATA MEMORY CHARACTERISTICS

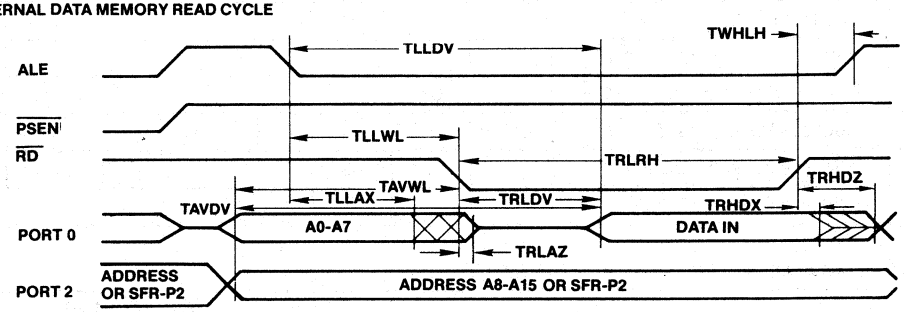
Symbol	Parameter	12 MHz Clock			Variable Clock 1/TCLCL = 1.2 MHz to 12 MHz		
		Min	Max	Unit	Min	Max	Unit
TRLRH	$\overline{RD}$ Pulse Width	400		ns	6TCLCL-100		ns
TWLWH	$\overline{WR}$ Pulse Width	400		ns	6TCLCL-100		ns
TLLAX	Address Hold After ALE	48		ns	TCLCL-35		
TRLDV	$\overline{RD}$ To Valid Data In		250	ns		5TCLCL-165	ns
TRHDX	Data Hold After $\overline{RD}$	0		ns	0		ns
TRHDZ	Data Float After $\overline{RD}$		97	ns		2TCLCL-70	ns
TLLDV	ALE To Valid Data In		517	ns		8TCLCL-150	ns
TAVDV	Address To Valid Data In		585	ns		9TCLCL-165	ns
TLLWL	ALE To $\overline{WR}$ or $\overline{RD}$	200	300	ns	3TCLCL-50	3TCLCL + 50	ns
TAVWL	Address To $\overline{WR}$ or $\overline{RD}$	203		ns	4TCLCL-130		ns
TWHLH	$\overline{WR}$ or $\overline{RD}$ High To ALE High	43	123	ns	TCLCL-40	TCLCL + 40	ns
TDVWX	Data Valid To $\overline{WR}$ Transition	23		ns	TCLCL-60		ns
TQVWH	Data Setup Before $\overline{WR}$	433		ns	7TCLCL-150		ns
TWHQX	Data Hold After $\overline{WR}$	33		ns	TCLCL-50		ns
TRLAZ	Address Float After $\overline{RD}$		0	ns		0	ns

AC TIMING DIAGRAMS

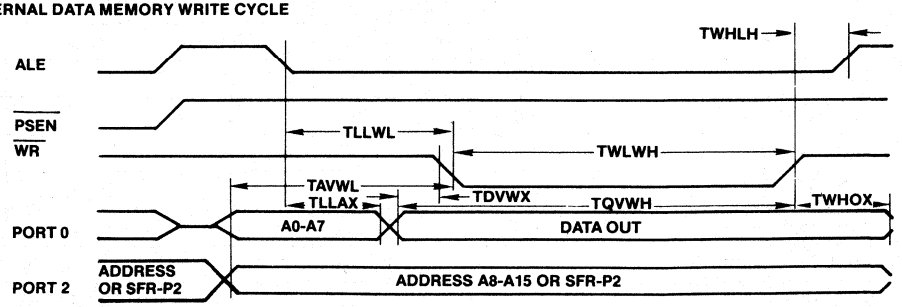
EXTERNAL PROGRAM MEMORY READ CYCLE



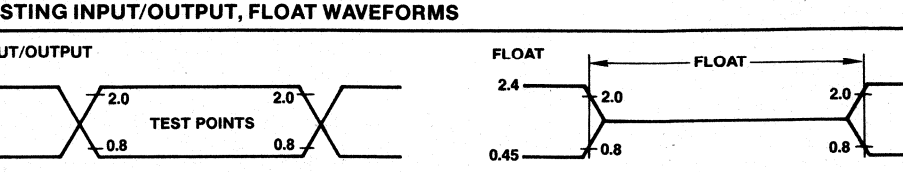
EXTERNAL DATA MEMORY READ CYCLE



EXTERNAL DATA MEMORY WRITE CYCLE

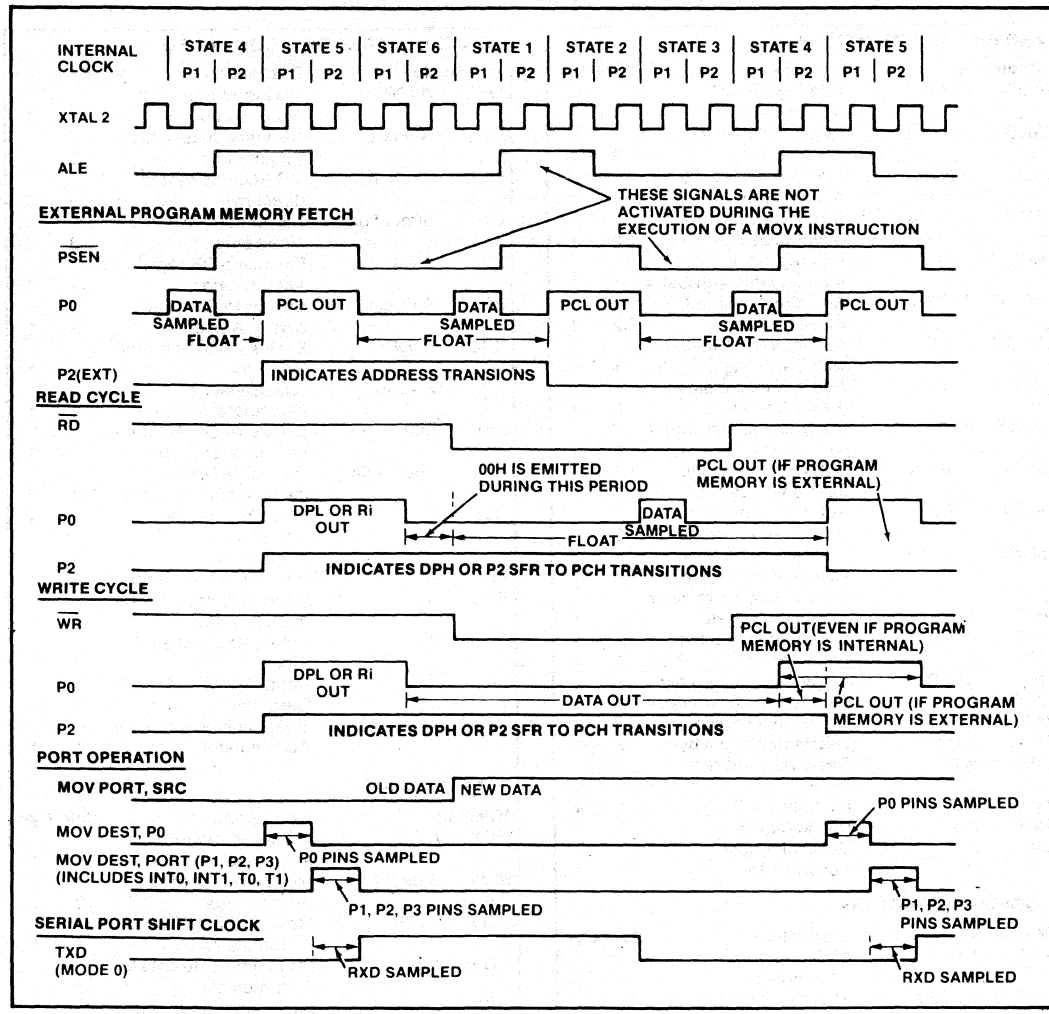


AC TESTING INPUT/OUTPUT, FLOAT WAVEFORMS



AC inputs during testing are driven at 2.4V for a logic "1" and 0.45V for a logic "0". Timing measurements are made at 2.0V for a logic "1" and 0.8V for a logic "0". For timing purposes, the float state is defined as the point at which a P0 pin sinks 3.2mA or sources 400 μA at the voltage test levels.

CLOCK WAVEFORMS



This diagram indicates when signals are clocked internally. The time it takes the signals to propagate to the pins, however, ranges from 25 to 125 ns. This propagation delay is dependent on variables such as temperature and pin loading. Propagation also varies from output to output and component to component. Typically though, (T_A = 25° C, fully loaded) RD and WR propagation delays are approximately 50 ns. The other signals are typically 85 ns. Propagation delays are incorporated in the AC specifications.

Table 1. MCS®-51 Instruction Set Description

ARITHMETIC OPERATIONS

Mnemonic		Description	Byte	Cyc
ADD	A,Rn	Add register to Accumulator	1	1
ADD	A,direct	Add direct byte to Accumulator	2	1
ADD	A,@Ri	Add indirect RAM to Accumulator	1	1
ADD	A,#data	Add immediate data to Accumulator	2	1
ADDC	A,Rn	Add register to Accumulator with Carry	1	1
ADDC	A,direct	Add direct byte to A with Carry flag	2	1
ADDC	A,@Ri	Add indirect RAM to A with Carry flag	1	1
ADDC	A,#data	Add immediate data to A with Carry flag	2	1
SUBB	A,Rn	Subtract register from A with Borrow	1	1
SUBB	A,direct	Subtract direct byte from A with Borrow	2	1
SUBB	A,@Ri	Subtract indirect RAM from A with Borrow	1	1
SUBB	A,#data	Subtract immed data from A with Borrow	2	1
INC	A	Increment Accumulator	1	1
INC	Rn	Increment register	1	1
INC	direct	Increment direct byte	2	1
INC	@Ri	Increment indirect RAM	1	1
INC	DPTR	Increment Data Pointer	1	2
DEC	A	Decrement Accumulator	1	1
DEC	Rn	Decrement register	1	1
DEC	direct	Decrement direct byte	2	1
DEC	@Ri	Decrement indirect RAM	1	1
MUL	AB	Multiply A & B	1	4
DIV	AB	Divide A by B	1	4
DA	A	Decimal Adjust Accumulator	1	1

LOGICAL OPERATIONS

Mnemonic		Destination	Byte	Cyc
ANL	A,Rn	AND register to Accumulator	1	1
ANL	A,direct	AND direct byte to Accumulator	2	1
ANL	A,@Ri	AND indirect RAM to Accumulator	1	1
ANL	A,#data	AND immediate data to Accumulator	2	1
ANL	direct,A	AND Accumulator to direct byte	2	1
ANL	direct,#data	AND immediate data to direct byte	3	2
ORL	A,Rn	OR register to Accumulator	1	1
ORL	A,direct	OR direct byte to Accumulator	2	1

LOGICAL OPERATIONS (CONTINUED)

Mnemonic		Destination	Byte	Cyc
ORL	A,@Ri	OR indirect RAM to Accumulator	1	1
ORL	A,#data	OR immediate data to Accumulator	2	1
ORL	direct,A	OR Accumulator to direct byte	2	1
ORL	direct,#data	OR immediate data to direct byte	3	2
XRL	A,Rn	Exclusive-OR register to Accumulator	1	1
XRL	A,direct	Exclusive-OR direct byte to Accumulator	2	1
XRL	A,@Ri	Exclusive-OR indirect RAM to A	1	1
XRL	A,#data	Exclusive-OR immediate data to A	2	1
XRL	direct,A	Exclusive-OR Accumulator to direct byte	2	1
XRL	direct,#data	Exclusive-OR immediate data to direct	3	2
CLR	A	Clear Accumulator	1	1
CPL	A	Complement Accumulator	1	1
RL	A	Rotate Accumulator Left	1	1
RLC	A	Rotate A Left through the Carry flag	1	1
RR	A	Rotate Accumulator Right	1	1
RRC	A	Rotate A Right through Carry flag	1	1
SWAP	A	Swap nibbles within the Accumulator	1	1

DATA TRANSFER

Mnemonic		Description	Byte	Cyc
MOV	A,Rn	Move register to Accumulator	1	1
MOV	A,direct	Move direct byte to Accumulator	2	1
MOV	A,@Ri	Move indirect RAM to Accumulator	1	1
MOV	A,#data	Mov immediate data to Accumulator	2	1
MOV	Rn,A	Move Accumulator to register	1	1
MOV	Rn,direct	Move direct byte to register	2	2
MOV	Rn,#data	Move immediate data to register	2	1
MOV	direct,A	Move Accumulator to direct byte	2	1
MOV	direct,Rn	Move register to direct byte	2	2
MOV	direct,direct	Move direct byte to direct	3	2
MOV	direct,@Ri	Move indirect RAM to direct byte	2	2

Table 1. (Cont.)

DATA TRANSFER (CONTINUED)

Mnemonic		Description	Byte	Cyc
MOV	direct,#data	Move immediate data to direct byte	3	2
MOV	@Ri,A	Move Accumulator to indirect RAM	1	1
MOV	@Ri,direct	Move direct byte to indirect RAM	2	2
MOV	@Ri,#data	Move immediate data to indirect RAM	2	1
MOV	DPTR,#data16	Load Data Pointer with a 16-bit constant	3	2
MOVC	A,@A+DPTR	Move Code byte relative to DPTR to A	1	2
MOVC	A,@A+PC	Move Code byte relative to PC to A	1	2
MOVX	A,@Ri	Move External RAM (8-bit addr) to A	1	2
MOVX	A,@DPTR	Move External RAM (16-bit addr) to A	1	2
MOVX	@Ri,A	Move A to External RAM (8-bit addr)	1	2
MOVX	@DPTR,A	Move A to External RAM (16-bit addr)	1	2
PUSH	direct	Push direct byte onto stack	2	2
POP	direct	Pop direct byte from stack	2	2
XCH	A,Rn	Exchange register with Accumulator	1	1
XCH	A,direct	Exchange direct byte with Accumulator	2	1
XCH	A,@Ri	Exchange indirect RAM with A	1	1
XCHD	A,@Ri	Exchange low-order Digit ind RAM w A	1	1

BOOLEAN VARIABLE MANIPULATION

Mnemonic		Description	Byte	Cyc
CLR	C	Clear Carry flag	1	1
CLR	bit	Clear direct bit	2	1
SETB	C	Set Carry flag	1	1
SETB	bit	Set direct Bit	2	1
CPL	C	Complement Carry flag	1	1
CPL	bit	Complement direct bit	2	1
ANL	C,bit	AND direct bit to Carry flag	2	2
ANL	C,1 bit	AND complement of direct bit to Carry	2	2
ORL	C/bit	OR direct bit to Carry flag	2	2
ORL	C,1 bit	OR complement of direct bit to Carry	2	2
MOV	C/bit	Move direct bit to Carry flag	2	1
MOV	bit,C	Move Carry flag to direct bit	2	2

PROGRAM AND MACHINE CONTROL

Mnemonic		Description	Byte	Cyc
ACALL	addr11	Absolute Subroutine Call	2	2
LCALL	addr16	Long Subroutine Call	3	2
RET		Return from subroutine	1	2
RETI		Return from interrupt	1	2
AJMP	addr11	Absolute Jump	2	2
LJMP	addr16	Long Jump	3	2
SJMP	rel	Short Jump (relative addr)	2	2
JMP	@A+DPTR	Jump indirect relative to the DPTR	1	2
JZ	rel	Jump if Accumulator is Zero	2	2
JNZ	rel	Jump if Accumulator is Not Zero	2	2
JC	rel	Jump if Carry flag is set	2	2
JNC	rel	Jump if No Carry flag	2	2
JB	bit,rel	Jump if direct Bit set	3	2
JNB	bit,rel	Jump if direct Bit Not set	3	2
JBC	bit,rel	Jump if direct Bit is set & Clear bit	3	2
CJNE	A,direct,rel	Compare direct to A & Jump if Not Equal	3	2
CJNE	A,#data,rel	Comp, immed, to A & Jump if Not Equal	3	2
CJNE	Rn,#data,rel	Comp, immed, to reg & Jump if Not Equal	3	2
CJNE	@Ri,#data,rel	Comp, immed, to ind, & Jump if Not Equal	3	2
DJNZ	Rn,rel	Decrement register & Jump if Not Zero	2	2
DJNZ	direct,rel	Decrement direct & Jump if Not Zero	3	2
NOP		No operation	1	1

Notes on data addressing modes:

Rn —Working register R0-R7
direct —128 internal RAM locations, any I/O port, control or status register
@Ri —Indirect internal RAM location addressed by register R0 or R1
#data —8-bit constant included in instruction
#data16 —16-bit constant included as bytes 2 & 3 of instruction
bit —128 software flags, any I/O pin, control or status bit

Notes on program addressing modes:

addr16 —Destination address for LCALL & LJMP may be anywhere within the 64-K program memory address space
Addr11 —Destination address for ACALL & AJMP will be within the same 2-K page of program memory as the first byte of the following instruction
rel —SJMP and all conditional jumps include an 8-bit offset byte, Range is +127-128 bytes relative to first byte of the following instruction

Table 2. Instruction Opcodes in Hexadecimal Order

Hex Code	Number of Bytes	Mnemonic	Operands
00	1	NOP	
01	2	AJMP	code addr
02	3	LJMP	code addr
03	1	RR	A
04	1	INC	A
05	2	INC	data addr
06	1	INC	@R0
07	1	INC	@R1
08	1	INC	R0
09	1	INC	R1
0A	1	INC	R2
0B	1	INC	R3
0C	1	INC	R4
0D	1	INC	R5
0E	1	INC	R6
0F	1	INC	R7
10	3	JBC	bit addr, code addr
11	2	ACALL	code addr
12	3	LCALL	code addr
13	1	RRC	A
14	1	DEC	A
15	2	DEC	data addr
16	1	DEC	@R0
17	1	DEC	@R1
18	1	DEC	R0
19	1	DEC	R1
1A	1	DEC	R2
1B	1	DEC	R3
1C	1	DEC	R4
1D	1	DEC	R5
1E	1	DEC	R6
1F	1	DEC	R7
20	3	JB	bit addr, code addr
21	2	AJMP	code addr
22	1	RET	
23	1	RL	A
24	2	ADD	A,#data
25	2	ADD	A,data addr
26	1	ADD	A,@R0
27	1	ADD	A,@R1
28	1	ADD	A,R0
29	1	ADD	A,R1
2A	1	ADD	A,R2
2B	1	ADD	A,R3
2C	1	ADD	A,R4
2D	1	ADD	A,R5
2E	1	ADD	A,R6
2F	1	ADD	A,R7
30	3	JNB	bit addr, code addr
31	2	ACALL	code addr
32	1	RETI	
33	1	RLC	A
34	2	ADDC	A,#data
35	2	ADDC	A,data addr
36	1	ADDC	A,@R0
37	1	ADDC	A,@R1
38	1	ADDC	A,R0
39	1	ADDC	A,R1
3A	1	ADDC	A,R2
3B	1	ADDC	A,R3
3C	1	ADDC	A,R4
3D	1	ADDC	A,R5
3E	1	ADDC	A,R6
3F	1	ADDC	A,R7
40	2	JC	code addr
41	2	AJMP	code addr
42	2	ORL	data addr,A
43	3	ORL	data addr,#data
44	2	ORL	A,#data
45	2	ORL	A,data addr
46	1	ORL	A,@R0
47	1	ORL	A,@R1
48	1	ORL	A,R0
49	1	ORL	A,R1
4A	1	ORL	A,R2
4B	1	ORL	A,R3
4C	1	ORL	A,R4
4D	1	ORL	A,R5
4E	1	ORL	A,R6
4F	1	ORL	A,R7
50	2	JNC	code addr
51	2	ACALL	code addr
52	2	ANL	data addr,A
53	3	ANL	data addr,#data
54	2	ANL	A,#data
55	2	ANL	A,data addr
56	1	ANL	A,@R0
57	1	ANL	A@R1
58	1	ANL	A,R0
59	1	ANL	A,R1
5A	1	ANL	A,R2
5B	1	ANL	A,R3
5C	1	ANL	A,R4
5D	1	ANL	A,R5
5E	1	ANL	A,R6
5F	1	ANL	A,R7
60	2	JZ	code addr
61	2	AJMP	code addr
62	2	XRL	data addr.A
63	3	XRL	data addr,#data
64	2	XRL	A,#data
65	2	XRL	A,data addr

Table 2. (Cont.)

