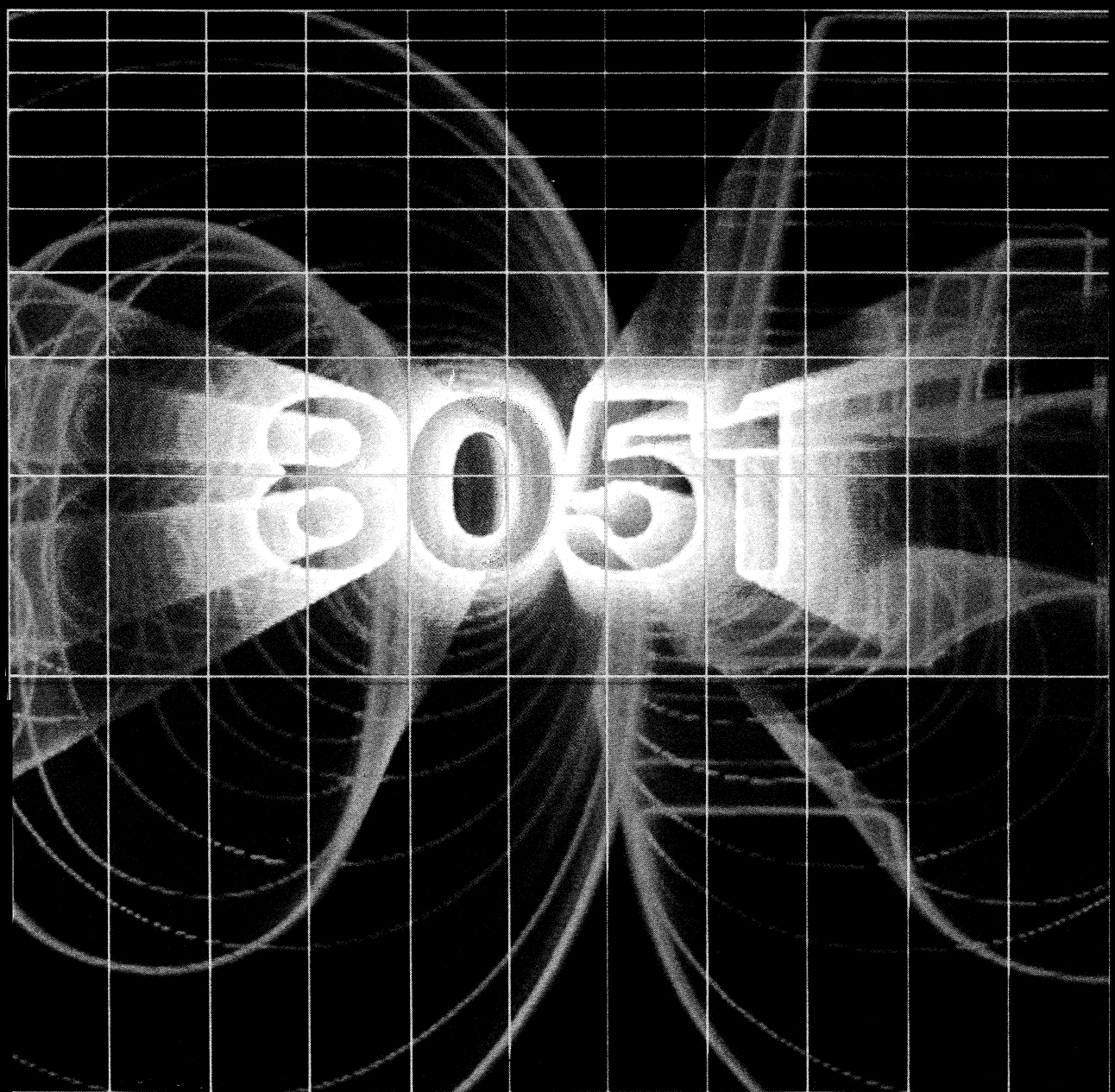
intel®

MCS-51™ Family of Single Chip Microcomputers User's Manual



Part No. 121517-001

MCS-51® FAMILY OF SINGLE-CHIP MICROCOMPUTERS USER'S MANUAL

JULY 1981

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

Intel Corporation
Literature Department SV3-3
3065 Bowers Avenue
Santa Clara, CA 95051

Table of Contents

CHAPTER 1
Introduction

1.0 Intel's Complete Line of Single-Chip Microcomputers 1-1
1.1 Introduction to MCS-51 1-2
1.2 8051 Family 1-2
1.3 8051 Family Development System and Software Support 1-3
1.3.1 8051 Software Development Package (ASM51 and CONV51) 1-3
1.3.2 8051 Macro Assembler (ASM51) 1-3
1.3.3 8048 to 8051 Assembly Language Converter Utility Program (CONV51) 1-4
1.3.4 8051 Emulation Board (EM-51) 1-4
1.3.5 8051 In-Circuit Emulator (ICE-51) 1-4
1.3.6 Personality Card for Universal PROM Programmer (UPP-551) 1-4
1.3.7 8051 Workshop 1-4
1.3.8 INSITE™ Library 1-4

CHAPTER 2
8051 Architecture

2.0 Macro-View of the 8051 Architecture 2-1
2.1 8051 CPU (Functional Description) 2-1
2.2 CPU Hardware 2-2
2.2.1 Instruction Decoder 2-2
2.2.2 Program Counter 2-2
2.2.3 Internal Program Memory 2-4
2.2.4 Internal Data Memory 2-4
2.2.5 Arithmetic Section 2-5
2.2.6 Program Control Section 2-5
2.3 On-Chip Peripherals 2-6
2.3.1 Macro-View of the 8051 Interrupt System 2-6
2.3.2 Interrupt System (Functional Description) 2-6
2.3.3 Macro-View of the 8051 I/O System 2-11
2.3.4 Ports and I/O Pins (Functional Description) 2-13
2.3.5 Macro-View of the 8051 Timer/Event Counters 2-18
2.3.6 Timer/Counter Mode Selection (Functional Description) 2-18
2.3.7 Macro-View of the 8051 Serial Communication Port 2-22
2.3.8 Serial Channel (Functional Description) 2-24
2.4 External Interface 2-28
2.4.1 Processor Reset and Initialization 2-28
2.4.2 Power Down (Standby) Operation of Internal RAM 2-29
2.4.3 Oscillator and Timer Circuitry 2-29
2.4.4 EPROM Programming 2-29
2.4.5 8051 Family Pin Description 2-30

CHAPTER 3
Memory Organization, Addressing Modes, and Data Manipulation

3.0 Memory Organization 3-1
3.1 Operand Addressing 3-3
3.1.1 Register Addressing 3-4
3.1.2 Direct Addressing 3-4
3.1.3 Immediate Addressing 3-5
3.1.4 Base-Register- plus Index-Register-Indirect Addressing 3-5

CHAPTER 3 (Cont.)

3.2 Data Manipulation 3-5
3.3 Boolean Processor 3-6
3.4 Data Transfer Operations 3-6
3.5 Logic Operations 3-8
3.6 Arithmetic Operations 3-9
3.7 Control Transfer 3-11

CHAPTER 4

8051 Instruction Set

4.0 What the Instruction Set Is 4-1
4.1 Organization of the Instruction Set 4-1
4.1.1 Data Transfer 4-2
4.1.2 Logic 4-2
4.1.3 Arithmetic 4-3
4.1.4 Control Transfer 4-5
4.2 Instruction Definitions 4-10
4.3 Expanded 8051 Family 4-64

CHAPTER 5

Software Routines

5.1 8051 Programming Techniques 5-1
5.1.1 Radix Conversion Routines 5-1
5.1.2 Multiple Precision Arithmetic 5-2
5.1.3 Table Look-Up Sequences 5-3
5.1.4 Saving CPU Status during Interrupts 5-5
5.1.5 Passing Parameters on the Stack 5-6
5.1.6 N-Way Branching 5-8
5.1.7 Computing Branch Destinations at Run Time 5-10
5.1.8 In-Line-Code Parameter-Passing 5-11
5.2 Peripheral Interfacing Techniques 5-13
5.2.1 I/O Port Reconfiguration (First Approach) 5-13
5.2.2 I/O Port Reconfiguration (Second Approach) 5-15
5.2.3 8243 Interfacing 5-16
5.2.4 Software Delay Timing 5-17
5.2.5 Serial Port and Timer Mode Configuration 5-17
5.2.6 Simple Serial I/O Drivers 5-18
5.2.7 Transmitting Serial Port Character Strings 5-19
5.2.8 Recognizing and Processing Special Cases 5-20
5.2.9 Buffering Serial Port Output Characters 5-21
5.2.10 Synchronizing Timer Overflows 5-22
5.2.11 Reading a Timer/Counter "On-the-Fly" 5-23

CHAPTER 6

Applications

An Introduction to the Intel MCS-51 Single Chip Microcomputer Family 6-1
Using the Intel MCS-51 Boolean Processing Capabilities 6-31

CHAPTER 7

MCS-51 Component Specifications 7-1

CHAPTER 8
Component Data Sheets

RAM

2114A 1024 x 4 Bit Static RAM 8-1
21821 4096 x 8-Bit Pseudostatic RAM 8-5
2118 Family 16,384 x 1 Bit Dynamic RAM 8-6

EPROM

2716 16K (2K x 8) UV Erasable PROM 8-17
2732A 32K (4K x 8) UV Erasable PROM 8-21
2816 8-27
2764 (8K x 8) UV Erasable PROM 8-40

8051 Peripherals

8155H 8-46

8048 Family

8020H HMOS Single-Component 8-Bit Microcomputer 8-58
8021H HMOS Single-Component 8-Bit Microcomputer 8-64
8022H High Performance Single-Component 8-Bit Microcomputer with On-Chip A/D Converter 8-73
8048H/ 8048H-1/8035HL/8035HL-1 HMOS Single-Component 8-Bit Microcomputer 8-74
8048L Special Low Power Consumption Single-Component 8-Bit Microcomputer 8-81
8049H/8039HL HMOS Single-Component 8-Bit Microcomputer 8-88
8748/8748-6/8748-8/8035 Single-Component 8-Bit Microcomputer 8-95
8749H/8749H-8/8039H/8039H-8 HMOS Single-Chip EPROM Microcomputer 8-104
8243 MCS-48® Input/Output Expander 8-113
I8048H New High Performance HMOS Single-Component 8-Bit Microcomputer 8-119
I8049H/8039H New High Performance Single-Component 8-Bit Microcomputer 8-120
I8031/8051 Single-Component 8-Bit Microcomputer 8-121
I8243 MCS-48 Input/Output Expander 8-122
M8048/M8748/M8035L Single-Component 8-Bit Microcomputer 8-128

CHAPTER 9
Development Support Tools

Model 225 Intellec® Series II/85 9-1
Intellec® Single/Double Density Flexible Disk System 9-6
8051 Software Development Package 9-10
ICE-51™ 8051 In-Circuit Emulator 9-13
UPP-103 Universal PROM Programmer 9-19
SDK-51 MCS-51 System Design Kit 9-21

APPENDIX A

PL/M-80 Description of 8051 Instruction Set A-1

Introduction 1

CHAPTER 1
INTRODUCTION

1.0 INTEL'S COMPLETE LINE OF SINGLE-CHIP MICROCOMPUTERS

In 1976 Intel introduced the 8048 microcomputer. This marked the first time in history that technology permitted a complete 8-bit computer to be fabricated on a single silicon die. This single chip can control a limitless variety of products ranging from appliances to automobiles to computer terminals.

Since 1976 Intel has offered products for the full range of single-chip microcomputer applications by pushing the 8048's architecture in several directions. The 8049 ran nearly twice as fast as the 8748/8048 while doubling the amount of on-chip program memory and data memory. Applications requiring solely external program memory were satisfied with the 8035 and 8039. Cost sensitive and less I/O intensive applications incorporated the 8021 which executed a subset of the 8048's instruction set at a slower speed. Finally, the 8022 integrated an 8-bit A/D converter onto the 8021 die to allow the chip to interface directly to a world in which most signals are analog. Figure 1-1 positions these products on a performance versus die-size curve.

Now, thanks to the density of HMOS, technology has once again permitted the birth of a microcomputer with performance to leap into new product areas. The 8051 achieves a 10X function/speed improvement over the 8048 by packing 60,000 transistors onto a die about 230 mils square.

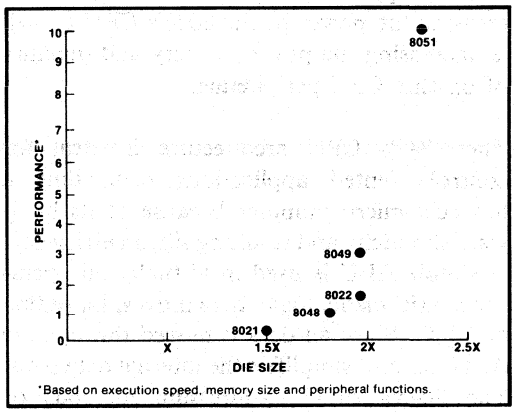


Figure 1-1. Performance Versus Cost

The 8051 family addresses applications in the high-end of the single-chip computer market. It is the highest performance microcomputer family in the world and out-performs all microprocessors and microcomputers in control oriented applications. It offers an upward compatible growth path for 8048 users with ten times the power of the 8048 as shown in Table 1-1.

- **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**
- **2½X To 10X Execution Speed**
- **1.4X Die Size**

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

1.1 INTRODUCTION TO MCS-51

The goal of the 8051 is to extend the architecture of the industry standard 8048 single-chip microcomputer into the 80's. This meant increasing the power of the 8048's CPU as well as increasing the power, variety and quantity of on-chip CPU peripherals.

The 8048's CPU architecture is ideal for control-oriented applications demanding a low-cost microcomputer because of its hardware simplicity and resulting silicon efficiency. A simple ALU is used in virtually all operations: arithmetic, logic, data moves, bit testing and I/O. Since all data is moved through the ALU this also simplifies the internal data path. The 8048's simple addressing methods of Register-, Register-Indirect- and Immediate-Addressing minimize hardware. The conditional branch logic simply concatenates an immediate value to the upper bits of the program counter to economize on silicon, but results in page boundaries. The simplicity of the table-look-up circuitry also results in page boundaries. The user flags and test pins provided for monitoring program and external status in an efficient manner are limited to two of each. This architecture, and the choice of instruction encodings that it permits, results in 1,024 byte programs of unsurpassed byte efficiency.

1.2 8051 FAMILY

The 8051 is a stand-alone high-performance single-chip computer 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 new 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. A Logic Symbol is shown in Figure 1-2.

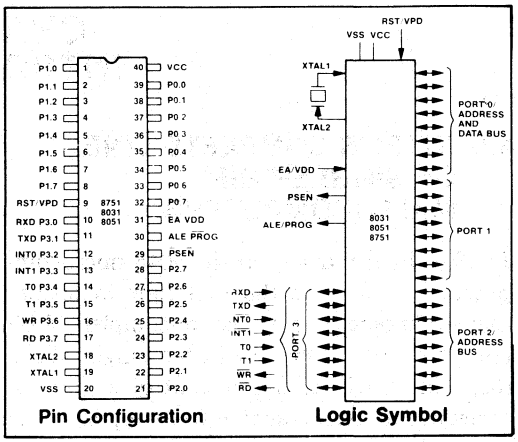


Figure 1-2

The 8031 is a control-oriented CPU without on-chip program memory. It can address 64K-bytes of external Program Memory in addition to 64K-bytes of External Data Memory. For systems requiring extra capability, each member of the 8051 family can be expanded using standard memories and the byte oriented MCS-80 and MCS-85 peripherals. The 8051 is an 8031 with the lower 4K-bytes of Program Memory filled with on-chip mask programmable ROM while the 8751 has 4K-bytes of UV-light-erasable/electrically-programmable ROM.

The three pin-compatible versions of this component reduce development problems to a minimum and provide maximum flexibility. The 8751 is well suited for development, prototyping, low-volume production and applications requiring field updates; the 8051 for low-cost, high-volume production; and the 8031

for applications desiring the flexibility of external Program Memory which can easily be modified and updated in the field.

1.3 8051 FAMILY DEVELOPMENT SYSTEM AND SOFTWARE SUPPORT

The 8051 is supported by a total range of Intel development tools. This broad range of support shortens the product development cycle and thus brings the product to market sooner.

- ASM51 Absolute macro assembler for the 8051.
- CONV51 8048 assembly language source code to 8051 assembly source code conversion program.
- EM-51 8051/8751 emulator board that uses a modified 8051 and an EPROM.
- ICE-51™ Real-time in-circuit emulator.
- SKD-51 System design kit for developing user Prototype around the 8051.
- UPP-551 8751 personality card for UPP-103 Universal PROM Programmer.
- 8051 Workshop.

1.3.1 8051 Software Development Package (ASM51 and CONV51)

The 8051 software development package provides development system support for the powerful 8051 family of single-chip microcomputers. The package contains a symbolic macro assembler and a 8048 to 8051 source code converter. This diskette-based software package runs under ISIS-II on any Intellec® Microcomputer Development System with 64K bytes of memory.

1.3.2 8051 Macro Assembler (ASM51)

The 8051 macro assembler translates symbolic 8051 assembly language instructions into machine executable object code. These assembly language mnemonics are easier to program and are more readable than binary or hexadecimal machine instructions. Also, by allowing the programmer to give symbolic names to memory locations rather than absolute addresses, software design and debug are performed more quickly and reliably.

ASM51 provides symbolic access for the many useful addressing methods in the 8051 architecture which reference bit, nibble and byte locations.

The assembler supports macro definitions and calls. This provides a convenient means of programming a frequently used code sequence only once. The assembler also provides conditional assembly capabilities. Cross referencing is provided in the symbol table listing, which shows the user the lines in which each symbol was defined and referenced.

If an 8051 program contains errors, the assembler provides a comprehensive set of error diagnostics, which are included in the assembly listing.

The object code generated may be used to program the 8751 EPROM version of the chip or sent to Intel for fabricating the 8051 ROM version. The assembler output can also be debugged using the ICE-51 in-circuit emulator.

1.3.3 8048 to 8051 Assembly Language Converter Utility Program (CONV51)

The 8048 to 8051 assembly language converter is a utility to help users of the MCS-48 family of microcomputers upgrade their designs to the high performance 8051 architecture. By converting 8048 source code to 8051 source code, the investment in software developed for the 8048 is maintained when the system is upgraded.

1.3.4 Emulation Board (EM-51)

The EM-51 8051 emulation board is a small (2.85" x 5.25") board which emulates an 8031/8051/8751 microcomputer using standard PROMs or EPROMs in place of the 8051's on-chip program memory. The board includes a modified 8051 microcomputer, supporting circuits, and two sockets for program memory. The user may select two 2716 EPROMs, a 2732 EPROM, or two 3636 bipolar PROMs depending on crystal frequency and power requirements.

1.3.5 8051 In-Circuit Emulator (ICE-51™)

The 8051 In-Circuit Emulator resides in the Intellec development system. The development system interfaces with the user's 8051 system through an in-cable buffer box with the cable terminating in an 8051 pin-compatible plug. Together these replace the 8051 device in the system. With the emulator plug in place, the designer can exercise the system in real-time while collecting up to 255 instruction cycles of real-time data. In addition, he can single step the system program.

Static RAM memory is available in the ICE-51 buffer box to emulate the 8051's internal and external program memories and external data memory. The designer can display and alter the contents of the replacement memory in the ICE-51 buffer box, internal 8051 registers, internal data RAM, and Special Function Registers. Symbolic reference capability allows the designer to use meaningful symbols provided by ASM51 rather than absolute values when examining and modifying these memory, register, flag, and I/O locations in his system.

1.3.6 Personality Card for Universal PROM Programmer (UPP-551)

The UPP-551 is a personality card for the UPP-103 Universal PROM Programmer. The Universal PROM Programmer is an Intellec system peripheral capable of programming and verifying the 8751 when the UPP-551 is inserted. Programming and verification operations are initiated from the Intellec development system console and are controlled by the Universal PROM Mapper (UPM) program.

1.3.7 8051 Workshop

The workshop provides the design engineer or system designer hands-on experience with the 8051 microcomputers. The course includes explanation of the Intel 8051 architecture, system timing and input/output design. Lab sessions will allow the attendee to gain detailed familiarity with the 8051 family and support tools.

1.3.8 INSITE™ Library

The INSITE Library contains 8051 utilities and applications programs.

8051 Architecture

2

CHAPTER 2
8051 ARCHITECTURE

2.0 MACRO-VIEW OF THE 8051 ARCHITECTURE

On a single die the 8051 microcomputer combines CPU; non-volatile 4K x 8 read-only program memory; volatile 128 x 8 read/write data memory; 32 I/O lines; two 16-bit timer/event counters; a five-source, two-priority-level, nested interrupt structure; serial I/O port for either multiprocessor communications, I/O expansion, or full duplex UART; and on-chip oscillator and clock circuits. This section will provide an overview of the 8051 by providing a high-level description of its major elements: the CPU architecture and the on-chip functions peripheral to the CPU. The generic term "8051" is also used to refer collectively to the 8031, 8051, and 8751. A Block Diagram is shown in Figure 2-1.

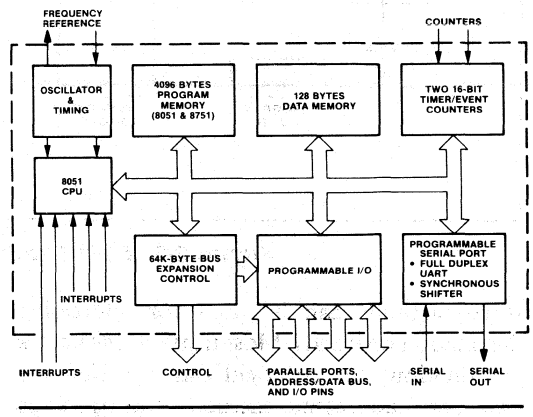


Figure 2-1. Block Diagram

2.1 8051 CPU (FUNCTIONAL DESCRIPTION)

The 8051 CPU manipulates operands in four memory spaces. These are the 64K-byte Program Memory, 64K-byte External Data Memory, 384-byte Internal Data Memory and 16-bit Program Counter spaces. The Internal Data Memory address space is further divided into the 256-byte Internal Data RAM and 128-byte Special Function Register (SFR) address spaces shown in Figure 2-2. Four Register Banks (each with eight registers), 128 addressable bits, and the stack reside in the Internal Data RAM. The stack depth is limited only by the available Internal Data RAM and its location is determined by the 8-bit Stack Pointer. All registers except the Program Counter and the four 8-Register Banks reside in the Special Function Register address space. These memory mapped registers include arithmetic registers, pointers, I/O ports, and registers for the interrupt system, timers and serial channel. 128 bit locations in the SFR address space are addressable as bits. The 8051 contains 128 bytes of Internal Data RAM and 20 SFRs.

The 8051 provides a non-paged Program Memory address space to accommodate relocatable code. Conditional branches are performed relative to the Program Counter. The register-indirect jump permits branching relative to a 16-bit base register with an offset provided by an 8-bit index register. Sixteen-bit jumps and calls permit branching to any location in the contiguous 64K Program Memory address space.

The 8051 has five methods for addressing source operands: Register, Direct, Register-Indirect, Immediate, and Base-Register- plus Index-Register- Indirect Addressing. The first

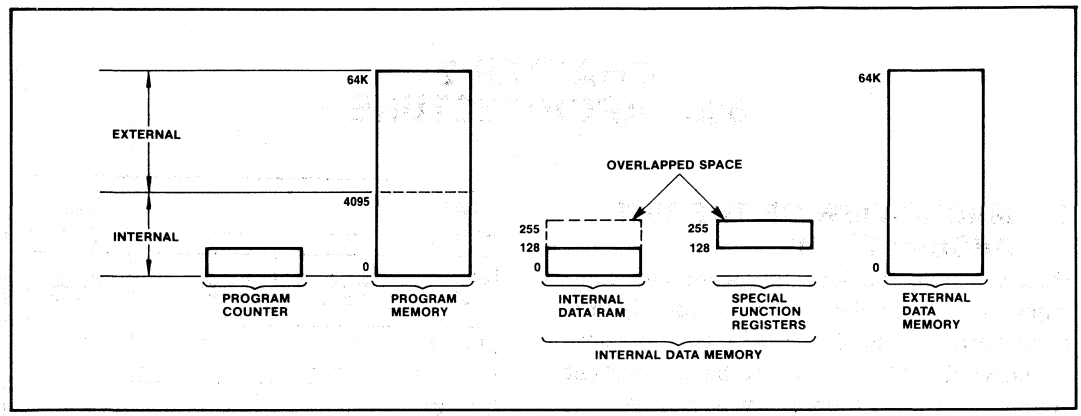


Figure 2-2. 8051 Family Memory Organization

three methods can be used for addressing destination operands. 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.

Any register in the four 8-Register Banks can be accessed through Register, Direct, or Register-Indirect Addressing; the 128 bytes of Internal Data RAM through Direct or Register-Indirect Addressing; and the Special Function Registers through Direct Addressing. External Data Memory is accessed through Register-Indirect Addressing. Look-Up-Tables resident in Program Memory can be accessed through Base-Register- plus Index-Register-Indirect Addressing.

The 8051 is classified as an 8-bit machine since the internal ROM, RAM, Special Function Registers, Arithmetic/Logic Unit and external data bus are each 8 bits wide. The 8051 performs operations on bit, nibble, byte and double-byte data types.

The 8051 has extensive facilities for byte transfer, logic, and integer arithmetic operations. It excells at bit handling since data transfer, logic and conditional branch operations can be performed directly on Boolean variables.

2.2 CPU HARDWARE

This section describes the hardware architecture of the 8051's CPU in detail. The interrupt system and on-chip functions peripheral to the CPU are described in subsequent sections. A detailed 8051 Functional Block Diagram is displayed in Figure 2-3.

2.2.1 Instruction Decoder

Each program instruction is decoded by the instruction decoder. This unit generates the internal signals that control the functions of each unit within the CPU section. These signals control the sources and destination of data, as well as the function of the Arithmetic/Logic Unit (ALU).

2.2.2 Program Counter

The 16-bit Program Counter (PC) controls the sequence in which the instructions stored in

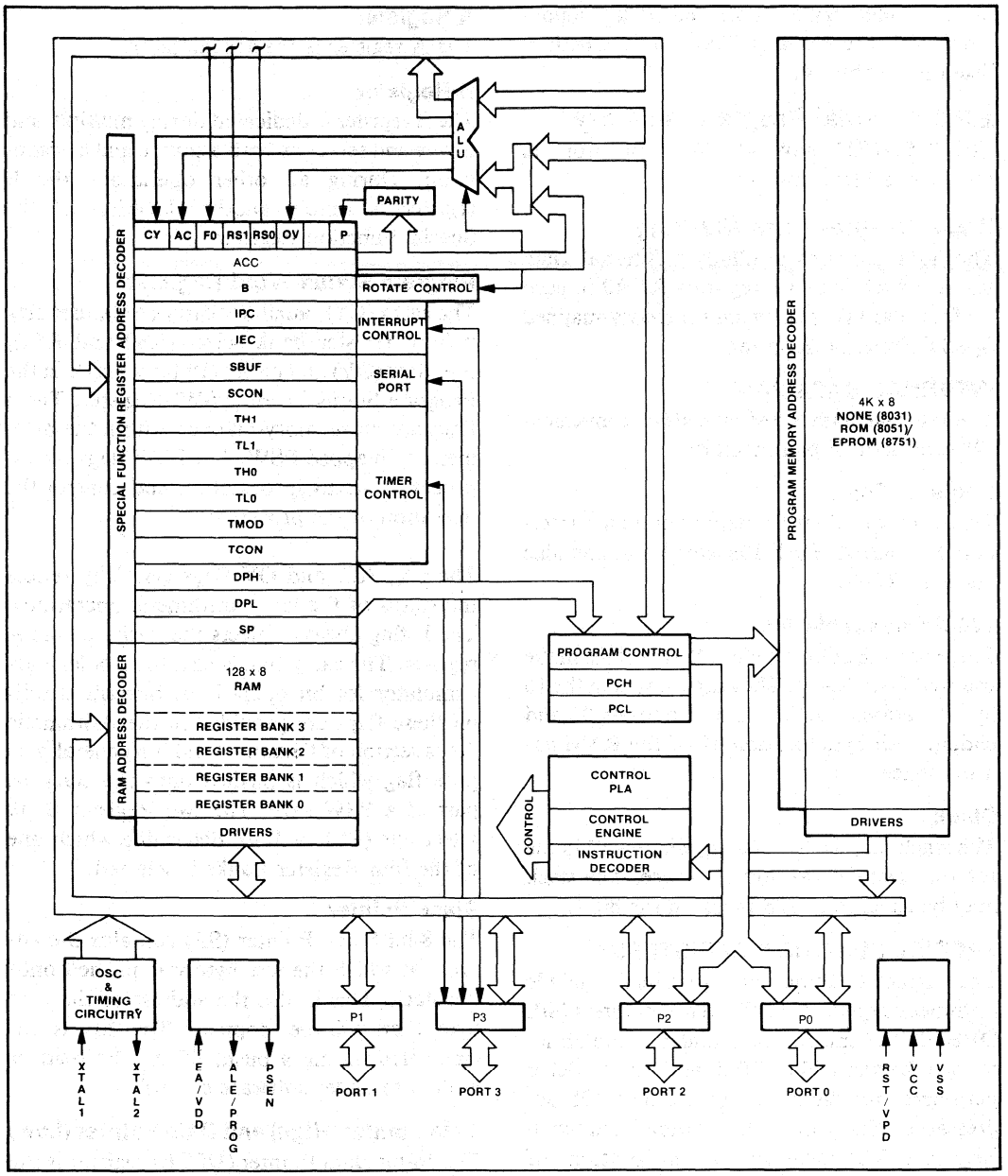


Figure 2-3. 8051 Family Functional Block Diagram

program memory are executed. It is manipulated with the Control Transfer instructions listed in Chapter 4.

2.2.3 Internal Program Memory

The 8051/8751 have 4K bytes of program memory resident on-chip.

2.2.4 Internal Data Memory

The 8051 contains a 128-byte Internal Data RAM (which includes registers R7-R0 in each of four Banks), and twenty memory-mapped Special Function Registers.

INTERNAL DATA RAM

The Internal Data RAM provides a convenient 128-byte scratch pad memory.

Register Banks

There are four Register Banks within the Internal Data RAM. Each Register Bank contains registers R7-R0.

128 Addressable Bits

There are 128 addressable software flags in the Internal Data RAM. They are located in the 16 byte locations starting at byte address 32 and ending with byte location 47 of the RAM address space.

Stack

The stack may be located anywhere within the Internal Data RAM address space. The stack may be as large as 128 bytes on the 8051.

SPECIAL FUNCTION REGISTERS

The Special Function Registers include arithmetic registers (A, B, PSW), pointers (SP, DPH, DPL) and registers that provide an interface between the CPU and the on-chip peripheral functions. These are also 128 addressable bits within the Special Function Registers. The memory-mapped locations of these registers and bits are discussed in Chapter 4.

A Register

The A register is the accumulator.

B Register

The B register is dedicated during multiply and divide and serves as both a source and a destination. During all other operations the B register is simply another location of the Special Function Register space.

Program Status Word Register

The carry (CY), auxiliary carry (AC), user flag 0 (F0), register bank select (RS0 and RS1), overflow (OV) and parity (P) flags reside in the Program Status Word (PSW) Register. These flags are bit-memory-mapped within the byte-memory-mapped PSW. The PSW flags record processor status information and control the operation of the processor.

The CY, AC, and OV flags generally reflect the status of the latest arithmetic operations. The P flag always reflects the parity of the A register. The carry flag is also the Boolean accumulator for bit operations. Specific details on these flags are provided in the Arithmetic Flags section of Chapter 4. F0 is a general purpose flag which is pushed onto the stack as part of a PSW save. The two Register Bank select bits (RS1 or RS0) determine which one of the four Register Banks is selected.

Stack Pointer

The 8-bit Stack Pointer (SP) contains the address at which the last byte was pushed onto the stack. This is also the address of the next byte that will be popped. The SP is incremented during a push. SP can be read or written to under software control.

Data Pointer (High) and Data Pointer (Low)

The 16-bit Data Pointer (DPTR) register is the concatenation of registers DPH (data pointer's high-order byte) and DPL (data pointer's low-

order byte). The DPTR is used in Register-Indirect Addressing to move Program Memory constants, to move External Data Memory variables, and to branch over the 64K Program Memory address space.

Port 3, Port 2, Port 1, Port 0

The four ports provide 32 I/O lines to interface to the external world. All four ports are both byte and bit addressable. The 8051 also allows memory expansion using Port 0 (P0) and Port 2 (P2) while Port 3 (P3) contains special control signals such as the read and write strobes. Port 1 (P1) is used for I/O only.

Interrupt Priority Register

The Interrupt Priority (IPC) register contains the control bits to set an interrupt to a desired level. A bit set to a one gives the particular interrupt a high priority listing.

Interrupt Enable Register

The Interrupt Enable (IEC) register stores the enable bits for each of the five interrupt sources. Also included is a global enable/disable bit of the interrupt system.

Timer/Counter Mode Register

Within the Timer Mode (TMOD) register are the bits that select which operations each timer/counter will do.

Timer/Counter Control Register

The timer/counters are controlled by the Timer/Counter Control (TCON) register bits. The start/stop bits for the timer/counters along with the overflow and interrupt request flags are mapped in TCON.

Timer/Counter 1 (High), Timer/Counter 1 (Low), Timer/Counter 0 (High), Timer/Counter 0 (Low)

There are four register locations for the two 16-bit timer/counters. These registers can be read or written to, to give the programmer easy access to the timer/counters. TH1 and TH0 refer to the 8 high-order bits of timer/counter 1 and 0, respectively. TL1 and TL0 refer to the low-order bits of both timer/counter 1 and 0.

Serial Control Register

This control register (SCON) has bits that enable reception of the serial port. Selecting the operating mode of the serial port is accomplished with bits in this register also.

Serial Data Buffer

The Serial Data Buffer (SBUF) register is used to hold serial port input or output data depending on whether the serial port is receiving or transmitting data.

2.2.5 Arithmetic Section

The arithmetic section of the processor performs many data manipulation functions and is comprised of the Arithmetic/Logic Unit (ALU), A register, B register and PSW register. The ALU accepts 8-bit data words from one or two sources and generates an 8-bit result under the control of the instruction decoder. The ALU performs the arithmetic operations of add, subtract, multiply, divide, increment, decrement, BCD-decimal-add-adjust and compare, and the logic operations of and, or, exclusive or, complement and rotate [right, left, or nibble swap (left four)].

2.2.6 Program Control Section

The program control section controls the sequence in which the instructions stored in program memory are executed. The conditional branch logic enables conditions internal and external to the processor to cause a change in the sequence of program execution.

2.3 ON-CHIP PERIPHERALS

2.3.1 Macro-View of the 8051 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 μs to 7 μs when using a 12 MHz crystal.

The 8051 acknowledges interrupt requests from five sources: Two from external sources via the $\overline{\text{INT0}}$ and $\overline{\text{INT1}}$ 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 in Figure 2-4.

2.3.2 Interrupt System (Functional Description)

Interrupts result in a transfer of control to a new program location. The program servicing the request begins at this address. In the 8051 there are five hardware resources that can

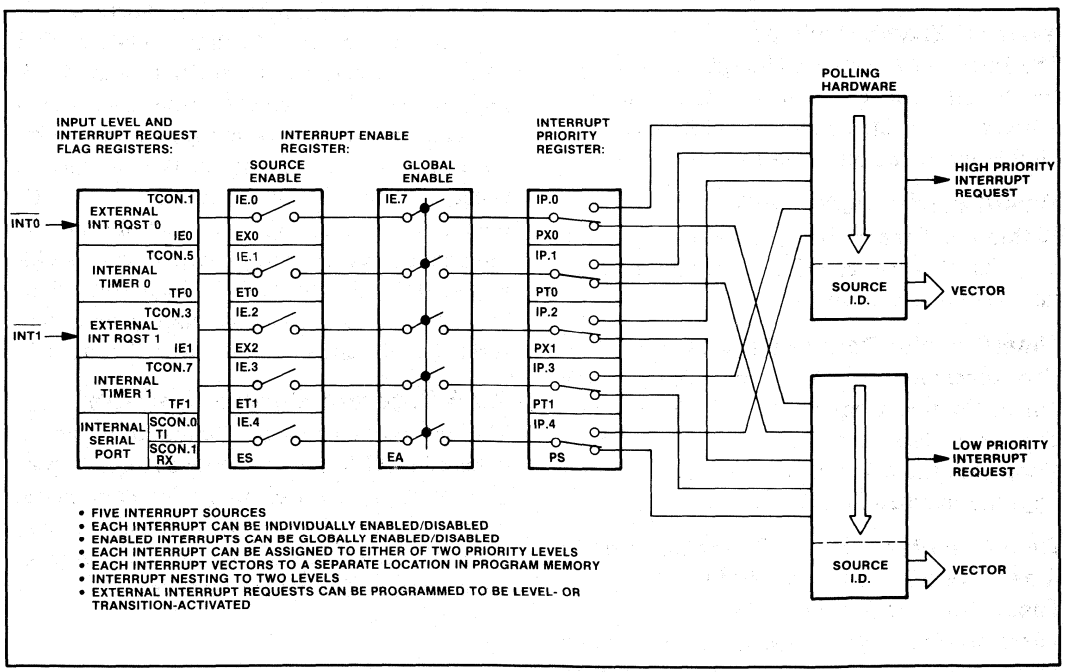


Figure 2-4. 8051 Interrupt System

Interrupt Source	Starting Address
External Request 0	3 (0003 H)
Internal Timer/ Counter 0	11 (000B H)
External Request 1	19 (0013 H)
Internal Timer/ Counter 1	27 (001B H)
Internal Serial Port	35 (0023 H)

Table 2-1. Program Memory Location of Interrupt Service Programs

generate an interrupt request. The starting address of the interrupt service program for each interrupt source is shown in Table 2-1.

A resource requests an interrupt by setting its associated interrupt request flag in the TCON or SCON register, as detailed in Table 2-2. The interrupt request will be acknowledged if its interrupt enable bit in the Interrupt Enable register (shown in Table 2-3) is set and if it is the highest priority resource requesting an interrupt. A resource's interrupt priority level is established as high or low by the polarity of a bit in the Interrupt Priority register. These bit assignments are shown in Table 2-4. Setting the resource's associated bit to a one (1) programs it to the higher level. The priority of multiple interrupt requests occurring simultaneously and assigned to the same priority level is also shown in Table 2.4.

Interrupt Source	Request Flag	Bit Location
External Request 0	IE0	TCON.1
Internal Timer/ Counter 0	TF0	TCON.5
External Request 1	IE1	TCON.3
Internal Timer/ Counter 1	TF1	TCON.7
Internal Serial Port (xmit)	TI	SCON.1
Internal Serial Port (rcvr)	RI	SCON.0

Table 2-2. Interrupt Request Flags

(MSB)							(LSB)
EA	—	—	ES	ET1	EX1	ET0	EX0

Function	Enable	Bit Location
Enable All control bit. Cleared by software to disable all interrupts, independent of the state of IE.4-IE.0	EA	IE.7
(reserved)	—	IE.6
(reserved)	—	IE.5
Enable Serial port control bit. Set/cleared by software to enable/disable interrupts from TI or RI flags.	ES	IE.4
Enable Timer 1 control bit. Set/cleared by software to enable/disable interrupts from timer/ counter 1.	ET1	IE.3
Enable External interrupt 1 control bit. Set/cleared by software to enable/disable interrupts from INT1.	EX1	IE.2
Enable Timer 0 control bit. Set/cleared by software to enable/disable interrupts from timer/ counter 0.	ET0	IE.1
Enable External interrupt 0 control bit. Set/cleared by software to enable/disable interrupts from INT0.	EX0	IE.0

Table 2-3. IE—Interrupt Enable Register

Interrupt Source	Priority Flag	Priority Within Level	Bit Location
Reserved	None		IP.7
Reserved	None		IP.6
Reserved	None		IP.5
Internal Serial Port	PS	.4 (lowest)	IP.4
Internal Timer/ Counter 1	PT1	.3	IP.3
External Request 1	PX1	.2	IP.2
Internal Timer/ Counter 0	PT0	.1	IP.1
External Request 0	PX0	.0 (highest)	IP.0

Table 2-4. Interrupt Priority Flags

The servicing of a resource's interrupt request occurs at the end of the instruction-in-progress. The processor transfers control to the starting address of this resource's interrupt service program and begins execution. Within the Interrupt Enable register (IE) there are six addressable flags. Five flags enable/disable the five interrupt sources when set/cleared. Setting/clearing the sixth flag permits a global enable/disable of each enabled interrupt request.

Setting/clearing a bit in the Interrupt Priority register (IP) establishes its associated interrupt request as a high/low priority (Table 2-5). If a low-priority level interrupt is being serviced, a high-priority level interrupt will interrupt it. However, an interrupt source cannot interrupt a service program of the same or higher level.

The processor records the active priority level(s) by setting internal flip-flop(s). One of these non-addressable flip-flops is set while a

(MSB)							(LSB)
—	—	—	PS	PT1	PX1	PT0	PX0

Function	Priority	Bit Location
(reserved)	—	IP.7
(reserved)	—	IP.6
(reserved)	—	IP.5
Serial Port Priority control bit. Set/cleared by software to specify high/low priority interrupts for Serial port.	PS	IP.4
Timer 1 Priority control bit. Set/cleared by software to specify high/low priority interrupts for timer/counter 1.	PT1	IP.3
External interrupt 1 Priority control bit. Set/cleared by software to specify high/low priority interrupts for INT1.	PX1	IP.2
Timer 0 Priority control bit. Set/cleared by software to specify high/low priority interrupts for timer/counter 0.	PT0	IP.1
External interrupt 0 Priority control bit. Set/cleared by software to specify high/low priority interrupts for INT0.	PX0	IP.0

Table 2-5. IP—Interrupt Priority Control Register

low-level interrupt is being serviced. The other flip-flop is set while the high-level interrupt is being serviced. The appropriate flip-flop is set when the processor transfers control to the ser-

vice program. The flip-flop corresponding to the interrupt level being serviced is reset when the processor executes an RETI Instruction.

To summarize, the sequence of events for an interrupt is: A resource provokes an interrupt by setting its associated interrupt request bit to let the processor know an interrupt condition has occurred. The CPU's internal hardware latches the interrupt request near the falling-edge of ALE in the tenth, twenty-second, thirty-fourth and forty-sixth oscillator period of the instruction-in-progress. The interrupt request is conditioned by bits in the interrupt enable and interrupt priority registers. The processor acknowledges the interrupt by setting one of the two internal "priority-level active" flip-flops and performing a hardware subroutine call. The call pushes the PC (but not the PSW) onto the stack and, for most sources, clears the interrupt request flag. The service program is then executed. Control is returned to the main program when the RETI instruction is executed. The RETI instruction also clears one of the internal "priority-level active" flip-flops.

Most interrupt request flags (IE0, IE1, TF0 and TF1) are cleared when the processor transfers control to the first instruction of the interrupt service program. The TI and RI interrupt request flags are the exceptions and must be cleared as part of the serial port's interrupt service program.

The process whereby a high-level interrupt request interrupts a low-level interrupt service program is called nesting. In this case the address of the next instruction in the low-priority service program is pushed onto the stack, the stack pointer is incremented by two (2) and processor control is transferred to the Program Memory location of the first instruction of the high-level service program. The last instruction of the high-priority interrupt service program must be an RETI instruction. This instruction clears the high "priority-level-active" flip-flop. RETI also returns processor control to the next instruction of the low-level interrupt service program. Since the lower "priority-level-active" flip-flop has remained set, high priority interrupts are re-enabled while further low priority interrupts remain disabled.

The highest-priority interrupt request gets serviced at the end of the instruction-in-progress unless the request is made in the last fourteen oscillator periods of the instruction-in-progress. Under this circumstance, the next instruction will also execute before the interrupt's subroutine call is made. The first instruction of the service program will begin execution twenty-four oscillator periods (the time required for the hardware subroutine call) after the completion of the instruction-in-progress or, under the circumstances mentioned earlier, twenty-four oscillator periods after the next instruction.

Thus, the greatest delay in response to an interrupt request is 86 oscillator periods (approximately 7 μsec @ 12 MHz). Examples of the best and worst case conditions are illustrated in Table 2-6.

EXTERNAL INTERRUPTS

The external interrupt request inputs ($\overline{\text{INT0}}$ and $\overline{\text{INT1}}$) can be programmed for either transition-activated or level-activated operation. Control of the external interrupts is provided by the four low-order bits of TCON (Table 2-7). When IT0 and IT1 are set to one (1), interrupt requests on $\overline{\text{INT0}}$ and $\overline{\text{INT1}}$ are

Instruction	Time (Oscillator Periods) Best Case	Worst Case
1) External interrupt request generated immediately before (best)/ after (worst) the pin is sampled. (Time until end of bus cycle.)	2+ϵ	2-ϵ
2) Current or next instruction finishes in 12 oscillator periods	12	12
3) Next instruction is MUL or DIV	don't care	48
4) Internal latency for hardware subroutine call	24	24
	38	86

Table 2-6. Best and Worst Case Response to Interrupt Request

Function	Flag	Bit Location
External Interrupt Request Flag 1	IE1	TCON.3
Input $\overline{\text{INT1}}$ Transition Activated	IT1	TCON.2
External Interrupt Request Flag 0	IE0	TCON.1
Input $\overline{\text{INT0}}$ Transition Activated	IT0	TCON.0

Table 2-7. Function of Bits in TCON (Lower Nibble)

transition-activated (high-to-low); or else they are low-level activated. IE0 and IE1 are the interrupt request flags. These flags are set when their corresponding interrupt request inputs at $\overline{\text{INT0}}$ and $\overline{\text{INT1}}$, respectively, are low when sampled by the 8051 and the transition-activated scheme is selected by IT0 and IT1. When IT0 and IT1 are programmed for level-activated interrupts, the IE0 and IE1 flags are not affected by the inputs at $\overline{\text{INT0}}$ and $\overline{\text{INT1}}$, respectively.

TRANSITION-ACTIVATED INTERRUPTS

The external interrupt request inputs ($\overline{\text{INT0}}$ and $\overline{\text{INT1}}$) can be programmed for high-to-low transition-activated operation. For transition-activated operation, the input must remain low for greater than twelve oscillator periods, but need not be synchronous with the oscillator. It is internally latched by the 8051 near the falling-edge of ALE during an instruction's tenth, twenty-second, thirty-fourth and forty-sixth oscillator periods and, if the input is low, IE0 or IE1 is set. The upward transition of a transition-activated input may occur at any time after the twelve oscillator period latching time, but the input must remain high for twelve oscillator periods before reactivation.

LEVEL-ACTIVATED INTERRUPTS

The external interrupt request inputs ($\overline{\text{INT0}}$ and $\overline{\text{INT1}}$) can be programmed for level-activated operation. The input is sampled by the 8051 near the falling-edge of ALE during the instruction's tenth, twenty-second, thirty-fourth and forty-sixth oscillator periods. If the input is low during the sampling that occurs fourteen oscillator periods before the end of the instruction in progress, an interrupt subroutine call is made. The level-activated input need be low only during the sampling that occurs fourteen oscillator periods before the end of the instruction-in-progress and may remain low during the entire execution of the service program. However, the input must be raised before the service program completes to avoid possibly envoking a second interrupt.

2.3.3 Macro-View of the 8051 I/O System

The 8051 has instructions that treat its 32 I/O lines as 32 individually addessable bits and as four parallel 8-bit ports addressable as Ports 0, 1, 2 and 3. Ports 0, 2 and 3 can also assume other functions. Port 0 provides the multiplexed low-order address and data bus used for expanding the 8051 with standard memories and peripherals. Port 2 provides the high-order address bus when expanding the 8051 with external Program Memory or more than 256 bytes of External Data Memory. The pins of Port 3 can be configured individually to provide external interrupt request inputs, counter inputs, the serial port's receiver input and transmitter output, and to generate the control signals used for reading and writing External Data Memory. The generation or use of an alternate function on a Port 3 pin is done automatically by the 8051 as long as the pin is configured as an input.

OPEN DRAIN I/O PINS

Each pin of Port 0 can be configured as an open drain output or as a high impedance input. Resetting the microcomputer programs each pin as an input by writing a one (1) to the pin. If a zero (0) is later written to the pin it becomes configured as an output and will continuously sink current. Re-writing the pin to a one (1) will place its output driver in a high-impedance state and configure the pin as an input. Each I/O pin of Port 0 can sink two TTL loads.

QUASI-BIDIRECTIONAL I/O PINS

Ports 1, 2 and 3 are quasi-bidirectional buffers. Each quasi-bidirectional buffer has an internal pull up resistor of approximately 10K-to 40K ohms connected between its I/O pin and the positive power supply pin. Resetting the microcomputer programs each pin as an input by writing a one (1) to the pin. If a zero (0) is later written to the pin it becomes configured as an output and will continuously sink current. Any pin that is configured as an output will be reconfigured as an input when a one (1) is written to the pin. Simultaneous to this reconfiguration the output driver in the quasi-bidirectional buffer will source current for two oscillator periods. Since the output driver sources current only when a bit previously written to a zero (0) is updated to a one (1), a pin programmed as an input will not source current into the TTL gate that is driving it if the pin is later written with another one (1). Since the quasi-bidirectional output driver sources current for only two oscillator periods, an internal pullup resistor of approximately 20K to 40K ohms is provided to hold the external driver's loading at a TTL high level. Ports 1, 2 and 3 can sink/source one TTL load.

MICROPROCESSOR BUS

A microprocessor bus is provided to permit the 8051 to solve a wide range of problems and to allow the upward growth of user products. This multiplexed address and data bus provides an interface compatible with standard memories, MCS-80 peripherals and the MCS-85 memories that include on-chip programmable I/O ports and timing functions. These are summarized in the 8051 Microcomputer Expansion Components chart of Table 2-8.

When accessing external memory the high-order address is emitted on Port 2 and the low-order address on Port 0. The ALE signal is provided for strobing the address into an external latch. The program store enable ($\overline{\text{PSEN}}$) signal is provided for enabling an external memory device to Port 0 during a read from the Program Memory address space. When the MOVX instruction is executed Port 3

	Category	I.D.	Description	Comments
Compatible MCS-80/85 Components	I/O Expander		8 Line I/O Expander (Shift Register)	Low Cost I/O Expander
	Standard EPROMs	2716-1 2732A	2K x 8 350 ns Light Erasable 4K x 8 250 ns Light Erasable	User Programmable and erasable
	Standard RAMs	2114A 2148 2142-2	1K x 4 100 ns RAM 1K x 4 70 ns RAM 1K x 4 200 ns RAM	Data memory can be easily expanded using standard NMOS RAMs.
	Multiplexed Address/ Data RAMs	8185A	1K x 8 300 ns RAM	
	Standard I/O	8212 8282 8283 8255A 8251A	8-Bit I/O Port 8-Bit I/O Port 8-Bit I/O Port Programmable Peripheral Interface Programmable Communications Interface	Serves as Address Latch or I/O port. Three 8-bit programmable I/O ports. Serial Communications Receiver/Transmitter.
	Standard Peripherals	8205 8286 8287 8253A 8279 8291 8292	1 of 8 Binary Decoder Bi-directional Bus Driver Bi-directional Bus Driver (Inverting) Programmable Interval Timer Programmable Keyboard/Display Interface (128 Keys) GPIB Talker/Listener GPIB Controller	MCS-80 and MCS-85 peripheral devices are compatible with the 8051 allowing easy addition of specialized interfaces.
	Universal Peripheral Interfaces	8041A 8741A	ROM Program Memory EPROM Program Memory	User programmable to perform custom I/O and control functions.
	Memories with on-chip I/O and Peripheral Functions.	8155-2 8355-2 8755-2	256 x 8 330 ns RAM 2K x 8 300 ns ROM 2K x 8 300 ns EPROM	Future MCS-80/85 devices will also be compatible.

Table 2-8. 8051 Microcomputer Expansion Components

automatically generates the read ($\overline{RD}$) signal for enabling an External Data Memory device to Port 0 or generates the write ($\overline{WR}$) signal for strobing the external memory device with the data emitted by Port 0. Port 0 emits the address and data to the external memory through a push/pull driver that can sink/source two TTL loads. At the end of the read/write bus cycle Port 0 is automatically reprogrammed to its high impedance state and Port 2 is returned to the state it had prior to the bus cycle. The 8051 generates the address, data and control signals needed by memory and I/O devices in a manner that minimize the requirements placed on external program and data memories. At 12 MHz, the Program Memory cycle time is

500ns and the access times required from stable address and $\overline{PSEN}$ are approximately 320ns and 150ns respectively. The External Data Memory cycle time is 1μs and the access time required from stable address and from read ($\overline{RD}$) or write ($\overline{WR}$) command is approximately 250ns.

2.3.4 Ports and I/O Pins (Functional Description)

There are 32 I/O pins configured as four 8-bit ports. Each pin can be individually and independently programmed as an input or an output and each can be reconfigured dynamically (i.e., on-the-fly) under software control.

The instructions that perform a read of, operation on, and write to a port's bit/byte are INC, DEC, CPL, JBC, CJNE, DJNZ, ANL, ORL, and XRL. The source read by these operations is the last value that was written to the port, without regard to the levels being applied at the pins. This insures that bits written to a one (1) for use as inputs are not inadvertently cleared. See Figure 2-5A.

An instruction that uses a port's bit/byte as a source operand reads a value that is the logical and of the last value written to the bit/byte and the polarity being applied to the pin/pins by an external device (this assumes that none of the 8051's electrical specs are being violated). An instruction that reads a bit/byte, operates on the content, and writes the result back to the bit/byte, reads the last value written to the bit/byte instead of the logic level at the pin/pins. Pins comprising a single port can be made into a mixed collection of inputs and outputs by writing a "one" to each pin that is to be an input. Each time an instruction uses a port as the destination, the operation must

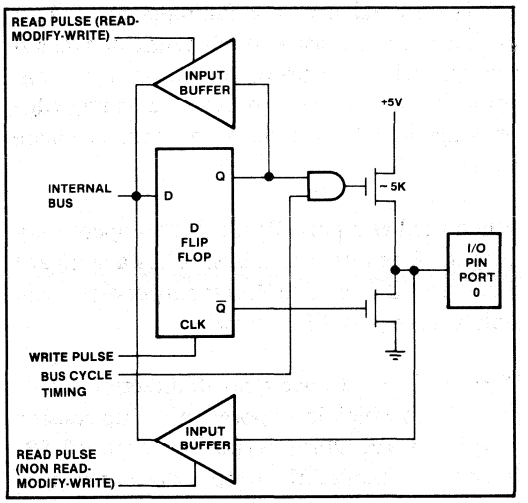


Figure 2-5A. "Bidirectional" Port Structure

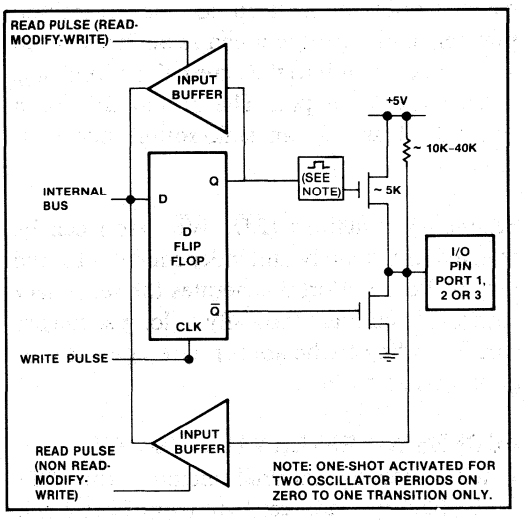


Figure 2-5B. "Quasi-Bidirectional" Port Structure

write "ones" to those bits that correspond to the input pins. An input to a port pin need not be synchronized to the oscillator. Each port

pin is sampled near the falling-edge of ALE during the read instruction's tenth or twenty-second oscillator period. If an input is in transition when it is sampled near the falling-edge of ALE it will be read as an indeterminate value.

When used as a port, Port 0 has an open-drain output. When used as a bus, it has a standard three-state driver. The Port 0 output driver can sink/source two TTL loads.

Ports 1, 2 and 3 have quasi-bidirectional output drivers which incorporate a pullup resistor of 10K- to 40K-Ohms as shown in Figure 2-5B. In ports 1, 2 and 3 the output driver provides source current for two oscillator periods if, and only if, software updates the bit in the output latch from zero (0) to a one (1). Sourcing current only on a "zero to one" transition prevents a pin, programmed as an input, from sourcing current into the external device that is driving the input pin. The output drivers in Ports 1, 2 and 3 can sink/source one TTL load.

Secondary functions ($\overline{RD}$, $\overline{WR}$, etc.) can be selected individually and independently for the pins of Port 3. Port 3 generates the secondary control signals automatically as long as the pin corresponding to the appropriate signal is programmed as an input.

ACCESSING EXTERNAL MEMORY

When accessing external memory the 8051 emits the upper address byte from Port 2 and the lower address byte, as well as the data, from Port 0. It uses ALE, $\overline{PSEN}$ and two pins from Port 3 ($\overline{RD}$ and $\overline{WR}$) for memory control. ALE is used for latching the address into the external memory. The $\overline{PSEN}$ signal enables the external Program Memory to Port 0, the $\overline{RD}$ signal enables External Data Memory to Port 0 and the $\overline{WR}$ signal latches the data byte emitted by Port 0 into the External Data Memory. Externally the $\overline{PSEN}$ and $\overline{RD}$ signals can be combined logically if a contiguous external program and data memory space (similar to a "von Neuman" machine) is desired. The P3.7 ($\overline{RD}$) and P3.6 ($\overline{WR}$) output latches must be programmed to a one (1) if External Data Memory is to be accessed. When P3.7 and P3.6 are programmed as $\overline{RD}$ and $\overline{WR}$ respectively, the remaining pins of Port 3 may be individually programmed as desired.

The 8051 can address 64K bytes of external Program Memory when the $\overline{EA}$ pin is tied low. When $\overline{EA}$ is high, the 8051 fetches instructions from internal Program Memory when the address is between 0 and 4095 and from external Program Memory when the addressed memory location is between 4096 and 64K. In either case, Ports 2 and 0 are automatically configured as an external bus based on the value of the PC. Instruction execution times are the same for code fetched from internal or external Program Memory.

Up to 64K of External Data Memory can be accessed using the MOVX instructions. These instructions automatically configure Port 0, and often Port 2, as an external bus. The MOVX instructions use the DPTR, R1 or R0 register as a pointer into the External Data Memory. The 16-bit DPTR register is used when successive accesses cover a wide range of the 64K space. The 8-bit R1 or R0 registers provide greatest byte efficiency when successive accesses are constrained to a 256-byte block of the External Data Memory space. When using R1 and R0 a subsequent block can be accessed by updating the output latch of Port 2. Port 2 is not affected by execution of a

MOVX that uses R1 or R0 such that, if 32K or less of external memory is present, only part of Port 2 needs to be used for selecting the desired block; the remaining pins can be used for I/O. When a MOVX using DPTR is executed, the value in Port 2's output latch is altered only during the external access and then is returned to its prior value. This permits efficient external block moves by interleaving MOVX instructions that use DPTR and R1 or R0.

The ALE signal is generated every sixth oscillator period during reads from either internal or external Program Memory. The $\overline{PSEN}$ signal is generated every sixth oscillator period when reading the external Program Memory. When a read or write from External Data Memory is being performed, a single ALE and a $\overline{RD}$ or a $\overline{WR}$ signal is generated during a twelve oscillator period interval.

If the CPU does not address External Data Memory then ALE is generated every sixth oscillator period and can be used as an external clock. When External Data Memory is present, external logic may be used to combine the occurrence of $\overline{RD}$, $\overline{WR}$, and $\overline{ALE}$ to generate an external clock with a period equal to six oscillator periods.

The 8051 always fetches an even number of bytes from its Program Memory. If an odd number of bytes are executed prior to a branch or to an External Data Memory access, the non-executed byte is ignored by the 8051. If an instruction requires more oscillator periods for its execution than for its fetch, the first byte of the next instruction is fetched repeatedly while the first instruction completes execution.

For the MOVC instruction the op-code is fetched in the first six-oscillator period, the first byte of the next instruction is fetched during the second six-oscillator period, the table entry is fetched in a third six-oscillator period and the first byte of the next instruction is again fetched in the fourth six-oscillator period.

ACCESSING EXTERNAL MEMORY — OPERATION OF PORTS

The Port 0 is time multiplexed to permit transfer of both addresses and data. This bus is used directly by memory and peripheral devices that incorporate on-chip address latching (MCS-85 memories with peripherals), or it can be demultiplexed with an address latch to generate a non-multiplexed bus (MCS-80 peripherals and memory). During an external access the low-order byte of the address and the data (for a write) is emitted by the Port 0 output drivers. Ones (1's) are automatically written to Port 0 at the very end of the bus cycle. Since the Port 0 output latches will contain ones (1's) at the end of the bus cycle, Port 0 will be in its high impedance state when a bus cycle is not in progress. Port 2 emits the upper 8-bits of the address when a MOVX instruction using DPTR is executed. Port 2's output drivers provide source current for two oscillator periods when emitting the address. Port 2's internal pullup resistors sustain the high level.

ACCESSING EXTERNAL MEMORY — BUS CYCLE TIMING

Program Memory Read Sequence

Each Program Memory bus cycle consists of six oscillator periods. These are referred to as T1, T2, T3, T4, T5 and T6 on Figure 2-6. The address is emitted from the processor during T3. Data transfer occurs on the bus during T5, T6 and the following bus cycle's T1. When

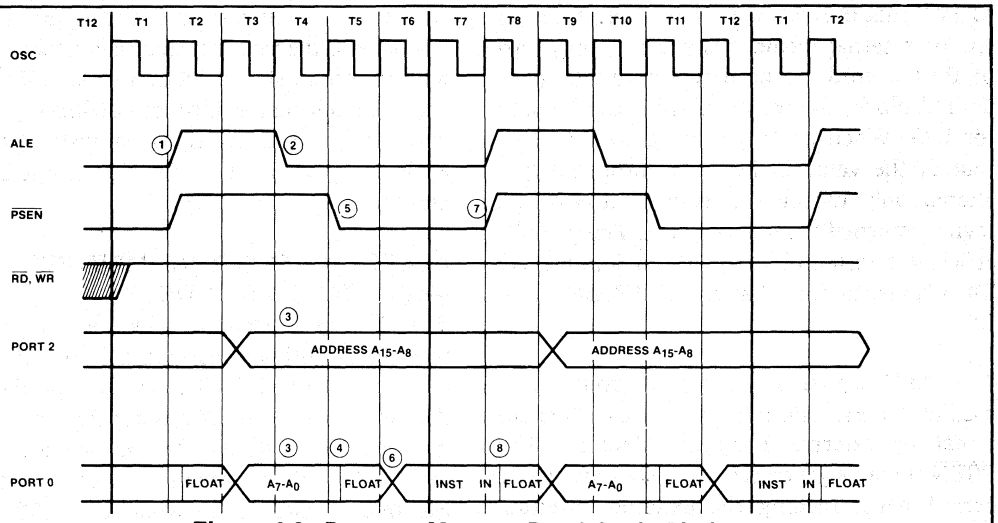


Figure 2-6. Program Memory Read Cycle Timing

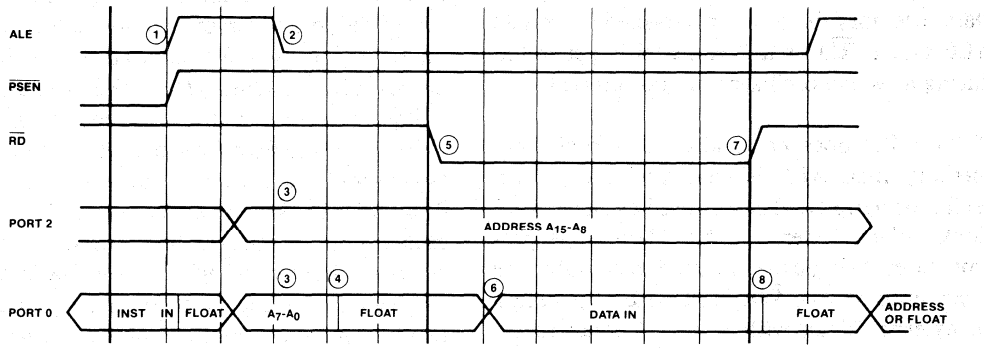
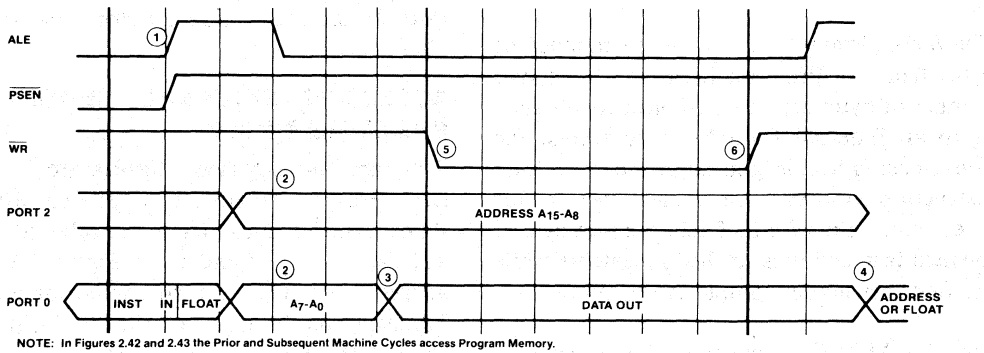


Figure 2-7. Data Memory Read Cycle Timing



NOTE: In Figures 2.42 and 2.43 the Prior and Subsequent Machine Cycles access Program Memory.

Figure 2-8. Data Memory Write Cycle Timing

fetching from external Program Memory, the 8051 will always fetch an even number of bytes. If an odd number of bytes are executed prior to a branch or an External Data Memory access the non-executed byte will be ignored by the 8051. An even number of idle bus cycles (each 6 oscillator periods in duration) can occur between external bus cycles when the processor is fetching from Internal Program Memory. The read cycle begins during T2, with the assertion of address latch enable signal ALE ①. The falling edge of ALE ② is used to latch the address information, which is present on the bus at this time ③, into the 8282 latch if a non-multiplexed bus is required. At T5, the address is removed from the Port 0 bus and the processor's bus drivers go to the high-impedance state ④. The program memory read control signal ($\overline{PSEN}$) ⑤ is also asserted during T5. $\overline{PSEN}$ causes the addressed device to enable its bus drivers to the now-released bus. At some later time, valid instruction data will become available on the bus ⑥. When the 8051 subsequently returns $\overline{PSEN}$ to the high level ⑦, the addressed device will then float its bus drivers, relinquishing the bus again ⑧.

Data Memory Read Sequence

Each External Data Memory bus cycle consists of twelve oscillator periods. These are shown at T1 through T12 on Figure 2-7. The twelve period External Data Memory cycle allows the 8051 to use peripherals that are relatively slower than its program memories. The address is emitted from the processor during T3. Data transfer occurs on the bus during T7 through T12. T5 and T6 is the period during which the direction of the bus is changed for the read operation. The read cycle begins during T2, with the assertion of address latch enable signal ALE ①. The falling edge of ALE ② is used to latch the address information, which is present on the bus at this time ③, into the 8282 latch if a non-multiplexed bus is desired. At T5, the address is removed from the Port 0 bus and the processor's bus drivers go to the high-impedance state ④. The data memory read control signal $\overline{RD}$ ⑤, is asserted during T7. $\overline{RD}$ causes the addressed device to enable its bus drivers to the now-released bus. At some later time, valid data will become available on the bus ⑥. When the 8051 subsequently returns $\overline{RD}$ to the high level ⑦, the addressed device will then float its bus drivers, relinquishing the bus again ⑧.

Data Memory Write Sequence

The write cycle (Figure 2-8), like the read cycle, begins with the assertion of ALE ① and the emission of an address ②. In T6, the processor emits the data to be written into the addressed data memory location ③. This data remains valid on the bus until the end of the following bus cycle's T2 ④. The write signal $\overline{WR}$ goes low at T6 ⑤ and remains active through T12 ⑥.

2.3.5 Macro-View of the 8051 Timer/Event Counters

The 8051 contains two 16-bit counters for measuring time intervals, measuring pulse widths, counting events and generating precise, periodic interrupt requests. Each can be programmed independently to operate similar to an 8048 8-bit timer with divide by 32 prescaler or as an 8-bit counter with divide by 32 prescaler (Mode 0), as a 16-bit time-interval or event counter (Mode 1), or as an 8-bit time-interval or event counter with automatic reload upon overflow (Mode 2).

Additionally, counter 0 can be programmed to a mode that divides it into one 8-bit time-interval or event counter and one 8-bit time-interval counter (Mode 3). When counter 0 is in Mode 3, counter 1 can be programmed to any of the three aforementioned modes, although it cannot set an interrupt request flag or generate an interrupt. This mode is useful because counter 1's overflow can be used to pulse the serial port's transmission-rate generator. Along with their multiple operating modes and 16-bit precision, the counters can also handle very high input frequencies. These range from 0.1 MHz to 1.0 MHz (for 1.2 MHz to 12 MHz crystal) when programmed for an input that is a division by 12 of the oscillator frequency and from 0 Hz to an upper limit of 50 KHz to 0.5 MHz (for 1.2 MHz to 12 MHz crystal) when programmed for external inputs. Both internal and external inputs can be gated to the counter by a second external source for directly measuring pulse widths.

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 auto-reload value). The operating modes and input sources are summarized in Figures 2-9A and 2-9B.

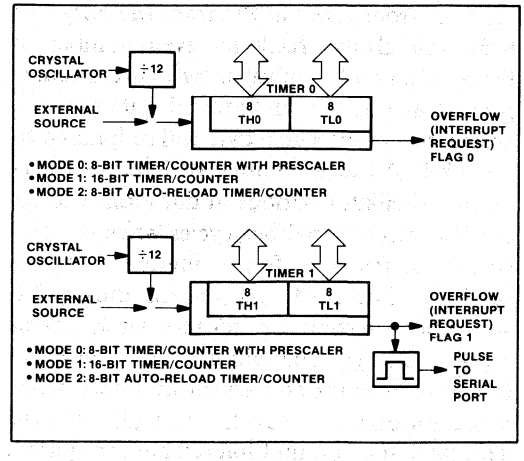


Figure 2-9A. Timer/Event Counter Modes 0,1 and 2

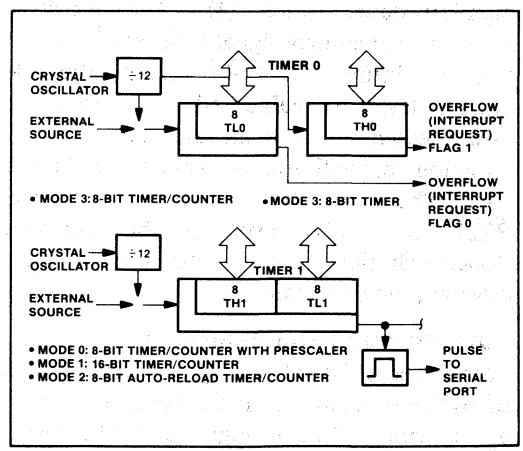


Figure 2-9B. Timer/Event Counter 0 in Mode 3

2.3.6 Timer/Counter Mode Selection (Functional Description)

Counter 1 can be configured in one of four modes:

Mode 0) Provides an 8-bit counter with a divide-by-32 prescaler or an 8-bit timer with a divide-by-32 pre-

scaler. A read/write of TH1 accesses counter 1's bits 12-5. A read/write of TL1 accesses counter 1's bits 7-0. TL1 bits 4-0 are the prescaler (counter 1's bits 4-0) while bits 7-5 are indeterminate and should be ignored. The programmer should clear the prescaler (counter 1's bits 4-0) before setting the run flag.

Mode 1) Configures counter 1 as a 16-bit timer/counter.

Mode 2) Configures counter 1 as an 8-bit auto-reload timer/counter. TH1 holds the reload value. TL1 is incremented. The value in TH1 is reloaded into TL1 when TL1 overflows from all ones (1's). An 8048 compatible counter is achieved by configuring to mode 2 after zero-ing TH1.

Mode 3) When counter 1's mode is reprogrammed to mode 3 (from mode 0, 1 or 2), it disables the incrementing of the counter. This mode is provided as an alternative to using the TR1 bit (TCON.6) to start and stop counter 1.

The serial port receives a pulse each time that counter 1 overflows. The standard UART modes divide this pulse rate to generate the transmission rate.

Counter 0 can also be configured in one of four modes:

Modes 0-2) Modes 0-2 are the same as for counter 1.

Modes 3) In Mode 3, the configuration of TH0 is not affected by the bits in TMOD or TCON (see next section). It is configured solely as an 8-bit timer that is enabled for incrementing by TCON's TR1 bit. Upon TH0's overflow the TF1 flag gets set. Thus, neither TR1 nor TF1 is available to counter 1 when counter 0 is in Mode 3. The function of TR1 can be done by placing counter 1 in Mode 3, so only the function of TF1 is actually given up by counter 1. In Mode 3, TL0 is configured as an 8-bit timer/counter and is controlled, as usual, by the Gate (TMOD.3), C/$\overline{T}$ (TMOD.2), TR0 (TCON.4) and TF0 (TCON.5) control bits.

CONFIGURING THE TIMER/COUNTER INPUT

The use of the timer/counters is determined by two 8-bit registers, TMOD (timer mode) and TCON (timer control). The counter input circuitry is shown in Figures 2-10A and 2-10B. The input to the counter circuitry is from an external reference (for use as a counter), or from the on-chip oscillator (for use as a timer), depending on whether TMOD's C/$\overline{T}$ bit is set or cleared, respectively. When used as a time base, the on-chip oscillator frequency is divided by twelve (12) before being input to the counter circuitry. When TMOD's Gate bit is set (1), the external reference input (T1, T0) or the oscillator input is gated to the counter conditional upon a second external input ($\overline{INT0}$, $\overline{INT1}$) being high. When the Gate bit is zero (0), the external reference or oscillator input is unconditionally enabled. In either case, the normal interrupt function of $\overline{INT0}$ and $\overline{INT1}$ is not affected by the counter's operation. If enabled, an interrupt will occur when the input at $\overline{INT0}$ or $\overline{INT1}$ is low. The counters are enabled for incrementing when TCON's TR1 and TR0 bits are set. When the counters overflow

the TF1 and TF0 bits in TCON get set and interrupt requests are generated. The functions of the bits in TCON are shown in Table 2-9. The functions of the bits in TMOD are shown in Table 2-10.

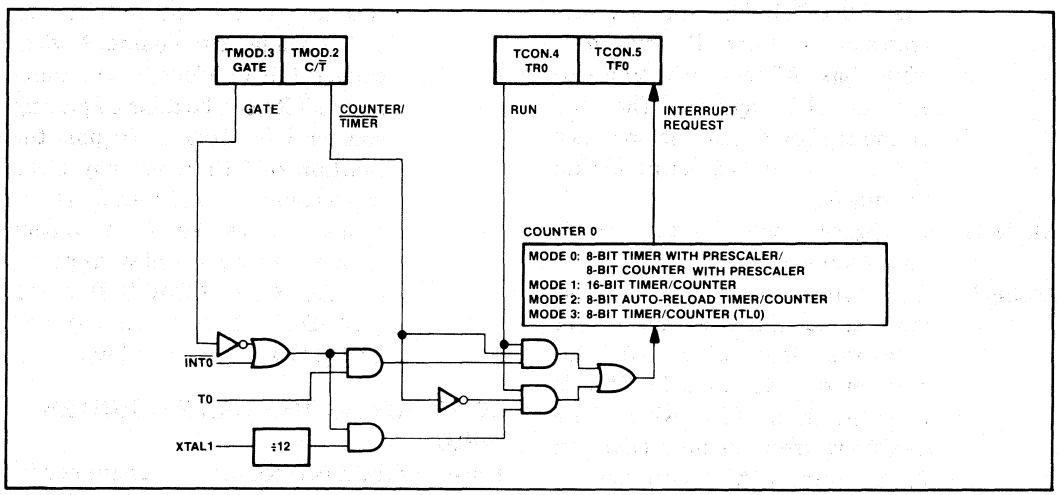


Figure 2-10A. Timer/Event Counter 0 Control and Status Flag Circuitry

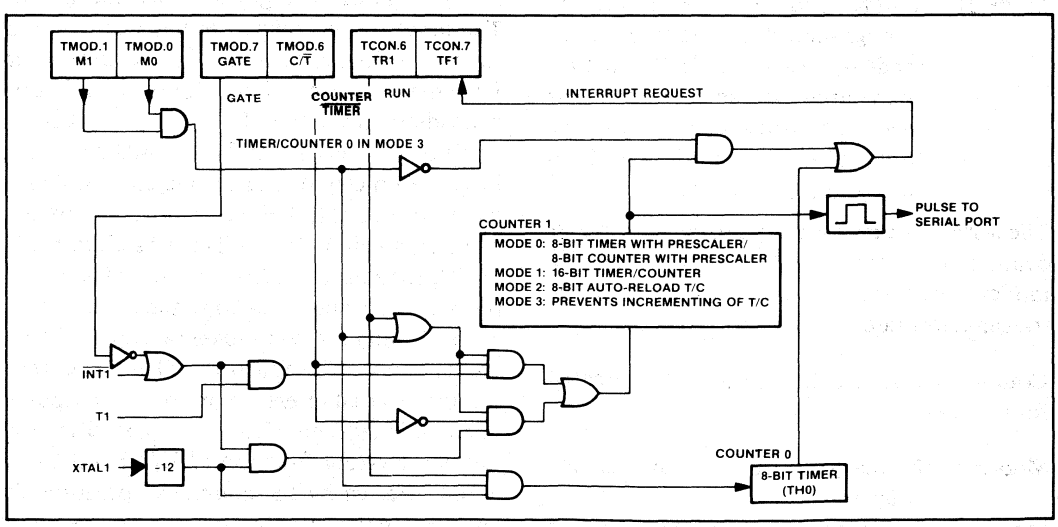


Figure 2-10B. Timer/Event Counter 1 Control and Status Flag Circuitry

(MSB)							(LSB)
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

Function	Flag	Bit Location	Function	Flag	Bit Location
Timer 1 overflow Flag. Set by hardware on timer/counter overflow. Cleared when interrupt processed.	TF1	TCON.7	Interrupt 1 Edge flag. Set by hardware when external interrupt edge detected. Cleared when interrupt processed.	IE1	TCON.3
Timer 1 Run control bit. Set/cleared by software to turn timer/counter on/off.	TR1	TCON.6	Interrupt 1 Type control bit. Set/cleared by software to specify falling edge/low level triggered external interrupts.	IT1	TCON.2
Timer 0 overflow Flag. Set by hardware on timer/counter overflow. Cleared when interrupt processed.	TF0	TCON.5	Interrupt 0 Edge flag. Set by hardware when external interrupt edge detected. Cleared when interrupt processed.	IE0	TCON.1
Timer 0 Run control bit. Set/cleared by software to turn timer/counter on/off.	TR0	TCON.4	Interrupt 0 Type control bit. Set/cleared by software to specify falling edge/low level triggered external interrupts.	IT0	TCON.0

Table 2-9. TCON—Timer/Counter Control/Status Register

OPERATION

The counter circuitry counts up to all 1's and then overflows to either 0's or the reload value. Upon overflow, TF1 or TF0 gets set. When an instruction changes the timer's mode or alters its control bits, the actual change occurs at the end of the instruction's execution.

The T1 and T0 inputs are sampled near the falling-edge of ALE in the tenth, twenty-second, thirty-fourth and forty-sixth oscillator periods of the instruction-in-progress. They are also sampled in the twenty-second oscillator period of MOVX despite the absence of ALE. Thus, an external reference's high and low times must each be a minimum of twelve oscillator periods in duration. There is a twelve oscillator period delay from when a toggled input (transition from high to low) is sampled to when the counter is incremented.

READING AND RELOADING THE TIMER/COUNTERS

The timer/counters can be read and reloaded on the fly. However, the 16-bit timer/counters must be read and loaded as two 8-bit bytes. During a read the potential "phasing error" can be programmed around, as follows:

```
RTC  MOV A, TH0
     MOV B, TL0
     CJNE A, TH0, RTC
```

(MSB)							(LSB)
GATE	C/T	M1	M0	GATE	C/T	M1	MO
TIMER 1				TIMER 0			

Function	Flag	Bit Location
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.	GATE	TMOD.7
Timer or Counter Selector. Cleared for Timer operation (input from internal system clock.) Set for Counter operation (input from "Tx" input pin).	C/T	TMOD.6,.2

Table 2-10. TMOD—Timer/Counter Mode Register

2.3.7 Macro-View of the 8051 Serial Communication Port

The 8051 has a serial I/O port that is useful for serially linking peripheral devices as well as multiple 8051s through standard asynchronous protocols with full-duplex operations. The serial port also has a synchronous mode for expansion of I/O lines using CMOS and TTL shift registers. This hardware serial communications interface saves ROM code and permits a much higher transmission rate than could be achieved through software. In response to a serial port interrupt request the CPU has only to read/write the serial port's buffer to service the serial link. A block diagram of the serial port is shown in Figures 2-11A and B.

The full-duplex serial I/O port provides asynchronous modes to facilitate communications with standard UART devices, such as printers and CRT terminals, or communications with other 8051s in multi-processor systems. 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. Double buffering

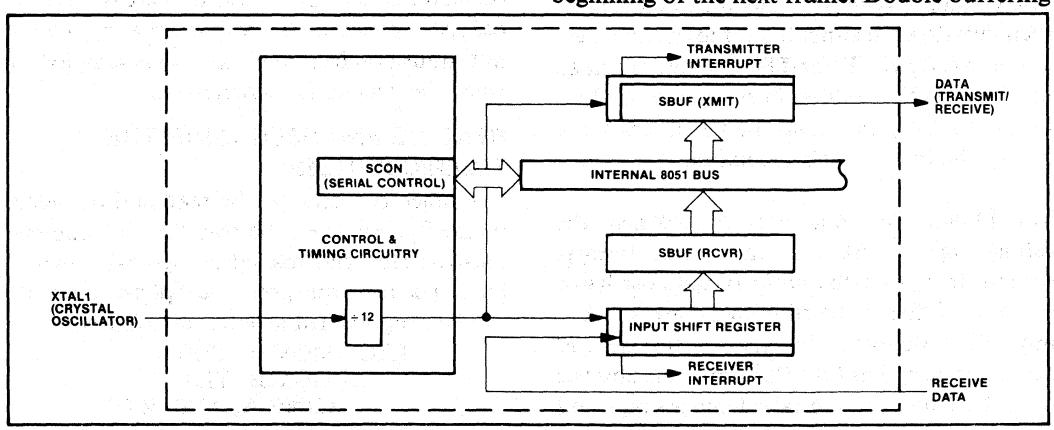


Figure 2-11A. Serial Port—Synchronous Mode 0

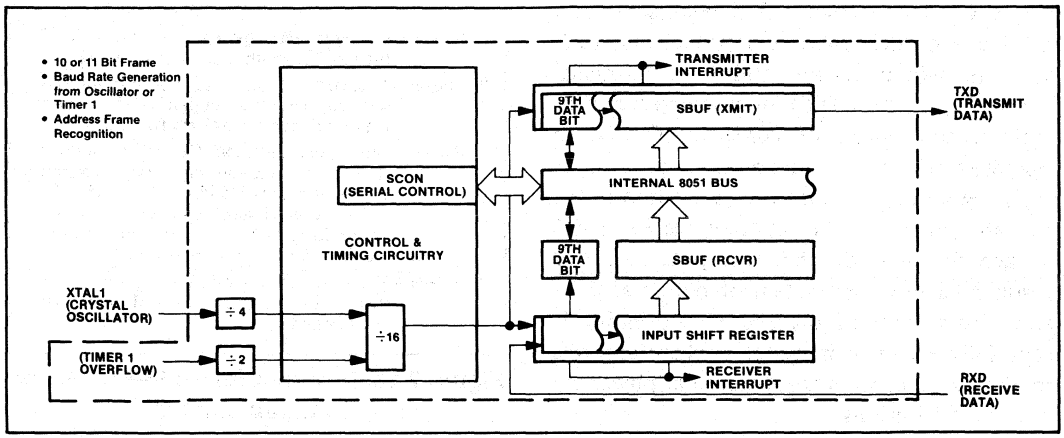


Figure 2-11B. Serial Port—UART Modes 1, 2, and 3

of the transmitter is not needed since the 8051 can generally maintain the serial link at its maximum rate without it. A minor degradation in transmission rate can occur in rare events such as when the servicing of the transmitter has to wait for a lengthy interrupt service program to complete. In asynchronous modes, false start-bit rejection is provided on received frames. For noise rejection a best two-out-of-three vote is taken on three samples near the center of each received bit.

When interfacing with standard UART devices the serial channel can be programmed to a mode (Mode 1) that transmits/ receives a ten-bit frame or programmed to a mode (Mode 2 or 3) that transmits/receives an eleven-bit frame as shown in Figure 2-12. The frame consists of a start bit, eight or nine data bits and a stop bit. In Modes 1 and 3, the transmission-rate timing circuitry receives a pulse from counter 1 each time the counter overflows. The input to counter 1 can be an external source or a division by 12 of the oscillator frequency. The auto-reload mode of the counter provides communication rates of 122 to 31,250 bits per second (including start and stop bits) for a 12 MHz crystal. In Mode 2 the communication rate is a division of 64 of the oscillator frequency yielding a transmission rate of 187,500 bits per second (including start and stop bits) for a 12 MHz crystal.

Distributed processing offers a faster, more powerful system that can be provided by a

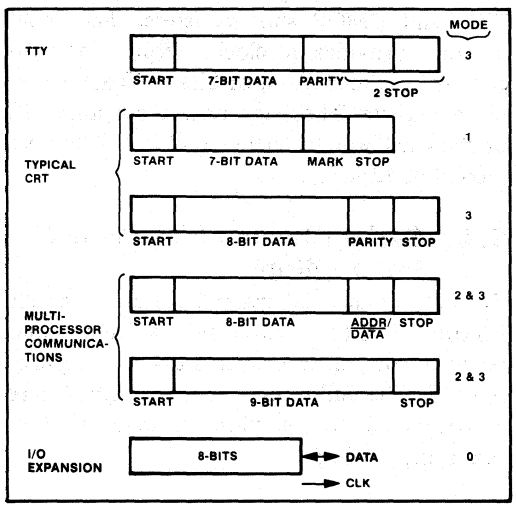


Figure 2-12. Typical Frame Formats

single CPU processor. This results from a hierarchy of interconnected processors, each with its own memories and I/O. In multiprocessing, a host 8051 microcomputer controls a multiplicity of 8051s configured to operate simultaneously on separate portions of the program, each controlling a portion of the overall process. The interconnected 8051s reduce the load on the host processor and result in a low-cost system of data transmission. This form of distributed processing is especially effective in systems where controls in a complex process are required at physically separated locations.

In Modes 2 and 3 the automatic wake-up of slave processors through interrupt driven address-frame recognition is provided to facilitate interprocessor communications. The protocol for interprocessor communications is shown in Figure 2-13.

In synchronous mode (Mode 0) the high-speed serial port provides an efficient, low-cost method of expanding I/O lines using standard TTL and CMOS shift registers. The serial channel provides a clock output for synchronizing the shifting of bits to/from an external register. The data rate is a division by 12 of the oscillator frequency and is 1M bits per second at 12 MHz.

2.3.8 Serial Channel (Functional Description)

The 8051 has a serial channel useful for serially linking UART (universal asynchronous receiver/transmitter) devices and for expanding I/O. This full-duplex serial I/O port can be programmed to function in one of four operating modes:

Mode 0) Synchronous I/O expansion using TTL or CMOS shift registers

1. **Slaves** – Configure serial port to interrupt CPU if the received ninth data bit is a one (1).
2. **Master** – Transmit frame containing address in first 8 data bits and set ninth data bit (i.e., ninth data bit designates address frame).
3. **Slaves** – Serial port interrupts CPU when address frame is received. Interrupt service program compares received address to its address. The slave which has been addressed reconfigures its serial port to interrupt the CPU on all subsequent transmissions.
4. **Master** – Transmit control frames and data frames (these will be accepted only by the previously addressed slave.)

Figure 2-13. Protocol for Multi-Processor Communications

Mode 1) UART interface with 10-bit frame and variable transmission rate

Mode 2) UART interface with 11-bit frame and fixed transmission rate

Mode 3) UART interface with 11-bit frame and variable transmission rate

Modes 2 and 3 also provide automatic wake-up of slave processors through interrupt driven address-frame recognition for multiprocessor communications.

SERIAL PORT CONTROL AND DATA BUFFER REGISTERS

Data for transmission and from reception reside in the serial port buffer register (SBUF). A write to SBUF updates the transmitter register, while a read from SBUF reads a buffer that is updated by the receiver register if/when flag RI is reset. 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. In general double buffering of the transmitter is not needed for the high performance 8051 to maintain the serial link at its maximum rate. A minor degradation in data rate can occur in rare events such as when the servicing of the transmitter has to wait for a

(MSB)							(LSB)
SM0	SM1	SM2	REN	TB8	RB8	TI	RI

Function	Flag	Bit Location
Serial port Mode control bit 0. Set/cleared by software (see note).	SM0	SCON.7
Serial Port Mode control bit 1. Set/cleared by software (see note).	SM1	SCON.6
Serial Port Mode control bit 2. Set by software to disable reception of frames for which bit 8 is zero.	SM2	SCON.5
Receiver Enable control bit. Set/cleared by software to enable/disable serial data reception.	REN	SCON.4
Transmit Bit 8. Set/cleared by hardware to determine state of ninth data bit transmitted in 9-bit UART mode.	TB8	SCON.3
Receive Bit 8. Set/cleared by hardware to indicate state of ninth data bit received.	RB8	SCON.2
Transmit Interrupt flag. Set by hardware when byte transmitted. Cleared by software after servicing.	TI	SCON.1
Received Interrupt flag. Set by hardware whey byte received. Cleared by software after servicing.	RI	SCON.0

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.

Table 2-11. SCON—Serial Port Control/ Status Register

lengthy interrupt service program to complete. In asynchronous mode, false start-bit rejection is provided on received frames. A two-out-of-three vote is taken on each received bit for noise rejection. The serial port's control and the monitoring of its status is provided by the serial port control register (SCON). The contents of the 8-bit SCON register are shown in Table 2-11.

Mode control bits SM0 and SM1 program the serial port in one of four operating modes. A detailed description of the modes is provided below. The receiver-enable bit (REN) resets the receiver's start/stop logic. When software sets REN to one (1), the receiver's transmission-rate generator is initialized and reception is enabled. REN must be set as part of the serial channel's initialization program. When REN is cleared, reception is disabled.

The CPU is informed that the transmitter portion of the SBUF is empty or the receiver portion is full by TI and RI respectively. TI and RI must be cleared as part of the interrupt service program so as not to continuously interrupt the CPU. Since TI and RI are or-ed together to generate the serial port's interrupt request, they must be polled to determine the source of the interrupt.

OPERATING MODES

Operating Mode 0

The I/O expansion mode, Mode 0, is used to expand the number of input and output pins. In this mode, a clock output is provided for synchronizing the shifting of bits into or from an external register. Eight bits will be shifted out each time a data byte is written to the serial channel's data buffer (SBUF), even if TI is set. Each time software clears the RI flag, eight bits are shifted into SBUF before the RI flag is

again set. The receiver must be enabled [i.e., REN set to one (1)] for reception to occur.

The synchronizing clock is output on pin P3.1 and toggles from high to low near the falling-edge of ALE in the fifteenth oscillator period following execution of the instruction that updated SBUF or cleared the RI flag. It then toggles near the falling-edge of ALE in each subsequent sixth oscillator period until 8-bits are transferred. The eighth rising-edge of clock (P3.1) sets the RI or TI flag. At this point shifting is complete and the clock is once again high. The first bit is shifted out of P3.0 at the beginning of the eighteenth oscillator period following the instruction that updated SBUF. The first bit shifted in from P3.0 is latched by the clock's rising-edge in the twenty-fourth oscillator period following the instruction that cleared the RI flag. One bit is shifted every twelfth oscillator period until all eight bits have been shifted.

Operating Modes 1-3

In the UART Modes (i.e., 1 through 3), the transmission rate is sub-divided into 16 "ticks." The value of a received bit is determined by taking a majority vote after it has been sampled during the seventh, eighth and ninth "ticks". If two or three ones (1's) are detected, the bit will be given a one (1) value; if two or three zeros (0's) are detected, the bit will be given a zero (0) value.

Until a start bit arrives, the receiver samples the RXD input pin (P3.0) every "tick". One-half bit time (eight "ticks") after the start bit is detected (i.e., a low input level was sampled on "tick" one), the serial port checks its validity (majority vote from "ticks" seven, eight and nine) and accepts or rejects it. This provides rejection or false start bits.

The contents of the receiver's input shift register is moved to SBUF and RB8 (Modes 2 and 3), and RI is set, when a frame's ninth (Mode 1) or tenth (Modes 2 and 3) bit is received. Upon reception of a second frame's ninth or tenth bit, the data bits in the shift register are again transferred to SBUF and RB8, but only if software has reset the RI flag. If RI has not been reset, the overrun will occur since the shift register will continue to accept bits. Double buffering the receiver provides the CPU with one frame-time in which to empty the SBUF and RB8 registers. The RI flag is set and bit RB8 is loaded during the ninth "tick" of the received frame's ninth or tenth bit. The serial port begins looking for the next start bit one-half bit time after the center of a stop bit is received.

Data is transmitted from the TXD output pin (P3.1) each time a byte is written to SBUF, even if TI is set. Transmission of the start bit begins at the end of the instruction that updates SBUF. TI is set at the beginning of the transmitted tenth (Mode 1) or eleventh (Modes 2 or 3) bit. After TI becomes set, if SBUF is written-to prior to the end of the stop (tenth or eleventh) bit, the transmission of the next frame's start bit will not begin until the end of the stop bit.