Hex Code	Number of Bytes	Mnemonic	Operands
66	1	XRL	A,@R0
67	1	XRL	A,@R1
68	1	XRL	A,R0
69	1	XRL	A,R1
6A	1	XRL	A,R2
6B	1	XRL	A,R3
6C	1	XRL	A,R4
6D	1	XRL	A,R5
6E	1	XRL	A,R6
6F	1	XRL	A,R7
70	2	JNZ	code addr
71	2	ACALL	code addr
72	2	ORL	C,bit addr
73	1	JMP	@A+DPTR
74	2	MOV	A,#data
75	3	MOV	data addr,#data
76	2	MOV	@R0,#data
77	2	MOV	@R1,#data
78	2	MOV	R0,#data
79	2	MOV	R1,#data
7A	2	MOV	R2,#data
7B	2	MOV	R3,#data
7C	2	MOV	R4,#data
7D	2	MOV	R5,#data
7E	2	MOV	R6,#data
7F	2	MOV	R7,#data
80	2	SJMP	code addr
81	2	AJMP	code addr
82	2	ANL	C,bit addr
83	1	MOVC	A,@A+PC
84	1	DIV	AB
85	3	MOV	data addr, data addr
86	2	MOV	data addr,@R0
87	2	MOV	data addr,@R1
88	2	MOV	data addr,R0
89	2	MOV	data addr,R1
8A	2	MOV	data addr,R2
8B	2	MOV	data addr,R3
8C	2	MOV	data addr,R4
8D	2	MOV	data addr,R5
8E	2	MOV	data addr,R6
8F	2	MOV	data addr,R7
90	3	MOV	DPTR,#data
91	2	ACALL	code addr
92	2	MOV	bit addr,C
93	1	MOVC	A,@A+DPTR
94	2	SUBB	A,#data
95	2	SUBB	A,data addr
96	1	SUBB	A,@R0
97	1	SUBB	A,@R1
98	1	SUBB	A,R0
99	1	SUBB	A,R1
9A	1	SUBB	A,R2
9B	1	SUBB	A,R3
9C	1	SUBB	A,R4
9D	1	SUBB	A,R5
9E	1	SUBB	A,R6
9F	1	SUBB	A,R7
A0	2	ORL	C,/bit addr
A1	2	AJMP	code addr
A2	2	MOV	C,bit addr
A3	1	INC	DPTR
A4	1	MUL	AB
A5		reserved	
A6	2	MOV	@R0, data addr
A7	2	MOV	@R1,data addr
A8	2	MOV	R0,data addr
A9	2	MOV	R1,data addr
AA	2	MOV	R2,data addr
AB	2	MOV	R3,data addr
AC	2	MOV	R4,data addr
AD	2	MOV	R5,data addr
AE	2	MOV	R6,data addr
AF	2	MOV	R7,data addr
B0	2	ANL	C,/bit addr
B1	2	ACALL	code addr
B2	2	CPL	bit addr
B3	1	CPL	C
B4	3	CJNE	A,#data,code addr
B5	3	CJNE	A,data addr,code addr
B6	3	CJNE	@R0,#data,code addr
B7	3	CJNE	@R1,#data,code addr
B8	3	CJNE	R0,#data,code addr
B9	3	CJNE	R1,#data,code addr
BA	3	CJNE	R2,#data,code addr
BB	3	CJNE	R3,#data,code addr
BC	3	CJNE	R4,#data,code addr
BD	3	CJNE	R5,#data,code addr
BE	3	CJNE	R6,#data,code addr
BF	3	CJNE	R7,#data,code addr
C0	2	PUSH	data addr
C1	2	AJMP	code addr
C2	2	CLR	bit addr
C3	1	CLR	C
C4	1	SWAP	A
C5	2	XCH	A,data addr
C6	1	XCH	A,@R0
C7	1	XCH	A,@R1
C8	1	XCH	A,R0
C9	1	XCH	A,R1
CA	1	XCH	A,R2
CB	1	XCH	A,R3

Table 2. (Cont.)

Hex Code	Number of Bytes	Mnemonic	Operands
CC	1	XCH	A,R4
CD	1	XCH	A,R5
CE	1	XCH	A,R6
CF	1	XCH	A,R7
D0	2	POP	data addr
D1	2	ACALL	code addr
D2	2	SETB	bit addr
D3	1	SETB	C
D4	1	DA	A
D5	3	DJNZ	data addr,code addr
D6	1	XCHD	A,@R0
D7	1	XCHD	A,@R1
D8	2	DJNZ	R0,code addr
D9	2	DJNZ	R1,code addr
DA	2	DJNZ	R2,code addr
DB	2	DJNZ	R3,code addr
DC	2	DJNZ	R4,code addr
DD	2	DJNZ	R5,code addr
DE	2	DJNZ	R6,code addr
DF	2	DJNZ	R7,code addr
E0	1	MOVX	A,@DPTR
E1	2	AJMP	code addr
E2	1	MOVX	A,@R0
E3	1	MOVX	A,@R1
E4	1	CLR	A
E5	2	MOV	A,data addr

Hex Code	Number of Bytes	Mnemonic	Operands
E6	1	MOV	A,@R0
E7	1	MOV	A,@R1
E8	1	MOV	A,R0
E9	1	MOV	A,R1
EA	1	MOV	A,R2
EB	1	MOV	A,R3
EC	1	MOV	A,R4
ED	1	MOV	A,R5
EE	1	MOV	A,R6
EF	1	MOV	A,R7
F0	1	MOVX	@DPTR,A
F1	2	ACALL	code addr
F2	1	MOVX	@R0,A
F3	1	MOVX	@R1,A
F4	1	CPL	A
F5	2	MOV	data addr,A
F6	1	MOV	@R0,A
F7	1	MOV	@R1,A
F8	1	MOV	R0,A
F9	1	MOV	R1,A
FA	1	MOV	R2,A
FB	1	MOV	R3,A
FC	1	MOV	R4,A
FD	1	MOV	R5,A
FE	1	MOV	R6,A
FF	1	MOV	R7,A

PRELIMINARY

80C31/80C51
CHMOS SINGLE-COMPONENT 8-BIT MICROCOMPUTER

- **80C31—Control-Oriented CPU with RAM and I/O**
- **80C51—An 80C31 with Factory Mask-Programmable ROM**

- **8031/8051 Compatible**
  - **4K x 8 ROM**
  - **128 x 8 RAM**
  - **Two 16-Bit Timer/Counters**
  - **Full-Duplex Serial Channel**
  - **Boolean Processor**
- **Low Power Consumption:**
  - **Normal Operation—24 mA @ 5V, 12 MHz**
  - **Idle Mode—3mA @ 5V, 12 MHz**
  - **Power Down Mode—50μA @ 2V**
- **CMOS and TTL Compatible**

The 80C31/80C51 is a functionally compatible 8031AH/8051AH microcomputer fabricated on Intel®'s CHMOS process. CHMOS is a technology which combines the high speed and density characteristics of HMOS-II with the low power attributes of CMOS. This combination expands the effectiveness of the powerful MCS®-51 architecture and instruction set.

Like the 8031AH/8051AH, the 80C31/80C51 has the following features: 4K bytes of ROM (8051AH/80C51 only); 128 bytes of RAM; 32 I/O lines; two 16-bit timer/counters; a five-source two-level interrupt structure; a full duplex serial port; and on-chip oscillator and clock circuitry. In addition, the 80C31/80C51 has two software selectable modes of reduced activity for further power reduction—Idle and Power Down.

Idle mode freezes the CPU while allowing the RAM, timers, serial port and interrupt system to continue functioning. Power Down mode saves the RAM contents but freezes the oscillator causing all other chip functions to be inoperative.

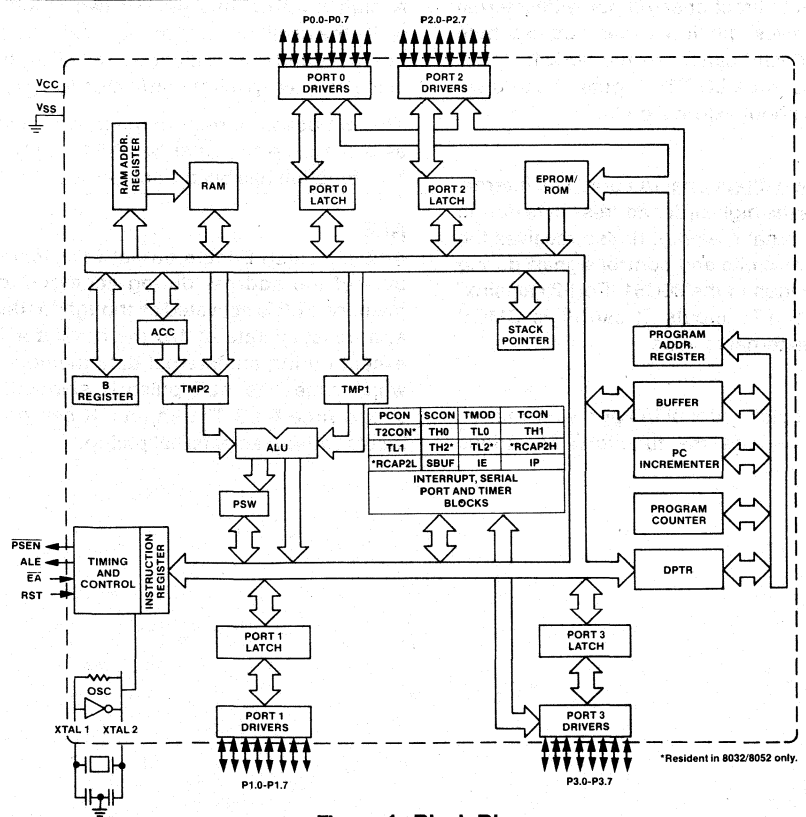


Figure 1. Block Diagram

80C31/80C51 PIN DESCRIPTIONS

V_{SS}
Circuit ground potential.

V_{CC}
Supply voltage during normal, Idle, and Power Down operation.

Port 0
Port 0 is an 8-bit open drain bidirectional I/O port. It is also the multiplexed low-order address and data bus during accesses to external memory. It also outputs instruction bytes during program verification. External pullups are required during program verification. Port 0 can sink eight LS TTL inputs.

Port 1
Port 1 is an 8-bit bidirectional I/O port with internal pullups. It receives the low-order address byte during program verification. In the 80C51, Port 1 can sink/source three LS TTL inputs. It can drive CMOS inputs without external pullups.

Port 2
Port 2 is an 8-bit bidirectional I/O port with internal pullups. It emits the high-order address byte during accesses to external memory. It also receives the high-order address bits and control signals during program verification in the 80C51. Port 2 can sink/source three LS TTL inputs. It can drive CMOS inputs without external pullups.

Port 3
Port 3 is an 8-bit bidirectional I/O port with internal pullups. It also serves the functions of various special features of the MCS-51 Family, as listed below:

Port Pin	Alternate Function
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	INT0 (external interrupt)
P3.3	INT1 (external interrupt)
P3.4	T0 (Timer 0 external input)
P3.5	T1 (Timer 1 external input)
P3.6	WR (external Data Memory write strobe)
P3.7	RD (external Data Memory read strobe)

Port 3 can sink/source three LS TTL inputs. It can drive CMOS inputs without external pullups.

RST
A high level on this pin for two machine cycles while the oscillator is running resets the device. An internal pulldown resistor permits Power-On reset using only a capacitor connected to V_{CC}.

This pin does not receive the power down voltage as is the case for other MCS-51 family members. This function has been transferred to the V_{CC} pin.

ALE
Address Latch Enable output for latching the low byte of the address during accesses to external memory. ALE is activated as though for this purpose at a constant rate of 1/6 the oscillator frequency except during an external data memory access at which time one ALE pulse is skipped. ALE can sink/source 8 LS TTL inputs. It can drive CMOS inputs without an external pullup.

PSEN

Program Store Enable output is the read strobe to external Program Memory. PSEN is activated twice each machine cycle during fetches from external Program Memory. (However, when executing out of external Program Memory, two activations of PSEN are skipped during each access to external Data Memory.) PSEN is not activated during fetches from internal Program Memory. PSEN can sink/source 8 LS TTL inputs. It can drive CMOS inputs without an external pullup.

EA

When EA is held high, the CPU executes out of internal Program Memory (unless the Program Counter exceeds 0FFFH). When EA is held low, the CPU executes only out of external Program Memory. EA must not be floated.

XTAL1

Input to the inverting amplifier that forms the oscillator. Receives the external oscillator signal when air external oscillator is used.

XTAL2

Output of the inverting amplifier that forms the oscillator, and input to the internal clock generator. This pin should be floated when an external oscillator is used.

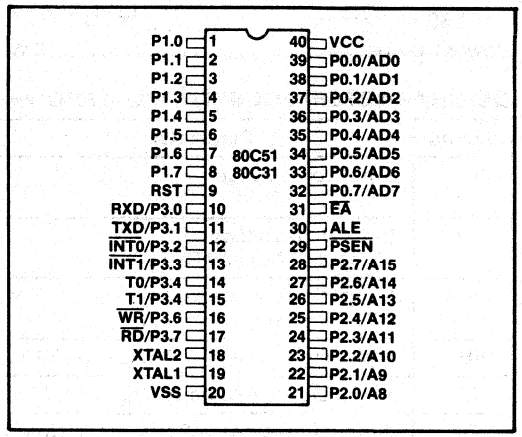


Figure 2. Pin Configuration

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias . . . 0°C to 70°C

Storage Temperature . . . . . -65°C to +150°C

Voltage from V_{CC} to V_{SS} . . . . . -0.5V to +7V

Voltage from Any Pin to V_{SS} . . . . . -0.5V to V_{CC} +0.5V

Power Dissipation . . . . . 2 W

**NOTICE: Stresses at or above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions may affect device reliability.*

DC CHARACTERISTICS (T_A = 0°C to 70°C; V_{CC} = 4.0V to 6.0V; V_{SS} = 0V)

Symbol	Parameter	Min	Max	Unit	Test Conditions
VIL	Input Low Voltage	-0.5 -0.5	0.8 0.5	V V	V_{CC}>4.5V V_{CC}<4.5V
VIH	Input High Voltage (Except XTALs and RST)	2.0 2.5	V_{CC} V_{CC}	V V	V_{CC} < 5.5V V_{CC} > 5.5V
VIH1	Input High Voltage to RST For Reset	3.0 3.5	V_{CC} V_{CC}	V V	V_{CC} < 5.5 V_{CC} > 5.5
VIH2	Input High Voltage To XTAL1 and XTAL2	0.8V_{CC}	V_{CC}	V	
VPD	Power Down Voltage To V_{CC} in PD Mode	2.0	6.0	V	
VOL	Output Low Voltage Ports 1, 2, 3 (Note 1)		0.45	V	IOL = 1.6mA
VOL1	Output Low Voltage Port 0, ALE, $\overline{PSEN}$ (Note 1)		0.45	V	IOL = 3.2mA
VOH	Output High Voltage Ports 1, 2, 3	0.9V_{CC}		V	IOH = 10μA
		2.4		V	IOH = -80μA V_{CC} = 5V ± 10%
VOH1	Output High Voltage Port 0 (in External Bus Mode), ALE, $\overline{PSEN}$	0.9V_{CC}		V	IOH = -40μA
		2.4		V	IOH = -400μA VCC = 5V ± 10%
IIL	Logical 0 Input Current Ports 1, 2, 3		-50	μA	Vin = 0.45V
ILI	Input Leakage Current		±10	μA	Vin = V_{SS} or Vin = V_{CC}
ICC	Power Supply Current Normal Operation	All outputs disconnected; V_{CC} = 5V; Rise and fall times of clock drive < 10 nsec.			
			24	mA	f_{OSC} = 12MHz
			2.4	mA	f_{OSC} = 1.2MHz
ICC1	Idle Mode		3.0	mA	f_{OSC} = 12MHz
			0.3	mA	f_{OSC} = 1.2MHz
ICC2	Power Supply Current (Power Down Mode)		50	μA	V_{CC} = 2V
RRST	RST Pulldown Resistor	4		kΩ	
CIO	Capacitance of I/O Buffer		10	pF	f_c = 1MHz

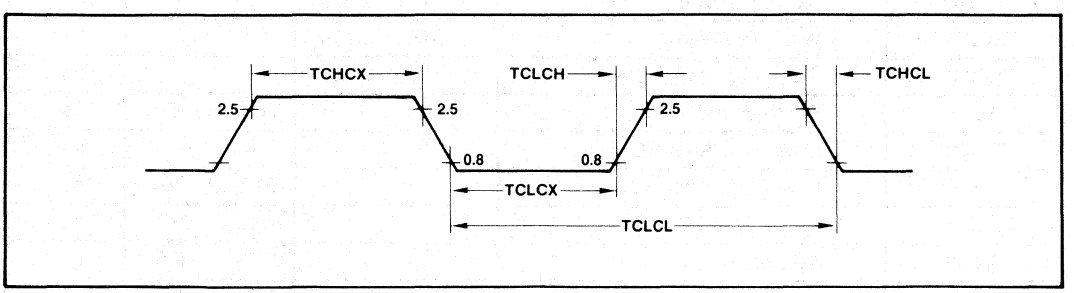
See page 5 for Notes.