In Modes 2 and 3, if SM2 is set, frames are received but an interrupt request is generated only when the received data bit 8 (RB8) is a one (1). This feature permits interrupt generated wake-up during interprocessor communications when multiple 8051's are connected to a serial bus. Thus, data bit 8 (RB8) awakens all processors on the serial bus only when the master is changing the address to a different processor. Each processor not addressed then ignores the subsequent transmission of the control information and data. A

1. **The hardware in each slave's serial port begins by listening for an address. Receipt of an address frame will force an interrupt if the slave's SM2 bit is set to one (1) to enable "interrupt on address frame only".**
2. **The master than transmits a fram containing the 8-bit address of the slave that is to receive the subsequent commands and data. A transmitted address frame has its ninth data bit (TB8) set equal to one (1).**
3. **When the address frame is received, each slave's serial port interrupts its CPU. The CPU then compares the address sent to its own.**
4. **The 8051 slave which has been addressed then resets its SM2 bit to zero (0) to receive all subsequent transmissions. All other 8051's leave their SM2 bits at a one (1) to ignore transmissions until a new address arrives.**
5. **The master device then sends control information and data, which in turn is accepted by the previously addressed 8051 [i.e., the one that had set its SM2 bit to zero (0)].**

Figure 2-14. Protocol for Multi-Processor Communications

protocol for multi-8051 serial communications is shown in Figure 2-14. The SM2 bit has no effect in Modes 0 and 1.

THE SERIAL FRAME

A frame is a string of bits. The frame transmitted and received in Mode 0 is 8 bits in length. The data bits of the frame are transmitted SBUF.0 first and SBUF.7 last.

The frame transmitted and received in Mode 1 is ten bits in length. The frame transmitted and received in Modes 2 and 3 is eleven bits in length. These frames consist of one start bit, eight or nine data bits and a stop bit. Data bits 0-7 are loaded into SBUF.0-SBUF.7 respectively, and data bit 8 into RB8 (receive) or TB8 (transmit). The data bits of the frame are

```
MOV C,P      ; Parity moved to carry (byte
               already in A).
MOV TB8,C    ; Put carry into Transmitter Bit 8
MOV SBUF,A;  Load Transmit Register
```

Figure 2-15. Generating Parity and Transmitting Frame

transmitted least significant bit first (SBUF.0) and TB8 last. With nominal software overhead, the last data bit can be made a parity bit, as shown in Figure 2-15.

TRANSMISSION RATE GENERATION

The proper timing for the serial I/O data is provided by a transmission-rate generator. On-board the 8051, three different methods of transmission rate generation are provided. The transmission-rate achievable is dependent upon the operating mode of the serial port.

In the I/O expansion mode (Mode 0) the oscillator frequency is simply divided by 12 to generate the transmission rate. This produces a transmission rate of 1M bits per second at 12 MHz. If Modes 1 or 3 are being used, the transmission rate can be generated from the oscillator frequency or from an external reference frequency. In these modes, either one-twelfth the oscillator frequency or the T1 input frequency is divided by 256-minus-the-value-in-TH1 (counter 1 must be configured in auto-reload mode by software) and then divided by 32 to generate the transmission rate. When the oscillator frequency input (rather than T1) is selected, this method produces a transmission rate of 122 to 31,250 bits per second (including start and stop bits) at 12 MHz. The T1 external input is selected by setting the C/$\overline{T}$ bit to one (1). When Mode 2 is used, the oscillator frequency is simply divided by 64 to generate the transmission rate. This produces a

transmission rate of 187,500 bits per second (including start and stop bits) at 12 MHz.

UART ERROR CONDITIONS

There are two UART error conditions that should be accounted for when designing systems that use the serial channel.

First, the 8051's serial channel provides no indication that a valid stop bit has been received. However, since a start bit is detected as a high-level to low-level transition, the UART will not receive additional frames if a stop bit is not received.

Second, the RI flag is set and SBUF and RB8 are loaded from the receiver's input shift register when the received last data bit (i.e. ninth or tenth received bit) is sampled. As long as RI is set, the loading of SBUF, the updating of RB8 and the generation of further receiver interrupts is inhibited. Thus, overrun will occur if the programmer does not reset RI before reception of the next frame's last data bit since the receiver's input shift register will shift in a third frame.

Register	Content
PC	0000H
SP	07H
PSW, DPH, DPL, A, B,	00H
IP	E0H or 00H
IE	60H or 00H
SCON, TCON, TMOD,	00H
TH1, TH0	00H
TL1, TL0	00H
SBUF	Indeterminate
Port 3-Port 0	FFH (configures all I/O pins as inputs)
Internal RAM	Unchanged if VPD applied; else indeterminate

Table 2-12. Register Initialization

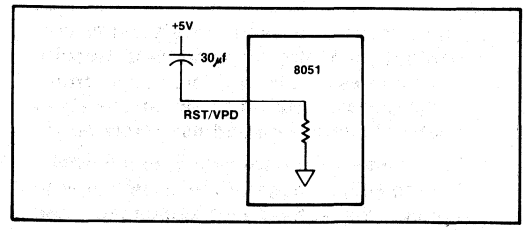


Figure 2-16. Power-On Reset

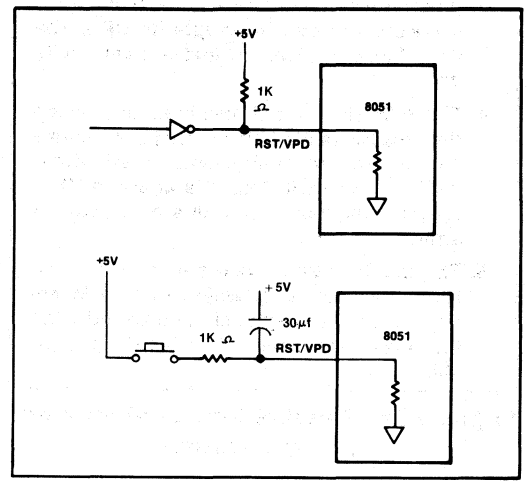


Figure 2-17. External Reset

2.4 EXTERNAL INTERFACE

2.4.1 Processor Reset and Initialization

Processor initialization is accomplished with activation of the RST/VPD pin. To reset the processor, this pin should be held high for at least twenty-four oscillator periods. Upon powering up, RST/VPD should be held high for at least 1 ms after the power supply stabilizes to allow the oscillator to stabilize. Upon receipt of RST, the processor ceases instruction execution and remains dormant for the duration of the pulse. The pins assume their initialization states within 24 oscillator periods. The low-going transition then initiates

a sequence which requires approximately twelve oscillator periods to execute before ALE is generated and normal operation commences with the instruction at absolute location 0000H. This sequence ends with registers initialized as shown in Table 2-12.

In addition, certain of the control pins are driven to a TTL high level during initialization. These are ALE/$\overline{PROG}$ and $\overline{PSEN}$. Thus, no ALE or $\overline{PSEN}$ signals are generated while RST/VPD is high. When the processor is reset all ports are immediately written with ones (1's).

The Schmitt-trigger input has a small internal pull down resistor which permits power-on reset (as shown in Figure 2-16) using only a small capacitor tied to VCC. A conventional external reset circuit, such as that in Figure 2-17, can also be used.

2.4.2 Power Down (Standby) Operation of Internal RAM

Data can be maintained valid in the Internal Data RAM while the remainder of the 8051 is powered down. When powered down, the 8051 consumes about 10% of its normal operating power. During normal operation, both the CPU and the internal RAM derive their power from VCC. However, the internal RAM will derive its power from RST/VPD when the voltage on VCC is more than a diode drop below that on RST/VPD.

When a power-supply failure is imminent, the user's system generates a "power-failure" signal to interrupt the processor via $\overline{INT0}$ or $\overline{INT1}$. This power-failure signal must be early enough to allow the 8051 to save all data that is relevant for recovery before VCC falls below its operating limit. The program servicing the power-failure interrupt request must save any important data and machine status into the Internal Data RAM. The service program must also enable the backup power supply to the RST/VPD pin. Applying power to the RST/VPD pin resets the 8051 and retains the internal RAM data valid as the VCC power supply falls below limit. Normal operation resumes when RST/VPD is returned low. Figure 2-18 showns the waveforms for the power-down sequence.

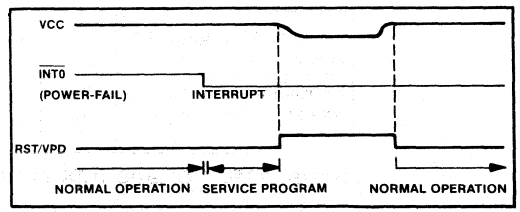


Figure 2-18. Power-Down Sequence

2.4.3 Oscillator and Timing Circuitry

Timing generation for the 8051 is completely self-contained, except for the frequency reference which can be a crystal or external clock source. The on-board oscillator is a parallel anti-resonant circuit with a frequency range of 1.2 to 12 MHz. The XTAL2 pin is the output of a high-gain amplifier, while XTAL1 is its input. A crystal connected between XTAL1 and XTAL2 provides the feedback and phase shift required for oscillation. The 1.2 to 12 MHz range is also accomodated when an external TTL compatible clock is applied to XTAL1 as the frequency source.

2.4.4 EPROM Programming

The 8751 is programmed, and the 8051 and 8751 are verified, using the UPP-551 programming card. For programming and verification, address is input on Port 1 and Port pins 2.0-2.3. Pins P2.4 and P2.5 are held to a TTL low (V_{IL}). Data is input and output through Port 0. RST/VPD is held at TTL high level

PINS / MODE	VPD/RST	PSEN	PROG/ALE	VDD/EA	P24-26	P27
PROGRAM	V_{IH1}	V_{IL}	V_{IH}	V_{PP}	V_{IL}	V_{IL}
VERIFY	V_{IH1}	V_{IL}	V_{IL}	V_{CC}	V_{IL}	V_{IL} (enable) V_{IH} (disable)

Figure 2-19. Voltage Inputs for EPROM Programming/Verifying

(V_{IH1}) and PSEN is held at TTL low level (V_{IL}) during program and verify. To program, ALE/PROG is held at a TTL low level (V_{IL}). The programming supply voltage is held at 21V. The $\overline{EA}$/VDD pin receives this programming supply voltage. ALE/PROG is held at TTL high level (V_{IH}) to verify the program. Port pin 2.7 forces the Port 0 output drivers to the high impedance state when held at a TTL high level and is held at a TTL low level for verification. Erasure of an 8751 will leave the EPROM programmed to an all one's (1's) state.

When $\overline{EA}$/VDD is at 21V, the 8751 is in the programming mode. It is necessary to put a capacitor between $\overline{EA}$/VDD and ground to block spurious voltage transients. ALE/PROG receives the 50 msec, active low TTL program pulse when the address and data are stable. A program pulse must be applied at each address location to be programmed. You can program any location at any time – either sequentially or at random.

A verify may be performed on the programmed bits to ensure correct programming. Data is output on Port 0 when pin 2.7 is at a TTL low level (V_{IH}). Pull-up resistors must be provided on Port 0 pins during verification. This verification mode can also be used to check the 8051's ROM pattern.

Figure 2-19 describes the levels needed on the pins to program and verify the 8751.

2.4.5 8051 Family Pin Description

V_{SS}
Circuit ground potential.

V_{CC}
+5V power supply during operation, programming and 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 input and output during programming and verification. Port 0 can sink/source two 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 programming and verification. Port 1 can sink/source one TTL load.

Port 2
Port 2 is an 8-bit quasi-bidirectional I/O port. It also emits the high-order 8 bits of address when accessing external memory. It is used for the high-order address and the control signals during programming and verification. Port 2 can sink/source one TTL load.

Port 3
Port 3 is an 8-bit quasi-bidirectional I/O port. It also contains the interrupt, timer, serial port and RD and WR pins that are used by various options. The output latch corresponding to a special function must be programmed to a one (1) for that function to operate. Port 3 can

sink/source one TTL load. The special functions are assigned to the pins of Port 3, as follows:

- RXD/data (P3.0). Serial port's receiver data input (asynchronous) or data input/output (synchronous).
- TXD/clock (P3.1). Serial port's transmitter data output (asynchronous) or clock output (synchronous).
- $\overline{INT0}$ (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.
- T0 (P3.4). Input to counter 0.
- T1 (P3.5). Input to counter 1.
- $\overline{WR}$ (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/V_{PD}

A low to high transition on this pin (at approximately 3V) resets the 8051. If VPD is held within its spec (approximately +5V), 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. A small internal resistor permits power-on reset using only a capacitator connected to VCC.

ALE/$\overline{PROG}$

Provides Address Latch Enable output used for latching the address into external memory during normal operation. Receives the program pulse input during EPROM programming.

$\overline{PSEN}$

The Program Store Enable output is a control signal that enables the external Program Memory to the bus during normal fetch operations.

EA/V_{DD}

When held at a TTL high level, the 8051 executes instructions from the internal ROM/EPROM when the PC is less than 4096. When held at a TTL low level, the 8051 fetches all instructions from external Program Memory. The pin also receives the 21V EPROM programming supply voltage.

XTAL 1

Input to the oscillator's high gain amplifier. A crystal or external source can be used.

XTAL 2

Output from the oscillator's amplifier. Required when a crystal is used.

3

Memory Organization, Addressing Modes, and Data Manipulation

CHAPTER 3
MEMORY ORGANIZATION, ADDRESSING MODES, AND DATA MANIPULATION

3.0 MEMORY ORGANIZATION

In the 8051 family the memory is organized over three address spaces and the program counter. The memory spaces shown in Figure 2-1 are the:

- 16-bit Program Counter
- 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 8051 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 8051 and 8751 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 8051 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 3-1.

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 127 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.

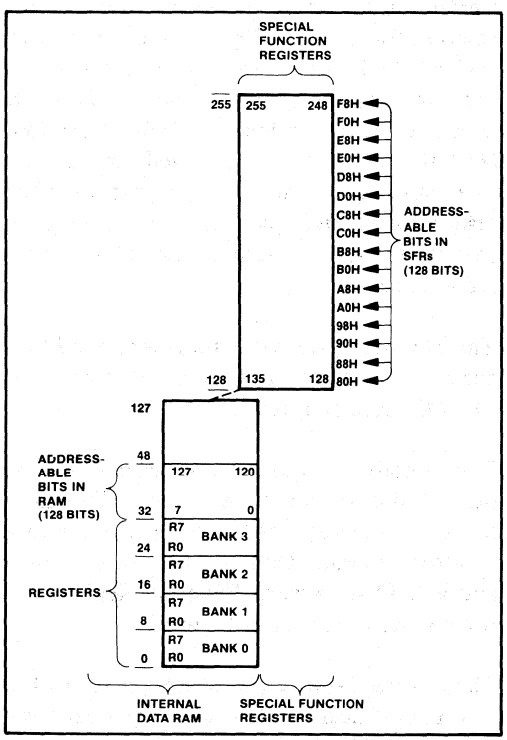


Figure 3-1. Internal Data Memory Address Space

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 addition, 128 bit locations within the Special Function Register address space can be accessed using Direct Addressing. These bits reside in the Special Function Register byte locations divisible by eight. The twenty Special Function Registers are listed in Figure 3-2. Their mapping in the Special Function Register address space is shown in Figures 3-3 and 3-4.

Performing a read from a location of the Internal Data memory where neither a byte of Internal Data RAM (i.e., RAM addresses 128-255) 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.

ARITHMETIC REGISTERS:
ACCumulator*, B register*,
Program 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 I/O PORT:
Serial CONtrol*, Serial data BUFfer

***Bits in these registers are bit addressable**

Figure 3-2. Special Function Registers

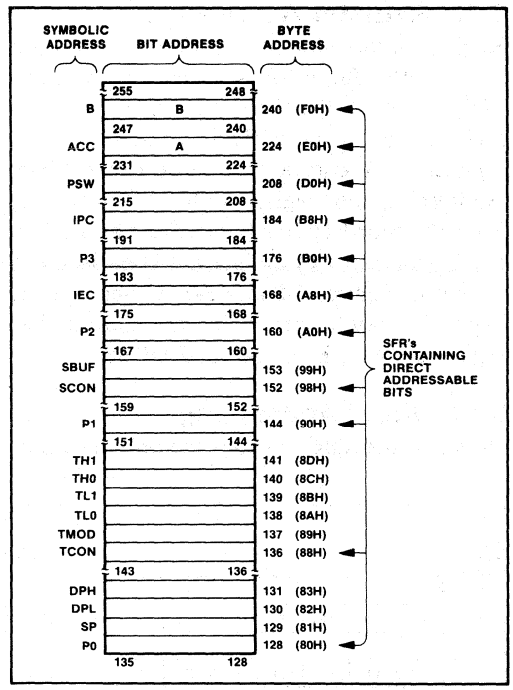


Figure 3-3. Mapping of Special Function Registers

3.1 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 8051 require 0 (NOP only), 1, 2, 3 or 4 operands, these five addressing methods are used in combinations to provide the 8051 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 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 3-5. The following paragraphs describe the five addressing methods.

3.1.1 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 3-6 and 3-7. Other Internal Data Memory locations that are addressed as registers are A, B, C, AB and DPTR.

3.1.2 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 3-3, 3-4, 3-6, and 3-7.

Register-Indirect Addressing using the content of R1 or R0 in the selected Register Bank, or

Direct Byte Address	(MSB)			Bit Addresses				(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	FO	RS1	RS0	OV		P	
208	D7	D6	D5	D4	D3	D2	D1	D0	PSW
				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
	SMO	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 3-4. 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 Internal Data RAM address space**
  - **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 3-5. Operand Addressing Methods

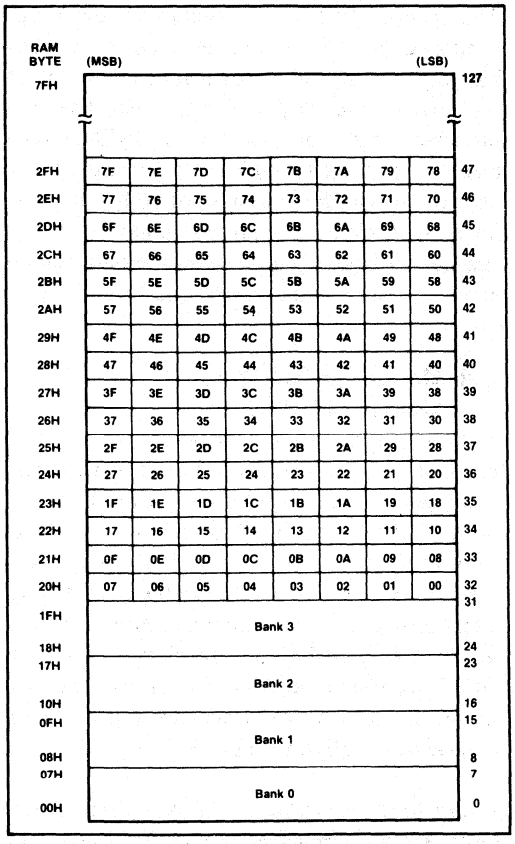


Figure 3-6. RAM Bit Addresses

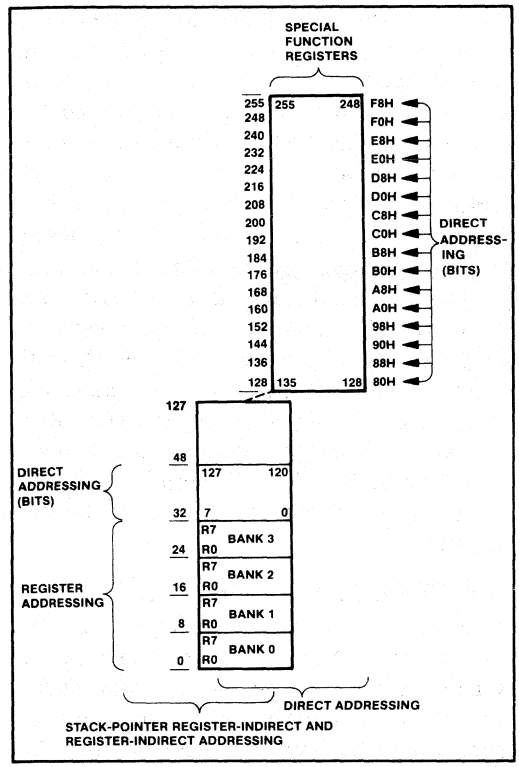


Figure 3-7. Addressing Operands in Internal Data Memory

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.

3.1.3 Immediate Addressing

Immediate Addressing allows constants which are part of the instruction to be accessed from the Program Memory.

3.1.4 Base-Register- plus Index-Register- Indirect Addressing

Base-Register- plus Index-Register- Indirect Addressing simplifies accessing look-up-tables (LUT) resident in Program memory. A byte may be accessed from a LUT via an indirect move from a location whose address is the sum of a base register (the DPTR or PC) and the index register (A).

3.2 DATA MANIPULATION

The 8051 microcomputer is efficient both as an arithmetic processor and as a controller. In ad-

dition to the capabilities of its 8048 predecessor, the 8051 was enhanced with improved data transfer, logic manipulation, arithmetic processing, and real-time control capabilities. The 8051 performs operations on bit, nibble (4-bit), byte (8-bit) and double-byte (16-bit) data types. It is classified as an 8-bit machine since the internal ROM, RAM, Special Function Registers, Arithmetic/Logic Unit (ALU) and the external data bus are each 8-bits wide. The double-byte data type is used only by the Data Pointer and the Program Counter. The Data Pointer can be manipulated as a single double-byte register (DPTR) or as two locations in Internal Data Memory (DPH & DPL). The Program Counter is always manipulated as a single double-byte register.

3.3 BOOLEAN PROCESSOR

Although the Boolean processor is an integral part of the 8051's architecture, it may be considered an independent bit processor since it has its own instruction set, its own accumulator (the carry flag), and its own bit-addressable RAM and I/O.

The bit-manipulation instructions allow the Direct Addressing of 128 bits within the Internal Data RAM and 128 bits within the Special Function Registers. The Special Function Registers with an address evenly divisable by eight (P0, TCON, P1, SCON, P2, IEC, P3, IPC, PSW, A, and B) contain Direct Addressable bits. On any addressable bit, the Boolean processor can perform the bit operations of set, clear, complement, jump-if-set, jump-if-not-set, jump-if-set-then-clear and move to/from carry. Between any addressable bit (or its complement) and the carry flag it can perform the bit operation of logical and or logical or with the result returned to the carry flag.

The bit-manipulation instructions provide optimum code and speed efficiency in "bit-banging" applications such as the control of the 8051's on-chip peripherals. The Boolean processor also provides a straightforward means of converting logic equations (like those used in random logic design) directly into software. Complex cominatorial-logic functions can be resolved without extensive data movement, byte masking and test-and-branch trees.

3.4 DATA TRANSFER OPERATIONS

Look-up-tables resident in Program Memory can be accessed by indirect moves. A byte constant can be transferred to the A register (i.e., accumulator) from the Program memory location whose address is the sum of a base register (the PC or DPTR) and the index register (A). This provides a convenient means for programming translation algorithms such as ASCII to seven segment conversions. The Program Memory move operations are shown diagrammatically in Figure 3-8.

A byte location within a 256-byte block of External Data Memory can be accessed using R1 or R0 in Register-Indirect Addressing. Any location within the full 64K External Data Memory address space can be accessed through Register-Indirect Addressing using a 16-bit base register (i.e., the Data Pointer). These moves are shown in Figure 3-9.

The byte in-code-constant (immediate) moves and byte variable moves within the 8051 are highly orthogonal as detailed in Figure 3-10. When one considers that the accumulator and the registers in the Register Banks can be Direct Addressed, the two-operand data transfer operations allow a byte to be moved between any two of the RB registers, Internal

Data RAM, accumulator and Special Function Registers. Also, immediate operands can be moved to any of these locations. Of particular interest is the Direct Address to Direct Address move which permits the value in a port to be moved to the Internal Data RAM without using any RB registers or the accumulator. The Data Pointer register can be loaded with a double-byte immediate value. Also, the 8051's Boolean Processor can move any Direct Addressed bit to or from the carry register.

The A register can be exchanged with a register in the 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 Special Function Register. The least significant nibble of the A register can also be exchanged with the least

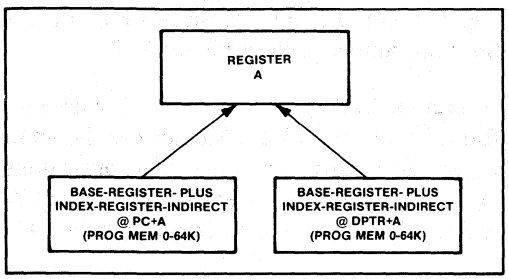


Figure 3-8. Program Memory Move Operations

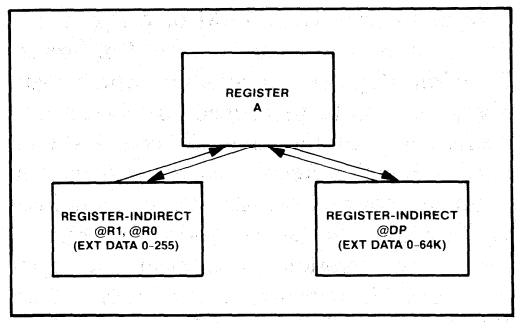


Figure 3-9. External Data Memory Move Operations

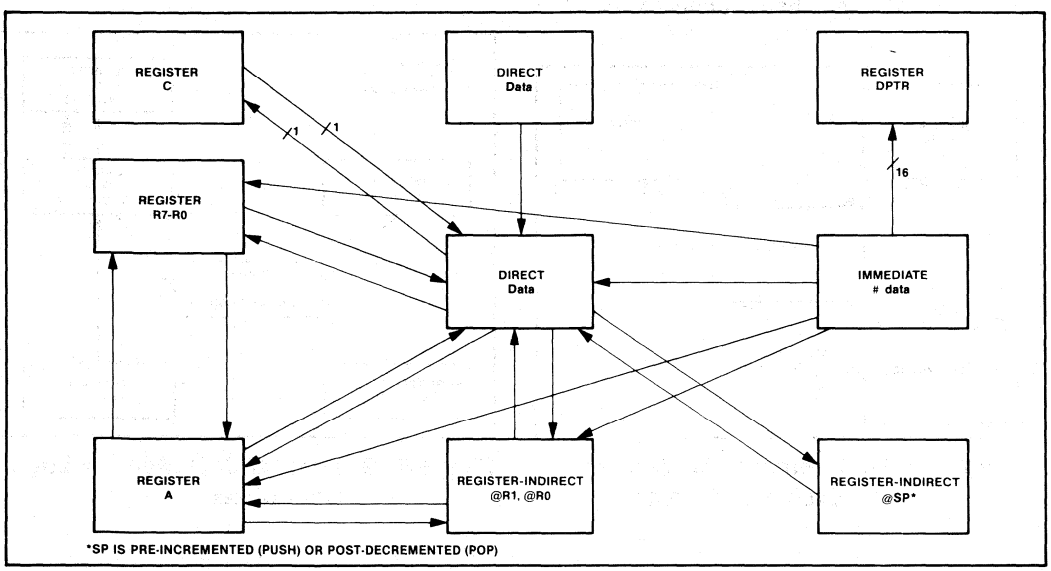


Figure 3-10. Internal Data Memory Move Operations

significant nibble of a Register-Indirect Addressed byte in the Internal Data RAM. The exchange operation is shown in Figure 3-11.

3.5 LOGIC OPERATIONS

The 8051 permits the logic operations of and, or, and exclusive-or to be performed on the A register by a second operand which can be immediate value, a register in the selected Register Bank, a Register-Indirect Addressed byte of Internal Data RAM or a Direct Addressed byte of Internal Data RAM or Special Function Register. In addition, these logic operations can be performed on a Direct Addressed byte of the Internal Data RAM or Special Function Register using the A register as the second operand. Also, use of Immediate Addressing with Direct Addressing permits these logic operations to set, clear or complement any bit anywhere in the Internal Data RAM or Special Function Registers without affecting the PSW, RB registers or accumulator. When one takes into account that registers R7-R0 and the accumulator can be Direct Addressed, the two-operand logic operations allow the destination (first operand) to be a byte in the Internal Data RAM, a Special Function Register, RB registers (R7-R0) or the accumulator while the choice of the second operand can be any of the aforementioned or an immediate value. The 8051 can also perform a logical or, or a logical and, between the Boolean accumulator (i.e, the carry flag) and any bit, or its complement, that can be accessed through Direct Addressing. The and, or, and exclusive-or logic operations are summarized in Figure 3-12.

In addition to the logic operations that are performed on Internal Data Memory as shown in Figure 3-12, there are also logic operations that are performed specifically on the A register. These are summarized in Figure 3-13.

In addition to the and and or bit logicals shown in Figure 3-12, there are logicals that can operate exclusively on a Direct Addressed bit.

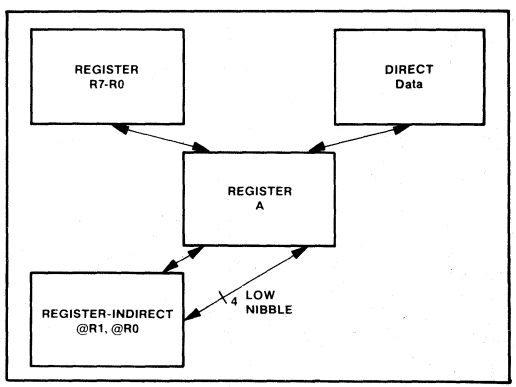


Figure 3-11. Internal Data Memory Exchange Operations

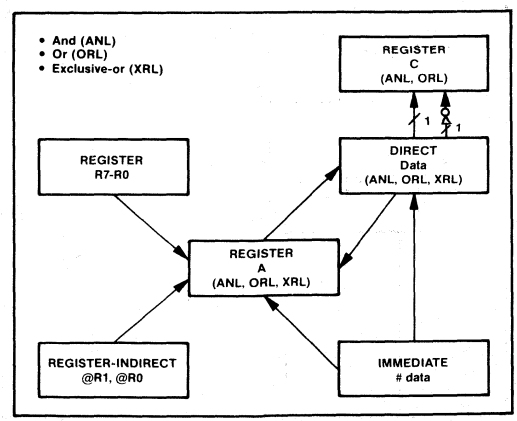


Figure 3-12. Internal Data Memory Logic Operations

These operations are listed in Figure 3-14. The carry flag is also addressed as a register and can be set, cleared or complemented with one-byte instructions.

3.6 ARITHMETIC OPERATIONS

Along with the existing 8048 arithmetic operations of add, increment, decrement, compare-to-zero, decrement-and-compare-to-zero, and decimal-add-adjust, the 8051 implemented subtract-with-borrow, compare, multiply and divide.

Only unsigned binary integer arithmetic is performed in the Arithmetic/Logic Unit. In the two-operand operations of add, add-with-carry and subtract-with-borrow, the A register is the first operand and receives the result of the operation.The second operand can be an immediate byte, a register in the selected Register Bank, a Register-Indirect Addressed byte or a Direct Addressed byte. These instructions affect the overflow, carry, auxiliary-carry and parity flag in the Program Status Word (PSW). The carry flag facilitates non-signed integer and multi-precision addition and subtraction and multi-precision rotation. Handling two's-complement-integer (signed) addition and subtraction can easily be accommodated with software's monitoring of the PSW's overflow flag. The auxiliary-carry flag simplifies BCD arithmetic. An operation that has an arithmetic aspect similar to a subtract is the compare-and-jump-if-not-equal operation. This operation performs a conditional branch if a register in the selected Register Bank, or an Indirect Addressed byte of Internal Data RAM, does not equal an immediate value; or if the A register does not equal a byte in the Direct Addressable Internal Data RAM, or a Special Function Register. While the destination operand is not updated and neither source operand is affected by the compare operation, the carry flag is. A summary of the two-operand add/subtract operations is shown in Figure 3-15.

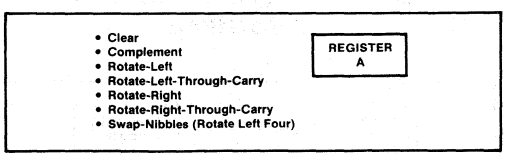


Figure 3-13. Internal Data Memory Logic Operations (Register A Specific)

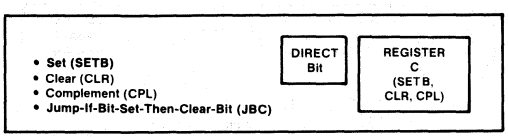


Figure 3-14. Internal Data Memory Logic Operations (Bit-Specific)

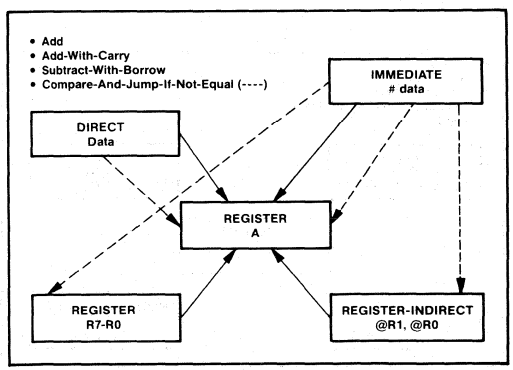


Figure 3-15. Internal Data Memory Arithmetic Operations

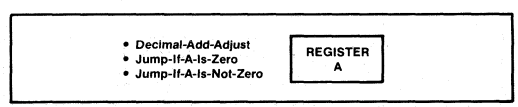


Figure 3-16. Internal Data Memory Arithmetic Operations (Register A Specific)

There are three arithmetic operations that operate exclusively on the A register. These are the decimal-adjust for BCD addition and the two test conditions shown in Figure 3-16. The decimal-adjust operation converts the result from a binary addition to two two-digit BCD values to yield the correct two-digit BDC result. During this operation the auxiliary-carry flag helps effect the proper adjustment. Conditional branches may be taken based on the value in the A register being zero or not zero.

The 8051 simplifies the implementation of software counters since the increment and decrement operations can be performed on the A register, a register in the selected Register Bank, an Indirect Addressed byte in the Internal Data RAM or a byte in the Direct Addressed Internal Data RAM or Special Function Register. The 16-bit Data Pointer can be incremented. For efficient loop control the decrement-and-jump-if-not-zero operation is provided. This operation can test a register in the selected Register Bank, any Special Function Register or any byte of Internal Data RAM accessible through Direct Addressing and force a branch if it is not zero. The increment/decrement operations are summarized in Figure 3-17.

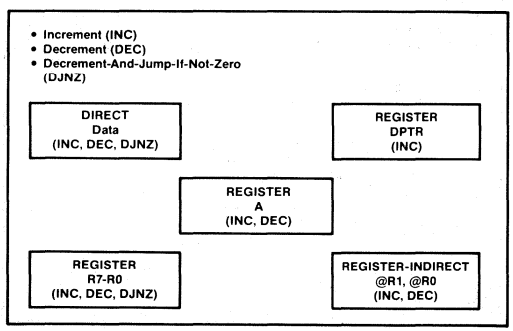


Figure 3-17. Internal Data Memory Arithmetic Operations

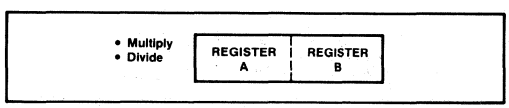


Figure 3-18. Internal Data Memory Arithmetic Operations (Register A with B Specific)

The multiply operation multiplies the one-byte A register by the one-byte B register and returns a double-byte result (MSB in B, LSB in A). The divide operation divides the one-byte A register by the one-byte B register and returns a byte quotient to the A register and a byte remainder to the B register. These are shown in Figure 3-18.

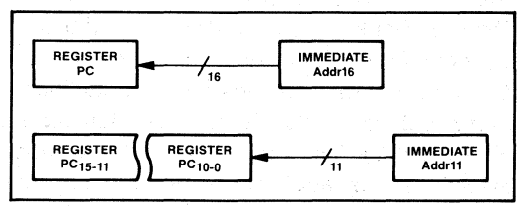


Figure 3-19. Unconditional Branch Operations

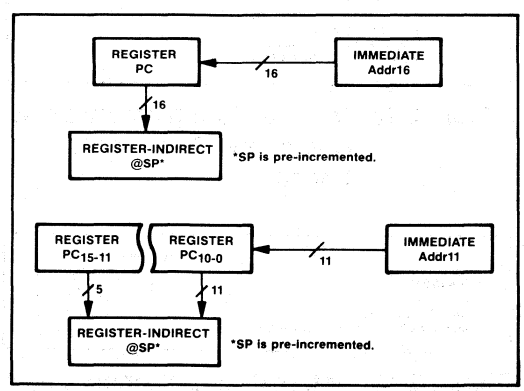


Figure 3-20. Call Operations

3.7 CONTROL TRANSFER

The 8051 has a non-paged Program Memory to accommodate relocatable code. The advantage of a non-paged memory is that a minor change to a program that causes a shift of the code's position in memory will not cause page boundary readjustments to be necessary. This also makes relocation possible. Relocation is desirable since it permits several programmers to write relocatable modules in various assembly and high-level languages which can later be linked together to form the machine object code.

Sixteen-bit jumps and calls are provided to allow branching to any location in the contiguous 64K Program Memory address space and preempt the need for Program Memory bank switching. Eleven-bit jumps and calls are also provided to maintain compatibility with the 8048 and to provide an efficient jump within a 2K program module. Unlike the 8048, the 8051's call operations do not push the Program Status Word (PSW) to the stack along with the Program Counter, since many subroutines written for the 8051 do not affect the PSW. Hence the 8051 return operations pop only the Program Counter. The 8051's branch, call and return operations are shown diagrammatically in Figures 3-19, 3-20, and 3-21, respectively.

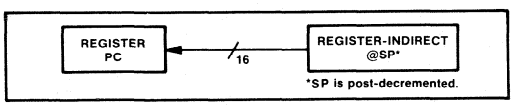


Figure 3-21. Return Operation

The 8051 also provides a method for performing conditional and unconditional branching relative to the starting address of the next instruction (PC - 128 to PC + 127). The bit test operations allow a conditional branch to be taken on the condition of a Direct Addressed bit being set or not set. The accumulator test operations allow a conditional branch based on the accumulator being zero or non-zero. Also provided are compare-and-jump-if-not-equal and decrement-and-compare-to-zero. These are shown in Figure 3-22.

The register-indirect jump in the 8051 permits branching relative to a base register (DPTR) with an offset provided by the non-signed integer value in the index register (A). This accommodates N-way branching. The indirect jump is shown in Figure 3-23.

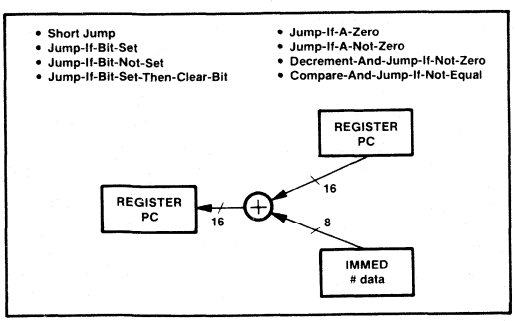


Figure 3-22. Unconditional Short Branch and Conditional Branch Operations

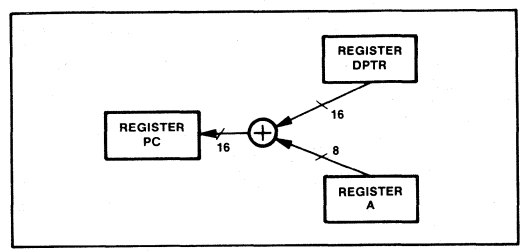


Figure 3-23. Unconditional Branch (Indirect) Operation

8051 Instruction Set 4

CHAPTER 4
8051 INSTRUCTION SET

4.0 WHAT THE INSTRUCTION SET IS

An instruction set is a set of codes that directs a computer to perform its operations. The ease of understanding the instruction set does not depend upon the structure of the machine codes that the computer recognizes, so much as it depends upon the structure of the symbolic language that is used to describe the machine codes.

The 8051 assembly language needs only forty-two mnemonics to specify the 8051's thirty-three functions. A function may have several mnemonics (e.g., MOV, MOVX, MOVC) since the function mnemonic specifies when the Program Memory or External Data Memory is used in conjunction with the Internal Data Memory. When the function mnemonics are combined with unique address combinations specified in the "destination, source" field, 111 instructions are possible. The "destination, source" field specifies the data type and the combination of addressing methods to be used to address the destination and source operands. A summary of the 8051 instruction set is provided in Table 4-1.

The syntax of most 8051 assembly language instructions consists of a function mnemonic followed by a "destination, source" operand field. Thus "MOV @R0, Data" may be interpreted as "The content of the Internal Data Memory location addressed by the content of Register 0 receives the content of the Internal Data Memory location addressed by Data." In two operand instructions, the destination address also serves as the address of the first source. As an example of this, "ANL Data, #5" may be interpreted as "The content of the Internal Data Memory location addressed by Data receives the result of the operation when the content of the memory location specified by Data is and-ed with the immediate 5."

The 8051's instruction set is an enhancement of the instruction set familiar to MCS-48 users. It is enhanced to allow expansion of on-chip CPU peripherals and to optimize byte efficiency and execution speed. Efficient use of program memory results from an instruction set consisting of 49 single-byte, 45 two-byte and 17 three-byte instructions. Most arithmetic, logical and branching operations can be performed using an instruction that appends either a short address or a long address. For example, Register Addressing allows a two-byte equivalent of the three byte Direct Addressing instructions. Also, short branches are more code efficient than long branches. 64 instructions execute in twelve oscillator periods, 45 instructions execute in twenty-four oscillator periods, and multiply and divide take only forty-eight oscillator periods. The number of bytes in each instruction and the number of oscillator periods required for execution are listed in Table 4-1.

4.1 ORGANIZATION OF THE INSTRUCTION SET

Instructions are described here in four functional groups:

- Data Transfer
- Arithmetic
- Logic
- Control Transfer

The Data Transfer, Arithmetic and Logic groups mentioned in the preceding list are further subdivided into an array of codes that specify whether the operation is to act upon immediate, RB register, accumulator, SFR or memory locations; whether bits, nibbles, bytes or double-bytes are to be processed; and what addressing methods are to be employed.

4.1.1 Data Transfer

Data transfer operations are divided into three classes:

- General Purpose
- Accumulator-Specific
- Address-Object

None affect the flag settings except a POP or MOV into the PSW.

General-Purpose Transfers

Three general-purpose data transfer operations are provided. These may be applied to most operands, though there are specific exceptions.

- 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 element currently addressed by SP.
- POP transfers a byte operand from the stack element addressed by the SP register to the destination operand and then decrements SP.

Accumulator-Specific Transfers

Four accumulator-specific transfer operations are provided:

- 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 register A.
- MOVX performs a byte move between the External Data Memory and the A register. The external address can be specified by the DPTR register (16-bit) or the R1 or R0 register (8-bit).
- MOVC performs the move of a byte from the Program Memory to register A as follows. The operand in the A register 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 A. MOVC is used for table-look-up byte translation and for accessing operands from code-in-line tables.

Address-Object Transfer

- MOV DPTR,#data loads 16-bits of immediate data into a pair of destination registers, DPH and DPL (DHP from low-order address, DPL from high-order address).

4.1.2 Logic

The 8051 performs the basic logic operations on both bit and byte operands.

Single-Operand Operations

Seven single-operand logical operations are provided:

- CLR is used to set either the A register, the C register, or any Direct Addressed bit to zero (0).

- SETB sets either the C register or any Direct Addressed bit to one (1).
- CPL either forms the one's complement of the operand in the A register and returns the result to the A register without affecting flags or forms the one's complement of the C register or any Direct Addressed bit.
- RL, RLC, RR, RRC, SWAP. Five rotate operations can be performed on the A register; 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 C flag becomes equal to the last bit rotated out. SWAP rotates the A register left four places to exchange bits 3 through 0 with bits 7 through 4.

Two-Operand Operations

Three two-operand logical operations are provided:

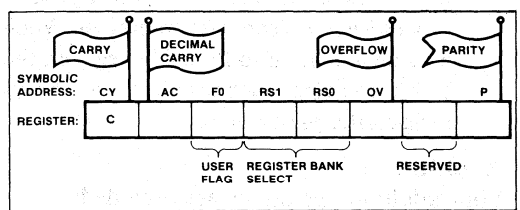
- ANL peforms the bitwise logical conjunction of two source operands (for both bit and byte operands) and returns the result to the location of the first operand.
- ORL performs the bitwise logical inclusive disjunction of two source operands (for both bit and byte operands) and returns the result of the location of the first operand.
- XRL performs the bitwise logical exclusive disjunction of the two source operands (byte operands) and returns the result to the location of the first operand.

4.1.3 Arithmetic

The 8051 provides the four basic mathematical operations. Only 8-bit operations using unsigned arithmetic are supported directly. The overflow flag permits the addition and subtraction operation to serve for both unsigned and signed binary integers. A correction operation is also provided to allow arithmetic to be performed directly on packed decimal (BCD) representations.

Flag Register Settings

Three one-bit flag registers are set or cleared by arithmetic operations to reflect certain properties of the result of the operation. These flags are not affected by the increment and decrement instructions. A fourth flag (P) denotes the parity of the eight accumulator bits. These flag registers are located in the Program Status Word (PSW) register. Their bit assignment are shown below. A list of the instructions that affect these flags is provided in the "8051 Instruction Set Summary" in Table 4-1.



Function	Flag	Bit Location
Carry Flag (also the C register)	CY:	PSW.7:
Auxiliary-Carry Flag	AC:	PSW.6:
Overflow Flag	OV:	PSW.2:
Parity Flag	P:	PSW.0:
User Flag 0	F0:	PSW.5:
reserved	---:	PSW.1:
Register Select MSb	RS1:	PSW.4:
Register Select LSb	RS0:	PSW.3:

Unless otherwise stated, the instructions obey these rules:

- CY is set if the operation result in a carry out of (during addition) or a borrow into (during subtraction) the high-order bit of the result; otherwise CY is cleared.
- AC is set if the operation results in a carry out of the low-order four bits of the result (during addition) or a borrow from the high-order bits into the low-order 4 bits (during subtraction); otherwise AC is cleared.
- OV is set if the operation results in a carry into the high-order bit of the result but not a carry out of the high-order bit, or vice versa; otherwise OV is cleared. OV is of use to two's-complement arithmetic, since it becomes set when the signed result cannot be represented in 8 bits.
- P is set if the module 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.

Addition

Four addition operations are provided:

- INC (increment) performs an addition of the source operand and one (1) and returns the result to the operand.
- ADD performs an addition between the A register and the second source operand and returns the result to the A register.
- ADDC (add with Carry) performs an addition between the A register and the second source operand; adds one (1) if the C flag is found previously set and returns the result to register A.
- DA (decimal-add-adjust for BCD addition) performs a correction to the sum which resulted from the binary addition of two two-digit decimal operands. The packed decimal sum formed by DA is returned to A. The carry flag is set if the BCD result is greater than 99; or else it is cleared.

Subtraction

Two subtraction operations are provided:

- SUBB (subtract with borrow) performs a subtraction of the second source operand from the first operand (the accumulator), subtracts one (1) if the C flag is found previously set and returns the result to the A register.
- DEC (decrement) performs a subtraction of one (1) from the source operand and returns the results to the operand.

Multiplication

- MUL performs an unsigned multiplication of the A register by the B register, returning a double-byte result. Register 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. C is cleared. AC remains unaltered.

Division

- DIV performs an unsigned division of the A register by the B register and returns the integer quotient to register 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. C is cleared. AC remains unaltered.

4.1.4 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. The three transfer operations are described below.

- ACALL and LCALL push the address of the next instruction onto the stack (PCL to low-order address, PCH to high-order address) and then transfer control to the target address. Absolute Call is a 2-byte instruction used when the target address is in the current 2K page. Long Call is a 3-byte instruction that addresses the full 64K program space. In ACALL, immediate data (i.e. and 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). The PC-relative short jump facilitates relocatable code.
- JMP @ A+DPTR performs a jump relative to the DPTR register. The operand in the A register 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. This indirect jump is also useful for implementing N-way branches.

Conditional Jumps

In the control transfer group, the 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 peforms 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.

Table 4-1. 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	O		
ADDC	X	X	X	CPL C	X		
SUBB	X	X	X	ANL C,bit	X		
MUL	O	X		ANL C,/bit	X		
DIV	O	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.

data — 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

addr16 — 16-bit destination address. Used by LCALL & LJMP. A branch can be anywhere within the 64K-byte Program Memory address space.

addr11 — 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 8048/8049.

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

ARITHMETIC OPERATIONS Cont.

Mnemonic		Description	Byte	Oscillator Period
SUBB	A,direct	Subtract direct byte from Acc with borrow	2	12
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

Table 4-1. Instruction Set Summary (continued)

ARITHMETIC OPERATIONS Cont.

Mnemonic		Description	Byte	Oscillator Period
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

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
XRL	A,@Ri	Exclusive-OR indirect RAM to Accumulator	1	12
XRL	A,#data	Exclusive-OR immediate data to Accumulator	2	12

LOGICAL OPERATIONS Cont.

Mnemonic		Description	Byte	Oscillator Period
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
RLC	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 4-1. 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
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

DATA TRANSFER Cont.

Mnemonic		Description	Byte	Oscillator Period
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 4-1. 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

PROGRAMING BRANCHING

Mnemonic		Description	Byte	Oscillator Period
ACALL	addr11	Absolute Subroutine Call	2	24
LCALL	addr16	Long Subroutine Call	3	24
RET		Return for Subroutine	1	24
RETI		Return for 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

PROGRAM BRANCHING Cont.

Mnemonic		Description	Byte	Oscillator Period
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	Comare 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

- CJNE compares the first operand to the second operand and performs a jump if they are not equal. C is set if the first operand is less than the second operand; else it is cleared. Comparisons can be made between the A register and Direct Addressable bytes in the Internal Data Memory or between an immediate value and either the A register, an RB register in the selected Register Bank, or a Register-Indirect addressed byte of the Internal Data RAM.
- DJNZ decrements the source operand and returns the result to the operand. A jump is performed if the result is not zero. The DJNZ instruction makes a RAM location efficient for use as a program loop counter by allowing the programmer to decrement and test the counter in a single instruction. 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.

Interrupts

Program execution control may be transferred by means of internal and external interrupts. All interrupts perform a transfer by pushing the Program Counter onto the stack and then branching to programs located at absolute locations 3, 11, 19, 27 and 35 in the Program Memory. The programmer must push all registers that will be altered by his interrupt service program onto the stack to avoid corruption. Only one interrupt transfer operation is necessary:

- RETI transfers control in a manner identical to RET. In addition, RETI reenables interrupts for the current priority level.

See section 2.3.1 for futher details on the operation and control of the interrupt system.

4.2 Instruction Definitions

The rest of this chapter defines all the instructions and operations which the MCS-51 CPU can perform. There is a separate section for each of the 51 basic operations, ordered alphabetically according to the operation mnemonic.

When an operation may apply to more than one data type (generally bit and byte data), the MCS-51 assembly language uses the same mnemonic for each, reducing the number of mnemonics the programmer must remember. The assembler determines which instruction is appropriate from the operands specified. Thus, the mnemonic "CLR" can operate on the eight-bit accumulator ("CLR A"), or on one-bit variables ("CLR F0"). The mnemonics ANL, ORL, CPL, and MOV can relate to more than one data type as well. These operations present each data type in a separate section.

Each section then describes the action taken by the operation, the flags and registers affected, and shows a short example of how an instruction might be used in a program. Next comes the number of bytes and machine cycles required, the corresponding binary machine-language encoding, and a symbolic description or restatement of the function implemented.

Note: Only the carry, auxiliary-carry, and overflow flags are discussed in these instruction descriptions. Since the parity bit (PSW.0) is recomputed *after every instruction cycle* any instruction that alters the accumulator — either inherently or as a special function

register — could affect the parity flag. 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 the generalized bit-manipulation instructions.

Nineteen operations allow more than one addressing mode for the source and/or destination operand. The headings for these sections show the instruction format with such operands enclosed in angle brackets (for example, MOV <dest-byte>, <src-byte>). The operation description tells what modes (or combinations of modes) are allowed, and gives the assembly language notation, byte and cycle counts, encoding format, and a symbolic description for each.

The information in this chapter is directed towards defining the capabilities of the MCS-51 architecture and hardware. For details on the assembly language or ASM51 capabilities refer to the *MCS-51 Macro Assembler User's Guide,* publication number 9800937.

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, opcode 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

$(A) \leftarrow (A) + ((R_i))$

ADD A,#data

Bytes: 2

Cycles: 1

Encoding:

0	0	1	0	0	1	0	0	immediate data

Operation: ADD

$(A) \leftarrow (A) + \#data$

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

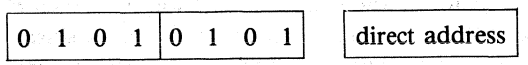
Operation: ANL

(A) ← (A) ∧ (Rn)

ANL A,direct

Bytes: 2
Cycles: 1

Encoding:

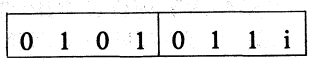


Operation: ANL
(A) ← (A) ∧ (direct)

ANL A,@Ri

Bytes: 1
Cycles: 1

Encoding:

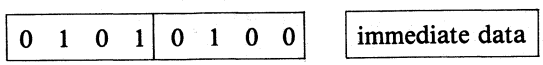


Operation: ANL
(A) ← (A) ∧ ((Ri))

ANL A,#data

Bytes: 2
Cycles: 1

Encoding:

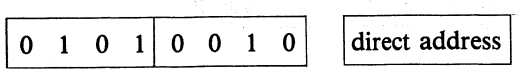


Operation: ANL
(A) ← (A) ∧ #data

ANL direct,A

Bytes: 2
Cycles: 1

Encoding:

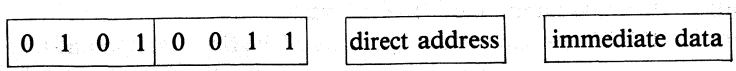


Operation: ANL
(direct) ← (direct) ∧ (A)

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:

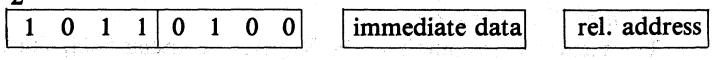
1 0 1 1	0 1 0 1	direct address	rel. address

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

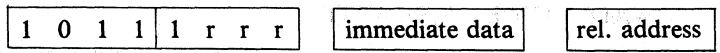
Bytes: 3
Cycles: 2
Encoding:



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

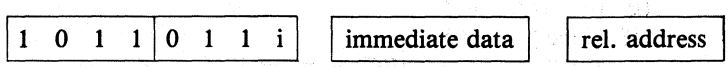
Bytes: 3
Cycles: 2
Encoding:



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:



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,

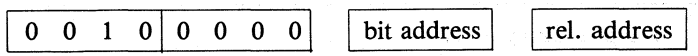
```
JB    P1.2,LABEL1
JB    ACC.2,LABEL2
```

will cause program execution to branch to the instruction at label LABEL2.

Bytes: 3

Cycles: 2

Encoding:



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,@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:

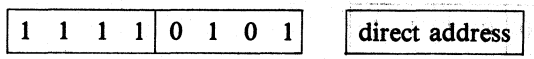
0	1	1	1	1	r	r	r	immediate data

Operation: MOV
(Rn) ← #data

MOV direct,A

Bytes: 2
Cycles: 1

Encoding:

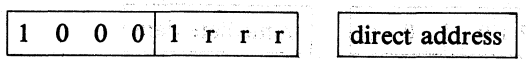


Operation: MOV
(direct)←(A)

MOV direct,Rn

Bytes: 2
Cycles: 2

Encoding:

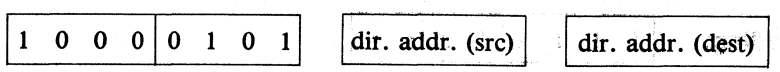


Operation: MOV
(direct)←(Rn)

MOV direct,direct

Bytes: 3
Cycles: 2

Encoding:

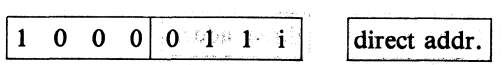


Operation: MOV
(direct)←(direct)

MOV direct,@Ri

Bytes: 2
Cycles: 2

Encoding:

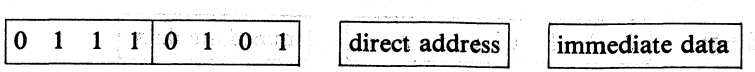


Operation: MOV
(direct)←((Ri))

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.

This is the only instruction which moves 16 bits of data at once.

Example: 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. P2 retains the high-order bits; any data previously on P2 is lost. 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 A

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
$(A)_{15-8} \leftarrow (A) \times (B)$
$(B)_{7-0}$

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

$(PC) \leftarrow (PC) + 1$

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

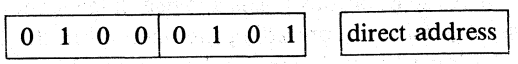
Operation: ORL

(A)←(A) ∨ (Rn)

ORL A,direct

Bytes: 2
Cycles: 1

Encoding:

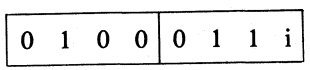


Operation: ORL
(A) ← (A) ∨ (direct)

ORL A,@Ri

Bytes: 1
Cycles: 1

Encoding:

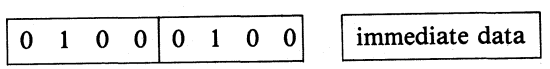


Operation: ORL
(A) ← (A) ∨ ((Ri))

ORL A,#data

Bytes: 2
Cycles: 1

Encoding:

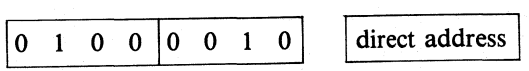


Operation: ORL
(A) ← (A) ∨ #data

ORL direct,A

Bytes: 2
Cycles: 1

Encoding:

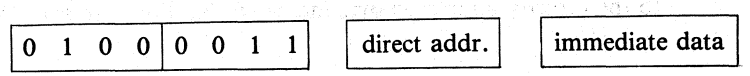


Operation: ORL
(direct) ← (direct) ∨ (A)

ORL direct,#data

Bytes: 3
Cycles: 2

Encoding:



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) ⇄ (Rn)

XCH A,direct

Bytes: 2

Cycles: 1

Encoding:

1	1	0	0	0	1	0	1

direct address

Operation: XCH

(A) ⇄ (direct)

XCH A,@Ri

Bytes: 1

Cycles: 1

Encoding:

1	1	0	0	0	1	1	i

Operation: XCH

(A) ⇄ ((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) ∀ ((Ri))

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

4.3 EXPANDED 8051 FAMILY

This section shows in very general terms some basic circuits for expanding the 8051 Family. As the product matures and Intel tests specific circuits, the User Manual will be updated. Also application notes will be published to help show actual, tested circuits designed by Intel personnel. The schematics included in this chapter should give the designer an insight into connecting external peripherals and memories to the 8051.

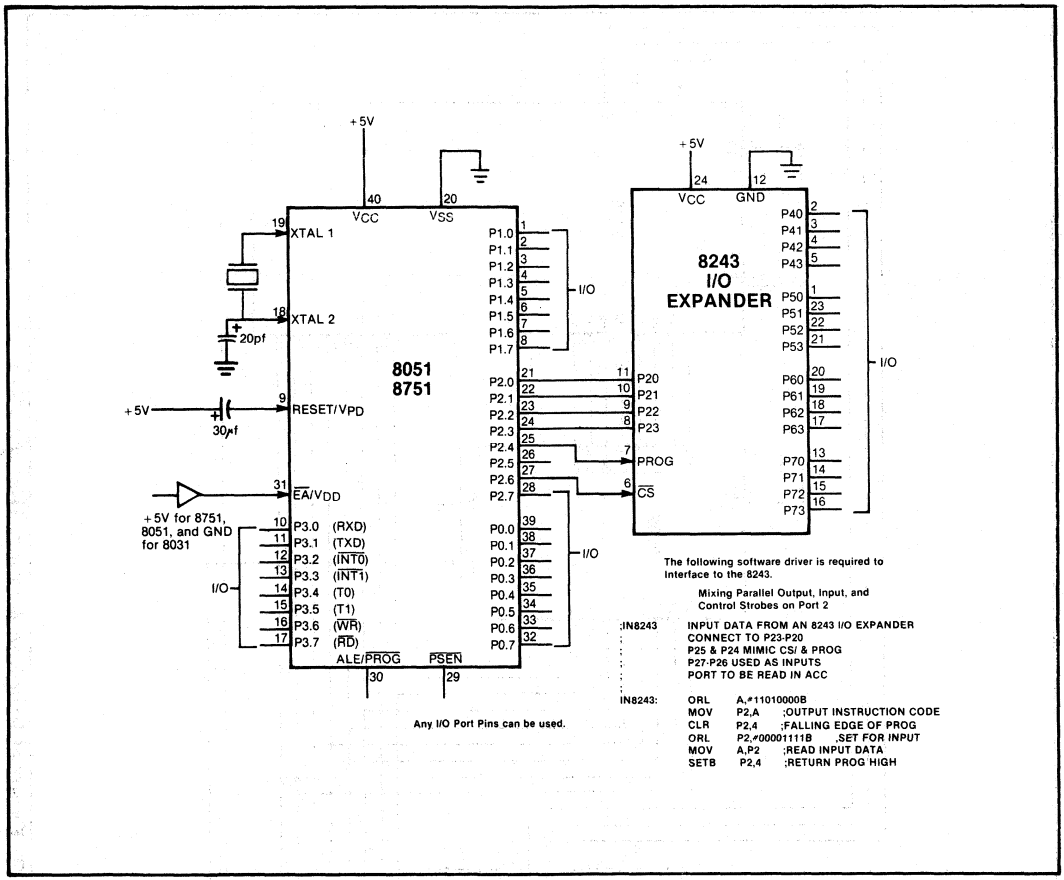


Figure 4-1. I/O Expansion Using an 8243

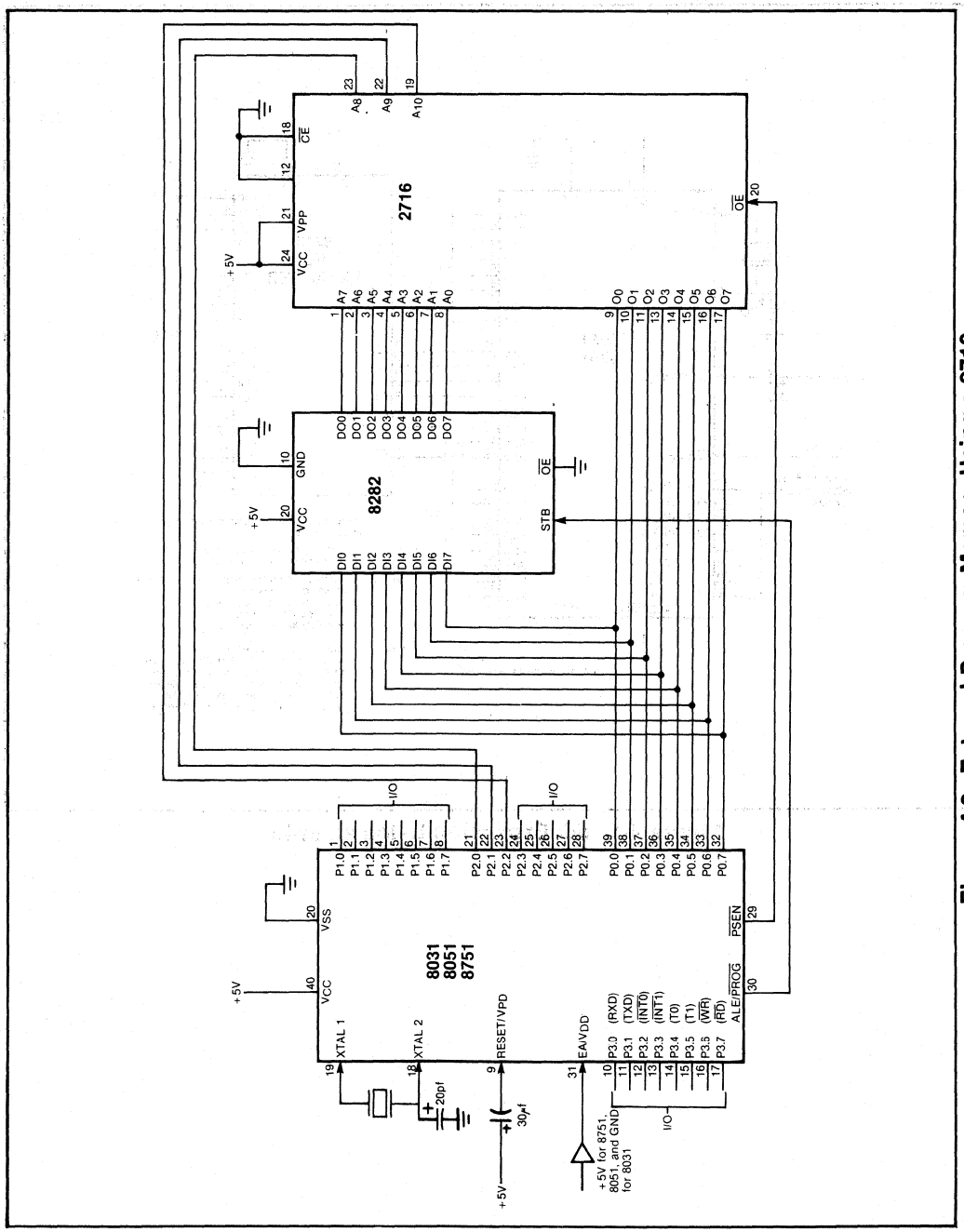


Figure 4-2. External Program Memory Using a 2716

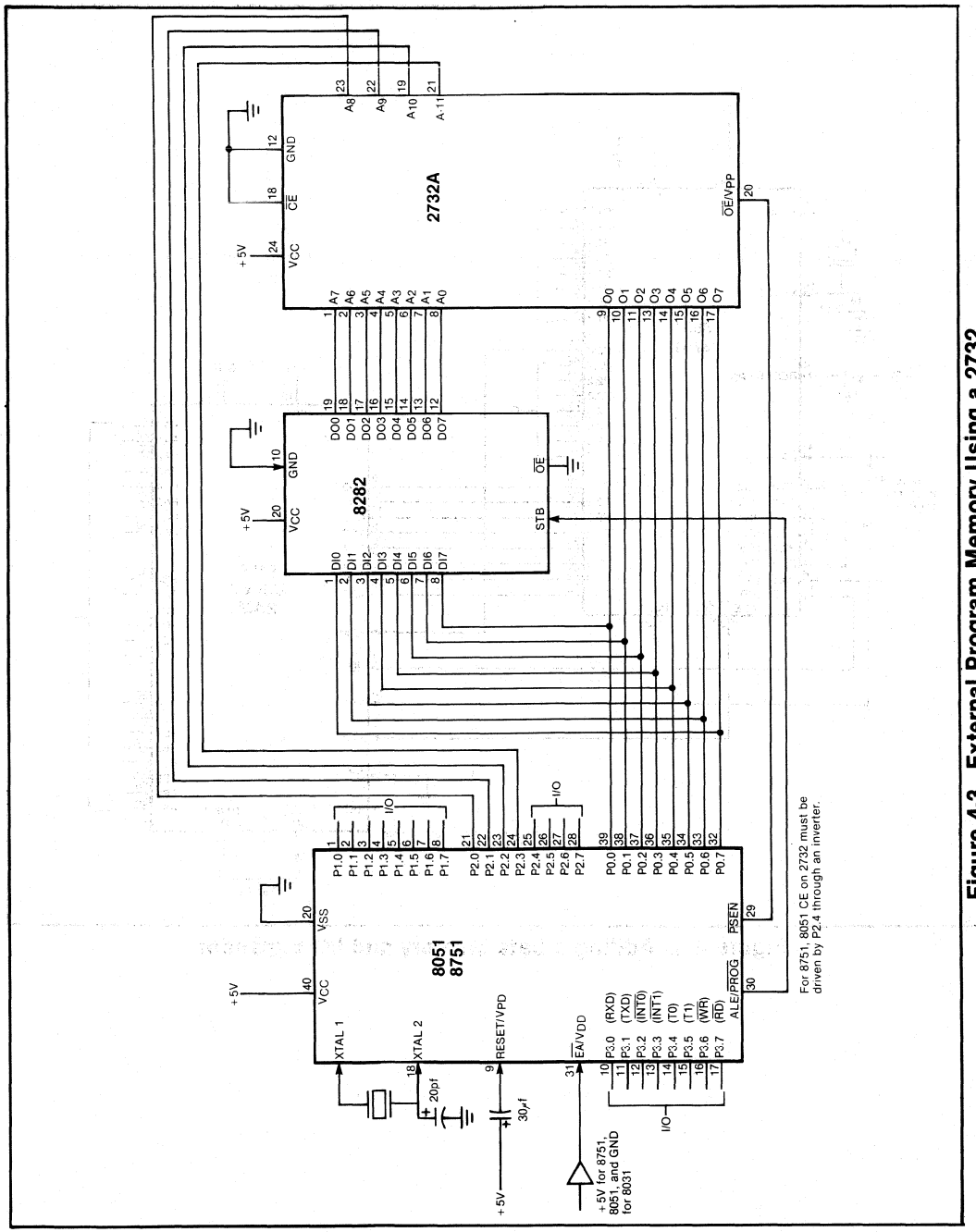


Figure 4-3. External Program Memory Using a 2732

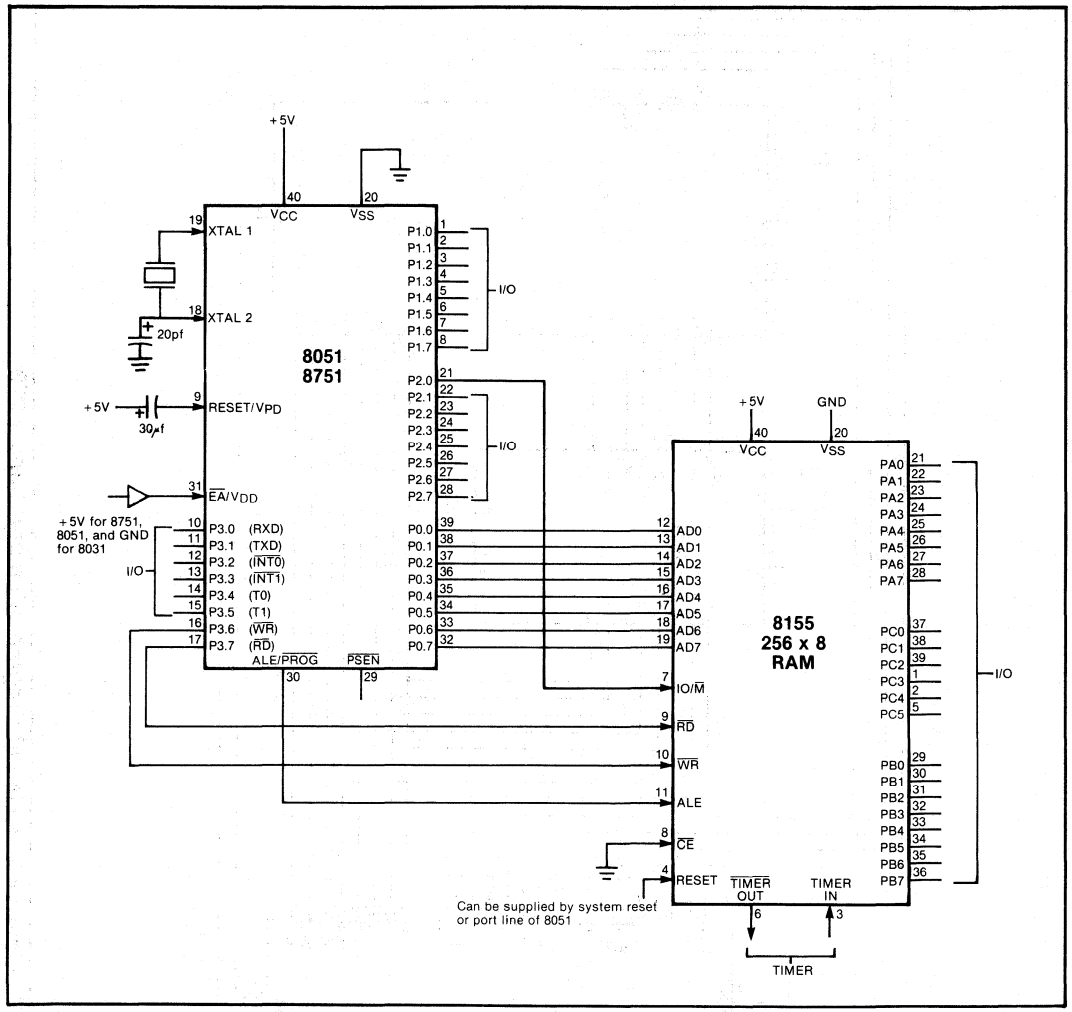


Figure 4-4. Adding a Data Memory and I/O Expander

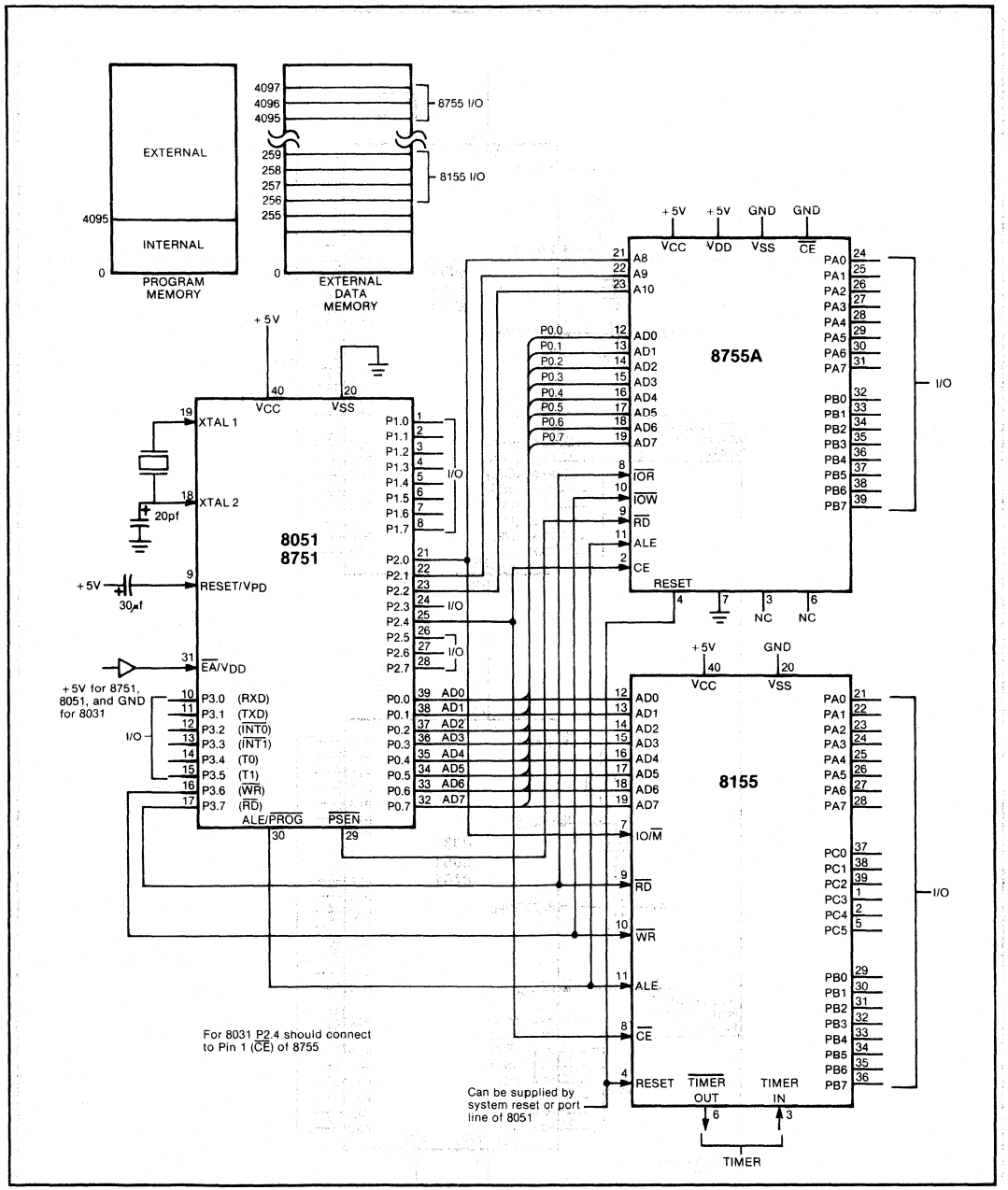


Figure 4-5. The Three-Chip System

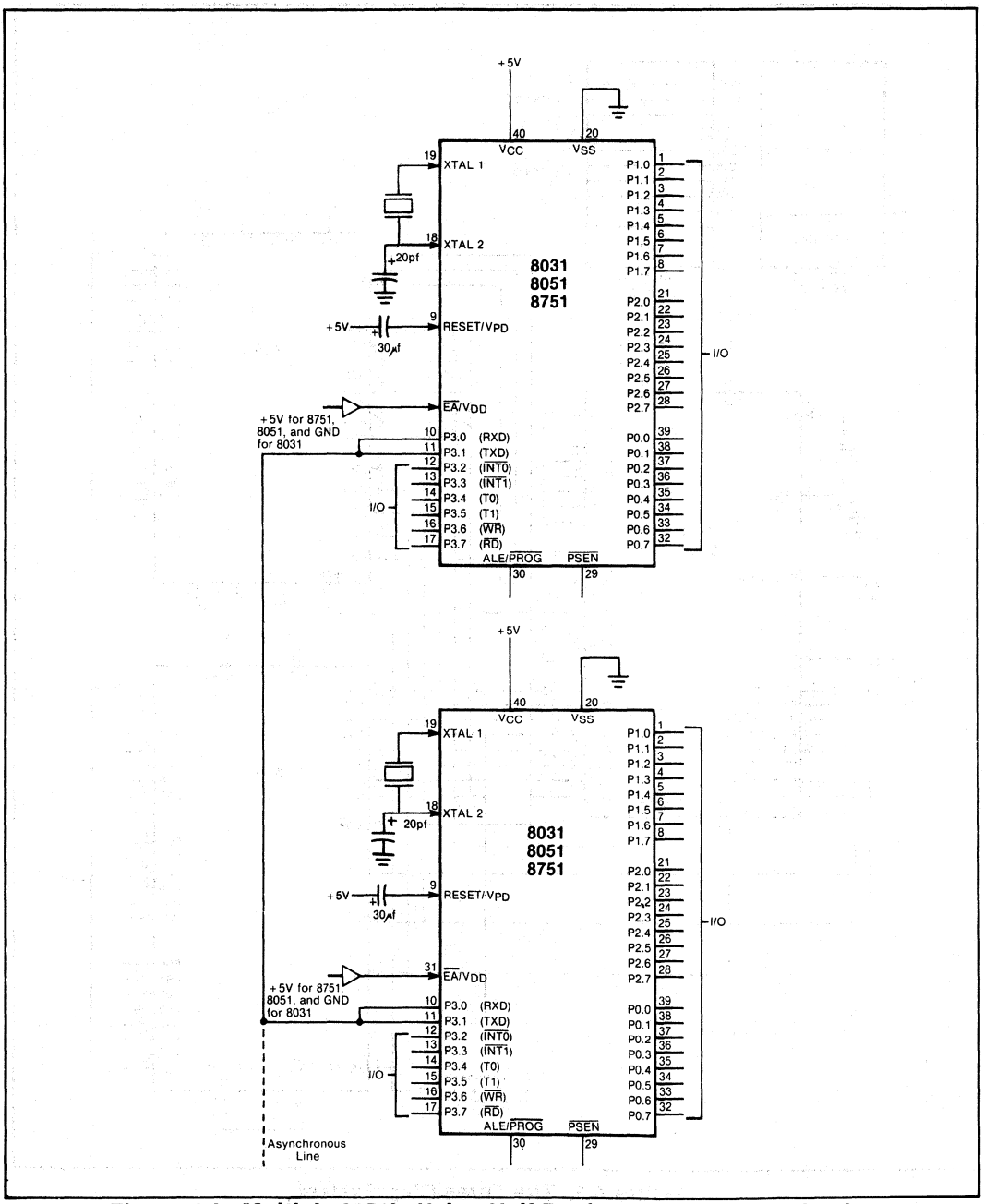


Figure 4-6. Multiple 8051's Using Half-Duplex Serial Communication

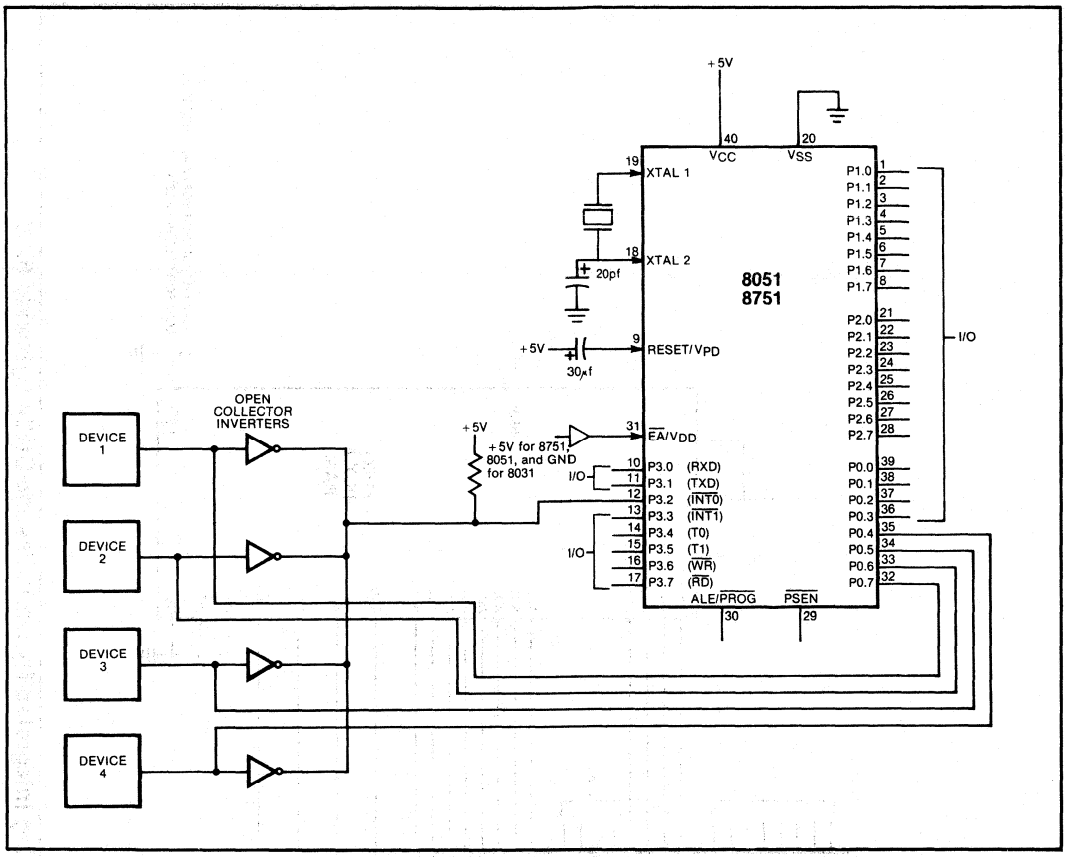


Figure 4-7. Multiple Interrupt Sources

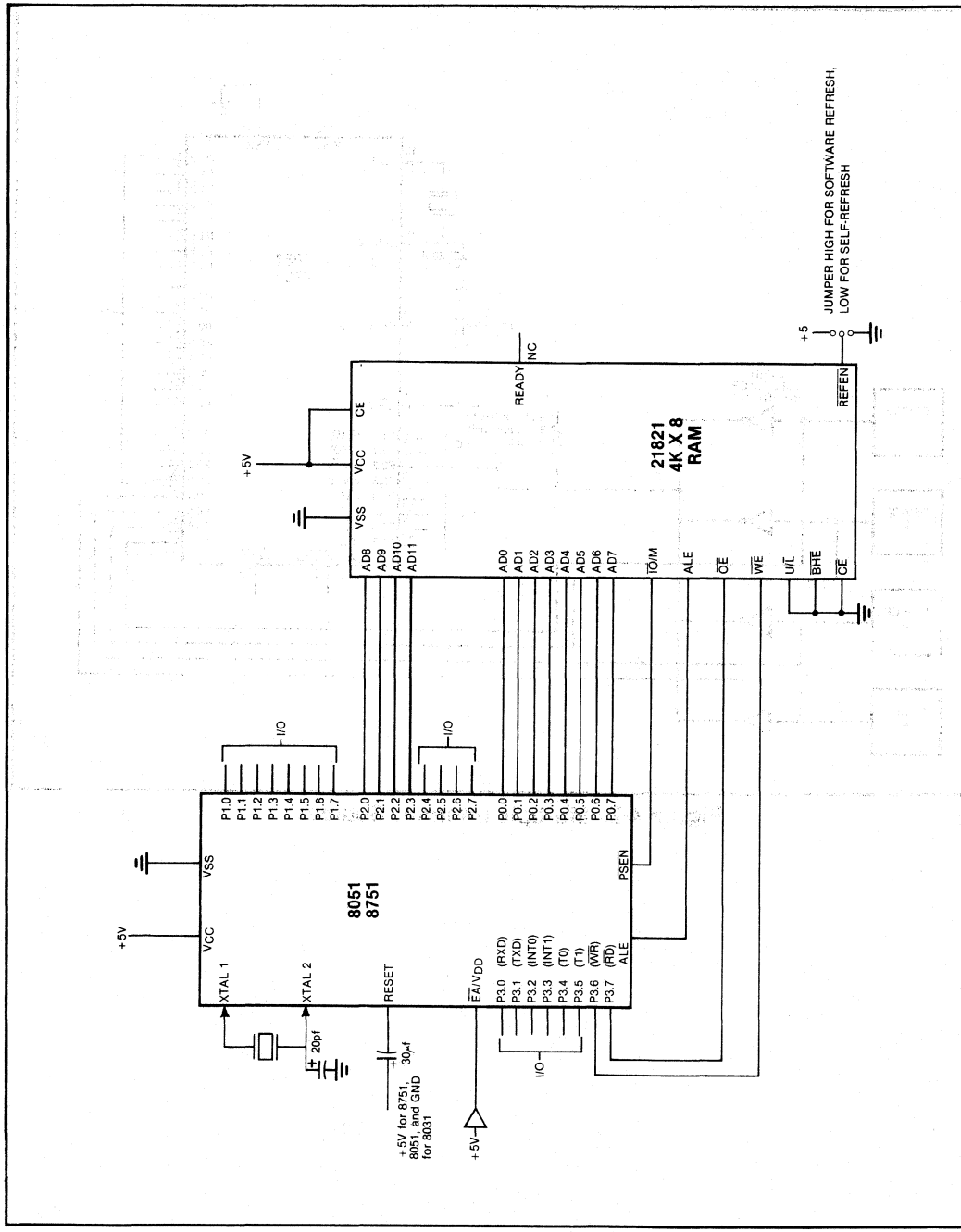


Figure 4-8. Interfacing Integrated Dynamic/Pseudo-Static Bytewide RAM

Software Routines

5

CHAPTER 5
SOFTWARE ROUTINES

Chapter 5 contains two sections:

- 8051 Programming Techniques
- Peripheral Interfacing Techniques

The first section has 8051 software examples for some common routines in controller applications. Some routines included 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.

5.1 8051 PROGRAMMING TECHNIQUES

5.1.1 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
```

5.1.2 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
```

5.1.3 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,

Entry Address = [BASE + (NDIMEN x INDEXI) + INDEXJ]

The subroutine MATRX1 can access an entry in any array with less than 255 elements (e.g., 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.

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)
```

5.1.4 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 5-1; 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.

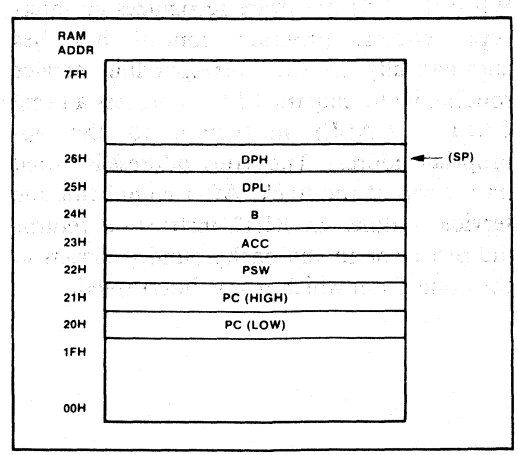


Figure 5-1. Stack contents during interrupt

5.1.5 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.

```
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 the 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
        ...     ......
```

5.1.6 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.

5.1.7 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 LOW-ORDER 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
;
```

5.1.8 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 accesses 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 parameter 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.

5.2 PERIPHERAL INTERFACING TECHNIQUES

5.2.1 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.

5.2.2 I/O Port Reconfiguration (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 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 a "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 BITS 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.
        RCC     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
```

5.2.3 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 5-2.

```
;
;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
```

5.2.4 Software Delay Timing

Many 8051 applications involve exact control over output timing. A software-generated output strobe, for instance, might have to be *exactly* 50 μsec. wide. The DJNZ operation can insert a one instruction software delay into a piece of code, adding a moderate time 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    $2,$
SETB    WR
```

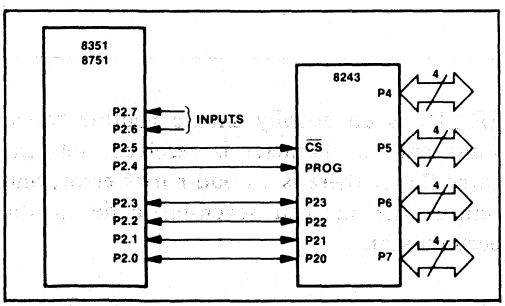


Figure 5-2 Connecting an 8051 with an 8243 I/O Expander

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.

5.2.5 Serial Port and Timer Mode 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,#11010010B
            MOV     TH1,# 13
            SETB    TR1
;           ...     ......
;
```

5.2.6 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
```

5.2.7 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
```

5.2.8 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  20 H
         MOV    CHAR,#0               ;REPLACE CONTROL CHARACTER WITH
                                      ;NULL CODE
PRINTC:                               ;PROCESS STANDARD PRINTING
;        ...    .....                 ;CHARACTER
         RET
;
```

5.2.9 Buffering Serial Port Output Characters

It is not always efficient to transmit characters through the serial port one-at-a-time. Most applications generate a short burst of characters all at once (English words or multi-digit numbers, for instance), with the bursts themselves occurring at longer intervals. Instead of waiting while the UART outputs each character, it would be more efficient if the background program could enter all the characters into a first-in first-out (FIFO) data structure, and continue about its business, letting an interrupt routine transmit each character as the serial port becomes available.

Assume there is a 16-byte output data buffer starting at 70H. QHEAD and QTAIL keep track of the head and tail portion of the buffer being used. The subroutine ENTERQ waits until there is space in the queue, then copies a character code from the accumulator to the queue.

```
QHEAD    DATA   6EH                    ;LAST BYTE ENTERED INTO QUEUE
QTAIL    DATA   6FH                    ;LAST BYTE READ FROM QUEUE.
BOTLIM   EQU    70H
TOPLIM   EQU    7FH
;
;        QUEUE IS EMPTY WHEN QHEAD = QTAIL AND
;        FULL WHEN QHEAD + 1 (WITHIN RANGE) = QTAIL.
         MOV    QHEAD,#TOPLIM
         MOV    QTAIL,#TOPLIM
;
;
ENTERQ:  MOV    R0,A                   ;SAVE ACC DATA
         MOV    A,QHEAD                ;LOAD HEAD POINTER
         INC    A                      ;PRE-INCREMENT POINTER
         CJNE   A,#TOPLIM+1,ENTQ__1
         MOV    A,#BOTLIM              ;RELOAD ON OVERFLOW
ENTQ__1: CJNE   A,QTAIL,ENTQ__2        ;TEST IF QUEUE FULL
         SJMP   ENTQ__1                ;LOOP UNTIL SPACE AVAILABLE
ENTQ__2: XCH    A,R0                   ;STORE POINTER AND RELOAD ACC
         MOV    @R0,A                  ;ENTER INTO QUEUE
         MOV    QHEAD,R0               ;UPDATE HEAD POINTER
         SETB   ES                     ;ENABLE SERIAL PORT INTERRUPTS
         RET
```

The interrupt routine DQUEUE is invoked when the transmitter is ready for another character. First it determines if any characters are available for transmission, indicated by QHEAD and QTAIL being not equal. If more data is available, it is written to the transmit buffer (SBUF) and the pointers are updated. If not, DQUEUE disables serial port interrupts and returns to the background program. ENTERQ will re-enable such interrupts as more data is available. (This example does not consider interrupt-driven serial input.)

```
        ORG     0023H
        PUSH    ACC                     ;SAVE CPU STATUS
        PUSH    PSW
        MOV     PSW,#30Q                ;SELECT BANK 3
DQUEUE: MOV     A,QTAIL
        CJNE    A,QHEAD,DQ_1            ;TEST IF QUEUE EMPTY
        CLR     ES                      ;IF SO, CLEAR ENABLE BIT AND RETURN
        SJMP    TI_RET
DQ_1:   CLR     TI                      ;ELSE ACKNOWLEDGE REQUEST
        INC     A                       ;COMPUTE NEXT BYTE'S ADDRESS
        CJNE    A,#TOPLIM+1,DQ_2
        MOV     A,#BOTLIM               :REVISE ACC IF POINTER OVERFLOWED
DQ_2:   MOV     R0,A                    ;LOAD INDEX REGISTER
        MOV     SBUF,@R0                ;RELOAD TRANSMITTER
        MOV     QTAIL,A                 ;SAVE LAST POINTER USED.
TI_RET: POP     PSW                     ;RESTORE STATUS AND RETURN
        POP     A
        RETI
```

5.2.10 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 application 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    TH1                 ;RESTART TIMER
;       ...     .....
```

5.2.11 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
```

Applications

6

AN INTRODUCTION TO THE INTEL MCS-51™ SINGLE-CHIP MICROCOMPUTER FAMILY

Contents

1. **INTRODUCTION** 6-2

   Family Overview 6-2
   Microcomputer Background Concepts 6-3

2. **ARCHITECTURE AND ORGANIZATION** 6-5

   Central Processing Unit 6-6
   Memory Spaces 6-9
   Input/Output Ports 6-10
   Special Peripheral Functions 6-11

3. **INSTRUCTION SET AND ADDRESSING MODES** 6-15

   Data Addressing Modes 6-15
   Addressing Mode Combinations 6-18
   Advantages of Symbolic Addressing 6-18
   Arithmetic Instruction Usage 6-19
   Multiplication and Division 6-20
   Logical Byte Operations 6-20
   Program Control 6-21
   Operate-and-Branch Instructions 6-22
   Stack Operations 6-22
   Table Look-Up Instructions 6-23

4. **BOOLEAN PROCESSING INSTRUCTIONS** 6-25

   Direct Bit Addressing 6-25
   Bit Manipulation Instructions 6-25
   Combinatorial Logic Equations 6-26

5. **ON-CHIP PERIPHERAL FUNCTIONS** 6-28

   I/O Ports 6-28
   Serial Port and Timer 6-29

6. **SUMMARY** 6-30

AFN-01502A

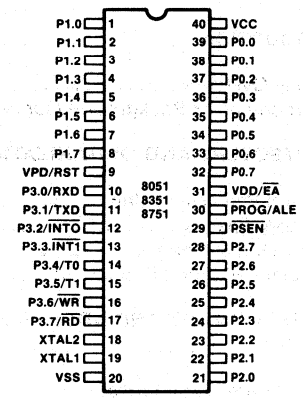


Figure 1a. 8051 Microcomputer Pinout Diagram

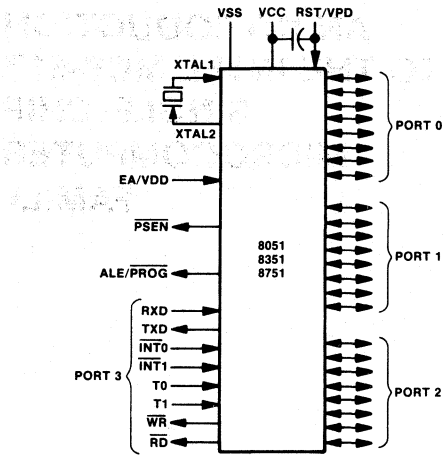


Figure 1b. 8051 Microcomputer Logic Symbol

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 *u*sec.
- Integrated Boolean Processor for control applications.
- Upwardly compatible with existing 8048 software.