Note 1: VOL is degraded when the 80C31/80C51 rapidly discharges external capacitance. This AC noise is most pronounced during emission of address data. When using external memory, locate the latch or buffer as close to the 80C31/80C51 as possible.

Datum	Emitting Ports	Degraded I/O Lines	VOL (peak) (max)
Address	P2, P0	P1, P3	0.8V
Write Data	P0	P1, P3, ALE	0.8V

EXTERNAL CLOCK DRIVE CHARACTERISTICS (XTAL1)

Symbol	Parameter	Variable Clock freq = 1.2 MHz to 12 MHz		Unit
		Min	Max	
TCLCL	Oscillator Period	83.3	833	ns
TCHCX	High Time	20		ns
TCLCX	Low Time	20		ns
TCLCH	Rise Time		20	ns
TCHCL	Fall Time		20	ns



AC CHARACTERISTICS (T_A = 0°C to 70°C; V_{CC} = 4.0V to 6.0V; V_{SS} = 0; Load Capacitance for Port 0, ALE, and $\overline{PSEN}$ = 100pf; Load Capacitance for All Other Outputs = 80pf.)

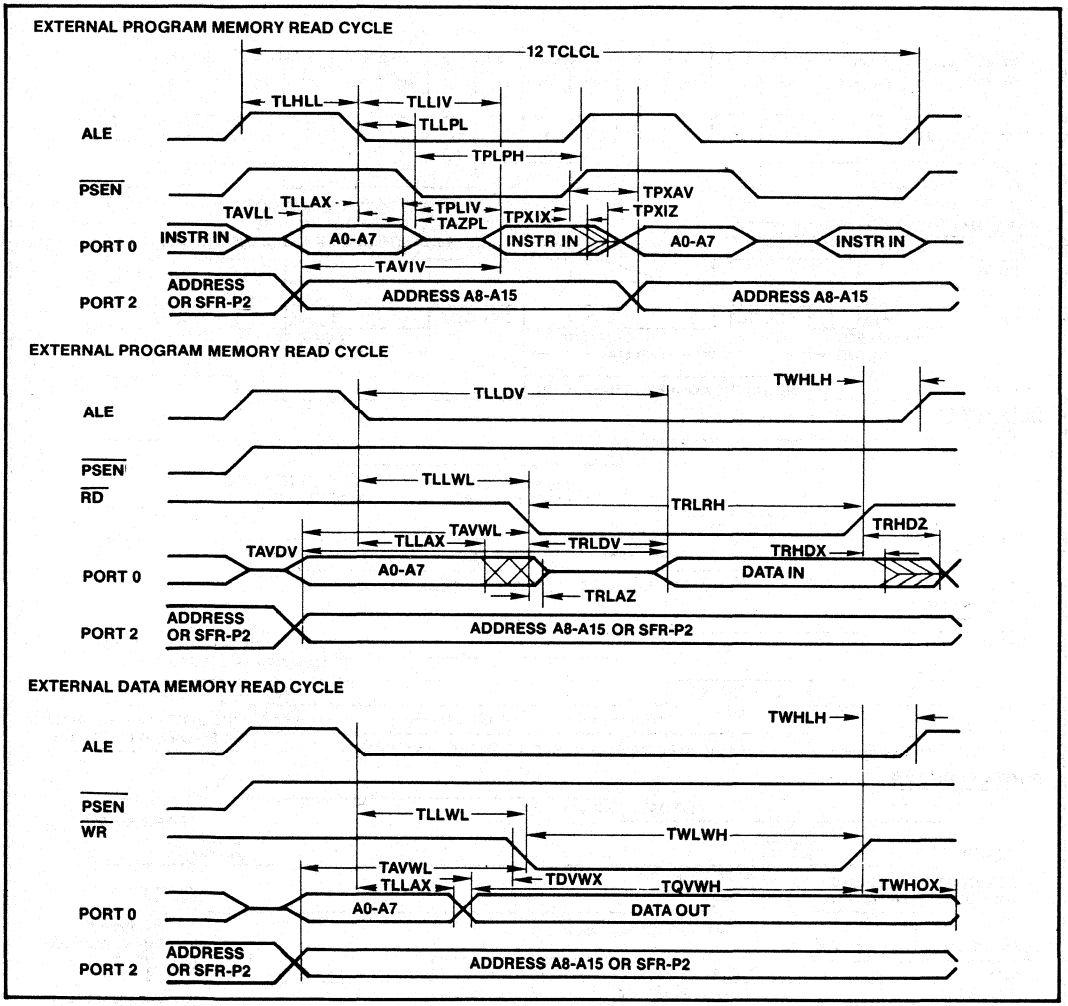
EXTERNAL PROGRAM MEMORY CHARACTERISTICS

Symbol	Parameter	Min	Max	Units
TLHLL	ALE Pulse Width	2TCLCL-40		ns
TAVLL	Address Valid to ALE	TCLCL-40		ns
TLLAX	Address Hold After ALE	TCLCL-35		ns
TLLIV	ALE to Valid Instr In		4TCLCL-100	ns
TLLPL	ALE to $\overline{PSEN}$	TCLCL-25		ns
TPLPH	$\overline{PSEN}$ Pulse Width	3TCLCL-35		ns
TPLIV	$\overline{PSEN}$ to Valid Instr In		3TCLCL-125	ns
TPXIX	Input Instr Hold After $\overline{PSEN}$	0		ns
TPXIZ	Input Instr Float After $\overline{PSEN}$		TCLCL-20	ns
TPXAV	$\overline{PSEN}$ to Address Valid	TCLCL-8		ns
TAVIV	Address to Valid Instr In		5TCLCL-115	ns
TAZPL	Address Float to $\overline{PSEN}$	0		ns

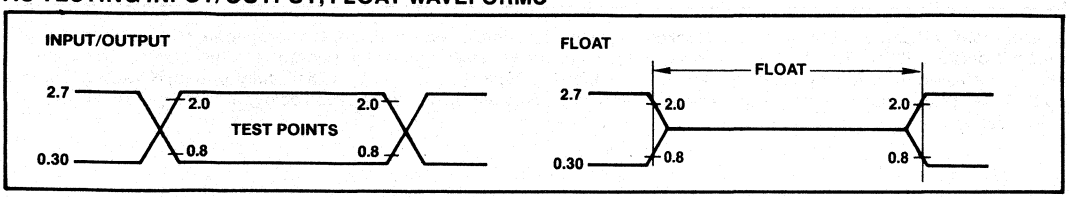
EXTERNAL DATA MEMORY CHARACTERISTICS

Symbol	Parameter	Min	Max	Units
TRLRH	$\overline{RD}$ Pulse Width	6TCLCL-100		ns
TWLWH	$\overline{WR}$ Pulse Width	6TCLCL-100		ns
TLLAX	Data Address Hold After ALE	TCLCL-35		ns
TRLDV	$\overline{RD}$ to Valid Data In		5TCLCL-165	ns
TRHDX	Data Hold After $\overline{RD}$	0		ns
TRHDZ	Data Float After $\overline{RD}$		2TCLCL-70	ns
TLLDV	ALE to Valid Data In		8TCLCL-150	ns
TAVDV	Address to Valid Data In		9TCLCL-165	ns
TLLWL	ALE to $\overline{WR}$ or $\overline{RD}$	3TCLCL-50	3TCLCL+50	ns
TAVWL	Address to $\overline{WR}$ or $\overline{RD}$	4TCLCL-130		ns
TDVWX	Data Valid to $\overline{WR}$ Transition	TCLCL-60		ns
TQVWH	Data Setup to $\overline{WR}$ High	7TCLCL-150		ns
TWHQX	Data Held After $\overline{WR}$	TCLCL-50		ns
TRLAZ	$\overline{RD}$ Low to Address Float		0	ns
TWHLH	$\overline{RD}$ or $\overline{WR}$ High to ALE High	TCLCL-50	TCLCL+50	ns

AC TIMING DIAGRAMS

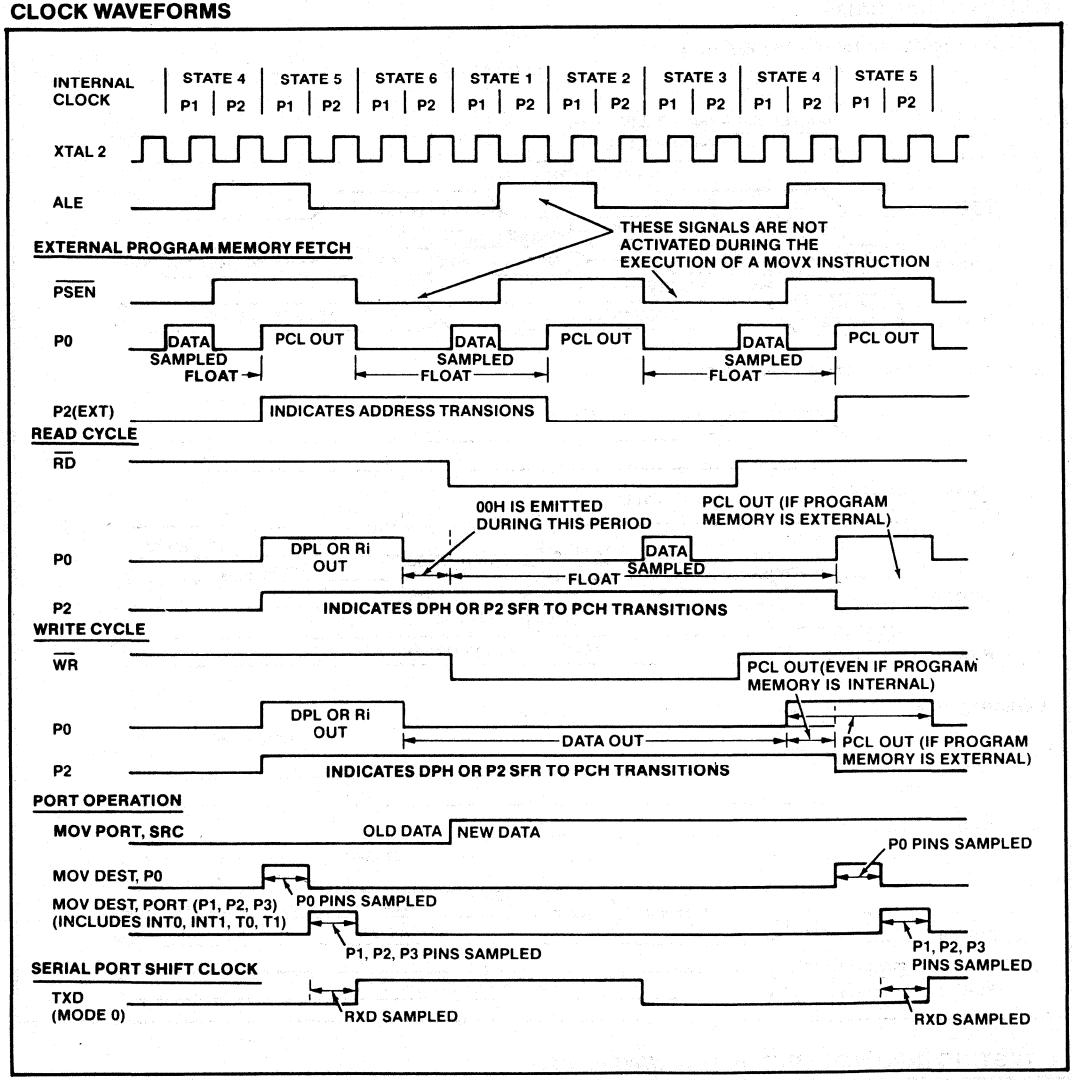


AC TESTING INPUT/OUTPUT, FLOAT WAVEFORMS



AC inputs during testing are driven at 2.4V for a logic "1" and 0.45V for a logic "0". Timing measurements are made at 2.0V for a logic "1" and 0.8V for a logic "0". For timing purposes, the float state is defined as the point at which a PO pin sinks 3.2 mA or sources 400μA at the voltage test levels.

CLOCK WAVEFORMS



This diagram indicates when signals are clocked internally. The time it takes the signals to propagate to the pins, however, ranges from 25 to 125 ns. This propagation delay is dependent on variables such as temperature and pin loading. Propagation also varies from output to output and component to component. Typically though, (T_A = 25° C, fully loaded) RD and WR propagation delays are approximately 50 ns. The other signals are typically 85 ns. Propagation delays are incorporated in the AC specifications.

Table 1. MCS®-51 Instruction Set Description

ARITHMETIC OPERATIONS

Mnemonic		Description	Byte	Cyc
ADD	A,Rn	Add register to Accumulator	1	1
ADD	A,direct	Add direct byte to Accumulator	2	1
ADD	A,@Ri	Add indirect RAM to Accumulator	1	1
ADD	A,#data	Add immediate data to Accumulator	2	1
ADDC	A,Rn	Add register to Accumulator with Carry	1	1
ADDC	A,direct	Add direct byte to A with Carry flag	2	1
ADDC	A,@Ri	Add indirect RAM to A with Carry flag	1	1
ADDC	A,#data	Add immediate data to A with Carry flag	2	1
SUBB	A,Rn	Subtract register from A with Borrow	1	1
SUBB	A,direct	Subtract direct byte from A with Borrow	2	1
SUBB	A,@Ri	Subtract indirect RAM from A with Borrow	1	1
SUBB	A,#data	Subtract immed data from A with Borrow	2	1
INC	A	Increment Accumulator	1	1
INC	Rn	Increment register	1	1
INC	direct	Increment direct byte	2	1
INC	@Ri	Increment indirect RAM	1	1
INC	DPTR	Increment Data Pointer	1	2
DEC	A	Decrement Accumulator	1	1
DEC	Rn	Decrement register	1	1
DEC	direct	Decrement direct byte	2	1
DEC	@Ri	Decrement indirect RAM	1	1
MUL	AB	Multiply A & B	1	4
DIV	AB	Divide A by B	1	4
DA	A	Decimal Adjust Accumulator	1	1

LOGICAL OPERATIONS

Mnemonic		Destination	Byte	Cyc
ANL	A,Rn	AND register to Accumulator	1	1
ANL	A,direct	AND direct byte to Accumulator	2	1
ANL	A,@Ri	AND indirect RAM to Accumulator	1	1
ANL	A,#data	AND immediate data to Accumulator	2	1
ANL	direct,A	AND Accumulator to direct byte	2	1
ANL	direct,#data	AND immediate data to direct byte	3	2
ORL	A,Rn	OR register to Accumulator	1	1
ORL	A,direct	OR direct byte to Accumulator	2	1

LOGICAL OPERATIONS (CONTINUED)

Mnemonic		Destination	Byte	Cyc
ORL	A,@Ri	OR indirect RAM to Accumulator	1	1
ORL	A,#data	OR immediate data to Accumulator	2	1
ORL	direct,A	OR Accumulator to direct byte	2	1
ORL	direct,#data	OR immediate data to direct byte	3	2
XRL	A,Rn	Exclusive-OR register to Accumulator	1	1
XRL	A,direct	Exclusive-OR direct byte to Accumulator	2	1
XRL	A,@Ri	Exclusive-OR indirect RAM to A	1	1
XRL	A,#data	Exclusive-OR immediate data to A	2	1
XRL	direct,A	Exclusive-OR Accumulator to direct byte	2	1
XRL	direct,#data	Exclusive-OR immediate data to direct	3	2
CLR	A	Clear Accumulator	1	1
CPL	A	Complement Accumulator	1	1
RL	A	Rotate Accumulator Left	1	1
RLC	A	Rotate A Left through the Carry flag	1	1
RR	A	Rotate Accumulator Right	1	1
RRC	A	Rotate A Right through Carry flag	1	1
SWAP	A	Swap nibbles within the Accumulator	1	1

DATA TRANSFER

Mnemonic		Description	Byte	Cyc
MOV	A,Rn	Move register to Accumulator	1	1
MOV	A,direct	Move direct byte to Accumulator	2	1
MOV	A,@Ri	Move indirect RAM to Accumulator	1	1
MOV	A,#data	Mov immediate data to Accumulator	2	1
MOV	Rn,A	Move Accumulator to register	1	1
MOV	Rn,direct	Move direct byte to register	2	2
MOV	Rn,#data	Move immediate data to register	2	1
MOV	direct,A	Move Accumulator to direct byte	2	1
MOV	direct,Rn	Move register to direct byte	2	2
MOV	direct,direct	Move direct byte to direct	3	2
MOV	direct,@Ri	Move indirect RAM to direct byte	2	2

Table 1. (Cont.)

DATA TRANSFER (CONTINUED)

Mnemonic		Description	Byte	Cyc
MOV	direct,#data	Move immediate data to direct byte	3	2
MOV	@Ri,A	Move Accumulator to indirect RAM	1	1
MOV	@Ri,direct	Move direct byte to indirect RAM	2	2
MOV	@Ri,#data	Move immediate data to indirect RAM	2	1
MOV	DPTR,#data16	Load Data Pointer with a 16-bit constant	3	2
MOVC	A,@A+DPTR	Move Code byte relative to DPTR to A	1	2
MOVC	A,@A+PC	Move Code byte relative to PC to A	1	2
MOVX	A,@Ri	Move External RAM (8-bit addr) to A	1	2
MOVX	A,@DPTR	Move External RAM (16-bit addr) to A	1	2
MOVX	@Ri,A	Move A to External RAM (8-bit addr)	1	2
MOVX	@DPTR,A	Move A to External RAM (16-bit addr)	1	2
PUSH	direct	Push direct byte onto stack	2	2
POP	direct	Pop direct byte from stack	2	2
XCH	A,Rn	Exchange register with Accumulator	1	1
XCH	A,direct	Exchange direct byte with Accumulator	2	1
XCH	A,@Ri	Exchange indirect RAM with A	1	1
XCHD	A,@Ri	Exchange low-order Digit ind RAM w A	1	1

BOOLEAN VARIABLE MANIPULATION

Mnemonic		Description	Byte	Cyc
CLR	C	Clear Carry flag	1	1
CLR	bit	Clear direct bit	2	1
SETB	C	Set Carry flag	1	1
SETB	bit	Set direct Bit	2	1
CPL	C	Complement Carry flag	1	1
CPL	bit	Complement direct bit	2	1
ANL	C,bit	AND direct bit to Carry flag	2	2
ANL	C,1 bit	AND complement of direct bit to Carry	2	2
ORL	C/bit	OR direct bit to Carry flag	2	2
ORL	C,1 bit	OR complement of direct bit to Carry	2	2
MOV	C/bit	Move direct bit to Carry flag	2	1
MOV	bit,C	Move Carry flag to direct bit	2	2

PROGRAM AND MACHINE CONTROL

Mnemonic		Description	Byte	Cyc
ACALL	addr11	Absolute Subroutine Call	2	2
LCALL	addr16	Long Subroutine Call	3	2
RET		Return from subroutine	1	2
RETI		Return from interrupt	1	2
AJMP	addr11	Absolute Jump	2	2
LJMP	addr16	Long Jump	3	2
SJMP	rel	Short Jump (relative addr)	2	2
JMP	@A+DPTR	Jump indirect relative to the DPTR	1	2
JZ	rel	Jump if Accumulator is Zero	2	2
JNZ	rel	Jump if Accumulator is Not Zero	2	2
JC	rel	Jump if Carry flag is set	2	2
JNC	rel	Jump if No Carry flag	2	2
JB	bit,rel	Jump if direct Bit set	3	2
JNB	bit,rel	Jump if direct Bit Not set	3	2
JBC	bit,rel	Jump if direct Bit is set & Clear bit	3	2
CJNE	A,direct,rel	Compare direct to A & Jump if Not Equal	3	2
CJNE	A,#data,rel	Comp, immed, to A & Jump if Not Equal	3	2
CJNE	Rn,#data,rel	Comp, immed, to reg & Jump if Not Equal	3	2
CJNE	@Ri,#data,rel	Comp, immed, to ind, & Jump if Not Equal	3	2
DJNZ	Rn,rel	Decrement register & Jump if Not Zero	2	2
DJNZ	direct,rel	Decrement direct & Jump if Not Zero	3	2
NOP		No operation	1	1

Notes on data addressing modes:

Rn —Working register R0-R7
direct —128 internal RAM locations, any I/O port, control or status register
@Ri —Indirect internal RAM location addressed by register R0 or R1
#data —8-bit constant included in instruction
#data16 —16-bit constant included as bytes 2 & 3 of instruction
bit —128 software flags, any I/O pin, control or status bit

Notes on program addressing modes:

addr16 —Destination address for LCALL & LJMP may be anywhere within the 64-K program memory address space
Addr11 —Destination address for ACALL & AJMP will be within the same 2-K page of program memory as the first byte of the following instruction
rel —SJMP and all conditional jumps include an 8-bit offset byte, Range is +127-128 bytes relative to first byte of the following instruction

Table 2. Instruction Opcodes in Hexadecimal Order

Hex Code	Number of Bytes	Mnemonic	Operands
00	1	NOP	
01	2	AJMP	code addr
02	3	LJMP	code addr
03	1	RR	A
04	1	INC	A
05	2	INC	data addr
06	1	INC	@R0
07	1	INC	@R1
08	1	INC	R0
09	1	INC	R1
0A	1	INC	R2
0B	1	INC	R3
0C	1	INC	R4
0D	1	INC	R5
0E	1	INC	R6
0F	1	INC	R7
10	3	JBC	bit addr, code addr
11	2	ACALL	code addr
12	3	LCALL	code addr
13	1	RRC	A
14	1	DEC	A
15	2	DEC	data addr
16	1	DEC	@R0
17	1	DEC	@R1
18	1	DEC	R0
19	1	DEC	R1
1A	1	DEC	R2
1B	1	DEC	R3
1C	1	DEC	R4
1D	1	DEC	R5
1E	1	DEC	R6
1F	1	DEC	R7
20	3	JB	bit addr, code addr
21	2	AJMP	code addr
22	1	RET	
23	1	RL	A
24	2	ADD	A,#data
25	2	ADD	A,data addr
26	1	ADD	A,@R0
27	1	ADD	A,@R1
28	1	ADD	A,R0
29	1	ADD	A,R1
2A	1	ADD	A,R2
2B	1	ADD	A,R3
2C	1	ADD	A,R4
2D	1	ADD	A,R5
2E	1	ADD	A,R6
2F	1	ADD	A,R7
30	3	JNB	bit addr, code addr
31	2	ACALL	code addr
32	1	RETI	
33	1	RLC	A
34	2	ADDC	A,#data
35	2	ADDC	A,data addr
36	1	ADDC	A,@R0
37	1	ADDC	A,@R1
38	1	ADDC	A,R0
39	1	ADDC	A,R1
3A	1	ADDC	A,R2
3B	1	ADDC	A,R3
3C	1	ADDC	A,R4
3D	1	ADDC	A,R5
3E	1	ADDC	A,R6
3F	1	ADDC	A,R7
40	2	JC	code addr
41	2	AJMP	code addr
42	2	ORL	data addr,A
43	3	ORL	data addr,#data
44	2	ORL	A,#data
45	2	ORL	A,data addr
46	1	ORL	A,@R0
47	1	ORL	A,@R1
48	1	ORL	A,R0
49	1	ORL	A,R1
4A	1	ORL	A,R2
4B	1	ORL	A,R3
4C	1	ORL	A,R4
4D	1	ORL	A,R5
4E	1	ORL	A,R6
4F	1	ORL	A,R7
50	2	JNC	code addr
51	2	ACALL	code addr
52	2	ANL	data addr,A
53	3	ANL	data addr,#data
54	2	ANL	A,#data
55	2	ANL	A,data addr
56	1	ANL	A,@R0
57	1	ANL	A@R1
58	1	ANL	A,R0
59	1	ANL	A,R1
5A	1	ANL	A,R2
5B	1	ANL	A,R3
5C	1	ANL	A,R4
5D	1	ANL	A,R5
5E	1	ANL	A,R6
5F	1	ANL	A,R7
60	2	JZ	code addr
61	2	AJMP	code addr
62	2	XRL	data addr.A
63	3	XRL	data addr,#data
64	2	XRL	A,#data
65	2	XRL	A,data addr

Table 2. (Cont.)

Hex Code	Number of Bytes	Mnemonic	Operands
66	1	XRL	A,@R0
67	1	XRL	A,@R1
68	1	XRL	A,R0
69	1	XRL	A,R1
6A	1	XRL	A,R2
6B	1	XRL	A,R3
6C	1	XRL	A,R4
6D	1	XRL	A,R5
6E	1	XRL	A,R6
6F	1	XRL	A,R7
70	2	JNZ	code addr
71	2	ACALL	code addr
72	2	ORL	C,bit addr
73	1	JMP	@A+DPTR
74	2	MOV	A,#data
75	3	MOV	data addr,#data
76	2	MOV	@R0,#data
77	2	MOV	@R1,#data
78	2	MOV	R0,#data
79	2	MOV	R1,#data
7A	2	MOV	R2,#data
7B	2	MOV	R3,#data
7C	2	MOV	R4,#data
7D	2	MOV	R5,#data
7E	2	MOV	R6,#data
7F	2	MOV	R7,#data
80	2	SJMP	code addr
81	2	AJMP	code addr
82	2	ANL	C,bit addr
83	1	MOVC	A,@A+PC
84	1	DIV	AB
85	3	MOV	data addr, data addr
86	2	MOV	data addr,@R0
87	2	MOV	data addr,@R1
88	2	MOV	data addr,R0
89	2	MOV	data addr,R1
8A	2	MOV	data addr,R2
8B	2	MOV	data addr,R3
8C	2	MOV	data addr,R4
8D	2	MOV	data addr,R5
8E	2	MOV	data addr,R6
8F	2	MOV	data addr,R7
90	3	MOV	DPTR,#data
91	2	ACALL	code addr
92	2	MOV	bit addr,C
93	1	MOVC	A,@A+DPTR
94	2	SUBB	A,#data
95	2	SUBB	A,data addr
96	1	SUBB	A,@R0
97	1	SUBB	A,@R1
98	1	SUBB	A,R0
99	1	SUBB	A,R1
9A	1	SUBB	A,R2
9B	1	SUBB	A,R3
9C	1	SUBB	A,R4
9D	1	SUBB	A,R5
9E	1	SUBB	A,R6
9F	1	SUBB	A,R7
A0	2	ORL	C,/bit addr
A1	2	AJMP	code addr
A2	2	MOV	C,bit addr
A3	1	INC	DPTR
A4	1	MUL	AB
A5		reserved	
A6	2	MOV	@R0,data adr
A7	2	MOV	@R1,data addr
A8	2	MOV	R0,data addr
A9	2	MOV	R1,data addr
AA	2	MOV	R2,data addr
AB	2	MOV	R3,data addr
AC	2	MOV	R4,data addr
AD	2	MOV	R5,data addr
AE	2	MOV	R6,data addr
AF	2	MOV	R7,data addr
B0	2	ANL	C,/bit addr
B1	2	ACALL	code addr
B2	2	CPL	bit addr
B3	1	CPL	C
B4	3	CJNE	A,#data,code addr
B5	3	CJNE	A,data addr,code addr
B6	3	CJNE	@R0,#data,code addr
B7	3	CJNE	@R1,#data,code addr
B8	3	CJNE	R0,#data,code addr
B9	3	CJNE	R1,#data,code addr
BA	3	CJNE	R2,#data,code addr
BB	3	CJNE	R3,#data,code addr
BC	3	CJNE	R4,#data,code addr
BD	3	CJNE	R5,#data,code addr
BE	3	CJNE	R6,#data,code addr
BF	3	CJNE	R7,#data,code addr
C0	2	PUSH	data addr
C1	2	AJMP	code addr
C2	2	CLR	bit addr
C3	1	CLR	C
C4	1	SWAP	A
C5	2	XCH	A,data addr
C6	1	XCH	A,@R0
C7	1	XCH	A,@R1
C8	1	XCH	A,R0
C9	1	XCH	A,R1
CA	1	XCH	A,R2
CB	1	XCH	A,R3

Table 2. (Cont.)

Hex Code	Number of Bytes	Mnemonic	Operands
CC	1	XCH	A,R4
CD	1	XCH	A,R5
CE	1	XCH	A,R6
CF	1	XCH	A,R7
D0	2	POP	data addr
D1	2	ACALL	code addr
D2	2	SETB	bit addr
D3	1	SETB	C
D4	1	DA	A
D5	3	DJNZ	data addr,code addr
D6	1	XCHD	A,@R0
D7	1	XCHD	A,@R1
D8	2	DJNZ	R0,code addr
D9	2	DJNZ	R1,code addr
DA	2	DJNZ	R2,code addr
DB	2	DJNZ	R3,code addr
DC	2	DJNZ	R4,code addr
DD	2	DJNZ	R5,code addr
DE	2	DJNZ	R6,code addr
DF	2	DJNZ	R7,code addr
E0	1	MOVX	A,@DPTR
E1	2	AJMP	code addr
E2	1	MOVX	A,@R0
E3	1	MOVX	A,@R1
E4	1	CLR	A
E5	2	MOV	A,data addr

Hex Code	Number of Bytes	Mnemonic	Operands
E6	1	MOV	A,@R0
E7	1	MOV	A,@R1
E8	1	MOV	A,R0
E9	1	MOV	A,R1
EA	1	MOV	A,R2
EB	1	MOV	A,R3
EC	1	MOV	A,R4
ED	1	MOV	A,R5
EE	1	MOV	A,R6
EF	1	MOV	A,R7
F0	1	MOVX	@DPTR,A
F1	2	ACALL	code addr
F2	1	MOVX	@R0,A
F3	1	MOVX	@R1,A
F4	1	CPL	A
F5	2	MOV	data addr,A
F6	1	MOV	@R0,A
F7	1	MOV	@R1,A
F8	1	MOV	R0,A
F9	1	MOV	R1,A
FA	1	MOV	R2,A
FB	1	MOV	R3,A
FC	1	MOV	R4,A
FD	1	MOV	R5,A
FE	1	MOV	R6,A
FF	1	MOV	R7,A

Application of Intel's 5V EPROM and ROM Family for Microprocessor Systems

Contents

INTRODUCTION 19-2

PINOUT EVOLUTION 19-2

SYSTEM ARCHITECTURE 19-2

BUS CONTENTION 19-3

THE MICROPROCESSOR/MEMORY INTERFACE ... 19-4

TERMINOLOGY 19-4

THE NEW INTEL FAMILY 19-5

PIN DEVICES AND PIN SITES 19-6

PRINTED CIRCUIT BOARD DESIGN 19-7

INTRODUCTION

This Application Note discusses how the new Intel family of 5 volt EPROMs and ROMs can be used with microprocessor systems. The pinout evolution and philosophy are explored in detail, which leads directly to system architecture. Particular emphasis will be placed on the pitfalls of bus contention and the microprocessor/memory interface. Finally, an actual printed circuit board layout is presented.

PINOUT EVOLUTION

As EPROM/ROM technology has evolved, there are often periods of confusion over EPROM and ROM pinouts, as ROM density usually leads EPROM density by a factor of two, but ultimately users want any given EPROM to have a ROM compatible part. As we have seen, after the 2716 16K EPROM was introduced, a new ROM pinout emerged and "triumphed" over an earlier "standard." The reason this ROM pinout change occurred is that as codes stabilize in user's systems and equipment, many users opt for the less expensive ROMs, which are mask programmable devices. At the same time, users often use the highest available density ROM so they combine modular firmware and minimize device count. Of course, many users never do go to the ROM stage with their equipment, preferring to minimize inventory levels and utilize standard designs that can be customized for final equipment configurations, but they always want the capability to do so if desired.

In addition, over the past few years, the development of microprocessors has been intimately entwined with both ROMs and EPROMS.

The 1702A and its ROM counterpart, the 1302, were completely adequate to support the requirements of the 4004 series of microprocessors. In order to support the 5 volt, 3MHz 8085A and 5MHz 8086, it is desirable to use a compatible device such as the Intel 5 volt 2716, whose 450ns access time is compatible with the microprocessor requirements. Some high performance versions of these processors may require selected versions of the 2716 (such as the 2716-1 with $t_{ACC} = 350ns$, or the 2716-2 with $t_{ACC} = 390ns$) depending on the actual system configuration.

Summarizing these events since the introduction of the Intel 1702A, which was the first EPROM, we can postulate the following hypothesis: at any point in time, the present EPROM determines the pinout for the next generation ROM. And, if the subsequent larger density EPROM is not ROM compatible, the ROM will change. Also, it can be seen that ROMs and EPROMs must evolve along with microprocessor developments—so memory performance does not limit system performance.

The devices which are discussed in this Application Note represent an extension of the 5 volt compatible family to 32K bit and 64K bit densities, while improving performance as discussed above. It also follows that the pinout for the 32K devices must be derived from the 2716 in ord to maintain socket compatibility. This 16K to 32K pino evolution is shown in Figure 1.

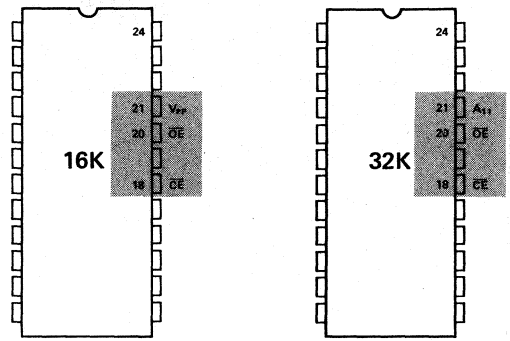


Figure 1. 16K EPROM Determines 32K ROM Pinout

SYSTEM ARCHITECTURE

As higher performance microprocessors have beco available, the architecture of microprocessor systems h been evolving, again placing demands on memory. F many years, system designers have been plagued with problem of bus contention when connecting multi memories to a common data bus. There have been vari schemes for avoiding the problem, but device manuf turers have been unable to design internal circuits t would guarantee that one memory device would be "o the bus before another device was selected. With sm memories (512x8 and 1Kx8), it has been traditional to c nect all the system address lines together and utilize the ference between t_{ACC} and t_{CO} to perform a decode select the correct device (as shown in Figure 2).