AFN-01502A

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

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.

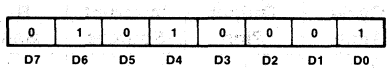


Figure 2. Representation of Bits Within an Eight-Bit "Byte" (Value shown = 01010001 Binary = 81 decimal).

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

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

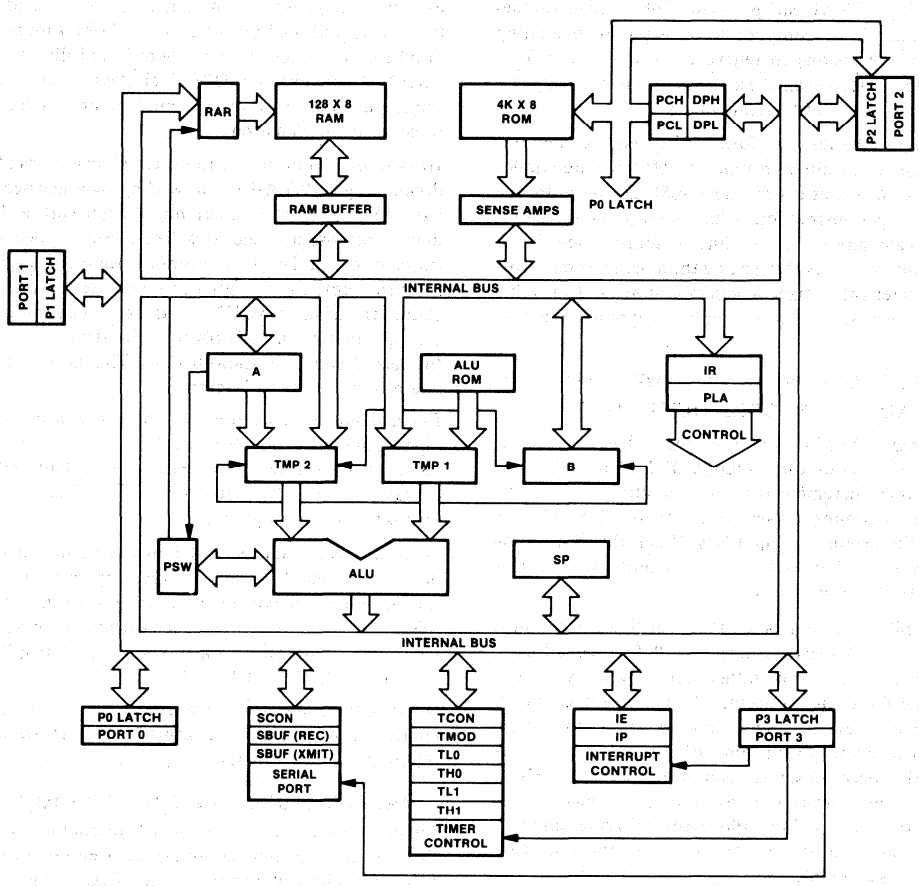


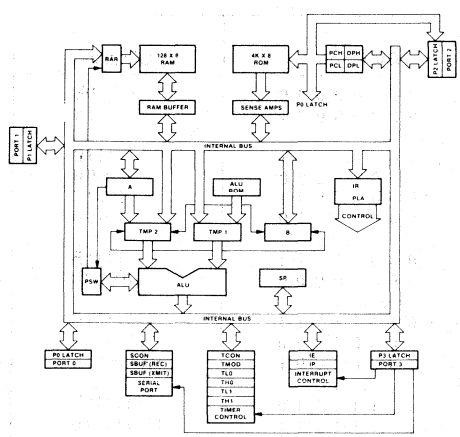
Figure 3. Block Diagram of 8051 Internal Structure

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

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.

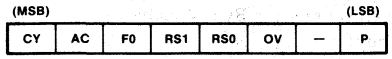
Observant readers will also notice that all of the 8048's page-oriented instructions (conditional jumps, JMPP, MOVP, MOVP3) have been replaced with corresponding but non-paged instructions. The 8051 instruction set is entirely *non*-page-oriented. The MCS-48™ "MOVP" instruction replacement and all conditional 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



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

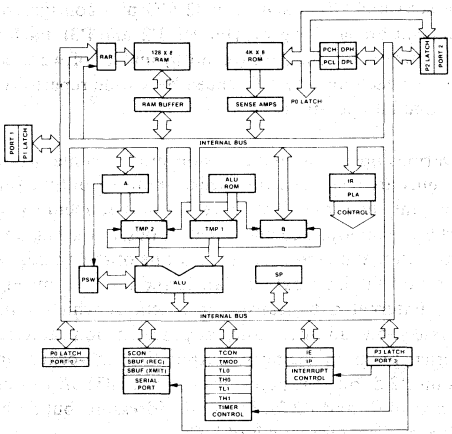
A special eight-bit register ("B") serves in the execution of the multiply and divide instructions. This register is used in conjunction with the accumulator as the second input operand and to return eight-bits of the result.

The MCS-51 family processors include a hardware stack within internal RAM, useful for subroutine linkage,

AFN-01502A

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

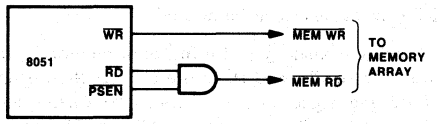


Figure 5. Combining External Program and Data Memory Arrays

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.

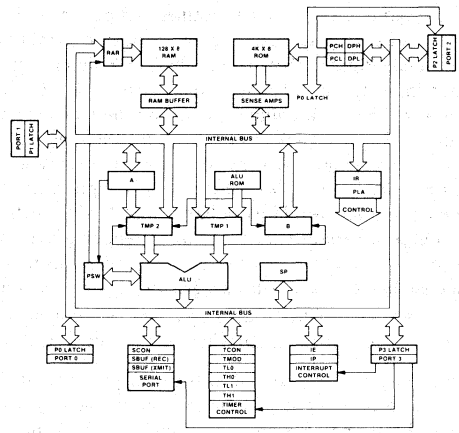
Thus, the MCS-51™ architecture supports several distinct "physical" address spaces, functionally separated at the hardware level by different addressing mechanisms, read and write control signals, or both:

- 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.

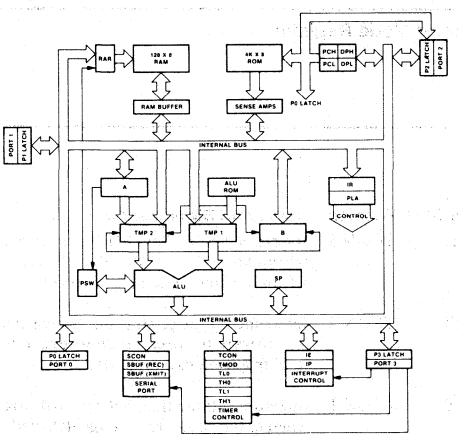
Even within a single port, I/O functions may be combined in many ways: input and output may be performed using different pins at the same time, or the same pins at different times; in parallel in some cases, and in serial in others; as test pins, or (in the case of Port 3) as additional special functions.

AFN-01502A

(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.

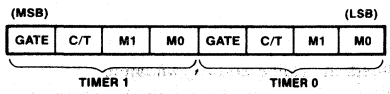
Serial Port Interface

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

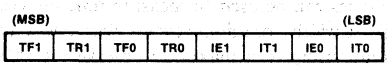


GATE — Gating control. When set, Timer/counter "x" is enabled only while "INTx" pin is high and "TRx" control bit is set. When cleared, timer/counter is enabled whenever "TRx" control bit is set.

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.

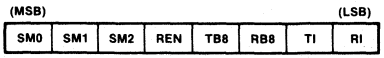
Figure 7. TMOD—Timer/Counter Mode Register



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



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

(MSB)							(LSB)
EA	—	—	ES	ET1	EX1	ET0	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	(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

(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.

Figure 11. IP—Interrupt Priority Control Register

AFN-01502A

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

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)
```

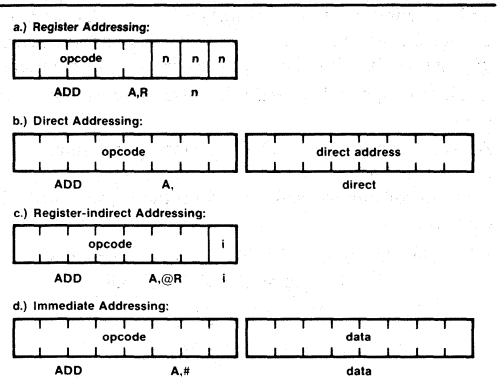


Figure 12. Data Addressing Machine Code Formats

AFN-01502A

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

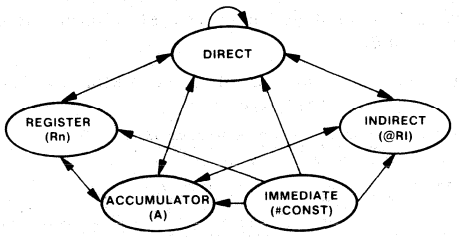


Figure 13. Road map for moving data bytes

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 is 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.

AFN-01502A

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
```

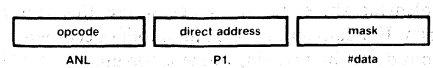


Figure 14. Instruction Pattern for Logical Operation Special Addressing Modes

AFN-01502A

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).

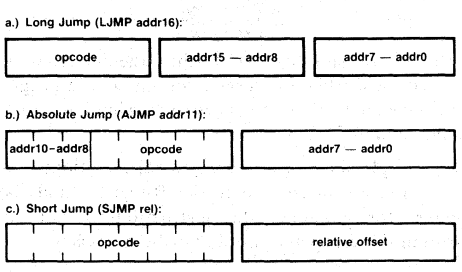


Figure 15. Jump Instruction Machine Code Formats

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

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

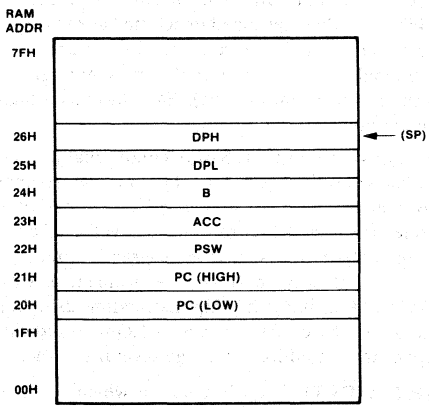


Figure 16. Stack contents during interrupt

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.

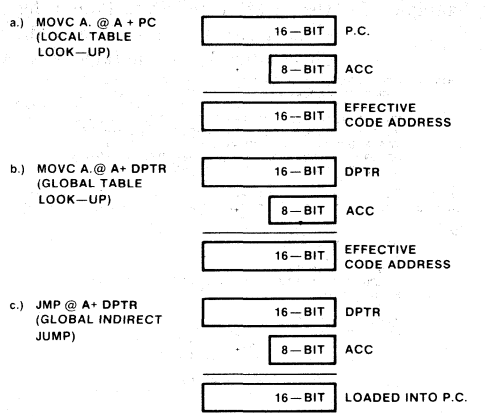


Figure 17. Operation of MOVC instructions

AFN-01502A

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

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.

AFN-01502A

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 18H	Bank 3							
17H 10H	Bank 2							
0FH 08H	Bank 1							
07H 00H	Bank 0							

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

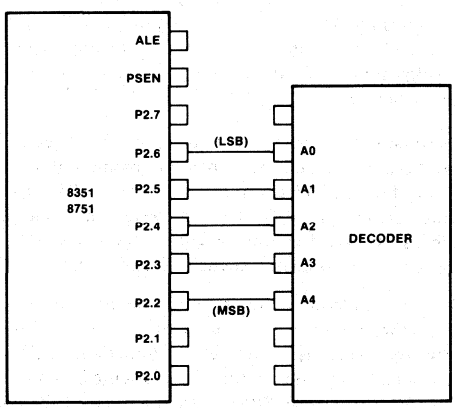


Figure 19. "Mismatch" Between I/O port and Decoder

Example 26—Re-ordering I/O Port Configuration

```
OUT_PZ: 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

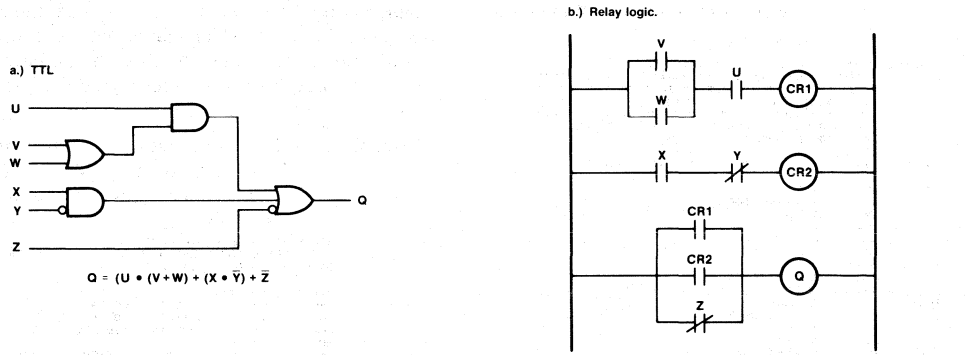


Figure 20. Implementations of Boolean functions

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.

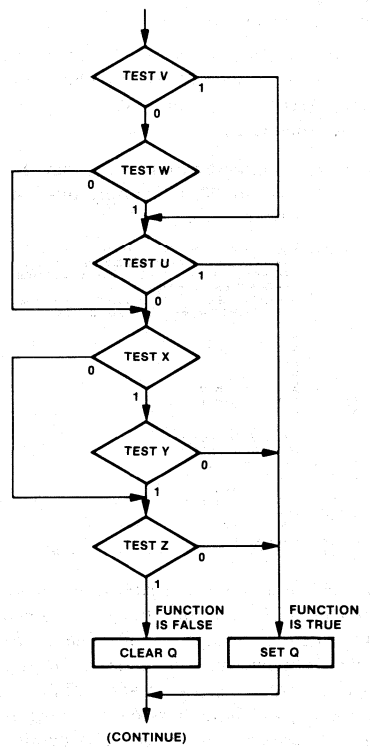


Figure 21. Flow chart for tree-branching logic implementation

AFN-01502A

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

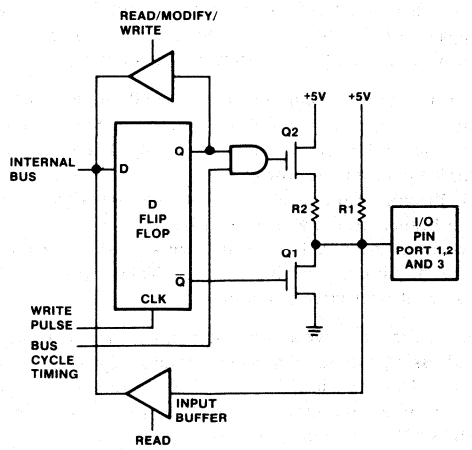


Figure 22. Pseudo-bidirectional I/O port circuitry

AFN-01502A

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
        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
        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
```

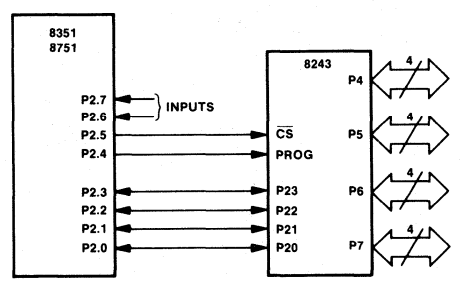


Figure 23. Connecting an 8051 with an 8243 I/O Expander

AFN-01502A

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,#11010010B
        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** 6-32

2. **BOOLEAN PROCESSOR** 6-32

   Processing Elements 6-33
   Direct Bit Addressing 6-34
   Instruction Set 6-39
   Simple Instruction Combinations 6-40

3. **BOOLEAN PROCESSOR APPLICATIONS** 6-41
   Design Example #1 6-42
   Design Example #2 6-45
   Design Example #3 6-46
   Design Example #4 6-49
   Design Example #5 6-54
   Additional Functions and Uses 6-59

4. **SUMMARY** 6-60

   APPENDIX A 6-61

01489A

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.

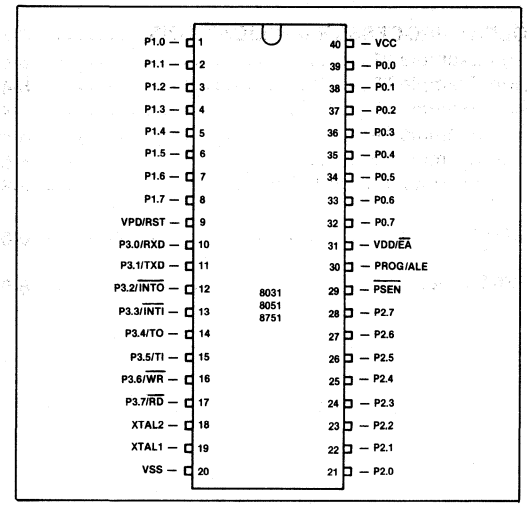


Figure 1. 8051 Family Pinout Diagram.

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

- 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.

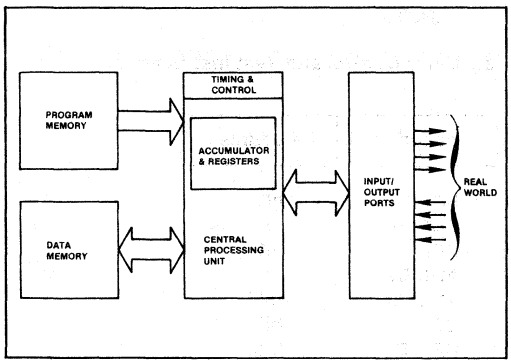


Figure 2. Block Diagram for Abstract Digital Computer.

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

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.

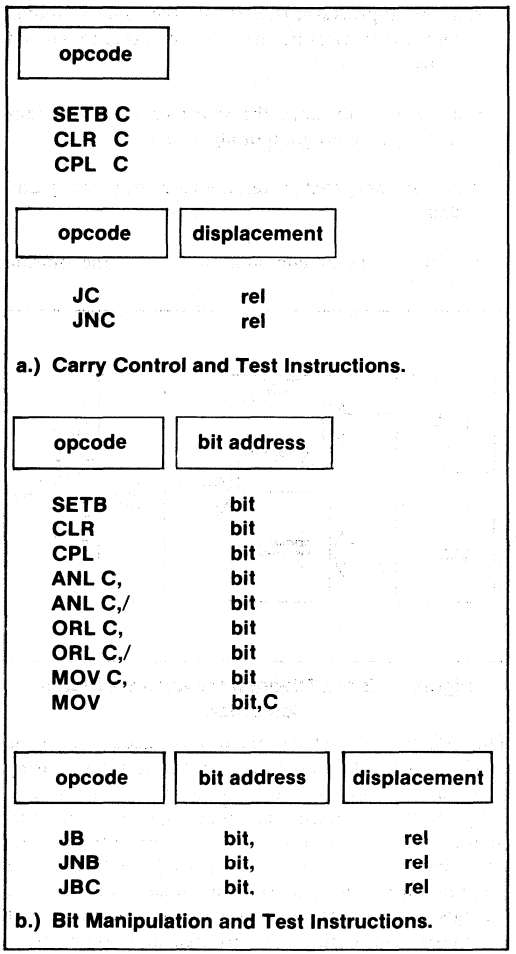


Figure 3. Bit Addressing Instruction Formats.

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).

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

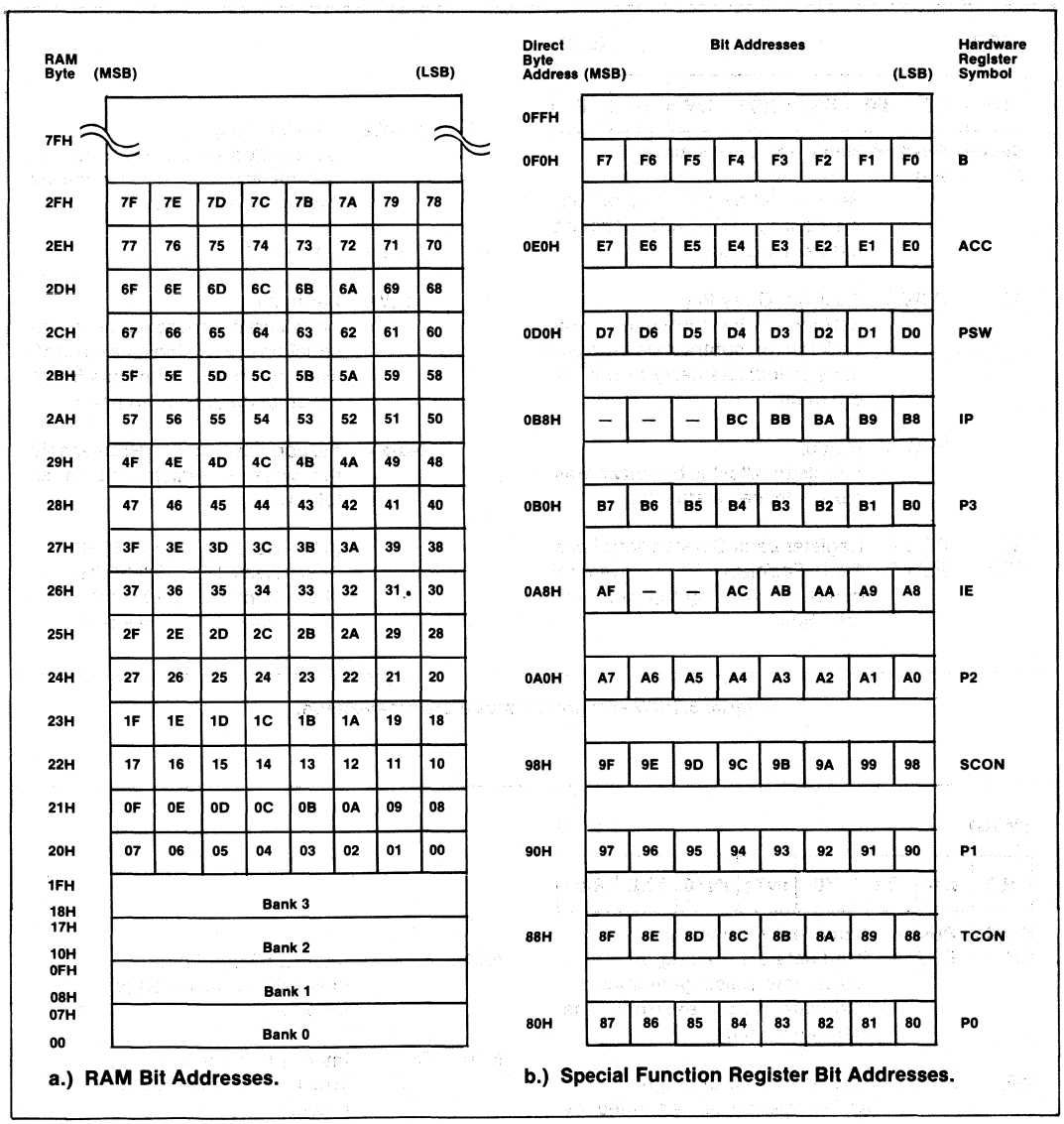


Figure 4. Bit Address Maps.

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

(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

(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

(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

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 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.)

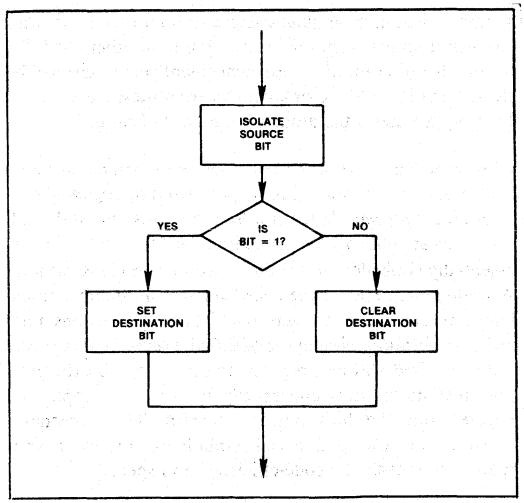


Figure 8. Bit Transfer Instruction Operation.

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

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

01489A

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

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

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.

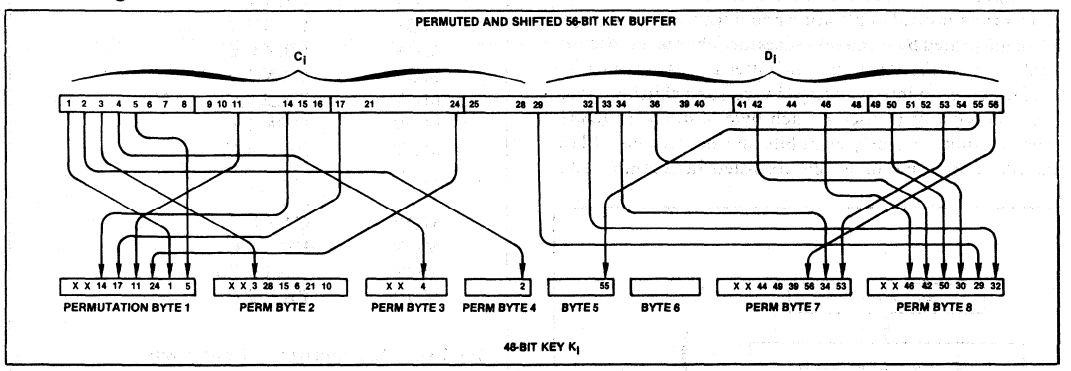


Figure 9. DES Key Schedule Transformation.

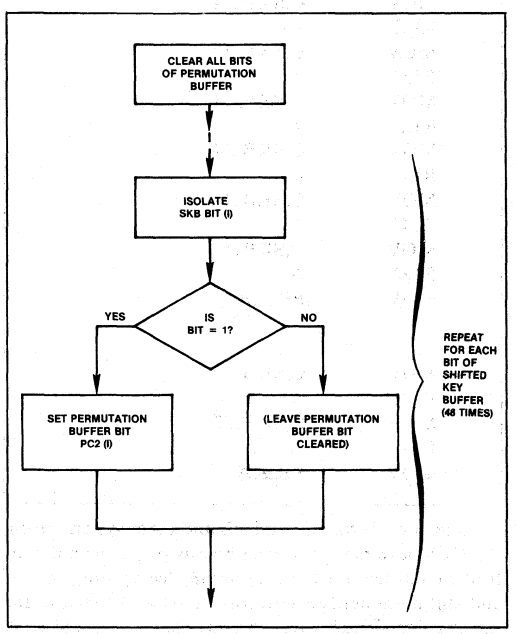


Figure 10.a. Flowchart for Key permutation attempted with a byte processor.

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

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.

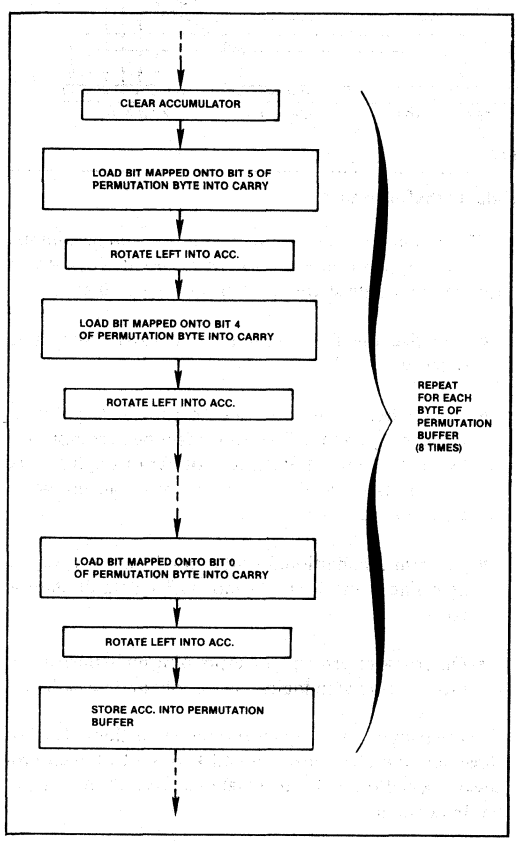


Figure 10.b. DES Key Permutation with Boolean Processor.

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

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.

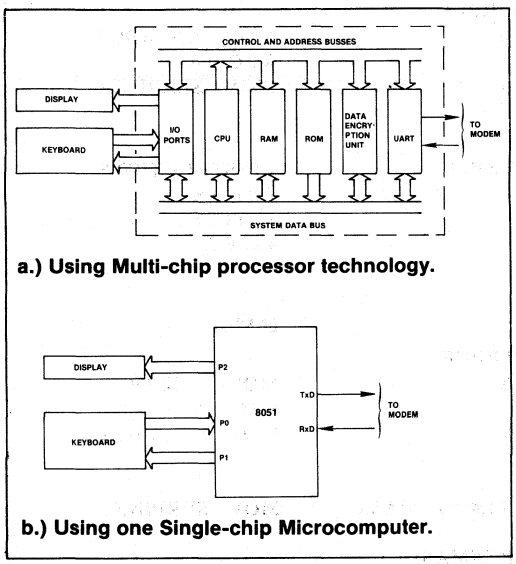


Figure 11. Secure Banking Terminal Block Diagram.

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.

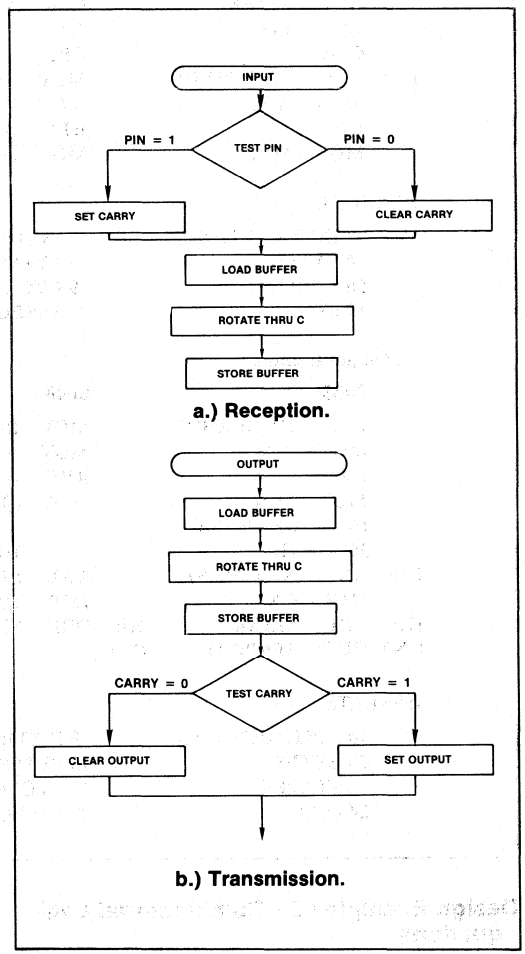


Figure 12. Serial I/O Algorithms.

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

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.

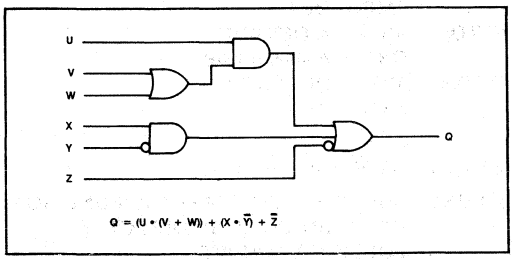
Figure 13 shows TTL and relay logic diagrams for a function of the six variables U through Z. Each is a solution of the equation,

$$Q = (U \cdot (V + W)) + (X \cdot \overline{Y}) + \overline{Z} \quad .$$

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

Figure 13. Hardware Implementations of Boolean functions.



a.) Using TTL:

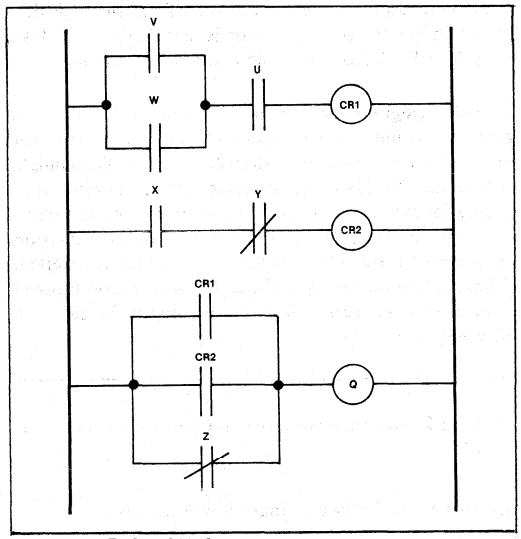
For the sake of comparison we will implement this function three ways, restricting the software to three proper subsets of the MCS-51™ instruction set. We will also assume that U and V are input pins from different input ports, W and X are status bits for two peripheral controllers, and Y and Z are software 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:

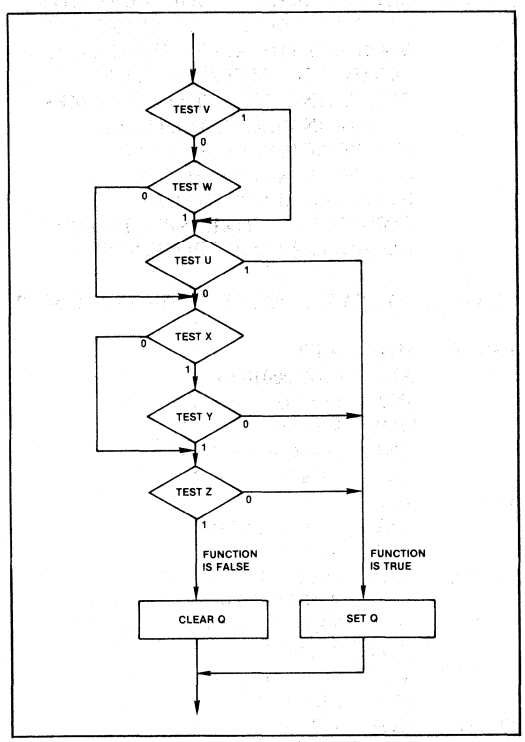


Figure 14. Flow chart for tree-branching algorithm.

01489A

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

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.)

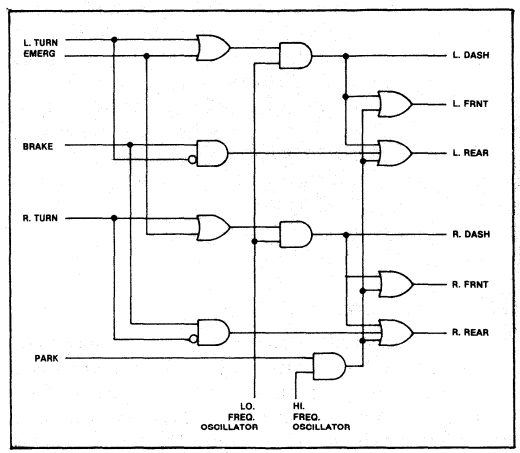


Figure 15. TTL logic implementation of automotive turn signals.

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

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.)

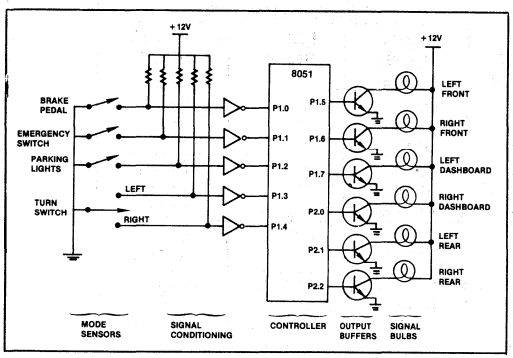
A multiple-conductor wiring harness runs to each corner of the car, behind the dash, up the steering column, and down to the blinker relay below. Connectors at each termination for each filament lead to extra cost and labor during construction, lower reliability and safety, and more costly repairs. And 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-

```
;                ...        .....
; 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

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.

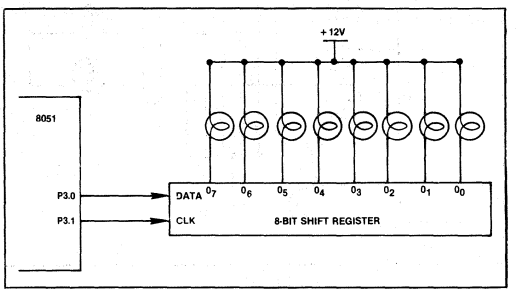


Figure 17. Output expansion using serial port.

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
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

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.

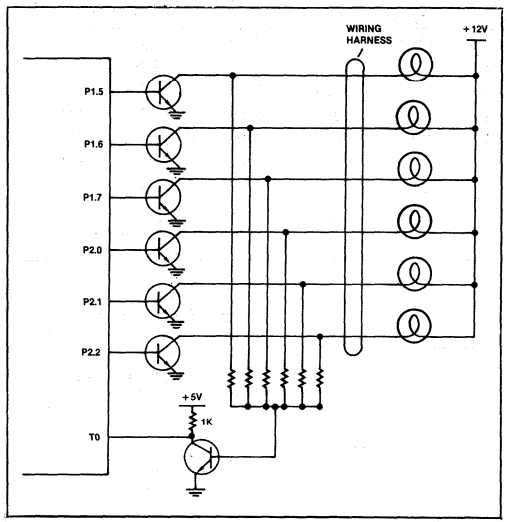


Figure 18.

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.

01489A

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.

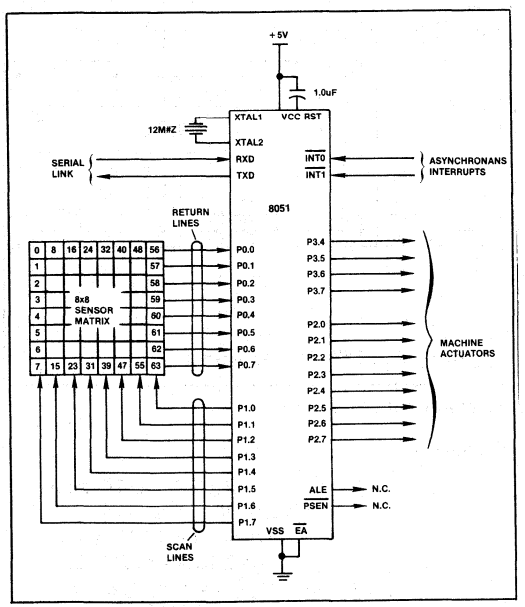


Figure 19. Block diagram of 64-input machine controller.

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

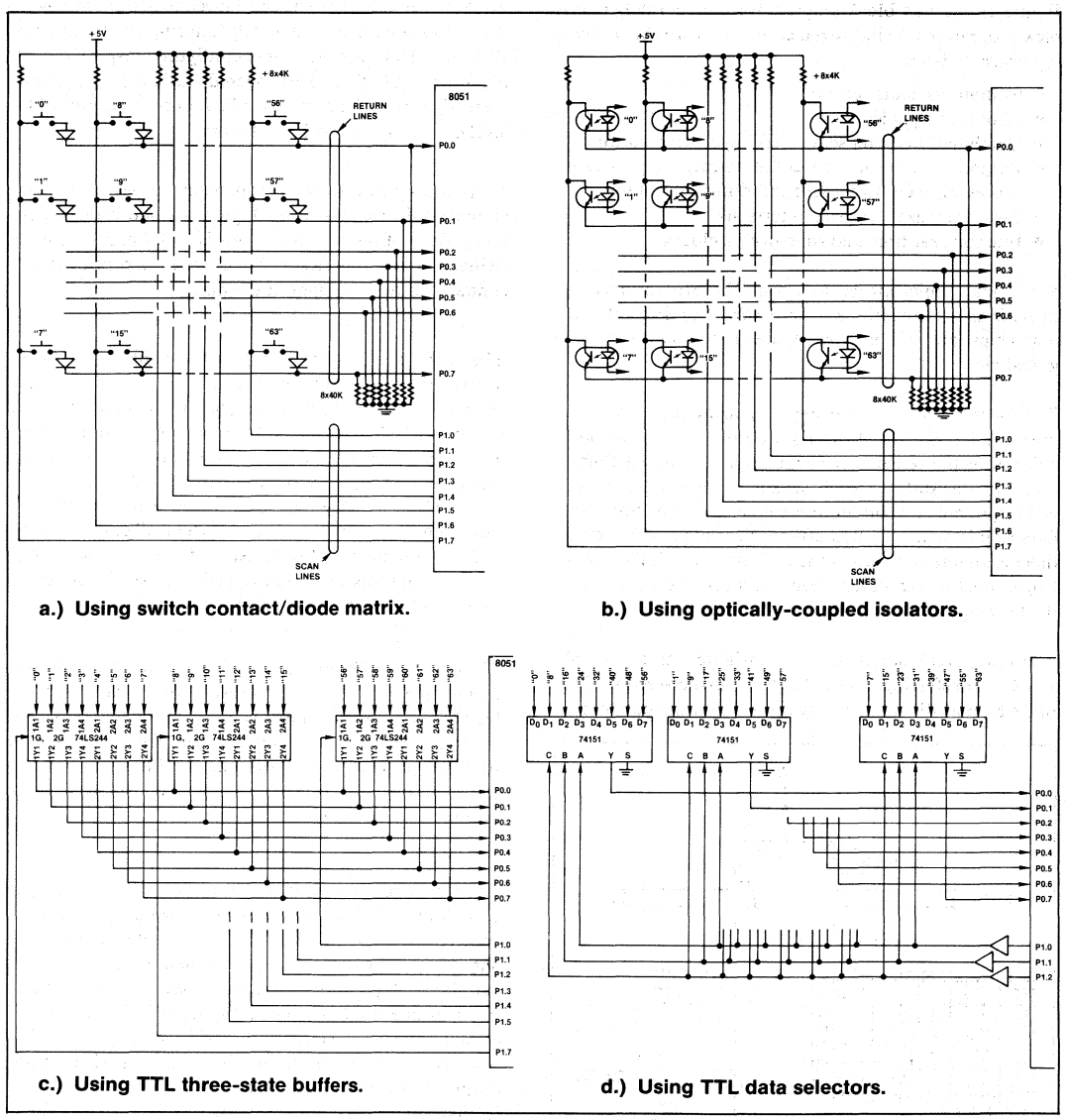


Figure 20. Sensor Matrix Implementation Methods.

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

```
        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
```

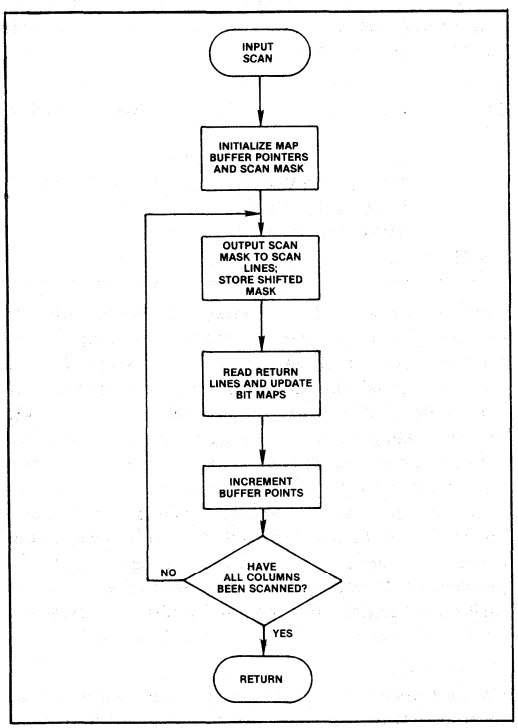


Figure 21. Flowchart for reading in sensor matrix.

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
;       ....      .....
```

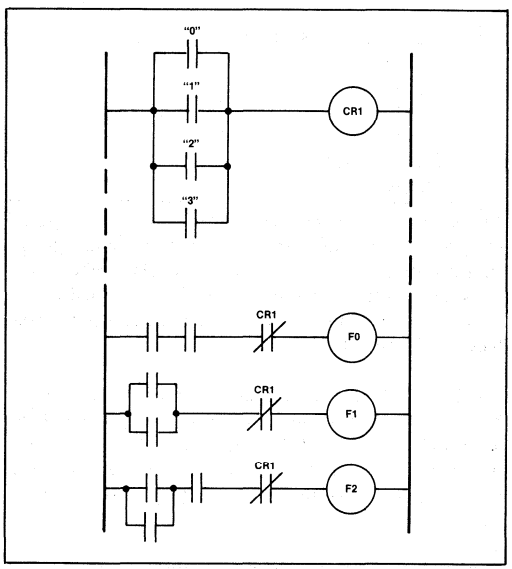


Figure 22. Ladder diagram for output override circuitry.

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.

```
;L_SET   SET FLAG 0 IF C=1
L_SET:    ORL   C,F0
          MOV   F0,C
;

;L_RSET RESET FLAG 0 IF C=1
L_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

01489A

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.

01489A

```
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, TO 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   00BFH  69 96 104#
THO . .   N DSEG   008CH  53 59
TLO . .   N DSEG   00BAH  58
TMOD. .   N DSEG   00B9H  60
TRO . .   N BSEG   00BCH  65
UPDATE.   L CSEG   0054H  55 69#

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

01489A

MCS-51 Component Specifications

7

Asset Component Specifications

PRELIMINARY

8031/8051/8751
SINGLE-COMPONENT 8-BIT MICROCOMPUTER

- 8031 - Control Oriented CPU With RAM and I/O
- 8051 - An 8031 With Factory Mask-Programmable ROM
- 8751 - An 8031 With User Programmable/Erasable EPROM

- 4K x 8 ROM/EPROM
- 128 x 8 RAM
- Four 8-Bit Ports, 32 I/O Lines
- Two 16-Bit Timer/Event Counters
- High-Performance Full-Duplex Serial Channel
- External Memory Expandable to 128K
- Compatible with MCS-80®/MCS-85® Peripherals
- Boolean Processor
- MCS-48® Architecture Enhanced with:
  - Non-Paged Jumps
  - Direct Addressing
  - Four 8-Register Banks
  - Stack Depth Up to 128-Bytes
  - Multiply, Divide, Subtract, Compare
- Most Instructions Execute in 1μs
- 4μs Multiply and Divide

The Intel® 8031/8051/8751 is a stand-alone, high-performance single-chip computer fabricated with Intel's highly-reliable +5 Volt, depletion-load, N-Channel, silicon-gate HMOS technology and packaged in a 40-pin DIP. It provides the hardware features, architectural enhancements and new instructions that are necessary to 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 8051/8751 contains a non-volatile 4K x 8 read-only program memory; a volatile 128 x 8 read/write data memory; 32 I/O lines; two 16-bit timer/counters; a five-source, two-priority-level, nested interrupt structure; a serial I/O port for either multi-processor communications, I/O expansion, or full duplex UART; and on-chip oscillator and clock circuits. The 8031 is identical, except that it lacks the program memory. For systems that require extra capability, the 8051 can be expanded using standard TTL compatible memories and the byte oriented MCS-80 and MCS-85 peripherals.

The 8051 microcomputer, like its 8048 predecessor, is efficient both as a controller and as an arithmetic processor. The 8051 has extensive facilities for binary and BCD arithmetic and excels in bit-handling capabilities. Efficient use of program memory results from an instruction set consisting of 44% one-byte, 41% two-byte, and 15% three-byte instructions. With a 12 MHz crystal, 58% of the instructions execute in 1μs, 40% in 2μs and multiply and divide require only 4μs. Among the many instructions added to the standard 8048 instruction set are multiply, divide, subtract and compare.

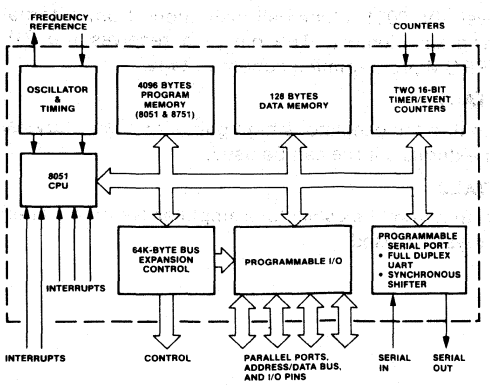


Figure 1.
Block Diagram

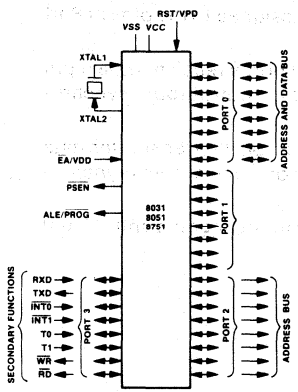


Figure 2.
Logic Symbol

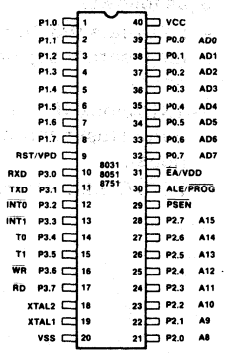


Figure 3. Pin
Configuration

AFN-01462B-01

8051 Family Pin Description

V_{SS}
Circuit ground potential.

V_{CC}
+5V power supply during operation, programming and 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 input and output during programming and verification. Port 0 can sink/source two 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 programming and verification. Port 1 can sink/source one TTL load.

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 programming and verification. Port 2 can sink/source one TTL load.

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 one TTL load. The secondary functions are assigned to the pins of Port 3, as follows:

- RXD/data (P3.0). Serial port's receiver data input (asynchronous) or data input/output (synchronous).
- TXD/clock (P3.1). Serial port's transmitter data output (asynchronous) or clock output (synchronous).
- $\overline{INT0}$ (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.
- T0 (P3.4). Input to counter 0.
- T1 (P3.5). Input to counter 1.
- $\overline{WR}$ (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/V_{PD}

A low to high transition on this pin (at approximately 3V) resets the 8051. If V_{PD} is held within its spec (approximately +5V), while V_{CC} drops below spec, V_{PD} will provide standby power to the RAM. When V_{PD} is low, the RAM's current is drawn from V_{CC}. A small internal resistor permits power-on reset using only a capacitor connected to V_{CC}.

ALE/$\overline{PROG}$

Provides Address Latch Enable output used for latching the address into external memory during normal operation. Receives the program pulse input during EPROM programming.

$\overline{PSEN}$

The Program Store Enable output is a control signal that enables the external Program Memory to the bus during normal fetch operations.

$\overline{EA}$/VDD

When held at a TTL high level, the 8051 executes instructions from the internal ROM/EPROM when the PC is less than 4096. When held at a TTL low level, the 8051 fetches all instuctions from external Program Memory. The pin also receives the 21V EPROM programming supply voltage.

XTAL1

Input to the oscillator's high gain amplifier. A crystal or external source can be used.

XTAL2

Output from the oscillator's amplifier. Required when a crystal 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.

D.C. CHARACTERISTICS (T_A = 0° C to 70° C; V_{CC} = 5V ± 5%; V_{SS} = 0V)

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
V_{IL}	Input Low Voltage (All except XTAL1)	-0.5		0.8	V	
V_{IL1}	Input Low Voltage (XTAL1)	-0.5		TBD	V	
V_{IH}	Input High Voltage (All Except XTAL1, RST/V_{PD})	2.0		V_{CC}+0.5	V	
V_{IH1}	Input High Voltage (XTAL1)	TBD		V_{CC}+0.5	V	
V_{IH2}	Input High Voltage (RST)	3.0		V_{CC} + 0.5	V	
V_{IH3}	Input High Voltage (V_{PD})	4.5		5.5	V	Power Down Only (V_{CC} = 0)
V_{OL}	Output Low Voltage (All Outputs Except Port 0)			0.45	V	I_{OL}=2 mA
V_{OL1}	Output Low Voltage (Port 0)			0.45	V	I_{OL}=4 mA
V_{OH}	Output High Voltage (All Outputs Except Port 0, ALE and $\overline{PSEN}$)	2.4			V	I_{OH}=-100 μA
V_{OH1}	Output High Voltage (ALE and $\overline{PSEN}$, Port 0 in External Bus Mode)	2.4			V	I_{OH}=-400 μA
I_{LO}	Pullup Resistor Current (P1, P2, P3)			500	μA	.45V≤ V_{IN} ≤V_{CC}
I_{LO1}	Output Leakage Current (P0)			±10	μA	.45V≤ V_{IN} ≤V_{CC}
I_{CC}	Power Supply Current (All Outputs Disconnected)			150	mA	T_A=25° C
I_{PD}	Power Down Supply Current			20	mA	T_A=25° C, V_{PD}=5V, V_{CC}=0V
C_{IO}	Capacitance Of I/O Buffer			10	pF	fc=1MHz

A.C. CHARACTERISTICS (T_A = 0°C to 70°C; V_{CC} = 5V ± 5%; V_{SS} = 0V; C_L for Port 0, ALE and $\overline{PSEN}$ Outputs = 150 pF; C_L for All Other Outputs = 80 pF)

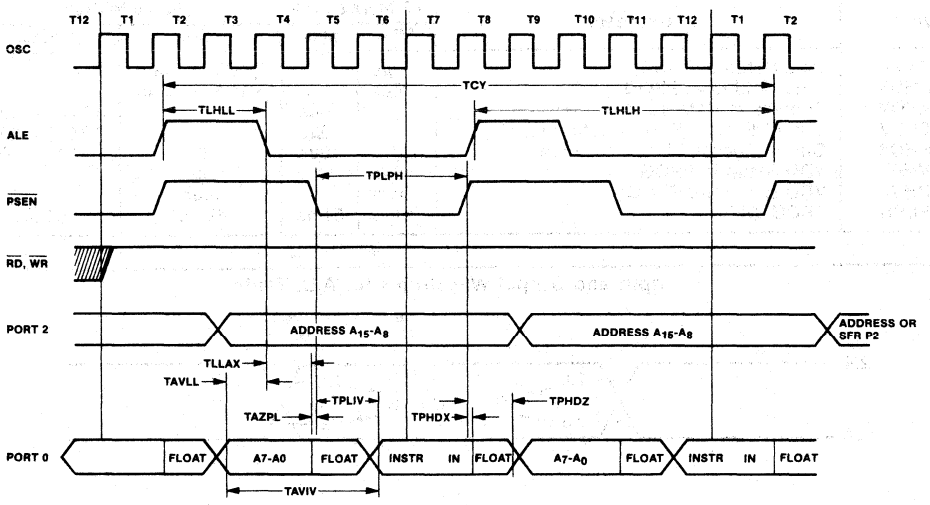
Program Memory Characteristics

Symbol	Parameter	12 MHz Clock			Variable Clock 1/TCLCL=1.2 MHz to 12 MHz		
		Min	Max	Units	Min	Max	Units
TCLCL	Oscillator Period	83		ns			ns
TCY	Min Instruction Cycle Time	1.0		μs	12TCLCL	12TCLCL	ns
TLHLL	ALE Pulse Width	140		ns	2TCLCL-30		ns
TAVLL	Address Set Up To ALE	60		ns	TCLCL-25		ns
TLLAX	Address Hold After ALE	50		ns	TCLCL-35		ns
TPLPH	$\overline{PSEN}$ Width	230		ns	3TCLCL-20		ns
TLHLH	$\overline{PSEN}$, ALE Cycle Time	500		ns	6TCLCL		ns
TPLIV	$\overline{PSEN}$ To Valid Instruction In		150	ns		3TCLCL-100	ns
TPHDX	Input Data Hold After $\overline{PSEN}$	0		ns	0		ns
TPHDZ	Input Data Float After $\overline{PSEN}$		75	ns		TCLCL-10	ns
TAVIV	Address To Valid Instr In		320	ns		5TCLCL-100	ns
TAZPL	Address Float To $\overline{PSEN}$	0		ns	0		ns

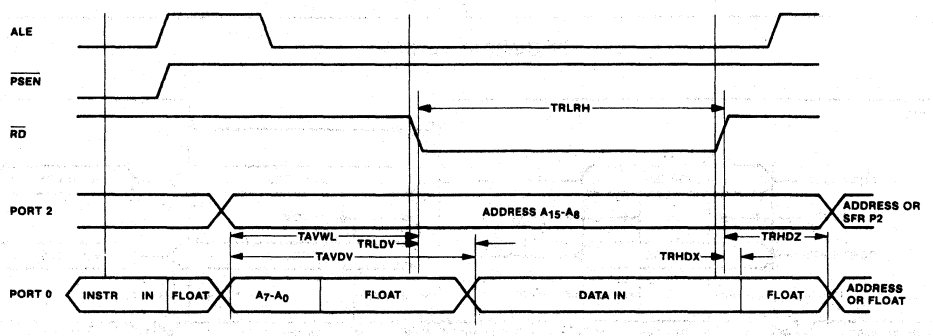
External Data Memory Characteristics

Symbol	Parameter	12 MHz Clock			Variable Clock		
		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
TRLDV	$\overline{RD}$ To Valid Data In		250	ns		5TCLCL-170	ns
TRHDX	Data Hold After $\overline{RD}$	0		ns	0		ns
TRHDZ	Data Float After $\overline{RD}$		100	ns		2TCLCL-70	ns
TAVDV	Address To Valid Data In		600	ns		9TCLCL-150	ns
TAVWL	Address To $\overline{WR}$ or $\overline{RD}$	200		ns	4TCLCL-130		ns
TDVWX	Data Valid To $\overline{WR}$ Transition			ns			ns
TQVWH	Data Setup Before $\overline{WR}$	400		ns	7TCLCL-180		ns
TWHQX	Data Held After $\overline{WR}$	80		ns	2TCLCL-90		ns

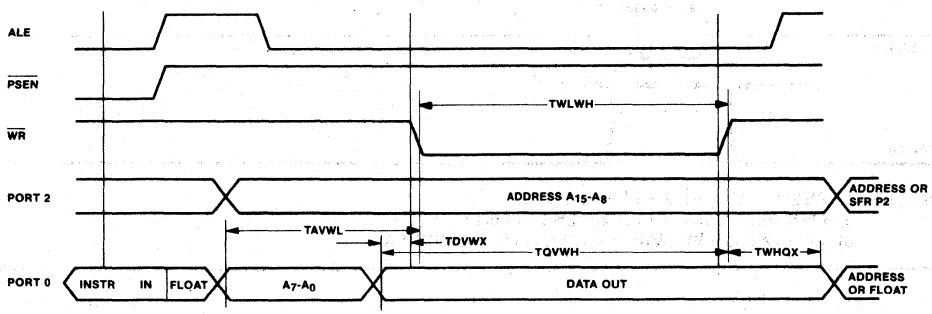
NOTE:
There are 2 to 8 ALE cycles per instruction. Clocks and state timing are shown on the timing diagram for reference purposes only. They are not accessible outside the package. TCY is the minimum instruction cycle time which consists of 12 oscillator clocks or two ALE cycles. Address setup and hold time from ALE are the same for data and program memory.



Program Memory Read Cycle



Data Memory Read Cycle

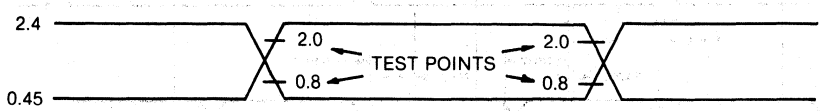


Data Memory Write Cycle

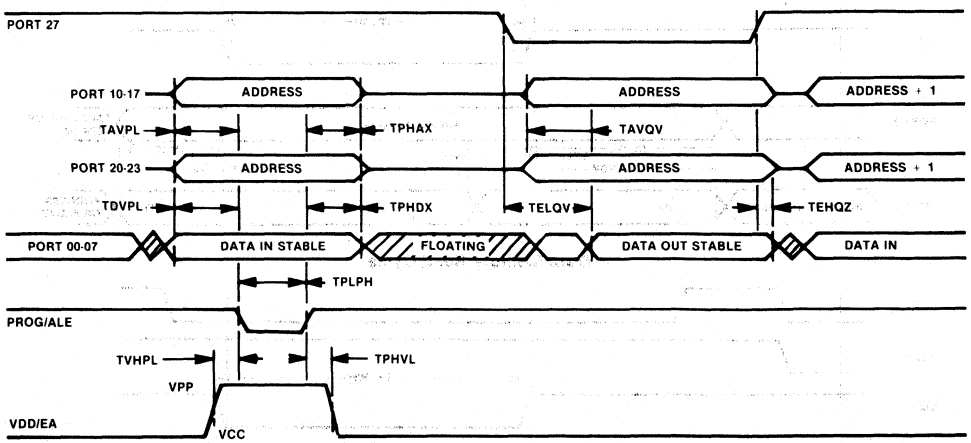
PROGRAM VERIFY

Symbol	Parameter	12 MHz Clock		Variable Clock	
		Min.	Max.	Min.	Max.
TDVPL	Data Setup to PROG	10μs		3 TCY + 10μs	
TPHDX	Data Hold from PROG	10μs		3 TCY + 10μs	
TAVQV	Address to Data Valid		10μs		3 TCY + 10μs
TELQV	Output Enable (P27) to Data Valid		10μs		3 TCY + 10μs
TEHQZ	Output Enable Off to Data Float	0	10μs	0	3 TCY + 10μs
TVHPL	VDD Setup to PROG	10μs		10μs	
TPHVL	VDD Hold after PROG	10μs		10μs	
TPLPH	PROG Width	49ms	51ms	49ms	51ms

Input and Output Waveforms for A.C. Tests



PROGRAM VERIFY



NOTES: 1) PSEN = VIL, VPD/RST = VIH1
2) MSB of address is Port 23, LSB is Port 0
3) All levels are VIL, VIH, VOL, VOH except VDD/EA.

Serial Port Timing — Shift Register Mode
Conditions: T_A = -40°C to 85°C; V_{CC} = 5V ± 10%; V_{SS} = OV; C_L = 80pF

Symbol	Parameter	12 MHz Clock		Variable Clock	
		Min	Max	Min	Max
TCLCL	Serial Port Clock Cycle Time	1.0 μs		TCY	
TQVCH	Output Data Setup to Clock Rising Edge	750 ns		5TCY/6-80 ns	
TCHQX	Output Data Hold After Clock Rising Edge	80 ns		TCY/6-80 ns	
TCHDV	Clock Rising Edge to Input Data Valid		750 ns		5TCY/6-80 ns
TCHDX	Input Data Hold After Clock Rising Edge	0		0	

Component Data Sheets

8

2114A
1024 X 4 BIT STATIC RAM

	2114AL-1	2114AL-2	2114AL-3	2114AL-4	2114A-4	2114A-5	2114A-6
Max. Access Time (ns)	100	120	150	200	200	250	300
Max. Current (ma)	40	40	40	40	70	70	70

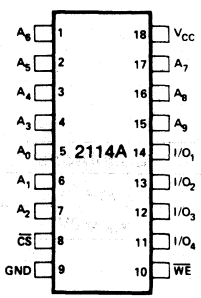
- **HMOS Technology**
- **Low Power, High Speed**
- **Identical Cycle and Access Times**
- **Single +5V Supply ±10%**
- **High Density 18 Pin Package**
- **Completely Static Memory - No Clock or Timing Strobe Required**
- **Directly TTL Compatible: All Inputs and Outputs**
- **Common Data Input and Output Using Three-State Outputs**
- **2114 Upgrade**

The Intel® 2114A is a 4096-bit static Random Access Memory organized as 1024 words by 4-bits using HMOS, a high performance MOS technology. It uses fully DC stable (static) circuitry throughout, in both the array and the decoding, therefore it requires no clocks or refreshing to operate. Data access is particularly simple since address setup times are not required. The data is read out nondestructively and has the same polarity as the input data. Common input/output pins are provided.

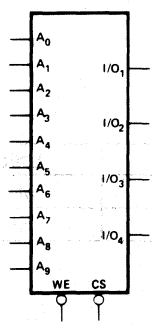
The 2114A is designed for memory applications where the high performance and high reliability of HMOS, low cost, large bit storage, and simple interfacing are important design objectives. The 2114A is placed in an 18-pin package for the highest possible density.

It is directly TTL compatible in all respects: inputs, outputs, and a single +5V supply. A separate Chip Select ($\overline{CS}$) lead allows easy selection of an individual package when outputs are or-tied.

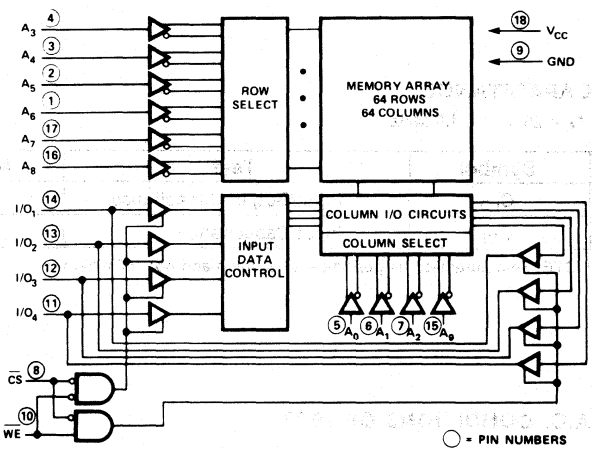
PIN CONFIGURATION



LOGIC SYMBOL



BLOCK DIAGRAM



PIN NAMES

A_0–A_9	ADDRESS INPUTS	V_{CC} POWER (+5V)
$\overline{WE}$	WRITE ENABLE	GND GROUND
$\overline{CS}$	CHIP SELECT	
I/O_1–I/O_4	DATA INPUT/OUTPUT	

March, 1981

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias ..... -10°C to 80°C
Storage Temperature ..... -65°C to 150°C
Voltage on any Pin
With Respect to Ground ..... -3.5V to +7V
Power Dissipation 1.0W
D.C. Output Current 5mA

*COMMENT: 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 is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. AND OPERATING CHARACTERISTICS

T_A = 0°C to 70°C, V_{CC} = 5V ± 10%, unless otherwise noted.

SYMBOL	PARAMETER	2114AL-1/L-2/L-3/L-4 Min.	Typ.[1]	Max.	2114A-4/-5/-6 Min.	Typ.[1]	Max.	UNIT	CONDITIONS
$\|I_{LI}\|$	Input Load Current (All Input Pins)			10			10	μA	V_{IN} = 0 to 5.5V
$\|I_{LO}\|$	I/O Leakage Current			10			10	μA	$\overline{CS}$ = V_{IH} $V_{I/O}$ = GND to VCC
I_{CC}	Power Supply Current		25	40		50	70	mA	V_{CC} = max, $I_{I/O}$ = 0 mA, T_A = 0°C
V_{IL}	Input Low Voltage	-3.0		0.8	-3.0		0.8	V	
V_{IH}	Input High Voltage	2.0		6.0	2.0		6.0	V	
I_{OL}	Output Low Current	2.1	9.0		2.1	9.0		mA	V_{OL} = 0.4V
I_{OH}	Output High Current	-1.0	-2.5		-1.0	-2.5		mA	V_{OH} = 2.4V
I_{OS}[2]	Output Short Circuit Current			40			40	mA	

NOTE: 1. Typical values are for T_A = 25°C and V_{CC} = 5.0V.
2. Duration not to exceed 30 seconds.

CAPACITANCE

T_A = 25°C, f = 1.0 MHz

Symbol	Test	Max.	Unit	Conditions
$C_{I/O}$	Input/Output Capacitance	5	pF	$V_{I/O}$ = 0V
C_{IN}	Input Capacitance	5	pF	V_{IN} = 0V

NOTE: This parameter is periodically sampled and not 100% tested.

A.C. CONDITIONS OF TEST

Input Pulse Levels ..... 0.8 Volt to 2.0 Volt
Input Rise and Fall Times ..... 10 nsec
Input and Output Timing Levels ..... 1.5 Volts
Output Load ..... 1 TTL Gate and C_L = 100 pF

A.C. CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = 5V ± 10%, unless otherwise noted.

READ CYCLE [1]

Symbol	Parameter	2114AL-1 Min.	2114AL-1 Max.	2114AL-2 Min.	2114AL-2 Max.	2114AL-3 Min.	2114AL-3 Max.	2114A-4/L-4 Min.	2114A-4/L-4 Max.	2114A-5 Min.	2114A-5 Max.	2114A-6 Min.	2114A-6 Max.	Unit
t_{RC}	Read Cycle Time	100		120		150		200		250		300		ns
t_A	Access Time		100		120		150		200		250		300	ns
t_{CO}	Chip Selection to Output Valid		70		70		70		70		85		85	ns
t_{CX}	Chip Selection to Output Active	10		10		10		10		10		10		ns
t_{OTD}	Output 3-state from Deselection		30		35		40		50		60		60	ns
t_{OHA}	Output Hold from Address Change	15		15		15		15		15		15		ns

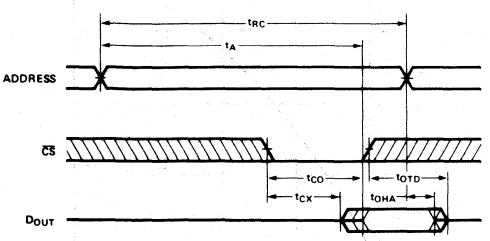
WRITE CYCLE [2]

Symbol	Parameter	2114AL-1 Min.	2114AL-1 Max.	2114AL-2 Min.	2114AL-2 Max.	2114AL-3 Min.	2114AL-3 Max.	2114A-4/L-4 Min.	2114A-4/L-4 Max.	2114A-5 Min.	2114A-5 Max.	2114A-6 Min.	2114A-6 Max.	Unit
t_{WC}	Write Cycle Time	100		120		150		200		250		300		ns
t_W	Write Time	75		75		90		120		135			135	ns
t_{WR}	Write Release Time	0		0		0		0		0		0		ns
t_{OTW}	Output 3-state from Write		30		35		40		50		60		60	ns
t_{DW}	Data to Write Time Overlap	70		70		90		120		135		135		ns
t_{DH}	Data Hold from Write Time	0		0		0		0		0		0		ns

NOTES:
1. A Read occurs during the overlap of a low $\overline{CS}$ and a high $\overline{WE}$.
2. A Write occurs during the overlap of a low $\overline{CS}$ and a low $\overline{WE}$. t_W is measured from the latter of $\overline{CS}$ or $\overline{WE}$ going low to the earlier of $\overline{CS}$ or $\overline{WE}$ going high.

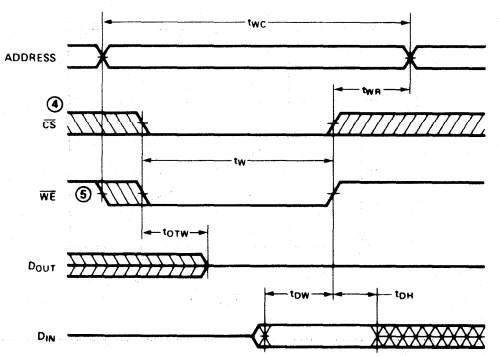
WAVEFORMS

READ CYCLE ③



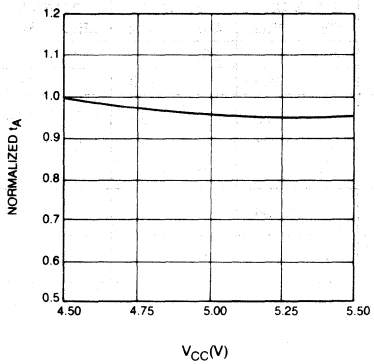
NOTES:
3. $\overline{WE}$ is high for a Read Cycle.
4. If the $\overline{CS}$ low transition occurs simultaneously with the $\overline{WE}$ low transition, the output buffers remain in a high impedance state.
5. $\overline{WE}$ must be high during all address transitions.

WRITE CYCLE

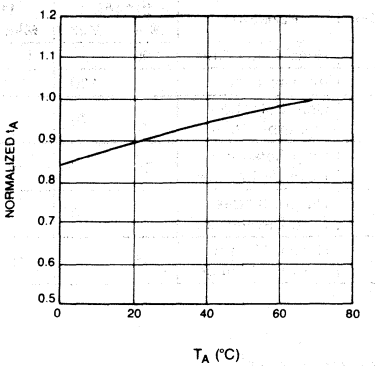


TYPICAL D.C. AND A.C. CHARACTERISTICS

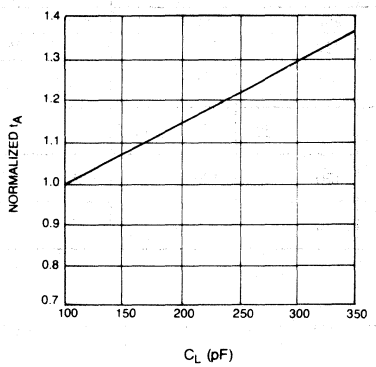
NORMALIZED ACCESS TIME VS. SUPPLY VOLTAGE



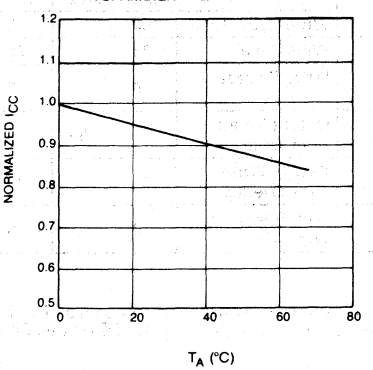
NORMALIZED ACCESS TIME VS. AMBIENT TEMPERATURE



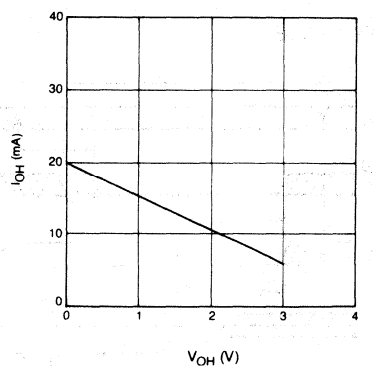
NORMALIZED ACCESS TIME VS. OUTPUT LOAD CAPACITANCE



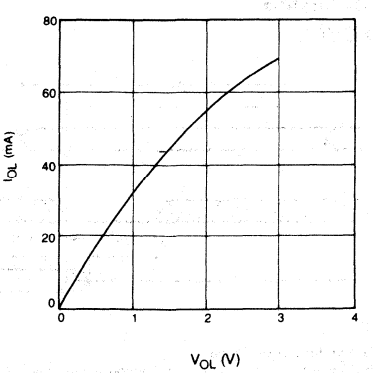
NORMALIZED POWER SUPPLY CURRENT VS. AMBIENT TEMPERATURE



OUTPUT SOURCE CURRENT VS. OUTPUT VOLTAGE



OUTPUT SINK CURRENT VS. OUTPUT VOLTAGE



ADVANCE INFORMATION

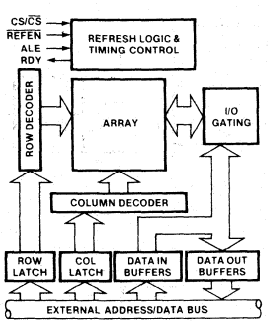
800 SERIES
21821
4096 x 8-BIT PSEUDOSTATIC RAM

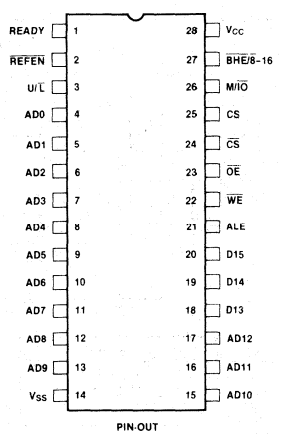
- **Multiplexed Address and Data Buses**
- **8 or 16-Bit System Capability**
- **On-Chip Arbitration**
- **Processor Handshake**
- **Proven HMOS-D2 Reliability**
- **50 mA Active, 10 mA Standby**
- **Fully Integrated Refresh**
- **Fully Integrated Control Logic**
- **Automatic Wake-Up**
- **Multiple Refresh Modes**
- **200 ns Access, 400 ns Cycle Times**

The Intel 21821 is a 4096-word by 8-bit pseudostatic random access memory — PSRAM. Integrating all refresh control circuitry at the chip level allows the system designer to take advantage of dynamic RAM density, performance and price without the added cost of designing the refresh control interface. The 21821 is intended for use with multiplexed address/data bus microprocessors such as the Intel iAPX 86 and iAPX 88. Flexible data bus control options permit the 21821 to be arranged for operation on either 8-bit or 16-bit data buses. All operating parameters have been optimized for high performance, no waitstate operation without TTL interface components. Output drive capabilities and access times are consistent with all present and future microprocessors.

Complete on-chip refresh control circuitry includes refresh address counting and multiplexing, refresh interval timing, and high speed request arbitration. Refresh operation may be controlled either externally by synchronously strobing the $\overline{\text{REFEN}}$ input 128 times in 2 msec, or internally by holding $\overline{\text{REFEN}}$ input to ground. In the internal mode of refresh operation, all refresh is automatic and nearly transparent to the user. A handshake signal to the processor, called READY, is provided to indicate when memory access occurs during refresh operation. READY is used to stop the microprocessor until the 21821 has finished doing refresh. READY, an open drain output, is then released to be pulled back high to V_{CC}, and the 21821 continues with the externally requested access cycle.

Data output location is controlled by two external pins. The internal 8-bit data bus can be shifted between the upper eight data I/O pins or the lower eight data I/O pins. This allows the same part to be used in either 8-bit or 16-bit systems. Multiple levels of chip selects permit linear selection of RAM by the highest order addresses.





PIN-OUT

PIN NAMES

AD0-AD12	ADDRESS/DATA I/O
D13-D15	DATA I/O
READY	READY
$\overline{\text{REFEN}}$	REFRESH ENABLE
$\overline{\text{WE}}$	WRITE ENABLE
$\overline{\text{OE}}$	OUTPUT ENABLE
ALE	ADDRESS LATCH ENABLE
$\overline{\text{BHE}}$/8-16	BYTE HIGH ENABLE/8-16
U/L	UPPER/LOWER
CS, $\overline{\text{CS}}$, M/$\overline{\text{IO}}$	CHIP SELECTS
V_{SS}	GROUND
V_{CC}	+5 VOLTS

2118 FAMILY
16,384 x 1 BIT DYNAMIC RAM

	2118-3	2118-4	2118-7
Maximum Access Time (ns)	100	120	150
Read, Write Cycle (ns)	235	270	320
Read-Modify-Write Cycle (ns)	285	320	410

- **Single +5V Supply, ±10% Tolerance**
- **HMOS Technology**
- **Low Power: 150 mW Max. Operating**
  11 mW Max. Standby
- **Low V_{DD} Current Transients**
- **All Inputs, Including Clocks, TTL Compatible**
- **$\overline{CAS}$ Controlled Output is Three-State, TTL Compatible**
- **$\overline{RAS}$ Only Refresh**
- **128 Refresh Cycles Required Every 2ms**
- **Page Mode and Hidden Refresh Capability**
- **Allows Negative Overshoot V_{IL} min = -2V**

The Intel® 2118 is a 16,384 word by 1-bit Dynamic MOS RAM designed to operate from a single +5V power supply. The 2118 is fabricated using HMOS — a production proven process for high performance, high reliability, and high storage density.

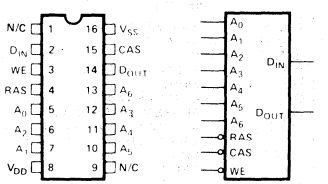
The 2118 uses a single transistor dynamic storage cell and advanced dynamic circuitry to achieve high speed with low power dissipation. The circuit design minimizes the current transients typical of dynamic RAM operation. These low current transients contribute to the high noise immunity of the 2118 in a system environment.

Multiplexing the 14 address bits into the 7 address input pins allows the 2118 to be packaged in the industry standard 16-pin DIP. The two 7-bit address words are latched into the 2118 by the two TTL clocks, Row Address Strobe ($\overline{RAS}$) and Column Address Strobe ($\overline{CAS}$). Non-critical timing requirements for $\overline{RAS}$ and $\overline{CAS}$ allow use of the address multiplexing technique while maintaining high performance.

The 2118 three-state output is controlled by $\overline{CAS}$, independent of $\overline{RAS}$. After a valid read or read-modify-write cycle, data is latched on the output by holding $\overline{CAS}$ low. The data out pin is returned to the high impedance state by returning $\overline{CAS}$ to a high state. The 2118 hidden refresh feature allows $\overline{CAS}$ to be held low to maintain latched data while $\overline{RAS}$ is used to execute $\overline{RAS}$-only refresh cycles.

The single transistor storage cell requires refreshing for data retention. Refreshing is accomplished by performing $\overline{RAS}$-only refresh cycles, hidden refresh cycles, or normal read or write cycles on the 128 address combinations of A_0 through A_6 during a 2ms period. A write cycle will refresh stored data on all bits of the selected row except the bit which is addressed.

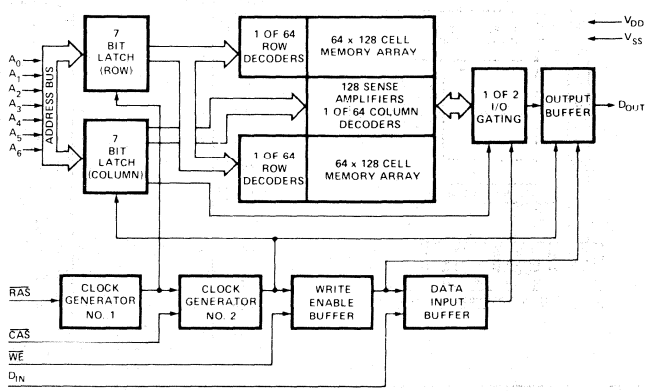
PIN CONFIGURATION



LOGIC SYMBOL

A_0-A_6	ADDRESS INPUTS
$\overline{CAS}$	COLUMN ADDRESS STROBE
D_{IN}	DATA IN
D_{OUT}	DATA OUT
$\overline{WE}$	WRITE ENABLE
$\overline{RAS}$	ROW ADDRESS STROBE
V_{DD}	POWER (+5V)
V_{SS}	GROUND

BLOCK DIAGRAM



ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias ... -10°C to +80°C
Storage Temperature ..... -65°C to +150°C
Voltage on Any Pin Relative to V_{SS} 7.5V
Data Out Current 50mA
Power Dissipation 1.0W

*COMMENT:

Stresses above those listed under "Absolute Maximum Rating" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or at any other condition 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. AND OPERATING CHARACTERISTICS[1]

T_A = 0°C to 70°C, V_{DD} = 5V ±10%, V_{SS} = 0V, unless otherwise noted.

Symbol	Parameter	Limits Min.	Typ.[2]	Max.	Unit	Test Conditions	Notes
$\|I_{LI}\|$	Input Load Current (any input)		0.1	10	µA	V_{IN}=V_{SS} to V_{DD}	
$\|I_{LO}\|$	Output Leakage Current for High Impedance State		0.1	10	µA	Chip Deselected: $\overline{CAS}$ at V_{IH}, V_{OUT} = 0 to 5.5V	
I_{DD1}	V_{DD} Supply Current, Standby		1.2	2	mA	$\overline{CAS}$ and $\overline{RAS}$ at V_{IH}	
I_{DD2}	V_{DD} Supply Current, Operating		23	27	mA	2118-3, t_{RC} = t_{RCMIN}	3
			21	25	mA	2118-4, t_{RC} = t_{RCMIN}	3
			19	23	mA	2118-7, t_{RC} = t_{RCMIN}	3
I_{DD3}	V_{DD} Supply Current, $\overline{RAS}$-Only Cycle		16	18	mA	2118-3, t_{RC} = t_{RCMIN}	3
			14	16	mA	2118-4, t_{RC} = t_{RCMIN}	3
			12	14	mA	2118-7, t_{RC} = t_{RCMIN}	3
I_{DD5}	V_{DD} Supply Current, Standby, Output Enabled		2	4	mA	$\overline{CAS}$ at V_{IL}, $\overline{RAS}$ at V_{IH}	3
V_{IL}	Input Low Voltage (all inputs)	-2.0		0.8	V		
V_{IH}	Input High Voltage (all inputs)	2.4		7.0	V		
V_{OL}	Output Low Voltage			0.4	V	I_{OL} = 4.2mA	
V_{OH}	Output High Voltage	2.4			V	I_{OH} = -5mA	

NOTES:
1. All voltages referenced to V_{SS}.
2. Typical values are for T_A = 25°C and nominal supply voltages.
3. I_{DD} is dependent on output loading when the device output is selected. Specified $I_{DD\ MAX}$ is measured with the output open.

CAPACITANCE[1]

T_A = 25°C, V_{DD} = 5V ±10%, V_{SS} = 0V, unless otherwise noted.

Symbol	Parameter	Typ.	Max.	Unit
C_{I1}	Address, Data In	3	5	pF
C_{I2}	$\overline{RAS}$, $\overline{CAS}$, $\overline{WE}$, Data Out	4	7	pF

NOTES:
1. Capacitance measured with Boonton Meter or effective capacitance calculated from the equation:
$C = \frac{I\Delta t}{\Delta V}$ with ΔV equal to 3 volts and power supplies at nominal levels.

A.C. CHARACTERISTICS[1,2,3]

T_A = 0°C to 70°C, V_{DD} = 5V ±10%, V_{SS} = 0V, unless otherwise noted.

READ, WRITE, READ-MODIFY-WRITE AND REFRESH CYCLES

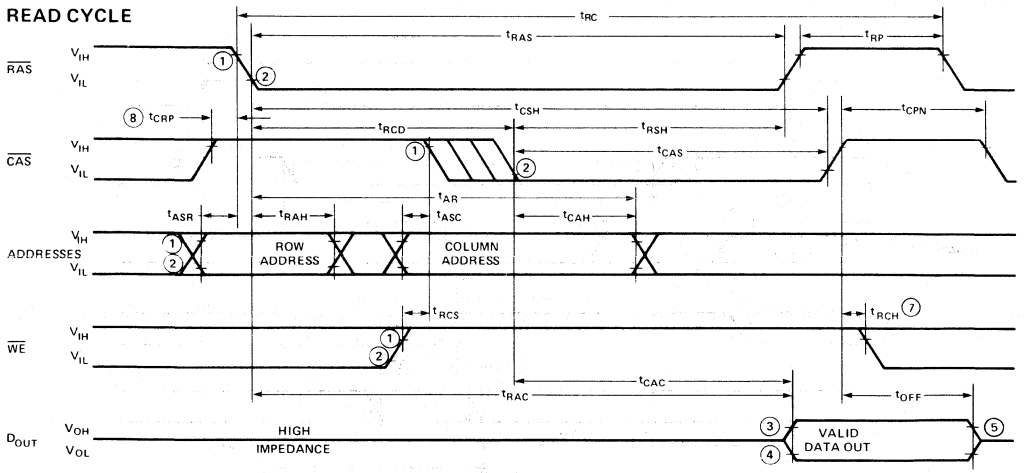
Symbol	Parameter	2118-3		2118-4		2118-7		Unit	Notes
		Min.	Max.	Min.	Max.	Min.	Max.		
t_{RAC}	Access Time From $\overline{RAS}$		100		120		150	ns	4,5
t_{CAC}	Access Time From $\overline{CAS}$		55		65		80	ns	4,5,6
t_{REF}	Time Between Refresh		2		2		2	ms	
t_{RP}	$\overline{RAS}$ Precharge Time	110		120		135		ns	
t_{CPN}	$\overline{CAS}$ Precharge Time(non-page cycles)	50		55		70		ns	
t_{CRP}	$\overline{CAS}$ to $\overline{RAS}$ Precharge Time	0		0		0		ns	
t_{RCD}	$\overline{RAS}$ to $\overline{CAS}$ Delay Time	25	45	25	55	25	70	ns	7
t_{RSH}	$\overline{RAS}$ Hold Time	70		85		105		ns	
t_{CSH}	$\overline{CAS}$ Hold Time	100		120		165		ns	
t_{ASR}	Row Address Set-Up Time	0		0		0		ns	
t_{RAH}	Row Address Hold Time	15		15		15		ns	
t_{ASC}	Column Address Set-Up Time	0		0		0		ns	
t_{CAH}	Column Address Hold Time	15		15		20		ns	
t_{AR}	Column Address Hold Time, to $\overline{RAS}$	60		70		90		ns	
t_T	Transition Time (Rise and Fall)	3	50	3	50	3	50	ns	8
t_{OFF}	Output Buffer Turn Off Delay	0	45	0	50	0	60	ns	
READ AND REFRESH CYCLES									
t_{RC}	Random Read Cycle Time	235		270		320		ns	
t_{RAS}	$\overline{RAS}$ Pulse Width	115	10000	140	10000	175	10000	ns	
t_{CAS}	$\overline{CAS}$ Pulse Width	55	10000	65	10000	95	10000	ns	
t_{RCS}	Read Command Set-Up Time	0		0		0		ns	
t_{RCH}	Read Command Hold Time	0		0		0		ns	
WRITE CYCLE									
t_{RC}	Random Write Cycle Time	235		270		320		ns	
t_{RAS}	$\overline{RAS}$ Pulse Width	115	10000	140	10000	175	10000	ns	
t_{CAS}	$\overline{CAS}$ Pulse Width	55	10000	65	10000	95	10000	ns	
t_{WCS}	Write Command Set-Up Time	0		0		0		ns	9
t_{WCH}	Write Command Hold Time	25		30		45		ns	
t_{WCR}	Write Command Hold Time, to $\overline{RAS}$	70		85		115		ns	
t_{WP}	Write Command Pulse Width	25		30		50		ns	
t_{RWL}	Write Command to $\overline{RAS}$ Lead Time	60		65		110		ns	
t_{CWL}	Write Command to $\overline{CAS}$ Lead Time	45		50		100		ns	
t_{DS}	Data-In Set-Up Time	0		0		0		ns	
t_{DH}	Data-In Hold Time	25		30		45		ns	
t_{DHR}	Data-In Hold Time, to $\overline{RAS}$	70		85		115		ns	
READ-MODIFY-WRITE CYCLE									
t_{RWC}	Read-Modify-Write Cycle Time	285		320		410		ns	
t_{RRW}	RMW Cycle $\overline{RAS}$ Pulse Width	165	10000	190	10000	265	10000	ns	
t_{CRW}	RMW Cycle $\overline{CAS}$ Pulse Width	105	10000	120	10000	185	10000	ns	
t_{RWD}	$\overline{RAS}$ to $\overline{WE}$ Delay	100		120		150		ns	9
t_{CWD}	$\overline{CAS}$ to $\overline{WE}$ Delay	55		65		80		ns	9

NOTES:

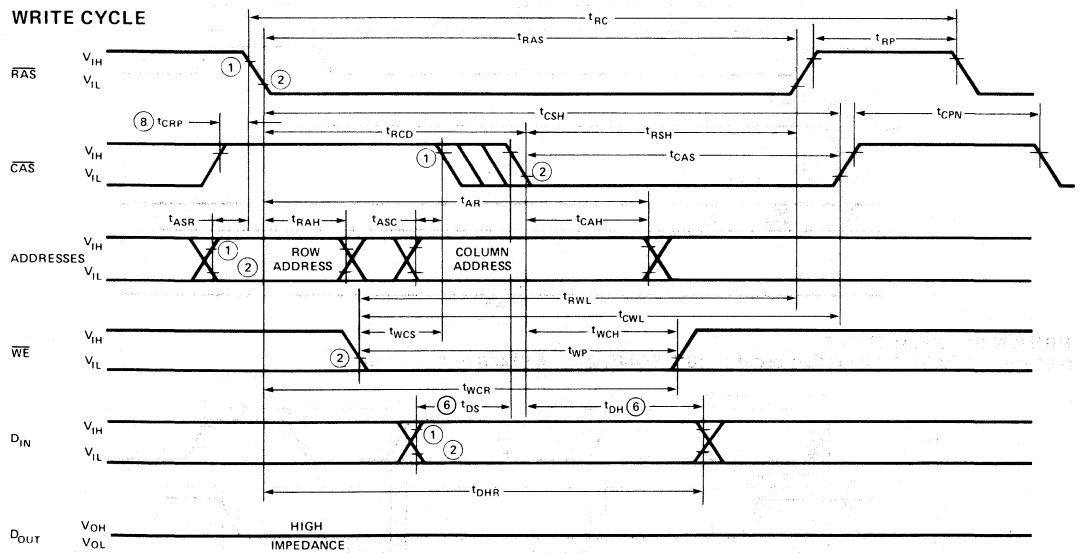
1. All voltages referenced to V_{SS}.
2. Eight cycles are required after power-up or prolonged periods (greater than 2ms) of $\overline{RAS}$ inactivity before proper device operation is achieved. Any 8 cycles which perform refresh are adequate for this purpose.
3. A.C. Characteristics assume t_T = 5ns.
4. Assume that $t_{RCD} \leq t_{RCD\,(max)}$. If t_{RCD} is greater than $t_{RCD\,(max)}$ then t_{RAC} will increase by the amount that t_{RCD} exceeds $t_{RCD\,(max)}$.
5. Load = 2 TTL loads and 100pF.
6. Assumes $t_{RCD} \leq t_{RCD\,(max)}$.
7. $t_{RCD\,(max)}$ is specified as a reference point only. If t_{RCD} is less than $t_{RCD\,(max)}$ access time is t_{RAC}. If t_{RCD} is greater than $t_{RCD\,(max)}$ access time is $t_{RCD} + t_{CAC}$.
8. t_T is measured between $V_{IH\,(min)}$ and $V_{IL\,(max)}$.
9. t_{WCS}, t_{CWD} and t_{RWD} are specified as reference points only. If $t_{WCS} \geq t_{WCS\,(min)}$ the cycle is an early write cycle and the data out pin will remain high impedance throughout the entire cycle. If $t_{CWD} \geq t_{CWD\,(min)}$ and $t_{RWD} \geq t_{RWD\,(min)}$, the cycle is a read-modify-write cycle and the data out will contain the data read from the selected address. If neither of the above conditions is satisfied, the condition of the data out is indeterminate.