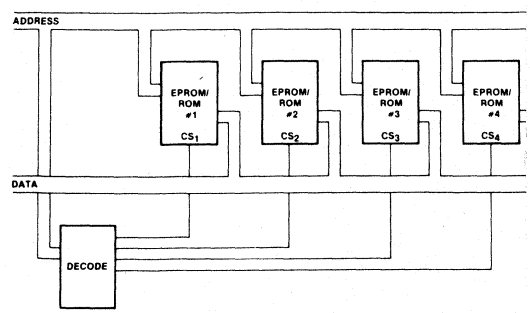


Figure 2. Single Control Line Architecture

With the 1702A, the chip select to output delay was only 100ns shorter than the address access time; or to state it another way, the t_{ACC} time was 1000ns while the t_{CO} time was 900ns. The 1702A t_{ACC} performance of 1000ns was suitable for the 4004 series microprocessors, but the 8080 processor required that the corresponding numbers be reduced to t_{ACC} = 450ns and t_{CO} = 120ns. This allowed a substantial improvement in performance over the 4004 series of microprocessors, but placed a substantial burden on the memory. The 2708 was developed to be compatible with the 8080 both in access time and power supply requirements. A portion of each 8080 machine cycle time had to be devoted to the architecture of the system decoding scheme used. This devoted portion of the machine cycle included the time required for the system controller (8224) to perform its function before the actual decode process could begin.

Let's pause here and examine the actual decode scheme that was used so we can understand how the control functions that a memory device requires are related to system architecture.

The 2708 can be used to illustrate the problem of having a single control line. The 2708 has only one read control function, chip select ($\overline{CS}$), which is very fast (t_{CO} = 120ns) with respect to the overall access time (t_{ACC} = 450ns) of the 2708. It is this time difference (330ns) that is used to perform the decode function, as illustrated in Figure 3. The scheme works well and does not limit system performance, but it does lead to the possibility of bus contention.

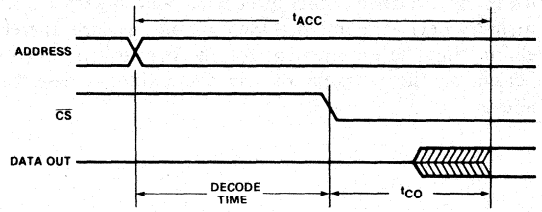


Figure 3. Single Line Control Architecture

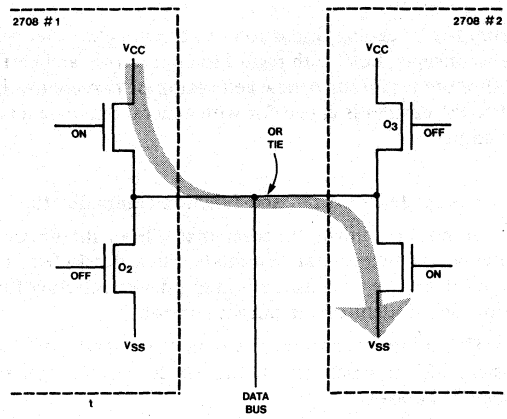


Figure 4. Results of Improper Timing when OR Tying Multiple Memories

BUS CONTENTION

There are actually two problems with the scheme described in the previous section. First, if one device in a multiple memory system has a relatively long deselect time, and a relatively fast decoder is used, it would be possible to have another device selected at the same time. If the two devices thus selected were reading opposite data; that is, device number one reading a HIGH and device number two reading a LOW, the output transistors of the two memory devices would effectively produce a short circuit, as Figure 4 illustrates. In this case, the current path is from V_{CC} on device number one to GND on device number two. This current is limited only by the "on" impedance of the MOS output transistors and can reach levels in excess of 200mA per device. If the MOS transistors have a lot of "extra" margin, the current is usually not destructive; however, an instantaneous load of 400mA can produce "glitches" on the V_{CC} supply — glitches large enough to cause standard TTL devices to drop bits or otherwise malfunction, thus causing incorrect address decode or generation.

The second problem with a single control line scheme is more subtle. As previously mentioned, there is only one control function available on the 2708 and any decoding scheme must use it out of necessity. In addition, any inadvertent changes in the state of the high order address lines that are inputs to the decoder will cause a change in the device that is selected. The result is the same as before — bus contention, only from a different source. The deselected device cannot get "off" the bus before the selected one is "on" the bus as the addresses rapidly change state. One approach to solving this problem would be to design (and specify as a maximum) devices with t_{DF} time less than t_{CO} time, thereby assuring that if one device is selected while another is simultaneously being deselected, there would be some small (20ns) margin. Even with this solution, the user would not be protected from devices which have very fast t_{CO} times (t_{CO} is specified as a maximum).

The only sure solution appears to be the use of an external bus driver/transceiver that has an independent enable function. Then that function, not the "device selecting function," or addresses, could control the flow of data "on" and "off" the bus, and any contention problems would be confined to a particular card or area of a large card. In fact, many systems are implemented that way — the use of bus drivers is not at all uncommon in large systems where the drive requirements of long, highly capacitive interconnecting lines must be taken into consideration — it also may be the reason why more system designers were not aware of the bus contention problem

until they took a previously large (multicard) system and, using an advanced microprocessor and higher density memory devices, combined them all on one card, thereby eliminating the requirement for the bus drivers, but experiencing the problem of bus contention as described above.

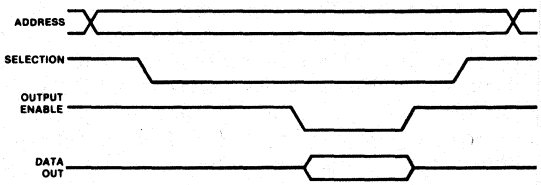


Figure 5. Two Control Line Architecture

THE MICROPROCESSOR/MEMORY INTERFACE

From the foregoing discussion, it becomes clear that some new concepts, both with regard to architecture and performance are required. A new generation of two control line EPROM devices is called for with general requirements as listed below:

1. Complete ROM pin and function compatibility.
2. A power control function that allows the device to enter a low-power standby mode when deselected. This function can be used as the primary device selecting function, independent of the output control.
3. Capability to control the data "on" and "off" the system bus, independent of the device selecting function identified above.
4. Access time compatible with the high performance microprocessors that are currently available.

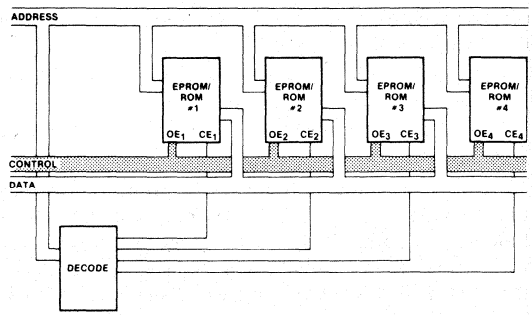


Figure 6. Two Control Line Architecture

Now let's examine the system architecture that is required to implement the two line control and prevent bus contention. This is shown in the form of a timing diagram (Figure 5). As before, addresses are used to generate the unique device selecting function, but a separate and independent Output Enable (OE) control is now used to gate data "on" and "off" the system data bus. With this scheme, bus contention is completely eliminated as the processor determines the time during which data must be present on the bus and then releases the bus by way of the Output Enable line, thus freeing the bus for use by other devices, either memories or peripheral devices. This type of architecture can be easily accomplished if the memory devices have two control functions, and the system is implemented according to the block diagram shown in Figure 6. It differs from the previous block diagram (shown in Figure 2) in that the control bus, which is connected to all memory Output Enable pins, provides separate and independent control over the data bus. In this way, the microprocessor is always in control of the system; while in the previous system, the microprocessor passed control to the particular memory device and then waited for data to become available. Another way to look at it is,with a single control line the system is always asynchronous with respect to microprocessor/memory communications. By using two control lines, the memory is synchronized to the processor.

TERMINOLOGY

Some of the terminology applied to the functions of t Intel 5 volt compatible family may be confusing or u familiar to many EPROM/ROM users, so the vario terms are defined here. Actually, the nomenclature w developed by various standards groups and is reiterat here to avoid confusion as we begin a detailed discussi of the devices themselves.

First of all, Chip Enable (CE) must be defined, as it is t primary device selection pin. By agreed standards, th function which substantially affects power dissipation called CE. Any memory device that has a CE function h both an active and standby power level associated with

Output Enable (OE) is the signal that controls the outp The fundamental purpose of OE is to provide a complet separate means of controlling the output buffer of t memory device, thereby eliminating bus contention.

Chip Select (CS) is a signal that gets logically ANDed wi addresses. In a completely static device, CS must rema stable throughout the entire device cycle, and its functi is equivalent to Output Enable (OE).

THE NEW INTEL FAMILY

Figure 7 shows the new Intel 5 volt compatible family of EPROMs and ROMs. In order to take advantage of the modular compatibility offered by the family, the functional compatibility of device pins 18, 19 and 21 must be understood. (Shaded area in Figure 7.)

First, we must examine the compatibility of the two oldest EPROM members of the 5 volt family — the 8K (2758) and the 16K (2716).

Pin 21 (V_{PP}) is normally connected to V_{CC} for read only applications of both devices, and pin 19 is either at GND (V_{IL}) for the 8K 2758 or connected to A_{10} for the 16K 2716. Further details on either of these devices can be found in Section 9 of the 1977 Edition of the Intel Memory Design Handbook, or Section 4 of the 1978 Intel Data Catalog.

The 32K (4Kx8) devices, which have identical pinouts for both the ROM and EPROM, will now be discussed. Pin 18 is $\overline{CE}$. Pin 19 is A_{10}, while pin 20 is $\overline{OE}$. As was pointed out before, Output Enable is the function which allows independent control of the data "on" and "off" the output bus. As Figure 7 indicates, V_{PP} (the programming voltage for the 2732 EPROM) is now multiplexed with $\overline{OE}$ on pin 20. Pin 21 becomes A_{11}, which is the additional address bit that is required as the density increases from 16K to 32K.

Pin 21 is the only pin that requires any special consideration when designing a system to accept the 8K, the 16K, or the 32K device. With the 8K and the 16K devices, pin 21 must be connected to V_{CC}, while with the 32K and higher density devices, it must be connected to A_{11}. This is easily accomplished by making sure the printed circuit trace links all pin 21's together as though they were an address line and allowing for a jumper that will connect pin 21 to either V_{CC} or A_{11} at the edge of the array (this technique can be seen in the "Printed Circuit Board Design" section and in Figure 8). Connecting the pin 21's together in this manner is acceptable as the read current requirement for V_{PP} is 4mA maximum per device — low enough to be handled by a signal trace, but too high for an address driver to provide directly.

The highest density member of the family is a 64K ROM which is also shown in Figure 7. In order to maintain total compatibility it is packaged in a standard 28-pin package.

It may seem as though the 28 pin package is not compatible with the rest of the family, but referring again to Figure 7, note that the lower 24 pins are identical to the 24 pin 8K, 16K and 32K devices. To allow for total compatibility within the family: printed circuit boards must be laid out to accommodate 28 pin sites; a jumper must be included to accommodate pin 21 as shown in Figure 8, and when using 64K devices, CS_2 (Pin 26) must be mask coded active high. This compatibility can also be seen graphically in Figures 9 and 10. The upper portion of the figure shows how 24 pin devices are used in the 28 pin sites. The two control lines ($\overline{CE}$ and $\overline{OE}$) remain unchanged as discussed earlier, and A_{12}, the next address bit required for a 64K bit device, is connected to pin 2 of the 28 pin site. The lower portion of the figure illustrates the use of 28 pin devices. Address bit A_{12} is already connected to the right pin, and the chip selects (CS_1 and CS_2) are connected to the V_{CC} power distribution grid. This configuration would require that both CS_1 and CS_2 be coded active high.

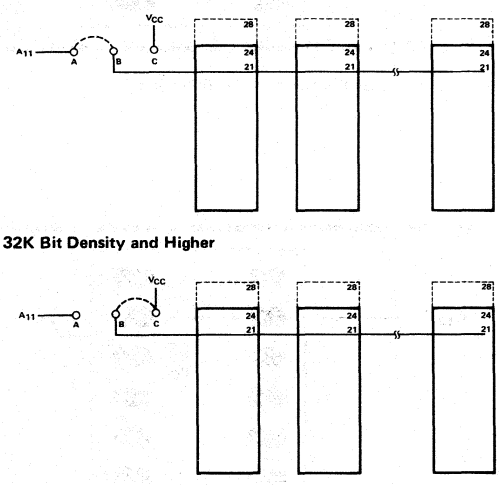


Figure 8. Pin 21 Connections for Various Density Devices

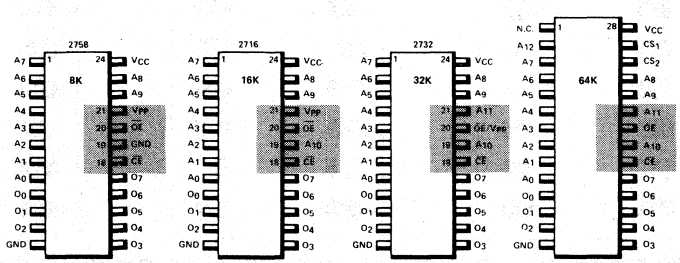
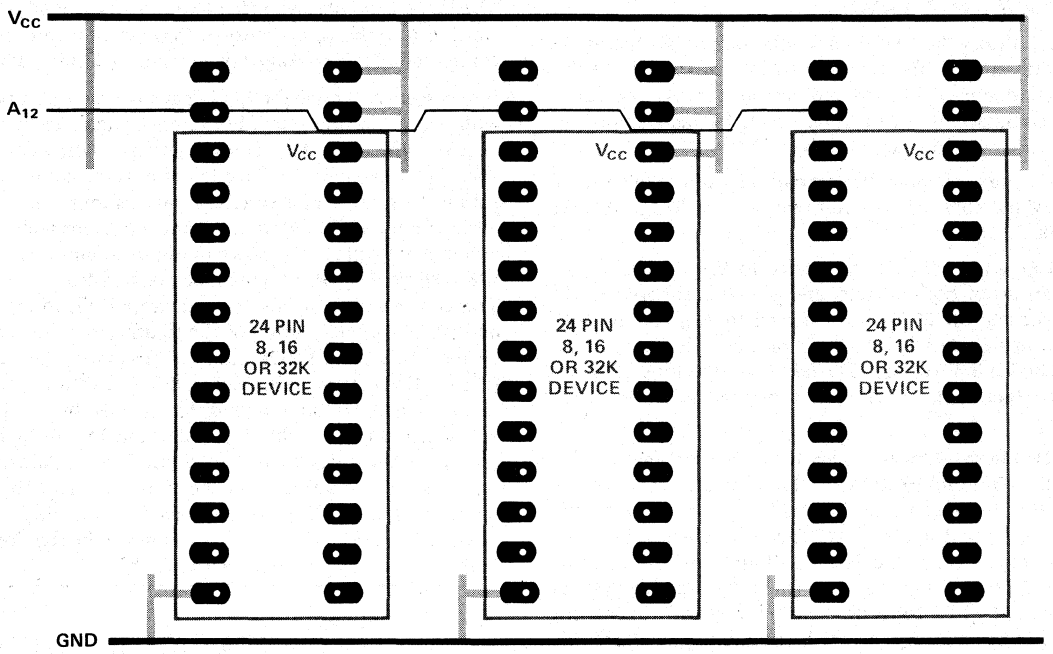
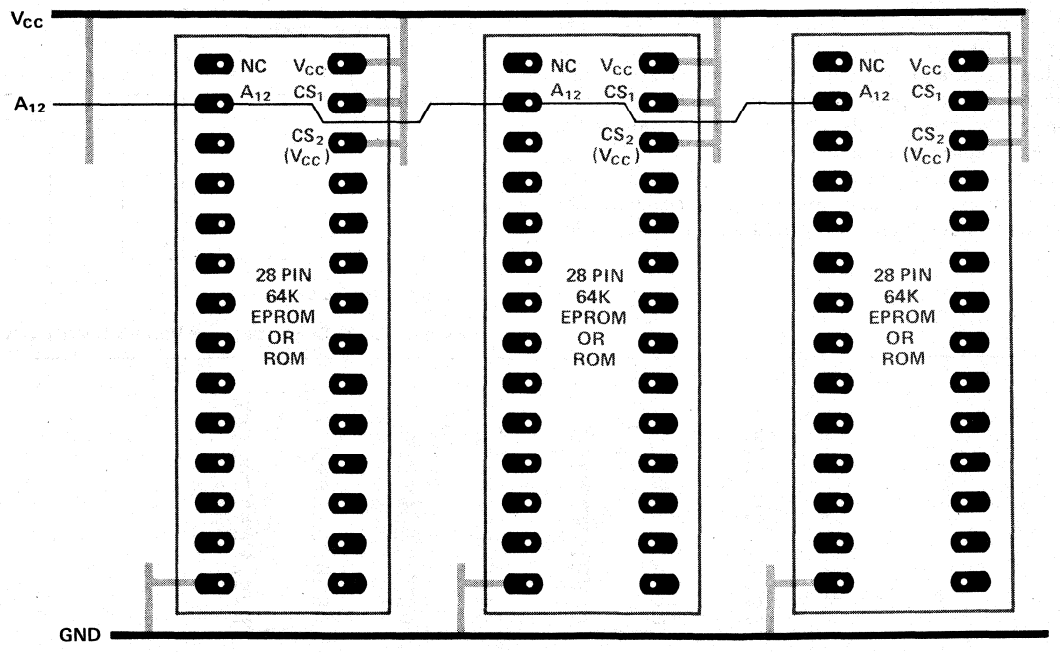


Figure 7. 5 Volt EPROM/ROM Compatible Family



24 Pin Devices and 28 Pin Sites



28 Pin Devices and 28 Pin Sites

Figure 9.

In some systems, additional logic may be used to implement a "special" decode of CS_1 to allow for ROM to overlap RAM when a system bootstrap program is being loaded; that additional logic should be implemented with CS_1; CS_2 should be coded active high in order to preserve total compatibility.

To summarize, the selection of a 28 pin package for 64K devices has several benefits of importance to present and future system designs:

1. Two line control philosophy (separate $\overline{CE}$ and $\overline{OE}$ functions) is preserved at the 64K bit level.
2. 64K EPROM compatibility is allowed for by maintaining a pin for the V_{PP} function.
3. The next generation (128K bit ROM) must be in a 28 pin package.

If CS_2 (pin 26) is mask coded to be active high and connected to V_{CC}, and the jumper provision for pin 21 is included on the card as described above, any member of the family can be plugged into the same socket — 1K, 2K, 4K or 8K bytes — without any card modification or redesign. In addition, future devices of higher density will fit in the same pinout.

PRINTED CIRCUIT BOARD DESIGN

The I_{CC} waveform for the 2332 and the 2364 is shown in Figure 10. The supply current, I_{CC}, has three segments that are of concern to the system designer — the standby level, active level and the transient peaks that are produced on the rising and falling edges of Chip Enable. The transient currents must be suppressed by properly selected decoupling capacitors. High quality, high frequency ceramic capacitors of small physical size with low inherent inductance should be used. In addition, bulk decoupling must be provided, usually near where the power supply is connected to the array. The purpose of the bulk decoupling is to overcome the voltage droop caused by the inductive effects of the PC board traces. Electrolytic or tantalum capacitors are suitable for bulk decoupling. The following capacitance values and locations are recommended for the 2332 and 2364:

1. A 0.1μF ceramic capacitor between V_{CC} and GND at every other device.

2. A 4.7μF electrolytic capacitor between V_{CC} and GND for each eight devices.

A printed circuit board layout for a total array of 16 devices is shown in Figure 11. This printed circuit layout incorporates a power supply distribution system such that the power supply and ground traces on the PC board are gridded both vertically and horizontally at each memory device; this technique minimizes the power distribution system impedance and enhances the effect of the decoupling capacitors. Provisions are included for all address inputs, output enable inputs, data outputs and decoded chip enable inputs. The 0.1μF capacitors referred to above are included for every other device (indicated by the legend C2) while the bulk decoupling capacitor is shown at the upper left-hand corner (indicated by the legend C1). The layout consists of four rows of four 28-pin device sites each and embodies all of the concepts explained above. Note that pins 28, 27 and 26 are all connected to V_{CC}. This requires that when ordering mask programmed 2364 64K ROMs, the order must specify that CS_1 and CS_2 be coded active HIGH. The single jumper provision discussed in the previous section is also included at the upper lefthand corner of the array (indicated by A, B, and C). Pad B is connected to pin 21 of all devices in the array; pad A should be connected to the A_{11} address driver and pad C is connected to V_{CC}. For use with 32K bit or larger devices, a jumper must be installed between pads A and B; for use with the 2716 (16K) or the 2758 (8K), the jumper must be installed between pads B and C.

A full size (2x) artwork film is included on the last page of this Application Note. The entire array, or segments of it can be photographed and used directly as part of a system board.

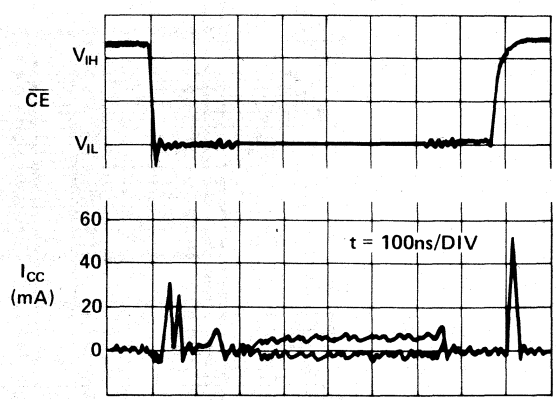


Figure 10. Typical I_{CC} Current vs Time

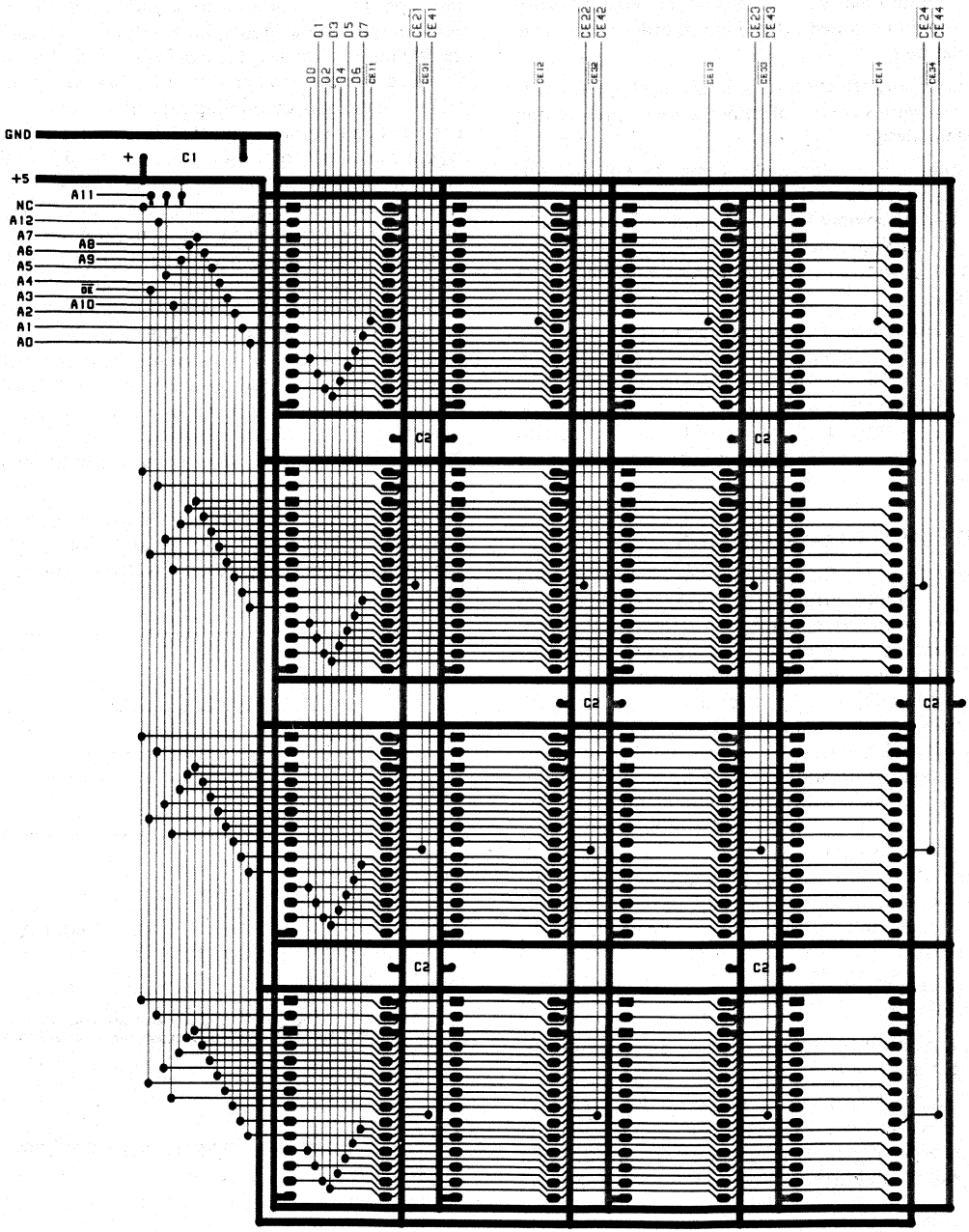


Figure 11. Printed Circuit Board Layout of 16 Devices

CRYSTALS: Specifications for Intel Components

Contents

INTRODUCTION 19-10

CRYSTAL OPERATION — BRIEF THEORETICAL EXPLANATION 19-10

CIRCUIT CONFIGURATIONS FROM VARIOUS MANUALS/EXPLANATION 19-11

Series 10 pF Capacitor 19-11
Parallel 20 pF Capacitor to Ground 19-11
8224 Overtone Application (Tank Circuit) 19-11
Precise Timing Applications 19-12

WHAT IF I USE A CRYSTAL OTHER THAN SPECIFIED? 19-12

SPECIFICATIONS 19-12

Intel Component Crystal Requirements 19-12
Suggested Suppliers, Part Numbers 19-13

INTRODUCTION

The following brief note is intended to answer the simpler questions on crystal specifications and their operation with the various Intel components. First, a theoretical explanation of the crystal is given to aid the user in understanding crystal operation. This includes a discussion of the parameters necessary for proper specification to the vendor. Following this section are explanations of the various crystal-capacitor configurations seen in the Intel User's Manuals and data sheets; why they are suggested for proper crystal operation and what might happen if they weren't there.

The final section of this note provides a list of suggested crystal specifications, suppliers, and part numbers for the highest frequency crystals possible for the various Intel components that require them. In no way does this list represent the only crystals or suppliers available. This section is conveniently preceded by a discussion of problem areas that may result if a user is using the wrong crystal required for the component.

CRYSTAL OPERATION — BRIEF THEORETICAL EXPLANATION

Understanding Crystal Operation

Crystals are piezoelectric devices which transform voltage energy to mechanical vibrations and voltage oscillations. The frequency of the crystal is largely dependent on its thickness, with thinner crystals producing a higher frequency.

Crystals are generally specified as being series or parallel resonant, but all crystals are in actuality both. Vendors supply crystals as series or parallel resonant based on the desired frequency and the crystal's relative ability to generate the frequency in that mode. On a conceptual basis, when using a crystal as series resonant, its output is in phase with its input, whereas using the crystal as parallel resonant will result in a phase shift from its input to output.

Different LSI components prefer different crystals due to the nature of their internal oscillator design. In general, Intel bipolar components have a non-inverting, bidirectional drive oscillator, whereas NMOS components use an inverting oscillator. Non-inverting oscillators prefer series resonant crystals (as the series resonant crystal has 0 degree net phase shift), while inverting oscillators prefer crystals which are parallel resonant. Since a crystal has both a series and parallel operating frequency, many times any crystal will seem to work when connected to a component.

When giving the specifications to a crystal vendor for a crystal, it is helpful to understand its equivalent circuit as shown in Figure 1. The impedance of this circuit (neglecting R to simplify matters for conceptual purposes) can be calculated and plotted against frequency (Figure 2). This frequency-impedance plot illustrates the two different operating modes of crystal. ω_s (series resonance) occurs when the impedance (reactance) is zero and ω_p (parallel resonance) occurs when the impedance goes to infinity and appears inductive.

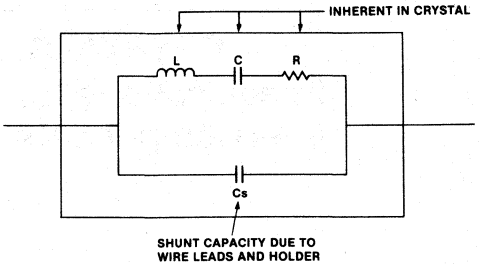
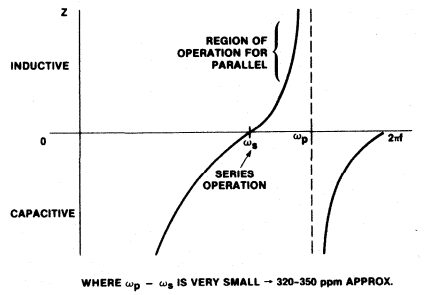


Figure 1

$$Z = \frac{(1/SC + SL)\,1/SC_s}{1/SC + SL + 1/SC_s} = \frac{-j(\omega^2 - \omega_s^2)}{\omega C_s(\omega^2 - \omega_p^2)} \quad \text{WHERE } \omega_s = 1/\sqrt{LC} \quad \omega_p = 1/\sqrt{L[(CC_s)/(C + C_s)]}$$



WHERE $\omega_p - \omega_s$ IS VERY SMALL → 320-350 ppm APPROX.

Figure 2

When operating at series resonance (ω_s) the equivalent circuit of the crystal becomes a simple resistor Rs (Figure 3; remember, R was neglected in the impedance calculation). This Rs value must be specified to the crystal vendor when buying a crystal.

This parameter becomes a problem with lower frequency or overtone crystals (thicker, more resistance) and a buffer that doesn't have sufficient gain to drive those crystals (i.e., loop gain becomes less than 1). Overtone crystals also have Rs problems as their Rs is associated with the fundamental frequency of the crystal, not the 3rd harmonic or overtone. The 8224 is particularly sensitive to Rs with 27 MHz overtone applications.

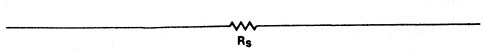


Figure 3

Conversely, if operating at ωp (parallel resonance), the crystal appears inductive in the circuit (Figure 4). Since the crystal appears inductive, any changes in reactance that the crystal sees will have the effect of pulling the frequency of the crystal. As a result of this, the amount of load capacitance seen by the crystal in the circuit configuration becomes important. This load capacitance, CL, is the dynamic capacity of the total circuit measured across the terminals of the crystal. The amount of this capacitance should always be specified to the crystal vendor if the crystal will be operating at parallel resonance.

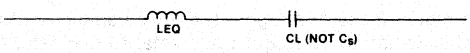


Figure 4

CIRCUIT CONFIGURATIONS FROM VARIOUS MANUALS/EXPLANATIONS

Series 10 pF Capacitor Included (Figure 5)

This additional capacitor is recommended at times to debias the crystal. Due to the component's internal circuit, a small DC bias may exist across the crystal which would strain the crystalline structure. It is also provided for trimming the frequency of the crystal to compensate for the loading effects of the component.

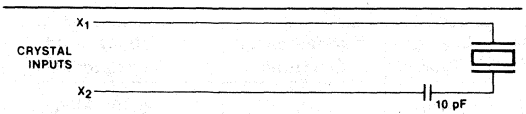


Figure 5

Parallel 20 pF Capacitors to Ground (Figure 6)

Crystals can oscillate at several different frequencies, each emanating from a different direction of vibration in the crystal. For a crystal to oscillate during startup in its fundamental frequency, it is best for the crystal to see the slew rate (Figure 7) of the pulse provided from the oscillator to be as close to the operating frequency as possible.

These 20 pF capacitors act as a high frequency filter to create a slew rate closer to the fundamental frequency of the crystal. As can be guessed, lower frequency crystals are more susceptible to the problem of not starting up in the fundamental frequency.

Capacitors are placed on both sides of the crystal as some components have bidirectional drive buffers (i.e., 1/2 of cycle drive from one side, other half from opposite side). A crystal that needs these extra 20 pFs to ground will be characterized by starting up at a 3rd or 5th harmonic instead of the fundamental frequency. The CL specifications in the specification section takes into consideration these extra 20 pF capacitors required for some Intel components for proper operation.

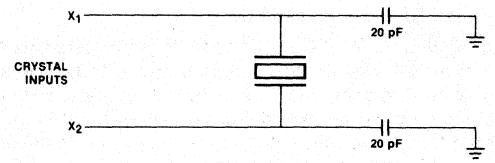


Figure 6

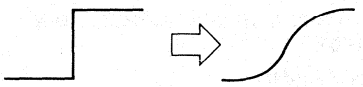


Figure 7

Tank Circuit

On some Intel components, provision is made for a tank circuit. This is for the use of an overtone crystal; i.e., one that is working at a harmonic (generally its 3rd). The tank circuitry is a filter to bypass the lower and higher, unwanted frequencies to ground while appearing "open" to the desired frequency. It is necessary to use tank circuits and overtone crystals when in the 25+ MHz range and above. Fundamental crystals are difficult to make in this frequency range as the crystal must be thinner for higher frequencies.

A circuit that has been used for the 8224 in 27 MHz overtone crystal applications is shown in Figure 8.

This filter can be approximated through formulas where afterwards it will be necessary to tweak the component values for optimization. The formula used to get the original component values is:

$$f = \frac{1}{2\pi\sqrt{L_1C_1}} \quad \text{where } f = \text{overtone frequency}$$

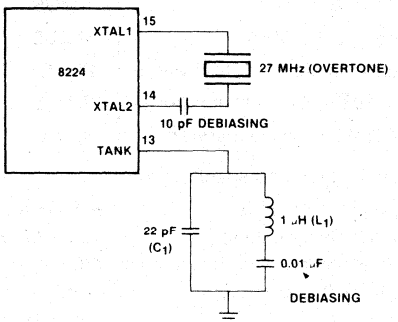


Figure 8

Precise Timing Applications

For applications where precise timing is required, using an external drive could produce better results. The accuracy of the component clock over temperature will be as accurate as the external drive. It is difficult to guarantee the temperature stability of the output frequency of the Intel component as fabrication process parameters vary, causing large ranges of input impedance and hence a large range of loading for the crystal.

WHAT IF I USE A CRYSTAL OTHER THAN SPECIFIED?

Series vs. Parallel

As discussed in the theoretical section, all crystals have a series and parallel operating mode. Placing a series crystal on a device requiring a parallel (i.e., there is an inverting oscillator between the two inputs) will force the crystal to oscillate in its parallel mode (and vice versa). A system with the wrong crystal will exhibit its clock frequency shifted a small percentage (about 320 to 350 parts per million) from the specified crystal frequency. When using the wrong crystal, any attempts to trim the frequency to its specified value by using small parallel (series if series crystal) variable capacitors will cause the crystal to stop oscillating, as predicted by theory. If the correct crystal is being used, trimming can be done.

In applications where accuracy is not important, series crystals are sometimes substituted for parallel in the circuit. For instance, the 8048 has been characterized to be compatible with the series color burst TV crystal (3.579545 MHz). If this crystal is used, a small frequency shift will occur, as noted above.

Insufficient Drive Level

The drive level specified is the maximum amount of power that is expected for the crystal to dissipate. If the crystal can't handle this level, frequency drift may occur or possible fracture of the crystal. In other words, if the crystal used cannot handle the oscillator drive level, long term reliability problems may occur.

Rs Too High

The higher Rs is, the higher the drive capability of the oscillator has to be to get the crystal to oscillate. Too much Rs may result in the oscillator not being able to drive the crystal; i.e., the loop gain is less than one. Overtone applications are particularly sensitive to this as thicker crystals are used (lower fundamental frequency, more resistance).