WAVEFORMS

READ CYCLE



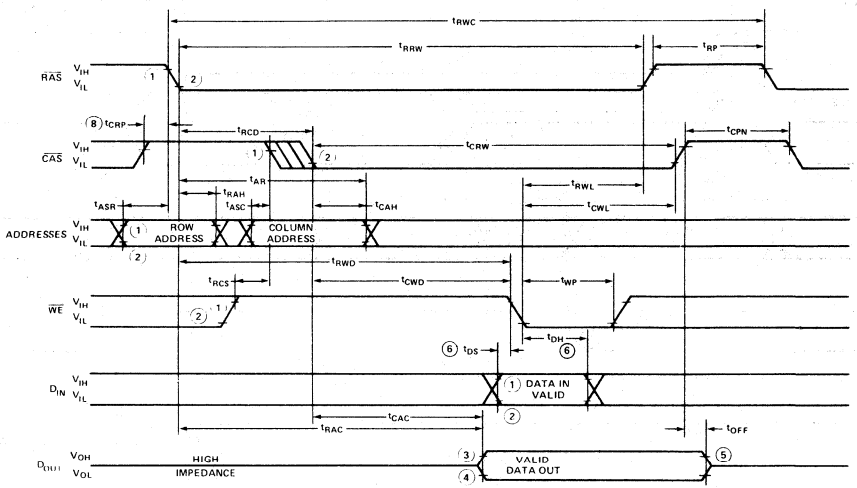
WRITE CYCLE



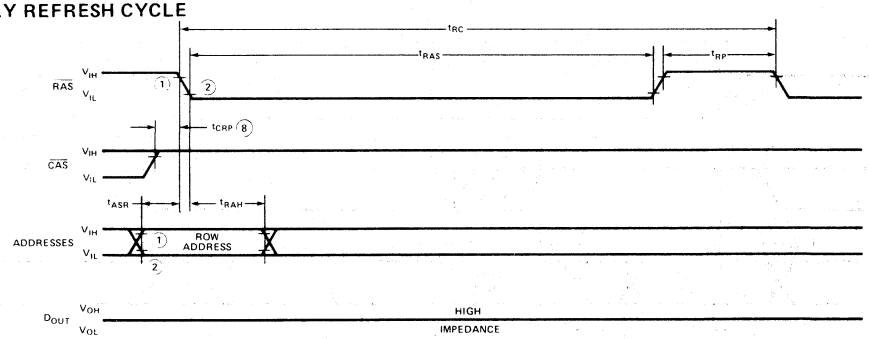
NOTES: 1,2. $V_{IH\ MIN}$ AND $V_{IL\ MAX}$ ARE REFERENCE LEVELS FOR MEASURING TIMING OF INPUT SIGNALS.
3,4. $V_{OH\ MIN}$ AND $V_{OL\ MAX}$ ARE REFERENCE LEVELS FOR MEASURING TIMING OF D_{OUT}.
5. t_{OFF} IS MEASURED TO $I_{OUT} \leq |I_{LO}|$.
6. t_{DS} AND t_{DH} ARE REFERENCED TO $\overline{CAS}$ OR $\overline{WE}$, WHICHEVER OCCURS LAST.
7. t_{RCH} IS REFERENCED TO THE TRAILING EDGE OF $\overline{CAS}$ OR $\overline{RAS}$, WHICHEVER OCCURS FIRST.
8. t_{CRP} REQUIREMENT IS ONLY APPLICABLE FOR $\overline{RAS}$/$\overline{CAS}$ CYCLES PRECEEDED BY A $\overline{CAS}$-ONLY CYCLE (i.e., FOR SYSTEMS WHERE $\overline{CAS}$ HAS NOT BEEN DECODED WITH $\overline{RAS}$).

WAVEFORMS

READ-MODIFY-WRITE CYCLE

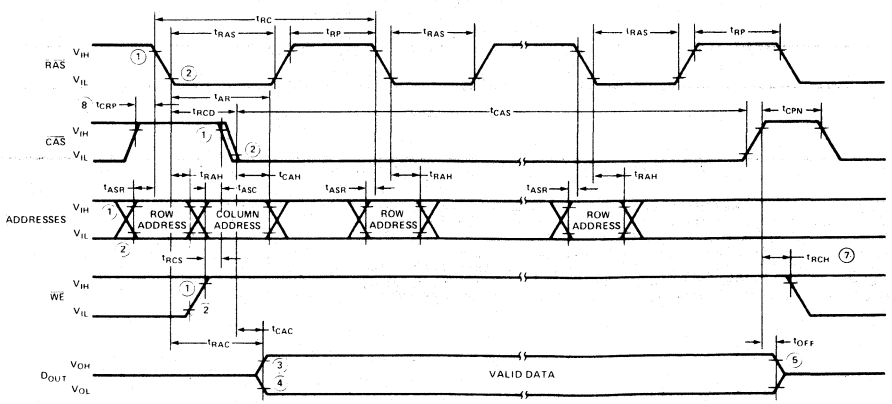


RAS-ONLY REFRESH CYCLE



HIDDEN REFRESH CYCLE

(For Hidden Refresh Operation order 2118-3 S6445, 2118-4 S6446 or 2118-7 S6447)



NOTES:
1,2. $V_{IH\ MIN}$ AND $V_{IL\ MAX}$ ARE REFERENCE LEVELS FOR MEASURING TIMING OF INPUT SIGNALS.
3,4. $V_{OH\ MIN}$ AND $V_{OL\ MAX}$ ARE REFERENCE LEVELS FOR MEASURING TIMING OF D_{OUT}.
5. t_{OFF} IS MEASURED TO $|I_{OUT}| \leq |I_{LO}|$.
6. t_{DS} AND t_{DH} ARE REFERENCED TO $\overline{CAS}$ OR $\overline{WE}$, WHICHEVER OCCURS LAST.
7. t_{RCH} IS REFERENCED TO THE TRAILING EDGE OF $\overline{CAS}$ OR $\overline{RAS}$, WHICHEVER OCCURS FIRST.
8. t_{CRP} REQUIREMENT IS ONLY APPLICABLE FOR $\overline{RAS}$/$\overline{CAS}$ CYCLES PRECEEDED BY A $\overline{CAS}$-ONLY CYCLE (i.e., FOR SYSTEMS WHERE $\overline{CAS}$ HAS NOT BEEN DECODED WITH $\overline{RAS}$).

D.C. AND A.C. CHARACTERISTICS, PAGE MODE [7,8,11]

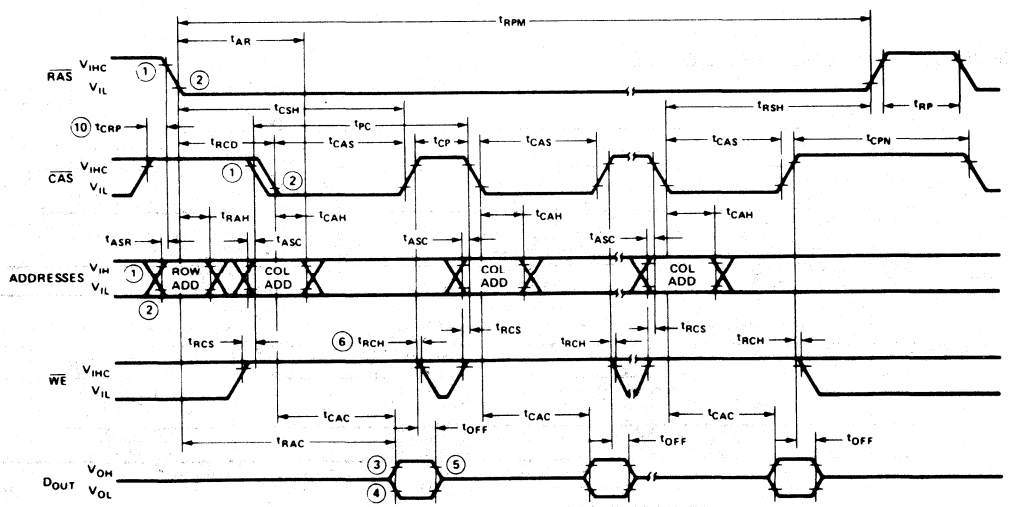
T_A = 0°C to 70°C, V_{DD} = 5V ±10%, V_{SS} = 0V, unless otherwise noted.

For Page Mode Operation order 2118-3 S6329, 2118-4 S6330, or 2118-7 S6331.

Symbol	Parameter	2118-3 S6329 Min.	2118-3 S6329 Max.	2118-4 S6330 Min.	2118-4 S6330 Max.	2118-7 S6331 Min.	2118-7 S6331 Max.	Unit	Notes
t_{PC}	Page Mode Read or Write Cycle	125		145		190		ns	
t_{PCM}	Page Mode Read Modify Write Cycle	175		200		280		ns	
t_{CP}	$\overline{CAS}$ Precharge Time, Page Cycle	60		70		85		ns	
t_{RPM}	$\overline{RAS}$ Pulse Width, Page Mode	115	10000	140	10000	175	10000	ns	
t_{CAS}	$\overline{CAS}$ Pulse Width	55	10000	65	10000	95	10000	ns	
I_{DD4}	V_{DD} Supply Current Page Mode, Minimum t_{PC}, Minimum t_{CAS}		20		17		15	mA	

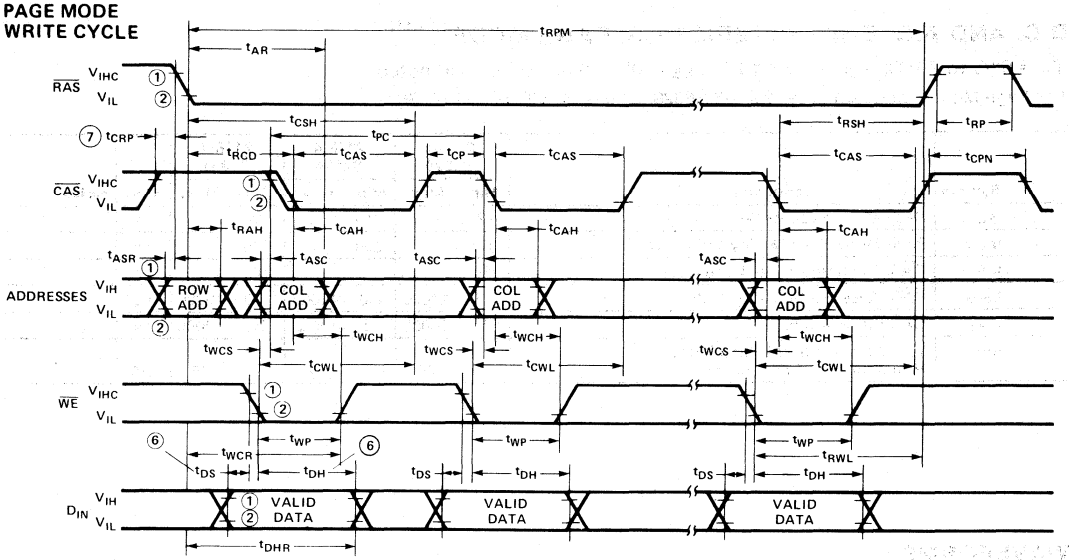
WAVEFORMS

PAGE MODE READ CYCLE

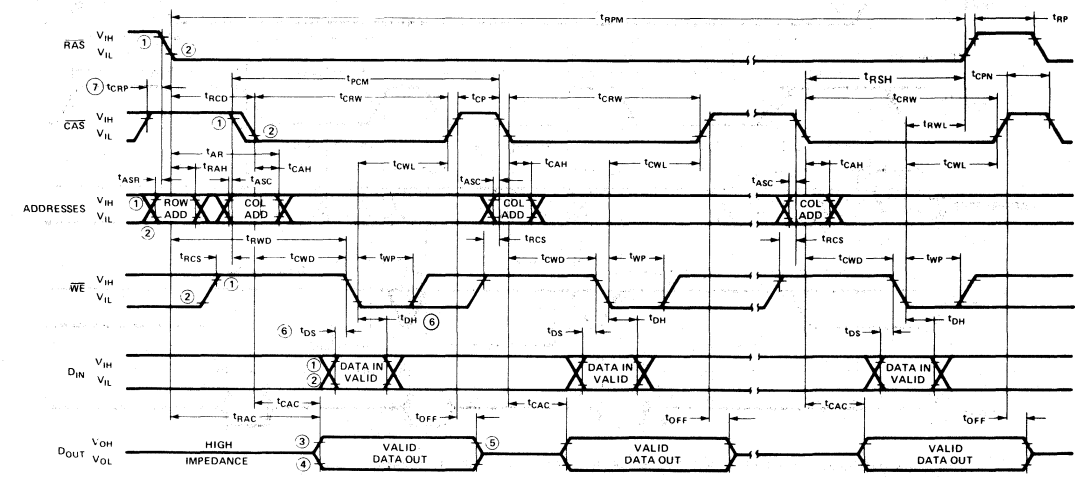


NOTES:
1,2. $V_{IH\ MIN}$ AND $V_{IL\ MAX}$ ARE REFERENCE LEVELS FOR MEASURING TIMING OF INPUT SIGNALS.
3,4. $V_{OH\ MIN}$ AND $V_{OL\ MAX}$ ARE REFERENCE LEVELS FOR MEASURING TIMING OF D_{OUT}.
5. t_{OFF} IS MEASURED TO $|I_{OUT} \cdot I_{LO}|$.
6. t_{RCH} IS REFERENCED TO THE TRAILING EDGE OF $\overline{CAS}$ OR $\overline{RAS}$, WHICHEVER OCCURS FIRST.
7. ALL VOLTAGES REFERENCED TO V_{SS}.
8. AC CHARACTERISTIC ASSUME t_T = 5ns.
9. SEE THE TYPICAL CHARACTERISTICS SECTION FOR VALUES OF THIS PARAMETER UNDER ALTERNATE CONDITIONS.
10. t_{CRP} REQUIREMENT IS ONLY APPLICABLE FOR $\overline{RAS}$/$\overline{CAS}$ CYCLES PRECEEDED BY A $\overline{CAS}$-ONLY CYCLE (i.e., FOR SYSTEMS WHERE $\overline{CAS}$ HAS NOT BEEN DECODED WITH $\overline{RAS}$).
11. ALL PREVIOUSLY SPECIFIED A.C. AND D.C. CHARACTERISTICS ARE APPLICABLE TO THEIR RESPECTIVE PAGE MODE DEVICE (i.e., 2118-3, S6329 WILL OPERATE AS A 2118-3).

PAGE MODE WRITE CYCLE

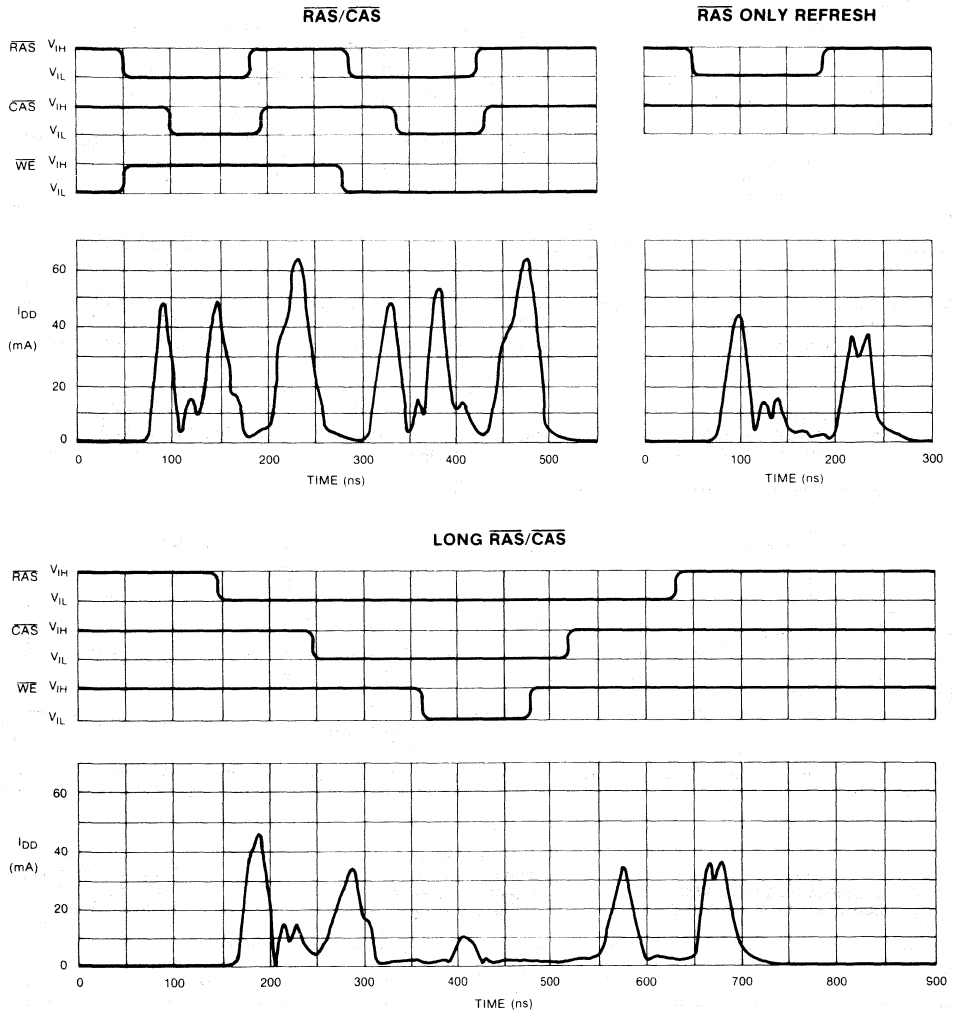


PAGE MODE READ-MODIFY-WRITE CYCLE



NOTES:
1,2. $V_{IH\ MIN}$ AND $V_{IL\ MAX}$ ARE REFERENCE LEVELS FOR MEASURING TIMING OF INPUT SIGNALS.
3,4. $V_{OH\ MIN}$ AND $V_{OL\ MAX}$ ARE REFERENCE LEVELS FOR MEASURING TIMING OF D_{OUT}.
5. t_{OFF} IS MEASURED TO $|I_{OUT}| = |I_{LO}|$.
6. t_{DS} AND t_{DH} ARE REFERENCED TO $\overline{CAS}$ OR $\overline{WE}$, WHICHEVER OCCURS LAST.
7. t_{RCH} IS REFERENCED TO THE TRAILING EDGE OF $\overline{CAS}$ OR $\overline{RAS}$, WHICHEVER OCCURS FIRST.
8. t_{CRP} REQUIREMENT IS ONLY APPLICABLE FOR $\overline{RAS}$/$\overline{CAS}$ CYCLES PRECEEDED BY A $\overline{CAS}$-ONLY CYCLE (i.e., FOR SYSTEMS WHERE $\overline{CAS}$ HAS NOT BEEN DECODED WITH $\overline{RAS}$).

TYPICAL SUPPLY CURRENT WAVEFORMS

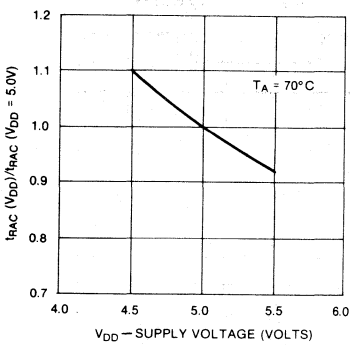


Typical power supply waveforms vs. time are shown for the $\overline{RAS}/\overline{CAS}$ timings of Read/Write, Read/Write (Long $\overline{RAS}/\overline{CAS}$), and $\overline{RAS}$-only refresh cycles. I_{DD} current transients at the $\overline{RAS}$ and $\overline{CAS}$ edges require adequate decoupling of these supplies.

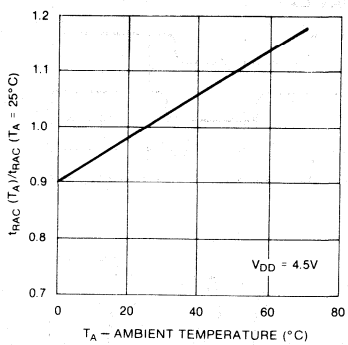
The effects of cycle time, V_{DD} supply voltage and ambient temperature on the I_{DD} current are shown in graphs included in the Typical Characteristics Section. Each family of curves for I_{DD1}, I_{DD2}, and I_{DD3} is related by a common point at $V_{DD} = 5.0V$ and $T_A = 25°C$ for two given t_{RAS} pulse widths. The typical I_{DD} current for a given condition of cycle time, V_{DD} and T_A can be determined by combining the effects of the appropriate family of curves.

TYPICAL CHARACTERISTICS

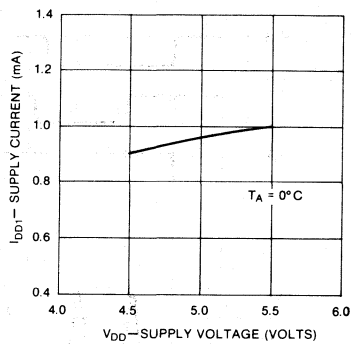
GRAPH 1
TYPICAL ACCESS TIME
t_{RAC} (NORMALIZED) VS. V_{DD}



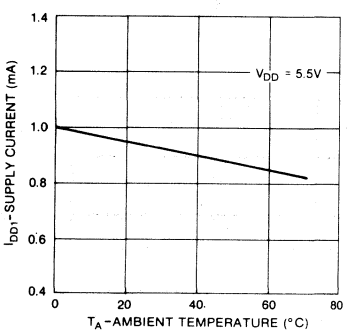
GRAPH 2
TYPICAL ACCESS TIME
t_{RAC} (NORMALIZED) VS.
AMBIENT TEMPERATURE



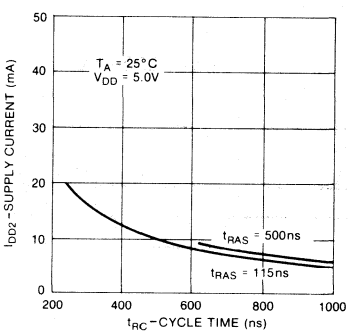
GRAPH 3
TYPICAL STANDBY CURRENT
I_{DD1} VS. V_{DD}



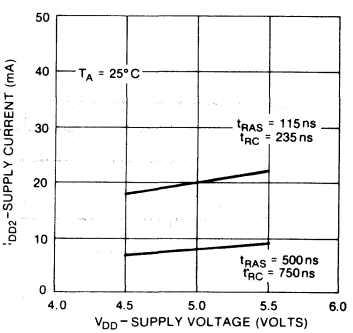
GRAPH 4
TYPICAL STANDBY CURRENT
I_{DD1} VS. AMBIENT TEMPERATURE



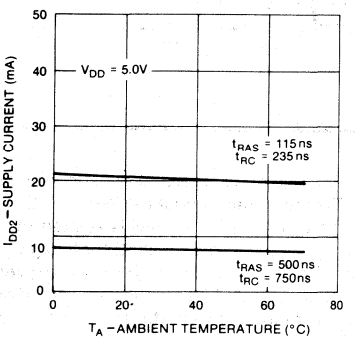
GRAPH 5
TYPICAL OPERATING CURRENT
I_{DD2} VS. t_{RC}



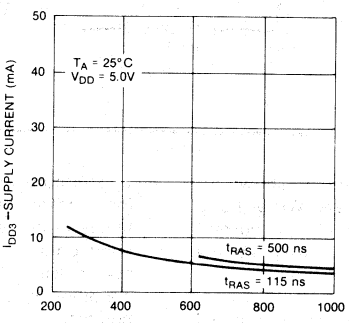
GRAPH 6
TYPICAL OPERATING CURRENT
I_{DD2} VS. V_{DD}



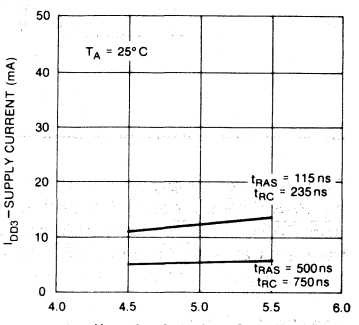
GRAPH 7
TYPICAL OPERATING CURRENT
I_{DD2} VS. AMBIENT TEMPERATURE



GRAPH 8
TYPICAL $\overline{RAS}$ ONLY
REFRESH CURRENT
I_{DD3} VS. t_{RC}

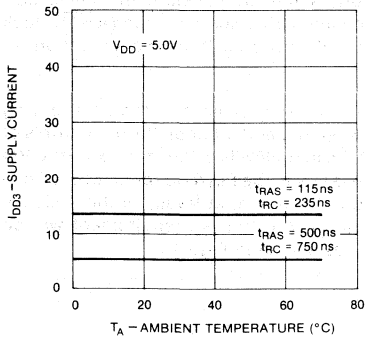


GRAPH 9
TYPICAL $\overline{RAS}$ ONLY
REFRESH CYCLE
I_{DD3} VS. V_{DD}

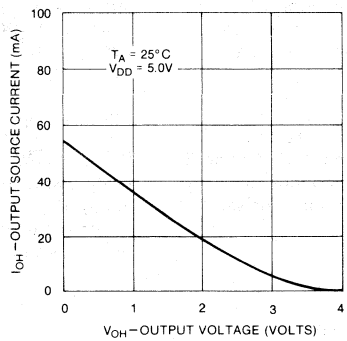


TYPICAL CHARACTERISTICS

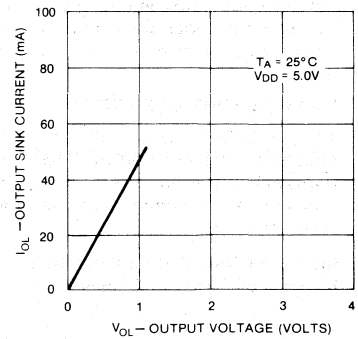
GRAPH 10
TYPICAL $\overline{RAS}$ ONLY
REFRESH CURRENT
I_{DD3} VS. AMBIENT TEMPERATURE



GRAPH 11
TYPICAL OUTPUT SOURCE CURRENT
I_{OH} VS. OUTPUT VOLTAGE V_{OH}



GRAPH 12
TYPICAL OUTPUT SINK CURRENT
I_{OL} VS. OUTPUT VOLTAGE V_{OL}



DEVICE DESCRIPTION

The Intel® 2118 is produced with HMOS, a high performance MOS technology which incorporates on chip substrate bias generation. This process, combined with new circuit design concepts, allows the 2118 to operate from a single +5V power supply, eliminating the +12V and -5V requirements. Pins 1 and 9 are not connected, which allows P.C.B. layout for future higher density memory generations.

The 2118 is functionally compatible with the industry standard 16-pin 16K dynamic RAMs, except for the power supply requirements. Replacing the +12V supply with a +5V supply and eliminating the -5V bias altogether, allows simple upgrade both in power and performance. To achieve total speed performance upgrade, however, the timing ciruitry must be modified to accommodate the higher performance.

READ CYCLE

A Read cycle is performed by maintaining Write Enable ($\overline{WE}$) high during a $\overline{RAS}/\overline{CAS}$ operation. The output pin of a selected device will remain in a high impedance state until valid data appears at the output at access time.

Device access time, t_{ACC}, is the longer of the two calculated intervals:

1. $t_{ACC} = t_{RAC}$ OR 2. $t_{ACC} = t_{RCD} + t_{CAC}$

Access time from $\overline{RAS}$, t_{RAC}, and access time from $\overline{CAS}$, t_{CAC}, are device parameters. Row to column address strobe delay time, t_{RCD}, are system dependent timing parameters. For example, substituting the device parameters of the 2118-3 yields:

3. $t_{ACC} = t_{RAC}$ = 100nsec for 25nsec $\leq t_{RCD} \leq$ 45nsec

OR

4. $t_{ACC} = t_{RCD} + t_{CAC} = t_{RCD}$ + 55nsec for $t_{RCD} >$ 45nsec

Note that if 25nsec $\leq t_{RCD} \leq$ 45nsec device access time is determined by equation 3 and is equal to t_{RAC}. If $t_{RCD} >$ 45nsec access time is determined by equation 4. This 20nsec interval (shown in the t_{RCD} inequality in equation 3) in which the falling edge of $\overline{CAS}$ can occur without affecting access time is provided to allow for system timing skew in the generation of $\overline{CAS}$.

REFRESH CYCLES

Each of the 128 rows of the 2118 must be refreshed every 2 milliseconds to maintain data. Any memory cycle:

1. Read Cycle
2. Write Cycle (Early Write, Delayed Write or Read-Modify-Write)
3. $\overline{RAS}$-only Cycle

refreshes the selected row as defined by the low order ($\overline{RAS}$) addresses. Any Write cycle, of course, may change the state of the selected cell. Using a Read, Write, or Read-Modify-Write cycle for refresh is not recommended for systems which utilize "wire-OR" outputs since output bus contention will occur.

A $\overline{RAS}$-only refresh cycle is the recommended technique for most applications to provide for data retention. A $\overline{RAS}$-only refresh cycle maintains the D_{OUT} in the high

impedance state with a typical power reduction of 30% over a Read or Write cycle.

$\overline{RAS}/\overline{CAS}$ TIMING

$\overline{RAS}$ and $\overline{CAS}$ have minimum pulse widths as defined by t_{RAS} and t_{CAS} respectively. These minimum pulse widths must be maintained for proper device operation and data integrity. A cycle, once begun by bringing $\overline{RAS}$ and/or $\overline{CAS}$ low must not be ended or aborted prior to fulfilling the minimum clock signal pulse width(s). A new cycle can not begin until the minimum precharge time, t_{RP}, has been met.

DATA OUTPUT OPERATION

The 2118 Data Output (D_{OUT}), which has three-state capability, is controlled by $\overline{CAS}$. During $\overline{CAS}$ high state ($\overline{CAS}$ at V_{IH}) the output is in the high impedance state. The following table summarizes the D_{OUT} state for various types of cycles.

Intel 2118 Data Output Operation for Various Types of Cycles

Type of Cycle	D_{OUT} State
Read Cycle	Data From Addressed Memory Cell
Early Write Cycle	HI-Z
$\overline{RAS}$-Only Refresh Cycle	HI-Z
$\overline{CAS}$-Only Cycle	HI-Z
Read/Modify/Write Cycle	Data From Addressed Memory Cell
Delayed Write Cycle	Indeterminate

HIDDEN REFRESH

An optional feature of the 2118 is that refresh cycles may be performed while maintaining valid data at the output pin. This feature is referred to as Hidden Refresh. Hidden Refresh is performed by holding $\overline{CAS}$ at V_{IL} and taking $\overline{RAS}$ high and after a specified precharge period (t_{RP}), executing a "$\overline{RAS}$-Only" refresh cycle, but with $\overline{CAS}$ held low (see Figure 1.)

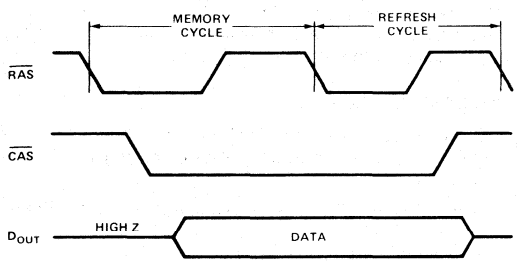


Figure 1. Hidden Refresh Cycle.

This feature allows a refresh cycle to be "hidden" among data cycles without affecting the data availability.

POWER ON

After the application of the V_{DD} supply, or after extended periods of bias (greater than 2ms) without clocks, the device must perform a minimum of eight (8) initialization cycles (any combination of cycles containing a $\overline{RAS}$ clock such as $\overline{RAS}$-only refresh) prior to normal operation.

The V_{DD} current (I_{DD}) requirement of the 2118 during power on is, however, dependent upon the input levels of $\overline{RAS}$ and $\overline{CAS}$. If the input levels of these clocks are at V_{IH} or V_{DD}, whichever is lower, the I_{DD} requirement per device is I_{DD1} (I_{DD} standby). If the input levels for these clocks are lower than V_{IH} or V_{DD} the I_{DD} requirement will be greater than I_{DD1}, as shown in Figure 2.

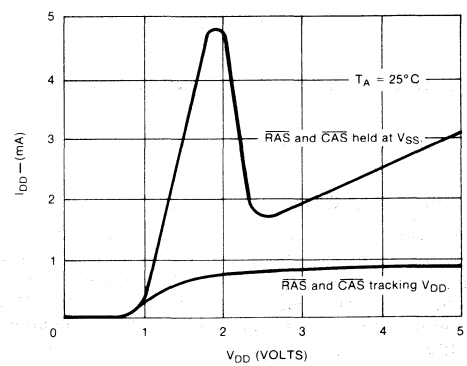


Figure 2. Typical I_{DD} VS V_{DD} during power up.

For large systems, this current requirement for I_{DD} could be substantially more than that for which the system has been designed. A system which has been designed, assuming the majority of devices to be operating in the refresh/standby mode, may produce sufficient I_{DD} loading such that the power supply may current limit. To assure that the system will not experience such loading during power on, a pullup resistor for each clock input to V_{DD} to maintain the non-selected current level (I_{DD1}) for the power supply is recommended.

2716*
16K (2K x 8) UV ERASABLE PROM

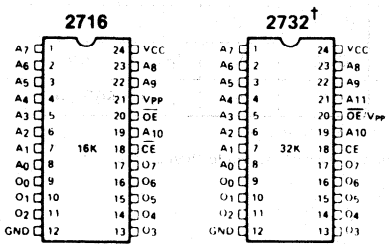
- **Fast Access Time**
  - **— 350 ns Max. 2716-1**
  - **— 390 ns Max. 2716-2**
  - **— 450 ns Max. 2716**
  - **— 490 ns Max. 2716-5**
  - **— 650 ns Max. 2716-6**
- **Single +5V Power Supply**
- **Low Power Dissipation**
  - **— 525 mW Max. Active Power**
  - **— 132 mW Max. Standby Power**
- **Pin Compatible to Intel® 2732 EPROM**
- **Simple Programming Requirements**
  - **— Single Location Programming**
  - **— Programs with One 50 ms Pulse**
- **Inputs and Outputs TTL Compatible during Read and Program**
- **Completely Static**

The Intel® 2716 is a 16,384-bit ultraviolet erasable and electrically programmable read-only memory (EPROM). The 2716 operates from a single 5-volt power supply, has a static standby mode, and features fast single address location programming. It makes designing with EPROMs faster, easier and more economical.

The 2716, with its single 5-volt supply and with an access time up to 350 ns, is ideal for use with the newer high performance +5V microprocessors such as Intel's 8085 and 8086. A selected 2716-5 and 2716-6 is available for slower speed applications. The 2716 is also the first EPROM with a static standby mode which reduces the power dissipation without increasing access time. The maximum active power dissipation is 525 mW while the maximum standby power dissipation is only 132 mW, a 75% savings.

The 2716 has the simplest and fastest method yet devised for programming EPROMs – single pulse TTL level programming. No need for high voltage pulsing because all programming controls are handled by TTL signals. Program any location at any time—either individually, sequentially or at random, with the 2716's single address location programming. Total programming time for all 16,384 bits is only 100 seconds.

PIN CONFIGURATION



†Refer to 2732 data sheet for specifications

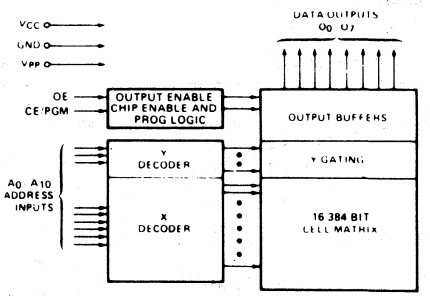
PIN NAMES

A_0– A_{10}	ADDRESSES
$\overline{CE}$/PGM	CHIP ENABLE/PROGRAM
$\overline{OE}$	OUTPUT ENABLE
O_0–O_7	OUTPUTS

MODE SELECTION

MODE \ PINS	$\overline{CE}$/PGM (18)	$\overline{OE}$ (20)	V_{PP} (21)	V_{CC} (24)	OUTPUTS (9-11, 13-17)
Read	V_{IL}	V_{IL}	+5	+5	D_{OUT}
Standby	V_{IH}	Don't Care	+5	+5	High Z
Program	Pulsed V_{IL} to V_{IH}	V_{IH}	+25	+5	D_{IN}
Program Verify	V_{IL}	V_{IL}	+25	+5	D_{OUT}
Program Inhibit	V_{IL}	V_{IH}	+25	+5	High Z

BLOCK DIAGRAM



PROGRAMMING

The programming specifications are described in the Data Catalog PROM/ROM Programming Instructions Section.

Absolute Maximum Ratings*

Temperature Under Bias . . . . . -10°C to +80°C
Storage Temperature . . . . . -65°C to +125°C
All Input or Output Voltages with Respect to Ground . . . . . +6V to -0.3V
V_{PP} Supply Voltage with Respect to Ground During Program . . . . . +26.5V to -0.3V

*COMMENT: 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 and AC Operating Conditions During Read

	2716	2716-1	2716-2	2716-5	2716-6
Temperature Range	0°C – 70°C	0°C – 70°C	0°C – 70°C	0°C – 70°C	0°C – 70°C
V_{CC} Power Supply [1,2]	5V ±5%	5V ±10%	5V ±5%	5V ±5%	5V ±5%
V_{PP} Power Supply [2]	V_{CC}	V_{CC}	V_{CC}	V_{CC}	V_{CC}

READ OPERATION

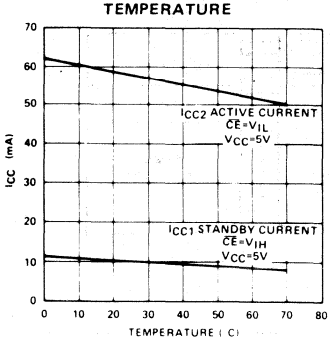
D.C. and Operating Characteristics

Symbol	Parameter	Limits			Unit	Conditions
		Min.	Typ. [3]	Max.		
I_{LI}	Input Load Current			10	μA	V_{IN} = 5.25V
I_{LO}	Output Leakage Current			10	μA	V_{OUT} = 5.25V
I_{PP1} [2]	V_{PP} Current			5	mA	V_{PP} = 5.25V
I_{CC1} [2]	V_{CC} Current (Standby)		10	25	mA	$\overline{CE}$ = V_{IH}, $\overline{OE}$ = V_{IL}
I_{CC2} [2]	V_{CC} Current (Active)		57	100	mA	$\overline{OE}$ = $\overline{CE}$ = V_{IL}
V_{IL}	Input Low Voltage	-0.1		0.8	V	
V_{IH}	Input High Voltage	2.0		V_{CC}+1	V	
V_{OL}	Output Low Voltage			0.45	V	I_{OL} = 2.1 mA
V_{OH}	Output High Voltage	2.4			V	I_{OH} = -400 μA

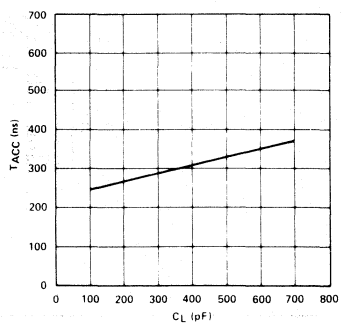
NOTES: 1. V_{CC} must be applied simultaneously or before V_{PP} and removed simultaneously or after V_{PP}.
2. V_{PP} may be connected directly to V_{CC} except during programming. The supply current would then be the sum of I_{CC} and I_{PP1}.
3. Typical values are for T_A = 25°C and nominal supply voltages.
4. This parameter is only sampled and is not 100% tested.

Typical Characteristics

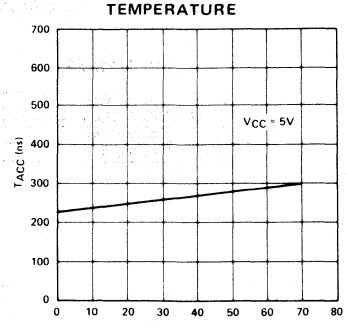
I_{CC} CURRENT vs. TEMPERATURE



ACCESS TIME vs. CAPACITANCE



ACCESS TIME vs. TEMPERATURE



A.C. Characteristics

Symbol	Parameter	Limits (ns)										Test Conditions
		2716		2716-1		2716-2		2716-5		2716-6		
		Min.	Max.	Min.	Max.	Min.	Max.	Min.	Max.	Min.	Max.	
t_{ACC}	Address to Output Delay		450		350		390		450		450	$\overline{CE} = \overline{OE} = V_{IL}$
t_{CE}	$\overline{CE}$ to Output Delay		450		350		390		490		650	$\overline{OE} = V_{IL}$
t_{OE}	Output Enable to Output Delay		120		120		120		160		200	$\overline{CE} = V_{IL}$
t_{DF}	Output Enable High to Output Float	0	100	0	100	0	100	0	100	0	100	$\overline{CE} = V_{IL}$
t_{OH}	Output Hold from Addresses, $\overline{CE}$ or $\overline{OE}$ Whichever Occurred First	0		0		0		0		0		$\overline{CE} = \overline{OE} = V_{IL}$

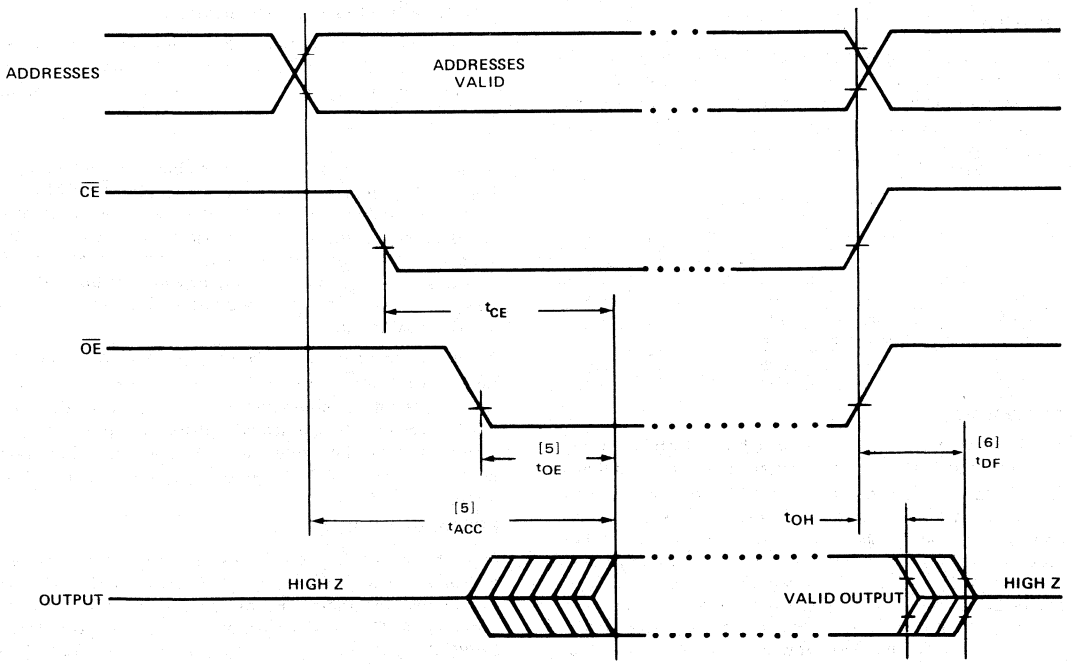
Capacitance [4] $T_A = 25°C$, f = 1 MHz

Symbol	Parameter	Typ.	Max.	Unit	Conditions
C_{IN}	Input Capacitance	4	6	pF	$V_{IN} = 0V$
C_{OUT}	Output Capacitance	8	12	pF	$V_{OUT} = 0V$

A.C. Test Conditions:

Output Load: 1 TTL gate and C_L = 100 pF
Input Rise and Fall Times: ≤20 ns
Input Pulse Levels: 0.8V to 2.2V
Timing Measurement Reference Level:
Inputs 1V and 2V
Outputs 0.8V and 2V

A. C. Waveforms [1]



NOTE:
1. V_{CC} must be applied simultaneously or before V_{PP} and removed simultaneously or after V_{PP}.
2. V_{PP} may be connected directly to V_{CC} except during programming. The supply current would then be the sum of I_{CC} and I_{PP1}.
3. Typical values are for $T_A = 25°C$ and nominal supply voltages.
4. This parameter is only sampled and is not 100% tested.
5. OE may be delayed up to $t_{ACC} - t_{OE}$ after the falling edge of CE without impact on t_{ACC}.
6. t_{DF} is specified from OE or CE, whichever occurs first.

ERASURE CHARACTERISTICS

The erasure characteristics of the 2716 are such that erasure begins to occur when exposed to light with wavelengths shorter than approximately 4000 Angstroms (Å). It should be noted that sunlight and certain types of fluorescent lamps have wavelengths in the 3000–4000Å range. Data show that constant exposure to room level fluorescent lighting could erase the typical 2716 in approximately 3 years, while it would take approximatley 1 week to cause erasure when exposed to direct sunlight. If the 2716 is to be exposed to these types of lighting conditions for extended periods of time, opaque labels are available from Intel which should be placed over the 2716 window to prevent unintentional erasure.

The recommended erasure procedure (see Data Catalog PROM/ROM Programming Instruction Section) for the 2716 is exposure to shortwave ultraviolet light which has a wavelength of 2537 Angstroms (Å). The integrated dose (i.e., UV intensity X exposure time) for erasure should be a minimum of 15 W-sec/cm^2. The erasure time with this dosage is approximately 15 to 20 minutes using an ultraviolet lamp with a 12000 μW/cm^2 power rating. The 2716 should be placed within 1 inch of the lamp tubes during erasure. Some lamps have a filter on their tubes which should be removed before erasure.

DEVICE OPERATION

The five modes of operation of the 2716 are listed in Table I. It should be noted that all inputs for the five modes are at TTL levels. The power supplies required are a +5V V_{CC} and a V_{PP}. The V_{PP} power supply must be at 25V during the three programming modes, and must be at 5V in the other two modes.

TABLE I. MODE SELECTION

MODE \ PINS	$\overline{CE}$/PGM (18)	$\overline{OE}$ (20)	V_{PP} (21)	V_{CC} (24)	OUTPUTS (9-11, 13-17)
Read	V_{IL}	V_{IL}	+5	+5	D_{OUT}
Standby	V_{IH}	Don't Care	+5	+5	High Z
Program	Pulsed V_{IL} to V_{IH}	V_{IH}	+25	+5	D_{IN}
Program Verify	V_{IL}	V_{IL}	+25	+5	D_{OUT}
Program Inhibit	V_{IL}	V_{IH}	+25	+5	High Z

READ MODE

The 2716 has two control functions, both of which must be logically satisfied in order to obtain data at the outputs. Chip Enable ($\overline{CE}$) is the power control and should be used for device selection. Output Enable ($\overline{OE}$) is the output control and should be used to gate data to the output pins, independent of device selection. Assuming that addresses are stable, address access time (t_{ACC}) is equal to the delay from $\overline{CE}$ to output (t_{CE}). Data is available at the outputs 120 ns (t_{OE}) after the falling edge of $\overline{OE}$, assuming that $\overline{CE}$ has been low and addresses have been stable for at least $t_{ACC} - t_{OE}$.

STANDBY MODE

The 2716 has a standby mode which reduces the active power dissipation by 75%, from 525 mW to 132 mW. The 2716 is placed in the standby mode by applying a TTL high signal to the $\overline{CE}$ input. When in standby mode, the outputs are in a high impedence state, independent of the $\overline{OE}$ input.

OUTPUT OR-TIEING

Because 2716's are usually used in larger memory arrays, Intel has provided a 2 line control function that accomodates this use of multiple memory connections. The two line control function allows for:

a) the lowest possible memory power dissipation, and
b) complete assurance that output bus contention will not occur.

To most efficiently use these two control lines, it is recommended that $\overline{CE}$ (pin 18) be decoded and used as the primary device selecting function, while $\overline{OE}$ (pin 20) be made a common connection to all devices in the array and connected to the READ line from the system control bus. This assures that all deselected memory devices are in their low power standby mode and that the output pins are only active when data is desired from a particular memory device.

PROGRAMMING

Initially, and after each erasure, all bits of the 2716 are in the "1" state. Data is introduced by selectively programming "0's" into the desired bit locations. Although only "0's" will be programmed, both "1's" and "0's" can be presented in the data word. The only way to change a "0" to a "1" is by ultraviolet light erasure.

The 2716 is in the programming mode when the V_{PP} power supply is at 25V and $\overline{OE}$ is at V_{IH}. The data to be programmed is applied 8 bits in parallel to the data output pins. The levels required for the address and data inputs are TTL.

When the address and data are stable, a 50 msec, active high, TTL program pulse is applied to the $\overline{CE}$/PGM input. A program pulse must be applied at each address location to be programmed. You can program any location at any time — either individually, sequentially, or at random. The program pulse has a maximum width of 55 msec. The 2716 must not be programmed with a DC signal applied to the $\overline{CE}$/PGM input.

Programming of multiple 2716s in parallel with the same data can be easily accomplished due to the simplicity of the programming requirements. Like inputs of the paralleled 2716s may be connected together when they are programmed with the same data. A high level TTL pulse applied to the $\overline{CE}$/PGM input programs the paralleled 2716s.

PROGRAM INHIBIT

Programming of multiple 2716s in parallel with different data is also easily accomplished. Except for $\overline{CE}$/PGM, all like inputs (including $\overline{OE}$) of the parallel 2716s may be common. A TTL level program pulse applied to a 2716's $\overline{CE}$/PGM input with V_{PP} at 25V will program that 2716. A low level $\overline{CE}$/PGM input inhibits the other 2716 from being programmed.

PROGRAM VERIFY

A verify should be performed on the programmed bits to determine that they were correctly programmed. The verify may be performed wth V_{PP} at 25V. Except during programming and program verify, V_{PP} must be at 5V.

2732A
32K (4K x 8) UV ERASABLE PROM

- **200ns (2732A-2) Maximum Access Time . . . HMOS*-E Technology**
- **Compatible to High Speed 8mHz 8086-2 MPU . . .Zero WAIT State**
- **Two Line Control**
- **Pin Compatible to 2764 EPROM**
- **Industry Standard Pinout . . . JEDEC Approved**
- **Low Standby Current . . . 35mA Max.**

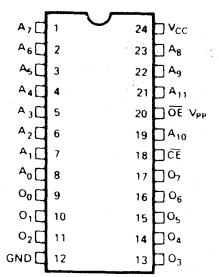
The Intel 2732A is a 5V only, 32,768 bit ultraviolet erasable and electrically programmable read-only memory (EPROM). It is pin compatible to Intel's 450ns 2732. The standard 2732A's access time is 250ns with speed selection (2732A-2) available at 200ns. The access time is compatible to high performance microprocessors, such as the 8mHz 8086-2. In these systems, the 2732A allows the microprocessor to operate without the addition of WAIT states.

An important 2732A feature is the separate output control, Output Enable ($\overline{OE}$), from the Chip Enable control ($\overline{CE}$). The $\overline{OE}$ control eliminates bus contention in multiple bus microprocessor systems. Intel's Application Note AP-72 describes the microprocessor system implementation of the $\overline{OE}$ and $\overline{CE}$ controls on Intel's EPROMs. AP-72 is available from Intel's Literature Department.

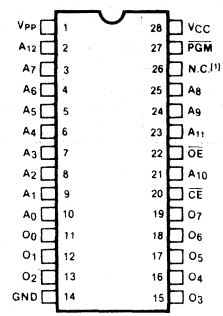
The 2732A has a standby mode which reduces the power dissipation without increasing access time. The maximum active current is 150mA, while the maximum standby current is only 35mA, a 75% saving. The standby mode is achieved by applying a TTL-high signal to the $\overline{CE}$ input.

The 2732A is fabricated with HMOS*-E technology, Intel's high speed N-channel MOS Silicon Gate Technology.

2732A PIN CONFIGURATION



2764 PIN CONFIGURATION



[1]For total compatibility from 2732A provide a trace to pin 26

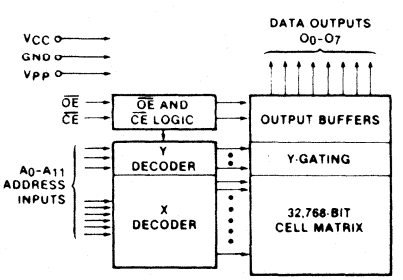
MODE SELECTION

MODE \ PINS	$\overline{CE}$ (18)	$\overline{OE}/V_{PP}$ (20)	V_{CC} (24)	OUTPUTS (9-11,13-17)
Read	V_{IL}	V_{IL}	+5	D_{OUT}
Standby	V_{IH}	Don't Care	+5	High Z
Program	V_{IL}	V_{PP}	+5	D_{IN}
Program Verify	V_{IL}	V_{IL}	+5	D_{OUT}
Program Inhibit	V_{IH}	V_{PP}	+5	High Z

PIN NAMES

A_0-A_{11}	ADDRESSES
$\overline{CE}$	CHIP ENABLE
$\overline{OE}$	OUTPUT ENABLE
O_0-O_7	OUTPUTS

BLOCK DIAGRAM



*HMOS is a patented process of Intel Corporation.

PROGRAMMING

The programming specifications are described in the Data Catalog PROM/ROM Programming Instructions Section.

Absolute Maximum Ratings*

Temperature Under Bias ..... −10°C to +80°C
Storage Temperature ..... −65°C to +125°C
All Input or Output Voltages with
Respect to Ground ..... +6V to −0.3V
V_{PP} Supply Voltage with Respect to Ground
During Programming ..... +22V to −0.3V

*COMMENT: 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 and AC Operating Conditions During Read

	2732A	2732A-2	2732A-3
Operating Temperature Range	0°C — 70°C	0°C — 70°C	0°C — 70°C
V_{CC} Power Supply	5V ± 5%	5V ± 5%	5V ± 5%

READ OPERATION

D.C. and Operating Characteristics

Symbol	Parameter	Limits			Unit	Conditions
		Min.	Typ.[1]	Max.		
I_{IL}	Input Load Current			10	μA	$V_{IN} = 5.25V$
I_{LO}	Output Leakage Current			10	μA	$V_{OUT} = 5.25V$
I_{CC1}	V_{CC} Current (Standby)			35	mA	$\overline{CE} = V_{IH}$, $\overline{OE} = V_{IL}$
I_{CC2}	V_{CC} Current (Active)			150	mA	$\overline{OE} = \overline{CE} = V_{IL}$
V_{IL}	Input Low Voltage	−0.1		0.8	V	
V_{IH}	Input High Voltage	2.0		$V_{CC}+1$	V	
V_{OL}	Output Low Voltage			0.45	V	$I_{OL} = 2.1mA$
V_{OH}	Output High Voltage	2.4			V	$I_{OH} = -400\mu A$

NOTES: 1. Typical values are for $T_A = 25°C$ and nominal supply voltages.

A.C. Characteristics

Symbol	Parameter	2732A Limits			2732A-2 Limits			2732A-3 Limits			Unit	Test Conditions
		Min	Typ[1]	Max	Min	Typ[1]	Max	Min	Typ[1]	Max		
t_{ACC}	Address to Output Delay			250			200			300	ns	$\overline{CE} = \overline{OE} = V_{IL}$
t_{CE}	$\overline{CE}$ to Output Delay			250			200			300	ns	$\overline{OE} = V_{IL}$
t_{OE}	Output Enable to Output Delay	10		100	10		70	10		150	ns	$\overline{CE} = V_{IL}$
t_{DF}	Output Enable High to Output Float	0		90	0		60	0		130	ns	$\overline{CE} = V_{IL}$
t_{OH}	Output Hold from Addresses, $\overline{CE}$ or $\overline{OE}$ Whichever Occurred First	0			0			0			ns	$\overline{CE} = \overline{OE} = V_{IL}$

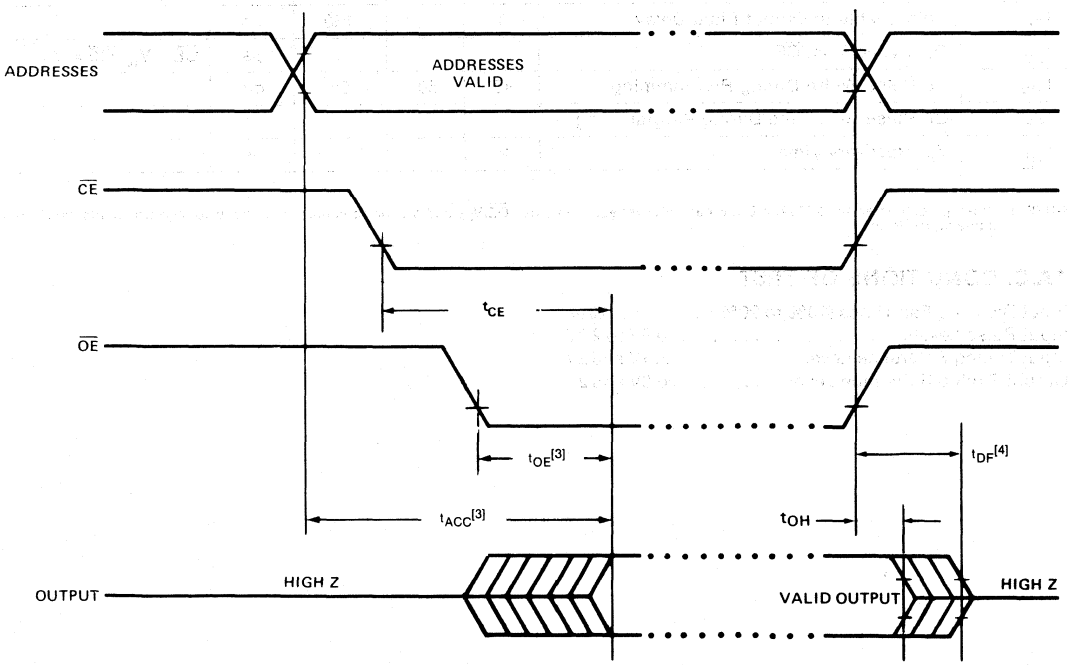
CAPACITANCE [1] $T_A = 25°C$, f = 1MHz

Symbol	Parameter	Typ.	Max.	Unit	Conditions
C_{IN1}	Input Capacitance Except $\overline{OE}/V_{PP}$	4	6	pF	$V_{IN} = 0V$
C_{IN2}	$\overline{OE}/V_{PP}$ Input Capacitance		20	pF	$V_{IN} = 0V$
C_{OUT}	Output Capacitance		12	pF	$V_{OUT} = 0V$

A.C. TEST CONDITIONS

Output Load: 1 TTL gate and C_L = 100pF
Input Rise and Fall Times: ≤ 20ns
Input Pulse Levels: 0.8V to 2.2V
Timing Measurement Reference Level:
Inputs 1V and 2V
Outputs 0.8V and 2V

A. C. Waveforms



NOTE: 1. Typical values are for $T_A = 25°C$ and nominal supply voltages.
2. This parameter is only sampled and is not 100% tested.
3. $\overline{OE}$ may be delayed up to $t_{ACC} - t_{OE}$ after the falling edge of $\overline{CE}$ without impact on t_{ACC}.
4. t_{DF} is specified from $\overline{OE}$ or $\overline{CE}$, whichever occurs first.

PROGRAMMING[1]

D.C. PROGRAMMING CHARACTERISTICS: $T_A = 25 \pm 5°C$, $V_{CC} = 5V \pm 5\%$, $V_{PP} = 21V \pm 0.5V$

Symbol	Parameter	Limits			Unit	Test Conditions
		Min.	Typ.	Max.		
I_{LI}	Input Current (All Inputs)			10	μA	$V_{IN} = V_{IL}$ or V_{IH}
V_{OL}	Output Low Voltage During Verify			0.45	V	$I_{OL} = 2.1$ mA
V_{OH}	Output High Voltage During Verify	2.4			V	$I_{OH} = -400$ μA
I_{CC}	V_{CC} Supply Current		85	150	mA	
V_{IL}	Input Low Level (All Inputs)	−0.1		0.8	V	
V_{IH}	Input High Level (All Inputs Except OE/V_{PP})	2.0		$V_{CC}+1$	V	
I_{PP}	V_{PP} Supply Current			30	mA	$\overline{CE} = V_{IL}$, $\overline{OE} = V_{PP}$

A.C. PROGRAMMING CHARACTERISTICS: $T_A = 25 \pm 5°C$, $V_{CC} = 5V \pm 5\%$, $V_{PP} = 21V \pm 0.5V$

Symbol	Parameter	Limits			Unit	Test Conditions*
		Min.	Typ.	Max.		
t_{AS}	Address Setup Time	2			μs	
t_{OES}	$\overline{OE}$ Setup Time	2			μs	
t_{DS}	Data Setup Time	2			μs	
t_{AH}	Address Hold Time	0			μs	
t_{OEH}	OE Hold Time	2			μs	
t_{DH}	Data Hold Time	2			μs	
t_{DF}	Chip Enable to Output Float Delay	0		130	ns	
t_{DV}	Data Valid from $\overline{CE}$			1	μs	$\overline{CE} = V_{IL}$, $\overline{OE} = V_{IL}$
t_{PW}	$\overline{CE}$ Pulse Width During Programming	45	50	55	ms	
t_{PRT}	$\overline{OE}$ Pulse Rise Time During Programming	50			ns	
t_{VR}	V_{PP} Recovery Time	2			μs	

NOTE: 1. When programming the 2723A, a 0.1μF capacitor is required across $\overline{OE}/V_{PP}$ and ground to suppress spurious voltage transients which may damage the device.

*A.C. CONDITIONS OF TEST

Input Rise and Fall Times (10% to 90%) 20ns
Input Pulse Levels ..... 0.8V to 2.2V
Input Timing Reference Level ..... 1V and 2V
Output Timing Reference Level ..... 0.8V and 2V

ERASURE CHARACTERISTICS

The erasure characteristics of the 2732A are such that erasure begins to occur when exposed to light with wavelengths shorter than approximately 4000 Angstroms (Å). It should be noted that sunlight and certain types of fluorescent lamps have wavelengths in the 3000-4000Å range. Data show that constant exposure to room level fluorescent lighting could erase the typical 2732A in approximately 3 years, while it would take approximately 1 week to cause erasure when exposed to direct sunlight. If the 2732A is to be exposed to these types of lighting conditions for extended periods of time, opaque labels are available from Intel which should be placed over the 2732A window to prevent unintentional erasure.

The recommended erasure procedure for the 2732A is exposure to shortwave ultraviolet light which has a wavelength of 2537 Angstroms (Å). The integrated dose (i.e., UV intensity X exposure time) for erasure should be a minimum of 15 W-sec/cm^2. The erasure time with this dosage is approximately 15 to 20 minutes using an ultraviolet lamp with 12000μW/cm^2 power rating. The 2732A should be placed within 1 inch of the lamp tubes during erasure. Some lamps have a filter on their tubes which should be removed before erasure.

DEVICE OPERATION

The five modes of operation of the 2732A are listed in Table 1. A single 5V power supply is required in the read mode. All inputs are TTL levels except for $\overline{OE}/V_{PP}$ during programming. In the program mode the $\overline{OE}/V_{PP}$ input is pulsed from a TTL level to 21V.

TABLE 1. Mode Selection

MODE \ PINS	CE (18)	OE/V_{PP} (20)	V_{CC} (24)	OUTPUTS (9-11,13-17)
Read	V_{IL}	V_{IL}	+5	D_{OUT}
Standby	V_{IH}	Don't Care	+5	High Z
Program	V_{IL}	V_{PP}	+5	D_{IN}
Program Verify	V_{IL}	V_{IL}	+5	D_{OUT}
Program Inhibit	V_{IH}	V_{PP}	+5	High Z

Read Mode

The 2732A has two control functions, both of which must be logically satisfied in order to obtain data at the outputs. Chip Enable ($\overline{CE}$) is the power control and should be used for device selection. Output Enable ($\overline{OE}$) is the output control and should be used to gate data to the output pins, independent of device selection. Assuming that addresses are stable, address access time (t_{ACC}) is equal to the delay from $\overline{CE}$ to output (t_{CE}). Data is available at the outputs after the falling edge of $\overline{OE}$, assuming that $\overline{CE}$ has been low and addresses have been stable for at least $t_{ACC} - t_{OE}$.

Standby Mode

The 2732A has a standby mode which reduces the active power current by 75%, from 150mA to 35mA. The 2732A is placed in the standby mode by applying a TTL high signal to the $\overline{CE}$ input. When in standby mode, the outputs are in a high impedance state, independent of the $\overline{OE}$ input.

Output OR-Tieing

Because EPROMs are usually used in larger memory arrays, Intel has provided a 2 line control function that accomodates this use of multiple memory connection. The two line control function allows for:

a) the lowest possible memory power dissipation, and
b) complete assurance that output bus contention will not occur.

To most efficiently use these two control lines, it is recommended that $\overline{CE}$ (pin 18) be decoded and used as the primary device selecting function, while $\overline{OE}$ (pin 20) be made a common connection to all devices in the array and connected to the READ line from the system control bus. This assures that all deselected memory devices are in their low power standby mode and that the output pins are only active when data is desired from a particular memory device.

PROGRAMMING (See Programming Instruction Section for Waveforms.)

Programming is the same as Intel's 450ns 2732 except for the programming voltage. In the program mode the 2732A $\overline{OE}/V_{PP}$ input is pulsed from a TTL low level to 21V (25V for the 2732). ***Exceeding 21.5V will damage the 2732A.***

Initially, and after each erasure, all bits of the 2732A are in the "1" state. Data is introduced by selectively programming "0's" into the desired bit locations. Although only "0's" will be programmed, both "1's" and "0's" can be present in the data word. The only way to change a "0" to a "1" is by ultraviolet light erasure.

The 2732A is in the programming mode when the $\overline{OE}/V_{PP}$ input is at 21V. It is required that a 0.1μF capacitor be placed across $\overline{OE}/V_{PP}$ and ground to suppress spurious voltage transients which may damage the device. The data to be programmed is applied 8 bits in parallel to the data output pins. The levels required for the address and data inputs are TTL.

When the address and data are stable, a 50msec, active low, TTL program pulse is applied to the $\overline{CE}$ input. A program pulse must be applied at each address location to be programmed. You can program any location at any time — either individually, sequentially, or at random. The program pulse has a maximum width of 55msec. The 2732A must not be programmed with a DC signal applied to the $\overline{CE}$ input.

Programming of mulitple 2732As in parallel with the same data can be easily accomplished due to the simplicity of the programming requirements. Like inputs of the paralleled 2732As may be connected together when they are programmed with the same data. A low level TTL pulse applied to the $\overline{CE}$ input programs the paralleled 2732As.

Program Inhibit

Programming of mulitple 2732As in parallel with different data is also easily accomplished. Except for $\overline{CE}$, all like inputs (including $\overline{OE}$) of the parallel 2732As may be common. A TTL level program pulse applied to a 2732A's $\overline{CE}$ input with $\overline{OE}/V_{PP}$ at 21V will program that 2732A. A high level $\overline{CE}$ input inhibits the other 2732As from being programmed.

Program Verify

A verify should be performed on the programmed bits to determine that they were correctly programmed. The verify is accomplished with $\overline{OE}/V_{PP}$ and $\overline{CE}$ at V_{IL}. Data should be verified t_{DV} after the falling edge of $\overline{CE}$.

2816
16K (2K x 8) ELECTRICALLY ERASABLE PROM

- **HMOS*-E FLOTOX Cell Design**
- **Reliable Floating Gate Technology**
- **Very Fast Access Time:**
  —2816, 250 ns Max.
  —2816-3, 350 ns Max.
  —2816-4, 450 ns Max.
- **Single Byte Erase/Write Capability**
- **10 ms Byte Erase/Write Time**
- **Chip Erase Time of 10 ms**
- **Conforms to JEDEC Byte-Wide Family Standard**
- **Microprocessor Compatible Architecture**
- **Low Power Dissipation:**
  —Active Current, 110 mA Max.
  —Standby Current, 50 mA Max.
- **Erase/Write Specifications Guaranteed 0-70°C**

The Intel® 2816 is a 16,384 bit electrically erasable programmable read-only memory (E^2PROM). The 2816 can be easily erased and reprogrammed on a byte basis. A chip erase function is also provided. The device operates from a 5-volt power supply in the read mode; writing and erasing are accomplished by providing a single 21-volt pulse.

The 2816, with its very fast read access speed, is compatible with high performance microprocessors such as the 8086-2. Using the fast access speed allows zero wait operation in large system configurations.

The electrical erase/write capability of the 2816 makes it ideal for a wide variety of applications requiring in-system, non-volatile erase and write. Never before has in-system alterability been possible with this combination of density, performance and flexibility. Any byte can be erased or written in 10 ms without affecting the data in any other byte. Alternatively, the entire memory can be erased in 10 ms allowing the total time to rewrite all 2K bytes to be cut by 50%. The 2816 provides a significant increase in flexibility allowing new applications (dynamic reconfiguration, continuous calibration) never before possible.

The 2816 E^2PROM possesses Intel's 2-line control architecture to eliminate bus contention in a system environment. A power down mode is also featured; in the standby mode power consumption is reduced by over 55% without increasing access time. The standby mode is achieved by applying a TTL-high signal to the CE input.

Byte erase and write are controlled entirely by TTL signal levels, yet require no control signals beyond $\overline{CE}$ and $\overline{OE}$. For byte write a selected chip ($\overline{CE}$ = TTL low) senses the 21V V_{PP} pulse and automatically goes into write mode. Byte erase mode is identical to byte write except that data-in must be all logic ones (TTL-high). Never before has an in-system alteration of non-volatile information been implemented with such simple control.

*HMOS-E is a patented process of Intel Corporation.

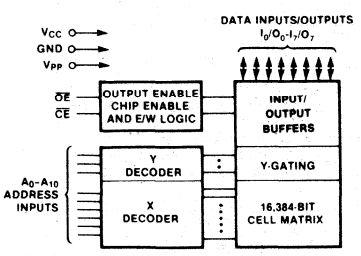


Figure 1. 2816 Functional Block Diagram

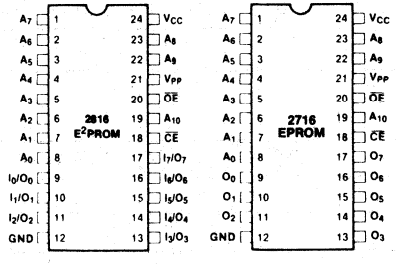


Figure 2. Pin Configuration

PIN NAMES

A_0-A_{10}	ADDRESSES
$\overline{CE}$	CHIP ENABLE
$\overline{OE}$	OUTPUT ENABLE
O_0-O_7	DATA OUTPUTS
I_0-I_7	DATA INPUTS
V_{PP}	PROGRAM VOLTAGE

APRIL 1981
AFN-01635B

DEVICE OPERATION

The 2816 has six modes of operation, listed in Table 1. All operational modes are designed to provide maximum microprocessor compatibility and system consistency. The device pinout is a part of Intel's JEDEC approved byte wide Non-Volatile Memory family, allowing appropriate and cost-effective density and functionality upgrades.

All control inputs are TTL compatible with the exception of chip erase. The V_{PP} voltage must be pulsed to 21 volts during write and erase, and held to 4 to 6 volts during the other two modes.

Table 1. Mode Selection $V_{CC}=+5V$

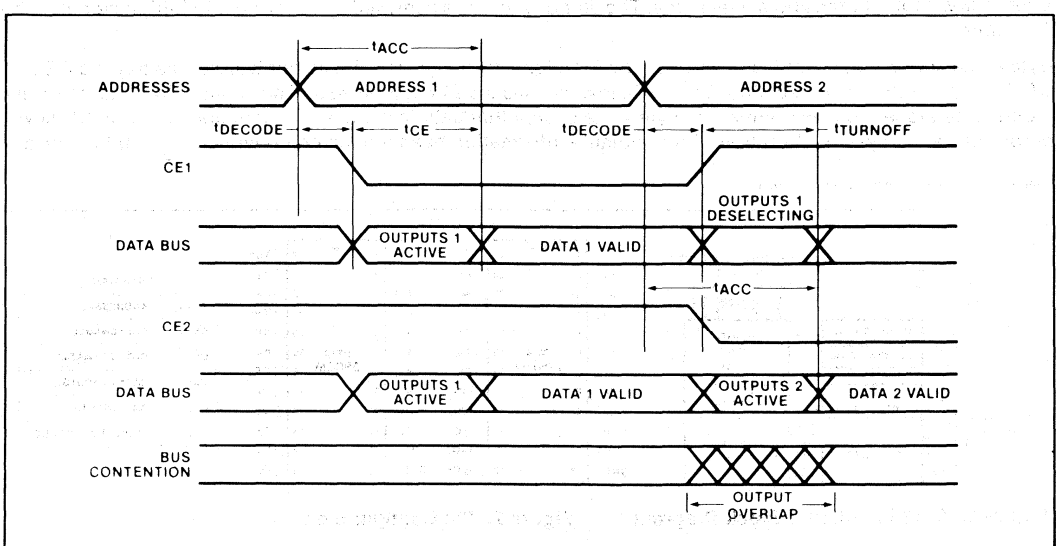
PIN / MODE	$\overline{CE}$ (18)	$\overline{OE}$ (20)	V_{PP} (21)	INPUTS/ OUTPUTS
READ	V_{IL}	V_{IL}	+4 to +6	D_{OUT}
STANDBY	V_{IH}	DON'T CARE	+4 to +6	HIGH Z
BYTE ERASE	V_{IL}	V_{IH}	+21	$D_{IN}=V_{IH}$
BYTE WRITE	V_{IL}	V_{IH}	+21	D_{IN}
CHIP ERASE	V_{IL}	+9 to +15V	+21	[10] $D_{IN}=V_{IH}$
E/W INHIBIT	V_{IH}	DON'T CARE	+4 to +22V	HIGH Z

Read Mode

Optimal system efficiency depends to a great extent on a tightly coupled microprocessor/memory interface. The E^2PROM device should respond rapidly with data to allow the highest possible CPU performance. The 2816 satisfies this high performance requirement because of access times typically less than 250 ns. Program execution directly out of electrically erasable memory has never before been possible; the 2816 opens this new, powerful applications segment.

The 2816 uses Intel's proven 2-line control architecture for read operation. Figure 3 shows the timing disadvantages of a single-line control architecture. 2-line control, shown in Figure 4, has been developed by Intel to solve this bus contention and the associated system reliability problems. Both $\overline{CE}$ and $\overline{OE}$ must be at logic low levels to obtain information from the device. Chip enable ($\overline{CE}$) is the power control pin and should be used for device selection. The output enable ($\overline{OE}$) pin serves to gate internal data to the output pins. Assuming that the address inputs are stable, address access time (t_{ACC}) is equal to the delay from $\overline{CE}$ to output (t_{CE}). Data is available at the outputs after a time delay of t_{OE}, assuming that $\overline{CE}$ has been low and addresses have been stable for at least t_{ACC}-t_{OE}.

Figure 5 shows a typical system interconnection. Here the 2816 contains program information that the 8086 requires for system function.



For footnotes see page 13.

Figure 3. Single-Line Control and Bus Contention

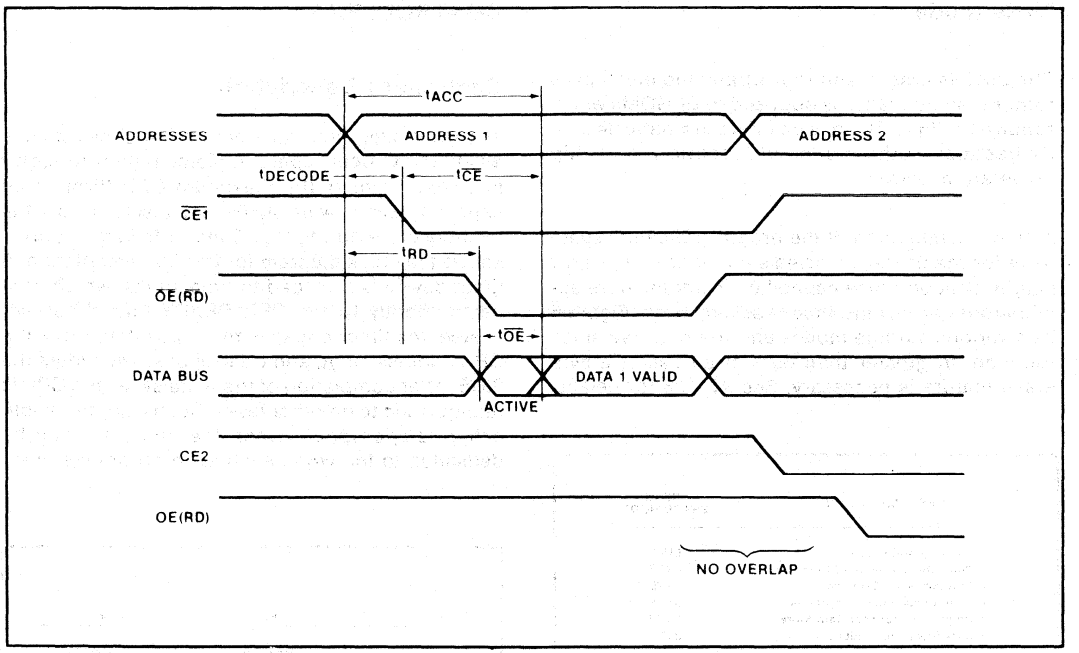


Figure 4. Two-Line Control Architecture

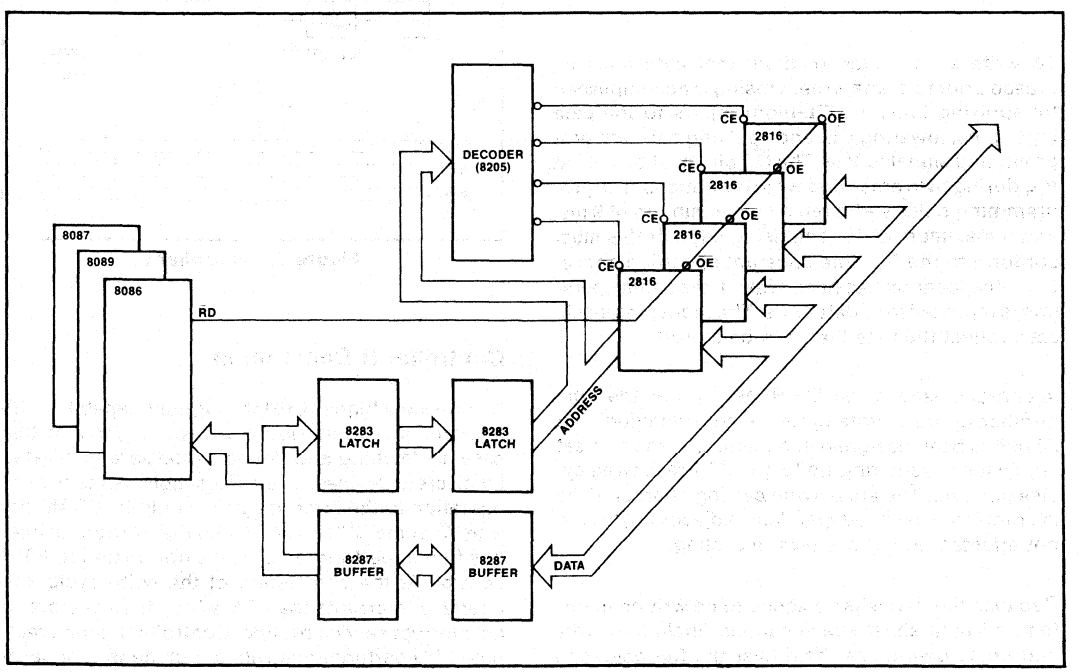


Figure 5. iAPX 86/2816 Read Architecture

For footnotes see page 13.

Write Mode

The 2816 is erased and reprogrammed electrically rather than optically, as opposed to EPROMs which require UV light. The device offers dramatic flexibility because both byte (single location) and chip erase are possible.

A close examination of the broad application spectrum for the E^2 device reveals an inherent need for single location erase capability. Program store applications can be classified in several ways. Figure 6 lists various storage modes and the required erase function. In greater than 80% of all cases, a byte erase feature is necessary. See AP-106 for details.

APPLICATION TYPE	IDEAL ERASE MODE
• Strict Program Store	CHIP
• Relocatable Program Structures	BYTE
• Program Store Extension	BYTE
• Program Execution Constants	BYTE
• Program Dependent Data Store	BYTE
• Data Store Applications	BYTE

Figure 6. Microprocessor Storage Types

To write a particular location, that byte must be erased prior to a data write. Erasing is accomplished by applying logic 1 (TTL-high) inputs to the data input pins, lowering $\overline{CE}$, and applying a 21-volt programming signal to V_{PP}. The $\overline{OE}$ pin must be held at V_{IH} during byte erase and write operations. The programming pulse width must be a minimum of 9 ms, and a maximum of 15. The rising edge of V_{PP} must conform to the RC time constant specified above. Once the location has been erased, the same operation is repeated for a data write. The input pins in this case reflect the byte that is to be stored.

A characteristic of all E^2PROMs is that the total number of erase/write cycles is not unlimited. The 2816 has been designed and manufactured to meet applications requiring up to 1×10^4 erase/write cycles per byte. The erase/write cycling characteristic is completely byte independent. Adjacent bytes are not affected during erase/write cycling.

Because the device is designed to be written in system, all data sheet specifications (including write and erase operations) hold over the full operating temperature range (0-70°C).

For footnotes see page 13.

CONTROLLERS

Controller I Description

The Controller I interface provides the lowest cost, smallest P.C. board space implementation, though it is unable to offer the maximum CPU throughput capability since wait states are inserted into the memory cycle during the 10 ms write time. Figure 7 shows the block diagram for this implementation. A timer device is provided to time 10 ms, which connects directly to the CPU READY line. When activated, the timer engages the V_{PP} switch, locks the CPU address, data, and control bus, and writes the 2816. After completion of the write cycle, the CPU is relinquished to do other tasks. Such a control application is appropriate when the processor can be dedicated to the write, such as in program store.

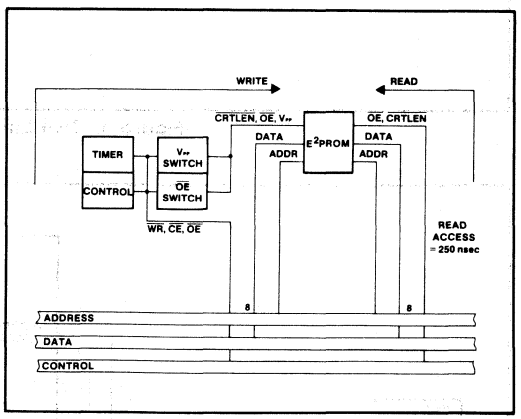


Figure 7. Controller I

Controller II Description

To provide a higher CPU throughput capability, the interface shown in Figure 8 was designed. In this case, all latching and timing signals are generated by discrete devices. The CPU simply sends a write operation to the interface as it would to a RAM device. After the CPU has engaged the write sequence, it is free to perform other tasks not related to 2816 control. At the completion of the write cycle, the interface interrupts the CPU which then vectors to an interrupt service routine. Controller II offers real-time CPU performance with a high degree of hardware overhead.

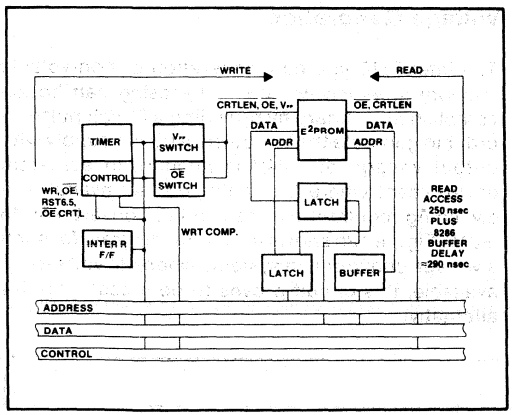


Figure 8. Controller II

Controller III Description

The Controller III implementation was designed to provide the real-time processing capability of Controller II, without the large hardware overhead. See Figure 9. In this design an Intel 8155 I/O port timer device is used to advantage. The ports provide the latching of data and address during the write cycle, while the timer performs accurate pulsing of the V_{PP} for the required duration. Much of the hardware has been reduced through the 8155. The interrupt structure of Controller II is used as well. Read access is very fast despite a multiplexer and a buffer delay.

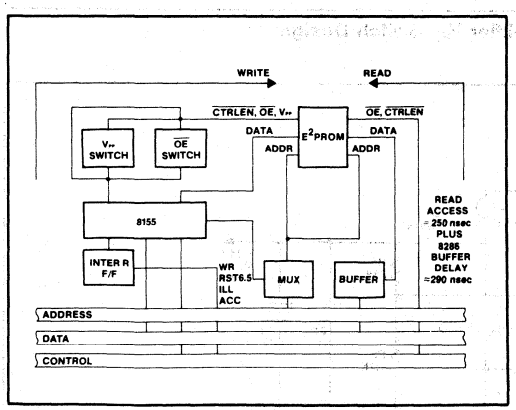


Figure 9. Controller III

Controller IV Description

Data store applications were in mind for the Controller IV design shown in Figure 10. In this case, read access was not a concern, though write erase access and hardware overhead were exceptionally important. This controller takes the 2816 completely off-line for both read and write operations. The write cycle is accomplished in the same way as in Controller III. Reading, however, is accomplished through several I/O operations.

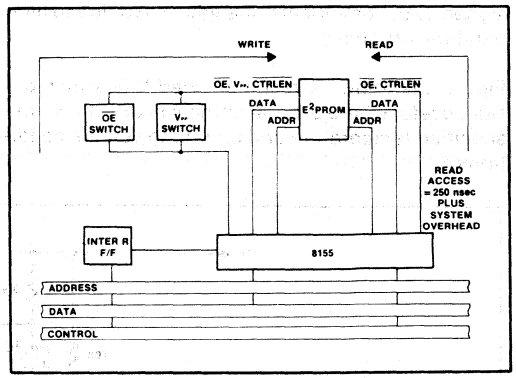


Figure 10. Controller IV

Chip Erase Mode

Should one wish to erase the entire 2816 array at once, the device offers a chip erase function. When the chip erase function is performed all 2K bytes are returned to a logic 1 (FF) state.

The 2816's chip erase function is engaged when the output enable ($\overline{OE}$) pin is raised above 9 volts. When $\overline{OE}$ is greater than 9 volts and $\overline{CE}$ and V_{PP} are in the normal write mode, the entire array is erased. This chip erase function takes approximately 10 ms. The data input pins must be held to a TTL high level during this time. Figure 11 is a recommended $\overline{OE}$ control switch.

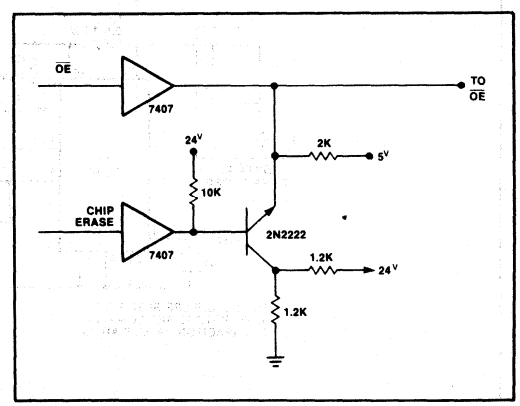


Figure 11. $\overline{OE}$ Chip Erase Control

For footnotes see page 13.

V_{PP} Pulse

The shape of the V_{PP} pulse is important in ensuring long term reliability and operating characteristics. V_{PP} must rise to 21V through an RC waveform (exponential). The T_{PRC} specification has been designed to accommodate changes of RC due to temperature variations.

Figure 12a shows a recommended V_{PP} switch design, useful where programming will occur over the specified temperature and operating voltage conditions.

Voltage Generation

The Intel 2816 is a new generation of non-volatile memory in which writing and erasing can be accomplished on board by providing a 21 volt pulse. In order to generate the V_{PP} pulse, a power supply with output voltage of +24V is needed. In a system environment where this voltage is not available, a switching regulator can be used to convert +5V to +24V. Figure 12b shows the circuit diagram for such a voltage converter. In systems where 24 volts is not available, this circuit proves to be a cost effective alternative.

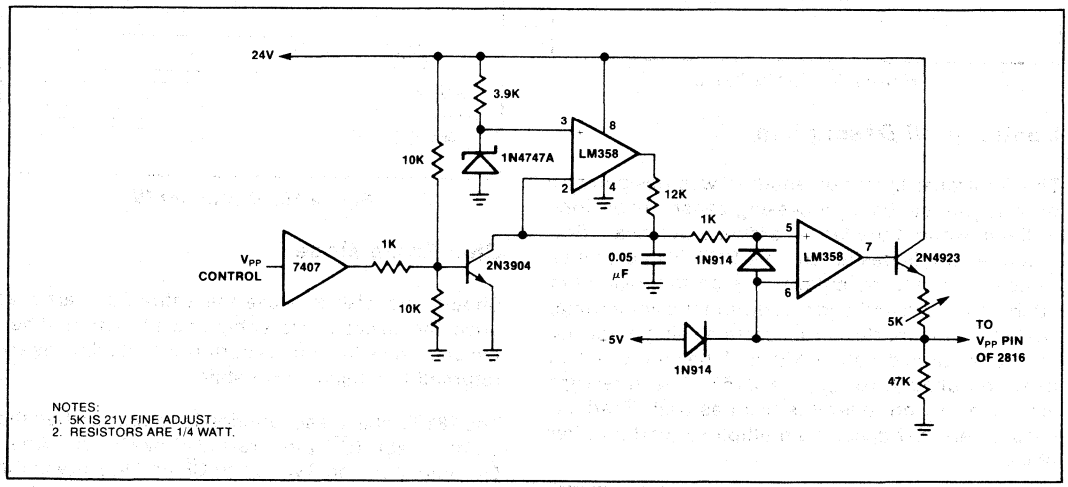


Figure 12a. Operational Amplifier V_{PP} Switch Design[12]

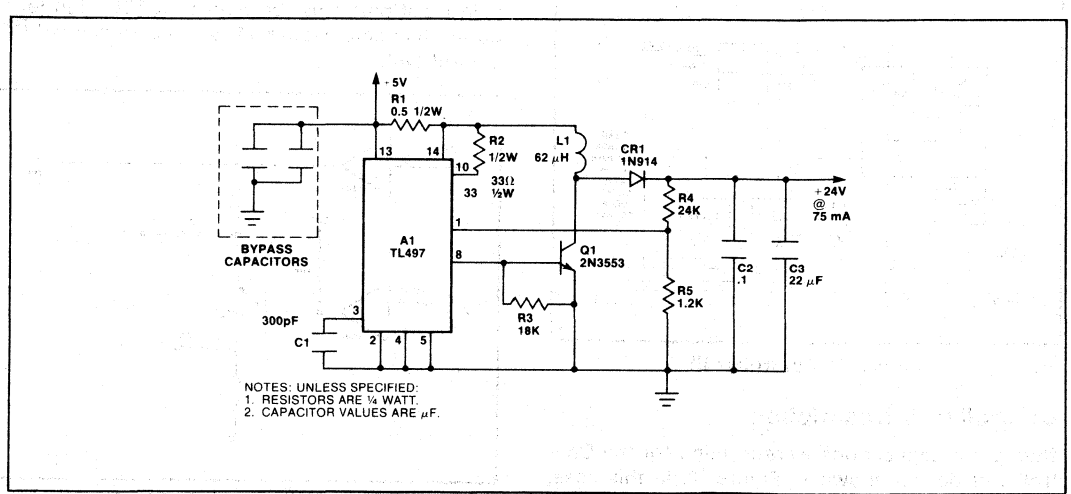


Figure 12b. Step-up Regulator Converts +5V Into +24V

For footnotes see page 13.

Applications

The 2816 E^2PROM is a new and powerful addition to the non-volatile family. It offers a high degree of RAM-like flexibility while retaining the non-volatile characteristics of ROM.

Because of these device parameters, the device is ideal for new and future designs as well as a replacement for existing ROM devices. Some of these potential uses are listed below:

1. Calibration constants storage (continuous calibration).
2. Software alterable control stores (dynamic reconfiguration).
3. Remote communications programming.
4. PC and NC Industrial Applications.
5. CRT terminal configuration and custom graphic and font sets.
6. Military replacements for core memory and fuse-link PROMs.
7. Point of sale terminals.
8. Remote alterable look-up tables.
9. Printer and communications controllers.
10. Remote data gathering.

Because of these device attributes, applications never before possible can now be realized in high performance, consistent microprocessing systems.

Figures 13, 14, 15, and 16 are block diagrams of some typical applications. These applications are explained as follows:

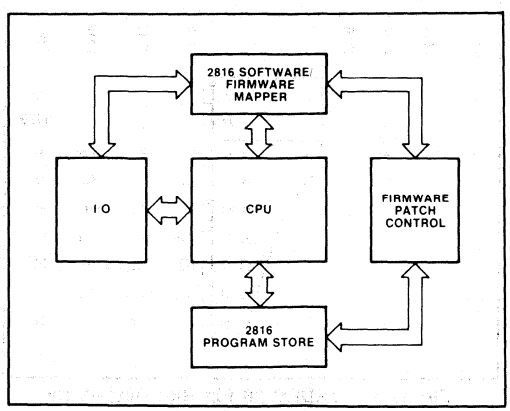


Figure 13. Dynamic Reconfiguration

For footnotes see page 13.

DYNAMIC RECONFIGURATION

The ability of a computer system to alter its operating software while running is now possible with the 2816. The system can monitor external factors, as well as change loop constants, subroutines and other software features in real-time. Figure 13 illustrates this optimal performance. In memory systems, the 2816 can be used to map around hard memory failures in real-time, allowing self-healing memory systems. Such a self-correcting mechanism extends the operating time and reduces service costs to the end user.

CONTINUOUS SELF-CALIBRATION

A high cost of machine service and downtime is due to instrument calibration and readjustments. Use of the 2816 and microprocessor based instruments to contain calibration constants allows features never before possible. See Figure 14. The instrument can now continuously calibrate itself, without expensive downtime in service interaction. The 2816 allows this flexibility and reduction of service costs.

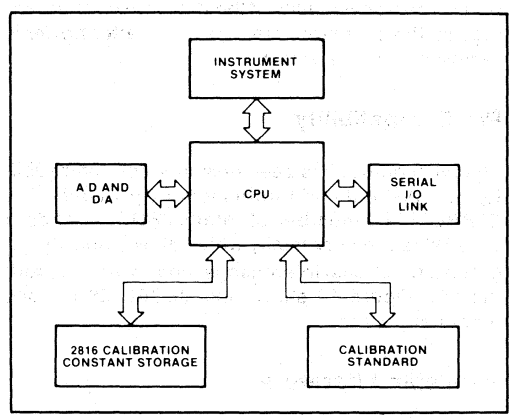


Figure 14. Continuous Self-Calibration

CRT TERMINAL

Custom fonts, graphics characters, and individual configurations can all benefit from the features of the 2816. A CRT terminal, shown in Figure 15, can now be enhanced by using the E^2 as a replacement for jumpers and dip switches. It can also be used as a programmable character generator, and in graphics configuration.

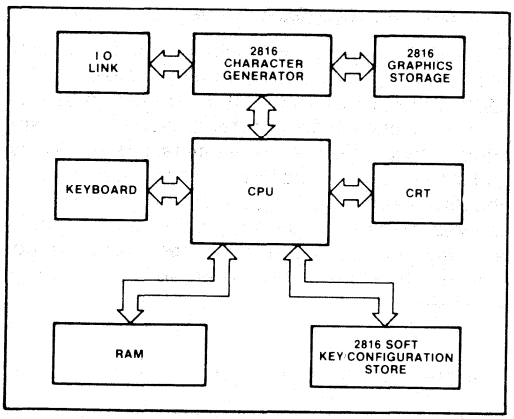


Figure 15. CRT Terminal

POINT OF SALE TERMINAL

Using the 2816 to contain non-volatile price and product descriptions, as shown in Figure 16, is an ideal application in point of sale terminals. With the ability of the 2816 to be altered in-system comes the capability to remotely (over telephone lines) configure the look up table from a central data base computer. The non-volatility of the 2816 is used to advantage as the data store remains intact after power is removed from the system.

Pin Compatibility

The 2816 pinout has been designed for compatibility with present and future memory products. The E^2PROM is a member of Intel's JEDEC standard Byte-Wide memory family which allows density upgrades, functional interchange, and extended product life. Figure 17 shows this JEDEC 28 pin site pinout approach.

Available Literature

To give the system designer an opportunity to more thoroughly understand the device attributes and uses, a library of E^2 information is available. The following list is a brief synopsis:

AP-101—The 2816 Electrical Description

AP-102—2816 Microprocessor Interface Considerations

AP-103—Programming E^2PROM with a Single 5-Volt Power Supply

AP-104—Extending E^2 Endurance—Software Techniques

For footnotes see page 13.

AP-105—Microprocessor Interface—Competitive System Comparisons

AP-106—2816 Byte Erase—Architecture Implications

AP-107—Hardware and Software Download Techniques with 2816

E^2PROM Applications Handbook

To obtain this literature contact your local Field Sales office. In addition, your Field Applications Engineer can discuss with you the controller interfaces for different MPU system configurations.

All of the above literature will be available at the end of Q2 1981. The E^2PROM Applications Handbook is available now.

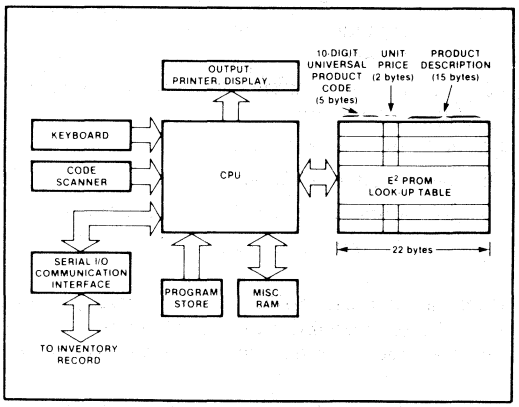


Figure 16. POS Terminal

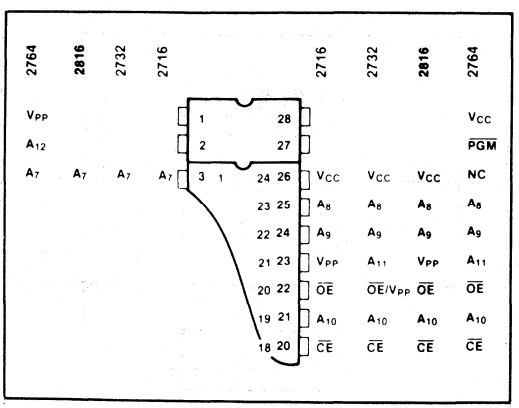


Figure 17. JEDEC 28 Pin Site Byte-Wide Philosophy

Standby Mode

The 2816 has a standby mode which reduces active power dissipation by 55% from 110 mA to 50 mA. The 2816 is placed in the standby mode by applying a TTL high signal to the CE input. When in the standby mode, the outputs are in a high impedance state, independent of the OE input.

Output OR-TIEING

Because 2816s are usually used in larger memory arrays, Intel has provided a 2-line control function that accommodates this use of multiple memory connections. The 2-line control function allows low power dissipation (by deselecting unused devices), and the removal of bus contention from the system environment.

To most effectively use these two control lines, it is recommended that $\overline{CE}$ (pin 18) be decoded from addresses as the primary device selection function. $\overline{OE}$ (pin 20) should be made a common connection to all devices in system, and connected to the $\overline{RD}$ line from the system control bus. This assures that all deselected memory devices are in their low power standby mode and that the output pins are only active when data is desired from a particular memory device.

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias ..... −10°C to +80°C
Storage Temperature ..... −65°C to +125°C
All Input or Output Voltages with Respect to Ground ..... +6V to −0.3V
V_{PP} Supply Voltage with Respect to Ground During Write/Erase ..... +22.5V to −0.1V
Maximum Duration of V_{PP} Supply at 22V During E/W Inhibit ..... 24 Hrs.
Maximum Duration of V_{PP} Supply at 22V During Write/Erase ..... 15 ms[8]

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. AND A.C. OPERATING CONDITIONS DURING READ AND WRITE

	2816	2816-3	2816-4
Temperature Range	0°C-70°C	0°C-70°C	0°C-70°C
V_{CC} Power Supply[9]	5V ± 5%	5V ± 5%	5V ± 5%

D.C. CHARACTERISTICS

READ

Symbol	Parameter	Limits			Units	Test Conditions
		Min.	Typ.[1]	Max.		
I_{LI}	Input Leakage Current			10	μA	V_{IN} = 5.25V
I_{LO}	Output Leakage Current			10	μA	V_{OUT} = 5.25V
I_{CC2}	V_{CC} Current (Active)		50	110	mA	OE = CE = V_{IL}
I_{CC1}	V_{CC} Current (Standby)		25	50	mA	CE = V_{IH}
$I_{PP(R)}$	V_{PP} Current (Read)			5	mA	CE = V_{IL}, V_{PP} = 4 to 6
V_{IL}	Input Low Voltage	−0.1		8	V	
V_{IH}	Input High Voltage	2.0		V_{CC}+1	V	
V_{OL}	Output Low Voltage			.45	V	I_{OL} = 2.1 mA
V_{OH}	Output High Voltage	2.4			V	I_{OH} = −400 μA
V_{PP}	Read Voltage	4		6	V	

WRITE

Symbol	Parameter	Limits			Units	Test Conditions
		Min.	Typ.[1]	Max.		
V_{PP}	Write/Erase Voltage	20	21	22	V	
$I_{PP(W)}$	V_{PP} Current (Byte Erase/Write)		9	15	mA	CE = V_{IL}
V_{OE}	OE Voltage (Chip Erase)	9		15	V	I_{OE} = 10 μA
$I_{PP(I)}$	V_{PP} Current Inhibit			5	mA	V_{PP} = 22, CE = VIH
$I_{PP(C)}$	V_{PP} Current (Chip Erase)		3	5	mA	

For footnotes see page 13.

CAPACITANCE[1] T_A = 25°C, f = 1 MHz

Symbol	Parameter	Typ.	Max.	Units	Test Conditions
C_{IN}	Input Capacitance	5	10	pF	V_{IN} = 0V
C_{OUT}	Output Capacitance		10	pF	V_{OUT} = 0V
$C_{V_{CC}}$	V_{CC} Capacitance		500	pF	OE = CE = V_{IH}
$C_{V_{PP}}$	V_{PP} Capacitance		50	pF	OE = CE = V_{IH}

A.C. TEST CONDITIONS

Output Load: 1TTL gate and C_L = 100 pF
Input Pulse Levels: 0.45 to 2.4V
Timing Measurement Reference Level: Input 1V and 2V
Output .8V and 2V

A.C. CHARACTERISTICS

READ

Symbol	Parameter	2816 Limits			2816-3 Limits			2816-4 Limits			Units	Test Conditions
		Min.	Typ.[1]	Max.	Min.	Typ.[1]	Max.	Min.	Typ.[1]	Max.		
t_{ACC}	Address to Output Delay		200	250		300	350		400	450	ns	CE = OE = V_{IL}
t_{CE}	CE to Output Delay		200	250		300	350		400	450	ns	OE = V_{IL}
t_{OE}	Output Enable to Output Delay	10		100	10		120	10		150	ns	CE = V_{IL}
t_{DF}	Output Enable High to Output Float	0		80	0		100	0		130	ns	CE = V_{IL}
t_{OH}	Output Hold from Addresses, CE or OE Whichever Occurred First	0			0			0			ns	CE = OE = V_{IL}

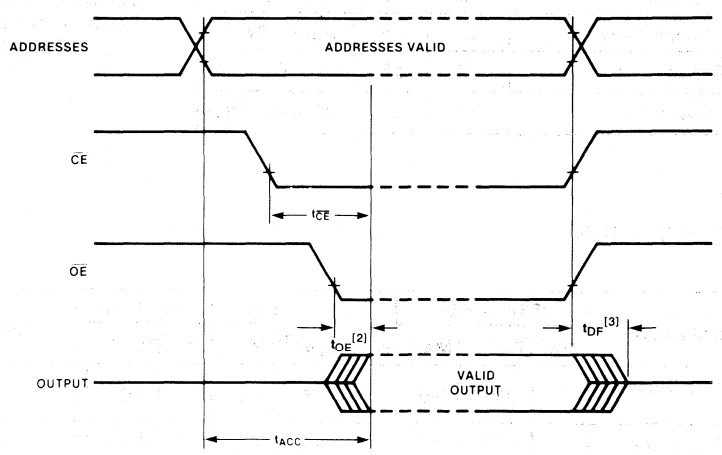
WRITE

Symbol	Parameter	Limits			Units	Test Conditions
		Min.	Typ.[1]	Max.		
t_{AS}	Add to V_{PP} Set-Up Time	150			ns	
t_{CS}	CE to V_{PP} Set-Up Time	150			ns	
t_{DS}[10]	Data to V_{PP} Set-Up Time	0			ns	
t_{DH}[10]	Data Hold Time	50			ns	V_{PP} = 6V
t_{WP}[8]	Write Pulse Width	9	10	15	ms	
t_{WR}	Write Recovery Time	50			ns	V_{PP} = 6V
t_{OS}	Chip Erase Set-Up Time	0			ns	V_{PP} = 6V, V_{OE} = 9V
t_{OH}	Chip Erase Hold Time	0			ns	V_{PP} = 6V, V_{OE} = 9V
t_{PRC}	V_{PP} RC Time Constant	450	600	750	μs	
t_{PFT}[7]	V_{PP} Fall Time			100	μs	V_{PP} = 6V
t_{BOS}	Byte Erase/Write Set-Up Time	0			ns	V_{PP} = 6V
t_{BOH}	Byte Erase/Write Hold Time	0			ns	V_{PP} = 6V

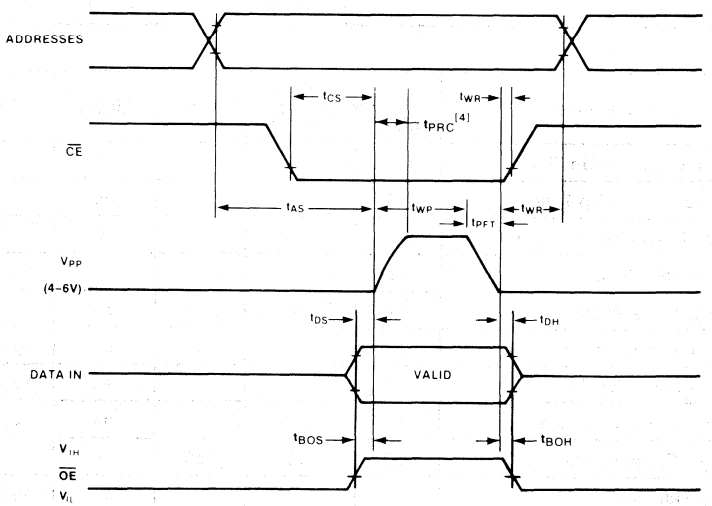
For footnotes see page 13.

WAVEFORMS

READ



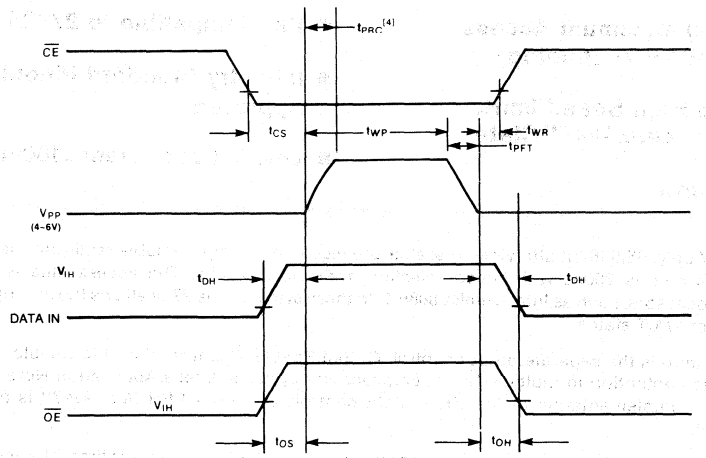
BYTE ERASE[5] OR WRITE



For footnotes see page 13.

WAVEFORMS (Continued)

CHIP ERASE[6]



NOTES:

1. This parameter is only sampled and not 100% tested.
2. OE may be delayed up to 230 ns after falling edge of CE without impact on t_{ACC} for 2816.
3. t_{DF} is specified from OE or CE whichever occurs first.
4. The rising edge of V_{PP} must follow an exponential waveform. That waveform's time constant is specified as t_{PRC}. See Intel's AP-102 for details.
5. Prior to a data write, an erase operation must be performed. For erase, data in = V_{IH}.
6. In the chip erase mode $D_{IN} = V_{IH}$.
7. To allow immediate read verify capability, V_{PP} can be driven low in less than 50 ns. See AP-101 for more information.
8. Adherence to TWP specification is important to device reliability.
9. To prevent spurious device erasure or write, V_{CC} must be applied simultaneously or before 21 volt application of V_{PP}. V_{PP} cannot be driven to 21 volts without previously applying V_{CC}.
10. The data in set up and hold times for chip erase are identical to those specified for byte erase.
11. This switch includes automatic voltage shutdown on power fail.

2764
(8K x 8) UV ERASABLE PROM

- **200 ns (2764-2) Maximum Access Time . . . HMOS*-E Technology**
- **Compatible to High Speed 8mHz 8086-2 MPU . . . Zero WAIT State**
- **Two Line Control**
- **Pin Compatible to 2732A EPROM**
- **Industry Standard Pinout . . . JEDEC Approved**
- **Low Active Current...100mA Max.**

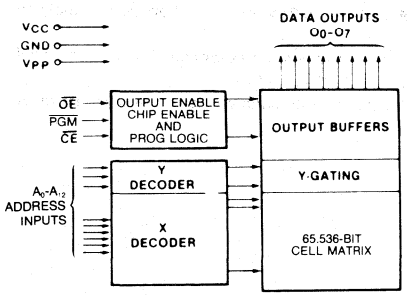
The Intel® 2764 is a 5V only, 65,536-bit ultraviolet erasable and electrically programmable read-only memory (EPROM). The standard 2764 access time is 250ns with speed selection available at 200ns. The access time is compatible to high performance microprocessors, such as Intel's 8mHz 8086-2. In these systems, the 2764 allows the microprocessor to operate without the addition of WAIT states.

An important 2764 feature is the separate output control, Output Enable ($\overline{OE}$) from the Chip Enable control ($\overline{CE}$). The $\overline{OE}$ control eliminates bus contention in multiple bus microprocessor systems. Intel's Application Note AP-72 describes the microprocessor system implementation of the $\overline{OE}$ and $\overline{CE}$ controls on Intel's EPROMs. AP-72 is available from Intel's Literature Department.

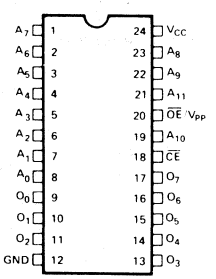
The 2764 has a standby mode which reduces the power dissipation without increasing access time. The active current is 100mA, while the standby current is only 50mA. The standby mode is achieved by applying a TTL-high signal to the $\overline{CE}$ input.

The 2764 is fabricated with HMOS*-E technology, Intel's high-speed N-channel MOS Silicon Gate Technology.

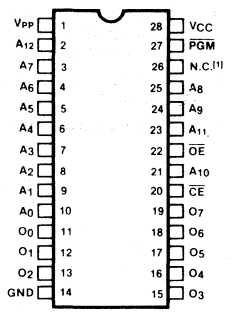
BLOCK DIAGRAM



2732A PIN CONFIGURATION



2764 PIN CONFIGURATION



[1] For total compatibility and upgradability from the 2732A and ROMs provide a trace to pin 26.

MODE SELECTION

MODE \ PINS	CE (20)	OE (22)	PGM (27)	V_{PP} (1)	V_{CC} (28)	Outputs (11-13, 15-19)
Read	V_{IL}	V_{IL}	V_{IH}	V_{CC}	V_{CC}	D_{OUT}
Standby	V_{IH}	x	x	V_{CC}	V_{CC}	High Z
Program	V_{IL}	x	V_{IL}	V_{PP}	V_{CC}	D_{IN}
Program Verify	V_{IL}	V_{IL}	V_{IH}	V_{PP}	V_{CC}	D_{OUT}
Program Inhibit	V_{IH}	x	x	V_{PP}	V_{CC}	High Z

x can be either V_{IL} or V_{IH}

PIN NAMES

A_0-A_{12}	ADDRESSES
$\overline{CE}$	CHIP ENABLE
$\overline{OE}$	OUTPUT ENABLE
O_0-O_7	OUTPUTS
$\overline{PGM}$	PROGRAM
N.C.	NO CONNECT

*HMOS is a patented process of Intel Corporation.

ABSOLUTE MAXIMUM RATINGS*

Temperature Under Bias .....−10°C to +80°C
Storage Temperature .....−65°C to +125°C
All Input or Output Voltages with
Respect to Ground .....+6V to −0.6V
V_{PP} Supply Voltage with Respect to Ground
During Programming .....+22V to −0.6V

*COMMENT

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. and A.C. Operating Conditions During Read

	2764	2764-2	2764-3	2764-4
Operating Temperature Range	0°C–70°C	0°C–70°C	0°C–70°C	0°C–70°C
V_{CC} Power Supply[1,2]	5V± 5%	5V ± 5%	5V ± 5%	5V ±5%
V_{PP} Voltage[2]	$V_{PP} = V_{CC}$	$V_{PP} = V_{CC}$	$V_{PP} = V_{CC}$	$V_{PP} = V_{CC}$

READ OPERATION

D.C. AND OPERATING CHARACTERISTICS

Symbol	Parameter	Limits			Unit	Conditions
		Min	Typ[3]	Max		
I_{LI}	Input Load Current			10	μA	$V_{IN} = 5.25V$
I_{LO}	Output Leakage Current			10	μA	$V_{OUT} = 5.25V$
I_{PP1}[2]	V_{PP} Current Read			15	mA	$V_{PP} = 5.25V$
I_{CC1}[2]	V_{CC} Current Standby			50	mA	$\overline{CE} = V_{IH}$
I_{CC2}[2]	V_{CC} Current Active		70	100	mA	$\overline{CE} = \overline{OE} = V_{IL}$
V_{IL}	Input Low Voltage	−.1		+.8	V	
V_{IH}	Input High Voltage	2.0		$V_{CC}+1$	V	
V_{OL}	Output Low Voltage			.45	V	$I_{OL} = 2.1$ mA
V_{OH}	Output High Voltage	2.4			V	$I_{OH} = -400$ μA

NOTES: 1. V_{CC} must be applied simultaneously or before V_{PP} and removed simultaneously or after V_{PP}.
2. V_{PP} may be connected directly to V_{CC} except during programming. The supply current would then be the sum of I_{CC} and I_{PP1}.
3. Typical values are for $T_A = 25°C$ and nominal supply voltages.

A.C. CHARACTERISTICS

Symbol	Parameter	2764-2 Limits		2764 Limits		2764-3 Limits		2764-4 Limits		Unit	Test Conditions
		Min	Max	Min	Max	Min	Max	Min	Max		
t_{ACC}	Address to Output Delay		200		250		300		450	ns	$CE=OE=V_{IL}$
t_{CE}	CE to Output Delay		200		250		300		450	ns	$OE=V_{IL}$
t_{OE}	Output Enable to Output Delay	10	70	10	100	10	150	10	150	ns	$CE=V_{IL}$
t_{DF}	Output Enable High to Output Float	0	60	0	90	0	130	0	130	ns	$CE=V_{IL}$
t_{OH}	Output Hold from Addresses, CE or OE Whichever Occurred First	0		0		0		0		ns	$CE=OE=V_{IL}$

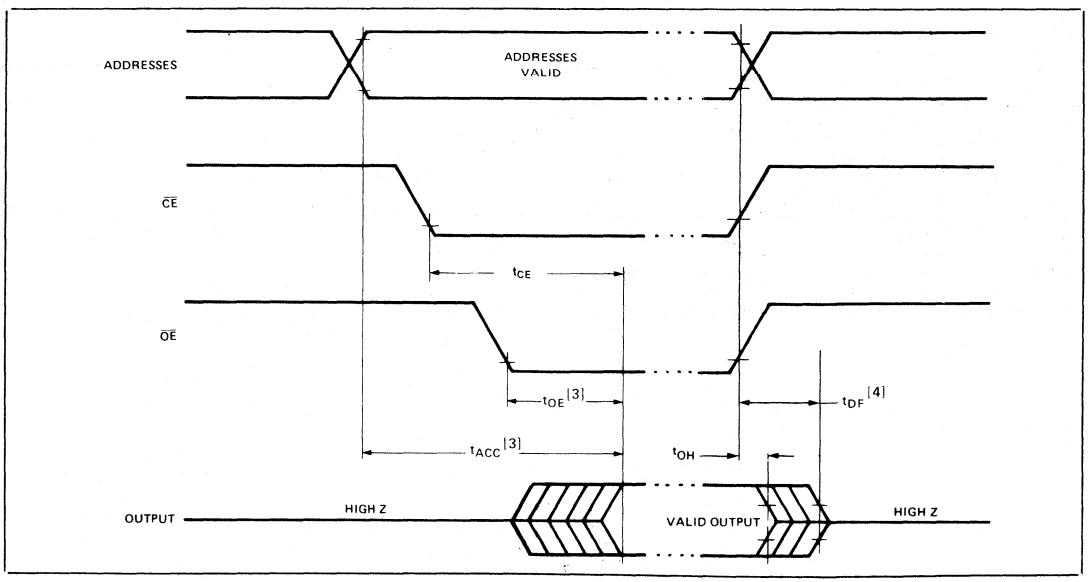
CAPACITANCE [1] $T_A = 25°C$, f = 1MHz

Symbol	Parameter	Typ.	Max.	Unit	Conditions
C_{IN}	Input Capacitance	4	6	pF	$V_{IN}=0V$
C_{OUT}	Output Capacitance	8	12	pF	$V_{OUT}=0V$

A.C. TEST CONDITIONS

Output Load: 1 TTL gate and C_L = 100pF
Input Rise and Fall Times: ≤ 20ns
Input Pulse Levels: 0.8V to 2.2V
Timing Measurement Reference Level:
Inputs 1V and 2V
Outputs 0.8V and 2V

A.C. WAVEFORMS



NOTES: 1. Typical values are for T_A = 25°C and nominal supply voltages.
2. This parameter is only sampled and is not 100% tested.
3. $\overline{OE}$ may be delayed up to $t_{ACC} - t_{CE}$ after the falling edge of $\overline{CE}$ without impact on t_{ACC}.
4. t_{DF} is specified from $\overline{OE}$ or $\overline{CE}$, whichever occurs first.

PROGRAMMING

D.C. PROGRAMMING CHARACTERISTICS: $T_A = 25 \pm 5°C$, $V_{CC} = 5V \pm 5\%$, $V_{PP} = 21V \pm 0.5V$ (see Note 1)

Symbol	Parameter	Limits				Test Conditions
		Min.	Typ.	Max.	Unit	
I_{LI}	Input Current (All Inputs)			10	μA	$V_{IN} = V_{IL}$ or V_{IH}
V_{OL}	Output Low Voltage During Verify			0.45	V	$I_{OL} = 2.1$ mA
V_{OH}	Output High Voltage During Verify	2.4			V	$I_{OH} = -400\ \mu A$
I_{CC2}	V_{CC} Supply Current (Active)			100	mA	
V_{IL}	Input Low Level (All Inputs)	−0.1		0.8	V	
V_{IH}	Input High Level	2.0		$V_{CC}+1$	V	
I_{PP}	V_{PP} Supply Current			30	mA	$\overline{CE} = V_{IL} = \overline{PGM}$

A.C. PROGRAMMING CHARACTERISTICS: $T_A = 25 \pm 5°C$, $V_{CC} = 5V \pm 5\%$, $V_{PP} = 21V \pm 0.5V$ (see Note 1)

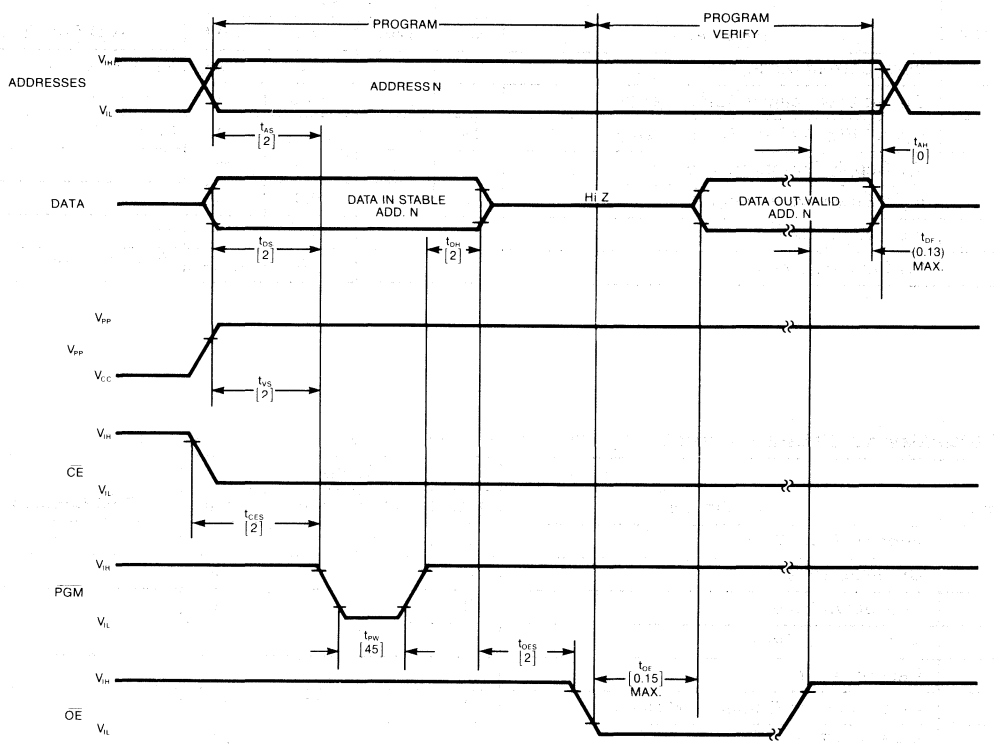
Symbol	Parameter	Limits				Test Conditions*
		Min.	Typ.	Max.	Unit	
t_{AS}	Address Setup Time	2			μs	
t_{OES}	$\overline{OE}$ Setup Time	2			μs	
t_{DS}	Data Setup Time	2			μs	
t_{AH}	Address Hold Time	0			μs	
t_{DH}	Data Hold Time	2			μs	
t_{DF}	Chip Enable to Output Float Delay	0		130	ns	
t_{VS}	V_{PP} Setup Time	2			μs	
t_{PW}	$\overline{PGM}$ Pulse Width During Programming	45	50	55	ms	
t_{CES}	$\overline{CE}$ Setup Time	2			μs	
t_{OE}	Data Valid from $\overline{OE}$			150	ns	

*A.C. CONDITIONS OF TEST

Input Rise and Fall Times (10% to 90%) 20ns
Input Pulse Levels . . . . . 0.8V to 2.2V
Input Timing Reference Level . . . . . 1V and 2V
Output Timing Reference Level . . . . . 0.8V and 2.0V

NOTE: 1. V_{CC} must be applied simultaneously or before V_{PP} and removed simultaneously or after V_{PP}.

PROGRAMMING WAVEFORMS



1. ALL TIMES SHOWN IN [ ] ARE MINIMUM AND IN μSEC UNLESS OTHERWISE SPECIFIED.
2. THE INPUT TIMING REFERENCE LEVEL IS 1V FOR A V_{IL} AND 2V FOR A V_{IH}.

ERASURE CHARACTERISTICS

The erasure characteristics of the 2764 are such that erasure begins to occur when exposed to light with wavelengths shorter than approximately 4000 Angstroms (Å). It should be noted that sunlight and certain types of fluorescent lamps have wavelengths in the 3000–4000 Å range. Data show that constant exposure to room level fluorescent lighting could erase the typical 2764 in approximately 3 years, while it would take approximately 1 week to cause erasure when exposed to direct sunlight. If the 2764 is to be exposed to these types of lighting conditions for extended periods of time, opaque labels are available from Intel which should be placed over the 2764 window to prevent unintentional erasure.

The recommended erasure procedure for the 2764 is exposure to shortwave ultraviolet light which has a wavelength of 2537 Angstroms (Å). The integrated dose (i.e., UV intensity X exposure time) for erasure should be a minimum of 15 W-sec/cm^2. The erasure time with this dosage is approximately 15 to 20 minutes using an ultraviolet lamp with 12000 μW/cm^2 power rating. The 2764 should be placed within 1 inch of the lamp tubes during erasure. Some lamps have a filter on their tubes which should be removed before erasure.

DEVICE OPERATION

The five modes of operation of the 2764 are listed in Table 1. A single 5V power supply is required in the read mode. All inputs are TTL levels except for V_{PP}.

TABLE 1. MODE SELECTION

MODE \ PINS	$\overline{CE}$ (20)	$\overline{OE}$ (22)	$\overline{PGM}$ (27)	V_{PP} (1)	V_{CC} (28)	Outputs (11-13, 15-19)
Read	V_{IL}	V_{IL}	V_{IH}	V_{CC}	V_{CC}	D_{OUT}
Standby	V_{IH}	x	x	V_{CC}	V_{CC}	High Z
Program	V_{IL}	x	V_{IL}	V_{PP}	V_{CC}	D_{IN}
Program Verify	V_{IL}	V_{IL}	V_{IH}	V_{PP}	V_{CC}	D_{OUT}
Program Inhibit	V_{IH}	x	x	V_{PP}	V_{CC}	High Z

x can be either V_{IL} or V_{IH}

READ MODE

The 2764 has two control functions, both of which must be logically satisfied in order to obtain data at the outputs. Chip Enable ($\overline{CE}$) is the power control and should be used for device selection. Output Enable ($\overline{OE}$) is the output control and should be used to gate data to the output pins, independent of device selection. Assuming that addresses are stable, address access time (t_{ACC}) is equal to the delay from $\overline{CE}$ to output ($t_{\overline{CE}}$). Data is available at the outputs after the falling edge of $\overline{OE}$, assuming that $\overline{CE}$ has been low and addresses have been stable for at least $t_{ACC} - t_{\overline{OE}}$.

Standby Mode

The 2764 has a standby mode which reduces the active power current from 100mA to 50mA. The 2764 is placed in the standby mode by applying a TTL high signal to the $\overline{CE}$ input. When in standby mode, the outputs are in a high impedance state, independent of the $\overline{OE}$ input.

Output OR-Tieing

Because EPROMs are usually used in larger memory arrays, Intel has provided a 2 line control function that accommodates this use of multiple memory connection. The two line control function allows for:

a) the lowest possible memory power dissipation, and
b) complete assurance that output bus contention will not occur.

To most efficiently use these two control lines, it is recommended that $\overline{CE}$ (pin 20) be decoded and used as the primary device selecting function, while $\overline{OE}$ (pin 22) be made a common connection to all devices in the array and connected to the READ line from the system control bus. This assures that all deselected memory devices are in their low power standby mode and that the output pins are only active when data is desired from a particular memory device.

Programming

Programming is the same as Intel's 2732A except that $\overline{OE}/V_{PP}$ is not multiplexed. They have separate pins. Like the 2732A, ***exceeding 21.5V will damage the 2764.***

Initially, and after each erasure, all bits of the 2764 are in the "1" state. Data is introduced by selectively programming "0s" into the desired bit locations. Although only "0s" will be programmed, both "1s" and "0s" can be present in the data word. The only way to change a "0" to a "1" is by ultraviolet light erasure.

The 2764 is in the programming mode when V_{PP} input is at 21V and $\overline{CE}$ and $\overline{PGM}$ are both at TTL low. The data to be programmed is applied 8 bits in parallel to the data output pins. The levels required for the address and data inputs are TTL.

For programming, $\overline{CE}$ should be kept TTL low at all times while V_{PP} is kept at 21V. When the address and data are stable, a 50 msec, active low, TTL program pulse is applied to $\overline{PGM}$ input. A program pulse must be applied at each address location to be programmed. You can program any location at any time—either individually, sequentially, or at random. The program pulse has a maximum width of 55 msec.

Programming of multiple 2764s in parallel with the same data can be easily accomplished due to the simplicity of the programming requirements. Like inputs of the paralleled 2764s may be connected together when they are programmed with the same data. A low level TTL pulse applied to the $\overline{PGM}$ input programs the paralleled 2764s.

Program Inhibit

Programming of multiple 2764s in parallel with different data is also easily accomplished. A high level $\overline{CE}$ or $\overline{PGM}$ input inhibits the other 2764s from being programmed. Except for $\overline{CE}$ (or $\overline{PGM}$), all like inputs (including $\overline{OE}$) of the parallel 2764s may be common. A TTL low level pulse applied to a 2764 $\overline{CE}$ and $\overline{PGM}$ input with V_{PP} at 21V will program that 2764.

Program Verify

A verify should be performed on the programmed bits to determine that they were correctly programmed. The verify is accomplished with $\overline{CE}$ and $\overline{OE}$ at V_{IL}. However, $\overline{PGM}$ is at V_{IH}.

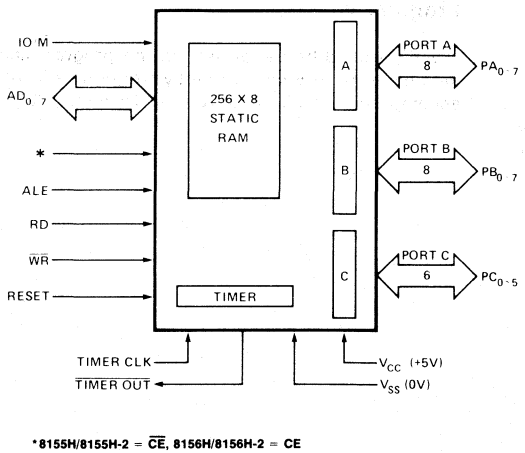
8155H/8156H/8155H-2/8156H-2
2048-BIT STATIC HMOS RAM WITH I/O PORTS AND TIMER

- **Single +5V Power Supply with 10% Voltage Margins**
- **30% Lower Power Consumption than the 8155 and 8156**
- **100% Compatible with 8155 and 8156**
- **256 Word x 8 Bits**
- **Completely Static Operation**
- **Internal Address Latch**
- **2 Programmable 8-Bit I/O Ports**
- **1 Programmable 6-Bit I/O Port**
- **Programmable 14-Bit Binary Counter/ Timer**
- **Compatible with 8085AH, 8085A and 8088 CPU**
- **Multiplexed Address and Data Bus**

The Intel® 8155H and 8156H are RAM and I/O chips implemented in N-Channel, depletion load, silicon gate technology (HMOS), to be used in the 8085AH and 8088 microprocessor systems. The RAM portion is designed with 2048 static cells organized as 256 x 8. They have a maximum access time of 400 ns to permit use with no wait states in 8085AH CPU. The 8155H-2 and 8156H-2 have maximum access times of 330 ns for use with the 8085AH-2 and the full-speed 5 MHz 8088 CPU.

The I/O portion consists of three general purpose I/O ports. One of the three ports can be programmed to be status pins, thus allowing the other two ports to operate in handshake mode.

A 14-bit programmable counter/timer is also included on chip to provide either a square wave or terminal count pulse for the CPU system depending on timer mode.



*8155H/8155H-2 = $\overline{CE}$, 8156H/8156H-2 = CE

Figure 1. Block Diagram

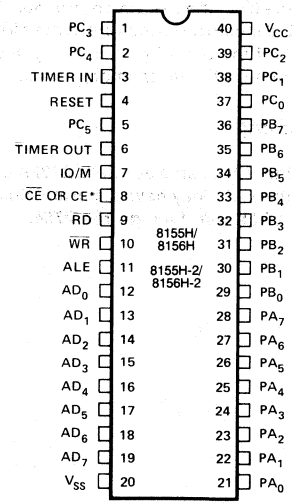


Figure 2. Pin Configuration

APRIL 1981
AFN-02201D

Table 1. Pin Description

Symbol	Type	Name and Function
RESET	I	**Reset:** Pulse provided by the 8085AH to initialize the system (connect to 8085AH RESET OUT). Input high on this line resets the chip and initializes the three I/O ports to input mode. The width of RESET pulse should typically be two 8085AH clock cycle times.
AD_{0-7}	I/O	**Address/Data:** 3-state Address/Data lines that interface with the CPU lower 8-bit Address/Data Bus. The 8-bit address is latched into the address latch inside the 8155H/56H on the falling edge of ALE. The address can be either for the memory section or the I/O section depending on the IO/$\overline{M}$ input. The 8-bit data is either written into the chip or read from the chip, depending on the $\overline{WR}$ or $\overline{RD}$ input signal.
CE or $\overline{CE}$	I	**Chip Enable:** On the 8155H, this pin is $\overline{CE}$ and is ACTIVE LOW. On the 8156H, this pin is CE and is ACTIVE HIGH.
$\overline{RD}$	I	**Read Control:** Input low on this line with the Chip Enable active enables and AD_{0-7} buffers. If IO/$\overline{M}$ pin is low, the RAM content will be read out to the AD bus. Otherwise the content of the selected I/O port or command/status registers will be read to the AD bus.
$\overline{WR}$	I	**Write Control:** Input low on this line with the Chip Enable active causes the data on the Address/Data bus to be written to the RAM or I/O ports and command/status register, depending on IO/$\overline{M}$.
ALE	I	**Address Latch Enable:** This control signal latches both the address on the AD_{0-7} lines and the state of the Chip Enable and IO/$\overline{M}$ into the chip at the falling edge of ALE.
IO/$\overline{M}$	I	**I/O Memory:** Selects memory if low and I/O and command/status registers if high.
PA_{0-7}(8)	I/O	**Port A:** These 8 pins are general purpose I/O pins. The in/out direction is selected by programming the command register.
PB_{0-7}(8)	I/O	**Port B:** These 8 pins are general purpose I/O pins. The in/out direction is selected by programming the command register.
PC_{0-5}(6)	I/O	**Port C:** These 6 pins can function as either input port, output port, or as control signals for PA and PB. Programming is done through the command register. When PC_{0-5} are used as control signals, they will provide the following: PC_0 — A INTR (Port A Interrupt) PC_1 — ABF (Port A Buffer Full) PC_2 — $\overline{A\ STB}$ (Port A Strobe) PC_3 — B INTR (Port B Interrupt) PC_4 — B BF (Port B Buffer Full) PC_5 — $\overline{B\ STB}$ (Port B Strobe)
TIMER IN	I	**Timer Input:** Input to the counter-timer.
$\overline{TIMER\ OUT}$	O	**Timer Output:** This output can be either a square wave or a pulse, depending on the timer mode.
V_{CC}		**Voltage:** +5 volt supply.
V_{SS}		**Ground:** Ground reference.

FUNCTIONAL DESCRIPTION

The 8155H/8156H contains the following:

- 2k Bit Static RAM organized as 256 x 8
- Two 8-bit I/O ports (PA & PB) and one 6-bit I/O port (PC)
- 14-bit timer-counter

The IO/$\overline{M}$ (IO/Memory Select) pin selects either the five registers (Command, Status, PA_{0-7}, PB_{0-7}, PC_{0-5}) or the memory (RAM) portion.

The 8-bit address on the Address/Data lines, Chip Enable input CE or $\overline{CE}$, and IO/$\overline{M}$ are all latched on-chip at the falling edge of ALE.

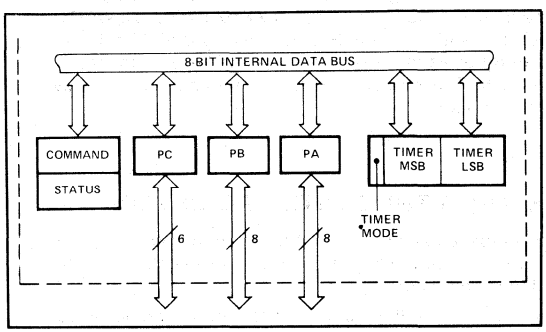


Figure 3. 8155H/8156H Internal Registers

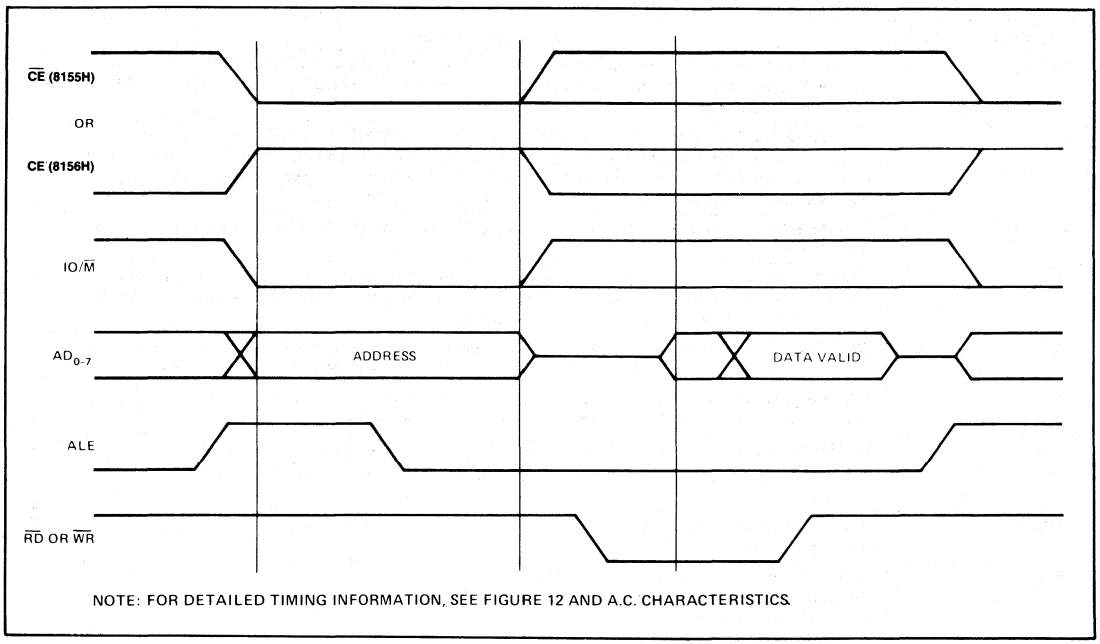


Figure 4. 8155H/8156H On-Board Memory Read/Write Cycle

PROGRAMMING OF THE COMMAND REGISTER

The command register consists of eight latches. Four bits (0-3) define the mode of the ports, two bits (4-5) enable or disable the interrupt from port C when it acts as control port, and the last two bits (6-7) are for the timer.

The command register contents can be altered at any time by using the I/O address XXXXX000 during a WRITE operation with the Chip Enable active and IO/$\overline{M}$ = 1. The meaning of each bit of the command byte is defined in Figure 5. The contents of the command register may never be read.

READING THE STATUS REGISTER

The status register consists of seven latches, one for each bit; six (0-5) for the status of the ports and one (6) for the status of the timer.

The status of the timer and the I/O section can be polled by reading the Status Register (Address XXXXX000). Status word format is shown in Figure 6. Note that you may never write to the status register since the command register shares the same I/O address and the command register is selected when a write to that address is issued.

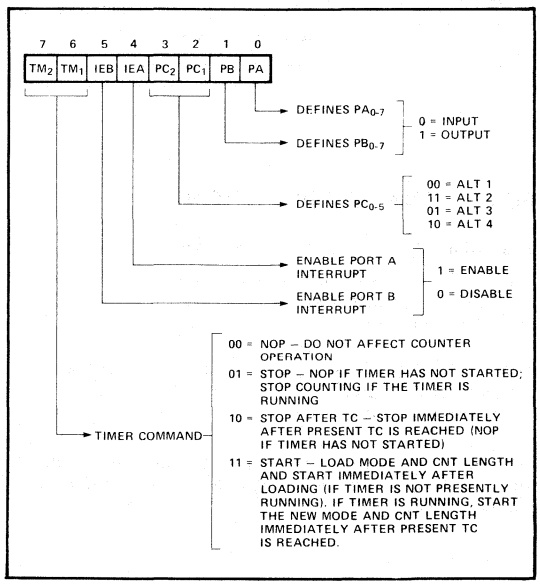


Figure 5. Command Register Bit Assignment

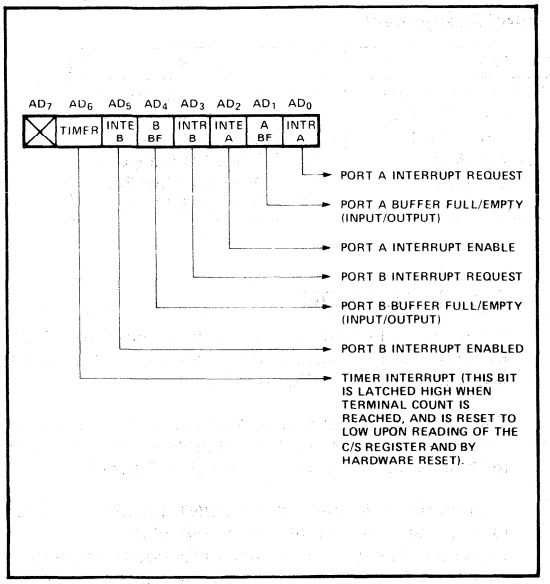


Figure 6. Status Register Bit Assignment

INPUT/OUTPUT SECTION

The I/O section of the 8155H/8156H consists of five registers: (See Figure 7.)

- **Command/Status Register (C/S)** — Both registers are assigned the address XXXXX000. The C/S address serves the dual purpose.

  When the C/S registers are selected during WRITE operation, a command is written into the command register. The contents of this register are *not* accessible through the pins.

  When the C/S (XXXXX000) is selected during a READ operation, the status information of the I/O ports and the timer becomes available on the AD_{0-7} lines.

- **PA Register** — This register can be programmed to be either input or output ports depending on the status of the contents of the C/S Register. Also depending on the command, this port can operate in either the basic mode or the strobed mode (See timing diagram). The I/O pins assigned in relation to this register are PA_{0-7}. The address of this register is XXXXX001.

- **PB Register** — This register functions the same as PA Register. The I/O pins assigned are PB_{0-7}. The address of this register is XXXXX010.

- **PC Register** — This register has the address XXXXX011 and contains only 6 bits. The 6 bits can be programmed to be either input ports, output ports or as control signals for PA and PB by properly programming the AD_2 and AD_3 bits of the C/S register.

  When PC_{0-5} is used as a control port, 3 bits are assigned for Port A and 3 for Port B. The first bit is an interrupt that the 8155H sends out. The second is an output signal indicating whether the buffer is full or empty, and the third is an input pin to accept a strobe for the strobed input mode. (See Table 2.)

When the 'C' port is programmed to either ALT3 or ALT4, the control signals for PA and PB are initialized as follows:

CONTROL	INPUT MODE	OUTPUT MODE
BF	Low	Low
INTR	Low	High
STB	Input Control	Input Control

I/O ADDRESS†								SELECTION
A7	A6	A5	A4	A3	A2	A1	A0	
X	X	X	X	X	0	0	0	Interval Command Status Register
X	X	X	X	X	0	0	1	General Purpose I O Port A
X	X	X	X	X	0	1	0	General Purpose I O Port B
X	X	X	X	X	0	1	1	Port C — General Purpose I O or Control
X	X	X	X	X	1	0	0	Low-Order 8 bits of Timer Count
X	X	X	X	X	1	0	1	High 6 bits of Timer Count and 2 bits of Timer Mode

X: Don't Care.

†: I/O Address must be qualified by CE = 1 (8156H) or $\overline{CE}$ = 0 (8155H) and IO/$\overline{M}$ = 1 in order to select the appropriate register.

Figure 7. I/O Port and Timer Addressing Scheme

Figure 8 shows how I/O PORTS A and B are structured within the 8155H and 8156H:

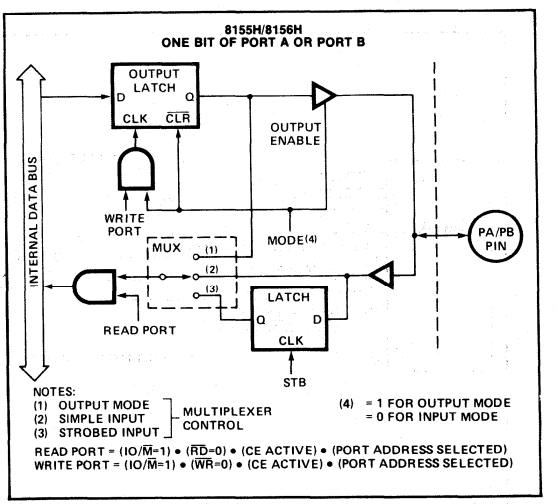


Figure 8. 8155H/8156H Port Functions

Table 2. Port Control Assignment

Pin	ALT 1	ALT 2	ALT 3	ALT 4
PC0	Input Port	Output Port	A INTR (Port A Interrupt)	A INTR (Port A Interrupt)
PC1	Input Port	Output Port	A BF (Port A Buffer Full)	A BF (Port A Buffer Full)
PC2	Input Port	Output Port	A $\overline{STB}$ (Port A Strobe)	A $\overline{STB}$ (Port A Strobe)
PC3	Input Port	Output Port	Output Port	B INTR (Port B Interrupt)
PC4	Input Port	Output Port	Output Port	B BF (Port B Buffer Full)
PC5	Input Port	Output Port	Output Port	B $\overline{STB}$ (Port B Strobe)

Note in the diagram that when the I/O ports are programmed to be output ports, the contents of the output ports can still be read by a READ operation when appropriately addressed.

The outputs of the 8155H/8156H are "glitch-free" meaning that you can write a "1" to a bit position that was previously "1" and the level at the output pin will not change.

Note also that the output latch is cleared when the port enters the input mode. The output latch cannot be loaded by writing to the port if the port is in the input mode. The result is that each time a port mode is changed from input to output, the output pins will go low. When the 8155H/56H is RESET, the output latches are all cleared and all 3 ports enter the input mode.

When in the ALT 1 or ALT 2 modes, the bits of PORT C are structured like the diagram above in the simple input or output mode, respectively.

Reading from an input port with nothing connected to the pins will provide unpredictable results.

Figure 9 shows how the 8155H/8156H I/O ports might be configured in a typical MCS-85 system.

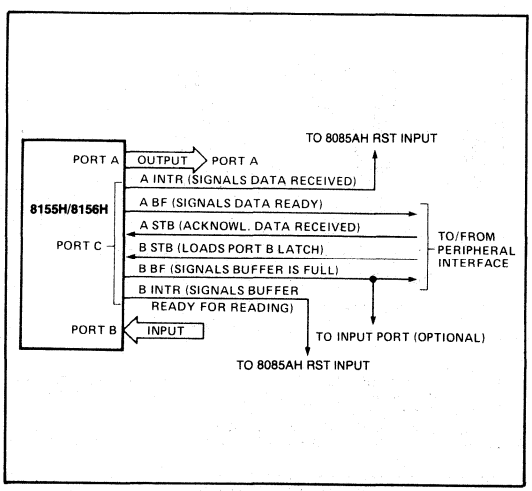


Figure 9. Example: Command Register = 00111001

TIMER SECTION

The timer is a 14-bit down-counter that counts the TIMER IN pulses and provides either a square wave or pulse when terminal count (TC) is reached.

The timer has the I/O address XXXXX100 for the low order byte of the register and the I/O address XXXXX101 for the high order byte of the register. (See Figure 7.)

To program the timer, the COUNT LENGTH REG is loaded first, one byte at a time, by selecting the timer addresses. Bits 0-13 of the high order count register will specify the length of the next count and bits 14-15 of the high order register will specify the timer output mode (see Figure 10). The value loaded into the count length register can have any value from 2H through 3FFH in Bits 0–13.

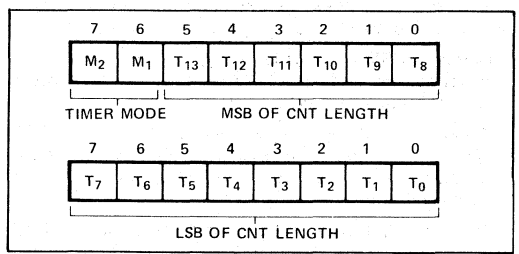


Figure 10. Timer Format

There are four modes to choose from: M2 and M1 define the timer mode, as shown in Figure 11.

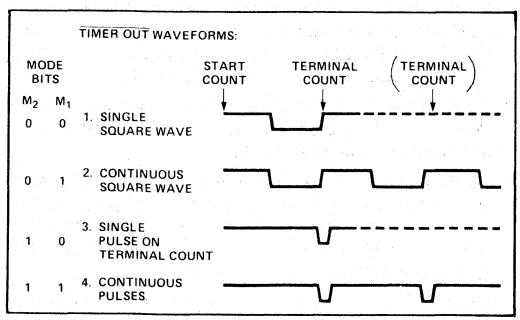


Figure 11. Timer Modes

Bits 6-7 (TM_2 and TM_1) of command register contents are used to start and stop the counter. There are four commands to choose from:

TM_2	TM_1	
0	0	NOP — Do not affect counter operation.
0	1	STOP — NOP if timer has not started; stop counting if the timer is running.
1	0	STOP AFTER TC — Stop immediately after present TC is reached (NOP if timer has not started)
1	1	START — Load mode and CNT length and start immediately after loading (if timer is not presently running). If timer is running, start the new mode and CNT length immediately after present TC is reached.

Note that while the counter is counting, you may load a new count and mode into the count length registers. Before the new count and mode will be used by the counter, you must issue a START command to the counter. This applies even though you may only want to change the count and use the previous mode.

In case of an odd-numbered count, the first half-cycle of the squarewave output, which is high, is one count longer than the second (low) half-cycle, as shown in Figure 12.

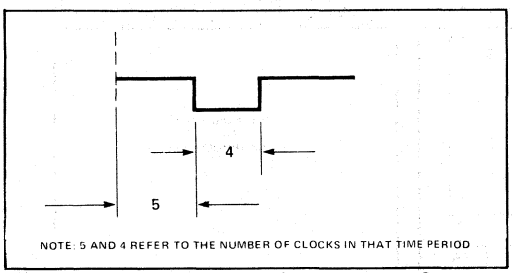


Figure 12. Asymmetrical Square-Wave Output Resulting from Count of 9

The counter in the 8155H is not initialized to any particular mode or count when hardware RESET occurs, but RESET does *stop* the counting. Therefore, counting cannot begin following RESET until a START command is issued via the C/S register.

Please note that the timer circuit on the 8155H/8156H chip is designed to be a square-wave timer, not an event counter. To achieve this, it counts down by twos twice in completing one cycle. Thus, its registers do not contain values directly representing the number of TIMER IN pulses received. You cannot load an initial value of 1 into the count register and cause the timer to operate, as its terminal count value is 10 (binary) or 2 (decimal). (For the detection of single pulses, it is suggested that one of the hardware interrupt pins on the 8085AH be used.) After the timer has started counting down, the values residing in the count registers can be used to calculate the actual number of TIMER IN pulses required to complete the timer cycle if desired. To obtain the remaining count, perform the following operations in order:

1. Stop the count
2. Read in the 16-bit value from the count length registers
3. Reset the upper two mode bits
4. Reset the carry and rotate right one position all 16 bits through carry
5. If carry is set, add 1/2 of the full original count (1/2 full count — 1 if full count is odd).

Note: If you started with an odd count and you read the count length register before the third count pulse occurs, you will not be able to discern whether one or two counts has occurred. Regardless of this, the 8155H/56H always counts out the right number of pulses in generating the TIMER OUT waveforms.

8085A MINIMUM SYSTEM CONFIGURATION

Figure 13a shows a minimum system using three chips, containing:

- 256 Bytes RAM
- 2K Bytes ROM
- 38 I/O Pins
- 1 Interval Timer
- 4 Interrupt Levels

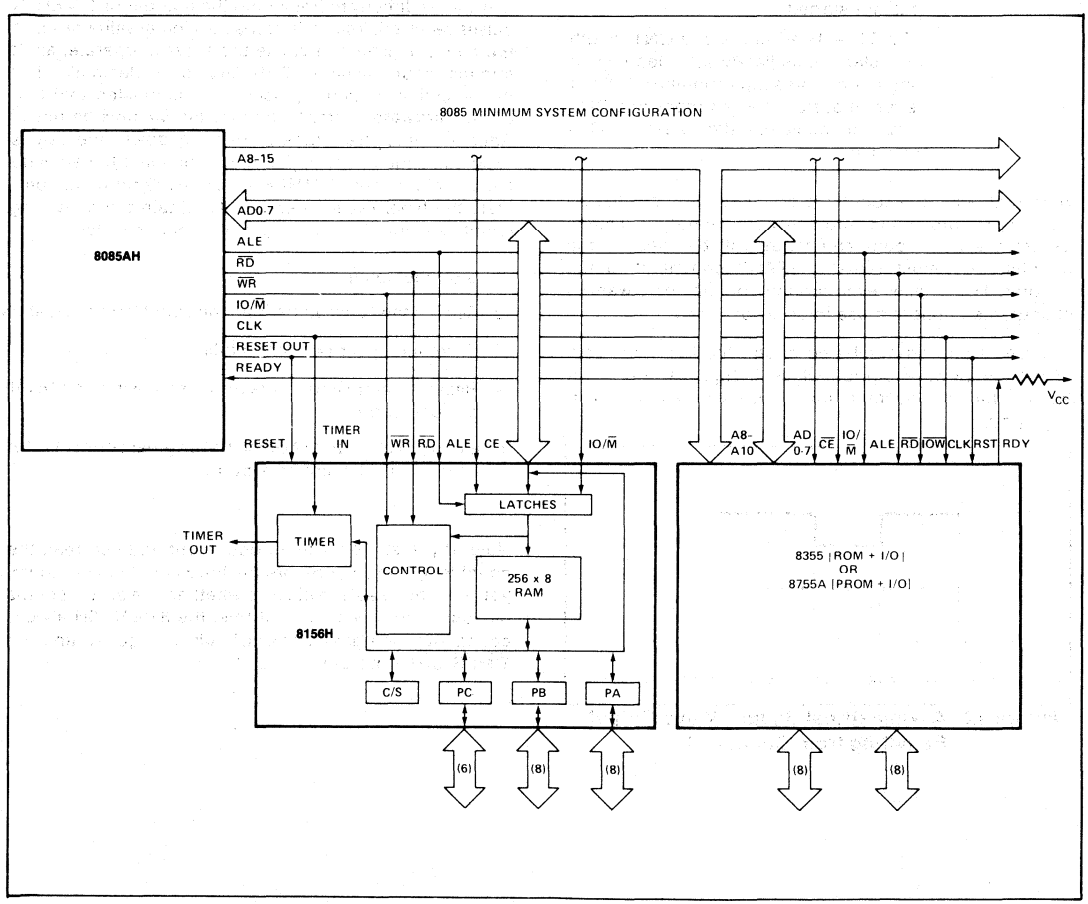


Figure 13a. 8085AH Minimum System Configuration (Memory Mapped I/O)

8088 FIVE CHIP SYSTEM

Figure 13b shows a five chip system containing:

- 1.25K Bytes RAM
- 2K Bytes ROM
- 38 I/O Pins
- 1 Interval Timer
- 2 Interrupt Levels

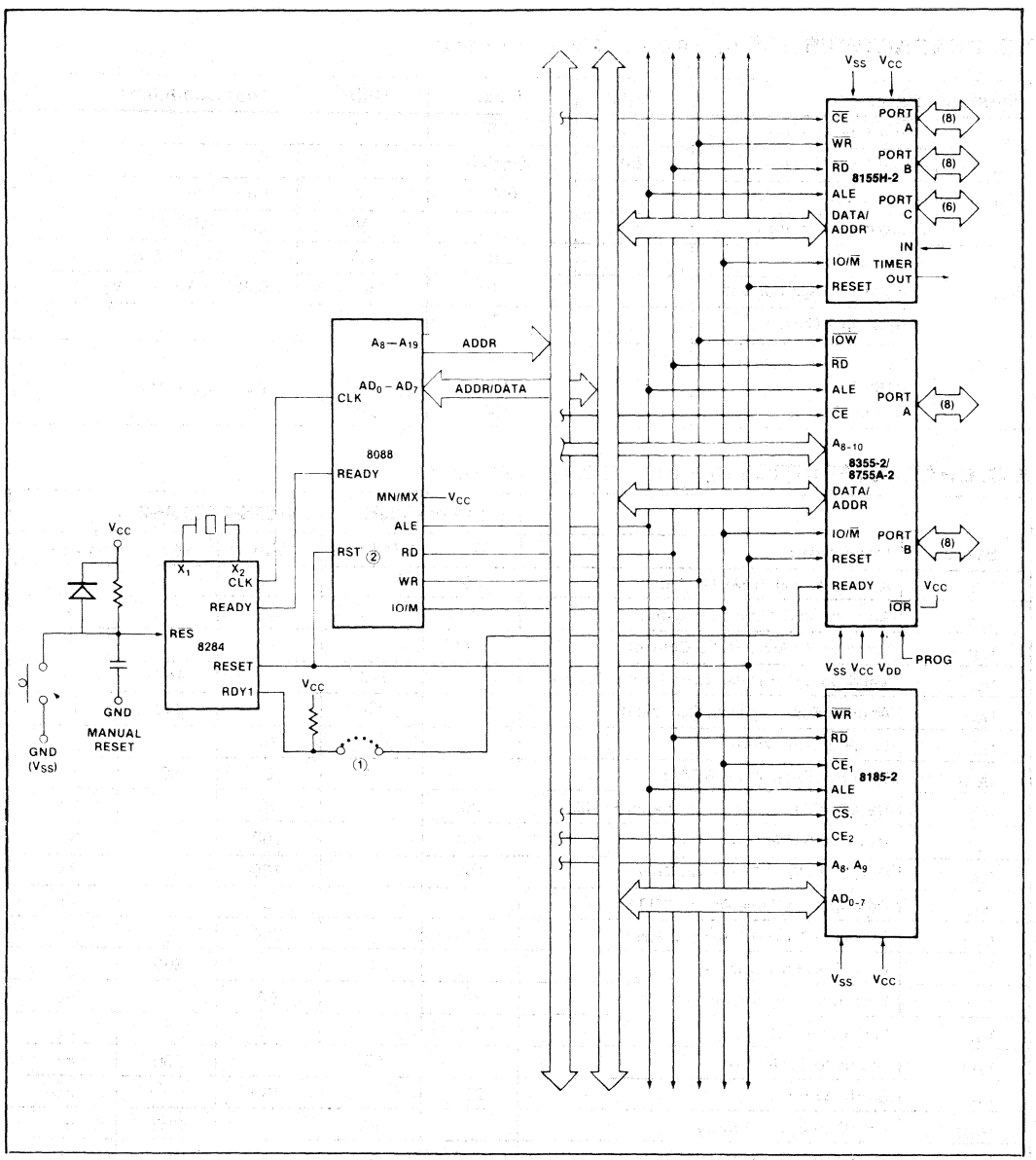


Figure 13b. 8088 Five Chip System Configuration

ABSOLUTE MAXIMUM RATINGS*

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 1.5W

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.	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_{OL}	Output Low Voltage		0.45	V	I_{OL} = 2mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = -400μA
I_{IL}	Input Leakage		±10	μA	$0V \leq V_{IN} \leq V_{CC}$
I_{LO}	Output Leakage Current		± 10	μA	$0.45V \leqslant V_{OUT} \leqslant V_{CC}$
I_{CC}	V_{CC} Supply Current		125	mA	
I_{IL}(CE)	Chip Enable Leakage 8155H 8156H		 +100 -100	 μA μA	$0V \leq V_{IN} \leq V_{CC}$

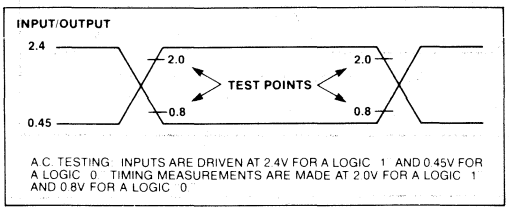
A.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5V ±10%)

		8155H/8156H		8155H-2/8156H-2		
Symbol	Parameter	Min.	Max.	Min.	Max.	Units
t_{AL}	Address to Latch Set Up Time	50		30		ns
t_{LA}	Address Hold Time after Latch	80		30		ns
t_{LC}	Latch to READ/WRITE Control	100		40		ns
t_{RD}	Valid Data Out Delay from READ Control		170		140	ns
t_{AD}	Address Stable to Data Out Valid		400		330	ns
t_{LL}	Latch Enable Width	100		70		ns
t_{RDF}	Data Bus Float After READ	0	100	0	80	ns
t_{CL}	READ/WRITE Control to Latch Enable	20		10		ns
t_{CC}	READ/WRITE Control Width	250		200		ns
t_{DW}	Data In to WRITE Set Up Time	150		100		ns
t_{WD}	Data In Hold Time After WRITE	0		0		ns
t_{RV}	Recovery Time Between Controls	300		200		ns
t_{WP}	WRITE to Port Output		400		300	ns
t_{PR}	Port Input Setup Time	70		50		ns
t_{RP}	Port Input Hold Time	50		10		ns
t_{SBF}	Strobe to Buffer Full		400		300	ns
t_{SS}	Strobe Width	200		150		ns
t_{RBE}	READ to Buffer Empty		400		300	ns
t_{SI}	Strobe to INTR On		400		300	ns

A.C. CHARACTERISTICS (Continued) (T_A = 0°C to 70°C, V_{CC} = 5V ±10%)

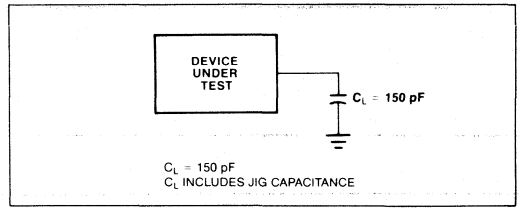
		8155H/8156H		8155H-2/8156H-2		
Symbol	**Parameter**	**Min.**	**Max.**	**Min.**	**Max.**	**Units**
t_{RDI}	READ to INTR Off		400		300	ns
t_{PSS}	Port Setup Time to Strobe Strobe	50		0		ns
t_{PHS}	Port Hold Time After Strobe	120		100		ns
t_{SBE}	Strobe to Buffer Empty		400		300	ns
t_{WBF}	WRITE to Buffer Full		400		300	ns
t_{WI}	WRITE to INTR Off		400		300	ns
t_{TL}	TIMER-IN to $\overline{\text{TIMER-OUT}}$ Low		400		300	ns
t_{TH}	TIMER-IN to $\overline{\text{TIMER-OUT}}$ High		400		300	ns
t_{RDE}	Data Bus Enable from READ Control	10		10		ns
t_1	TIMER-IN Low Time	80		40		ns
t_2	TIMER-IN High Time	120		70		ns

A.C. TESTING INPUT, OUTPUT WAVEFORM



A.C. 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".

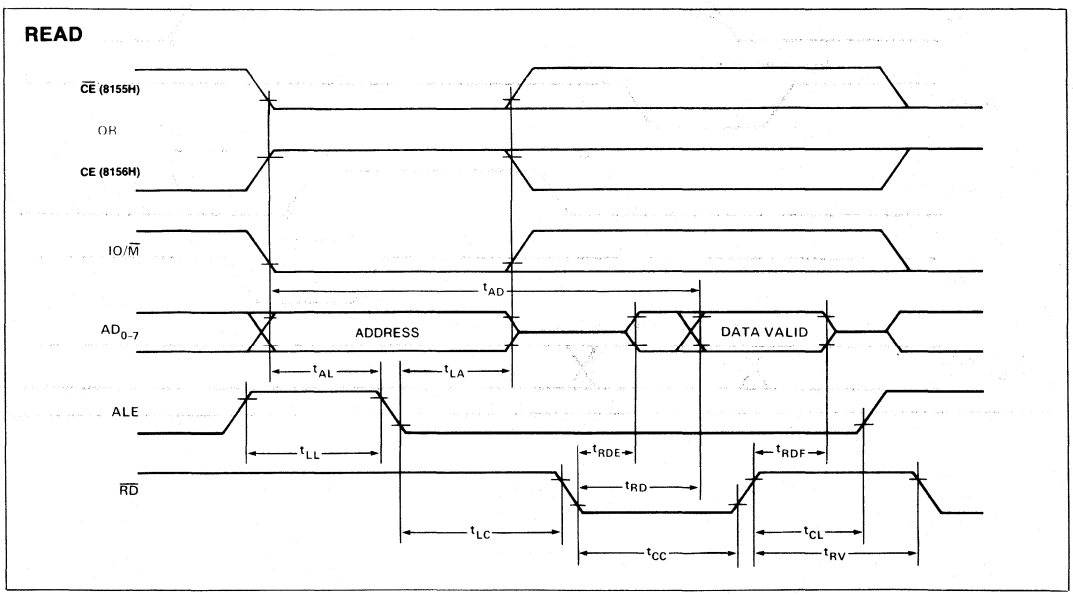
A.C. TESTING LOAD CIRCUIT



C_L = 150 pF
C_L INCLUDES JIG CAPACITANCE

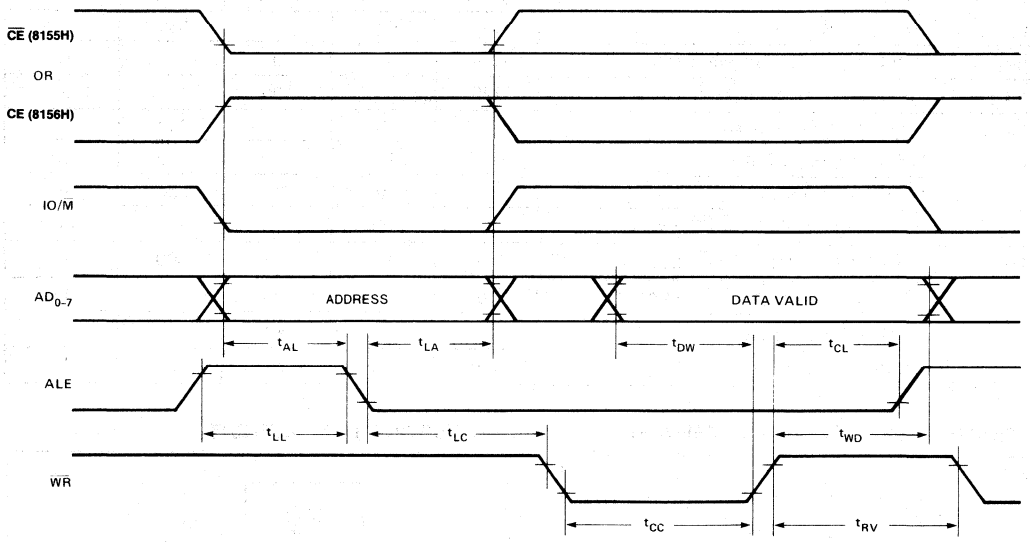
WAVEFORMS

READ

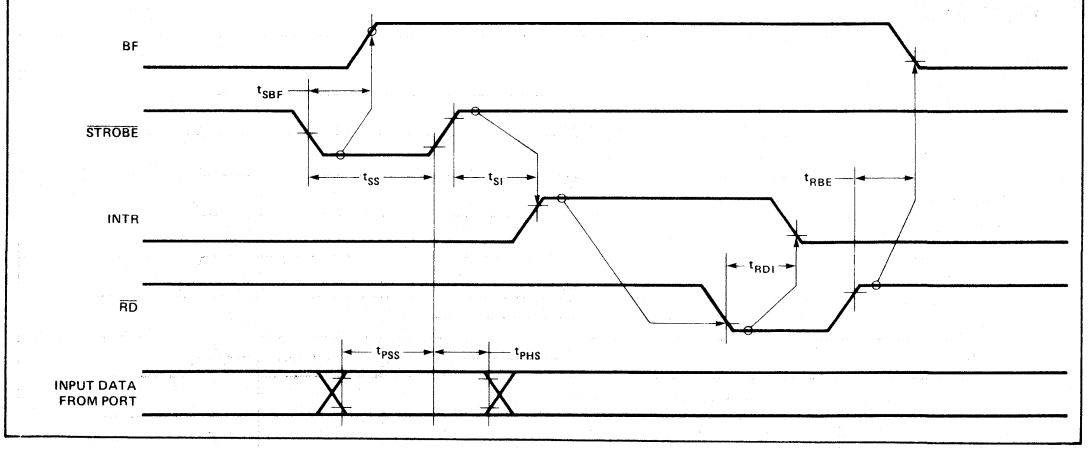


WAVEFORMS (Continued)

WRITE

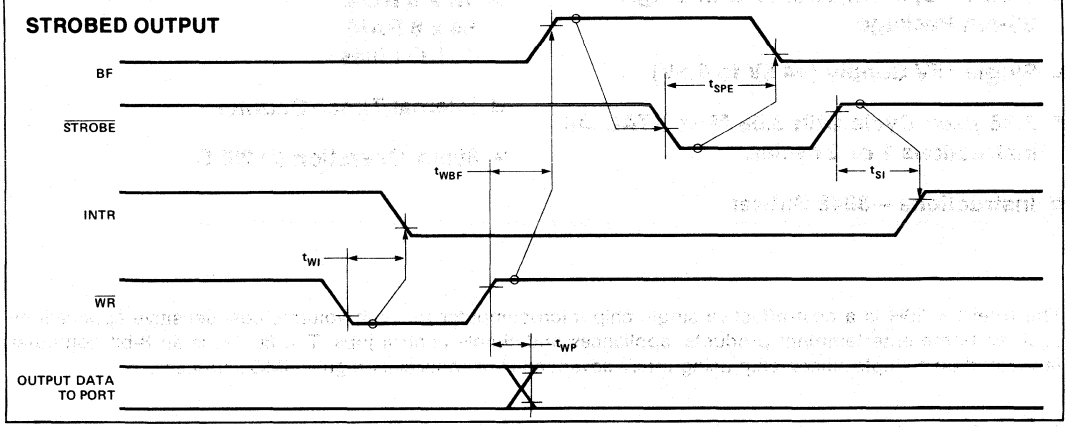


STROBED INPUT

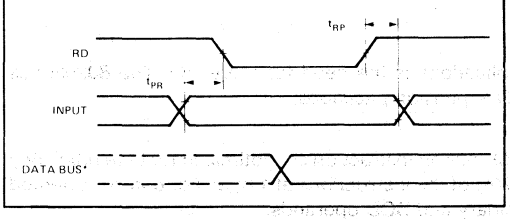


WAVEFORMS (Continued)

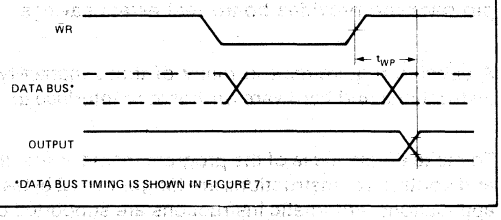
STROBED OUTPUT



BASIC INPUT

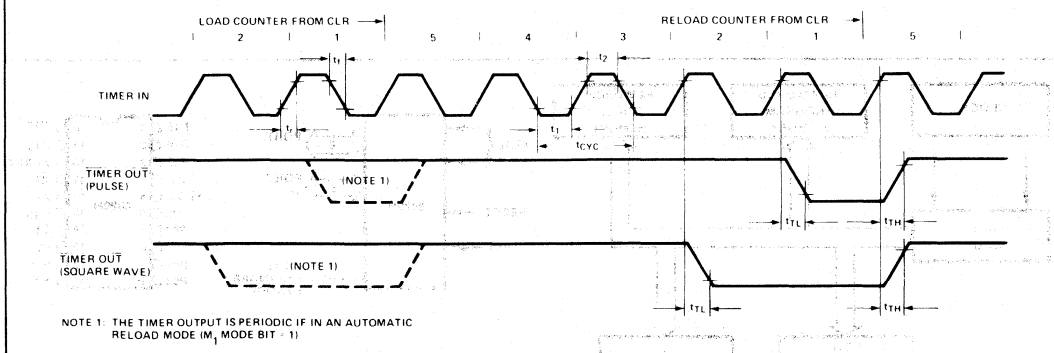


BASIC OUTPUT



*DATA BUS TIMING IS SHOWN IN FIGURE 7.7

TIMER OUTPUT COUNTDOWN FROM 5 TO 1



NOTE 1: THE TIMER OUTPUT IS PERIODIC IF IN AN AUTOMATIC RELOAD MODE (M_1 MODE BIT = 1)

8020H

HMOS SINGLE-COMPONENT 8-BIT MICROCOMPUTER

- **8-bit CPU, ROM, RAM, I/O in Single 20-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.**
- **Instructions—8048 Subset**
- **1K x 8 ROM**
  64 x 8 RAM
  13 I/O Lines
- **Internal Timer/Counter**
- **30mA Operation @ 25° C**

The Intel® 8020H is a cost-effective single-chip microcomputer for high volume, cost-sensitive applications such as home entertainment products, appliances and simple control jobs. The 8020H is an 8-bit computer fabricated on a single silicon chip using Intel's advanced N-channel silicon gate HMOS process.

The 8020H key features include a subset of the industry standard 8048's instruction set optimized for the consumer an appliance marketplaces. Some of these features are a 1K × 8 program memory, a 64 × 8 data memory, 13 I/O lines and an 8-bit timer/counter in addition to on-board oscillator and clock circuits. The 20-pin package provides board real estate savings.

A specific requirement in many of these market-type applications is the need for more I/O. The 8020H has instructions and hardware on-board to interface to TTL I/O expansion packages.

To make efficient use of the program memory size, the 8020H has an instruction set optimized for byte efficiency and control. No instructions are longer than 2 bytes, with 70% of the instructions at 1 byte. For control-oriented applications, arithmetic instructions are supported using binary and BCD operands.

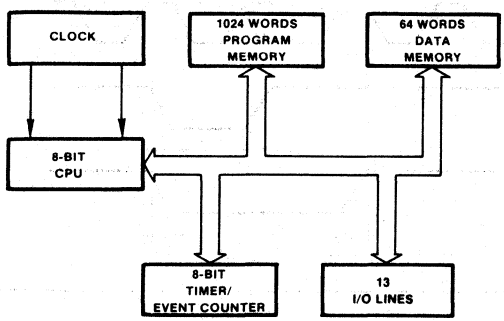


Figure 1. Block Diagram

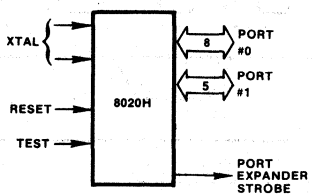


Figure 2. Logic Symbol

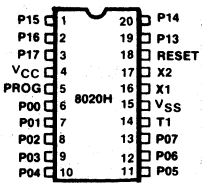


Figure 3. Pin Configuration

Table 1. Pin Description

Symbol	Pin No.	Function
V_{SS}	15	Circuit GND potential
V_{CC}	4	+5V power supply
PROG	5	Output strobe for TTL I/O expansion
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.
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

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_{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} + 0.45 ≤ 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	Vpp	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}

8020H FUNCTIONAL SPECIFICATIONS

The following is a functional description of the major elements of the 8020H.

Program Memory

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

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 4. 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 preset 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. The stack contents is incremented before being loaded into the program counter during a return from subroutine. 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 or decrement moves the address pointed to by two. Therefore, only even numbered addreses 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 stratchpad 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.

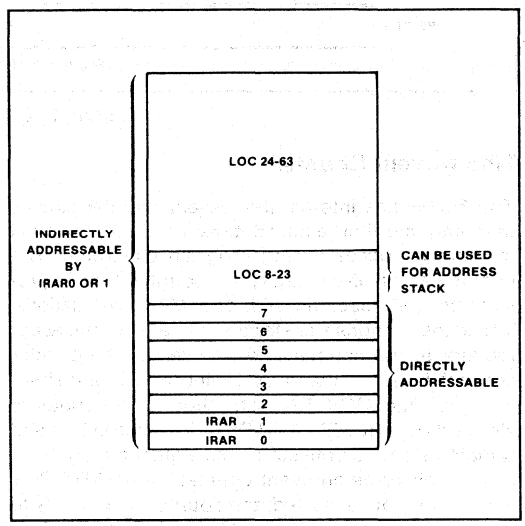


Figure 4. Internal RAM Organization.

Oscillator and Clock

The 8020H contains its own onboard oscillator and clock circuit, requiring only an external timing control element. This control element can be a crystal, inductor, or clock in. The capacitor normally required in inductor timing control operation is integrated onto the 8020H. All internal time slots are derived from the external element, and all outputs are a function of the oscillator frequency. Pins XTAL1 and XTAL2 are used to input the particular control element. An instruction cycle consists of 10 states, and each state is a time slot of 3 oscillator periods. (See Figure 5.) Therefore, to obtain a 10 μsec instruction cycle, a 3 MHz crystal should be used. An oscillator frequency of approximately 3 MHz may also be obtained by connecting a 470 μH inductor between XTAL1 and XTAL2. Note that the required inductance may vary and should be adjusted as necessary.

The 8020H 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 600K Hz, or improper operation may occur.

AFN-01935A-4

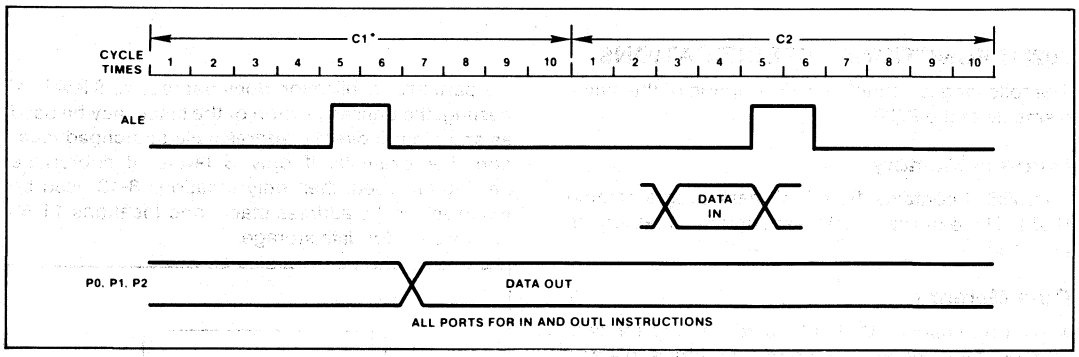


Figure 5. 8020H Timing Diagram

Timer/Event Counter

The 8020H has internal timer/event counter circuits that can monitor elapsed time or count external events that occur during program execution. The circuit 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 content is not affected by Reset, and is initialized solely 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 8748.

By a MOV T,A instruction, the contents of the accumulator are loaded to the timer. 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 are moved to the accumulator by the MOV A,T instruction without disturbing the counting process. The timer stops upon the STOP TCNT instruction.

The STRT CNT instruction connects the T1 input pin to the event counter input and enables the counter. Subsequent high-to-low transitions on T1 increment the counter. The maximum rate at which the counter can increment is once per three instruction cycles (30μs for a 3 MHz oscillator). There is no minimum frequency. T1 input must remain high for at least 500ns after each transition. The event counter is stopped by a STOP TCNT instruction.

Input/Output Capabilities

The 8020H I/O configurations are highly flexible. A number of different configurations are possible, tailoring an 8020H to a given task. Other than the power supply and dedicated pins, all other pins (13) can be used for input, output, or both, depending on the configuration.

All ports are quasi-bidirectional to facilitate standalone use. A simplified schematic of the quasi-bidirectional interface is shown in Figure 6. 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 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 large pulldown device sinks an external TTL load. When writing a "1", a large current is supplied through the large pullup device to allow a fast data transfer. After a short time (less than one instruction cycle), the large device is shut off and the small 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 large pullup device. As outputs, this port is essentially open drain.

AFN-01935A-5

By mask option the small 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.

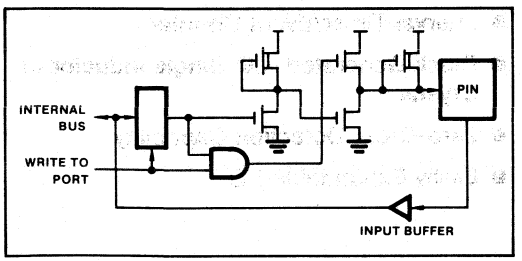


Figure 6. Quasi-Bidirectional Port Structure

T1 Input

The 8020H 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 7 permits the Test 1 input to detect when the input voltage crosses zero within ±5%; the voltage is then coupled through a 1.0μf 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 resistor that is useful for switch contact input or standard TTL.

The 8020H can use standard low cost TTL to expand the number of I/O lines.

CPU

The 8020H 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

Reset

A positive-going signal to the RESET input resets the necessary miscellaneous flip-flops and sets the program counter and stack pointer to zero.

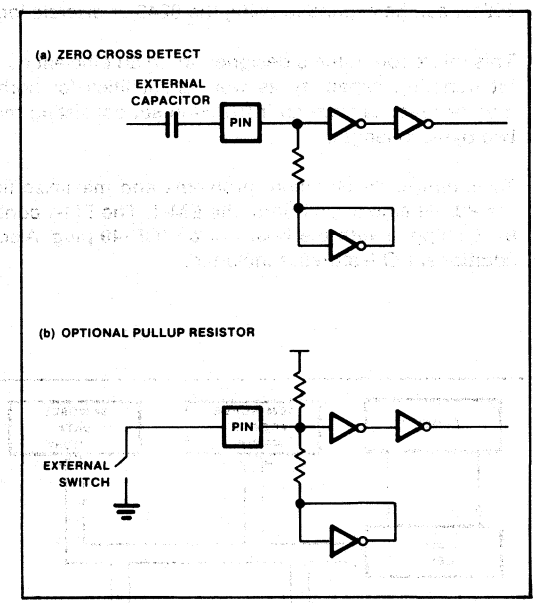


Figure 7. Test 1 Pin

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 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, a 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 8021H emulation board, the EM-1. The EM-1 contains a 40-pin socket which can accommodate either the 8748 shipped with the board or an ICE-49 plug. Also, the necessary discrete logic to reproduce the 8021H's additional I/O features is included.

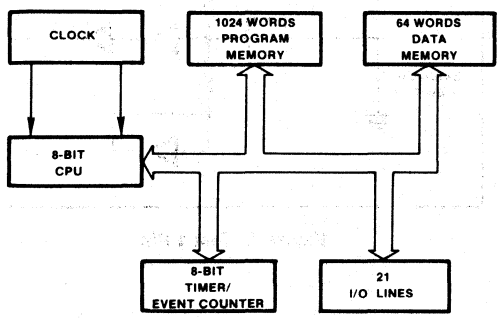


Figure 1. Block Diagram

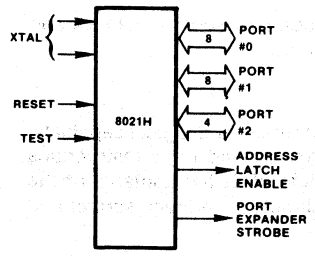


Figure 2. Logic Symbol

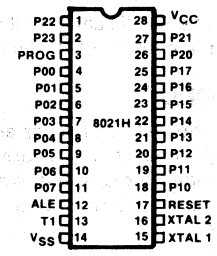


Figure 3. Pin Configuration

AFN-01936A-1

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.
ALE	12	Address Latch Enable. Signal occurring once every 30 input clocks, used as an output clock.
XTAL1	15	One side of crystal 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

AFN-01936A-2

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}

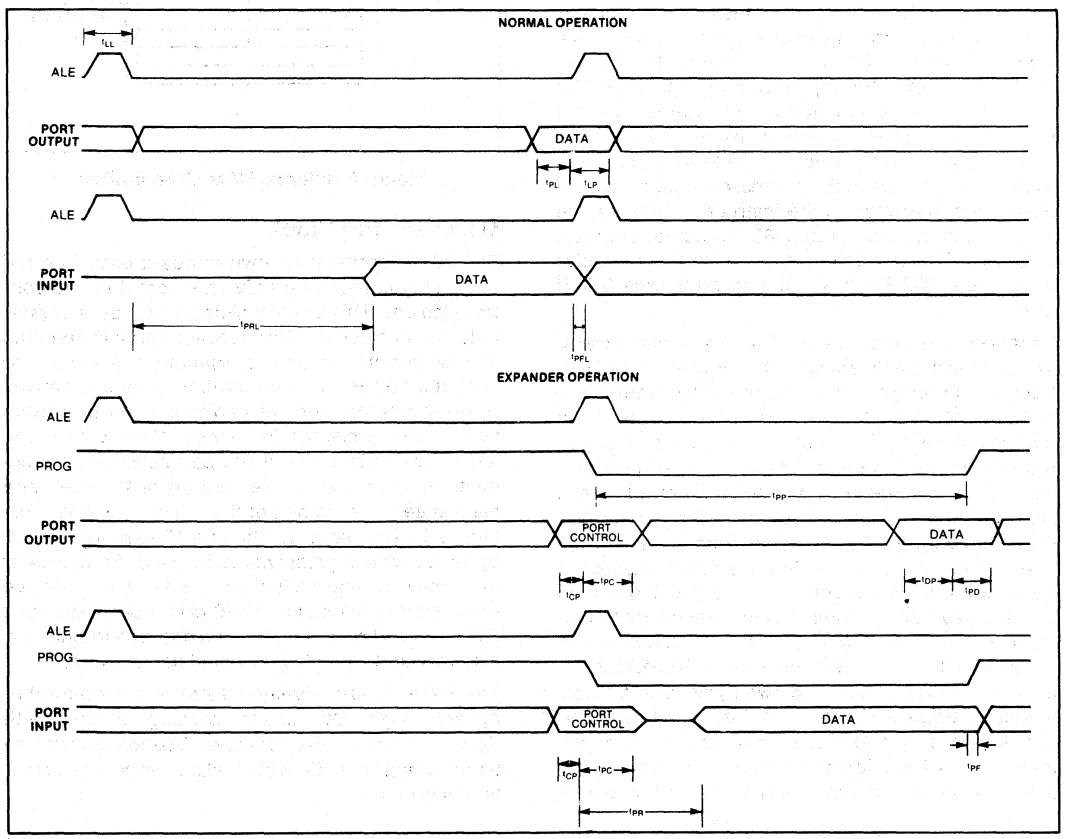
AFN-01936A-3

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)		4.0	µs	
	t_{PL}	Output Data Setup Time	o.6		µs	
	t_{LP}	Output Data Hold Time	0.6		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		
Expander Operation	t_{CP}	Port Control Setup Before Falling Edgle of PROG	0.3		µs	
	t_{CP}	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_{PD}	Output Data Hold Time	0.6		µs	
	t_{PF}	Input Data Hold Time	0	.15	µs	
	t_{PP}	PROG Pulse Width	6.0		µs	

PORT 2 TIMING



AFN-01936A-4

8021H FUNCTIONAL SPECIFICATIONS

The following is a functional description of the major elements of the 8021H.

Program Memory

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

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 4. 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 preset 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. The stack contents is incremented before being loaded into the program counter during a return from subroutine. 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 or 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.

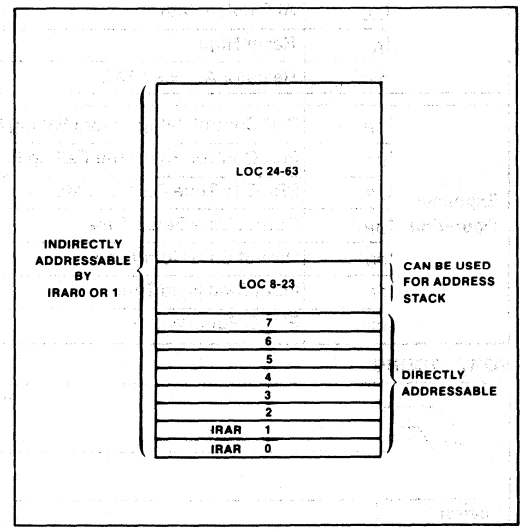


Figure 4. Internal RAM Organization

Oscillator and Clock

The 8021H contains its own onboard oscillator and clock circuit, requiring only an external timing control element. This control element can be a crystal, inductor, or clock in. The capacitor normally required in inductor timing control operation is integrated onto the 8021H. All internal time slots are derived from the external element, and all outputs are a function of the oscillator frequency. Pins XTAL1 and XTAL2 are used to input the particular control element. An instruction cycle consists of 10 states, and each state is a time slot of 3 oscillator periods. (See Figure 5.) Therefore, to obtain a 10 μsec instruction cycle, a 3 MHz crystal should be used. An oscillator frequency of approximately 3 MHz may also be obtained by connecting a 470 μH inductor between XTAL1 and XTAL2. Note that the required inductance may vary and should be adjusted as necessary.

The 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 600K Hz, or improper operation may occur.