SPECIFICATIONS

Intel Component Crystal Requirements

The following is a list of suggested specifications for crystals to be used with Intel components. In most instances the upper frequency limit is given, with exceptions being footnoted.

Component (Function)	Process	Component Divide By	Crystal Type	Fundamental Overtone	Upper Limit Frequency
1. 4201A (Clock Generator)	CMOS	—	Series	f	5.185 MHz
2. 8035/48/49, 8748 (8-Bit CPU)	NMOS	15	Parallel	f	6.0 MHz
3. 8748/8035-8 (8-Bit CPU)	NMOS	15	Parallel	f	3.6 MHz
4. 8041/8741 (Universal Peripheral Interface)	NMOS	15	Parallel	f	6.0 MHz
5. 8085A (8-Bit CPU)	NMOS	2	Parallel	f	6.25 MHz/6.144 MHz(1)
6. 8085A-2 (8-Bit CPU)	NMOS	2	Parallel	f	10.0 MHz
7. 8202 (Dynamic RAM Controller)	Bipolar	—	Series	f	25 MHz
8. 8224 (8080A Clock Generator)	Bipolar	—	Series	f/o	27 MHz/18.432 MHz(2)
9. 8284 (8086 Clock Generator)	Bipolar	3	Series	f	24 MHz/15 MHz(3)

Additional suggested specifications:

Frequency Tolerance:	± 0.005% (up to the user)
CL (Load Capacitance):	= 20-35 pF (not necessary when specifying series)
Rs (Equivalent Series Resistance):	<75 ohms
Cs (Shunt Capacitance):	<7 pF
Drive Level:	<10 MHz crystal 10 milliwatts >10 MHz crystal 5 milliwatts

Notes: 1. 6.144 MHz is commonly used as convenient baud rates can be generated from this frequency.

2. 27 MHz is max. 18.432 is common crystal used which gives maximum clock rate for 8080A. Fundamental crystal should be used for the 18.432 MHz application.

3. Used for either a 8 or 5 MHz output clock, respectively.

Holder specifications are up to the user. A standard popular one that provides ample lead length is HC-33/U (0.750"W x 0.765"H, 1.5" lead length with spacing of 0.486") and can be used for frequencies up to 4 MHz. After 4 MHz a smaller holder can be used such as HC-18/U (0.435"W x 0.530"H, 1.5" lead length with spacing of 0.192"). All crystals listed in the following table will fit in the HC-33/U holder. Other standard holders are available.

Suggested Suppliers, Part Numbers

The following are two vendors (which are among many) that supply crystals to the specifications given earlier and their part numbers (given in order of frequency). The user should make sure that the holder type associated with these part numbers is acceptable in their application.

f	Parallel/ Series	Crystek[1] Corp.	CTS Knight,[2] Inc.
3.6 MHz	P	**	**
5.185 MHz	S	CY8A	**
6.0 MHz	P	**	MP060
6.144 MHz	P	**	MP061
6.25 MHz	P	**	MP062
10.0 MHz	P	**	MP10A
15.0 MHz	S	CY15A	MP150
18.432	S	CY19B*	MP184*
24.0 MHz	S	**	MP240
25.0 MHz	S	**	MP250
27.0 MHz	S (overtone)	CY27A	MP270

*Intel also supplies a crystal numbered 8801 for this application.

**Contact vendor with the appropriate specifications.

Notes: 1. Address: 1000 Crystal Drive, Fort Meyers, Florida 33901
2. Address: 400 Reimann Ave., Sandwich, Illinois

The user is not limited to these vendors or frequencies. The frequency chosen by the user should take into consideration convertibility to desired baud rates and the system timings that must be met.

In summary, to obtain a crystal for the user's application, it is necessary to give the crystal vendor the following information:

Series or parallel
Fundamental or overtone
Rs (series), Cs (shunt)
CL if parallel
Drive Level
Frequency tolerance
Holder type

For a select few crystals, vendor numbers were given for two different vendors. With the above information, most vendors can make the desired crystal whether or not they have it as a standard part.

U.S. SALES OFFICES

ALABAMA

Intel Corp.
303 Williams Avenue, S.W.
Suite 1422
Huntsville 35801
Tel: (205) 533-9353

ARIZONA

Intel Corp.
11225 N. 28th Drive
Suite 214D
Phoenix 85029
Tel: (602) 869-4980

CALIFORNIA

Intel Corp.
1010 Hurley Way
Suite 300
Sacramento 95825
Tel: (916) 929-4078

Intel Corp.
7670 Opportunity Road
Suite 135
San Diego 92111
Tel: (714) 268-3563

Intel Corp.*
2000 East 4th Street
Suite 100
Santa Ana 92705
Tel: (619) 835-9642
TWX: 910-595-1114

Intel Corp.*
1350 Shorebird Way
Mt. View 94043
Tel: (415) 968-8086
TWX: 910-339-9279
910-338-0255

Intel Corp.*
5530 Corbin Avenue
Suite 120
Tarzana 91356
Tel: (213) 708-0333
TWX: 910-495-2045

COLORADO

Intel Corp.
4445 Northpark Drive
Suite 100
Colorado Springs 80907
Tel: (303) 594-6622

Intel Corp.*
650 S. Cherry Street
Suite 720
Denver 80222
Tel: (303) 321-8086
TWX: 910-931-2289

CONNECTICUT

Intel Corp.
36 Padanaram Road
Danbury 06810
Tel: (203) 792-8366
TWX: 710-456-1199

EMC Corp.
393 Center Street
Wallingford 06492
Tel: (203) 265-6991

FLORIDA

Intel Corp.
1500 N.W. 62nd Street
Suite 104
Ft. Lauderdale 33309
Tel: (305) 771-0600
TWX: 510-956-9407

Intel Corp.
500 N. Maitland
Suite 205
Maitland 32751
Tel: (305) 628-2393
TWX: 810-853-9219

GEORGIA

Intel Corp.
3300 Holcombe Bridge Road
Suite 225
Norcross 30092
Tel: (404) 449-0541

ILLINOIS

Intel Corp
2550 Golf Road,
Suite 815
Rolling Meadows 60008
Tel: (312) 981-7200
TWX: 910-651-5881

INDIANA

Intel Corp.
9100 Purdue Road
Suite 400
Indianapolis 46268
Tel: (317) 875-0623

IOWA

Intel Corp.
St. Andrews Building
1930 St. Andrews Drive N.E.
Cedar Rapids 52402
Tel: (319) 393-5510

KANSAS

Intel Corp.
8400 W. 110th Street
Suite 170
Overland Park 66210
Tel: (913) 642-8080

LOUISIANA

Industrial Digital Systems Corp.
2332 Severn Avenue
Suite 202
Metairie, LA 70001
Tel: (504) 831-8492

MARYLAND

Intel Corp.*
7257 Parkway Drive
Hanover 21076
Tel: (301) 796-7500
TWX: 710-862-1944

Intel Corp.
7833 Walker Drive
Greenbelt 20770
Tel: (301) 431-1200

MASSACHUSETTS

Intel Corp.*
27 Industrial Avenue
Chelmsford 01824
Tel: (617) 256-1800
TWX: 710-343-6333

EMC Corp.
385 Elliot Street
Newton 02164
Tel: (617) 244-4740
TWX: 922531

MICHIGAN

Intel Corp.*
26500 Northwestern Hwy.
Suite 401
Southfield 48075
Tel: (313) 353-0920
TWX: 810-244-4915

MINNESOTA

Intel Corp.
3500 W. 80th Street
Suite 360
Bloomington 55431
Tel: (612) 835-6722
TWX: 910-576-2867

MISSOURI

Intel Corp.
4203 Earth City Expressway
Suite 131
Earth City 63045
Tel: (314) 291-1990

NEW JERSEY

Intel Corp.*
Raritan Plaza III
Raritan Center
Edison 08837
Tel: (201) 225-3000
TWX: 710-480-6238

NEW MEXICO

Intel Corp.
1120 Juan Tabo N.E.
Albuquerque 87112
Tel: (505) 292-8086

NEW YORK

Intel Corp.*
300 Vanderbilt Motor Parkway
Hauppauge 11788
Tel: (516) 231-3300
TWX: 510-227-6236

Intel Corp.
80 Washington Street
Poughkeepsie 12601
Tel: (914) 473-2303
TWX: 510-248-0060

Intel Corp.*
211 White Spruce Boulevard
Rochester 14623
Tel: (716) 424-1050
TWX: 510-253-7391

T-Squared
6443 Ridings Road
Syracuse 13206
Tel: (315) 463-8592
TWX: 710-541-0554

T-Squared
7353 Pittsford
Victor Road
Victor 14564
Tel: (716) 924-9101
TWX: 510-254-8542

NORTH CAROLINA

Intel Corp.
2306 W. Meadowview Road
Suite 206
Greensboro 27407
Tel: (919) 294-1541

OHIO

Intel Corp.*
6500 Poe Avenue
Dayton 45414
Tel:(513) 890-5350
TWX: 810-450-2528

Intel Corp.*
Chagrin-Brainard Bldg., No. 300
28001 Chagrin Boulevard
Cleveland 44122
Tel: (216) 464-6915
TWX: 810-427-9298

OKLAHOMA

Intel Corp.
4157 S. Harvard Avenue
Suite 123
Tulsa 74135
Tel: (918) 749-8688

OREGON

Intel Corp.
10700 S.W. Beaverton
Hillsdale Highway
Suite 22
Beaverton 97005
Tel: (503) 641-8086
TWX: 910-467-8741

PENNSYLVANIA

Intel Corp.*
510 Pennsylvania Avenue
Fort Washington 19034
Tel: (215) 641-1000
TWX: 510-661-2077

Intel Corp.*
201 Penn Center Boulevard
Suite 301W
Pittsburgh 15235
Tel: (412) 823-4970

Q.E.D. Electronics
300 N. York Road
Hatboro 19040
Tel: (215) 674-9600

TEXAS

Intel Corp.*
12300 Ford Road
Suite 380
Dallas 75234
Tel: (214) 241-8087
TWX: 910-860-5617

Intel Corp.*
7322 S.W. Freeway
Suite 1490
Houston 77074
Tel: (713) 988-8086
TWX: 910-881-2490

Industrial Digital Systems Corp.
5925 Sovereign
Suite 101
Houston 77036
Tel: (713) 988-9421

Intel Corp.
313 E. Anderson Lane
Suite 314
Austin 78752
Tel: (512) 454-3628

UTAH

Intel Corp.
268 West 400 South
Salt Lake City 84101
Tel: (801) 533-8086

VIRGINIA

Intel Corp.
1603 Santa Rosa Road
Suite 109
Richmond 23288
Tel: (804) 282-5668

WASHINGTON

Intel Corp.
110 110th Avenue N.E.
Suite 510
Bellevue 98004
Tel: (206) 453-8086
TWX: 910-443-3002

WISCONSIN

Intel Corp.
450 N. Sunnyslope Road
Suite 130
Brookfield 53005
Tel: (414) 784-9060

*Field Application Location

U.S. DISTRIBUTORS

ALABAMA

†Arrow Electronics, Inc.
3611 Memorial Parkway So.
Huntsville 35405
Tel: (205) 882-2730

†Hamilton/Avnet Electronics
4812 Commercial Drive N.W.
Huntsville 35805
Tel: (205) 837-7210
TWX: 810-726-2162

†Pioneer/Huntsville
1207 Putnam Drive N.W.
Huntsville 35805
Tel: (205) 837-9300
TWX: 810-726-2197

ARIZONA

†Hamilton/Avnet Electronics
505 S. Madison Drive
Tempe 85281
Tel: (602) 231-5140
TWX: 910-950-0077

†Wyle Distribution Group
8155 N. 24th Street
Phoenix 85021
Tel: (602) 249-2232
TWX: 910-951-4282

CALIFORNIA

†Arrow Electronics, Inc.
521 Weddell Drive
Sunnyvale 94086
Tel: (408) 745-6600
TWX: 910-339-9371

†Arrow Electronics, Inc.
19748 Dearborn Street
Chatsworth 91311
Tel: (213) 701-7500
TWX: 910-493-2086

†Hamilton/Avnet Electronics
350 McCormick Avenue
Costa Mesa 92626
Tel: (714) 754-6051
TWX: 910-595-1928

†Hamilton/Avnet Electronics
19515 So. Vermont Avenue
Torrance 90502
Tel: (213) 615-3909
TWX: 910-349-6263

†Hamilton/Avnet Electronics
1175 Bordeaux Drive
Sunnyvale 94086
Tel: (408) 743-3300
TWX: 910-339-9332

†Hamilton/Avnet Electronics
4545 Viewridge Avenue
San Diego 92123
Tel: (714) 641-4109
TWX: 910-595-2638

†Hamilton/Avnet Electronics
10912 W. Washington Boulevard
Culver City 90230
Tel: (213) 558-2458
TWX: 910-340-6364

†Hamilton/Avnet Electronics
21050 Erwin Street
Woodland Hills 91367
Tel: (213) 883-0000
TWX: 910-494-2207

†Hamilton Electro Sales
3170 Pullman Street
Costa Mesa 92626
Tel: (714) 641-4109
TWX: 910-595-2638

†Hamilton/Avnet Electronics
4103 Northgate Boulevard
Sacramento 95834
Tel: (916) 920-3150

Kierulff Electronics, Inc.
3969 E. Bayshore Road
Palo Alto 94303
Tel: (415) 968-6292
TWX: 910-379-6430

Kierulff Electronics, Inc.
14101 Franklin Avenue
Tustin 92680
Tel: (714) 731-5711
TWX: 910-595-2599

Kierulff Electronics, Inc.
2585 Commerce Way
Los Angeles 90040
Tel: (213) 725-0325
TWX: 910-580-3666

†Wyle Distribution Group
124 Maryland Street
El Segundo 90245
Tel: (213) 322-8100
TWX: 910-348-7140 or 7111

†Wyle Distribution Group
9525 Chesapeake Drive
San Diego 92123
Tel: (714) 565-9171
TWX: 910-335-1590

†Wyle Distribution Group
3000 Bowers Avenue
Santa Clara 95051
Tel: (408) 727-2500
TWX: 910-338-0451 or 0451/0

†Wyle Distribution Group
17872 Cowan Avenue
Irvine 92714
Tel: (714) 641-1600
TWX: 910-595-1572

COLORADO

†Wyle Distribution Group
451 E. 124th Avenue
Thornton 80241
Tel: (303) 457-9953
TWX: 910-936-0770

†Hamilton/Avnet Electronics
8765 E. Orchard Road
Suite 708
Englewood 80111
Tel: (303) 740-1017
TWX: 910-935-0787

CONNECTICUT

†Arrow Electonics, Inc.
12 Beaumont Road
Wallingford 06492
Tel: (203) 265-7741
TWX: 710-220-1684

†Hamilton/Avnet Electronics
Commerce Industrial Park
Commerce Drive
Danbury 06810
Tel: (203) 797-2800
TWX: 710-456-9974

†Harvey Electronics
112 Main Street
Norwalk 06851
Tel: (203) 853-1515
TWX: 710-468-3373

FLORIDA

†Arrow Electronics, Inc.
1001 N.W. 62nd Street
Suite 108
Ft. Lauderdale 33309
Tel: (305) 776-7790
TWX: 510-955-9456

†Arrow Electronics, Inc.
50 Woodlake Drive W.
Bldg. B
Palm Bay 32905
Tel: (305) 725-1480
TWX: 510-959-6337

†Hamilton/Avnet Electronics
6801 N.W. 15th Way
Ft. Lauderdale 33309
Tel: (305) 971-2900
TWX: 510-956-3097

†Hamilton/Avnet Electronics
3197 Tech. Drive North
St. Petersburg 33702
Tel: (813) 576-3930
TWX: 810-863-0374

†Pioneer/Orlando
6220 S. Orange Blossom Trail
Suite 412
Orlando 32809
Tel: (305) 859-3600
TWX: 810-850-0177

†Pioneer/Ft. Lauderdale
1500 62nd Street N.W.
Suite 506
Ft. Lauderdale 33309
Tel: (305) 771-7520
TWX: 510-955-9653

GEORGIA

†Arrow Electronics, Inc.
2979 Pacific Drive
Norcross 30071
Tel: (404) 449-8252
TWX: 810-766-0439

†Hamilton/Avnet Electronics
5825 D. Peachtree Corners
Norcross 30092
Tel: (404) 447-7500
TWX: 810-766-0432

†Pioneer/Georgia
5835B Peachtree Corners E
Norcross 30092
Tel: (404) 448-1711
TWX: 810-766-4515

ILLINOIS

†Arrow Electronics, Inc.
2000 E. Alonquin Street
Schaumberg 60195
Tel: (312) 397-3440
TWX: 910-291-3544

†Hamilton/Avnet Electronics
1130 Thorndale Avenue
Bensenville 60106
Tel: (312) 860-7780
TWX: 910-227-0060

†Pioneer/Chicago
1551 Carmen Drive
Elk Grove Village 60007
Tel: (312) 437-9680
TWX: 910-262-1182

INDIANA

†Arrow Electronics, Inc.
2718 Rand Road
Indianapolis 46241
(317) 243-9353
TWX: 810-341-3119

†Hamilton/Avnet Electronics
485 Gradle Drive
Carmel 46032
Tel: (317) 844-9333
TWX: 810-260-3966

†Pioneer/Indiana
6408 Castleplace Drive
Indianapolis 46250
Tel: (317) 849-7300
TWX: 810-260-1794

KANSAS

†Hamilton/Avnet Electronics
9219 Quivera Road
Overland Park 66215
Tel: (913) 888-8900
TWX: 910-743-0005

MARYLAND

†Hamilton/Avnet Electronics
6822 Oak Hall Lane
Columbia 21045
Tel: (301) 995-3500
TWX: 710-862-1861

†Mesa Technology Corporation
16021 Industrial Drive
Gaithersburg 20877
Tel: (301) 948-4350
TWX: 710-828-9702

†Pioneer
9100 Gaither Road
Gaithersburg 20877
Tel: (301) 948-0710
TWX: 710-828-0545