AFN-01936A-5

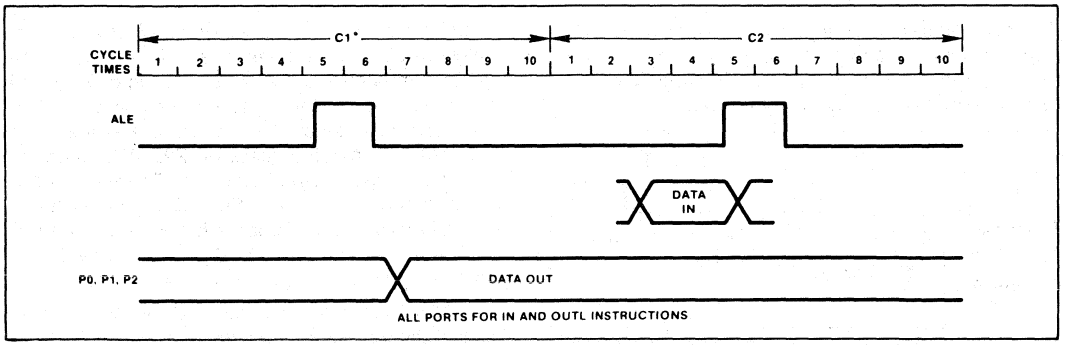


Figure 5. 8021H Timing Diagram

Timer/Event Counter

The 8021H has internal timer/event counter circuits that can monitor elapsed time or count external events that occur during program execution. The circuit 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 content is not affected by Reset, and is initialized solely 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 8748.

By a MOV T,A instruction, the contents of the accumulator are loaded to the timer. 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 are moved to the accumulator by the MOV A,T instruction without disturbing the counting process. The timer stops upon the STOP TCNT instruction.

The STRT CNT instruction connects the T1 input pin to the event counter input and enables the counter. Subsequent high-to-low transitions on T1 increment the counter. The maximum rate at which the counter can increment is once per three instruction cycles (30μs for a 3 MHz oscillator). There is no minimum frequency. T1 input must remain high for at least 500ns after each transition. The event counter is stopped by a STOP TCNT instruction.

Input/Output Capabilities

The 8021H I/O configurations are highly flexible. A number of different configurations are possible, tailoring an 8021H to a given task. Other than the power supply and dedicated pins, all other pins (20) can be used for input, output, or both, depending on the configuration.

All ports are quasi-bidirectional to facilitate standalone use. A simplified schematic of the quasi-bidirectional interface is shown in Figure 6. 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 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 large pulldown device sinks an external TTL load. When writing a "1", a large current is supplied through the large pullup device to allow a fast data transfer. After a short time (less than one instruction cycle), the large device is shut off and the small 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 large pullup device. As outputs, this port is essentially open drain.

AFN-01936A-6

By mask option the small 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.

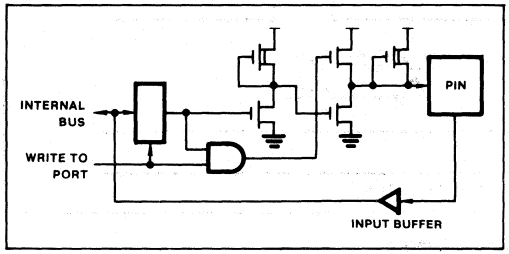


Figure 6. Quasi-Bidirectional Port Structure

T1 Input

The 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 7 permits the Test 1 input to detect when the input voltage crosses zero within ±5%; the voltage is then coupled through a 1.0μF 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 resistor that is useful for switch contact input or standard TTL.

High Current Outputs

Very 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

The 8021H 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 8. The timing diagram is shown in Figure 9.

The 8021H can also use standard low-cost TTL to expand the number of I/O lines.

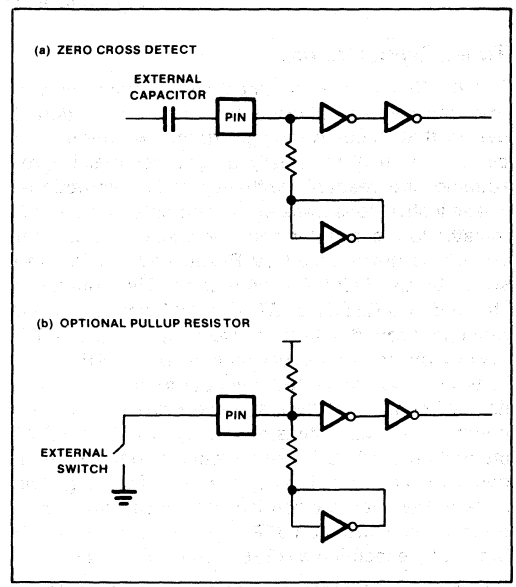


Figure 7. Test 1 Pin

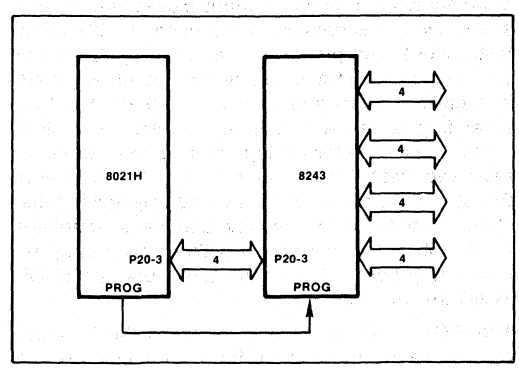


Figure 8. I/O Expander Interface

AFN-01936A-7

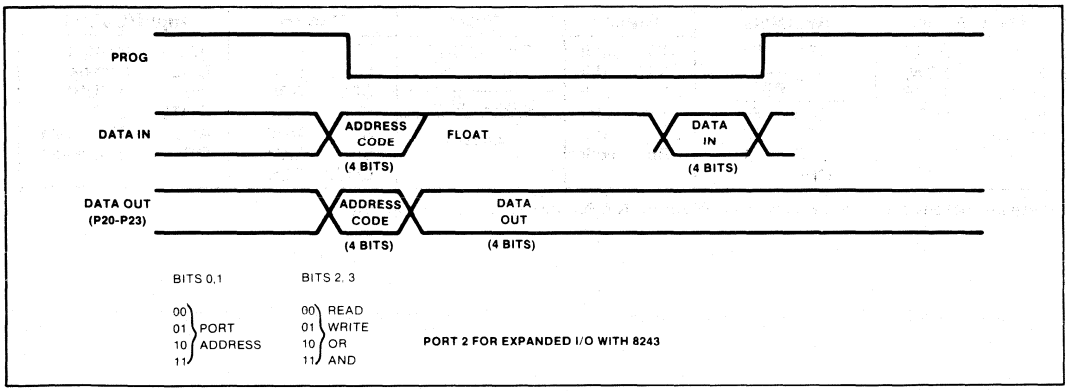


Figure 9. Expanded I/O Timing Diagram

CPU

The 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

Reset

A positive-going signal to the RESET input resets the necessary miscellaneous flip-flops and sets the program counter and stack pointer to zero.

DIFFERENCES BETWEEN THE 8021H AND THE 8748

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

1. *Pin Out*—As the 8021H is a 28-pin DIP, some form of adapter must be used to interface the 8021H socket to ICE-49. An emulation board, EM-1, has been designed to perform this function. The EM-1 also accounts for the increased flexibility of some 8021H I/O lines.
2. *Instruction Time*—The 8021H instruction cycle is 30 clock cycles long, the 8748 instruction cycle is 15 clocks long. Where exact timing is important, the 8748 breadboard part should be operated at half the 8021H clock rate.
3. *Test 1*—To facilitate developing time of day routines from 60 Hz, 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 resistors.
5. *Oscillator*—The 8021H has on-chip oscillator that is optimized for the single resistor mode. External connection will differ from the 8748.
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*—Very 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. *Instruction Set*—The following instructions, which are found in the 8748, 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	CLR	F0	JF0	addr	**Subroutine**		SEL	RB0	INS	A,BUS*
MOVX	@R,A	CPL	F0	JF1	addr	RETR		SEL	RB1	OUTL	BUS,A*
MOVP3	A,@A	CLR	F1	JNI	addr			SEL	MB0	ANL	BUS,#data
		CPL	F1	JBb	addr			SEL	MB1	ORL	BUS,#data
								ENT0	CLK		

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

AFN-01936A-9

ADVANCE INFORMATION

8022H
HIGH PERFORMANCE 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)
- Two Interrupts—External and Timer
- 2K x 8 ROM, 64 x 8 RAM, 28 I/O Lines
- 5 μsec Cycle; All Instructions 1 or 2 Cycles (6 MHz Clock)
- Instructions—8048 Subset
- Interval Time/Event Counter
- Clock Generated with Single Inductor or Crystal
- Easily Expanded I/O

The Intel® 8022H 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 8022H 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 8022H 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 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 8022H 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 8022H's hardware implementation of the A/D converter which simplifies interfacing to analog signals.

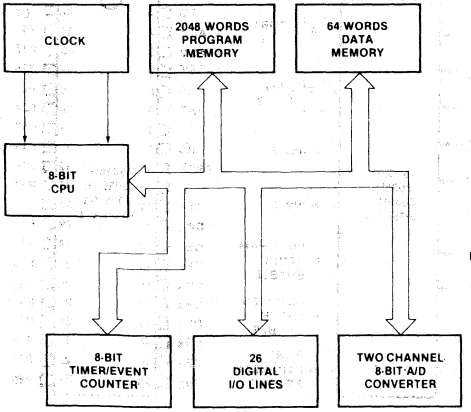


Figure 1. Block Diagram

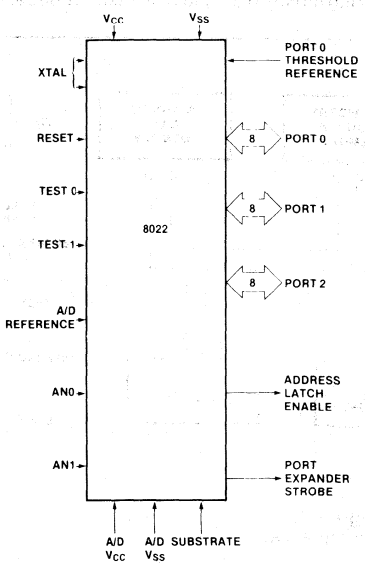


Figure 2. Logic Symbol

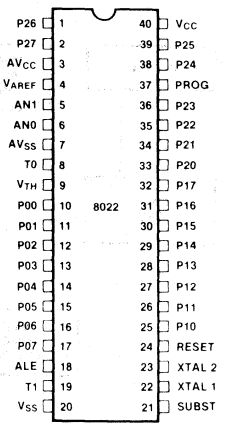


Figure 3. Pin Configuration

AFN-01814A

PRELIMINARY

8048H/8048H-1/8035HL/8035HL-1
HMOS SINGLE COMPONENT 8-BIT MICROCOMPUTER

- 8048H/8048H-1 Mask Programmable ROM
- 8035HL/8035HL-1 CPU Only with Power Down Mode

- 8-BIT CPU, ROM, RAM, I/O in Single Package
- High Performance HMOS
- Reduced Power Consumption
- 1.4 µsec and 1.9 µsec Cycle Versions All Instructions 1 or 2 Cycles
- Over 90 Instructions: 70% Single Byte
- 1K x 8 ROM 64 x RAM 27 I/O Lines
- Interval Timer/Event Counter
- Easily Expandable Memory and I/O
- Compatible with 8080/8085 Series Peripherals
- Two Single Level Interrupts

The Intel® 8048H/8048H-1/8035HL/8035HL-1 are totally self-sufficient, 8-bit parallel computers fabricated on single silicon chips using Intel's advanced N-channel silicon gate HMOS process.

The 8048H contains a 1K X 8 program memory, a 64 X 8 RAM data memory, 27 I/O lines, and an 8-bit timer/counter in addition to on-board oscillator and clock circuits. For systems that require extra capability the 8048H can be expanded using standard memories and MCS-80® /MCS-85® peripherals. The 8035HL is the equivalent of the 8048H without program memory and can be used with external ROM and RAM.

To reduce development problems to a minimum and provide maximum flexibility, a logically and functionally pin compatible version of the 8048H with UV-erasable user-programmable EPROM program memory is available. The 8748 will emulate the 8048H up to 6 MHz clock frequency with minor differences.

The 8048H is fully compatible with the 8048 when operated at 6MHz.

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.

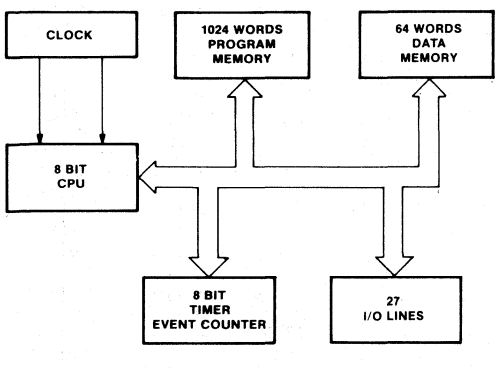


Figure 1. Block Diagram

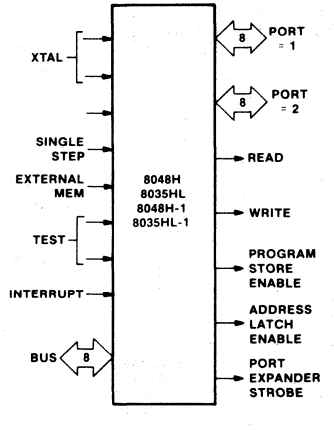


Figure 2. Logic Symbol

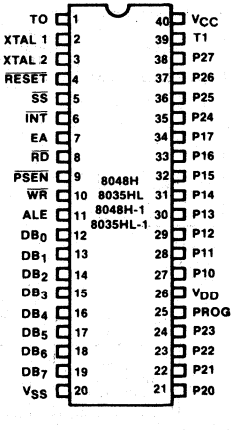


Figure 3. Pin Configuration (top view)

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 8243 I/O expander.
P10-P17 Port 1	27-34	8-bit quasi-bidirectional port.
P20-P27 Port 2	21-24	8-bit quasi-bidirectional port.
	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 $\overline{RD}$, $\overline{WR}$ 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 ENT0 CLK instruction.
T1	39	Input pin testable using the JT1, and JNT1 instructions. Can be designated the timer/counter input using the STRT CNT instruction.
$\overline{INT}$	6	Interrupt input. Initiates an interrupt if interrupt is enabled. Interrupt is disabled after a reset.
		Also testable with conditional jump instruction. (Active low)
$\overline{RD}$	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)
$\overline{RESET}$	4	Input which is used to initialize the processor. (Active low) (Non TTL V_{IH})
$\overline{WR}$	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.
$\overline{PSEN}$	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)
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.

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
JTO addr	Jump on TO = 1	2	2
JNTO addr	Jump on TO = 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
JN1 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
RETR	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

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	
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 counter	1	1
STOP TCNT	Stop timer/counter	1	1
EN TCNT1	Enable timer/counter interrupt	1	1
DIS TCNT1	Disable timer/counter interrupt	1	1

Control

Mnemonic	Description	Bytes	Cycles
EN 1	Enable external interrupt	1	1
DIS 1	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
ENT 0 CLK	Enable clock output on T0	1	1

Mnemonic	Description	Bytes	Cycles
NOP	No operation	1	1

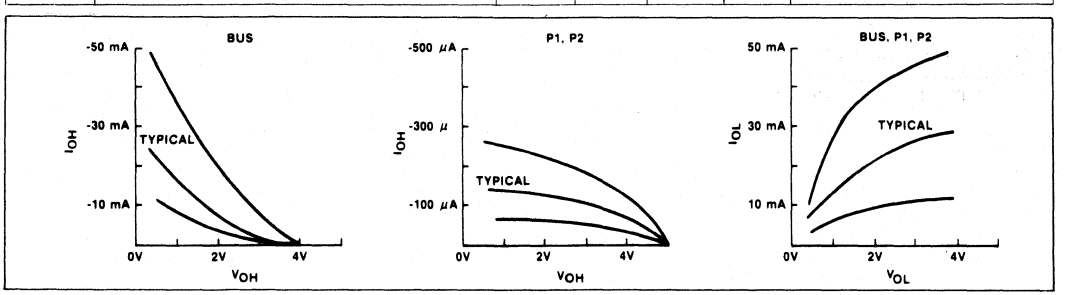
ABSOLUTE MAXIMUM RATINGS*

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 ..... 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
		Min.	Typ.	Max.		
V_{IL}	Input Low Voltage (All Except $\overline{RESET}$, X1, X2)	-.5		.8	V	
V_{IL1}	Input Low Voltage ($\overline{RESET}$, X1, X2)	-.5		.6	V	
V_{IH}	Input High Voltage (All Except XTAL1, XTAL2, $\overline{RESET}$)	2.0		V_{CC}	V	
V_{IH1}	Input High Voltage (X1, X2, $\overline{RESET}$)	3.8		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
VOH	Output High Voltage (BUS)	2.4			V	I_{OH} = -400μA
V_{OH1}	Output High Voltage ($\overline{RD}$, $\overline{WR}$, $\overline{PSEN}$, ALE)	2.4			V	I_{OH} = -100μA
V_{OH2}	Output High Voltage (All Other Outputs)	2.4			V	I_{OH} = -40μA
I_{L1}	Input Leakage Current (T1, $\overline{INT}$)			±10	μA	$V_{SS} \leq V_{IN} \leq V_{CC}$
I_{LI1}	Input Leakage Current (P10-P17, P20-P27, EA, $\overline{SS}$)			-500	μA	V_{SS} + .45$\leq V_{IN} \leq V_{CC}$
I_{L0}	Output Leakage Current (BUS, TO) (High Impedance State)			±10	μA	V_{SS} + .45$\leq V_{IN} \leq V_{CC}$
I_{DD}	V_{DD} Supply Current		4	8	mA	
I_{DD} + I_{CC}	Total Supply Current		40	80	mA	
V_{DD}	RAM Standby Pin Voltage	2.2		5.5	V	Standby Mode, Reset ≤0.6V



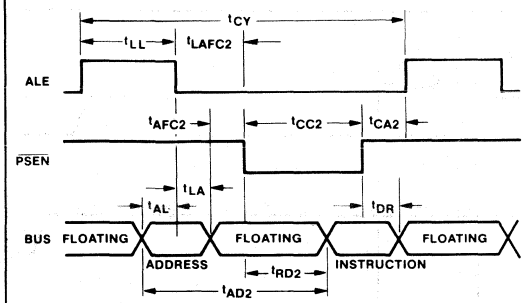
AFN-01491B-04

A.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = V_{DD} = 5V ± 10%, V_{SS} = 0V)

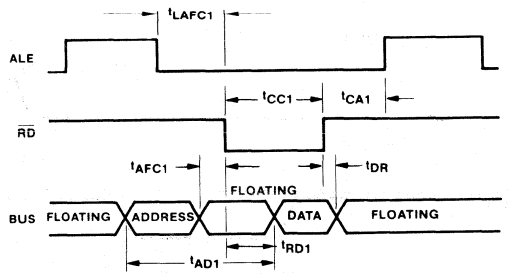
Symbol	Parameter	F (t_{CY})	8048H 8035HL 6 MHz Min.	8048H 8035HL 6 MHz Max.	8048H 8035HL 8 MHz Min.	8048H 8035HL 8 MHz Max.	8048H-1 8035HL-1 11 MHz Min.	8048H-1 8035HL-1 11 MHz Max.	Unit	Conditions (Note 1)
t_{LL}	ALE Pulse Width	7/30 t_{CY} -170	410		260		150			
t_{AL}	Addr Setup to ALE	1/5 t_{CY} -110	390		260		160			
t_{LA}	Addr Hold from ALE	1/15 t_{CY} -40	120		80		50			
t_{CC1}	Control Pulse Width ($\overline{RD}$, $\overline{WR}$)	1/2 t_{CY} -200	1050		730		480			
t_{CC2}	Control Pulse Width ($\overline{PSEN}$)	2/5 t_{CY} -200	800		550		350			
t_{DW}	Data Setup before $\overline{WR}$	13/30 t_{CY} -200	880		610		390			
t_{WD}	Data Hold after $\overline{WR}$	1/15 t_{CY} -50	350		220		40			(Note 2)
t_{DR}	Data Hold ($\overline{RD}$, $\overline{PSEN}$)	1/10 t_{CY} -30	0	220	0	160	0	110		
t_{RD1}	$\overline{RD}$ to Data in	2/5 t_{CY} -200		800		550		350		
t_{RD2}	$\overline{PSEN}$ to Data in	3/10 t_{CY} -200		550		360		210		
t_{AW}	Addr Setup to $\overline{WR}$	2/5 t_{CY} -150	850		600		300			
t_{AD1}	Addr Setup to Data ($\overline{RD}$)	23/30 t_{CY} -250		1670		1190		750		
t_{AD2}	Addr Setup to Data ($\overline{PSEN}$)	3/5 t_{CY} -250		1250		880		480		
t_{AFC1}	Addr Float to $\overline{RD}$, $\overline{WR}$	2/15 t_{CY} -40	290		210		140			
t_{AFC2}	Addr Float to $\overline{PSEN}$	1/30 t_{CY} -40	40		20		10			
t_{LAFC1}	ALE to Control ($\overline{RD}$, $\overline{WR}$)	1/5 t_{CY} -75	420		300		200			
t_{LAFC2}	ALE to Control ($\overline{PSEN}$)	1/10 t_{CY} -75	170		110		60			
t_{CA1}	Control to ALE ($\overline{RD}$, $\overline{WR}$, $\overline{PROG}$)	1/15 t_{CY} -40	120		80		50			
t_{CA2}	Control to ALE ($\overline{PSEN}$)	4/15 t_{CY} -40	620		460		320			
t_{CP}	Port Control Setup to $\overline{PROG}$	2/15 t_{CY} -80	210		140		100			
t_{PC}	Port Control Hold to $\overline{PROG}$	4/15 t_{CY} -200	460		300		160			
t_{PR}	$\overline{PROG}$ to P2 Input Valid	6/10 t_{CY} -120		1300		940		700		
t_{PF}	Input Data Hold from $\overline{PROG}$	1/10 t_{CY}		250	0	190	0	140		
t_{DP}	Output Data Setup	2/5 t_{CY} -150	850		600		400			
t_{PD}	Output Data Hold	1/10 t_{CY} -50	200		130		90			
t_{PP}	$\overline{PROG}$ Pulse Width	7/10 t_{CY} -250	1500		1060		700			
t_{PL}	Port 2 I/O Setup to ALE	4/15 t_{CY} -200	460		300		160			
t_{LP}	Port 2 I/O Hold to ALE	1/10 t_{CY} -100	150		80		40			
t_{PV}	Port Output from ALE	3/10 t_{CY} +100		850		660		510		
t_{CY}	Cycle Time		2.5		1.875		1.36			
t_{0PRR}	T0 Rep Rate	3/15 t_{CY}	500		370		270			

Notes:
1. Control Outputs CL = 80pF
   BUS Outputs CL = 150pF
2. BUS High Impedance Load 20pF

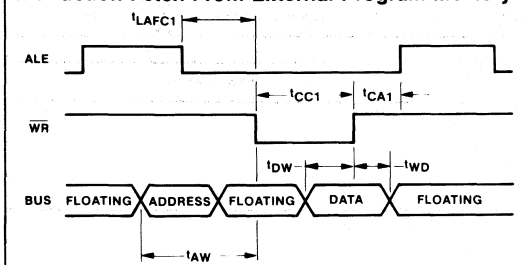
WAVEFORMS



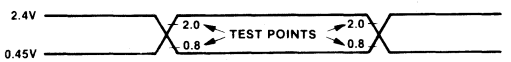
Instruction Fetch From External Program Memory



Read From External Data Memory

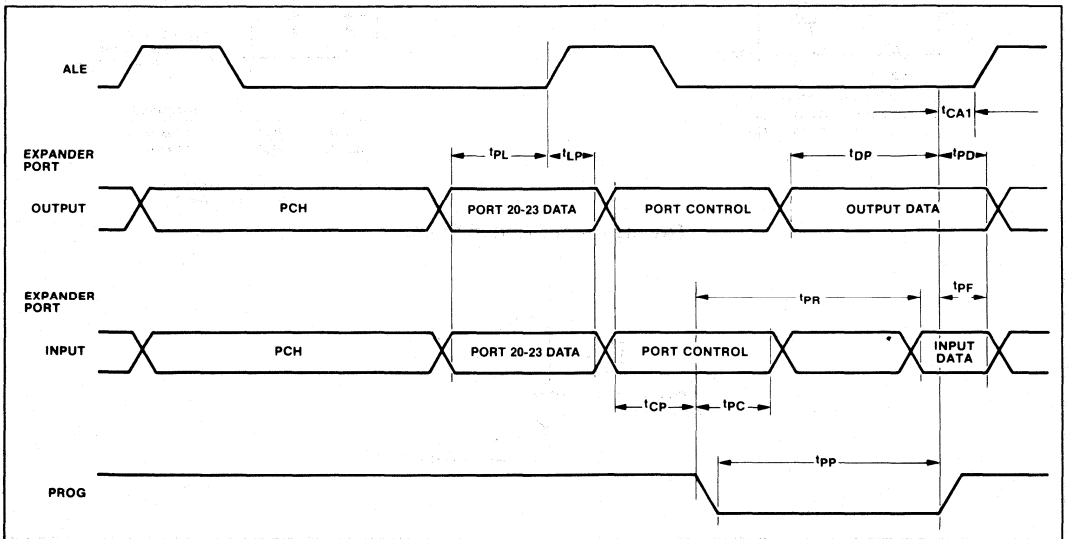


Write to External Data Memory



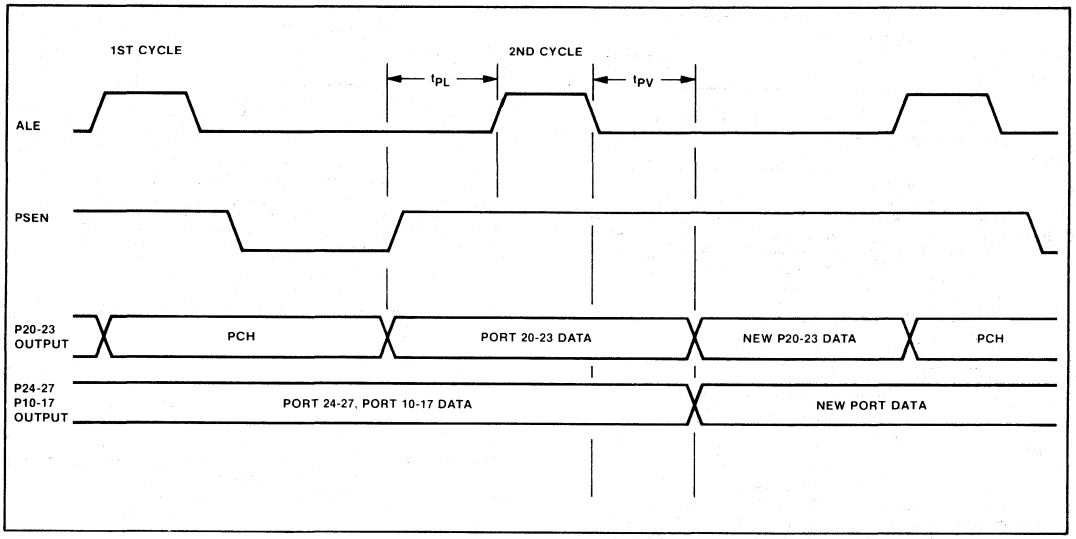
Input and Output for A.C.Tests

PORT 2 TIMING

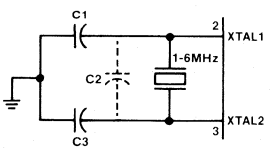


AFN-01491B-06

I/O PORT TIMING

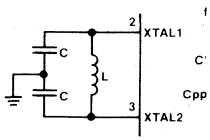


Crystal Oscillator Mode



C1 5pF + 1/2pF + STRAY 5pF
C2 CRYSTAL + STRAY 8pF
C3 20pF + 1pF + STRAY 5pF

LC Oscillator Mode



$f \approx \frac{1}{2\pi\sqrt{LC'}}$

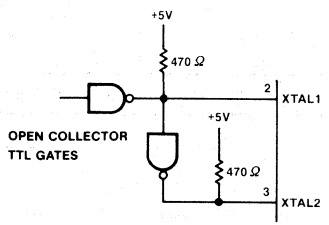
$C' \frac{C \quad 3Cpp}{2}$

Cpp ≈ 5-10 pF PIN-TO-PIN CAPACITANCE

L	C	NOMINAL f
45 μH	20pF	5.2 MHz
120 μH	20pF	3.2 MHz

EACH C SHOULD BE APPROXIMATELY 20pF. INCLUDING STRAY CAPACITANCE

Driving From External Source



AFN-01491B-07

PRELIMINARY

8048L
SPECIAL LOW POWER CONSUMPTION SINGLE COMPONENT 8-BIT MICROCOMPUTER

- **Typical Power Consumption 100mW**
- **Typical Standby Power 10mW**
  V_{DD} minimum of 2.2V
- **8-Bit CPU, ROM, RAM, I/O in Single Package**
- **4.17 μsec Instruction Cycle.**
  All Instructions 1 or 2 Cycles.
- **Over 90 Instructions: 70% Single Byte**
- **1K x 8 ROM**
  64 x 8 RAM
  27 I/O Lines
- **Interval Timer/Event Counter**
- **Easily Expandable Memory and I/O**
- **Compatible with 8080/8085 Series Peripherals**
- **Two Single Level Interrupts**

The Intel® 8048L 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, using special techniques to reduce operating and standby power consumption. The 8048L contains a 1K X 8 program memory, a 64 X 8 RAM data memory, 27 I/O lines, and an 8-bit timer/counter in addition to on-board oscillator and clock circuits. For systems that require extra capability the 8048L can be expanded using standard memories and MCS-80®/MCS-85® peripherals. The 8048L can be used with external ROM and RAM.

To reduce development problems to a minimum and provide maximum flexibility, a logically and functionally pin compatible version of the 8048L with UV-erasable user-programmable EPROM program memory is available. The 8748 will emulate the 8048L with greater power and other minor differences.

This microcontroller is designed to be an efficient controller as well as an arithmetic processor. The 8048L has 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 two bytes in length.

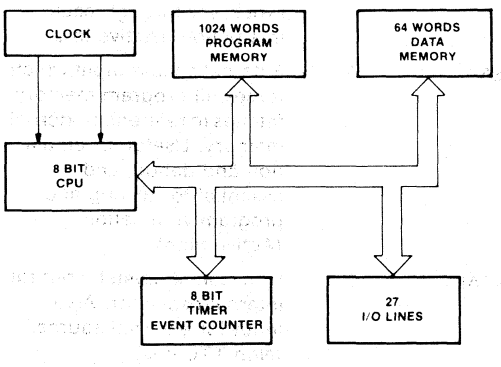


Figure 1.
8048L Block Diagram

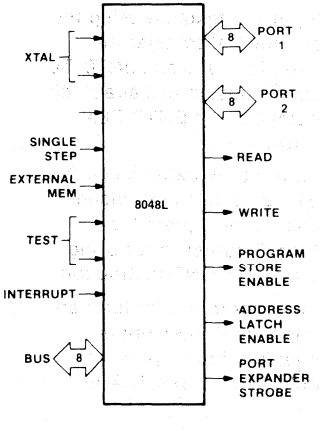


Figure 2.
8048L Logic Symbol

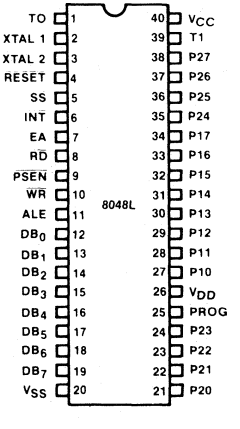


Figure 3.
8048L Pin Configuration

AFN-01591A-01

PIN DESCRIPTION

Designation	Pin =	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 8243 I/O expander.
P10-P17 Port 1	27-34	8-bit quasi-bidirectional port.
P20-27 Port 2	21-24	8-bit quasi-bidirectional port.
	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 $\overline{RD}$, $\overline{WR}$ 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 ENT0 CLK instruction.
T1	39	Input pin testable using the JT1, and JNT1 instructions. Can be designated the timer/counter input using the STRT CNT instruction.
$\overline{INT}$	6	Interrupt input. Initiates an interrupt if interrupt is enabled. Interrupt is disabled after a reset. Also testable with conditional jump instruction. (Active low)
$\overline{RD}$	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)
$\overline{RESET}$	4	Input which is used to initialize the processor. (Active low) (Non TTL V_{IH})
$\overline{WR}$	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.
$\overline{PSEN}$	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)
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.

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
JTO addr	Jump on TO = 1	2	2
JNTO addr	Jump on TO = 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
JN1 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

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	
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 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

Mnemonic	Description	Bytes	Cycles
EN 1	Enable external interrupt	1	1
DIS 1	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
ENT 0 CLK	Enable clock output on T0	1	1

Mnemonic	Description	Bytes	Cycles
NOP	No operation	1	1

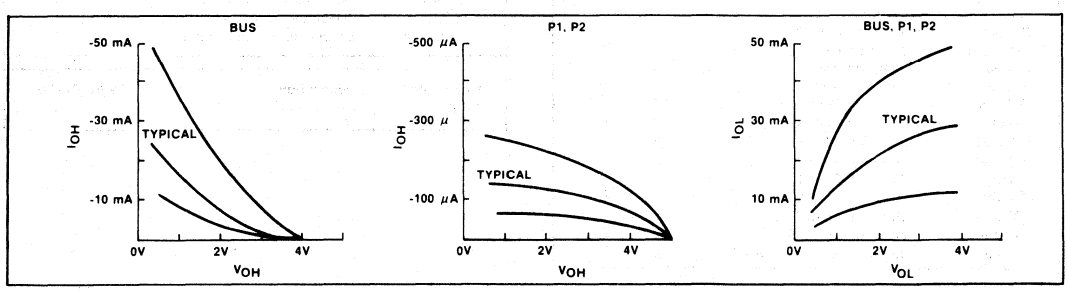
ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias ..... 0°C to 70°C
Storage Temperature ..... -65°C to + 125°C
Voltage On Any Pin With Respect to Ground ..... -0.5V to +7V
Power Dissipation ..... 1.5 Watt

** COMMENT 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. AND OPERATING CHARACTERISTICS TA = 0°C to 70°C, VCC = VDD = 5V ± 10%, VSS = 0V

Symbol	Parameter	Limits			Unit	Test Conditions
		Min.	Typ.	Max.		
V_{IL}	Input Low Voltage (All Except $\overline{RESET}$, X1, X2)	-.5		.8	V	
V_{IL1}	Input Low Voltage ($\overline{RESET}$, X1, X2)	-.5		.6	V	
V_{IH}	Input High Voltage (All Except XTAL1, XTAL2, $\overline{RESET}$)	2.0		V_{CC}	V	
V_{IH1}	Input High Voltage (X1, X2, $\overline{RESET}$)	3.8		V_{CC}	V	
V_{OL}	Output Low Voltage (BUS)			.45	V	V_{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)	2.4			V	I_{OH} = -400 μA
V_{OH1}	Output High Voltage ($\overline{RD}$, $\overline{WR}$, $\overline{PSEN}$, ALE)	2.4			V	I_{OH} = -100 μA
V_{OH2}	Output High Voltage (All Other Outputs)	2.4			V	I_{OH} = -40 μA
I_{L1}	Input Leakage Current (T1, $\overline{INT}$)			± 10	μA	$V_{SS} \leq V_{IN} \leq V_{CC}$
I_{LI1}	Input Leakage Current (P10-P17, P20-P27, EA, $\overline{SS}$)			-500	μA	$V_{SS} + .45 \leq V_{IN} \leq V_{CC}$
I_{L0}	Output Leakage Current (BUS, TO) (High Impedance State)			± 10	μA	$V_{SS} + .45 \leq V_{IN} \leq V_{CC}$
I_{DD}	V_{DD} Supply Current		2	4	mA	
$I_{DD} + I_{CC}$	Total Supply Current		20	40	mA	
V_{DD}	Ram Standby Pin Voltage	2.2		5.5	V	Standby Mode, Reset ≤ 0.6V



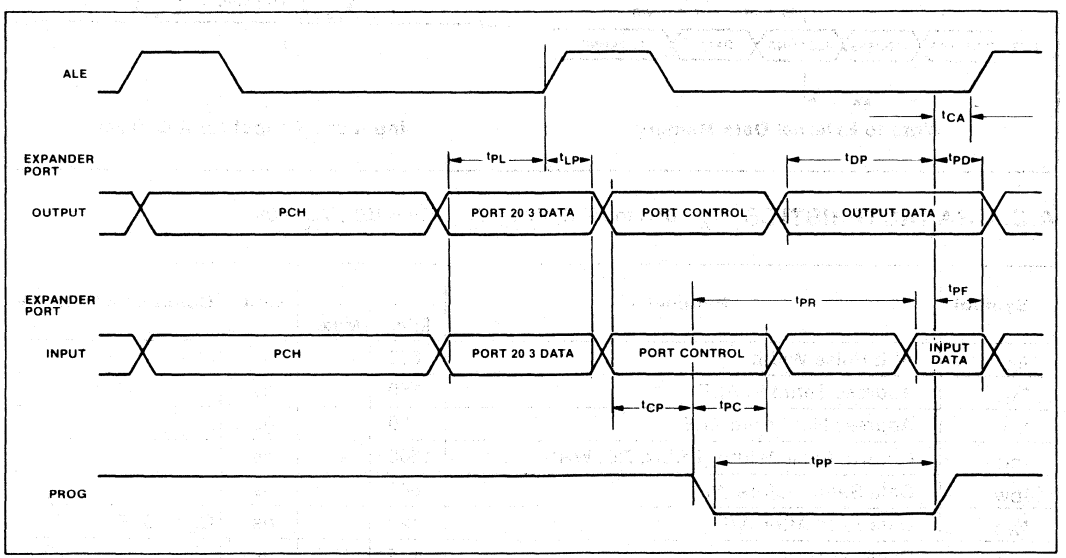
A.C. CHARACTERISTICS (PORT 2 TIMING)

T_A = 0° C to 70° C, V_{CC} = 5V + 10%, V_{SS} = 0V

TCY = 4.17 μS

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
t_{CP}	Port Control Setup Before Falling Edge of PROG	185		ns	
t_{PC}	Port Control Hold After Falling Edge of PROG	160		ns	
t_{PR}	PROG to Time P2 Input Must Be Valid		1.35	μs	
t_{PF}	Input Data Hold Time	0	250	ns	
t_{DP}	Output Data Setup Time	420		ns	
t_{PD}	Output Data Hold Time	110		ns	
t_{PP}	PROG Pulse Width	2.0		μs	
t_{PL}	Port 2 I/O Data Setup	585		ns	
t_{LP}	Port 2 I/O Data Hold	250		ns	

PORT 2 TIMING



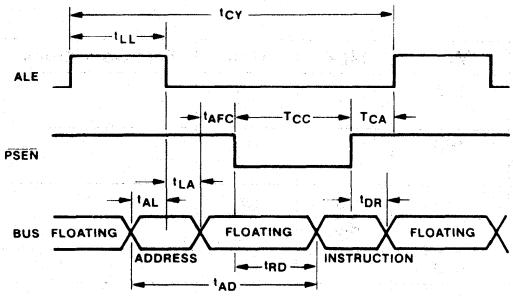
BUS TIMING AS A FUNCTION OF TCY *

SYMBOL	FUNCTION OF TCY			
T_{LL}	7/30	T_{CY}	MIN	
T_{AL}	2/15	T_{CY}	MIN	
T_{LA}	1/15	T_{CY}	MIN	
T_{CC} (1)	1/2	T_{CY}	MIN	T_{CC} (1) : $\overline{RD}/\overline{WR}$
T_{CC} (2)	2/5	T_{CY}	MIN	T_{CC} (2) : $\overline{PSEN}$
T_{DW}	13/30	T_{CY}	MIN	
T_{WD}	1/15	T_{CY}	MIN	
T_{DR}	0		MIN	

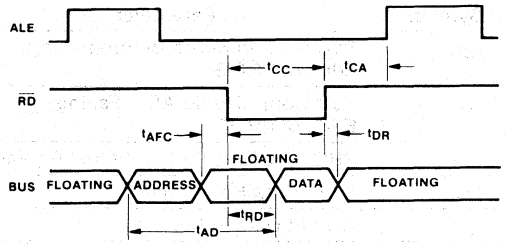
SYMBOL	FUNCTION OF TCY			
T_{RD} (1)	2/5	T_{CY}	MAX	T_{RD} (1) : $\overline{RD}$
T_{RD} (2)	3/10	T_{CY}	MAX	T_{RD} (2) : $\overline{PSEN}$
T_{AW}	1/3	T_{CY}	MIN	
T_{AD} (1)	11/15	T_{CY}	MAX	T_{AD} (1) : $\overline{RD}$
T_{AD} (2)	8/15	T_{CY}	MAX	T_{AD} (2) : $\overline{PSEN}$
T_{AFC} (1)	2/15	T_{CY}	MIN	T_{AFC} (1): $\overline{RD}$
T_{AFC} (2)	1/30	T_{CY}	MIN	T_{AFC} (2): $\overline{PSEN}$
T_{CA} (1)	1/15	T_{CY}	MIN	T_{CA} (1) : $\overline{RD}$, $\overline{WR}$
T_{CA} (2)	2/15	T_{CY}	MIN	T_{CA} (2) : $\overline{PSEN}$

*** APPROXIMATE VALUES NOT INCLUDING GATE DELAYS.**

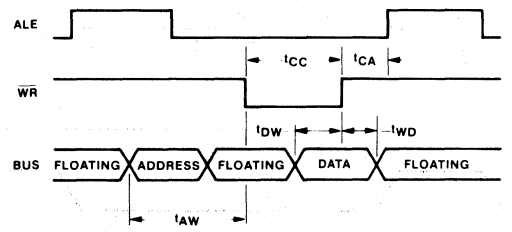
WAVEFORMS



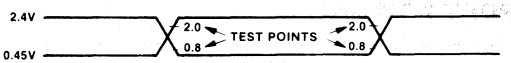
Instruction Fetch From External Program Memory



Read From External Data Memory



Write to External Data Memory



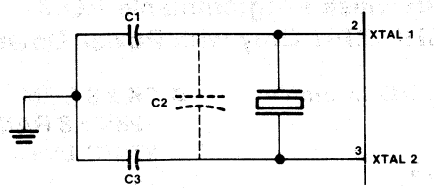
Input and Output for A.C.Tests

A.C. CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = V_{DD} = 5V + 10%, V_{SS} = 0V

Symbol	Parameter	Min.	Max.	Unit	Conditions (Note 1)
t_{LL}	ALE Pulse Width	600		ns	
t_{AL}	Address Setup to ALE	150		ns	
t_{LA}	Address Hold from ALE	80		ns	
t_{CC}	Control Pulse Width (PSEN, RD, WR)	1500		ns	
t_{DW}	Data Setup before WR	640		ns	
t_{WD}	Data Hold After WR	120		ns	C_L = 20pF
t_{CY}	Cycle Time	4.17	15.0	μs	
t_{DR}	Data Hold	0	200	ns	
t_{RD}	PSEN, RD to Data In		750	ns	
t_{AW}	Address Setup to WR	260		ns	
t_{AD}	Address Setup to Data In		1450	ns	
t_{AFC}	Address Float to RD, PSEN	0		ns	
t_{CA}	Control Pulse to ALE	20		ns	

Note 1: Control outputs: C_L = 80 pF
BUS Outputs: C_L = 150 pF

CRYSTAL OSCILLATOR MODE

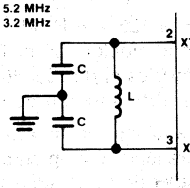


C1 = 5pF ± 1/2pF + STRAY < 5pF
C2 = CRYSTAL + STRAY < 8pF
C3 = 20pF ± 1pF + STRAY < 5pF

CRYSTAL SERIES RESISTANCE SHOULD BE LESS THAN 75Ω AT 6 MHz LESS THAN 180Ω AT 3.6MHz

LC OSCILLATOR MODE

L	C	NOMINAL 1
45pH	20pF	5.2 MHz
120pH	20pF	3.2 MHz



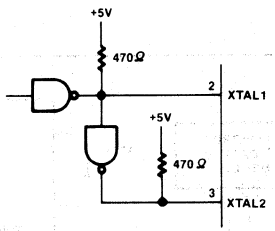
$F \quad \frac{1}{2\pi\sqrt{LC}}$

$C \quad \frac{C + Cpp}{2}$

Cpp = 5-10 pF PIN TO PIN CAPACITANCE

EACH C SHOULD BE APPROXIAMTELY 20pF. INCLUDING STRAY CAPACITANCE

DRIVING FROM EXTERNAL SOURCE



XTAL 1 MUST BE HIGH 35-65% OF THE PERIOD AND XTAL 2 MUST BE HIGH 35-65% OF THE PERIOD. RISE AND FALL TIMES MUST NOT EXCEED 20ns.

PRELIMINARY

8049H/8039HL
HMOS SINGLE COMPONENT 8-BIT MICROCOMPUTER

- **8049H Mask Programmable ROM**
- **8039HL CPU Only with Power Down Mode**

- **8-BIT CPU, ROM, RAM, I/O in Single Package**
- **High Performance HMOS**
- **Reduced Power Consumption**
- **1.4 usec and 1.9 μsec Cycle Versions All Instructions 1 or 2 Cycles.**
- **Over 90 instructions: 70% Single Byte**
- **2K x 8 ROM 128K x 8 RAM 27 I/O Lines**
- **Interval Timer/Event Counter**
- **Easily Expandable Memory and I/O**
- **Compatible with 8080/8085 Series Peripherals**
- **Two Single Level Interrupts**

The Intel® 8049H/8039HL are totally self-sufficient, 8-bit parallel computers fabricated on single silicon chips using Intel's advanced N-channel silicon gate HMOS process.

The 8049H contains a 2K X 8 program memory, a 128 X 8 RAM data memory, 27 I/O lines, and an 8-bit timer/counter in addition to on-board oscillator and clock circuits. For systems that require extra capability the 8049H can be expanded using standard memories and MCS-80®/MCS-85® peripherals. The 8039HL is the equivalent of the 8049H without program memory and can be used with external ROM and RAM.

To reduce development problems to a minimum and provide maximum flexibility, a logically and functionally pin compatible version of the 8049H with UV-erasable user-programmable EPROM program memory will soon be available. The 8749 will emulate the 8049H up to 11 MHz clock frequency with minor differences.

The 8049H is fully compatible with the 8049.

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.

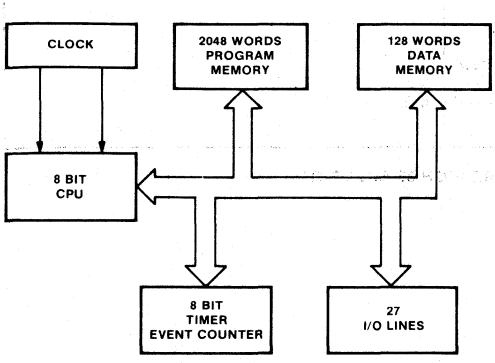


Figure 1. Block Diagram

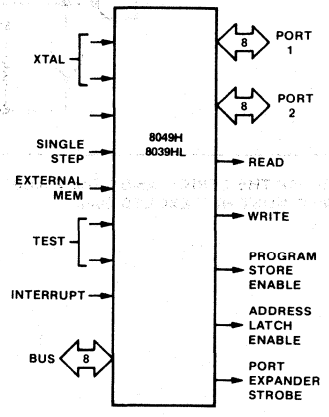


Figure 2. Logic Symbol

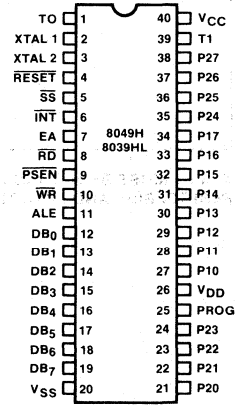


Figure 3. Pin Configuration

AFN-01784A-01

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 8243 I/O expander.
P10-P17 Port 1	27-34	8-bit quasi-bidirectional port.
P20-P27 Port 2	21-24	8-bit quasi-bidirectional port.
	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 $\overline{RD}$, $\overline{WR}$ 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 ENT0 CLK instruction.
T1	39	Input pin testable using the JT1, and JNT1 instructions. Can be designated the timer/counter input using the STRT CNT instruction.
$\overline{INT}$	6	Interrupt input. Initiates an interrupt if interrupt is enabled. Interrupt is disabled after a reset. Also testable with conditional jump instruction. (Active low)

Symbol	Pin No.	Function
$\overline{RD}$	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)
$\overline{RESET}$	4	Input which is used to initialize the processor. (Active low) (Non TTL V_{IH})
$\overline{WR}$	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.
$\overline{PSEN}$	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)
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.

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
JTO addr	Jump on TO = 1	2	2
JNTO addr	Jump on TO = 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
JN1 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
RETR	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

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 counter	1	1
STOP TCNT	Stop timer/counter	1	1
EN TCNT1	Enable timer/counter interrupt	1	1
DIS TCNT1	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
ENT 0 CLK	Enable clock output on T0	1	1

Mnemonic	Description	Bytes	Cycles
NOP	No operation	1	1

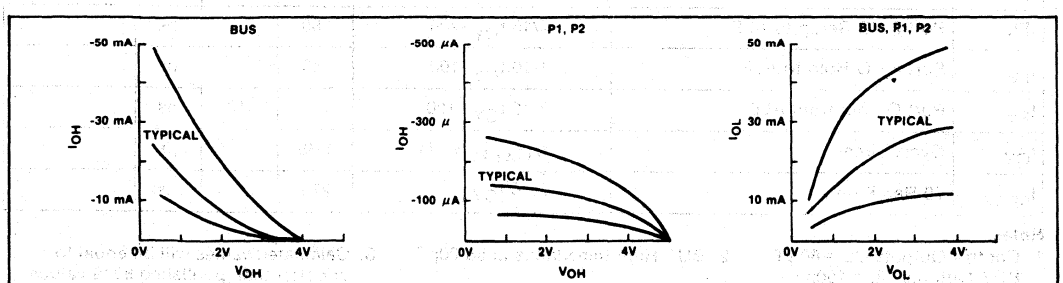
ABSOLUTE MAXIMUM RATINGS*

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 ..... 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 (TA = 0°C to 70°C, V_{CC} = V_{DD} = 5V ± 10%, V_{SS} = 0V)

Symbol	Parameter	Limits			Unit	Test Conditions
		Min.	Typ.	Max.		
V_{IL}	Input Low Voltage (All Except $\overline{RESET}$, X1, X2)	-.5		.8	V	
V_{IL1}	Input Low Voltage ($\overline{RESET}$, X1, X2)	-.5		.6	V	
V_{IH}	Input High Voltage (All Except XTAL1, XTAL2, $\overline{RESET}$)	2.0		V_{CC}	V	
V_{IH1}	Input High Voltage (X1, X2, $\overline{RESET}$)	3.8		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)	2.4			V	I_{OH} = -400 μA
V_{OH1}	Output High Voltage ($\overline{RD}$, $\overline{WR}$, $\overline{PSEN}$, ALE)	2.4			V	I_{OH} = -100 μA
V_{OH2}	Output High Voltage (All Other Outputs)	2.4			V	I_{OH} = -40 μA
I_{L1}	Input Leakage Current (T1, $\overline{INT}$)			± 10	μA	$V_{SS} \leq V_{IN} \leq V_{CC}$
I_{LI1}	Input Leakage Current (P10-P17, P20-P27, EA, $\overline{SS}$)			-500	μA	V_{SS} + .45 $\leq V_{IN} \leq V_{CC}$
I_{L0}	Output Leakage Current (BUS, TO) (High Impedance State)			± 10	μA	V_{SS} + .45 $\leq V_{IN} \leq V_{CC}$
I_{DD}	V_{DD} Supply Current		5	10	mA	
I_{DD} + I_{CC}	Total Supply Current		50	100	mA	



AFN-01784A-04

A.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = V_{DD} = 5V ± 10%, V_{SS} = 0V)

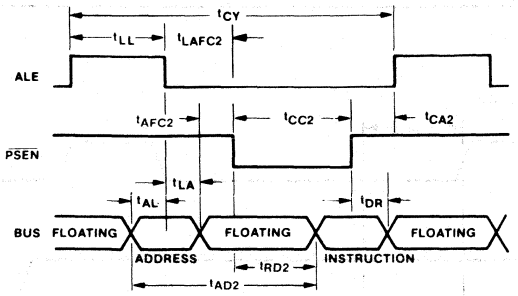
Symbol	Parameter	f (t_{CY}) (Note 3)	11 MHz Min.	11 MHz Max.	Unit	Conditions (Note 1)
t_{LL}	ALE Pulse Width	7/30 t_{CY} -170	150		ns	
t_{AL}	Addr Setup to ALE	1/5 t_{CY} -110	160		ns	
t_{LA}	Addr Hold from ALE	1/15 t_{CY} -40	50		ns	
t_{CC1}	Control Pulse Width ($\overline{RD}$, $\overline{WR}$)	1/2 t_{CY} -200	480		ns	
t_{CC2}	Control Pulse Width ($\overline{PSEN}$)	2/5 t_{CY} -200	350		ns	
t_{DW}	Data Setup before $\overline{WR}$	13/30 t_{CY} -200	390		ns	
t_{WD}	Data Hold after $\overline{WR}$	1/15 t_{CY} -50	40		ns	(Note 2)
t_{DR}	Data Hold ($\overline{RD}$, $\overline{PSEN}$)	1/10 t_{CY} -30	0	110	ns	
t_{RD1}	$\overline{RD}$ to Data in	2/5 t_{CY} -200		350	ns	
t_{RD2}	$\overline{PSEN}$ to Data in	3/10 t_{CY} -200		210	ns	
t_{AW}	Addr Setup to $\overline{WR}$	2/5 t_{CY} -150	400		ns	
t_{AD1}	Addr Setup to Data ($\overline{RD}$)	23/30 t_{CY} -250		800	ns	
t_{AD2}	Addr Setup to Data ($\overline{PSEN}$)	3/5 t_{CY} -250		570	ns	
t_{AFC1}	Addr Float to $\overline{RD}$, $\overline{WR}$	2/15 t_{CY} -40	140		ns	
t_{AFC2}	Addr Float to $\overline{PSEN}$	1/30 t_{CY} -40	10		ns	
t_{LAFC1}	ALE to Control, ($\overline{RD}$, $\overline{WR}$)	1/5 t_{CY} -75	200		ns	
t_{LAFC2}	ALE to Control ($\overline{PSEN}$)	1/10 t_{CY} -75	60		ns	
t_{CA1}	Control to ALE ($\overline{RD}$, $\overline{WR}$, $\overline{PROG}$)	1/15 t_{CY} -40	50		ns	
t_{CA2}	Control to ALE ($\overline{PSEN}$)	4/15 t_{CY} -40	320		ns	
t_{CP}	Port Control Setup to $\overline{PROG}$	2/15 t_{CY} -80	100		ns	
t_{PC}	Port Control Hold to $\overline{PROG}$	4/15 t_{CY} -200	160		ns	
t_{PR}	$\overline{PROG}$ to P2 Input Valid	6/10 t_{CY} -120		700	ns	
t_{PF}	Input Data Hold from $\overline{PROG}$	1/10 t_{CY}	0	140	ns	
t_{DP}	Output Data Setup	2/5 t_{CY} -150	400		ns	
t_{PD}	Output Data Hold	1/10 t_{CY} -50	90		ns	
t_{PP}	$\overline{PROG}$ Pulse Width	7/10 t_{CY} -250	700		ns	
tPL	Port 2 I/O Setup to ALE	4/15 t_{CY} -200	160		ns	
t_{LP}	Port 2 I/O Hold to ALE	1/10 t_{CY} -100	40		ns	
t_{PV}	Port Output from ALE	3/10 t_{CY} +100		510	ns	
t_{CY}	Cycle Time	1/(f_{XTAL} x 15)	1.36		µs	
t_{0PRR}	T0 Rep Rate	3/15 t_{CY}	270		ns	

Notes:

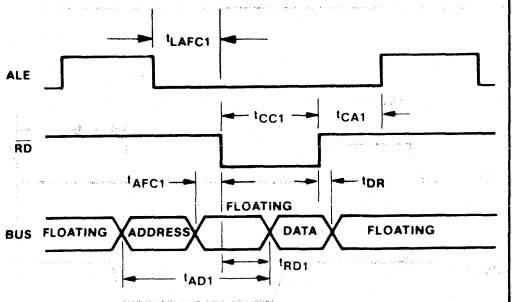
1. Control Outputs CL = 80 pF
   BUS Outputs CL = 150pF
2. BUS High Impedance Load 20pF
3. Calculated values will be equal to or better than published 8049 values. f(t_{cy}) assumes 50% duty cycle on X1, X2.
4. Interrupt pin must remain low for at least 3 t_{cy} to ensure proper operation.

AFN-01784A-05

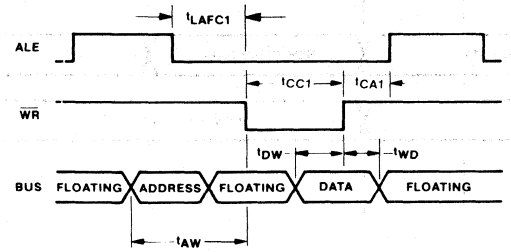
WAVEFORMS



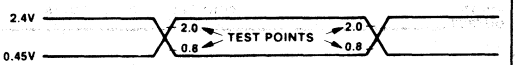
Instruction Fetch From External Program Memory



Read From External Data Memory

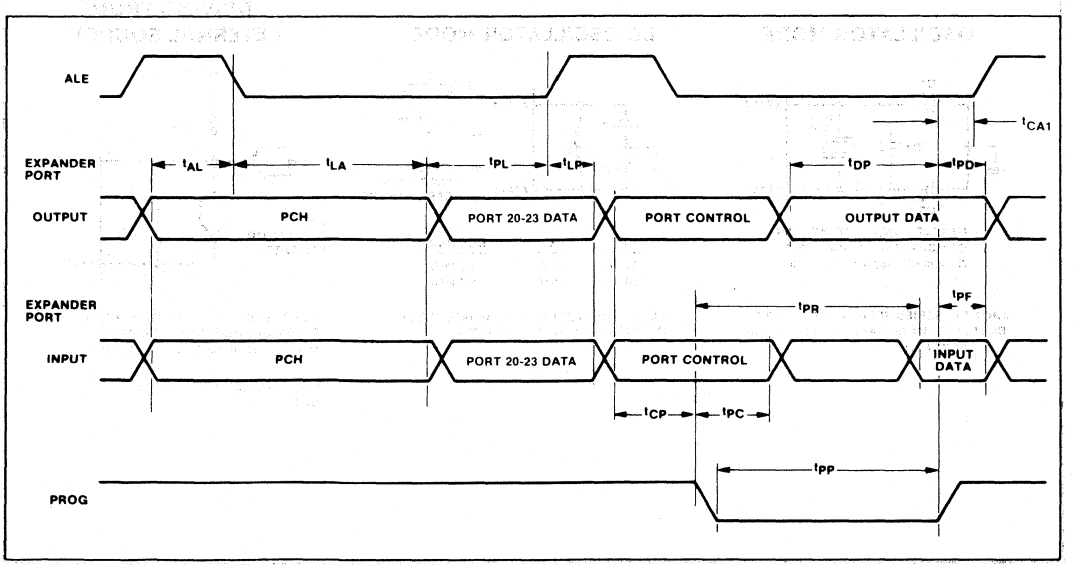


Write to External Data Memory

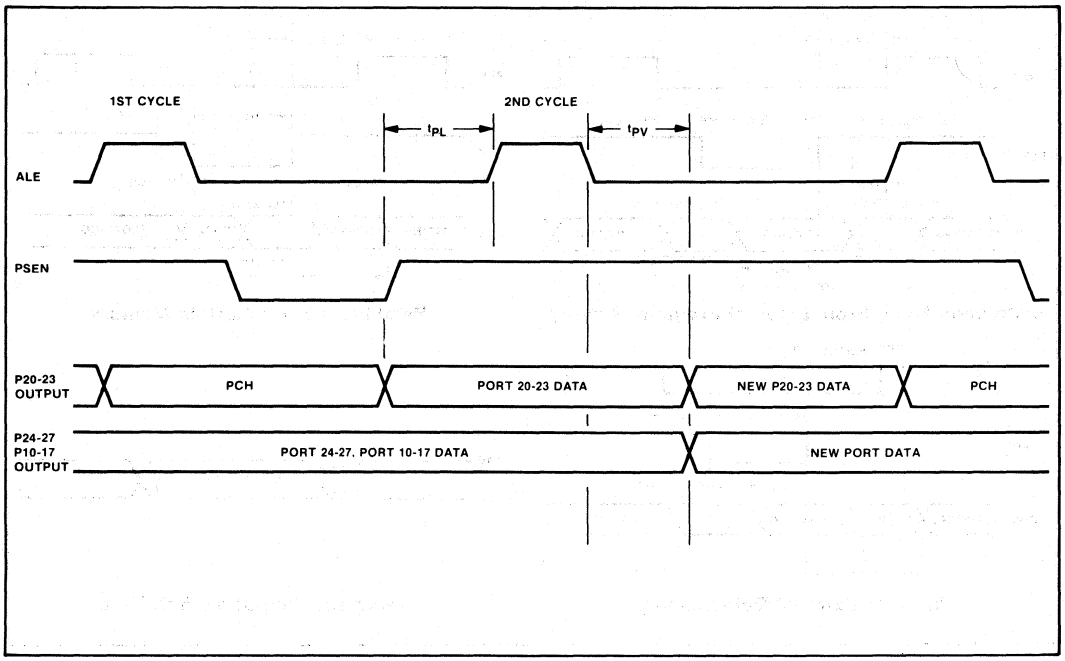


Input and Output for A.C. Tests

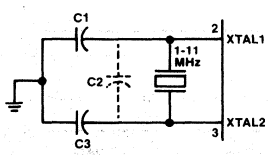
PORT 2 EXPANDER TIMING



I/O PORT TIMING



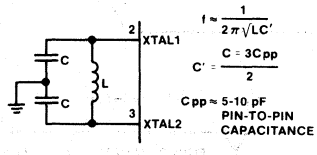
OSCILLATOR MODE



C1 = 5pF ± 1/2pF + STRAY < 5pF
C2 = CRYSTAL + STRAY < 8pF
C3 = 20pF ± 1pF + STRAY < 5pF

CRYSTAL SERIES RESISTANCE SHOULD BE LESS THAN 75 Ω AT 6MHz; LESS THAN 180 Ω AT 3.6MHz.

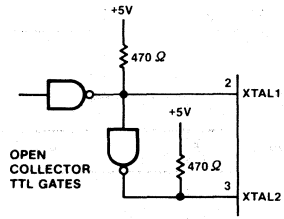
LC OSCILLATOR MODE



L	C	NOMINAL f
45 μH	20pF	5.2 MHz
120 μH	20pF	3.2 MHz

EACH C SHOULD BE APPROXIMATELY 20pF, INCLUDING STRAY CAPACITANCE.

DRIVING FROM EXTERNAL SOURCE



FOR THE 8049, XTAL1 MUST BE HIGH 35-65% OF THE PERIOD AND XTAL2 MUST BE HIGH 35-65% OF THE PERIOD.

RISE AND FALL TIMES MUST NOT EXCEED 20ns.

8748/8748-6/8748-8/8035
SINGLE COMPONENT 8-BIT MICROCOMPUTER

- 8748 User Programmable EPROM
- 8748-6 Up to 55C 8748
- 8748-8 Slow-Speed User-Programmable EPROM
- 8035 CPU Only

- 8-bit CPU, ROM, RAM, I/O in Single Package
- Single 5V Supply
- 2.5 μsec and 5.0 μsec Cycle Versions All Instructions 1 or 2 Cycles.
- Over 90 Instructions: 70% Single Byte
- 1K x 8 ROM/EPROM
- 64 x 8 RAM
  27 I/O Lines
- Interval Timer/Event Counter
- Easily Expandable Memory and I/O
- Compatible with 8080/8085 Series Peripherals
- Single Level Interrupt

The Intel® 8748/8748-6/8748-8/8035 are totally self-sufficient, 8-bit parallel computers fabricated on single silicon chips using Intel's N-channel silicon gate MOS process.

The 8748 contains a 1K x 8 UV-erasable, user-programmable program memory, a 64 x 8 RAM data memory, 27 I/O lines, and an 8-bit timer/counter in addition to on-board oscillator and clock circuits. For systems that require extra capability, the 8748 can be expanded using standard memories and MCS-80®/MCS-85® peripherals. The 8035 is the equivalent of an 8748 without program memory and can be used with external ROM and RAM. The 8748-6 is a 6 MHz 8748 up to 55C. To reduce development problems to a minimum and provide maximum flexibility, three interchangeable pin-compatible versions of this single component microcomputer exist: the 8748 with user-programmable and erasable EPROM program memory, the 8048 with factory-programmed mask ROM program memory for low cost, high volume production, and the 8035 without program memory for use with external program memories.

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 bit instructions and no instructions over 2 bytes in length.

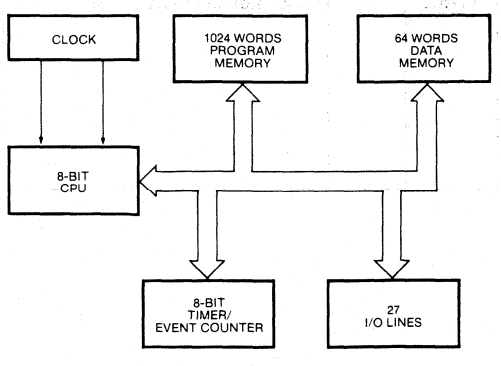


Figure 1. Block Diagram

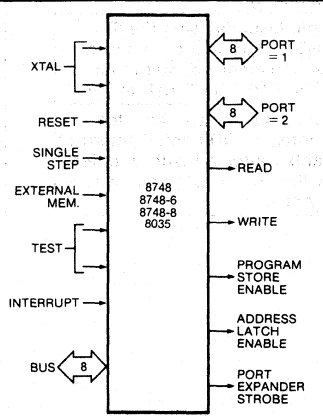


Figure 2. Logic Symbol

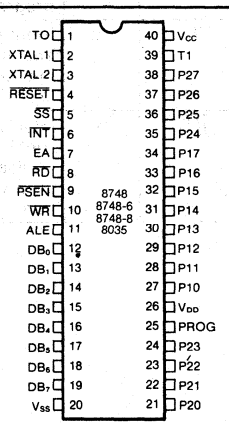


Figure 3. Pin Configuration

Table 1. Pin Description

Symbol	Pin No.	Function
V_{SS}	20	Circuit GND Potential
V_{DD}	26	Programming power supply; +25V during program, +5V during operation.
V_{CC}	40	Main power supply; +5V during operation and programming.
PROG	25	Program pulse (+23V) input pin during 8748 programming. Output strobe for 8243 I/O expander.
P10-P17 Port 1	27-34	8-bit quasi-bidirectional port.
P20-P27	21-24	8-bit quasi-bidirectional port.
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.
DB_0-DB_7 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 ENT0 CLK instruction. T0 is also used during programming.
T1	39	Input pin testable using the JT1, and JNT1 instructions. Can be designated the timer/counter input using the STRT CNT instruction.
$\overline{INT}$	6	Interrupt input. Initiates an interrupt if interrupt is enabled. Interrupt is disabled after a reset. Also testable with conditional jump instruction. (Active low)

Symbol	Pin No.	Function
$\overline{RD}$	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)
$\overline{RESET}$	4	Input which is used to initialize the processor. Also used during PROM programming verification, and power down. (Active low) (Non TTL V_{IH})
$\overline{WR}$	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.
$\overline{PSEN}$	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)
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.

Table 2. 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
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
	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 $\overline{INT}$ = 0	2	2
	JBb addr	Jump on accumulator bit	2	2

	Mnemonic	Description	Bytes	Cycles
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
	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
	ENT0 CLK	Enable clock output on T0	1	1
	NOP	No operation	1	1

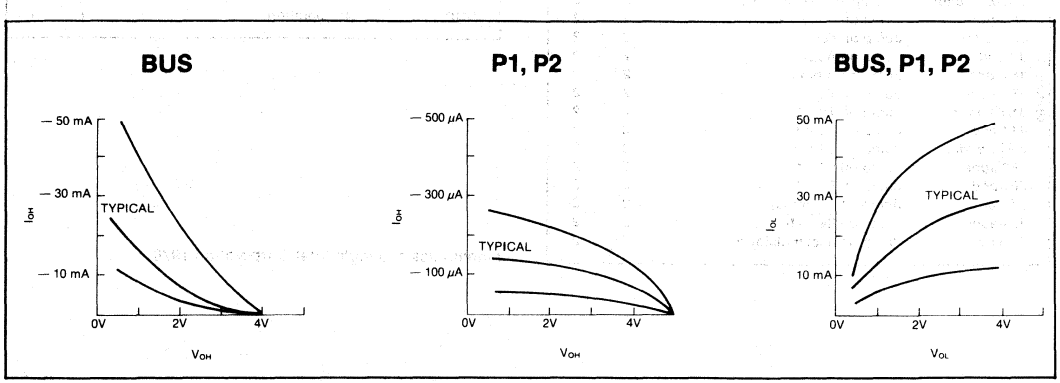
ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias ..... 0°C to 70°C
Storage Temperature ..... −65°C to +125°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 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.

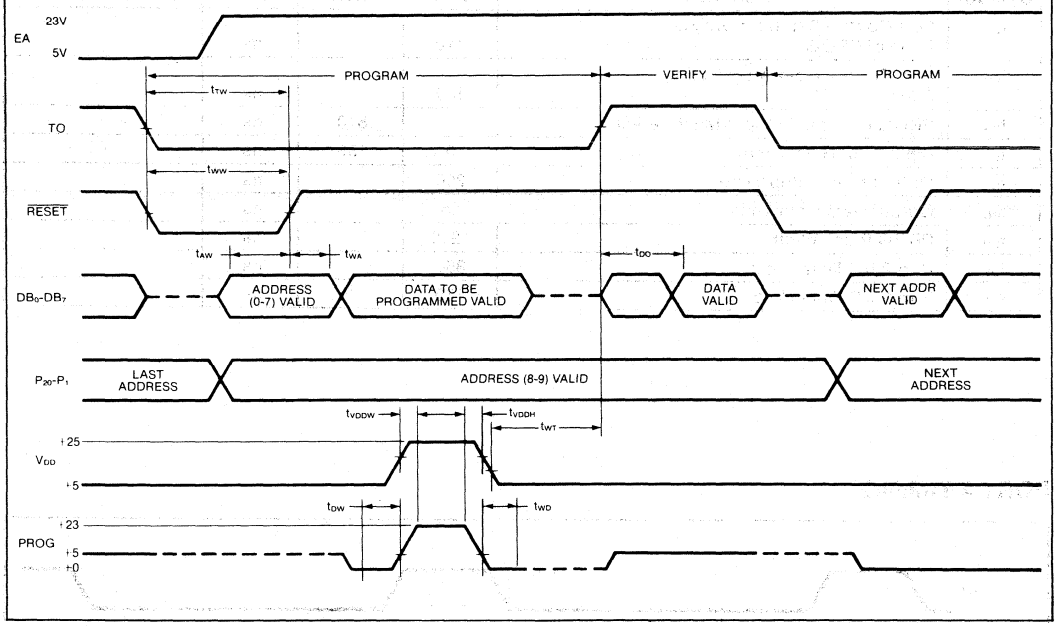
D.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = V_{DD} = +5V ± 10%, V_{SS} = 0V) (T_A = 0°C to 55°C for 8748-6)

Symbol	Parameter	Limits			Unit	Test Conditions
		Min	Typ	Max		
V_{IL}	Input Low Voltage (All Except $\overline{RESET}$, X1, X2)	−.5		.8	V	
V_{IL1}	Input Low Voltage ($\overline{RESET}$, X1, X2)	−.5		.6	V	
V_{IH}	Input High Voltage (All Except XTAL1, XTAL2, $\overline{RESET}$)	2.0		V_{CC}	V	
V_{IH1}	Input High Voltage (X1, X2, $\overline{RESET}$)	3.8		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)	2.4			V	I_{OH} = −400 μA
V_{OH1}	Output High Voltage ($\overline{RD}$, $\overline{WR}$, $\overline{PSEN}$, ALE)	2.4			V	I_{OH} = −100 μA
V_{OH2}	Output High Voltage (All Other Outputs)	2.4			V	I_{OH} = −40 μA
I_{LI}	Input Leakage Current (T1, $\overline{INT}$)			±10	μA	$V_{SS} \le V_{IN} \le V_{CC}$
I_{LI1}	Input Leakage Current (P10-P17, P20-P27, EA, $\overline{SS}$)			−500	μA	V_{SS} + .45 ≤ $V_{IN} \le V_{CC}$
I_{LO}	Output Leakage Current (BUS, TO) (High Impedance State)			±10	μA	V_{SS} + .45 ≤ $V_{IN} \le V_{CC}$
I_{DD}	V_{DD} Supply Current		5	15	mA	
$I_{DD} + I_{CC}$	Total Supply Current		60	135	mA	

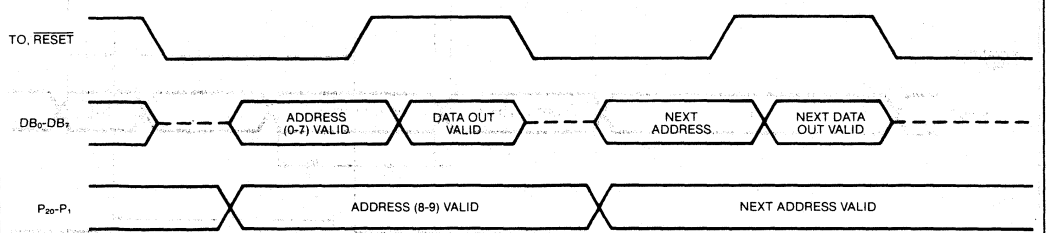


WAVEFORMS

COMBINATION PROGRAM/VERIFY MODE (EPROM's ONLY)



VERIFY MODE (ROM/EPROM)



NOTES:
1. PROG MUST FLOAT IF EA IS LOW (i.e., ≠23V).
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.

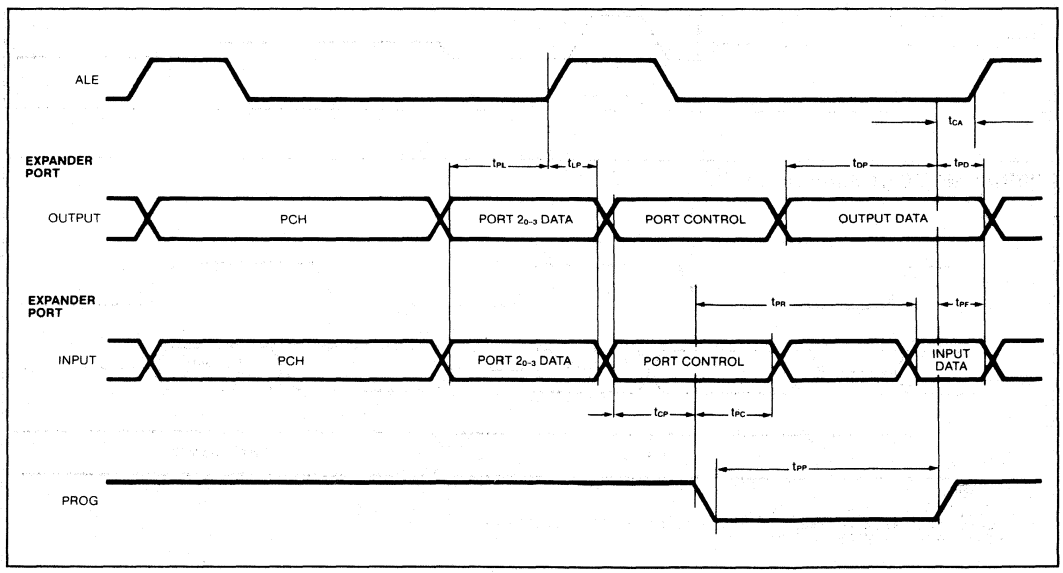
The 8748 EPROM can be programmed by either of two Intel products:

1. PROMPT-48 Microcomputer Design Aid, or
2. Universal PROM Programmer (UPP series) peripheral of the Intellec® Development System with a UPP-848 Personality Card.

A.C. CHARACTERISTICS (PORT 2 TIMING) (T_A = 0°C to 70°C, V_{CC} = 5V±10%, V_{SS} = 0V)
(T_A = 0°C to 55°C for 8748-6)

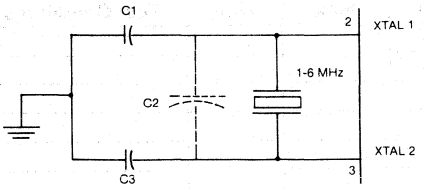
Symbol	Parameter	Min	Max	Unit	Test Conditions
t_{CP}	Port Control Setup Before Falling Edge of PROG	110		ns	
t_{CP}	Port Control Hold After Falling Edge of PROG	100		ns	
t_{PR}	PROG to Time P2 Input Must Be Valid		810	ns	
t_{PF}	Input Data Hold Time	0	150	ns	
t_{DP}	Output Data Setup Time	250		ns	
t_{PD}	Output Data Hold Time	65		ns	
t_{PP}	PROG Pulse Width	1200		ns	
t_{PL}	Port 2 I/O Data Setup	350		ns	
t_{LP}	Port 2 I/O Data Hold	150		ns	

PORT 2 TIMING



AFN-01873A-6

CRYSTAL OSCILLATOR MODE



C1 = 5pF ± ½pF + STRAY < 5 pF
C2 = CRYSTAL ± STRAY < 8 pF
C3 = 20pF ± 1pF ± STRAY < 5 pF

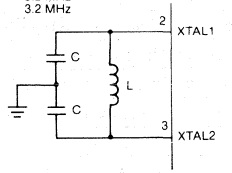
CRYSTAL SERIES RESISTANCE SHOULD BE LESS THAN 75Ω AT 6 MHz; LESS THAN 180Ω AT 3.6 MHz.

LC OSCILLATOR MODE

L	C	NOMINAL f
45 μH	20 pF	5.2 MHz
120 μH	20 pF	3.2 MHz

$f \approx \frac{1}{2\pi\sqrt{LC'}}$

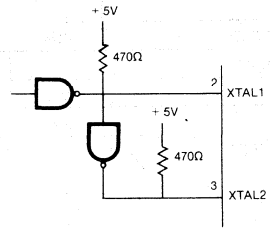
$C' = \frac{C + 3C_{PP}}{2}$



$C_{PP} \approx 5 - 10$ pF PIN-TO-PIN CAPACITANCE

EACH C SHOULD BE APPROXIMATELY 20 pF, INCLUDING STRAY CAPACITANCE.

DRIVING FROM EXTERNAL SOURCE



FOR XTAL 1 AND XTAL 2 DEFINE "HIGH" AS VOLTAGES ABOVE 1.6V AND "LOW" AS VOLTAGES BELOW 1.6V. THE DUTY CYCLE REQUIREMENTS FOR EXTERNALLY DRIVING XTAL 1 AND XTAL 2 USING THE CIRCUIT SHOWN ABOVE ARE AS FOLLOWS:

FOR THE 8748, XTAL MUST BE HIGH 45 — 50% OF THE PERIOD AND XTAL 2 MUST BE HIGH 50 — 55% OF THE PERIOD.
RISE AND FALL TIMES MUST NOT EXCEED 20 ns.

PROGRAMMING, VERIFYING, AND ERASING THE 8748 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
XTAL 1	Clock Input (1 to 3MHz)
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
P20-1	Address Input
V_{DD}	Programming Power Supply
PROG	Program Pulse Input

WARNING:

An attempt to program a missocketed 8748 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, $\overline{RESET} = 0V$, TEST 0 = 5V, EA = 5V, BUS and PROG floating.
2. Insert 8748 in programming socket.
3. TEST 0 = 0V (select program mode)
4. EA = 23V (activate program mode)
5. Address applied to BUS and P20-1
6. $\overline{RESET} = 5V$ (latch address)
7. Data applied to BUS
8. $V_{DD} = 25V$ (programming power)
9. PROG = 0V followed by one 50ms pulse to 23V
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 8748 is removed from socket.

AFN-01873A-7

A.C. TIMING SPECIFICATION FOR PROGRAMMING ($T_A = 25°C \pm 5°C$, $V_{CC} = 5V \pm 5\%$, $V_{DD} = 25V \pm 1V$)

Symbol	Parameter	Min	Max	Unit	Test Conditions
t_{AW}	Address Setup Time to $\overline{RESET}\uparrow$	$4t_{CY}$			
t_{WA}	Address Hold Time After $\overline{RESET}\uparrow$	$4t_{CY}$			
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}	$4t_{CY}$			
t_{VDDH}	V_{DD} Hold Time After PROG ↓	0			
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	2.0	μs	
t_{CY}	CPU Operation Cycle Time	5.0		μs	
t_{RE}	$\overline{RESET}$ Setup Time before EA↑	$4t_{CY}$			

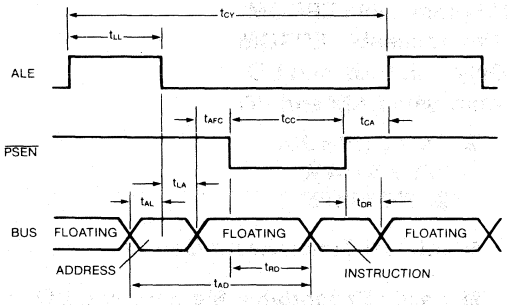
NOTE: If Test 0 is high t_{DO} can be triggered by $\overline{RESET}$ ↑

D.C. SPECIFICATION FOR PROGRAMMING ($T_A = 25°C \pm 5°C$, $V_{CC} = 5V \pm 5\%$, $V_{DD} = 25V \pm 1V$)

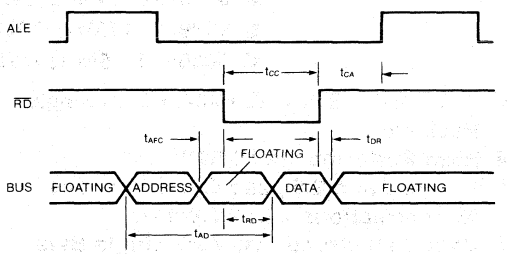
Symbol	Parameter	Min	Max	Unit	Test Conditions
V_{DOH}	V_{DD} Program Voltage High Level	24.0	26.0	V	
V_{DDL}	V_{DD} Voltage Low Level	4.75	5.25	V	
V_{PH}	PROG Program Voltage High Level	21.5	24.5	V	
V_{PL}	PROG Voltage Low Level		0.2	V	
V_{EAH}	EA Program or Verify Voltage High Level	21.5	24.5	V	8748
V_{EAL}	EA Voltage Low Level		5.25	V	
I_{DD}	V_{DD} High Voltage Supply Current		30.0	mA	
I_{PROG}	PROG High Voltage Supply Current		16.0	mA	
I_{EA}	EA High Voltage Supply Current		1.0	mA	

WAVEFORMS

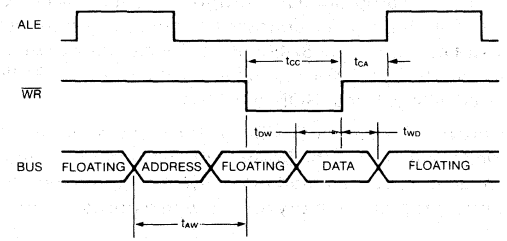
INSTRUCTION FETCH FROM EXTERNAL PROGRAM MEMORY

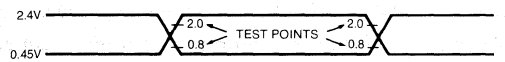


READ FROM EXTERNAL DATA MEMORY



WRITE TO EXTERNAL DATA MEMORY





A.C. CHARACTERISTICS (T_A = 0°C to 70°C*, V_{CC} = V_{DD} = +5V ± 10%, V_{SS} = 0V)

Symbol	Parameter	8748/8748-6/8035 Min	8748/8748-6/8035 Max	8748-8** Min	8748-8** Max	Unit	Conditions (Note 1)
t_{LL}	ALE Pulse Width	400		600		ns	
t_{AL}	Address Setup to ALE	120		150		ns	
t_{LA}	Address Hold from ALE	80		80		ns	
t_{CC}	Control Pulse Width ($\overline{PSEN}$, $\overline{RD}$, $\overline{WR}$)	700		1500		ns	
t_{DW}	Data Setup before $\overline{WR}$	500		640		ns	
t_{WD}	Data Hold After $\overline{WR}$	120		120		ns	C_L = 20pF
t_{CY}	Cycle Time	2.5	15.0	4.17	15.0	μs	6 MHz XTAL = 2.5 (3.6 MHz XTAL for — 8)
t_{DR}	Data Hold	0	200	0	200	ns	
t_{RD}	$\overline{PSEN}$, $\overline{RD}$ to Data In		500		750	ns	
t_{AW}	Address Setup to $\overline{WR}$	230		260		ns	
t_{AD}	Address Setup to Data In		950		1450	ns	
t_{AFC}	Address Float to $\overline{RD}$, $\overline{PSEN}$	0		0		ns	
t_{CA}	Control Pulse to ALE	10		20		ns	

NOTE 1: Control outputs: C_L = 80 pF; BUS Outputs: C_L = 150 pF; t_{CY} = 2.5 μs for standard parts, = 4.17 μs for -8 parts

*T_A = 0°C to 55°C for 8748-6

**V_{CC} and V_{DD} for 8748-8 is ± 5%

AFN-01873A-9

8749H/8749H-8/8039H/8039H-8
HMOS SINGLE-CHIP EPROM MICROCOMPUTER

- 8749H 11MHz User Programmable EPROM
- 8749H-8 6MHz User Programmable EPROM
- 8039H 11MHz CPU Only with RAM and I/O
- 8039H-8 6MHz CPU Only with RAM and I/O

- 8-BIT CPU, EPROM, RAM, I/O in Single Package
- High Performance HMOS
- 1.4 μsec and 2.5 μsec Cycle Versions All Instructions 1 or 2 Cycles.
- Over 90 instructions: 70% Single Byte
- Compatible with 8080/8085 Series Peripherals
- 2K x 8 EPROM 128 x 8 RAM 27 I/O Lines
- Interval Timer/Event Counter
- Easily Expandable Memory and I/O
- Two Single Level Interrupts

The Intel® 8749H/8749H-8 are totally self-sufficient, 8-bit parallel computers fabricated on single silicon chips using Intel's advanced N-channel silicon gate HMOS process.

The 8749H/8749H-8 contains on-chip a 2K X 8 UV-erasable, user-programmable program memory, a 128 X 8 RAM data memory, 27 I/O lines, 2 interrupt sources, and an 8-bit timer/counter in addition to on-board oscillator and clock circuits. For systems that require extra capability the 8749H/8749H-8 can be expanded using standard memories and MCS-80®/MCS-85® peripherals. The 8039H is the equivalent of the 8749H/8749H-8/8049H without program memory and can be used with external ROM and RAM.

To reduce development problems to a minimum and provide maximum flexibility, three interchangeable pin-compatible versions of this single component microcomputer exist: the 8749H with user-programmable and erasable EPROM program memory, the 8049H with factory-programmed mask ROM program memory for low cost, high volume production, and the 8039H without program memory for use with external program memories.

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.

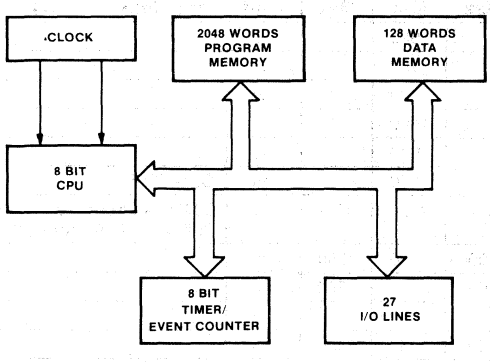


Figure 1. Block Diagram

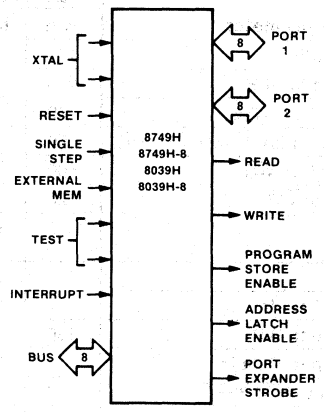


Figure 2. Logic Symbol

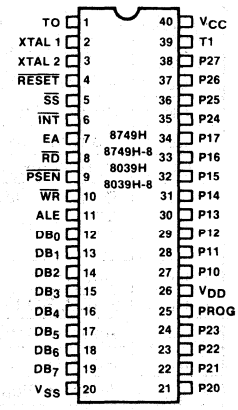


Figure 3. Pin Configuration

Table 1. Pin Description

Symbol	Pin No.	Function
V_{SS}	20	Circuit GND Potential
V_{DD}	26	Programming power supply; +20V during program, +5V during operation.
V_{CC}	40	Main power supply; +5V during operation and programming.
PROG	25	Program pulse (+18V) input pin during 8749H programming. Output strobe for 8243 I/O expander.
P10-P17 Port 1	27-34	8-bit quasi-bidirectional port.
P20-P27	21-24	8-bit quasi-bidirectional port.
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.
DB_0-DB_7 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 ENT0 CLK instruction. T0 is also used during programming.
T1	39	Input pin testable using the JT1, and JNT1 instructions. Can be designated the timer/counter input using the STRT CNT instruction.
$\overline{INT}$	6	Interrupt input. Initiates an interrupt if interrupt is enabled. Interrupt is disabled after a reset. Also testable with conditional jump instruction. (Active low)
$\overline{RD}$	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)
$\overline{RESET}$	4	Input which is used to initialize the processor. Also used during PROM programming verification, and power down. (Active low) (Non TTL V_{IH})
$\overline{WR}$	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.
$\overline{PSEN}$	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)
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.

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 TO = 1	2	2
JNT0 addr	Jump on TO = 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
JN1 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
RETR	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

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 counter	1	1
STOP TCNT	Stop timer/counter	1	1
EN TCNT1	Enable timer/counter interrupt	1	1
DIS TCNT1	Disable timer/counter interrupt	1	1

Control

Mnemonic	Description	Bytes	Cycles
EN 1	Enable external interrupt	1	1
DIS 1	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
ENT 0 CLK	Enable clock output on T0	1	1

Mnemonic	Description	Bytes	Cycles
NOP	No operation	1	1

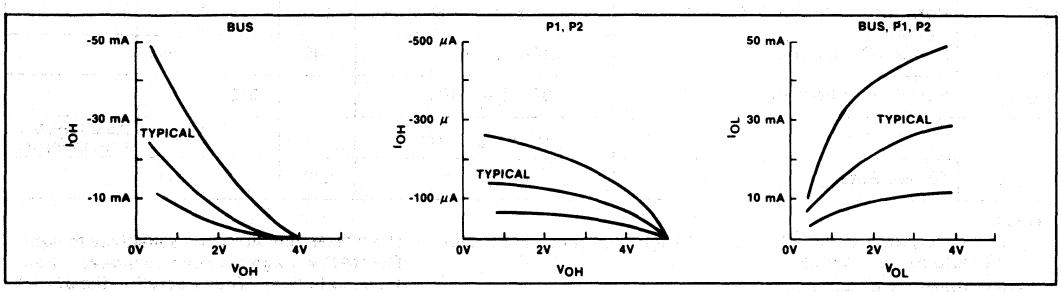
ABSOLUTE MAXIMUM RATINGS*

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 ..... 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
		Min	Typ	Max		
V_{IL}	Input Low Voltage (All Except $\overline{RESET}$, X1, X2)	-.5		.8	V	
V_{IL1}	Input Low Voltage ($\overline{RESET}$, X1, X2)	-.5		.6	V	
V_{IH}	Input High Voltage (All Except XTAL1, XTAL2, $\overline{RESET}$)	2.0		V_{CC}	V	
V_{IH1}	Input High Voltage (X1, X2, $\overline{RESET}$)	3.8		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)	2.4			V	I_{OH} = -400 μA
V_{OH1}	Output High Voltage ($\overline{RD}$, $\overline{WR}$, $\overline{PSEN}$, ALE)	2.4			V	I_{OH} = -100 μA
V_{OH2}	Output High Voltage (All Other Outputs)	2.4			V	I_{OH} = -40 μA
I_{L1}	Input Leakage Current (T1, $\overline{INT}$)			± 10	μA	$V_{SS} \leq V_{IN} \leq V_{CC}$
I_{LI1}	Input Leakage Current (P10-P17, P20-P27, EA, $\overline{SS}$)			-500	μA	V_{SS} + .45 $\leq V_{IN} \leq V_{CC}$
I_{L0}	Output Leakage Current (BUS, TO) (High Impedance State)			± 10	μA	V_{SS} + .45 $\leq V_{IN} \leq V_{CC}$
I_{DD}	V_{DD} Supply Current		5	10	mA	
I_{DD} + I_{CC}	Total Supply Current		50	110	mA	



AFN-01930A-4

A.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = V_{DD} = 5V ± 10%, V_{SS} = 0V)

Symbol	Parameter	f (t_{CY})	11 MHz		Unit	Conditions (Note 1)
			Min	Max		
t_{LL}	ALE Pulse Width	7/30 t_{CY} -170	150		ns	
t_{AL}	Addr Setup to ALE	1/5 t_{CY} -110	160		ns	
t_{LA}	Addr Hold from ALE	1/15 t_{CY} -40	50		ns	
t_{CC1}	Control Pulse Width ($\overline{RD}$, $\overline{WR}$)	1/2 t_{CY} -200	480		ns	
t_{CC2}	Control Pulse Width ($\overline{PSEN}$)	2/5 t_{CY} -200	350		ns	
t_{DW}	Data Setup before $\overline{WR}$	13/30 t_{CY} -200	390		ns	
t_{WD}	Data Hold after $\overline{WR}$	1/15 t_{CY} -50	40		ns	(Note 2)
t_{DR}	Data Hold ($\overline{RD}$, $\overline{PSEN}$)	1/10 t_{CY} -30	0	110	ns	
t_{RD1}	$\overline{RD}$ to Data in	2/5 t_{CY} -200		350	ns	
t_{RD2}	$\overline{PSEN}$ to Data in	3/10 t_{CY} -200		210	ns	
t_{AW}	Addr Setup to $\overline{WR}$	2/5 t_{CY} -150	400		ns	
t_{AD1}	Addr Setup to Data ($\overline{RD}$)	23/30 t_{CY} -250		800	ns	
t_{AD2}	Addr Setup to Data ($\overline{PSEN}$)	3/5 t_{CY} -250		570	ns	
t_{AFC1}	Addr Float to $\overline{RD}$, $\overline{WR}$	2/15 t_{CY} -40	140		ns	
t_{AFC2}	Addr Float to $\overline{PSEN}$	1/30 t_{CY} -40	10		ns	
t_{LAFC1}	ALE to Control, ($\overline{RD}$, $\overline{WR}$)	1/5 t_{CY} -75	200		ns	
t_{LAFC2}	ALE to Control ($\overline{PSEN}$)	1/10 t_{CY} -75	60		ns	
t_{CA1}	Control to ALE ($\overline{RD}$, $\overline{WR}$, $\overline{PROG}$)	1/15 t_{CY} -40	50		ns	
t_{CA2}	Control to ALE ($\overline{PSEN}$)	4/15 t_{CY} -40	320		ns	
t_{CP}	Port Control Setup to $\overline{PROG}$	2/15 t_{CY} -80	100		ns	
t_{PC}	Port Control Hold to $\overline{PROG}$	4/15 t_{CY} -200	160		ns	
t_{PR}	$\overline{PROG}$ to P2 Input Valid	6/10 t_{CY} -120		700	ns	
t_{PF}	Input Data Hold from $\overline{PROG}$	1/10 t_{CY}	0	140	ns	
t_{DP}	Output Data Setup	2/5 t_{CY} -150	400		ns	
t_{PD}	Output Data Hold	1/10 t_{CY} -50	90		ns	
t_{PP}	$\overline{PROG}$ Pulse Width	7/10 t_{CY} -250	700		ns	
tPL	Port 2 I/O Setup to ALE	4/15 t_{CY} -200	160		ns	
t_{LP}	Port 2 I/O Hold to ALE	1/10 t_{CY} -100	40		ns	
t_{PV}	Port Output from ALE	3/10 t_{CY} +100		510	ns	
t_{CY}	Cycle Time	1/(f_{XTAL} x 15)	1.36		µs	2.5 µS t_{CY}MIN. FOR 8749H-8
t_{0PRR}	T0 Rep Rate	3/15 t_{CY}	270		ns	

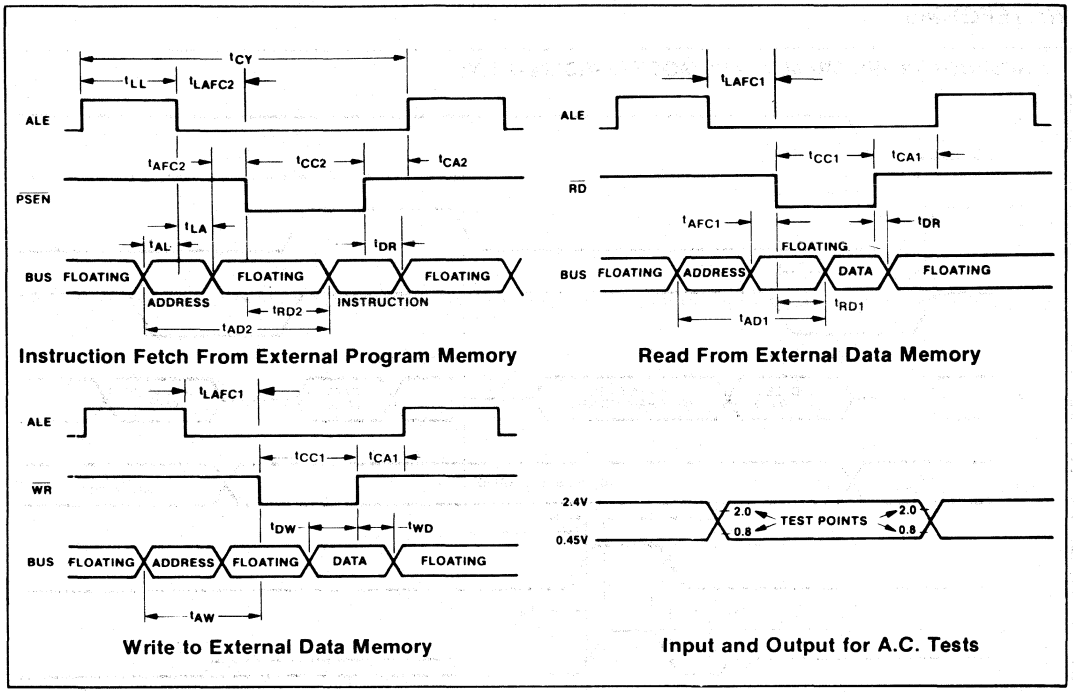
Notes:

1. Control Outputs CL = 80 pF
   BUS Outputs CL = 150pF
   When driven by an ext. clock, AC specs are based on 50% duty cycles.
2. BUS High Impedance Load 20pF

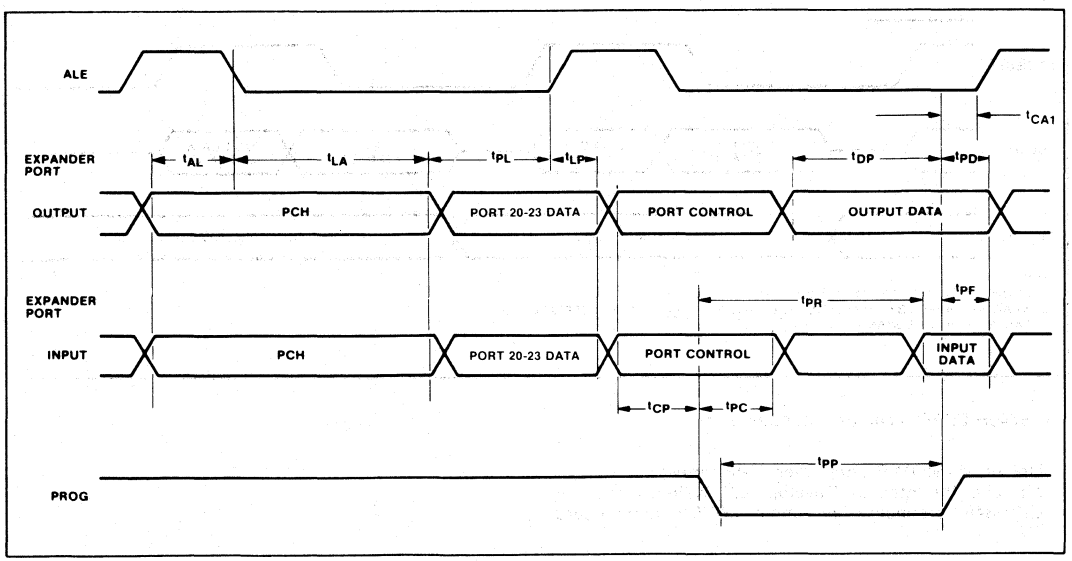
The 8749H exhibits some sensitivity to light. The RAM storage area can be altered when exposed to light. When operating the 8749H, the window must be covered with an opaque material.

AFN-01930A-5

WAVEFORMS



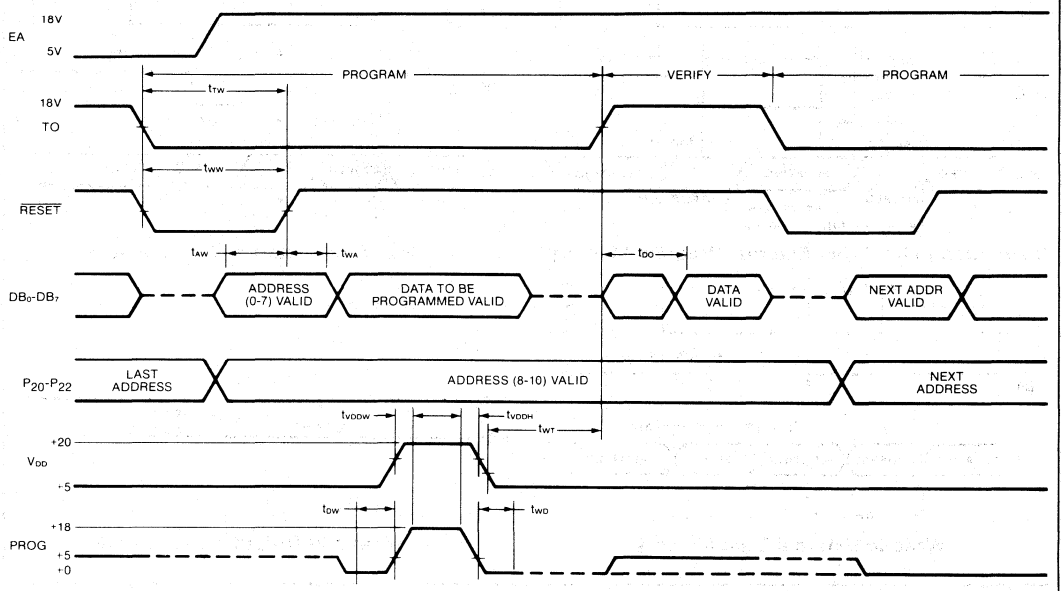
PORT 2 EXPANDER TIMING



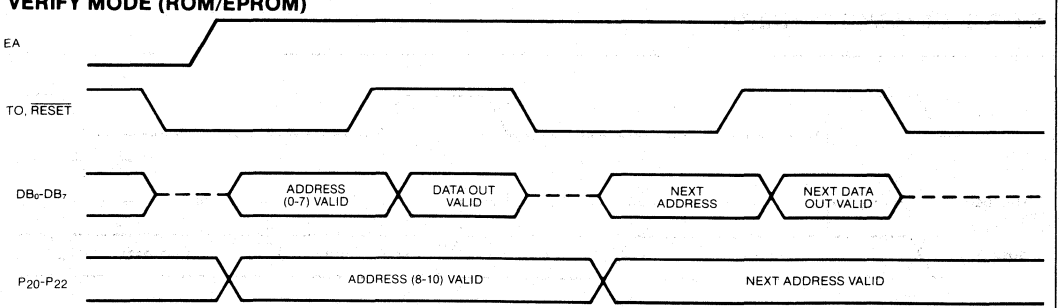
AFN-01930A-6

WAVEFORMS

COMBINATION PROGRAM/VERIFY MODE (EPROM's ONLY)



VERIFY MODE (ROM/EPROM)

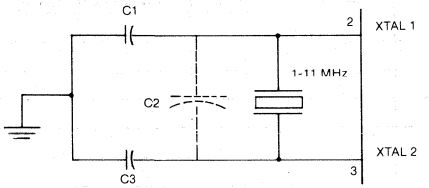


NOTES:
1. PROG MUST FLOAT IF EA IS LOW (i.e., ≠ 18V).
2. X_1 AND X_2 DRIVEN BY 3 MHz CLOCK WILL GIVE 5μsec t_{CY} THIS IS ACCEPTABLE FOR —8 PARTS AS WELL AS STANDARD PARTS.

The 8749H EPROM can be programmed by:

Universal PROM Programmer (UPP series) peripheral of the Intellec® Development System with a UPP-848 Personality Card and a UPP-549 adapter card.

CRYSTAL OSCILLATOR MODE



C1 = 5pF ± ½pF + STRAY < 5 pF
C2 = CRYSTAL ± STRAY < 8 pF
C3 = 20pF ± 1pF ± STRAY < 5 pF

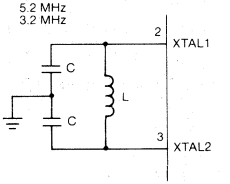
CRYSTAL SERIES RESISTANCE SHOULD BE LESS THAN 75Ω AT 6 MHz; LESS THAN 180Ω AT 3.6 MHz.

LC OSCILLATOR MODE

L	C	NOMINAL f
45 μH	20 pF	5.2 MHz
120 μH	20 pF	3.2 MHz

$f \approx \frac{1}{2\pi\sqrt{LC'}}$

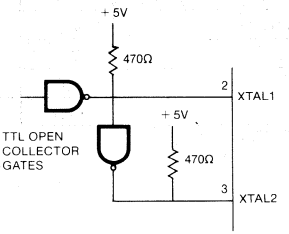
$C' = \frac{C + 3C_{PP}}{2}$



$C_{PP} \approx 5 - 10$ pF PIN-TO-PIN CAPACITANCE

EACH C SHOULD BE APPROXIMATELY 20 pF, INCLUDING STRAY CAPACITANCE

DRIVING FROM EXTERNAL SOURCE



FOR XTAL 1 AND XTAL 2 DEFINE "HIGH" AS VOLTAGES ABOVE 1.6V AND "LOW" AS VOLTAGES BELOW 1.6V. THE DUTY CYCLE REQUIREMENTS FOR EXTERNALLY DRIVING XTAL 1 AND XTAL 2 USING THE CIRCUIT SHOWN ABOVE ARE AS FOLLOWS:

FOR THE 8749H, XTAL 1 MUST BE HIGH 45-50% OF THE PERIOD AND XTAL 2 MUST BE HIGH 50-55% OF THE PERIOD.
RISE AND FALL TIMES MUST NOT EXCEED 20 ns.

PROGRAMMING, VERIFYING, AND ERASING THE 8749H 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
XTAL 1	Clock Input (1 to 3MHz)
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
P20-22	Address Input
V_{DD}	Programming Power Supply
PROG	Program Pulse Input

WARNING:

An attempt to program a missocketed 8749H 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, $\overline{RESET} = 0V$, TEST 0 = 5V, EA = 5V, BUS and PROG floating. P10 and P11 must be tied to ground.
2. Insert 8749H 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. $\overline{RESET} = 5V$ (latch address)
7. Data applied to BUS
8. $V_{DD} = 20V$ (programming power)
9. PROG = 0V 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. $\overline{RESET} = 0V$ and repeat from step 5
15. Programmer should be at conditions of step 1 when 8749H is removed from socket.

A.C. TIMING SPECIFICATION FOR PROGRAMMING (T_A = 25°C ± 5°C, V_{CC} = 5V ± 5%, V_{DD} = 20V ± 1V)

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}	$4t_{CY}$			
t_{VDDH}	V_{DD} Hold Time After PROG ↓	0			
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	2.0	μs	
t_{CY}	CPU Operation Cycle Time	5.0		μs	
t_{RE}	$\overline{RESET}$ Setup Time before EA↑	$4t_{CY}$			

NOTE: If Test 0 is high t_{DO} can be triggered by $\overline{RESET}$ ↑

D.C. SPECIFICATION FOR PROGRAMMING (T_A = 25°C ± 5°C, V_{CC} = 5V ± 5%, V_{DD} = 20V ± .5V)

Symbol	Parameter	Min	Max	Unit	Test Conditions
V_{DOH}	V_{DD} Program Voltage High Level	19.5	20.5	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		0.2	V	
V_{EAH}	EA Program or Verify Voltage High Level	17.5	18.5	V	
I_{DD}	V_{DD} High Voltage Supply Current		30.0	mA	
I_{PROG}	PROG High Voltage Supply Current		16.0	mA	
I_{EA}	EA High Voltage Supply Current		1.0	mA	

8243
MCS-48® INPUT/OUTPUT EXPANDER

- **Low Cost**
- **Simple Interface to MCS-48® Microcomputers**
- **Four 4-Bit I/O Ports**
- **AND and OR Directly to Ports**
- **24-Pin DIP**
- **Single 5V Supply**
- **High Output Drive**
- **Direct Extension of Resident 8048 I/O Ports**

The Intel® 8243 is an input/output expander designed specifically to provide a low cost means of I/O expansion for the MCS-48® family of single chip microcomputers. Fabricated in 5 volts NMOS, the 8243 combines low cost, single supply voltage and high drive current capability.

The 8243 consists of four 4-bit bidirectional static I/O ports and one 4-bit port which serves as an interface to the MCS-48 microcomputers. The 4-bit interface requires that only 4 I/O lines of the 8048 be used for I/O expansion, and also allows multiple 8243's to be added to the same bus.

The I/O ports of the 8243 serve as a direct extension of the resident I/O facilities of the MCS-48 microcomputers and are accessed by their own MOV, ANL, and ORL instructions.

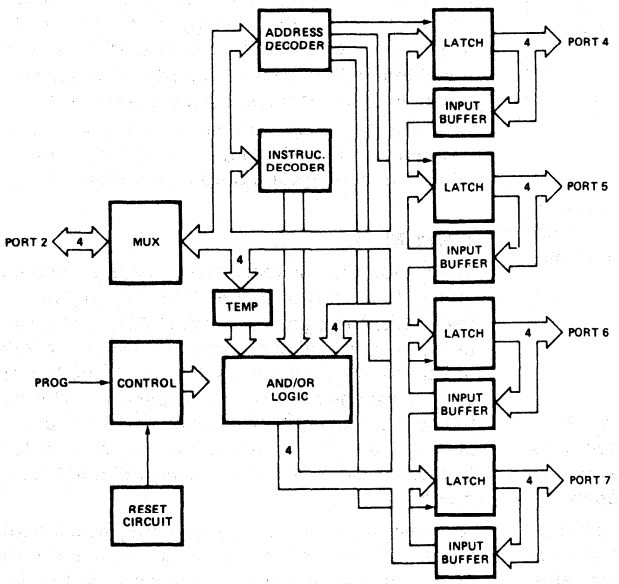


Figure 1. 8243 Block Diagram

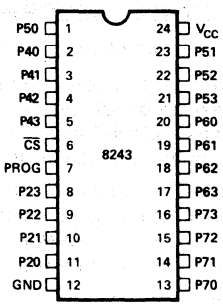


Figure 2. 8243 Pin Configuration

AFN-00214A-01

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 P50-P53 P60-P63 P70-P73	2-5 1, 23-21 20-17 13-16	Four (4) bit bi-directional I/O ports. 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 ports 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 8048 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 8048/8748/8035.

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 VCC drops below 1V.

		Address			Instruction
P21	P20	Code	P23	P22	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.

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.

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 ..... -0.5 V 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.

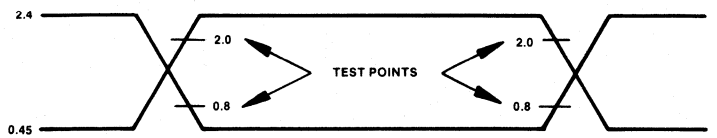
D.C. CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = 5V 10%

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} = 240μA
I_{IL1}	Input Leakage Ports 4-7	-10		20	μA	V_{in} = V_{CC} to OV
I_{IL2}	Input Leakage Port 2, CS, PROG	-10		10	μA	V_{in} = V_{CC} to OV
V_{OL3}	Output Low Voltage Port 2			.45	V	I_{OL} = 0.6 mA
I_{CC}	V_{CC} Supply Current		10	20	mA	
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

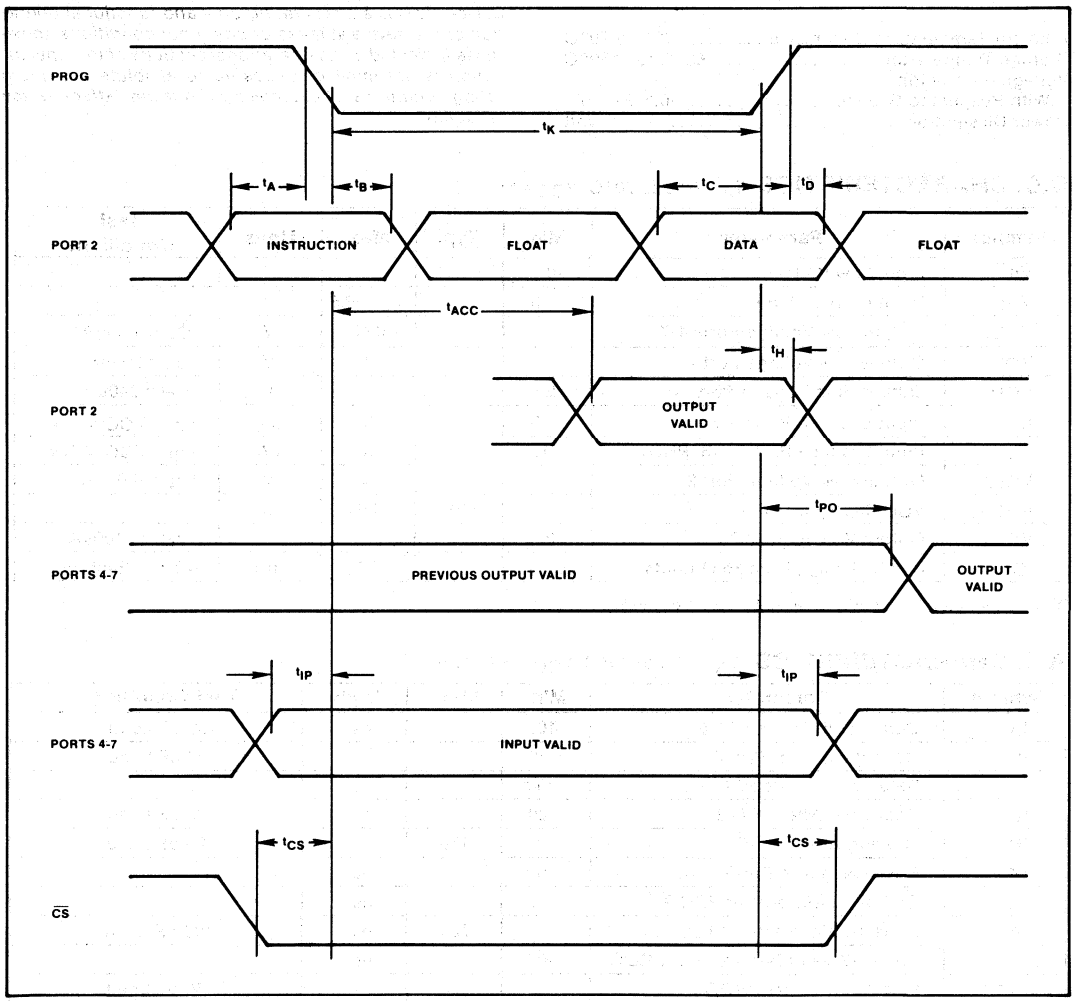
*See following graph for additional sink current capability

A.C. CHARACTERISTICS T_A = 0°C to 70°C, V_{CC} = 5V 10%

Symbol	Parameter	Min.	Max.	Units	Test Conditions
tA	Code Valid Before PROG	100		ns	80 pF Load
tB	Code Valid After PROG	60		ns	20 pF Load
tC	Data Valid Before PROG	200		ns	80 pF Load
tD	Data Valid After PROG	20		ns	20 pF Load
tH	Floating After PROG	0	150	ns	20 pF Load
tK	PROG Negative Pulse Width	700		ns	
tCS	CS Valid Before/After PROG	50		ns	
tPO	Ports 4-7 Valid After PROG		700	ns	100 pF Load
tLP1	Ports 4-7 Valid Before/After PROG	100		ns	
tACC	Port 2 Valid After PROG		650	ns	80 pF Load



WAVEFORMS


PROG
t_K
t_A
t_B
t_C
t_D
PORT 2
INSTRUCTION
FLOAT
DATA
FLOAT
t_{ACC}
t_H
PORT 2
OUTPUT VALID
t_{PO}
PORTS 4-7
PREVIOUS OUTPUT VALID
OUTPUT VALID
t_{IP}
t_{IP}
PORTS 4-7
INPUT VALID
t_{CS}
t_{CS}
$\overline{CS}$

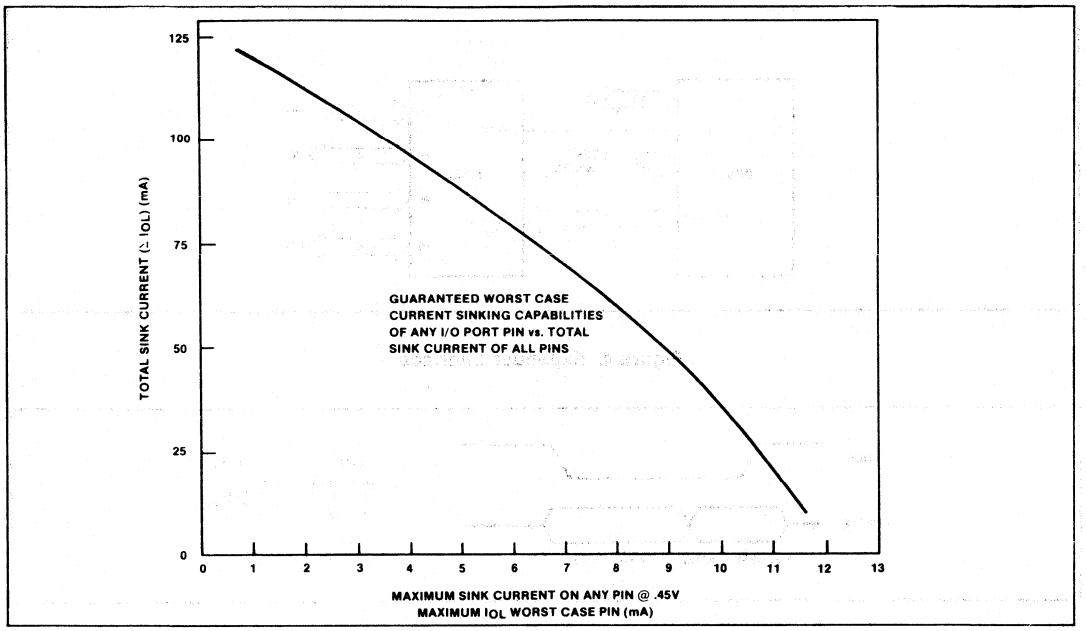


Figure 3

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 56mA. 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: A10 to 50KΩ pullup resistor to +5V should be added to 8243 outputs when driving to 5V CMOS directly.

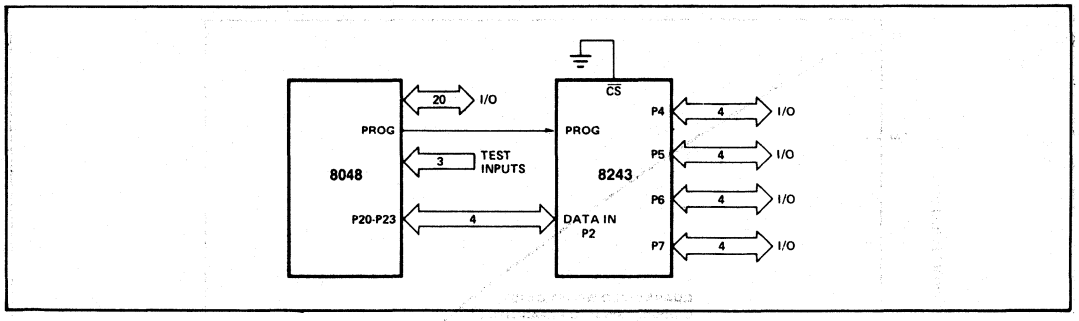


Figure 4. Expander Interface

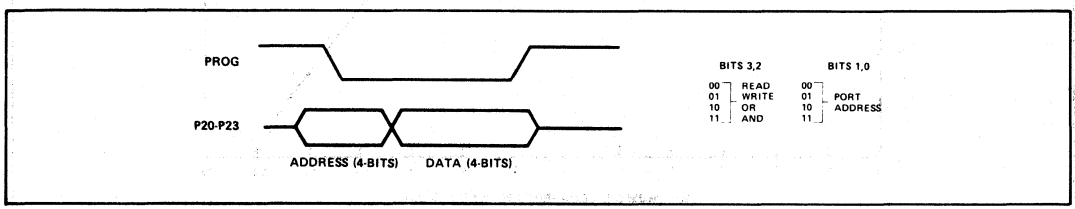


Figure 5. Output Expander Timing

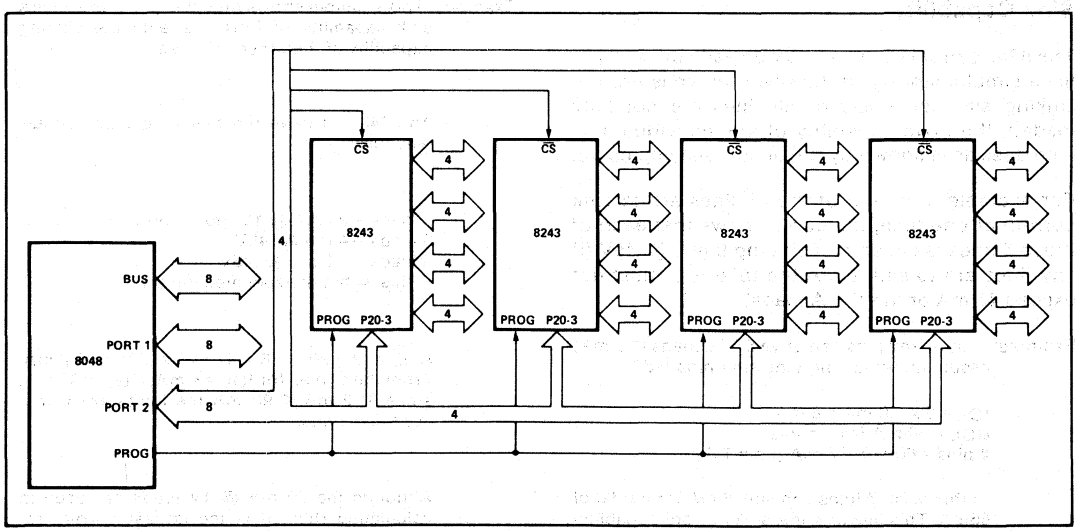


Figure 6. Using Multiple 8243's

I8048H
NEW HIGH PERFORMANCE HMOS SINGLE COMPONENT 8-BIT MICROCOMPUTER

INDUSTRIAL

- **I8048H Mask Programmable ROM**
- **RAM Power Down Mode**
- **Interchangeable with 8748**
- **8 MHz Operation**

- **8-Bit CPU, ROM, RAM, I/O in Single Package**
- **High Performance HMOS**
- **Reduced Power Consumption, Typically 50 mA, 100 mA at Extended Temperature**
- **1.9 μsec Cycle All instructions 1 or 2 Cycles**
- **Over 90 Instructions: 70% Single Byte**
- **-40° C to 85° C Operation**
- **1K x 8 ROM 64 x 8 RAM 27 I/O Lines**
- **Interval Timer/Event Counter**
- **Easily Expandable Memory and I/O**
- **Compatible with 8080/8085 Series Peripherals**
- **Two Single Level Interrupts**

The Intel® 8048H 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 8048H contains a 1K x 8 program memory, a 64 x 8 RAM data memory, 27 I/O lines, and an 8-bit timer/counter in addition to on-board oscillator and clock circuits. For systems that require extra capability the 8048H can be expanded using standard memories and MCS-80®/MCS-85® peripherals. The 8035HL is the equivalent of the 8048H without program memory and can be used with external ROM and RAM.

To reduce development problems to a minimum and provide maximum flexibility, a logically and functionally pin-compatible version of the 8048H with UV-erasable user-programmable EPROM program memory is available. The 8748 will emulate the I8048H up to 6 MHz clock frequency with minor differences.

The I8048H is fully compatible with the I8048 when operated at 6 MHz.

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.

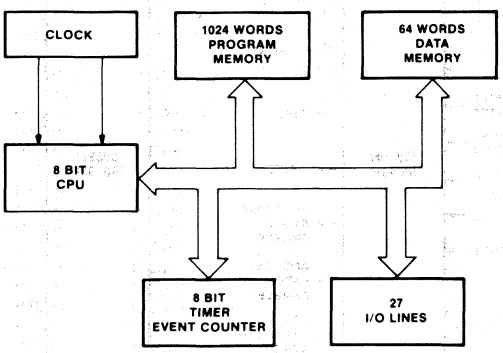


Figure 1. Block Diagram

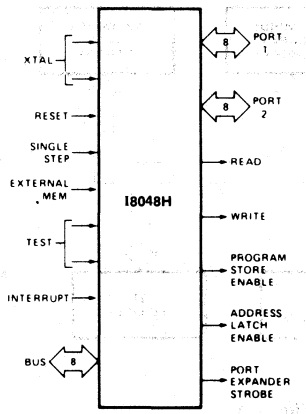


Figure 2. Logic Symbol

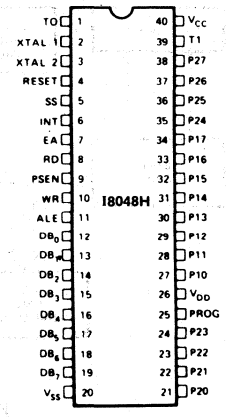


Figure 3. Pin Configuration

I8049H/8039H
NEW HIGH PERFORMANCE SINGLE COMPONENT 8-BIT MICROCOMPUTER

INDUSTRIAL

- **I8049H Mask Programmable ROM**
- **I8039H Requires External ROM or EPROM**
- **11 MHz Operation**

- **8-Bit CPU, ROM, RAM, I/O in Single Package**
- **Single 5V ± 10% Supply**
- **1.36 μsec Cycle; All instructions 1 or 2 Cycles**
- **Over 90 Instructions: 70% Single Byte**
- **Pin Compatible with 8048/8748**
- **2K x 8 ROM**
  128 x 8 RAM
  27 I/O Lines
- **Interval Timer/Event Counter**
- **Easily Expandable Memory and I/O**
- **Compatible with MCS Memory and I/O**
- **Single Level Interrupt**

The Intel® 8049/8039 is a totally self-sufficient 8-bit parallel computer fabricated on a single silicon chip using Intel's N-channel silicon gate MOS process.

The 8049 contains a 2K x 8 program memory, a 128 x 8 RAM data memory, 27 I/O lines, and an 8-bit timer/counter in addition to on-board oscillator and clock circuits. For systems that require extra capability, the 8049 can be expanded using standard memories and MCS-80®/MCS-85® peripherals. The 8039 is the equivalent to an 8049 without program memory.

To reduce development problems to a minimum and provide maximum flexibility, two interchangeable pin-compatible versions of this single component microcomputer exist: the 8049 with factory-programmed mask ROM program memory for low-cost high volume production, and the 8039 without program memory for use with external program memories in prototype and preproduction systems.

This microprocessor is designed to be an efficient controller as well as an arithmetic processor. The 8049 has 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 two bytes in length.

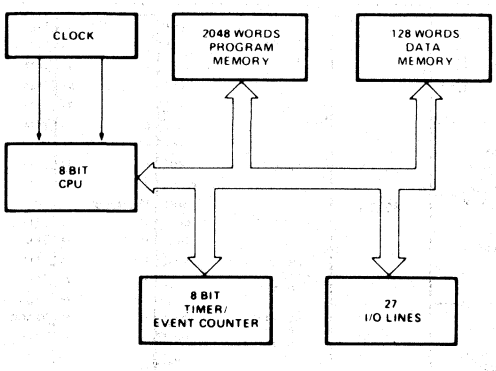


Figure 1. Block Diagram

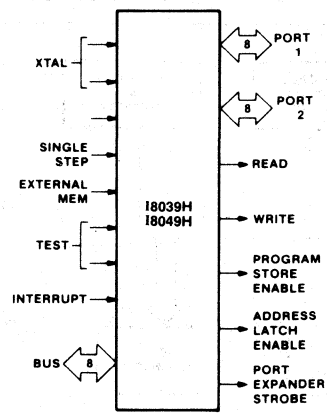


Figure 2. Logic Symbol

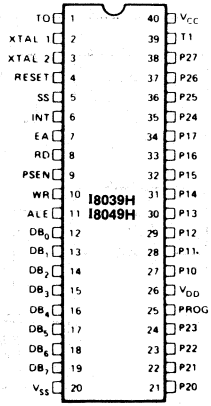


Figure 3. Pin Configuration

ADVANCE INFORMATION

I8031/8051
SINGLE-COMPONENT 8-BIT MICROCOMPUTER

INDUSTRIAL

- **8031 - Control Oriented CPU With RAM and I/O**
- **8051 - An 8031 With Factory Mask-Programmable ROM**

- **4K x 8 ROM**
- **128 x 8 RAM**
- **Four 8-Bit Ports, 32 I/O Lines**
- **Two 16-Bit Timer/Event Counters**
- **High-Performance Full-Duplex Serial Channel**
- **External Memory Expandable to 128K**
- **Compatible with MCS-80™/MCS-85™ Peripherals**
- **Boolean Processor**
- **MCS-48® Architecture Enhanced with:**
  - **Non-Paged Jumps**
  - **Direct Addressing**
  - **Four 8-Register Banks**
  - **Stack Depth Up to 128-Bytes**
  - **Multiply, Divide, Subtract, Compare**
- **Most Instructions Execute in 1 μs**
- **4 μs Multiply and Divide**

The Intel® 8031/8051 is a stand-alone, high-performance single-chip computer fabricated with Intel's highly-reliable +5 Volt, depletion-load, N-Channel, silicon-gate HMOS technology and packaged in a 40-pin DIP. It provides the hardware features, architectural enhancements and new instructions that are necessary to 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 8051 contains a non-volatile 4K x 8 read only program memory; a volatile 128 x 8 read/write data memory, 32 I/O lines; two 16-bit timer/counters; a five-source, two-priority-level, nested interrupt structure; a serial I/O port for either multi-processor communications, I/O expansion, or full duplex UART; and on-chip oscillator and clock circuits. The 8031 is identical, except that it lacks the program memory. For systems that require extra capability, the 8051 can be expanded using standard TTL compatible memories and the byte oriented MCS-80 and MCS-85 peripherals.

The 8051 microcomputer, like its 8048 predecessor, is efficient both as a controller and as an arithmetic processor. The 8051 has extensive facilities for binary and BCD arithmetic and excels in bit-handling capabilities. Efficient use of program memory results from an instruction set consisting of 44% one-byte, 41% two-byte, and 15% three-byte instructions. 58% of the instructions execute in 1 μs, 40% in 2 μs, and multiply and divide require only 4 μs. Among the many instructions added to the standard 8048 instruction set are multiply, divide, subtract and compare.

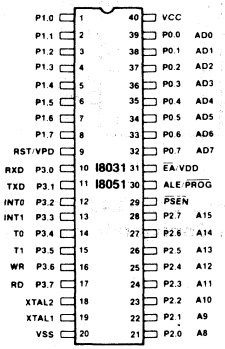


Figure 1. Pin Configuration

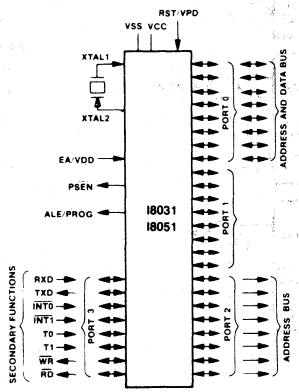


Figure 2. Logic Symbol

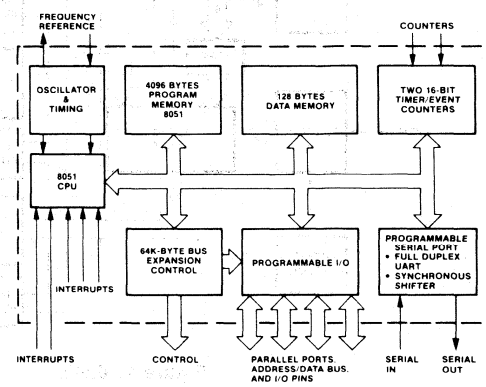


Figure 3. Block Diagram

I8243
MCS-48® INPUT/OUTPUT EXPANDER

- −40°C to +85°C Operation
- Low Cost
- Simple Interface to MCS-48® Microcomputers
- Four 4-Bit I/O Ports
- AND and OR Directly to Ports
- 24-Pin DIP
- Single 5V Supply
- High Output Drive
- Direct Extension of Resident 8048 I/O Ports

The Intel® 8243 is an input/output expander designed specifically to provide a low cost means of I/O expansion for the MCS-48® family of single chip microcomputers. Fabricated in 5 volts NMOS, the 8243 combines low cost, single supply voltage and high drive current capability.

The 8243 consists of four 4-bit bidirectional static I/O ports and one 4-bit port which serves as an interface to the MCS-48 microcomputers. The 4-bit interface requires that only 4 I/O lines of the 8048 be used for I/O expansion, and also allows multiple 8243's to be added to the same bus.

The I/O ports of the 8243 serve as a direct extension of the resident I/O facilities of the MCS-48 microcomputers and are accessed by their own MOV, ANL, and ORL instructions.

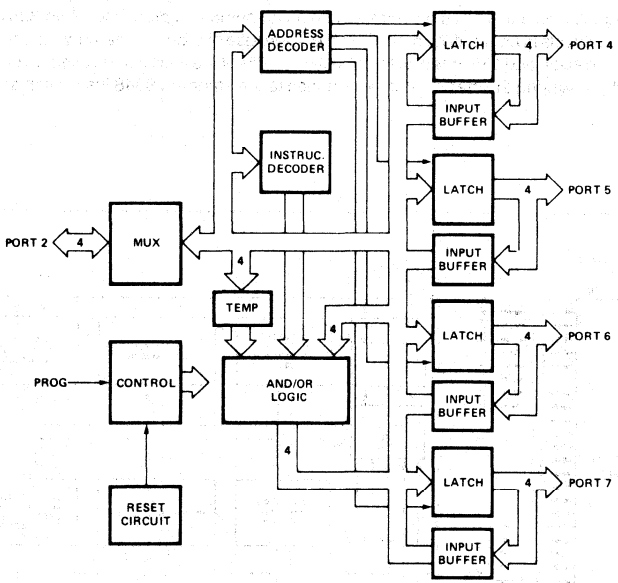


Figure 1. 8243
Block Diagram

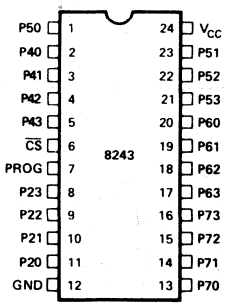


Figure 2. 8243
Pin Configuration

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 P50-P53 P60-P63 P70-P73	2-5 1, 23-21 20-17 13-16	Four (4) bit bi-directional I/O ports. 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 ports 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 8048 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 8048/8748/8035.

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 VCC 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.

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.

AFN-00861A-02

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias . . . . . −40°C to +85°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.

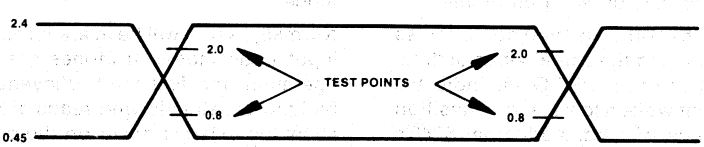
D.C. CHARACTERISTICS $T_A = -40°C$ to $+85°C$, $V_{CC} = 5V \pm 10\%$

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		VCC+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} = 240μA
I_{IL1}	Input Leakage Ports 4-7	-10		20	μA	V_{in} = VCC to OV
I_{IL2}	Input Leakage Port 2, CS, PROG	-10		10	μA	V_{in} = VCC to OV
V_{OL3}	Output Low Voltage Port 2			.45	V	I_{OL} = 0.6 mA
I_{CC}	VCC Supply Current		10	20	mA	
V_{OH2}	Output Voltage Port 2	2.4				I_{OH} = 100μA
I_{OL}	Sum of all I_{OL} from 16 Outputs			80	mA	4.5 mA Each Pin

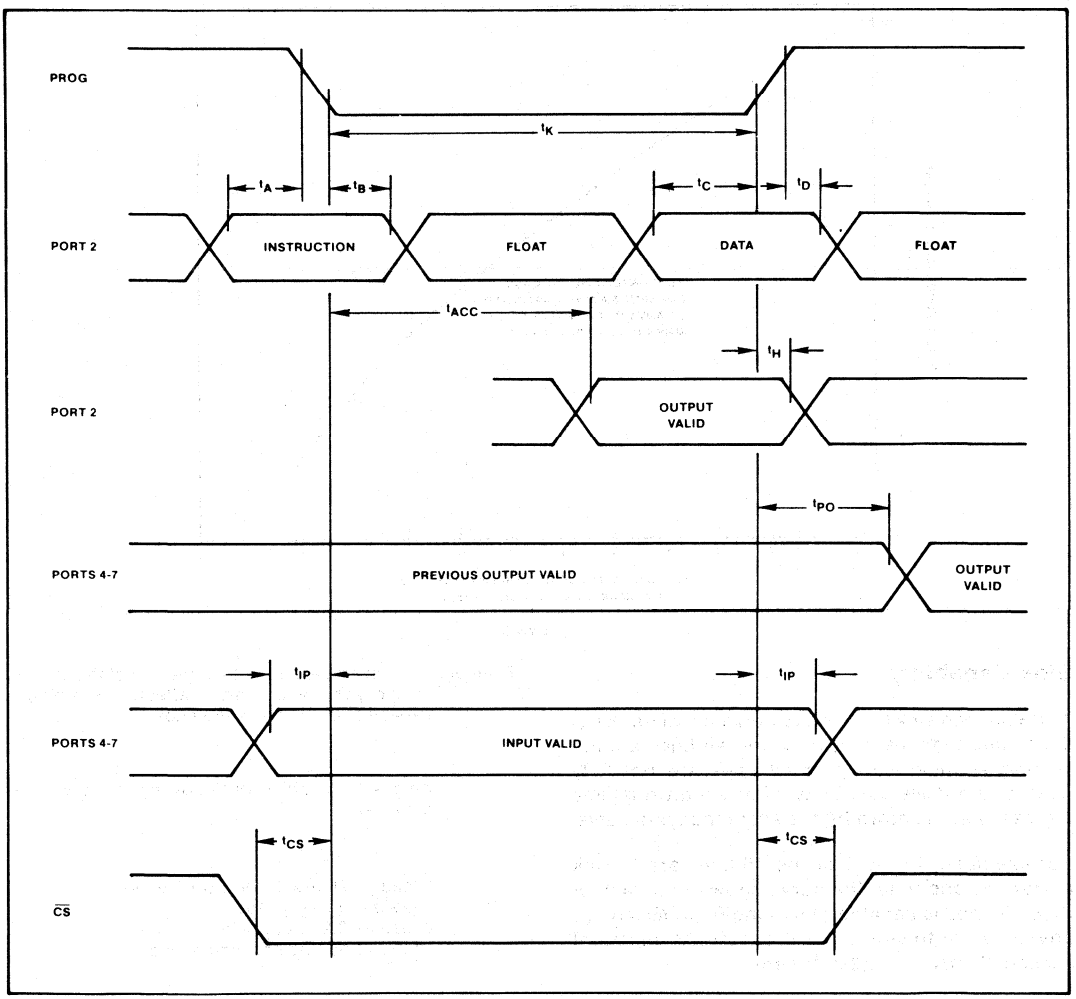
*See following graph for additional sink current capability

A.C. CHARACTERISTICS $T_A = -40°C$ to $+85°C$, $V_{CC} = 5V \pm 10\%$

Symbol	Parameter	Min.	Max.	Units	Test Conditions
tA	Code Valid Before PROG	100		ns	80 pF Load
tB	Code Valid After PROG	60		ns	20 pF Load
tC	Data Valid Before PROG	200		ns	80 pF Load
tD	Data Valid After PROG	20		ns	20 pF Load
tH	Floating After PROG	0	150	ns	20 pF Load
tK	PROG Negative Pulse Width	700		ns	
tCS	CS Valid Before/After PROG	50		ns	
tPO	Ports 4-7 Valid After PROG		700	ns	100 pF Load
tLP1	Ports 4-7 Valid Before/After PROG	100		ns	
tACC	Port 2 Valid After PROG		650	ns	80 pF Load



WAVEFORMS



AFN-00861A-04

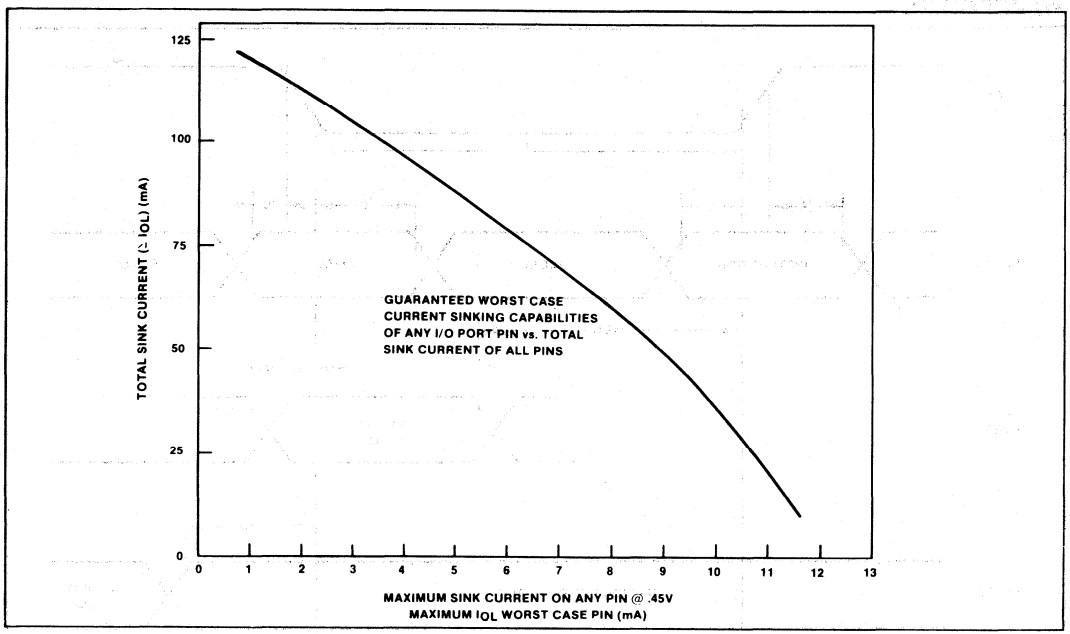


Figure 3

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 IOL 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?

IOL = 5 x 1.6 mA = 8 mA
ϵIOL = 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 56mA. 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?

ϵIOL = (2 x 20) + (8 x 4) + (6 x 3.2) = 91.2 mA.
From the curve: for IOL = 4 mA, ϵIOL ≈ 93 mA.
since 91.2 mA < 93 mA the loads are within specified limits.

Although the 20 mA @ 1V loads are used in calculating ϵIOL, it is the largest current required @ .45V which determines the maximum allowable ϵIOL.

NOTE: A10 to 50KΩ pullup resistor to +5V should be added to 8243 outputs when driving to 5V CMOS directly.

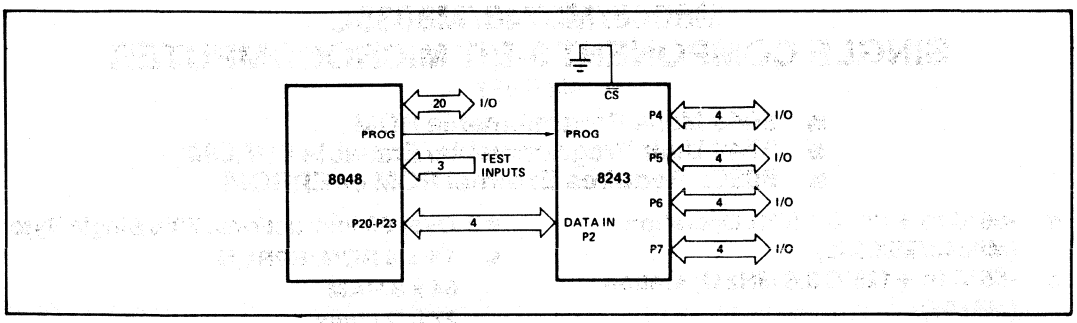


Figure 4. Expander Interface

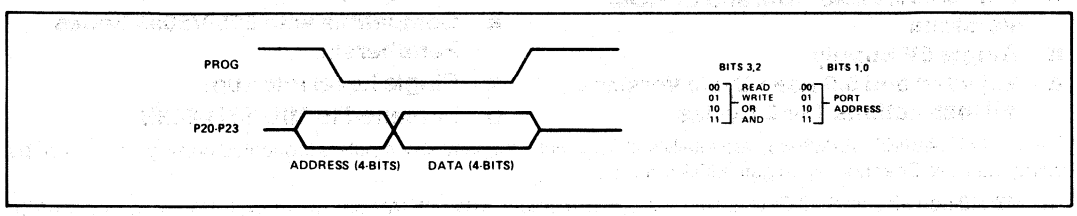


Figure 5. Output Expander Timing

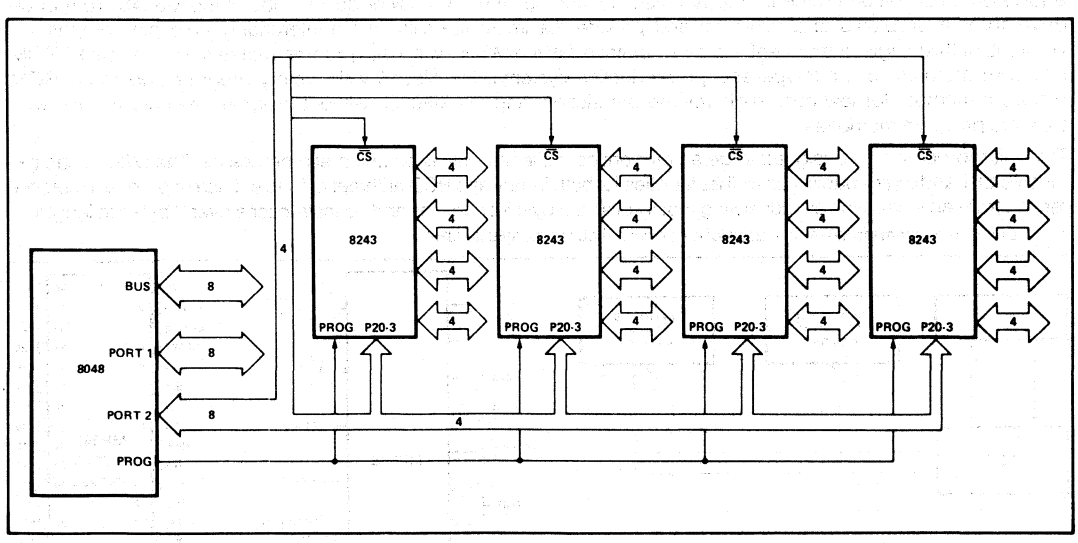


Figure 6. Using Multiple 8243's

M8048/M8748/M8035L
SINGLE COMPONENT 8-BIT MICROCOMPUTER

MILITARY

- **8048 Mask Programmable ROM**
- **8748 User Programmable/Erasable EPROM**
- **8035L Requires External ROM or EPROM**

- **-55°C to +125°C 6 MHz Operation (M8048/M8035L)**
- **-55°C to +125°C 3.6 MHz Operation (M8748)**
- **8-Bit CPU, ROM, RAM, I/O in Single Package**
- **Interchangeable ROM and EPROM Versions**
- **Single 5V Supply**
- **2.5 μsec and 5.0 μsec Cycle Versions All Instructions 1 or 2 Cycles.**
- **Over 90 Instructions: 70% Single Byte**
- **1K x 8 ROM/EPROM 64 x 8 RAM 27 I/O Lines**
- **Interval Timer/Event Counter**
- **Easily Expandable Memory and I/O**
- **Compatible with 8080/8085 Series Peripherals**
- **Single Level Interrupt**
- **Screened to MIL-STD-883B**

The Intel M8048/M8748/M8035L are totally self-sufficient 8-bit parallel computers fabricated on single silicon chips using Intel's N-Channel silicon gate MOS process.

The M8048 contains an 8-bit CPU, a 1K × 8 program memory, a 64 × 8 RAM data memory, 27 I/O lines, and an 8-bit timer/counter in addition to on-board oscillator and clock circuits. For systems that require extra capability, the M8048 can be expanded using standard memories and MCS-80®/MCS-85® peripherals. The M8035L is the equivalent of an M8048 without program memory, and has the RAM power down mode of the M8048. To reduce development problems to a minimum and provide maximum flexibility, three interchangeable pin-compatible* versions of this single component micro-computer exist: the M8748 with user-programmable and erasable EPROM program memory for prototype and preproduction systems, the M8048 with factory-programmed mask ROM program memory for low cost, high volume production, and the M8035L without program memory for use with external program memories.

This microprocessor is designed to be an efficient controller as well as an arithmetic processor. The M8048 has 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.

*V_{DD} is used to program the M8748 and used for low power standby on the M8048/8035L.

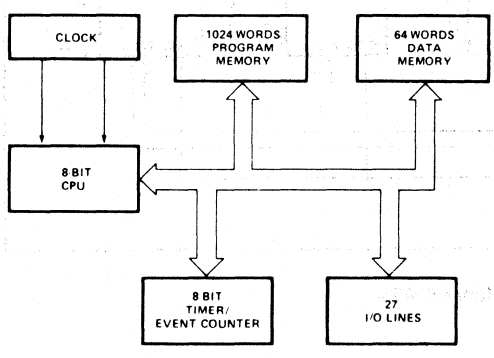


Figure 1. Block Diagram

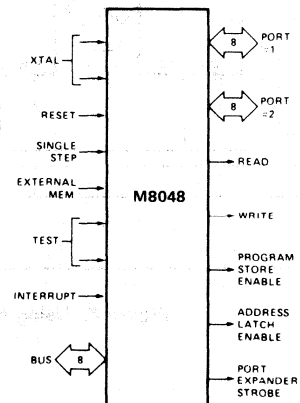


Figure 2. Logic Symbol

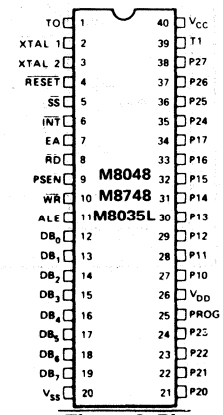


Figure 3. Pin Configuration

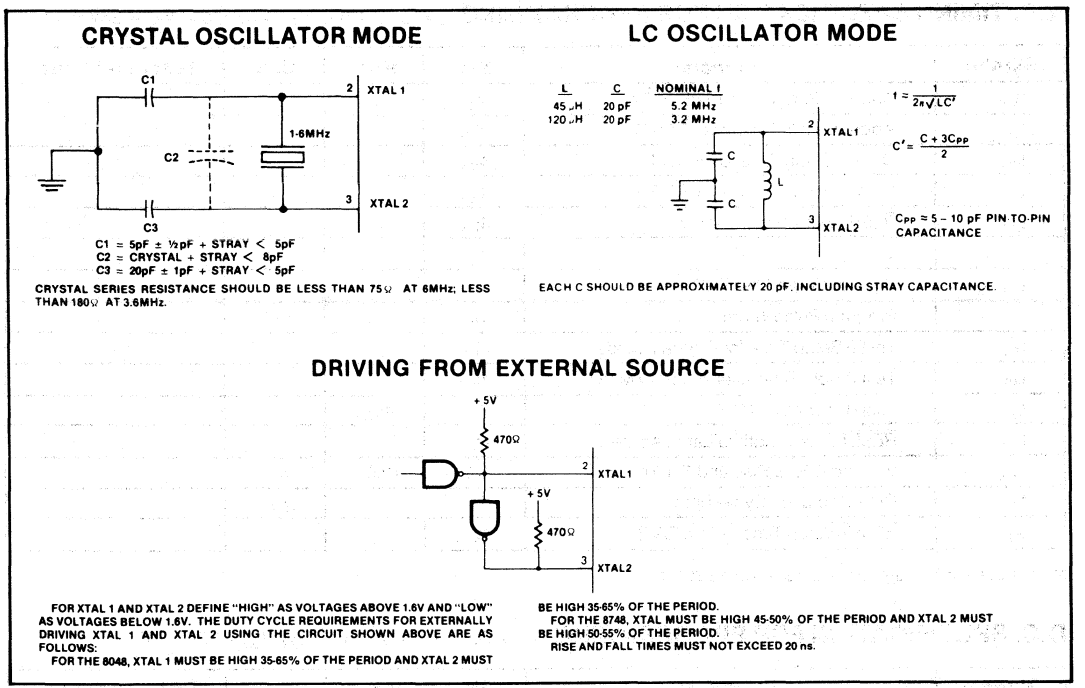


Figure 4

PROGRAMMING, VERIFYING, AND ERASING THE 8748 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
XTAL 1	Clock Input (1 to 6MHz)
$\overline{\text{Reset}}$	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
P20-1	Address Input
V_{DD}	Programming Power Supply
PROG	Program Pulse Input

WARNING:

An attempt to program a missocketed 8748 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, $\overline{\text{RESET}}$ = 0v, TEST 0 = 5v, EA = 5v, BUS and PROG floating.
2. Insert 8748 in programming socket
3. TEST 0 = 0v (select program mode)
4. EA = 23v (activate program mode)
5. Address applied to BUS and P20-1
6. $\overline{\text{RESET}}$ = 5v (latch address)
7. Data applied to BUS
8. V_{DD} = 25v (programming power)
9. PROG = 0v followed by one 50ms pulse to 23v
10. V_{DD} = 5v
11. TEST 0 = 5v (verify mode)
12. Read and verify data on BUS
13. TEST 0 = 0v
14. $\overline{\text{RESET}}$ = 0v and repeat from step 5
15. Programmer should be at condition of step 1 when 8748 is removed from socket.

A.C. TIMING SPECIFICATION FOR PROGRAMMING $T_A = 25°C \pm 5°C$, $V_{CC} = 5V \pm 5\%$, $V_{DD} = 25V \pm 1V$

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}	$4t_{CY}$			
t_{VDDH}	V_{DD} Hold Time After PROG ↓	0			
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	2.0	μs	
t_{CY}	CPU Operation Cycle Time	5.0		μs	
t_{RE}	$\overline{RESET}$ Setup Time Before EA ↑	$4t_{CY}$			

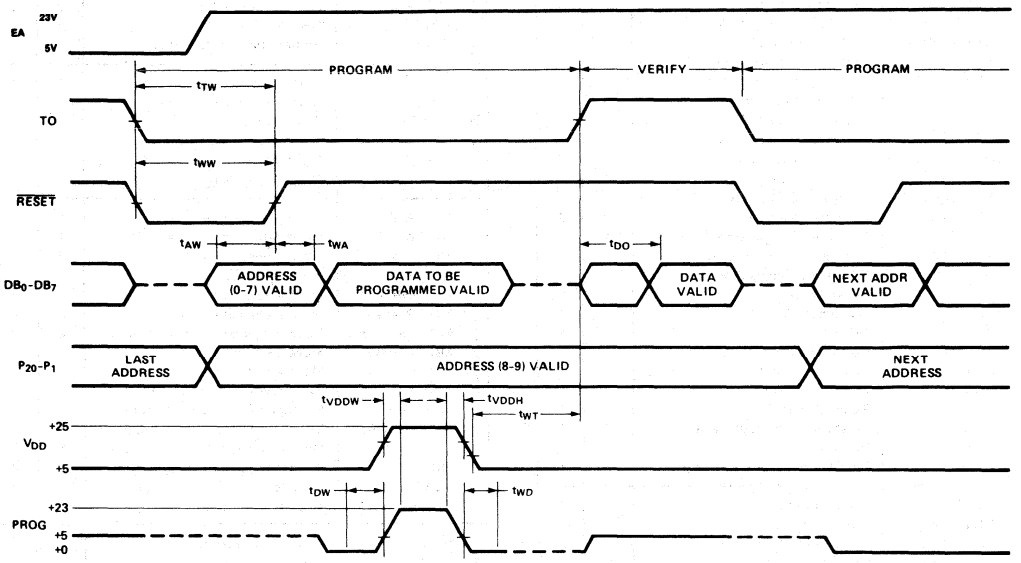
Note: If Test 0 is high t_{DO} can be triggered by $\overline{RESET}$ ↑.

D.C. SPECIFICATION FOR PROGRAMMING $T_A = 25°C \pm 5°C$, $V_{CC} = 5V \pm 5\%$, $V_{DD} = 25V \pm 1V$

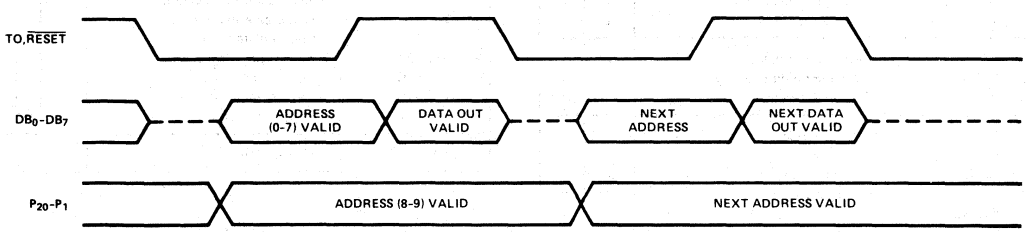
Symbol	Parameter	Min.	Max.	Unit	Test Conditions
V_{DOH}	V_{DD} Program Voltage High Level	24.0	26.0	V	
V_{DDL}	V_{DD} Voltage Low Level	4.75	5.25	V	
V_{PH}	PROG Program Voltage High Level	21.5	24.5	V	
V_{PL}	PROG Voltage Low Level		0.2	V	
V_{EAH}	EA Program or Verify Voltage High Level	21.5	24.5	V	8748
V_{EAH1}	EA1 Verify Voltage High Level	11.4	12.6	V	8048
V_{EAL}	EA Voltage Low Level		5.25	V	
I_{DD}	V_{DD} High Voltage Supply Current		30.0	mA	
I_{PROG}	PROG High Voltage Supply Current		16.0	mA	
I_{EA}	EA High Voltage Supply Current		1.0	mA	

WAVEFORMS

COMBINATION PROGRAM/VERIFY MODE (EPROM'S ONLY)



VERIFY MODE (ROM/EPROM)



NOTES:
1. **PROG MUST FLOAT IF EA IS LOW (i.e., ≠23V), OR IF T0 = 5V FOR THE 8748. FOR THE 8048 PROG MUST ALWAYS FLOAT.**
2. **X_1 AND X_2 DRIVEN BY 3 MHz CLOCK WILL GIVE 5μsec t_{CY}. THIS IS ACCEPTABLE FOR ALL PARTS.**

The 8748 EPROM can be programmed by either of two Intel products:

1. PROMPT-48 Microcomputer Design Aid, or
2. Universal PROM Programmer (UPP Series) peripheral of the Intellec® Development System with a UPP-848 Personality Card.

Table 1. 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
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
	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 $\overline{INT}$ = 0	2	2
	JBb addr	Jump on accumulator bit	2	2

	Mnemonic	Description	Bytes	Cycles
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
	XCH A, R	Exchange A and register	1	1
	XCHA, @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
	ENT0 CLK	Enable clock output on T0	1	1
	NOP	No operation	1	1

Table 2. Pin Description

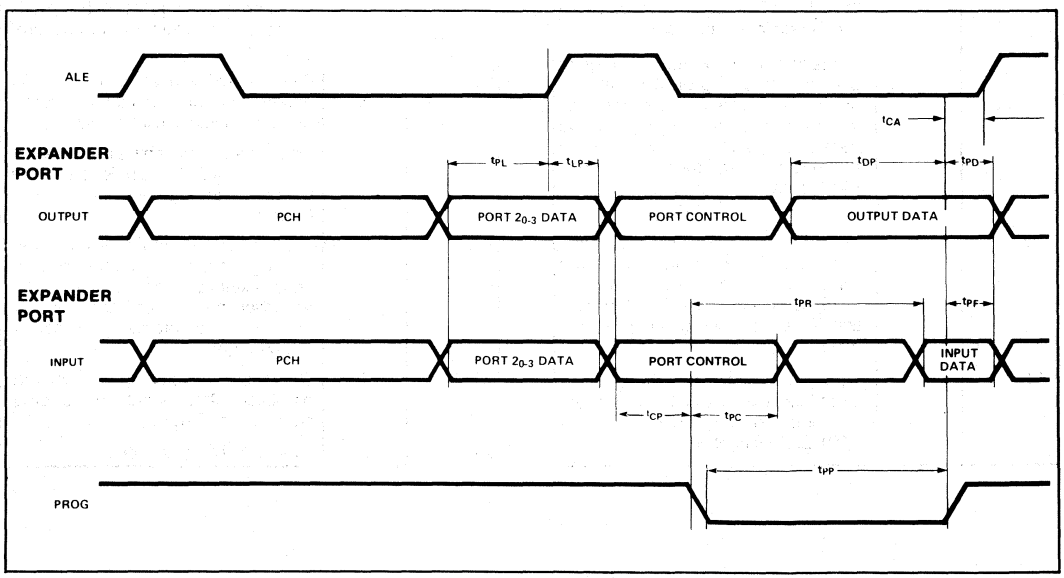
Symbol	Pin No.	Function
V_{SS}	20	Circuit GND potential
V_{DD}	26	Programming power supply; +25V during program, +5V during operation for both ROM and PROM. Low power standby pin in 8048 and 8035L.
V_{CC}	40	Main power supply; +5V during operation and programming.
PROG	25	Program pulse (+23V) input pin during 8748 programming.
		Output strobe for 8243 I/O expander.
P10-P17 Port 1	27-34	8-bit quasi-bidirectional port.
P20-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.
DB_0-DB_7 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 JNT0. T0 can be designated as a clock output using ENT0 CLK instruction. T0 is also used during programming.
T1	39	Input pin testable using the JT1, and JNT1 instructions. Can be designated the timer/counter input using the STRT CNT instruction.

Symbol	Pin No.	Function
$\overline{INT}$	6	Interrupt input. Initiates an interrupt if interrupt is enabled. Interrupt is disabled after a reset. Also testable with conditional jump instruction. (Active low)
$\overline{RD}$	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)
$\overline{RESET}$	4	Input which is used to initialize the processor. Also used during PROM programming verification, and power down. (Active low) (Non TTL V_{IH})
$\overline{WR}$	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.
$\overline{PSEN}$	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 junction with 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.

A.C. CHARACTERISTICS (PORT 2 TIMING) T_A = 55°C to 125°C, V_{CC} = +5V ± 10%, V_{SS} = 0V

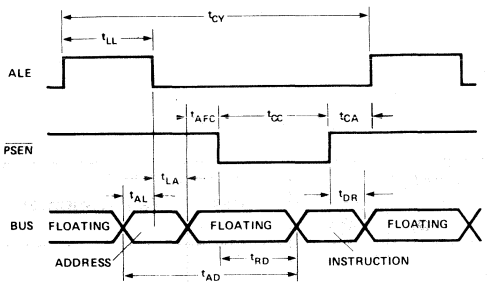
Symbol	Parameter	Min.	Max.	Unit	Test Conditions
t_{CP}	Port Control Setup Before Falling Edge of $\overline{PROG}$	115		ns	
t_{PC}	Port Control Hold After Falling Edge of $\overline{PROG}$	65		ns	
t_{PR}	PROG to Time P2 Input Must Be Valid		860	ns	
t_{PF}	Input Data Hold Time	0	160	ns	
t_{DP}	Output Data Setup Time	230		ns	
t_{PD}	Output Data Hold Time	25		ns	
t_{PP}	$\overline{PROG}$ Pulse Width	920		ns	
t_{PL}	Port 2 I/O Data Setup	300		ns	
t_{LP}	Port 2 I/O Data Hold	120		ns	

PORT 2 TIMING

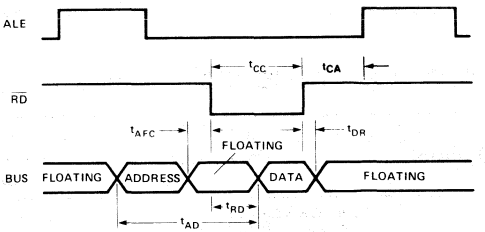


WAVEFORMS

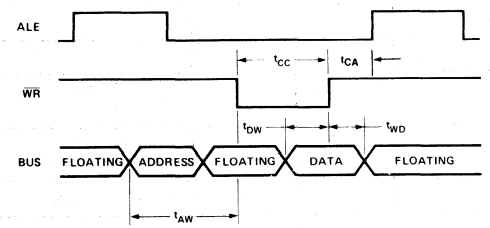
Instruction Fetch From External Program Memory



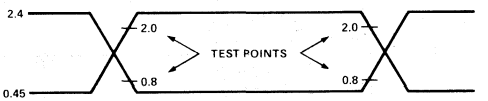
Read From External Data Memory



Write to External Data Memory



Input and Output Waveforms for A.C. Tests



A.C. CHARACTERISTICS T_A = -55°C to 125°C, $V_{CC} = V_{DD}$ = +5V ± 10%, V_{SS} = 0V

Symbol	Parameter	M8048 M8035L		M8748		Unit	Conditions (Note 1)
		Min.	Max.	Min.	Max.		
t_{LL}	ALE Pulse Width	200		300		ns	
t_{AL}	Address Setup to ALE	120		120		ns	
t_{LA}	Address Hold from ALE	80		80		ns	
t_{CC}	Control Pulse Width ($\overline{PSEN}$, $\overline{RD}$, $\overline{WR}$)	400		600		ns	
t_{DW}	Data Setup before $\overline{WR}$	420		600		ns	
t_{WD}	Data Hold After $\overline{WR}$	80		120		ns	C_L = 20pF
t_{CY}	Cycle Time	2.5	15.0	4.17	15.0	μs	(3.6 MHz XTAL 8748)
t_{DR}	Data Hold	0	200	0	200	ns	
t_{RD}	$\overline{PSEN}$, $\overline{RD}$ to Data In		400		600	ns	
t_{AW}	Address Setup to $\overline{WR}$	230		260		ns	
t_{AD}	Address Setup to Data In		600		900	ns	
t_{AFC}	Address Float to $\overline{RD}$, $\overline{PSEN}$	-40		-60		ns	
t_{CA}	Control Pulse to ALE	10		10		ns	

Note 1: Control outputs: C_L = 80 pF t_{CY} = 2.5 μs for 8048/8035L
BUS Outputs: C_L = 150 pF 4.17 μs for 8748

ABSOLUTE MAXIMUM RATINGS*

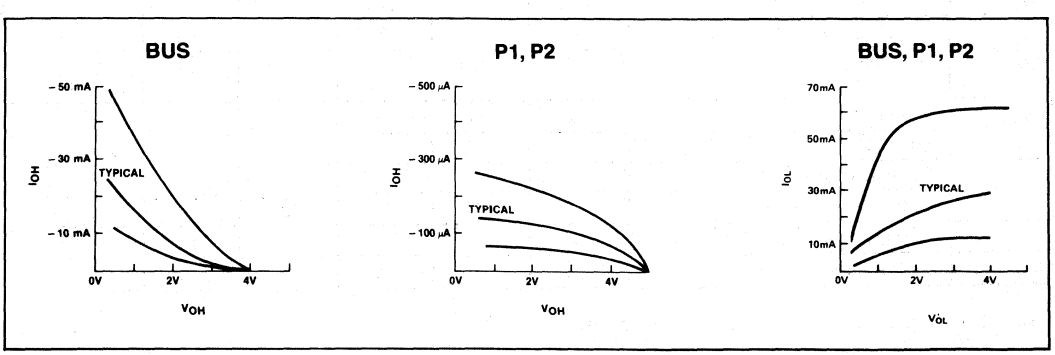
Ambient Temperature Under Bias
- 8748 ..... −55°C to +125°C
- 8048/8035L ..... −55°C to +125°C

Storage Temperature ..... −65°C to +125°C
Voltage On Any Pin With Respect to Ground ..... −0.5 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 the device at these or any other conditions above those indicated in the operational sections of this specification is not implied.

D.C. CHARACTERISTICS $T_A = -55°C$ to 125°C, $V_{CC} = V_{DD} = +5V \pm 10\%$, $V_{SS} = 0V$

Symbol	Parameter	Limits			Unit	Test Conditions
		Min.	Typ.	Max.		
V_{IL}	Input Low Voltage (All Except $\overline{RESET}$, X1, X2)	−.5		.7	V	
V_{IL1}	Input Low Voltage ($\overline{RESET}$, X1, X2)	−.5		.5	V	
V_{IH}	Input High Voltage (All Except XTAL1, XTAL2, $\overline{RESET}$)	2.3		V_{CC}	V	
V_{IH1}	Input High Voltage ($\overline{RESET}$, X1, X2)	3.8		V_{CC}	V	
V_{OL}	Output Low Voltage (BUS, $\overline{RD}$, $\overline{WR}$, $\overline{PSEN}$, ALE)			.45	V	$I_{OL} = 1.2mA$
V_{OL1}	Output Low Voltage (All Other Outputs)			.45	V	$I_{OL} = 0.8mA$
V_{OH}	Output High Voltage (BUS)	2.4			V	$I_{OH} = -240\mu A$
V_{OH1}	Output High Voltage ($\overline{RD}$, $\overline{WR}$, $\overline{PSEN}$, ALE)	2.4			V	$I_{OH} = -50\mu A$
V_{OH2}	Output High Voltage (All Other Outputs)	2.4			V	$I_{OH} = -30\mu A$
I_{LI}	Input Leakage Current (T1, $\overline{INT}$)			±10	μA	$V_{SS} \le V_{IN} \le V_{CC}$
I_{LI1}	Input Leakage Current (P10-P17, P20-P27, EA, $\overline{SS}$)			−700	μA	$V_{SS} + .45 \le V_{IN} \le V_{CC}$
I_{LO}	Output Leakage Current (BUS, TO) (High Impedance State)			±10	μA	$V_{SS} + .45 \le V_{IN} \le V_{CC}$
I_{DD}	V_{DD} Supply Current		10	25	mA	
$I_{DD} + I_{CC}$	Total Supply Current		80	155	mA	



AFN-00780A-09

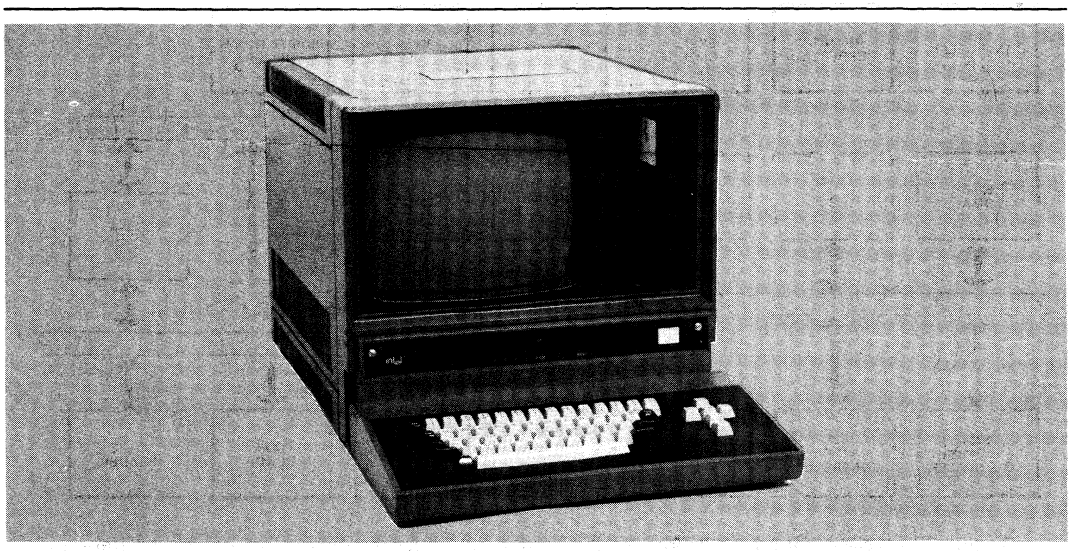
Development Support Tools

9

MODEL 225
INTELLEC® SERIES II/85
MICROCOMPUTER DEVELOPMENT SYSTEM

- **Complete microcomputer development system for MCS®-86, MCS®-85, MCS®-80, and MCS®-48 microprocessor families**
- **High performance 8085A-2 CPU, 64K bytes RAM memory, and 4K bytes ROM memory**
- **Self-test diagnostic capability**
- **Built-in interfaces for high speed paper tape reader/punch, printer, and universal PROM programmer**
- **Integral 250K byte floppy disk drive with total storage capacity expandable to over 2M bytes of floppy disk storage and 7.3M bytes of hard disk storage**
- **Powerful ISIS-II Disk Operating System with relocating macroassembler, linker, locater, and CRT based editor CREDIT**
- **Supports PL/M, FORTRAN, BASIC, PASCAL and COBOL high level languages**
- **Software compatible with previous Intellec® systems**

The Intellec Series II/85 Model 225 Microcomputer Development System is a performance enhanced, complete microcomputer development system integrated into one compact package. The Model 225 includes a CPU with 64K bytes of RAM, 4K bytes of ROM, a 2000-character CRT, detachable full ASCII keyboard with cursor controls and upper/lower case capability, and a 250K-byte floppy disk drive. Powerful ISIS-II Disk Operating System software allows the Model 225 to be used quickly and efficiently for assembling and debugging programs for Intel's MCS-86, MCS-85, MCS-80, or MCS-48 microprocessor families. ISIS-II performs all file handling operations for the user, leaving him free to concentrate on the details of his own application. When used with an optional in-circuit emulator (ICE™) module, the Model 225 provides all of the hardware and software development tools necessary for the rapid development of a microcomputer-based product. Optional storage peripherals provide over 2 million bytes of floppy disk, and 7.3 million of hard disk storage capacity.



121599-001 Rev. A

NCTIONAL DESCRIPTION

ardware Components

he Intellec Series II/85 Model 225 is a highly-integrated microcomputer development system consisting of a CRT chassis with a 6-slot cardcage, power supply, fans, cables, single floppy disk drive, and two printed circuit cards. A separate, full ASCII keyboard is connected with a cable. A block diagram of the Model 225 is shown in Figure 1.

CPU Cards — The master CPU card contains its own microprocessor, memory, I/O, interrupt and bus interface circuitry implemented with Intel's high technology LSI components. Known as the integrated processor card (IPC), it occupies the first slot in the cardcage. A second slave CPU card is responsible for all remaining I/O control including the CRT and keyboard interface. This card, mounted on the rear panel, also contains its own microprocessor, RAM and ROM memory, and I/O interface logic, thus, in effect, creating a dual processor environment. Known as the I/O controller (IOC), the slave CPU card communicates with the IPC over an 8-bit bidirectional data bus.

Expansion — Five remaining slots in the cardcage are available for system expansion. Additional expansion of 4 slots can be achieved through the addition of an Intellec Series II expansion chassis.

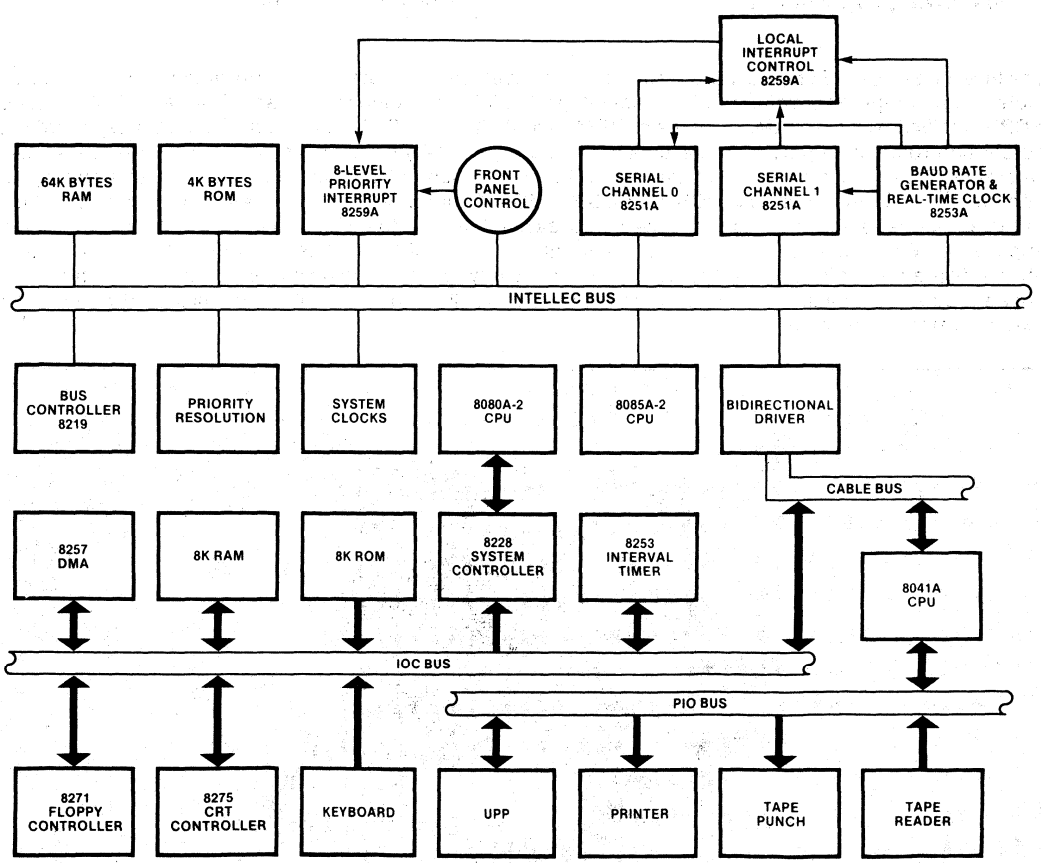


Figure 1. Intellec® Series II/85 Model 225 Microcomputer Development System Block Diagram

System Components

The heart of the IPC is an Intel NMOS 8-bit microprocessor, the 8085A-2, running at 4.0 MHz. 64K bytes of RAM memory are provided on the board using 16K RAMs. 4K of ROM is provided, preprogrammed with system bootstrap "self-test" diagnostics and the Intellec Series II/85 System Monitor. The eight-level vectored priority interrupt system allows interrupts to be individually masked. Using Intel's versatile 8259A interrupt controller, the interrupt system may be user programmed to respond to individual needs.

Input/Output

IPC Serial Channels — The I/O subsystem in the Model 225 consists of two parts: the IOC card and two serial channels on the IPC itself. Each serial channel is RS232 compatible and is capable of running asynchronously from 110 to 9600 baud or synchronously from 150 to 56K baud. Both may be connected to a user defined data set or terminal. One channel contains current loop adapters. Both channels are implemented using Intel's 8251A USART. They can be programmed to perform a variety of I/O functions. Baud rate selection is accomplished through an Intel 8253 interval timer. The 8253 also serves as a real-time clock for the entire system. I/O activity through both serial channels is signaled to the system through a second 8259A interrupt controller, operating in a polled mode nested to the primary 8259A.

IOC Interface — The remainder of system I/O activity takes place in the IOC. The IOC provides interface for the CRT, keyboard, and standard Intellec peripherals including printer, high speed paper tape reader/punch, and universal PROM programmer. The IOC contains its own independent microprocessor, an 8080A-2. The CPU controls all I/O operations as well as supervising communications with the IPC. 8K bytes of ROM contain all I/O control firmware. 8K bytes of RAM are used for CRT screen refresh storage. These do not occupy space in Intellec Series II main memory since the IOC is a totally independent microcomputer subsystem.

Integral CRT

Display — The CRT is a 12-inch raster scan type monitor with a 50/60 Hz vertical scan rate and 15.5kHz horizontal scan rate. Controls are provided for brightness and contrast adjustments. The interface to the CRT is provided through an Intel 8275 single-chip programmable CRT controller. The master processor on the IPC transfers a character for display to the IOC, where it is stored in RAM. The CRT controller reads a line at a time into its line buffer through an Intel 8257 DMA controller and then feeds one character at a time to the character generator to produce the video signal. Timing for the CRT control is provided by an Intel 8253 interval timer. The screen display is formatted as 25 rows of 80 characters. The full set of ASCII characters is displayed, including lower case alphas.

Keyboard — The keyboard interfaces directly to the IOC processor via an 8-bit data bus. The keyboard contains an Intel UPI-41™ Universal Peripheral Interface, which scans the keyboard, encodes the characters, and buffers the characters to provide N-key rollover. The keyboard itself is a high quality typewriter style keyboard containing the full ASCII character set. An upper/lower case switch allows the system to be used for document preparation. Cursor control keys are also provided.

Peripheral Interface

A UPI-41 Universal Peripheral Interface on the IOC board provides interface for other standard Intellec peripherals including a printer, high speed paper tape reader, high speed paper tape punch, and universal PROM programmer. Communication between the IPC and IOC is maintained over a separate 8-bit bidirectional data bus. Connectors for the four devices named above, as well as the two serial channels, are mounted directly on the IOC itself.

Control

User control is maintained through a front panel, consisting of a power switch and indicator, reset/boot switch, run/halt light, and eight interrupt switches and indicators. The front panel circuit board is attached directly to the IPC, allowing the eight interrupt switches to connect to the primary 8259A, as well as to the Intellec Series II bus.

Integral Floppy Disk Drive

The integral floppy disk is controlled by an Intel 8271 single chip, programmable floppy disk controller. It transfers data via an Intel 8257 DMA controller between an IOC RAM buffer and the diskette. The 8271 handles reading and writing of data, formatting diskettes, and reading status, all upon appropriate commands from the IOC microprocessor.

MULTIBUS™ Interface Capability

All Intellec Series II/85 models implement the industry standard MULTIBUS protocol. The MULTIBUS protocol enables several bus masters, such as CPU and DMA devices, to share the bus and memory by operating at different priority levels. Resolution of bus exchanges is synchronized by a bus clock signal derived independently from processor clocks. Read/write transfers may take place at rates up to 5 MHz. The bus structure is suitable for use with any Intel microcomputer family.

SPECIFICATIONS

Host Processor (IPC)

8085A-2 based, operating at 4.0 MHz.
RAM — 64K on the CPU card
ROM — 4K (2K in monitor, 2K in boot/diagnostic)
Bus — MULTIBUS™ bus, maximum transfer rate of 5 MHz
Clocks — Host processor, crystal controlled at 4.0 MHz, bus clock, crystal controlled at 9.8304 MHz

I/O Interfaces

Two Serial I/O Channels, RS232C, at 110-9600 baud (asynchronous) or 150-56K baud (synchronous). Baud rates and serial format fully programmable using Intel 8251A USARTs. Serial Channel 1 additionally provided with 20 mA current loop. Parallel I/O interfaces provided for paper tape punch, paper tape reader, printer, and UPP-103 Universal PROM Programmer.

Interrupts

8-level, maskable, nested priority interrupt network initiated from front panel or user selected devices.

Direct Memory Access (DMA)

Standard capability on MULTIBUS interface; implemented for user selected DMA devices through optional DMA module—maximum transfer rate of 5 MHz.

Memory Access Time

RAM — 470 ns max
PROM — 540 ns max

Integral Floppy Disk Drive

Floppy Disk System Capacity —
250K bytes (formatted)
Floppy Disk System Transfer Rate —
160K bits/sec
Floppy Disk System Access Time —
Track to Track: 10 ms max
Average Random Positioning: 260 ms
Rotational Speed: 360 rpm
Average Rotational Latency: 83 ms
Recording Mode: FM

Physical Characteristics

CHASSIS

Width — 17.37 in. (44.12 cm)
Height — 15.81 in. (40.16 cm)
Depth — 19.13 in. (48.59 cm)
Weight — 73 lb. (33 kg)

KEYBOARD

Width — 17.37 in. (44.12 cm)
Height — 3.0 in. (7.62 cm)
Depth — 9.0 in. (22.86 cm)
Weight — 6 lb. (3 kg)

Electrical Characteristics

DC POWER SUPPLY

Volts Supplied	Amps Supplied	Typical System Requirements
+ 5 ± 5%	30.0	17.0
+12 ± 5%	2.5	1.1
−12 ± 5%	0.3	0.1
−10 ± 5%	1.0	0.08
+15 ± 5%*	1.5	1.5
+24 ± 5%*	1.7	1.7

*Not available on bus.

AC REQUIREMENTS FOR MAINFRAME

110V, 60 Hz — 5.9 Amp
220V, 50 Hz — 3.0 Amp

Environmental Characteristics

Operating Temperature — 16°C to 32°C (61°F to 90°F)
Humidity — 20% to 80%

Equipment Supplied

Model 225 Chassis including:
- Integrated Processor Card (IPC)
- I/O Controller Board (IOC)
- CRT
- ROM-Resident System Monitor

Detachable keyboard
ISIS-II System Diskette with MCS-80/MCS-85 Macroassembler
ISIS-II CREDIT Diskette CRT-Based Text Editor

Documentation Supplied

A Guide to Microcomputer Development Systems, 9800558

Intellec® Series II Model 22X/23X Installation Manual, 9800559

ISIS-II System User's Guide, 9800306

Intellec® Series II Hardware Reference Manual, 9800556

8080/8085 Assembly Language Programming Manual, 9800301

ISIS-II 8080/8085 Assembler Operator's Manual, 9800292

Intellec® Series II Systems Monitor Source Listing, 9800605

Intellec® Series II Schematic Drawings, 9800554

ISIS-II CREDIT (CRT-Based Text Editor) User's Guide, 9800902

Additional manuals may be ordered from any Intel sales representative or distributor office, or from Intel Literature Department, 3065 Bowers Avenue, Santa Clara, California 95051.

ORDERING INFORMATION

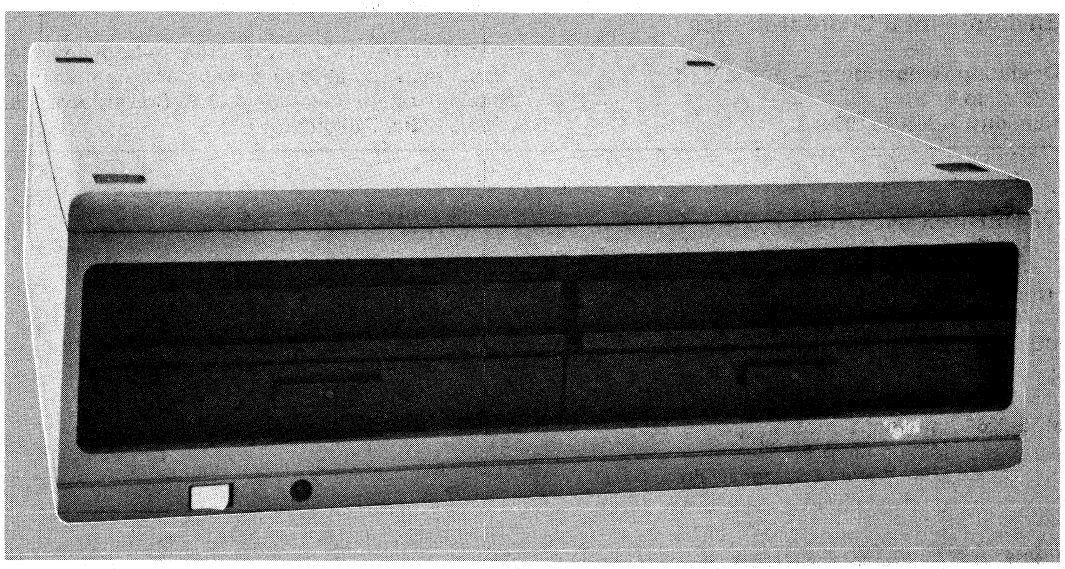
Part Number	Description
MDS-225*	Intellec® Series II/85 Model 225 Microcomputer Development System (110V/60 Hz)
MDS-226*	Intellec® Series II/85 Model 226 Microcomputer Development System (220V/50 Hz)

*"***MDS***"* is an ordering code only, and is not used as a product name or trademark. MDS® is a registered trademark of Mohawk Data Sciences Corp.