MASSACHUSETTS

†Hamilton/Avnet Electronics
50 Tower Office Park
Woburn 01801
Tel: (617) 935-9700
TWX: 710-393-0382

†Arrow Electronics, Inc.
1 Arrow Drive
Woburn 01801
Tel: (617) 933-8130
TWX: 710-393-6770

†Harvey/Boston
44 Hartwell Avenue
Lexington 02173
Tel: (617) 863-1200
TWX: 710-326-6617

MICHIGAN

†Arrow Electronics, Inc.
3810 Vcrsity Drive
Ann Arbor 48104
Tel: (313) 971-8220
TWX: 810-223-6020

†Pioneer/Michigan
13485 Stamford
Livonia 48150
Tel: (313) 525-1800
TWX: 810-242-3271

†Hamilton/Avnet Electronics
32487 Schoolcraft Road
Livonia 48150
Tel: (313) 522-4700
TWX: 810-242-8775

†Hamilton/Avnet Electronics
2215 29th Street S.E.
Space A5
Grand Rapids, 49508
Tel: (616) 243-8805
TWX: 810-273-6921

MINNESOTA

†Arrow Electronics, Inc.
5230 W. 73rd Street
Edina 55435
Tel: (612) 830-1800
TWX: 910-576-3125

†Hamilton/Avnet Electronics
10300 Bren Road East
Minnetonka 55343
Tel: (612) 932-0600
TWX: (910) 576-2720

Pioneer/Twin Cities
10203 Bren Road East
Minnetonka 55343
Tel: (612) 935-5444
TWX: 910-576-2738

MISSOURI

†Arrow Electronics, Inc.
2380 Schuetz
St. Louis 63141
Tel: (314) 567-6888
TWX: 910-764-6600

†Hamilton/Avnet Electronics
13743 Shoreline Court
Earth City 63045
Tel: (314) 344-1200
TWX: 910-762-0684

NEW HAMPSHIRE

†Arrow Electronics, Inc.
1 Perimeter Road
Manchester 03103
Tel: (603) 668-6968
TWX: 710-220-1684

NEW JERSEY

†Arrow Electronics, Inc.
Pleasant Valley Avenue
Moorestown 08057
Tel: (215) 928-1800
TWX: 710-897-0829

†Arrow Electronics, Inc.
285 Midland Avenue
Saddle Brook 07662
Tel: (201) 797-5800
TWX: 710-998-2206

†Arrow Electronics, Inc.
2 Industrial Road
Fairfield 07006
Tel: (201) 575-5300
TWX: 710-998-2206

†Hamilton/Avnet Electronics
1 Keystone Avenue
Bldg. 36
Cherry Hill 08003
Tel: (609) 424-0100
TWX: 710-940-0262

†Harvey Electronics
45 Route 46
Pinebrook 07058
Tel: (201) 575-3510
TWX: 710-734-4382

†MTI Systems Sales
383 Route 46 W
Fairfield 07006
Tel: (201) 227-5552

NEW MEXICO

†Alliance Electronics Inc.
11030 Cochiti S.E.
Albuquerque 87123
Tel: (505) 292-3360
TWX: 910-989-1151

†Hamilton/Avnet Electronics
2524 Baylor Drive S.E.
Albuquerque 87106
Tel: (505) 765-1500
TWX: 910-989-0614

NEW YORK

†Arrow Electronics, Inc.
900 Broad Hollow Road
Farmingdale 11735
Tel: (516) 694-6800
TWX: 510-224-6126

†Arrow Electronics, Inc.
3000 South Winton Road
Rochester 14623
Tel: (716) 275-0300
TWX: 510-253-4766

†Arrow Electronics, Inc.
7705 Maltage Drive
Liverpool 13088
Tel: (315) 652-1000
TWX: 710-545-0230

†Arrow Electronics, Inc.
20 Oser Avenue
Hauppauge 11788
Tel: (516) 231-1000
TWX: 510-227-6623

†Hamilton/Avnet Electronics
333 Metro Park
Rochester 14623
Tel: (716) 475-9130
TWX: 510-253-5470

†Hamilton/Avnet Electronics
16 Corporate Circle
E. Syracuse 13057
Tel: (315) 437-2641
TWX: 710-541-1560

†Hamilton/Avnet Electronics
5 Hub Drive
Melville, Long Island 11747
Tel: (516) 454-6000
TWX: 510-224-6166

†Harvey Electronics
P.O. Box 1208
Binghamton 13902
Tel: (607) 748-8211
TWX: 510-252-0893

†Microcomputer System Technical Demonstrator Centers

U.S. DISTRIBUTORS

NEW YORK (Cont.)

†Harvey Electronics
60 Crossways Park West
Woodbury, Long Island 11797
Tel: (516) 921-8700
TWX: 510-221-2184

†Harvey/Rochester
840 Fairport Park
Fairport 14450
Tel: (716) 381-7070
TWX: 510-253-7001

†MTI Systems Sales
38 Harbor Park Drive
Port Washington 11050
Tel: (516) 621-6200
TWX: 510-223-0846

NORTH CAROLINA

†Arrow Electronics, Inc.
938 Burke Street
Winston-Salem 27101
Tel: (919) 725-8711
TWX: 510-931-3169

†Arrow Electronics, Inc.
3117 Poplarwood Court
Suite 123
Raleigh 27265
Tel: (919) 876-3132
TWX: 510-928-1856

†Hamilton/Avnet Electronics
2803 Industrial Drive
Raleigh 27609
Tel: (919) 829-8030
TWX: 510-928-1836

†Pioneer/Carolina
103 Industrial Avenue
Greensboro 27406
Tel: (919) 273-4441
TWX: 510-925-1114

OHIO

†Arrow Electronics, Inc.
7620 McEwen Road
Centerville 45459
Tel: (513) 435-5563
TWX: 810-459-1611

†Arrow Electronics, Inc.
6238 Cochran Road
Solon 44139
Tel: (216) 248-3990
TWX: 810-427-9409

†Hamilton/Avnet Electronics
954 Senate Drive
Dayton 45459
Tel: (513) 433-0610
TWX: 810-450-2531

†Hamilton/Avnet Electronics
4588 Emery Industrial Parkway
Warrensville Heights 44128
Tel: (216) 831-3500
TWX: 810-427-9452

†Pioneer/Dayton
4433 Interpoint Boulevard
Dayton 45424
Tel: (513) 236-9900
TWX: 810-459-1622

†Pioneer/Cleveland
4800 E. 131st Street
Cleveland 44105
Tel: (216) 587-3600
TWX: 810-422-2211

OKLAHOMA

†Arrow Electronics, Inc.
4719 S. Memorial Drive
Tulsa 74145
Tel: (918) 665-7700

OREGON

†Almac Electronics Corporation
8022 S.W. Nimbus, Bldg. 7
Beaverton 97005
Tel: (503) 641-9070
TWX: 910-467-8743

†Hamilton/Avnet Electronics
6024 S.W. Jean Road
Bldg. C, Suite 10
Lake Oswego 97034
Tel: (503) 635-7848
TWX: 910-455-8179

PENNSYLVANIA

†Arrow Electronics, Inc.
650 Seco Road
Monroeville 15146
Tel: (412) 856-7000

†Pioneer/Pittsburgh
259 Kappa Drive
Pittsburgh 15238
Tel: (412) 782-2300
TWX: 710-795-3122

†Pioneer/Delaware Valley
261 Gibralter Road
Horsham 19044
Tel: (215) 674-4000
TWX: 510-665-6778

TEXAS

†Arrow Electronics, Inc.
13715 Gama Road
Dallas 75234
Tel: (214) 386-7500
TWX: 910-860-5377

†Arrow Electronics, Inc.
10700 Corporate Drive
Suite 100
Stafford 77477
Tel: (713) 491-4100
TWX: 910-880-4439

†Arrow Electronics, Inc.
10125 Metropolitan
Austin 78758
Tel: (512) 835-4100
TWX: 910-874-1348

†Hamilton/Avnet Electronics
2401 Rutland
Austin 78757
Tel: (512) 837-8911
TWX: 910-874-1319

†Hamilton/Avnet Electronics
2111 W. Walnut Hill Lane
Irving 75062
Tel: (214) 659-4100
TWX: 910-860-5929

†Hamilton/Avnet Electronics
8750 West Park
Houston 77063
Tel: (713) 780-1771
TWX: 910-881-5523

†Pioneer/Austin
9901 Burnet Road
Austin 78758
Tel: (512) 835-4000
TWX: 910-874-1323

†Pioneer/Dallas
13710 Omega Road
Dallas 75234
Tel: (214) 386-7300
TWX: 910-850-5563

†Pioneer/Houston
5853 Point West Drive
Houston 77036
Tel: (713) 988-5555
TWX: 910-576-2738

UTAH

†Hamilton/Avnet Electronics
1585 West 2100 South
Salt Lake City 84119
Tel: (801) 972-2800
TWX: 910-925-4018

†Arrow Electronics, Inc.
4980 Amelia Earhart Drive
Salt Lake City 84112
Tel: (801) 539-1135

WASHINGTON

†Almac Electronics Corporation
14360 S.E. Eastgate Way
Bellevue 98007
Tel: (206) 643-9992
TWX: 910-444-2067

†Arrow Electronics, Inc.
14320 N.E. 21st Street
Bellevue 98007
Tel: (206) 643-4800
TWX: 910-444-2017

†Hamilton/Avnet Electronics
14212 N.E. 21st Street
Bellevue 98005
Tel: (206) 453-5874
TWX: 910-443-2469

WISCONSIN

†Arrow Electronics, Inc.
430 W. Rausson Avenue
Oakcreek 53154
Tel: (414) 764-6600
TWX: 910-262-1193

†Hamilton/Avnet Electronics
2975 Moorland Road
New Berlin 53151
Tel: (414) 784-4510
TWX: 910-262-1182

†Microcomputer System Technical Demonstrator Centers

INTEL EUROPEAN SALES OFFICES

BELGIUM

Intel Corporation S.A.
Parc Seny
Rue du Moulin a Papier 51
Boite 1
B-1160 Brussels
Tel: (02) 661 07 11
TELEX: 24814

DENMARK

Intel Denmark A/S*
Lyngbyvej 32F 2nd Floor
DK-2100 Copenhagen East
Tel: (01) 18 20 00
TELEX: 19567

FINLAND

Intel Finland OY
Hameentie 103
SF - 00550 Helsinki 55
Tel: 0/716 955
TELEX: 123 332

FRANCE

Intel Corporation, S.A.R.L.*
5 Place de la Balance
Silic 223
94528 Rungis Cedex
Tel: (01) 687 22 21
TELEX: 270475

Intel Corporation, S.A.R.L.
Immeuble BBC
4 Quai des Etroits
69005 Lyon
Tel: (7) 842 40 89
TELEX: 305153

WEST GERMANY

Intel Semiconductor GmbH*
Seidlstrasse 27
D-8000 Muenchen 2
Tel: (89) 53891
TELEX: 05-23177 INTL D

Intel Semiconductor GmbH*
Mainzer Strasse 75
D-6200 Wiesbaden 1
Tel: (6121) 70 08 74
TELEX: 04186183 INTW D

Intel Semiconductor GmbH
Brueckstrasse 61
7012 Fellbach
West Germany
Tel: (711) 58 00 82
TELEX: 7254826 INTS D

Intel Semiconductor GmbH*
Hohenzollern Strasse 5*
3000 Hannover 1
Tel: (511) 34 40 81
TELEX: 923625 INTH D

Intel Semiconductor GmbH
Ober-Ratherstrasse 2
D-4000 Dusseldorf 30
Tel: (211) 65 10 54
TELEX: 08-58977 INTL D

ISRAEL

Intel Semiconductor Ltd.*
P.O. Box 1659
Haifa
Tel: 4/524 261
TELEX: 46511

ITALY

Intel Corporation Italia Spa*
Milanofiori, Palazzo E
20094 Assago (Milano)
Tel: (02) 824 00 06
TELEX: 315183 INTMIL

NETHERLANDS

Intel Semiconductor Nederland B.V.*
Alexanderpoort Building
Marten Meesweg 93
3068 Rotterdam
Tel: (10) 21 33 77
TELEX: 22283

NORWAY

Intel Norway A/S
P.O. Box 92
Hvamveien 4
N-2013
Skjetten
Tel: (2) 742 420
TELEX: 18018

SWEDEN

Intel Sweden A.B.*
Box 20092
Archimedesvagen 5
S-16120 Bromma
Tel: (08) 98 53 85
TELEX: 12261

SWITZERLAND

Intel Semiconductor A.G.*
Forchstrasse 95
CH 8032 Zurich
Tel: (01) 55 45 02
TELEX: 57989 ICH CH

UNITED KINGDOM

Intel Corporation (U.K.) Ltd.*
5 Hospital Street
Nantwich, Cheshire CW5 5RE
Tel: (0270) 626 560
TELEX: 36620

Intel Corporation (U.K.) Ltd.*
Pipers Way
Swindon, Wiltshire SN3 1RJ
Tel: (0793) 488.388
TELEX: 444447 INT SWN

*Field Application Location

EUROPEAN DISTRIBUTORS/REPRESENTATIVES

AUSTRIA

Bacher Elektronische Geraete GmbH
Rotemuehlgasse 26
A 1120 Vienna
Tel: (222) 83 63 96
TELEX: 11532 BASAT A

BELGIUM

Inelco Belgium S.A.
Ave. des Croix de Guerre 94
B1120 Brussels
Tel: (02) 216 01 60
TELEX: 25441

DENMARK

MultiKomponent A/S
Fabriksparken 31
DK-2600 Glostrup
Tel: (02) 45 66 45
TX: 33355

Scandinavian Semiconductor
Supply A/S
Nannasgade 18
DK-2200 Copenhagen
Tel: (01) 83 50 90
TELEX: 19037

FINLAND

Oy Fintronic AB
Melkonkatu 24 A
SF-00210
Helsinki 21
Tel: (0) 692 60 22
TELEX: 124 224 Ftron SF

FRANCE

Jermyn S.A.
rue Jules Ferry 35
93170 Bagnolet
Tel: (1) 859 04 04
TELEX: 213810 F

Metrologie
La Tour d' Asnieres
1, Avenue Laurent Cely
92606-Asnieres
Tel: (1) 791 44 44
TELEX: 611 448

Tekelec Airtronic
Cite des Bruyeres
Rue Carle Vernet
F-92310 Sevres
Tel: (01) 534 75 35
TELEX: 204552

WEST GERMANY

Electronic 2000 Vertriebs A.G.
Neumarkter Strasse 75
D-8000 Munich 80
Tel: (89) 43 40 61
TELEX: 522561 EIEC D

Jermyn GmbH
Postfach 1180
Schulstrasse 48
D-6277 Bad Camberg
Tel: (06434) 231
TELEX: 484426 JERM D

Celdis Enatechnik Systems GmbH
Schillerstrasse 14
D-2085 Quickborn-Hamburg
Tel: (4106) 6121
TELEX: 213590 ENA D

Proelectron Vertriebs GmbH
Max Planck Strasse 1-3
6072 Dreieich bei Frankfurt
Tel: (6103) 33564
TELEX: 417983

IRELAND

Micro Marketing
Glenageary Office Park
Glenageary
Co. Dublin
Tel: (1) 85 62 88
TELEX: 31584

ISRAEL

Eastronics Ltd.
11 Rozanis Street
P.O. Box 39300
Tel Aviv 61390
Tel: (3) 47 51 51
TELEX: 33638

ITALY

Eledra 3S S.P.A.
Viale Elvezia, 18
I 20154 Milano
Tel: (2) 34 97 51
TELEX: 332332

Intesi
Milanfiori Pal. E/5
20090 Assago
Milano
Tel: (02) 82470
TELEX: 311351

NETHERLANDS

Koning & Hartman
Koperwerf 30
P.O. Box 43220
2544 EN 's Gravenhage
Tel: 31 (70) 210.101
TELEX: 31528

NORWAY

Nordisk Elektronic (Norge) A/S
Postoffice Box 122
Smedsvingen 4
1364 Hvalstad
Tel: (2) 786 210
TELEX: 16963

PORTUGAL

Ditram
Componentes E Electronica LDA
Av. Miguel Bombarda, 133
P1000 Lisboa
Tel: (19) 545 313
TELEX: 14182 Brieks-P

SPAIN

Interface S.A.
Ronda San Pedro 22, 3
Barcelona 10
Tel: (3) 301 78 51
TWX: 51508

ITT SESA
Miguel Angel 23-3
Madrid 10
Tel: (1) 419 54 00
TELEX: 27707

SWEDEN

AB Gosta Backstrom
Box 12009
Alstroemergatan 22
S-10221 Stockholm 12
Tel: (8) 541 080
TELEX: 10135

Nordisk Electronik AB
Box 27301
Sandhamnsgatan 71
S-10254 Stockholm
Tel: (8) 635 040
TELEX: 10547

SWITZERLAND

Industrade AG
Gemsenstrasse 2
Postcheck 80 - 21190
CH-8021 Zurich
Tel: (01) 363 23 20
TELEX: 56788 INDEL CH

UNITED KINGDOM

Bytech Ltd.
Unit 57
London Road
Earley, Reading
Berkshire
Tel: (0734) 61031
TELEX: 848215

Comway Microsystems Ltd.
Market Street
UK-Bracknell, Berkshire
Tel: 44 (344) 55333
TELEX: 847201

DECADE Ltd.
100 School Road
Tilehurst
Reading, Berkshire
Tel: (0734) 413127
TELEX: 837953

Jermyn Industries
Vestry Estate
Sevenoaks, Kent
Tel: (0732) 450144
TELEX: 95142

M.E.D.L.
East Lane Road
North Wembley
Middlesex HA9 7PP
Tel: (01) 904 93 07
TELEX: 28817

Rapid Recall, Ltd.
Rapid House/Denmark St
High Wycombe
Berks, England HP11 2ER
Tel: (0494) 26 271
TELEX: 837931

YUGOSLAVIA

H. R. Microelectronics Enterprises
P.O. Box 5604
San Jose, California 95150
Tel: 408/978-8000
TELEX: 278-559