INTELLEC® SINGLE/DOUBLE DENSITY FLEXIBLE DISK SYSTEM

- **Flexible Disk System Providing High Speed Input/Output and Data Storage for Intellec® Microcomputer Development Systems**
- **Available in Both Single Density and Double Density Systems**
- **Data Recorded on Single Density Flexible Disk Is in IBM Soft-Sectored Format Which Allows ¼ Million Byte Data Capacity with Up to 200 Files Per Flexible Disk**
- **Data Recorded on Double Density Flexible Disk is in Soft-Sectored Format Which Allows ½ Million Byte Data Capacity with Up to 200 Files Per Flexible Disk**
- **Associated Software Supports Up to Four Double Density Drives and Two Single Density Drives, Providing Up to 2.5 Megabytes of Storage in One System**
- **Dynamic Allocation and Deallocation of Flexible Disk Sectors for Variable Length Files**

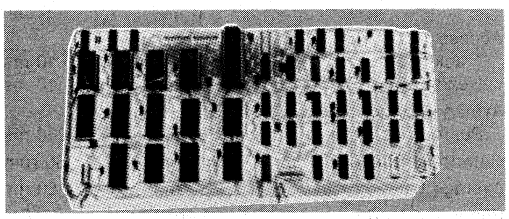
The Intellec Flexible Disk System is a sophisticated, general purpose, bulk storage peripheral for use with the Intellec Microcomputer Development System. The use of a flexible disk operating system significantly reduces program development time. The software system known as ISIS-11 (Intel System Implementation Supervisor), provides the ability to edit, assemble, compile, link, relocate, execute and debug programs, and performs all file management tasks for the user.



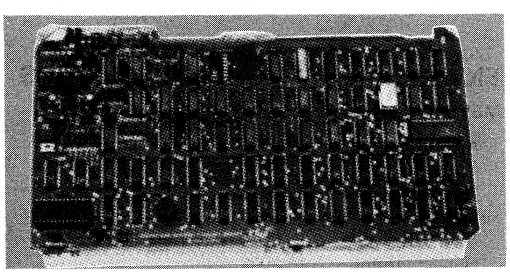
HARDWARE

The Intellec® flexible disk system provides direct access bulk storage, intelligent controller, and two flexible disk drives. Each single density drive provides ¼ million bytes of storage with a data transfer rate of 250,000 bits/second. The double density drive provides ½ million bytes of storage with a data transfer rate of 500,000 bits/second. The controllers are implemented with Intel's powerful Series 3000 Bipolar Microcomputer Set. The controllers provide interface to the Intellec System bus. Each single density controller will support two drives. Each double density controller will support up to four drives. The flexible disk system records all data in soft sector format.

The single/double density flexible disk controllers each consists of two boards, the Channel Board and the Interface Board. These two printed circuit boards reside in the Intellec System chassis. The boards are shown in the photograph, and are described in more detail in the following paragraphs.



SINGLE/DOUBLE DENSITY CHANNEL BOARD



DOUBLE DENSITY INTERFACE BOARD
(SINGLE DENSITY INTERFACE BOARD
IS SIMILAR TO THE ONE SHOWN ABOVE)

CHANNEL BOARD

The *Channel Board* is the primary control module within the flexible disk system. The Channel Board receives, decodes, and responds to channel commands from the Central Processor Unit (CPU) in the Intellec system. The Channel Board can access a block of Intellec system memory to determine the particular flexible disk operations to be performed and fetch the parameters required for the successful completion of the specified operation.

The control functions of the Channel Board have been achieved with an 8-bit microprogrammed processor, designed with Intel's Series 3000 Bipolar Microcomputer Set. This 8-bit processor includes four 3002 Central Processing Elements (2-bit slice per CPE), a 3001 Microprogram Control Unit, and 512 x 32 bits of 3604 programmable-read-only-memory (PROM) which stores the microprogram. It is the execution of the microprogram by the microcomputer set which actually effects the control capability of the Channel Board.

This board is the same for either single or double density drives, except that the Series 3000 microcode is different.

INTERFACE BOARD

The *Interface Board* provides the flexible disk controller with a means of communication with the flexible disk drives, as well as with the Intellec system bus. Under control of the microprogram being executed on the Channel Board, the Interface Board generates those signals which cause the read/write head on the selected drive to be loaded (i.e., to come in contact with the flexible disk platter), cause the head to move to the proper track and verify successful operation. The Interface Board accepts the data being read off the flexible disk, interprets synchronizing bit patterns, checks the validity of the data using a cyclic redundancy check (CRC) polynomial, and then transfers the data to the Channel Board.

During write operations, the Interface Board outputs the data and clock bits to the selected drive at the proper times, and generates the CRC characters which are then appended to the data.

When the flexible disk controller requires access to Intellec system memory, the Interface Board requests the DMA master control of the system bus, and generates the appropriate memory command. The Interface Board also acknowledges I/O commands as required by the Intellec bus.

The Flexible Disk System is capable of performing seven different operations: recalibrate, seek, format track, write data, write deleted data, read data, and verify CRC.

The channel board is different for single and double density drives, due to the different recording techniques used. The single density controller boards support one set of dual single density drives. The double density controller boards support up to two sets of dual double density drives (four drives total).

The double density controller may co-reside with the Intel single density controller to allow conversion of single density flexible disk to double density format, and provide up to 2.5M bytes of storage.

FLEXIBLE DISK DRIVE MODULES

Each flexible disk drive consists of read/write and control electronics, drive mechanisms, read/write head, track positioning mechanism, and the removable flexible disk platter. These components interact to perform the following functions:

- Interpret and generate control signals
- Move read/write head to selected track
- Read and write data

ASSOCIATED SOFTWARE — INTEL SYSTEMS IMPLEMENTATION SUPERVISOR (ISIS-II)

The Flexible Disk Drive System is to be used in conjunction with the ISIS-II Operating System. ISIS-II provides total file management capabilities, file editing, library management, run-time supports, and utility management.

ISIS-II provides automatic implementation of random access disk files. Up to 200 files may be stored on each ¼ million byte flexible disk for single density system or on each ½ million byte flexible disk for double density system. For more information, see the ISIS-II data specification sheet.

ISIS-II OPERATIONAL ENVIRONMENTAL

ISIS-II

32K bytes RAM memory
48K bytes when using Assembler Macro feature
64K bytes when using PL/M or Fortran
System Console
Single or Double density Flexible Disk Drive

HARDWARE SPECIFICATIONS

MEDIA

Single Density	Double Density
Flexible Disk	Double Density Specified Flexible Disk
One Recording Surface	One Recording Surface
IBM Soft Sector Format	Soft Sector Format
77 Tracks/Diskette	77 Tracks/Diskette
26 Sectors/Track	52 Sectors/Track
128 Bytes/Sector	128 Bytes/Sector

PHYSICAL CHARACTERISTICS

CHASSIS AND DRIVES

Mounting:	Table-Top
Height:	5.7 in. (14.5 cm)
Width:	17.6 in. (44.7 cm)
Depth:	19.4 in. (49.3 cm)
Weight:	43.0 lb. (19.5 kg)

ELECTRICAL CHARACTERISTICS

CHASSIS

DC Power Supplies
Supplied Internal to the Cabinet

AC Power Requirements
3-wire input with center conductor (earth ground) tied to chassis

Single-phase, 115 VAC; 60 Hz; 1.2 Amp Maximum (For a Typical Unit)
230 VAC; 50 Hz; 0.7 Amp Maximum (For a Typical Unit)

FLEXIBLE DISK OPERATING SYSTEM CONTROLLER

DC Power Requirements (All power supplied by Intellec Development System)

CHANNEL BOARD

Single Density	Double Density
5V @ 3.75A (typ), 5A (max)	5V @ 3.75A (typ), 5A (max)

INTERFACE BOARD

Single Density	Double Density
5V @ 1.5A (typ), 2.5A (max)	5V @ 1.5A (typ), 2.5A (max) – 10V @ 0.1A (typ), 0.2A (max)

FLEXIBLE DISK DRIVE PERFORMANCE SPECIFICATION

	Single Density	Double Density
Capacity (Unformatted):		
Per Disk	3.1 megabits	6.2 megabits
Per Track	41 kilobits	82 kilobits
Capacity (Formatted):		
Per Disk	2.05M bits	4.10 megabits
Per Track	26.6K bits	53.2 kilobits
Data Transfer Rate	250 kilobits/sec	500 kilobits/sec
Access Time:		
Track-to-Track	10 ms	10 ms
Head Settling Time	10 ms	10 ms
Average Random Positioning Time	260 ms	260 ms
Rotational Speed	360 rpm	360 rpm
Average Latency	83 ms	83 ms
Recording Mode	Frequency Modulation	M^2FM

ENVIRONMENTAL CHARACTERISTICS

MEDIA

Temperature:
Operating: 15.6°C to 51.7°C
Non-Operating: 5°C to 55°C
Humidity:
Operating: 8 to 80% (Wet bulb 29.4°C)
Non-Operating: 8 to 90%

DRIVES AND CHASSIS

Temperature:
Operating: 10°C to 38°C
Non-Operating: – 35°C to 65°C
Humidity:
Operating: 20% to 80% (Wet bulb 26.7°C)
Non-Operating: 5% to 95%

CONTROLLER BOARDS

Temperature:
Operating: 0 to 55°C
Non-Operating: – 55°C to 85°C
Humidity:
Operating: Up to 95% relative humidity without condensation
Non-Operating: All conditions without condensation of water or frost

EQUIPMENT SUPPLIED

SINGLE DENSITY	DOUBLE DENSITY
Cabinet, Power Supplies, Line Cord, Two Drives	Cabinet, Power Supplies, Line Cord, Two Drives
Single Density FDC Channel Board	Double Density FDC Channel Board
Single Density FDC Interface Board	Double Density FDC Interface Board
Dual Auxiliary Board Connector	Dual Auxiliary Board Connector
Flexible Disk Controller Cable	Flexible Disk Controller Cable
Flexible Disk Peripheral Cable	Flexible Disk Peripheral Cable
Hardware Reference Manual	Hardware Reference Manual
Reference Schematics	Reference Schematics
ISIS-II Single Density System Disk	ISIS-II Double Density System Disk
ISIS-II System User's Guide	ISIS-II System User's Guide

OPTIONAL EQUIPMENT

MDS-BLD*	10 Blank Flexible Disks
MDS-730/731*	Second Drive Cabinet with two additional drives

ORDERING INFORMATION

Part Number	Description
MDS-710/110V* 711/220V	Flexible Disk drive unit with two drives, single density drive controller, software, and cables.
MDS-720-110V* 721/220V	Flexible Disk drive unit with two drives, double density drive controller, software, and cables.
MDS-730/110V* 731/220V	Add-on drive unit with two drives and double density cable, without controller and software. Can be used with double density controller.

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

8051 SOFTWARE DEVELOPMENT PACKAGE

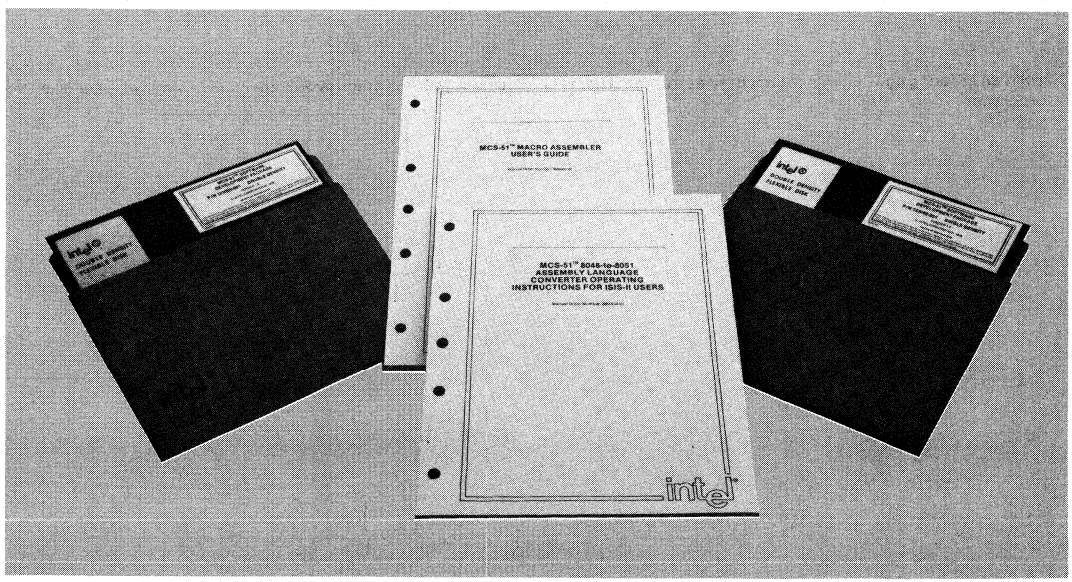
- **Symbolic assembly language programming for 8051 microcontrollers**
- **Extends Intellec® Microcomputer Development System to support 8051 program development**
- **Provides assembler output in standard Intel hex format**
- **Macro Assembler features conditional assembly and macro capabilities**
- **CONV51 Converter for translation of 8048 assembly language source code to 8051 assembly language source code**
- **Provides upward compatibility from the MCS-48™ family of single-chip microcontrollers**
- **Supports conversion of ASM48 source code macro definitions**

The 8051 software development package provides development system support for the powerful 8051 family of single chip microcomputers. The package contains a symbolic macro assembler and MCS-48 source code converter.

The assembler produces absolute machine code from 8051 macro assembly language instructions. This object code may be used to program the 8751 EPROM version of the chip. The assembler output may also be debugged using the ICE-51™ in-circuit emulator.

The converter translates 8048 assembly language instructions into 8051 source instructions to provide software compatibility between the two families of microcontrollers.

This diskette-based software package runs under ISIS-II on an Intellec Microcomputer Development System with 64K bytes of memory.



8051 MACRO ASSEMBLER

- **Supports 8051 family program development on Intellec® Microcomputer Development Systems**
- **Gives symbolic access to powerful 8051 hardware features**
- **Produces object file, listing file and error diagnostics**
- **Provides software support for many addressing and data allocation capabilities**
- **Symbolic Assembler supports symbol table, cross-reference, macro capabilities, and conditional assembly**

The 8051 Macro Assembler (ASM51) translates symbolic 8051 macro assembly language instructions into machine executable object code. These assembly language mnemonics are easier to program and are more readable than binary or hexadecimal machine instructions. Also, by allowing the programmer to give symbolic names to memory locations rather than absolute addresses, software design and debug are performed more quickly and reliably.

The assembler supports macro definitions and calls. This is a convenient way to program a frequently used code sequence only once. The assembler also provides conditional assembly capabilities.

Cross referencing is provided in the symbol table listing, showing the user the lines in which each symbol was defined and referenced.

ASM51 provides symbolic access to the many useful addressing features of the 8051 architecture. These features include referencing for bit and byte locations, and for providing 4-bit operations for BCD arithmetic. The assembler also provides symbolic access to hardware registers, I/O ports, control bits, and RAM addresses.

Math routines are enhanced by the MULtiply and DIVide instructions.

If an 8051 program contains errors, the assembler provides a comprehensive set of error diagnostics, which are included in the assembly listing or on another file. Program testing may be performed by using the Universal PROM Programmer and 8751 personality card to program the 8751 EPROM version of the chip, or by using the ICE-51 in-circuit emulator.

```
MCS-51 MACRO ASSEMBLER                                                    PAGE

ISIS-II MCS-51 MACRO ASSEMBLER
OBJECT MODULE PLACED IN :F1:CONVRT.HEX
ASSEMBLER INVOKED BY:  :F1:ASM51 :F1:CONVRT.51 SYMBOLS XREF

LOC  OBJ             LINE     SOURCE

                        1
                        2
                        3
                        4     ; This routine converts BCD to binary and binary to BCD
                        5     ;
                        6     ;
                        7     ;
                        8
0100                    9             ORG 100H
0100 E7                10     CONVRT: MOV A,@R1
0101 75F00A            11             MOV B,#10     ; Load B for division or multiplication by 10
0104 4008              12             JC BINBCD
0106 54F0              13     BCDBIN: ANL A,#0F0H     ; Mask out low order digit
0108 C4                14             SWAP A         ; Move high digit into low order nibble of accumulator
                       15     ; Multiply digit by 10
0109 A4                16             MUL AB         ; Multiply digit by 10
010A C7                17             XCH A,@R1      ; Store intermediate result and get new copy of BCD
010B 540F              18             ANL A,#0FH     ; Mask out high order digit
010D 27                19             ADD A,@R1      ; Add high order digit and low order digit
010E F7                20             MOV @R1,A      ; Store result
010F 22                21             RET
0110 B4                22     BINBCD: DIV AB         ; Divide by 10
0111 C4                23             SWAP A         ; Put quotient in high order nibble
0112 45F0              24             ORL A,B     ; Place remainder in low order nibble
0114 F7                25             MOV @R1,A      ; Store result
0115 22                26             RET
                       27             END

MCS-51 MACRO ASSEMBLER                                                    PAGE    2

XREF SYMBOL TABLE LISTING
---- ------ ----- -------

NAME    TYPE     VALUE AND REFERENCES

B . . . . N DSEG   00F0H  11 24
BCDBIN. . L CSEG   0106H  13#
BINBCD. . L CSEG   0110H  12 22#
CONVRT. . L CSEG   0100H  10#

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

Sample ASM51 Listing

CONV51
8048 TO 8051 ASSEMBLY LANGUAGE CONVERTER UTILITY PROGRAM

- **Enables software written for the MCS-48™ family to be upgraded to run on the 8051**
- **Maps each 8048 instruction to a corresponding 8051 instruction**
- **Preserves comments; translates 8048 macro definitions and calls**
- **Provides diagnostic information and warning messages embedded in the output listing**

The 8048 to 8051 Assembly Language Converter is a utility to help users of the MCS-48 family of microcomputers upgrade their designs with the high performance 8051 architecture. By converting 8048 source code to 8051 source code, the software investment developed for the 8048 is maintained when the system is upgraded.

The goal of the converter (CONV51) is to attain functional equivalence with the 8048 code by mapping each 8048 instruction to a corresponding 8051 instruction. In some cases a different instruction is produced because of the enhanced instruction set (e.g., bit CLR instead of ANL).

Although CONV51 tries to attain functional equivalence with each instruction, certain 8048 code sequences cannot be automatically converted. For example, a delay routine which depends on 8048 execution speed would require manual adjustment. A few instructions, in fact, have no 8051 equivalent (such as those involving P4-P7). Finally, there are a few areas of possible intervention such as PSW manipulation and interrupt processing, which at least require the user to confirm proper translation. The converter always warns the user when it cannot guarantee complete conversion.

CONV51 produces two files. The output file contains the ASM51 source program produced from the 8048 instructions. The listing file produces correlated listings of the input and output files, with warning messages in the output file to point out areas that may require users' intervention in the conversion.

SPECIFICATIONS

OPERATING ENVIRONMENT

Required Hardware:

Intellec Microcomputer Development System with

64K Bytes of RAM

Flexible Disk Drive(s)

System Console

—CRT or hard copy device

Optional Hardware:

Universal PROM Programmer
Line Printer
ICE-51 In-Circuit Emulator

Required Software:

ISIS-II Diskette Operating System (V3.4 or later)

Documentation Package:

MCS-51 Macro Assembler User's Guide

MCS-51 Macro Assembly Language Pocket Reference

MCS-51 8048-to-8051 Assembly Language Converter Operating Instructions for ISIS-II Users

ORDERING INFORMATION

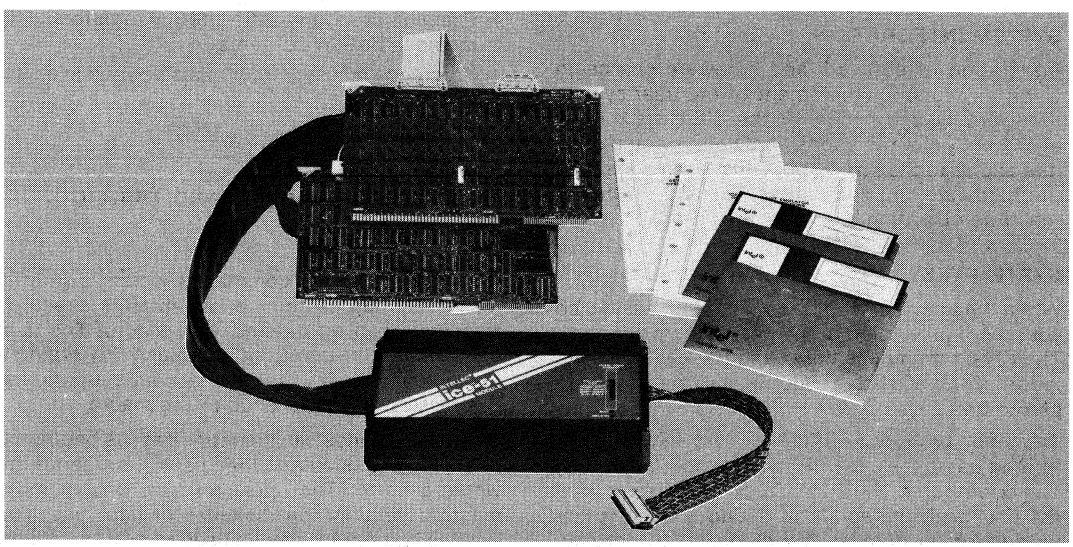
Part Number	Description
MCI-51-ASM	8051 Software Development Package

PRELIMINARY

ICE-51™
8051 IN-CIRCUIT EMULATOR

- **Precise, full-speed, real-time emulation**
  - **— Load, drive, timing characteristics**
  - **— Full-speed program RAM**
  - **— Serial and parallel ports**
- **User-specified breakpoints**
- **Execution trace**
  - **— User-specified qualifier registers**
  - **— Conditional trigger**
  - **— Symbolic groupings and display**
  - **— Instruction and frame modes**
- **Emulation timer**
- **Full symbolic debugging**
- **Single-line assembly and disassembly for program instruction changes**
- **Macro commands and conditional block constructs for automated debugging sessions**
- **HELP facility: ICE-51 command syntax reference at the console**
- **User confidence test of ICE-51 hardware**

The ICE-51 module resides in the Intellec® Microcomputer Development System and interfaces to any user-designed 8051 system through a cable terminating in an 8051 emulator microprocessor and a pin-compatible plug. The emulator processor, together with 8K bytes of user program RAM located in the ICE-51 buffer box, replaces the 8051 device in the user system while maintaining the 8051 electrical and timing characteristics. Powerful Intellec debugging functions are thus extended into the user system. Using the ICE-51 module, the designer can emulate the system's 8051 in real-time or single-step mode. Breakpoints allow the user to stop emulation on user-specified conditions, and a trace qualifier feature allows the conditional collection of 1000 frames of trace data. Using the single-line 8051 assembler the user may alter program memory using ASM51 mnemonics and symbolic references, without leaving the emulator environment. Frequently used command sequences can be combined into compound commands and identified as macros with user-defined names.



FUNCTIONAL DESCRIPTION

Integrated Hardware and Software Development

The ICE-51 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-51 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-51 emulator assists four stages of development:

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-51 debugging capabilities can be used in conjunction with the Intellec text editor and 8051 macroassembler to facilitate program development.

HARDWARE DEVELOPMENT

The ICE-51 module's precise emulation characteristics and full-speed program RAM make it a valuable tool for debugging hardware, including time-critical serial port, parallel port, and timer interfaces.

SYSTEM INTEGRATION

Integration of software and hardware can begin when any functional element of the user system hardware is connected to the 8051 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.

SYSTEM TEST

When the user's prototype is complete, it is tested with the final version of the user system software. The ICE-51 module is then used for real-time emulation of the 8051 to debug the system as a completed unit.

The final product verification test may be performed using the 8751 EPROM version of the 8051 microcomputer. Thus, the ICE-51 module provides the user with the ability to debug a prototype or production system at any stage in its development without introducing extraneous hardware or software test tools.

Symbolic Debugging

The ICE-51 emulator permits the user to define and use symbolic, rather than absolute, references to program and data memory addresses; additional symbols are predefined by the ICE-51 software for referencing registers, flags, and input/output ports. Thus, the user need not recall or look up the addresses of key locations in his program as they change with each assembly, or become involved with machine code.

When a symbol is used for memory reference in an ICE-51 emulator command, the emulator supplies the corresponding location as stored in the ICE-51 emulator symbol table. This table can be loaded with the symbol table produced by the assembler during application program assembly. The user can obtain the symbol table during software preparation simply by using the "DEBUG" switch in the ASM51 macroassembler. Furthermore, the user can interactively modify the emulator symbol table by adding new symbols or changing or deleting old ones. This feature provides great flexibility in debugging and minimizes the need to work with hexadecimal values.

Through symbolic references in combination with other features of the emulator, the user can easily:

- Interpret the results of emulation activity collected during trace.
- Disassemble program memory to mnemonics, or assemble mnemonic instructions to executable code.
- Examine or modify 8051 internal registers, data memory, or port contents.
- Reference labels or addresses defined in a user program.

Automated Debugging and Testing

MACRO COMMAND

A macro is a set of commands which is given a name. A group of commands which is executed frequently can be defined as a macro. The user can execute the group of commands by typing a colon followed by the macro name. Up to ten parameters may be passed to the macro.

Macro commands can be defined at the beginning of a debug session and then used throughout the whole session. The user can save one or more macro definitions on diskette for later use. The Intellec text editor may be used to edit the macro file. The macro definitions are easy to include in any later emulation session.

The power of the development system can be applied to manufacturing testing as well as development by writing test sequences as macros. The macros are stored on diskettes for use during system test.

COMPOUND COMMAND

Compound commands provide conditional execution of commands (IF command) and execution of commands repeatedly until certain conditions are met (COUNT, REPEAT commands).

Compound commands may be nested any number of times, and may be used in macro commands.

Example:

```
*DEFINE .I = 0          ; Define symbol .I to 0
*COUNT 100H            ; Repeat the following
                          commands 100H times.
.*IF .I AND 1 THEN     ; Check if .I is odd
..*BYTE .I = .I        ; Fill the memory at location .I
                          to value .I
..*END
.*.I = .I + 1          ; Increment .I by 1.
.*END                  ; Command executes upon
                          carriage-return after END
```

(The characters *, .*, and ..* shown in this example are system prompts which include an indication of the nesting level of compound commands.)

Operating Modes

The ICE-51 software is an Intellec RAM-based program that provides the user with easy-to-use commands for initiating emulation, defining breakpoints, controlling trace data collection, and displaying and controlling system parameters. ICE-51 commands are configured with a broad range of modifiers which provide the user with maximum flexibility in describing the operation to be performed.

EMULATION

The ICE-51 module can emulate the operation of a prototype 8051 system, at real-time speed (1.2 to 12 MHz) or in single steps. Emulation commands to the ICE-51 module control the process of setting up, running, and halting an emulation of the user's 8051-based system. Breakpoints and tracepoints enable the ICE-51 emulator to halt emulation and provide a detailed trace of execution in any part of the user's program. A summary of the emulation commands is shown in Table 1.

Breakpoints

The ICE-51 hardware includes two breakpoint registers that allow the user to halt emulation when specified conditions are met. The emulator continuously compares the values stored in the breakpoint registers with the status of specified address, opcode, operand, or port values, and halts emulation when this comparison is satisfied. When an instruction initiates a break, that instruction is executed completely before the break takes place. The ICE-51 emulator then regains control of the console and enters the Interrogation Mode. With the breakpoint feature, the user can request an emulation break when his program:

- Executes an instruction at a specific address or within a range of addresses.
- Executes a particular opcode.
- Receives a specific signal on a port pin.
- Fetches a particular operand from the user program memory.
- Fetches an operand from a specific address in program memory.

Table 1. Major Emulation Commands

Command	Description
GO	Begins real-time emulation and optionally specifies break conditions.
BR0, BR1, BR	Sets or displays either or both Breakpoint Registers used for stopping real-time emulation.
STEP	Performs single-step emulation.
QR0, QR1	Specifies match conditions for qualified trace.
TR	Specifies or displays trace-data collection conditions and optionally sets Qualifier Register (QR0, QR1).
Synchronization Line Commands	Set and display status of synchronization line outputs or latched inputs. Used to allow real-time emulation or trace to start and stop synchronously with external events.

Trace and Tracepoints

Tracing is used with both real-time and single-step emulation to record diagnostic information in the trace buffer as a program is executed. The information collected includes opcodes executed, port values, and memory addresses. The ICE—51 emulator collects 1000 frames of trace data.

This information can be displayed as assembler instruction mnemonics, if desired, for analysis during interrogation or single-step mode. The trace-collection facility may be set to run conditionally or unconditionally. Two unique trace qualifier registers, specified in the same way as break-

point registers, govern conditional trace activity. The qualifiers can be used to condition trace data collection to take place as follows:

- Under all conditions (forever).
- Only while the trace qualifier is satisfied.
- For the frames or instructions preceding the time when a trace qualifier is first satisfied (pre-trigger trace).
- For the frames or instructions after a trace qualifier is first satisfied (post-triggered trace).

Table 2 shows an example of a trace display.

Table 2. Trace Display (Instruction Mode)

```
*D ALL
FRAME  LOC    OBJ    INSTRUCTION                  P1   P2  P3   TOVF
0006:  0000H  FF     MOVX  @DPTR,A               00H  FFH FFH  0
0007:  0001H  F8     MOV   R0,A                   00H  FFH FFH  0
0011:  0002H  F4     CPL   A                      00H  FFH FFH  0
0015:  0003H  23     RLC   A                      00H  FFH FFH  0
0019:  0004H  75E0AA MOV   ACC,#0AAH              00H  FFH FFH  0
0027:  0007H  C3     CLR   C                      00H  FFH FFH  0
0031:  0008H  98     SUBB  A,R0                   AAH  FFH FFH  0
0035:  0009H  0000   NOP   0000                   AAH  FFH FFH  0
0043:  000BH  753A0F MOV   3AH,#.FIVE             AAH  FFH FFH  0
0051:  000EH  0156   AJMP  0056H                  AAH  FFH FFH  0
```

INTERROGATION AND UTILITY

Interrogation and utility commands give the user convenient access to detailed information about the user program and the state of the 8051 that is useful in debugging hardware and software. Changes can be made in memory and in the 8051 registers, flags, and port values. Commands are also provided for various utility operations such as loading and saving program files, defining symbols, displaying trace data, controlling system synchronization and returning control to ISIS-II. A summary of the basic interrogation and utility commands is shown in Table 3. Two time-saving emulator features are discussed below.

SINGLE-LINE ASSEMBLER/DISASSEMBLER — The single-line assembler/disassembler (ASM and DASM commands) permits the designer to examine and alter program memory using assembly language mnemonics, without leaving the emulator environment or requiring time-consuming program reassembly. When assembling new mnemonic instructions into program memory, previously defined symbolic references (from the original program assembly, or subsequently defined during the emulation session) may be used in the instruction operand field. The emulator will supply the absolute address or data values as stored in the emulator symbol table. These features eliminate user time spent translating to and from machine code and searching for absolute addresses, with a corresponding reduction in transcription errors.

Table 3. Major Interrogation and Utility Commands

Command	Description
HELP	Displays help messages for ICE-51 emulator command-entry assistance.
LOAD	Loads user object program (8051 code) into user program memory, and user symbols into ICE-51 emulator symbol table.
SAVE	Saves ICE-51 emulator symbol table and/or user object program in ISIS-II hexadecimal file.
LIST	Copies all emulator console input and output to ISIS-II file.
EXIT	Terminates ICE-51 emulator operation.
DEFINE	Defines ICE-51 emulator symbol or macro.
REMOVE	Removes ICE-51 emulator symbol or macro.
ASM	Assembles mnemonic instructions into user program memory.
DASM	Disassembles and displays user program memory contents.
Change/Display Commands	Change or display value of symbolic reference in ICE-51 emulator symbol table, contents of key-word references (including registers, I/O ports, and status flags), or memory references.
EVALUATE	Evaluates expression and displays resulting value.
MACRO	Displays ICE-51 macro or macros.
INTERRUPT	Displays serial, external, or timer interrupt register settings.
SECONDS	Displays contents of emulation timer, in microseconds.
Trace Commands	Position trace buffer pointer and select format for trace display.
PRINT	Displays trace data pointed to by trace buffer pointer.

HELP — The HELP file allows the user to display ICE-51 command syntax information at the Intellec console. By typing "HELP", a listing of all items for which help messages are available is displayed; typing "HELP <Item>" then displays relevant information about the item requested, including typical usage examples. Table 4 shows some sample HELP messages.

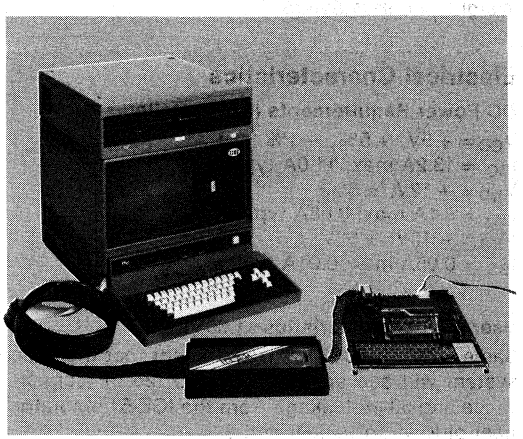


Figure 1. A Typical 8051 Development Configuration. The host system is an Intellec Model 225, plus 1 megabyte dual double-density flexible disk storage. The ICE-51 module is connected to an SDK-51 system design kit.

Emulation Accuracy

The speed and interface demands of a high-performance single-chip microcomputer require extremely accurate emulation, including full-speed, real-time operation with the full function of the microcomputer. The ICE-51 emulator achieves accurate emulation with an 8051 bond-out chip, a special configuration of the 8051 microcomputer family, as its emulation processor.

Each of the 40 pins on the user plug is connected directly to the corresponding 8051 pin on the bond-out chip. Thus the user system sees the emulator as an 8051 microcomputer at the 8051 socket. The resulting characteristics provide extremely accurate emulation of the 8051, including speed, timing characteristics, load and drive values, and crystal operation. The emulator may draw more power from the user system than a standard 8051 family device.

Additional bond-out pins provide signals such as internal address, data, clock, and control lines to the emulator buffer box. These signals let static RAM in the buffer box substitute for on-chip program ROM or EPROM or external program memory. The 8K bytes of full-speed RAM in the buffer box can be mapped in 4K blocks to anywhere within the 64K program memory space of the 8051. The bond-out chip also gives the emulator "backdoor" access to internal chip operation, so that the emulator can break and trace execution without interfering with the values on the user-system pins.

Table 4. HELP Command

```
*HELP
Help is available for the following items. Type HELP followed by
the item name. The help items cannot be abbreviated. (For more
information, type HELP HELP or HELP INFO.)
Emulation:    Trace Collection:  Misc:       <address>
GO CP  SYP    TR CP (BP CP) SY)  BASE        <CPU$keyword>
BR BRP BRI                       DISABLE     <expr>
STEP                             ENABLE      <ICE51$keyword>
              Trace Display:     ERROR       <identifier>
              TRACE MOVE PRINT   EVALUATE    <instruction>
              OLDEST NEWEST      HELP        <masked$constant>
Change/Display/Define/Remove:    INFO        <match$cond>
<CHANGE>  REMOVE CBYTE BBIT      <LIGHTS>    <numeric$constant>
<DISPLAY> SYMBOL IBYTE IASM      LIST        <partition>
REGISTER  RESET  PBYTE ASM       LOAD        <string>
SECONDS   WRITE  RBYTE MAP       SAVE        <string$constant>
DEFINE    STACK  XBYTE SY        SUFFIX      <symbolic$ref>
                                 SYMBOLIC    <system$symbols>
Macro:                Compound               <trace$reference>
DEFINE  DIR           Commands:              <unlimited$match$cond>
DISABLE ENABLE        COUNT                  <users$symbols>
INCLUDE PUT           IF
<MACRO$DISPLAY>       REPEAT
<MACRO$INVOCATION>
*
```

```
*HELP IF
IF - The conditional command allows conditional execution of one
or more commands based on the values of boolean conditions.
     IF <expr> [THEN] <cr>        <true$list>::=[<command> <cr>]@
      <true$list>                 <false$list>::=[<command> <cr>]@
      [ORIF <expr> <cr>           <command>::=An ICE-51 command.
      <true$list>]@
      [ELSE <cr>
      <false$list>]
     END
The <expr>s are evaluated in order as 16-bit unsigned integers.
If one is reached whose value has low-order bit 1 (TRUE), all
commands in the <true$list> following that <expr> are then
executed and all commands in the other <true$list>s and in the
<false$list> are skipped.  If all <expr>s have value with low-
order bit 0 (FALSE), then all commands in all <true$list>s are
skipped and, if ELSE is present, all commands in the <false$list>
are executed.
     (EX:  IF .LOOP=5 THEN
             STEP
             ELSE
             GO
             END)
```

SPECIFICATIONS

ICE-51 Operating Requirements

Intellec® Microcomputer Development System (64K RAM required)

System console

Intellec® Diskette Operating System (single or double density) ISIS-II v. 3.4 or later

Equipment Supplied

- Printed circuit boards (2)
- Emulation buffer box, Intellec interface cables, and user-interface cable with 8051 emulation processor
- Crystal power accessory
- Operating instructions manual
- Diskette-based ICE-51 software (single and double density)

Emulation Clock

User's system clock (1.2 to 12 MHz) or ICE-51 crystal power accessory (12 MHz)

Environmental Characteristics

Operating Temperature: 0° to 40°C
Operating Humidity: Up to 95% relative humidity without condensation.

Physical Characteristics

Printed Circuit Boards
Width: 12.00 in. (30.48 cm)
Height: 6.75 in. (17.15 cm)
Depth: 0.50 in. (1.27 cm)

Buffer Box
Width: 8.00 in. (20.32 cm)
Length: 12.00 in. (30.48 cm)
Depth: 1.75 in. (4.44 cm)
Weight: 4.0 lb (1.81 kg)

Electrical Characteristics

DC Power Requirements (from Intellec system)

$V_{CC} = +5V, +5\%, -1\%$
$I_{CC} = 13.2A$ max; 11.0A typical
$V_{DD} = +12V, \pm 5\%$
$I_{DD} = 0.1A$ max; 0.05A typical
$V_{BB} = -10V, \pm 5\%$
$I_{BB} = 0.05A$ max; 0.01A typical

User plug characteristics at 8051 socket

Same as 8031, 8051, or 8751, except that the user system will see an added load of 25 pF capacitance and 50 μA leakage from the ICE-51 emulator user plug at ports 0, 1, and 2.

ORDERING INFORMATION

Part Number	Description
MCI-51-ICE	8051 Microcontroller In-Circuit Emulator, cable assembly and interactive diskette software

UPP-103*
UNIVERSAL PROM PROGRAMMER

Replaces UPP-101, UPP-102 Universal PROM Programmers

Intellec development system peripheral for PROM programming and verification

Provides personality cards for programming all Intel PROM families

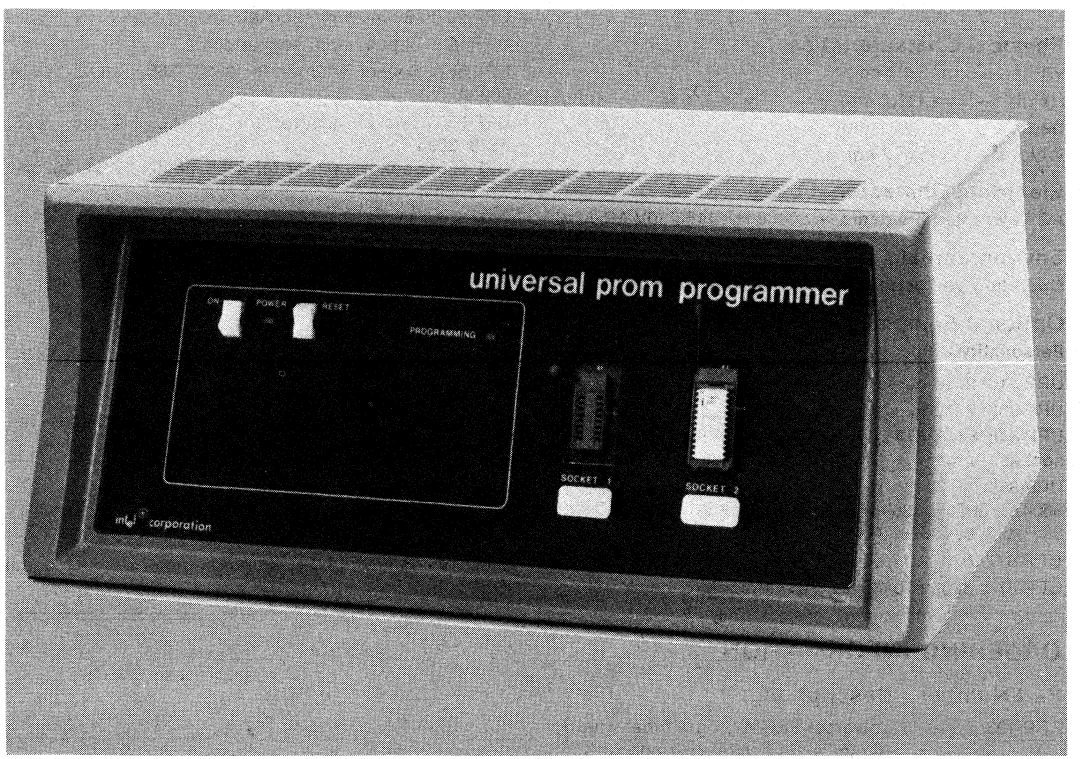
Provides zero insertion force sockets for both 16-pin and 24-pin PROMs

Universal PROM mapper software provides powerful data manipulation and programming commands

Provides flexible power source for system logic and programming pulse generation

Holds two personality cards to facilitate programming operations using several PROM types

The UPP-103 Universal PROM Programmer is an Intellec system peripheral capable of programming and verifying all of the Intel programmable ROMs (PROMs). In addition, the UPP-103 programs the PROM memory portions of the 8748 microcomputer, 8741 UPI, the 8755 PROM and I/O chip and the 2920 signal processor. Programming and verification operations are initiated from the Intellec development system console and are controlled by the universal PROM mapper (UPM) program.



FUNCTIONAL DESCRIPTION

Universal PROM Programmer

The basic Universal PROM Programmer (UPP) consists of a controller module, two personality card sockets, a front panel, power supplies, a chassis, and an Intellec development system interconnection cable. An Intel 4040-based intelligent controller monitors the commands from the Intellec System and controls the data transfer interface between the selected PROM personality card and the Intellec memory. A unique personality card contains the appropriate pulse generation functions for each Intel PROM family. Programming and verifying any Intel PROM may be accomplished by selecting and plugging in the appropriate personality card. The front panel contains a power-on switch and indicator, a reset switch, and two zero-force insertion sockets (one 16-pin and one 24-pin or two 24-pin). A central power supply provides power for system logic and for PROM programming pulse generation. The Universal PROM Programmer may be used as a table top unit or mounted in a standard 19-inch RETMA cabinet.

Universal PROM Mapper

The Universal PROM Mapper (UPM) is the software program used to control data transfer between paper tape or diskette files and a PROM plugged into the Universal PROM Programmer. It uses Intellec system memory for intermediate storage. The UPM transfers data in 8-bit HEX, BNPF, or binary object format between paper tape or diskette files and the Intellec system memory. While the data is in Intellec system memory, it can be displayed and changed. In addition, word length, bit position, and data sense can be adjusted as required for the PROM to be programmed. PROMs may also be duplicated or altered by copying the PROM contents into the Intellec system memory. Easy to use program and compare commands give the user complete control over programming and verification operations. The UPM eliminates the need for a variety of personalized PROM programming routines because it contains the programming algorithms for all Intel PROM families. The UPM (diskette based version) is included with the Universal PROM Programmer.

SPECIFICATIONS

Hardware Interface

Data — Two 8-bit unidirectional buses

Commands — 3 write commands, 2 read commands, one initiate command

Physical Characteristics

Width — 6 in. (14.7 cm)

Height — 7 in. (17.2 cm)

Depth — 17 in. (41.7 cm)

Weight — 18 lb (8.2 kg)

Electrical Characteristics

AC Power Requirements — 50-60 Hz; 115/230V AC: 80W

Environmental Characteristics

Operating Temperature — 0°C to 55°C

Optional Equipment

Personality Cards

UPP-816: 2716 personality card

UPP-832: 2732 personality card

UPP-848: 8748, 8741 personality card with 40-pin adaptor socket

UPP-865: 3602, 3622, 3602A, 3622A, 3621, 3604, 3624, 3604A, 3624A, 3604AL, 36046-6, 3605, 3605A, 3625, 3625A, 3608, 3628, 3636

UPP-872: 8702A/1702A personality card

UPP-878: 8708/8704/2708/2704 personality card

UPP-955: 8755A personality card with 40-pin adaptor socket

PROM Programming Sockets

UPP-501: 16-pin/24-pin socket pair

UPP-502: 24-pin/24-pin socket pair

UPP-562: Socket adaptor for 3621, 3602, 3622, 3602A, 3622A

UPP-555: Socket adaptor for 3604AL, 36046-6, 3608, 3628, 3636

UPP-566: Socket adaptor for 3605, 3625, 3605A, 3625A

Equipment Supplied

Cabinet

Power supplies

4040 intelligent controller module

Specified zero insertion force socket pair

Intellec development system interface cable

Universal PROM Mapper program (diskette-based version)

Reference Manuals

9800819 — Universal PROM Programmer User's Manual (SUPPLIED)

ORDERING INFORMATION

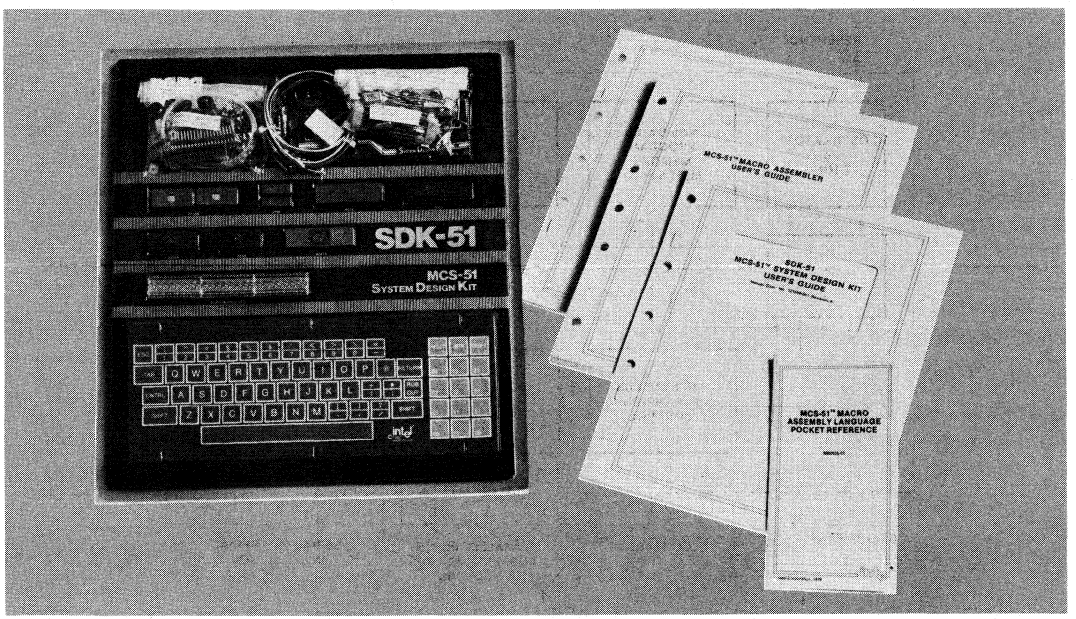
Part Number	Description
UPP-103	Universal PROM programmer with 16-pin/24-pin socket pair and 24-pin/24-pin socket pair.

PRELIMINARY

SDK-51
MCS-51 SYSTEM DESIGN KIT

- **Complete single-board microcomputer kit:**
  - **— Intel 8031 CPU**
  - **— ASCII keyboard and 24-character alpha-numeric display**
  - **— Wire-wrap area for custom circuitry**
  - **—User-configurable RAM**
  - **— Serial and parallel interfaces**
- **Extensive system software in ROM:**
  - **— Single-line assembler and disassembler**
  - **— System debugging commands**
    - **Go**
    - **Step**
    - **Breakpoints**
- **Interface software:**
  - **— Serial port**
  - **— Audio cassette**
  - **— Intellec® system**
- **User's guide, assembly manual, and MCS-51 design manuals**

The SDK-51 MCS-51 System Design Kit contains all of the components required to assemble a complete single-board microcomputer based on Intel's high-performance 8051 single-chip microcomputer. SDK-51 uses the external ROM version of the 8051 (8031). Once you have assembled the kit and supplied + 5V power, you can enter programs in MCS-51 assembly language mnemonics, translate them into MCS-51 object code, and run them under control of the system monitor. The kit supports optional memory and interface configurations, including a serial terminal link, audio cassette storage, EPROM program memory, and Intellec® development system upload and download capability.



FUNCTIONAL DESCRIPTION

The SDK-51 is a kit which includes hardware and software components to assemble a complete MCS-51 family single-board microcomputer. Only common laboratory tools and test equipment are required to assemble the kit. Assembly generally requires 5 to 10 hours, depending on the experience of the user.

The MCS-51 Microcomputer Series

MCS-51 is a series of high-performance single-chip microcomputers for use in sophisticated real-time applications such as instrumentation, industrial control and intelligent computer peripherals. The 8031, 8051, and 8751 microcomputers belong to the 8051 family, which is the first family in the MCS-51 series.

In addition to their advanced features for control applications, MCS-51 family devices have a microprocessor bus and arithmetic capability such as hardware multipy and divide instructions, which make the SDK-51 a versatile stand-alone microcomputer board.

The 8031, 8051, and 8751 CPUs

The 8031, 8051, and 8751 CPUs each combine, on a single chip, a 128 x 8 data RAM; 32 input/output lines; two 16-bit timer/event counters; a five-source, two-level nested interrupt structure; a serial I/O port; and on-chip oscillator and clock circuits. An 8051 block diagram is shown in Figure 1.

The 8031, the SDK-51's CPU, is a CPU without on-chip program memory. The 8031 can address 64K bytes of external program memory in addition to 64K bytes of external data memory. For systems requiring extra capability, each member of the 8051 family can be expanded using standard memories and the byte-oriented MCS-80 and MCS-85 peripherals. The 8051 is an 8031 with the lower 4K bytes of program memory filled with on-chip mask-programmable ROM while the 8751 has 4K bytes of ultraviolet light-erasable, electrically programmable ROM (EPROM).

The 8031 CPU operates at a 12 MHz clock rate, resulting in 4 μs multiply and divide and other instructions of 1 μs and 2 μs.

For additional information on the 8051 family, see the 8051 User's Manual or MCS-51 Macroassembler User's Guide.

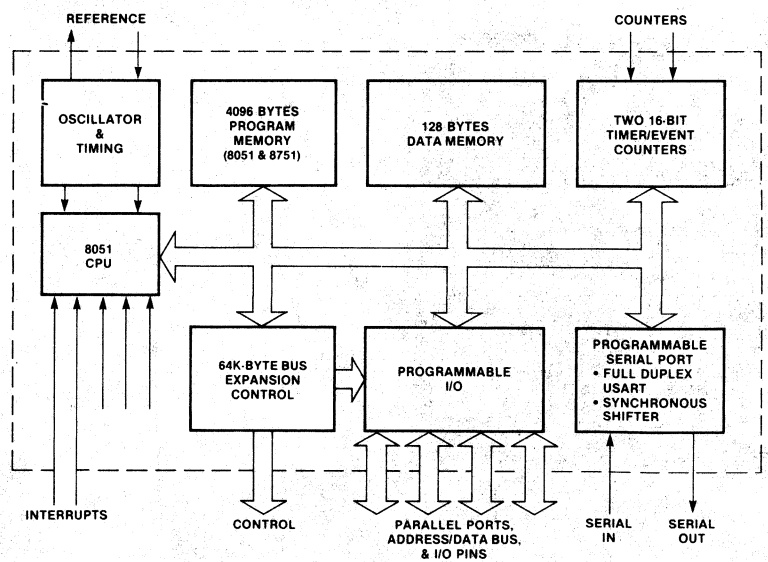


Figure 1. 8051 Block Diagram

System Software

A compact but powerful system monitor is contained in 8K bytes of pre-programmed ROM. The monitor includes system utilities such as command interpretation, user program debugging, and interface controls. Table 1 summarizes the SDK-51 monitor commands.

The ROM devices also include a single-line assembler and disassembler. The assembler lets you enter programs in MCS-51 assembly language mnemonics directly from the ASCII keyboard. The disassembler supports debugging by letting you look at MCS-51 instructions in mnemonic form during system interrogation.

Memory

The two 64K external memory spaces are combined into a single memory space which you can configure between program memory and data memory. The kit includes 1K-byte of static RAM. The board has space and printed circuitry for an additional 15K bytes of RAM and 8K bytes of ROM.

User Interface

The kit includes a typewriter-format, ASCII-subset keyboard and a 24-character, alpha-numeric LED

Table 1. SDK-51 Commands

Command	Operation
Set breakpoint	Define addresses for breaking execution.
Display cause	Ask the system why execution stopped.
Upload, download	Transfer files to and from Intellec® development system.
Save, load	Transfer files to and from optional cassette interface.
Set top of program memory	Define partition between program memory and data memory.
Set baud	Define baud rate value of serial port.
Display memory	Examine and change program memory or data locations.
Assemble	Translate an MCS-51 assembly mnemonic into object code.
Disassemble	Translate program memory into MCS-51 assembly language mnemonics.
Go	Start execution between a selected pair of addresses.
Step	Execute a specified number of instructions.

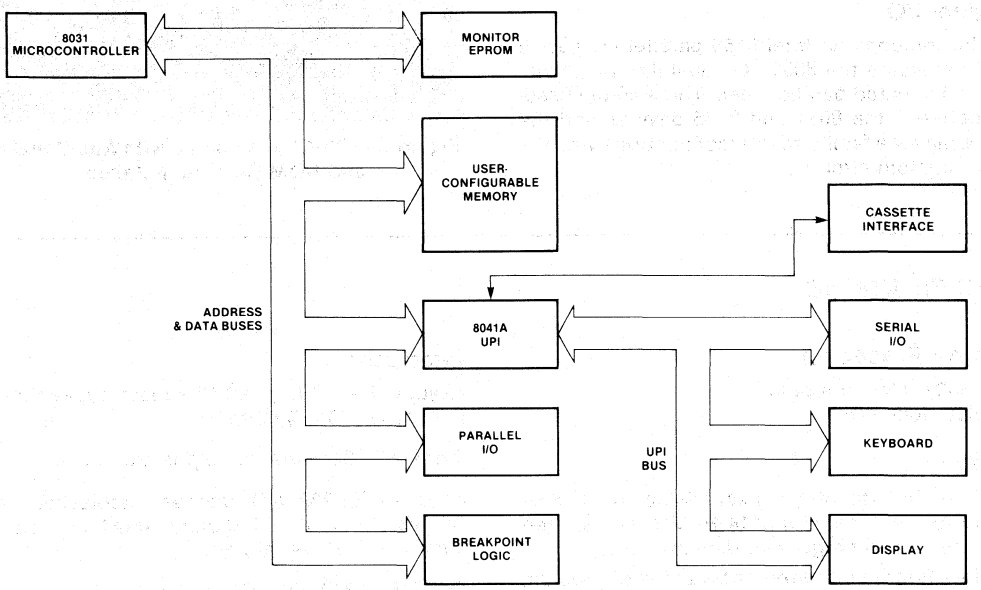


Figure 2. Block Diagram of SDK-51 System Design Kit

AFN-01792A

display. The standard keyboard and display provide full access to all of the SDK-51's capabilities. All of the SDK-51 interfaces are controlled by a pre-programmed Intel 8041 Universal Peripheral Interface.

A 3 × 4 matrix keyboard can be jumpered to port 1 of the 8031.

Optional Interfaces

TERMINAL

An RS-232-compatible CRT or printing terminal or a current-loop-interface terminal may be used as a listing device by connecting it to the board's serial interface connector and supplying + 12 and − 12 volts to the board.

AUDIO CASSETTE

The kit includes hardware, software, and user's guide instructions to connect and operate an audio cassette tape recorder for low-cost program and data storage.

INTELLEC SYTSTEM

An SDK-51 and an Intellec Model 800 or Series II development system with ISIS-II can upload and download files through the serial interface without adding any software to the Intellec system.

Parallel I/O

The kit includes an Intel 8155 parallel I/O device which expands the 8031 I/O capability by providing 22 dedicated parallel lines. Three 40-pin headers between the 8031 and 8155 devices and the wire-wrap area facilitate interconnections with the user's custom circuitry.

Debugging

Hardware breakpoint logic in the SDK-51 checks the address of a program or external data-memory access against values defined by the user and stops execution when it sees a "break" condition. After a breakpoint, you can examine and modify registers, memory locations, and other points in the system. A step command lets you execute instructions in a single-step mode.

Assembly and Test

The SDK-51 assembly manual describes hardware assembly in a step-by-step process that includes checking each hardware subsystem as it is installed. Building the system requires only a few common tools and standard laboratory instruments.

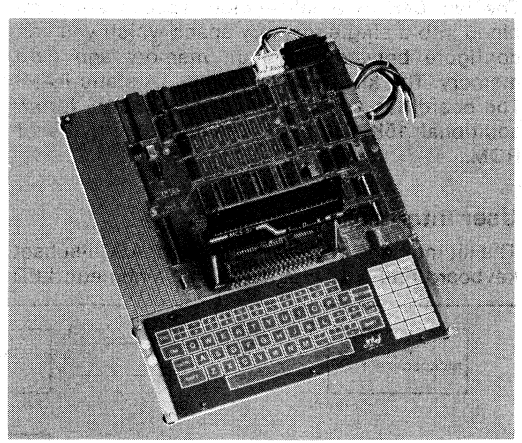


Figure 3. SDK-51 Assembled with Additional RAM and ROM Devices Installed

SPECIFICATIONS

Control Processor

Intel 8031 microcomputer
12 MHz clock rate

Memory

RAM — 1K-byte static, expandable in 1K segments to 16K-byte with 2114 RAM devices; user-configurable as program or data memory.

ROM — Printed circuitry for 8K bytes of program memory in 4K segments using 2732A EPROM devices.

Interfaces

Keyboard — 51-key, ASCII subset, typewriter format, 12-key (3 × 4) matrix

Display — 24-character, alpha-numeric

Serial — RS-232 with user-selectable baud rate. Printed circuitry for 110 baud 20 mA current loop interface. 8031 serial port.

Parallel — 22 lines, TTL compatible

Cassette — Audio cassette tape storage interface

Software

System monitor preprogrammed in on-board ROM

MCS-51 assembler and disassembler preprogrammed in on-board ROM

Interface control software preprogrammed in 8041's on-chip ROM

Assembly and Test Equipment Required

Needle-nose pliers

Small Phillips screwdriver

Small diagonal wire cutters

Soldering pencil, ≤30 watts, 1/16″ diameter tip

Rosin-core, 60-40 solder, 0.05″ diameter

Volt-Ohm-Milliammeter, 1 meg-ohm input impedance

Oscilloscope, 1 volt/division vertical sensitivity, 200 μs/division sweep rate, single trace, internal and external triggering

Physical Characteristics

Length — 13.5 in. (34.29 cm)
Width — 12 in. (30.48 cm)
Height — 4 in. (10.16 cm)
Weight — 3 lb (1.36 kg)

Electrical Characteristics

DC Power Requirement (supplied by user, cable included with kit)

Voltage	Current
+ 5V ±5%	3A
+12V ±5%*	100 mA
−12V ±5%*	100 mA

* ± 12 volts required only for operation with serial interface.

Environmental Characteristics

Operating Temperature — 0 to 40°C

Relative Humidity — 10% to 90%, non-condensing

Reference Manuals

SDK-51 User's Manual

SDK-51 Assembly Manual

SDK-51 Monitor Listing

MCS-51 Macro Assembler User's Guide

MCS-51 Macro Assembly Language Pocket Reference

ORDERING INFORMATION

Part Number	Description
MCI-51-SDK	MCS-51 System Design Kit

PLM Description of 8051 Instruction Set

A

```
ISIS-II PL/M-80 V3.1 COMPILATION OF MODULE SIM51
OBJECT MODULE PLACED IN SIM51.OBJ
COMPILER INVOKED BY:  :F1:PLM80 SIM51.PLM PRINT(:F1:SIM51.LST) XREF DATE(302)


            $TITLE('8051 INSTRUCTION SET SIMULATOR')


            /*  THE FOLLOWING IS A PLM-80 PROGRAM TO SIMULATE THE
                OPERATION OF THE INTEL 8051 INSTRUCTION SET.
                NO ATTEMPT HAS BEEN MADE TO SIMULATE THE I/O PORTS OR
                SPECIAL FUNCTION REGISTERS, THOUGH THERE ARE 'HOOKS' TO
                ACCESS EXTERNALLY-DEFINED PROCEDURES WHENEVER P0-P3 OR
                SBUF ARE READ OR WRITTEN.

            RELEVANT ENTRY POINTS:

            'INITIALIZE' - SIMULATE HARDWARE RESET;
                SUBROUTINE WITH NO INPUT PARAMETERS.

            'STEP' - SIMULATE EXECUTION OF ONE INSTRUCTION.
                INPUT PARAMETER = STARTING ADDRESS;
                VALUE RETURNED = UPDATED PC.

            'FETCH$SIM' - FETCH DATA FROM VARIOUS ADDRESS SPACES
                (SEE ROUTINE DEFINITION FOR PARAMETER DEFINITION).

            'STORE$SIM' - STORE DATA INTO VARIOUS ADDRESS SPACES
                (SEE ROUTINE DEFINITION FOR PARAMETER DEFINITION).

            ALL PARAMETERS PASSED TO AND FROM ALL ROUTINES ARE SIXTEEN-BIT.
```

```
                $EJECT

  1             SIM51:
                DO;

                /*  DEFINITIONS OF GLOBAL VARIABLES USED BY INDIVIDUAL
                    INSTRUCTION SIMULATION PROCEDURES:                    */

  2    1        DECLARE ROM$SIZE LITERALLY '4096';  /*  4K ROM SUPPORTED  */
  3    1        DECLARE INT$RAM(128) BYTE;
  4    1        DECLARE HARD$REG(128) BYTE;
  5    1        DECLARE USER$CODE(ROM$SIZE) BYTE PUBLIC;
  6    1        DECLARE EXTERN$RAM(256) BYTE;

  7    1        DECLARE REG$ADDR BYTE;
  8    1        DECLARE DIR$ADDR BYTE;
  9    1        DECLARE SOURCE$ADDR BYTE;
 10    1        DECLARE DEST$ADDR BYTE;
 11    1        DECLARE BIT$ADDR BYTE;
 12    1        DECLARE CODE$ADDR ADDRESS;
 13    1        DECLARE PAGED$EXTERNAL$ADDR BYTE;

 14    1        DECLARE REG$DATA BYTE;
 15    1        DECLARE DIR$DATA BYTE;
 16    1        DECLARE IND$DATA BYTE;
 17    1        DECLARE IMM$DATA BYTE;
 18    1        DECLARE BIT$DATA BYTE;
 19    1        DECLARE STACK$DATA BYTE;

 20    1        DECLARE PAGE$CODE BYTE;
 21    1        DECLARE PAGE$OFFSET BYTE;
 22    1        DECLARE DISPLACEMENT BYTE;

 23    1        DECLARE LINK$BIT BYTE;
 24    1        DECLARE LOW$NIB BYTE;
 25    1        DECLARE HIGH$NIB BYTE;
 26    1        DECLARE LOW$SOURCE$NIB BYTE;

 27    1        DECLARE ADD$TEMP ADDRESS;
 28    1        DECLARE SUB$TEMP BYTE;
 29    1        DECLARE MUL$TEMP ADDRESS;
 30    1        DECLARE DIV$TEMP BYTE;

 31    1        DECLARE PC ADDRESS;
 32    1        DECLARE OPCODE BYTE;
 33    1        DECLARE MACH$CYC ADDRESS PUBLIC;

 34    1        DECLARE BIT$REG$ADDR BYTE;
 35    1        DECLARE BIT$PATTERN BYTE;
 36    1        DECLARE BIT$MASK BYTE;
```

```
                $EJECT

                /*  PREDEFINED SPECIAL SYMBOLS FOR HARDWARE REGISTERS
                    (NOTE:  VALUES OFFSET BY 80H TO CORRESPOND TO INDEX INTO
                            HARD$REG ARRAY(0-127).          */

 37    1        DECLARE PO   LITERALLY 'HARD$REG(00H)';
 38    1        DECLARE SP   LITERALLY 'HARD$REG(01H)';
 39    1        DECLARE DPL  LITERALLY 'HARD$REG(02H)';
 40    1        DECLARE DPH  LITERALLY 'HARD$REG(03H)';
 41    1        DECLARE TCON LITERALLY 'HARD$REG(08H)';
 42    1        DECLARE TMOD LITERALLY 'HARD$REG(09H)';
 43    1        DECLARE TL0  LITERALLY 'HARD$REG(0AH)';
 44    1        DECLARE TL1  LITERALLY 'HARD$REG(0BH)';
 45    1        DECLARE TH0  LITERALLY 'HARD$REG(0CH)';
 46    1        DECLARE TH1  LITERALLY 'HARD$REG(0DH)';
 47    1        DECLARE P1   LITERALLY 'HARD$REG(10H)';
 48    1        DECLARE SCON LITERALLY 'HARD$REG(18H)';
 49    1        DECLARE SBUF LITERALLY 'HARD$REG(19H)';
 50    1        DECLARE P2   LITERALLY 'HARD$REG(20H)';
 51    1        DECLARE IE   LITERALLY 'HARD$REG(28H)';
 52    1        DECLARE P3   LITERALLY 'HARD$REG(30H)';
 53    1        DECLARE IP   LITERALLY 'HARD$REG(38H)';
 54    1        DECLARE PSW  LITERALLY 'HARD$REG(50H)';
 55    1        DECLARE ACC  LITERALLY 'HARD$REG(60H)';
 56    1        DECLARE B    LITERALLY 'HARD$REG(70H)';


                /*  HARDWARE REGISTER TYPE CODES ARE ASSIGNED AS FOLLOWS:
                        0 - REGISTER UNDEFINED OR BEYOND SCOPE OF SIMULATOR;
                        1 - REGISTER WRITTEN OR READ SIMPLY BY DIRECT ADDRESSING;
                        2 - I/O PORT;
                        3 - . . .       (RESERVED FOR EXPANSION)      */

 57    1        DECLARE HARD$REG$ATTRIB(128) BYTE DATA
                       (2,1,1,1,  0,0,0,0,  1,1,1,1,  1,1,0,0,
                        2,0,0,0,  0,0,0,0,  1,2,0,0,  0,0,0,0,
                        2,0,0,0,  0,0,0,0,  1,0,0,0,  0,0,0,0,
                        2,0,0,0,  0,0,0,0,  1,0,0,0,  0,0,0,0,
                        0,0,0,0,  0,0,0,0,  0,0,0,0,  0,0,0,0,
                        1,0,0,0,  0,0,0,0,  0,0,0,0,  0,0,0,0,
                        1,0,0,0,  0,0,0,0,  0,0,0,0,  0,0,0,0,
                        1,0,0,0,  0,0,0,0,  0,0,0,0,  0,0,0,0);

 58    1        DECLARE BIT$REG$MAP(32) BYTE DATA
                       ( 20H, 21H, 22H, 23H,   24H, 25H, 26H, 27H,
                         28H, 29H, 2AH, 2BH,   2CH, 2DH, 2EH, 2FH,
                         80H, 88H, 90H, 98H,  0A0H,0A8H,0B0H,0B8H,
                        0C0H,0C8H,0D0H,0D8H,  0E0H,0E8H,0F0H,0F8H);

 59    1        DECLARE MASK$TABLE(8) BYTE DATA
                       (00000001B,  00000010B,  00000100B,  00001000B,
                        00010000B,  00100000B,  01000000B,  10000000B);
```

```
                $EJECT

                /*  HOOKS FOR EXTERNAL I/O PORT AND ERROR
                    HANDLING ROUTINES:                          */

 60    1        PORT$OUTPUT:  PROCEDURE(PORT$NO,PORT$DATA) EXTERNAL;
 61    2            DECLARE (PORT$NO,PORT$DATA) BYTE;
 62    2        END PORT$OUTPUT;

 63    1        PORT$INPUT:  PROCEDURE(PORT$NO) BYTE EXTERNAL;
 64    2            DECLARE PORT$NO BYTE;
 65    2        END PORT$INPUT;


 66    1        DIR$ADDR$ERR:  PROCEDURE(HARD$REG$CODE) EXTERNAL;
 67    2            DECLARE HARD$REG$CODE BYTE;
 68    2        END DIR$ADDR$ERR;


 69    1        IND$ADDR$ERR:  PROCEDURE(ILLEGAL$IND$ADDR) EXTERNAL;
 70    2            DECLARE ILLEGAL$IND$ADDR BYTE;
 71    2        END IND$ADDR$ERR;


 72    1        STACK$ERR:  PROCEDURE(ILLEGAL$STACK$ADDR) EXTERNAL;
 73    2            DECLARE ILLEGAL$STACK$ADDR BYTE;
 74    2        END STACK$ERR;


 75    1        FETCH$PROG$ERR:  PROCEDURE(ILLEGAL$CODE$ADDR) EXTERNAL;
 76    2            DECLARE ILLEGAL$CODE$ADDR ADDRESS;
 77    2        END FETCH$PROG$ERR;
```

```
                $EJECT

                /*  VARIOUS MEMORY SPACE ACCESS ROUTINES:       */

 78   1         FETCH$PROGRAM:  PROCEDURE (CODE$ADDR) BYTE;
 79   2             DECLARE (CODE$ADDR) ADDRESS;
 80   2             IF CODE$ADDR < ROM$SIZE
                       THEN RETURN USER$CODE(CODE$ADDR);
 82   2                  ELSE DO;
 83   3                       CALL FETCH$PROG$ERR(CODE$ADDR);
 84   3                       RETURN 00H;
 85   3                       END;
 86   2         END FETCH$PROGRAM;



 87   1         FETCH$REG:  PROCEDURE (REG$NO) BYTE;
 88   2             DECLARE (REG$NO, REG$ADDR) BYTE;
 89   2             REG$ADDR=(PSW AND 00011000B) + REG$NO;
 90   2             RETURN INT$RAM(REG$ADDR);
 91   2         END FETCH$REG;


 92   1         STORE$REG:  PROCEDURE (REG$NO, DATA$VALUE);
 93   2             DECLARE (REG$NO, REG$ADDR, DATA$VALUE) BYTE;
 94   2             REG$ADDR=(PSW AND 00011000B) + REG$NO;
 95   2             INT$RAM(REG$ADDR)=DATA$VALUE;
 96   2         END STORE$REG;



 97   1         FETCH$DIR:  PROCEDURE(DIR$ADDR$NO) BYTE;
 98   2         DECLARE(DIR$ADDR$NO, HARD$REG$INDEX, HARD$REG$TYPE) BYTE;
 99   2             IF DIR$ADDR$NO <= 7FH THEN
100   2                 RETURN INT$RAM(DIR$ADDR$NO);
101   2             ELSE DO;
102   3                 HARD$REG$INDEX=DIR$ADDR$NO - 80H;
103   3                 HARD$REG$TYPE=HARD$REG$ATTRIB(HARD$REG$INDEX);
104   3                 DO CASE HARD$REG$TYPE;
105   4                     DO;
106   5                         CALL DIR$ADDR$ERR(DIR$ADDR$NO);
107   5                         RETURN 00H;
108   5                         END;
109   4                     RETURN HARD$REG(HARD$REG$INDEX);
110   4                     RETURN PORT$INPUT(DIR$ADDR$NO);
111   4                     END;
112   3                 END;
113   2         END FETCH$DIR;


114   1         FETCH$DIR$INT:  PROCEDURE(DIR$ADDR$NO) BYTE;
115   2         DECLARE(DIR$ADDR$NO, HARD$REG$INDEX, HARD$REG$TYPE) BYTE;
116   2             IF DIR$ADDR$NO <= 7FH THEN
117   2                 RETURN INT$RAM(DIR$ADDR$NO);
118   2             ELSE DO;
```

```
119   3               HARD$REG$INDEX=DIR$ADDR$NO - 80H;
120   3               HARD$REG$TYPE=HARD$REG$ATTRIB(HARD$REG$INDEX);
121   3               DO CASE HARD$REG$TYPE;
122   4                   DO;
123   5                       CALL DIR$ADDR$ERR(DIR$ADDR$NO);
124   5                       RETURN 00H;
125   5                       END;
126   4                   RETURN HARD$REG(HARD$REG$INDEX);
127   4                   RETURN HARD$REG(HARD$REG$INDEX);
128   4                   END;
129   3               END;
130   2       END FETCH$DIR$INT;


131   1       STORE$DIR:  PROCEDURE(DIR$ADDR$NO, DATA$VALUE);
132   2           DECLARE(DIR$ADDR$NO, HARD$REG$INDEX, HARD$REG$TYPE, DATA$VALUE)
133   2           IF DIR$ADDR$NO <= 7FH THEN
134   2               INT$RAM(DIR$ADDR$NO)=DATA$VALUE;
135   2           ELSE DO;
136   3               HARD$REG$INDEX=DIR$ADDR$NO - 80H;
137   3               HARD$REG$TYPE=HARD$REG$ATTRIB(HARD$REG$INDEX);
138   3               DO CASE HARD$REG$TYPE;
139   4                   CALL DIR$ADDR$ERR(DIR$ADDR$NO);
140   4                   HARD$REG(HARD$REG$INDEX)=DATA$VALUE;
141   4                   DO;
142   5                       HARD$REG(HARD$REG$INDEX)=DATA$VALUE;
143   5                       CALL PORT$OUTPUT(DIR$ADDR$NO, DATA$VALUE);
144   5                       END;
145   4                   END;
146   3               END;
147   2       END STORE$DIR;



148   1       FETCH$IND:  PROCEDURE(REG$NO) BYTE;
149   2           DECLARE (REG$NO, REG$ADDR, RAM$ADDR) BYTE;
150   2           REG$ADDR=(PSW AND 00011000B) + REG$NO;
151   2           RAM$ADDR=INT$RAM(REG$ADDR);
152   2           IF RAM$ADDR <= 7FH
                     THEN RETURN INT$RAM(RAM$ADDR),
154   2                ELSE DO;
155   3                     CALL IND$ADDR$ERR(RAM$ADDR);
156   3                     RETURN 00H;
157   3                     END;
158   2       END FETCH$IND;


159   1       STORE$IND:  PROCEDURE(REG$NO, DATA$VALUE);
160   2           DECLARE (REG$NO, REG$ADDR, RAM$ADDR, DATA$VALUE) BYTE;
161   2           REG$ADDR=(PSW AND 00011000B) + REG$NO;
162   2           RAM$ADDR=INT$RAM(REG$ADDR);
163   2           IF RAM$ADDR <= 7FH
                     THEN INT$RAM(RAM$ADDR)=DATA$VALUE;
165   2                ELSE CALL IND$ADDR$ERR(RAM$ADDR);
166   2       END STORE$IND;
```

```
167   1       FETCH$BIT:  PROCEDURE(BIT$ADDR) BYTE;
168   2           DECLARE BIT$ADDR BYTE;
169   2           BIT$REG$ADDR=BIT$REG$MAP(BIT$ADDR / 8);
170   2           BIT$PATTERN=FETCH$DIR(BIT$REG$ADDR);
171   2           BIT$MASK=MASK$TABLE(BIT$ADDR AND 00000111B);
172   2           IF (BIT$PATTERN AND BIT$MASK) = 0
                      THEN RETURN 00H;
174   2                 ELSE RETURN 1;
175   2       END FETCH$BIT;


176   1       FETCH$BIT$INT:  PROCEDURE(BIT$ADDR) BYTE;
177   2           DECLARE BIT$ADDR BYTE;
178   2           BIT$REG$ADDR=BIT$REG$MAP(BIT$ADDR / 8);
179   2           BIT$PATTERN=FETCH$DIR$INT(BIT$REG$ADDR);
180   2           BIT$MASK=MASK$TABLE(BIT$ADDR AND 00000111B);
181   2           IF (BIT$PATTERN AND BIT$MASK) = 0
                      THEN RETURN 00H;
183   2                 ELSE RETURN 1;
184   2       END FETCH$BIT$INT;


185   1       STORE$BIT:  PROCEDURE(BIT$ADDR,BIT$DATA);
186   2           DECLARE (BIT$ADDR,BIT$DATA) BYTE;
187   2           BIT$REG$ADDR=BIT$REG$MAP(BIT$ADDR / 8);
188   2           BIT$PATTERN=FETCH$DIR$INT(BIT$REG$ADDR);
189   2           BIT$MASK=MASK$TABLE(BIT$ADDR AND 00000111B);
190   2           IF BIT$DATA = 0
                      THEN BIT$PATTERN=BIT$PATTERN AND (NOT BIT$MASK);
192   2                 ELSE BIT$PATTERN=(BIT$PATTERN OR BIT$MASK);
193   2           CALL STORE$DIR(BIT$REG$ADDR,BIT$PATTERN);
194   2       END STORE$BIT;


195   1       PUSH$STACK:  PROCEDURE (DATA$VALUE);
196   2           DECLARE (DATA$VALUE) BYTE;
197   2           SP=SP+1;
198   2           IF SP <= 7FH
                      THEN INT$RAM(SP)=DATA$VALUE;
200   2                 ELSE CALL STACK$ERR(SP);
201   2       END PUSH$STACK;


202   1       POP$STACK:  PROCEDURE BYTE;
203   2           DECLARE (DATA$VALUE) BYTE;
204   2           IF SP <= 7FH
                      THEN DATA$VALUE=INT$RAM(SP);
206   2                 ELSE DO;
207   3                      CALL  STACK$ERR(SP);
208   3                      DATA$VALUE=00H;
209   3                      END;
210   2           SP=SP-1;
211   2           RETURN DATA$VALUE;
212   2       END POP$STACK;
```

```
213   1       FETCH$PAGED$EXTERNAL:  PROCEDURE (PAGED$EXTERN$ADDR) BYTE;
214   2           DECLARE (PAGED$EXTERN$ADDR) BYTE;
215   2           RETURN EXTERN$RAM(PAGED$EXTERN$ADDR);
216   2       END FETCH$PAGED$EXTERNAL;

217   1       STORE$PAGED$EXTERNAL:  PROCEDURE (PAGED$EXTERN$ADDR,DATA$BYTE);
218   2           DECLARE (PAGED$EXTERN$ADDR) BYTE;
219   2           DECLARE DATA$BYTE BYTE;
220   2           EXTERN$RAM(PAGED$EXTERN$ADDR) = DATA$BYTE;
221   2       END STORE$PAGED$EXTERNAL;

222   1       FETCH$LONG$EXTERNAL:  PROCEDURE (LONG$EXTERN$ADDR) BYTE;
223   2           DECLARE (LONG$EXTERN$ADDR) ADDRESS;
224   2           RETURN EXTERN$RAM(LONG$EXTERN$ADDR);
225   2       END FETCH$LONG$EXTERNAL;

226   1       STORE$LONG$EXTERNAL:  PROCEDURE (LONG$EXTERN$ADDR,DATA$BYTE);
227   2           DECLARE (LONG$EXTERN$ADDR) ADDRESS;
228   2           DECLARE DATA$BYTE BYTE;
229   2           EXTERN$RAM(LONG$EXTERN$ADDR) = DATA$BYTE;
230   2       END STORE$LONG$EXTERNAL;

231   1       SIGN$EXTENDED:  PROCEDURE (SIGNED$BYTE) ADDRESS;
232   2           DECLARE SIGNED$BYTE BYTE;
233   2           IF (SIGNED$BYTE AND 10000000B) = 0
                      THEN RETURN SIGNED$BYTE;
235   2                 ELSE RETURN (SIGNED$BYTE + 0FF00H);
236   2       END SIGN$EXTENDED;

237   1       PARITY$STATE:  PROCEDURE (DATA$BYTE) BYTE;
238   2           DECLARE (DATA$BYTE,PARITY$BIT) BYTE;
239   2           PARITY$BIT=0;
240   2           IF (DATA$BYTE AND 00000001B) <> 0
                      THEN PARITY$BIT=PARITY$BIT XOR 00000001B;
242   2           IF (DATA$BYTE AND 00000010B) <> 0
                      THEN PARITY$BIT=PARITY$BIT XOR 00000001B;
244   2           IF (DATA$BYTE AND 00000100B) <> 0
                      THEN PARITY$BIT=PARITY$BIT XOR 00000001B;
246   2           IF (DATA$BYTE AND 00001000B) <> 0
                      THEN PARITY$BIT=PARITY$BIT XOR 00000001B;
248   2           IF (DATA$BYTE AND 00010000B) <> 0
                      THEN PARITY$BIT=PARITY$BIT XOR 00000001B;
250   2           IF (DATA$BYTE AND 00100000B) <> 0
                      THEN PARITY$BIT=PARITY$BIT XOR 00000001B;
```

```
 252   2           IF (DATA$BYTE AND 01000000B) <> 0
                       THEN PARITY$BIT=PARITY$BIT XOR 00000001B;
 254   2           IF (DATA$BYTE AND 10000000B) <> 0
                       THEN PARITY$BIT=PARITY$BIT XOR 00000001B;
 256   2           RETURN PARITY$BIT;
 257   2       END PARITY$STATE;
```

```
                $EJECT
                /*  THE FOLLOWING CODE PROVIDES A SINGLE ENTRY POINT FOR
                    AN EXTERNAL ROUTINE TO READ DATA FROM ALL SIMULATOR ADDRESS S
                    THE FIRST CALLING PARAMETER GIVES UP TO 16 BITS OF ADDRESS;
                    THE SECOND SPECIFIES WHICH LOGICAL ADDRESS SPACE TO
                    READ, USING THE SCHEME:

                    0 = PROGRAM MEMORY
                    1 = WORKING REGISTER
                    2 = DIRECT ADDRESS (INPUTS FOR PORTS)
                    3 = INDIRECT THROUGH REGISTER SPECIFIED
                    4 = DIRECT BIT ADDRESS (DATA RIGHT-JUSTIFIED)
                    5 = PAGED EXTERNAL MEMORY
                    6 = 64K EXTERNAL MEMORY (ALL WRITES CURRENTLY TO PAGE 0)
                    7 = TOP-OF-STACK (SP UPDATED)

                    THE FUNCTION CALL RETURNS THE BYTE SO ADDRESSED.

258   1         FETCH$SIM:  PROCEDURE (DATA$ADDR,DATA$TYPE) ADDRESS PUBLIC;
259   2             DECLARE (RETURN$DATA,DATA$ADDR,DATA$TYPE) ADDRESS;
260   2             DECLARE DATA$ADDR$BYTE BYTE;
261   2             DATA$ADDR$BYTE=DATA$ADDR;
262   2             DO CASE DATA$TYPE;
263   3                 RETURN$DATA=FETCH$PROGRAM(DATA$ADDR);
264   3                 RETURN$DATA=FETCH$REG(DATA$ADDR$BYTE);
265   3                 RETURN$DATA=FETCH$DIR$INT(DATA$ADDR$BYTE);
266   3                 RETURN$DATA=FETCH$IND(DATA$ADDR$BYTE);
267   3                 RETURN$DATA=FETCH$BIT$INT(DATA$ADDR$BYTE);
268   3                 RETURN$DATA=FETCH$PAGED$EXTERNAL(DATA$ADDR$BYTE);
269   3                 RETURN$DATA=FETCH$LONG$EXTERNAL(DATA$ADDR);
270   3                 RETURN$DATA=POP$STACK;
271   3                 END;
272   2             RETURN RETURN$DATA;
273   2         END FETCH$SIM;



                /*  THE FOLLOWING CODE PROVIDES A SINGLE ENTRY POINT FOR
                    AN EXTERNAL ROUTINE TO WRITE DATA INTO ALL SIMULATOR ADDRESS
                    THE FIRST CALLING PARAMETER GIVES UP TO 16 BITS OF ADDRESS;
                    THE SECOND SPECIFIES WHICH LOGICAL ADDRESS SPACE TO
                    READ, USING THE SCHEME:

                    0 = PROGRAM MEMORY
                    1 = WORKING REGISTER
                    2 = DIRECT ADDRESS (OUTPUT LATCHES FOR PORTS)
                    3 = INDIRECT THROUGH REGISTER SPECIFIED
                    4 = DIRECT BIT ADDRESS (DATA RIGHT-JUSTIFIED)
                    5 = PAGED EXTERNAL MEMORY
                    6 = 64K EXTERNAL MEMORY (ALL WRITTEN CURRENTLY TO PAGE 0)
                    7 = TOP-OF-STACK (SP UPDATED)

                    THE THIRD PARAMETER HOLDS THE BYTE VALUE TO BE WRITTEN      */

274   1         STORE$SIM:  PROCEDURE (DATA$ADDR,DATA$TYPE,DATA$VALUE) PUBLIC;
275   2             DECLARE (DATA$ADDR,DATA$TYPE,DATA$VALUE) ADDRESS;
276   2             DECLARE (DATA$ADDR$BYTE,DATA$VALUE$BYTE) BYTE;
```

```
277   2           DATA$ADDR$BYTE=DATA$ADDR;
278   2           DATA$VALUE$BYTE=DATA$VALUE;
279   2           DO CASE DATA$TYPE;
280   3               USER$CODE(DATA$ADDR MOD ROM$SIZE)=DATA$VALUE$BYTE;
281   3               CALL STORE$REG(DATA$ADDR$BYTE,DATA$VALUE$BYTE);
282   3               CALL STORE$DIR(DATA$ADDR$BYTE,DATA$VALUE$BYTE);
283   3               CALL STORE$IND(DATA$ADDR$BYTE,DATA$VALUE$BYTE);
284   3               CALL STORE$BIT(DATA$ADDR$BYTE,DATA$VALUE$BYTE);
285   3               CALL STORE$PAGED$EXTERNAL(DATA$ADDR$BYTE,DATA$VALUE$BYTE);
286   3               CALL STORE$LONG$EXTERNAL(DATA$ADDR,DATA$VALUE$BYTE);
287   3               CALL PUSH$STACK(DATA$VALUE$BYTE);
288   3               END;
289   2       END STORE$SIM;
```

```
                    $EJECT
                    $INCLUDE (ISET51.PLM)
              =     /*  INDIVIDUAL INSTRUCTION PROCEDURES:  */
              =
              =
              =     /*  "ACALL   addr11"  INSTRUCTION:  */
              =
 290    1     =     ACALL$ADDR11:  PROCEDURE;
 291    2     =         PAGE$CODE=(OPCODE AND 11100000B) / 32;
 292    2     =         PAGE$OFFSET=FETCH$PROGRAM(PC+1);
 293    2     =         PC=PC+2;
 294    2     =         CALL PUSH$STACK(LOW(PC));
 295    2     =         CALL PUSH$STACK(HIGH(PC));
 296    2     =         PC=(PC AND 0F800H) + (PAGE$CODE * 100H) + PAGE$OFFSET;
 297    2     =     END ACALL$ADDR11;
              =
              =
              =     /*  "ADD     A,<src-byte>"  FUNCTION:  */
              =
 298    1     =     ADD$A:  PROCEDURE(DATA$BYTE);
 299    2     =         DECLARE DATA$BYTE BYTE;
 300    2     =         IF ((ACC AND 0FH)+(DATA$BYTE AND 0FH)) > 0FH
              =             THEN PSW=PSW OR  01000000B;
 302    2     =             ELSE PSW=PSW AND 10111111B;
 303    2     =         IF ((ACC AND 7FH)+(DATA$BYTE AND 7FH)) > 7FH
              =             THEN PSW=PSW OR  00000100B;
 305    2     =             ELSE PSW=PSW AND 11111011B;
 306    2     =         ADD$TEMP = (ACC);
 307    2     =         ADD$TEMP = (ADD$TEMP+DATA$BYTE);
 308    2     =         IF ADD$TEMP > 0FFH
              =             THEN PSW=(PSW OR  10000000B) XOR 00000100B;
 310    2     =             ELSE PSW=(PSW AND 01111111B);
 311    2     =         ACC=LOW(ADD$TEMP);
 312    2     =     END ADD$A;
              =
              =
              =     /*  "ADD     A,Rn"  INSTRUCTION:  */
              =
 313    1     =     ADD$A$REG:  PROCEDURE;
 314    2     =         REG$ADDR=OPCODE AND 00000111B;
 315    2     =         REG$DATA=FETCH$REG(REG$ADDR);
 316    2     =         CALL ADD$A(REG$DATA);
 317    2     =         PC=PC+1;
 318    2     =     END ADD$A$REG;
              =
              =
              =     /*  "ADD     A,direct"  INSTRUCTION:  */
              =
 319    1     =     ADD$A$DIR:  PROCEDURE;
 320    2     =         DIR$ADDR=FETCH$PROGRAM(PC+1);
 321    2     =         DIR$DATA=FETCH$DIR(DIR$ADDR);
 322    2     =         CALL ADD$A(DIR$DATA);
 323    2     =         PC=PC+2;
 324    2     =     END ADD$A$DIR;
              =
              =
              =     /*  "ADD     A,@Ri"  INSTRUCTION:  */
```

```
                =
325     1       =   ADD$A$IND:  PROCEDURE;
326     2       =       REG$ADDR=OPCODE AND 00000001B;
327     2       =       IND$DATA=FETCH$IND(REG$ADDR);
328     2       =       CALL ADD$A(IND$DATA);
329     2       =       PC=PC+1;
330     2       =   END ADD$A$IND;
                =
                =
                =   /*  "ADD     A,#data"  INSTRUCTION:  */
                =
331     1       =   ADD$A$IMM:  PROCEDURE;
332     2       =       IMM$DATA=FETCH$PROGRAM(PC+1);
333     2       =       CALL ADD$A(IMM$DATA);
334     2       =       PC=PC+2;
335     2       =   END ADD$A$IMM;
                =
                =
                =   /*  "ADDC    A,<src-byte>"  FUNCTION:  */
                =
336     1       =   ADDC$A:  PROCEDURE(DATA$BYTE);
337     2       =       DECLARE DATA$BYTE BYTE;
338     2       =       LINK$BIT = (PSW AND 10000000B) / 128;
339     2       =       IF ((ACC AND 0FH)+(DATA$BYTE AND 0FH)+LINK$BIT) > 0FH
                =           THEN PSW=PSW OR  01000000B;
341     2       =           ELSE PSW=PSW AND 10111111B;
342     2       =       IF ((ACC AND 7FH)+(DATA$BYTE AND 7FH)+LINK$BIT) > 7FH
                =           THEN PSW=PSW OR  00000100B;
344     2       =           ELSE PSW=PSW AND 11111011B;
345     2       =       ADD$TEMP = (ACC);
346     2       =       ADD$TEMP = (ADD$TEMP+DATA$BYTE);
347     2       =       ADD$TEMP = (ADD$TEMP+LINK$BIT);
348     2       =       IF ADD$TEMP > 0FFH
                =           THEN PSW=(PSW OR  10000000B) XOR 00000100B;
350     2       =           ELSE PSW=(PSW AND 01111111B);
351     2       =       ACC=LOW(ADD$TEMP);
352     2       =   END ADDC$A;
                =
                =
                =   /*  "ADDC    A,Rn"  INSTRUCTION:  */
                =
353     1       =   ADDC$A$REG:  PROCEDURE;
354     2       =       REG$ADDR=OPCODE AND 00000111B;
355     2       =       REG$DATA=FETCH$REG(REG$ADDR);
356     2       =       CALL ADDC$A(REG$DATA);
357     2       =       PC=PC+1;
358     2       =   END ADDC$A$REG;
                =
                =
                =   /*  "ADDC    A,direct"  INSTRUCTION:  */
                =
359     1       =   ADDC$A$DIR:  PROCEDURE;
360     2       =       DIR$ADDR=FETCH$PROGRAM(PC+1);
361     2       =       DIR$DATA=FETCH$DIR(DIR$ADDR);
362     2       =       CALL ADDC$A(DIR$DATA);
363     2       =       PC=PC+2;
364     2       =   END ADDC$A$DIR;
```

```
                =
                =
                =   /*  "ADDC    A,@Ri"  INSTRUCTION:  */
                =
 365    1       =   ADDC$A$IND:  PROCEDURE;
 366    2       =       REG$ADDR=OPCODE AND 00000001B;
 367    2       =       IND$DATA=FETCH$IND(REG$ADDR);
 368    2       =       CALL ADDC$A(IND$DATA);
 369    2       =       PC=PC+1;
 370    2       =   END ADDC$A$IND;
                =
                =
                =   /*  "ADDC    A,#data"  INSTRUCTION:  */
                =
 371    1       =   ADDC$A$IMM:  PROCEDURE;
 372    2       =       IMM$DATA=FETCH$PROGRAM(PC+1);
 373    2       =       CALL ADDC$A(IMM$DATA);
 374    2       =       PC=PC+2;
 375    2       =   END ADDC$A$IMM;
                =
                =
                =   /*  "AJMP    addr11"  INSTRUCTION:  */
                =
 376    1       =   AJMP$ADDR11:  PROCEDURE;
 377    2       =       PAGE$CODE=(OPCODE AND 11100000B) / 32;
 378    2       =       PAGE$OFFSET=FETCH$PROGRAM(PC+1);
 379    2       =       PC=PC+2;
 380    2       =       PC=(PC AND 0F800H) + (PAGE$CODE * 100H) + PAGE$OFFSET;
 381    2       =   END AJMP$ADDR11;
                =
                =
                =   /*  "ANL     A,Rn"  INSTRUCTION:  */
                =
 382    1       =   ANL$A$REG:  PROCEDURE;
 383    2       =       REG$ADDR=OPCODE AND 00000111B;
 384    2       =       REG$DATA=FETCH$REG(REG$ADDR);
 385    2       =       ACC=ACC AND REG$DATA;
 386    2       =       PC=PC+1;
 387    2       =   END ANL$A$REG;
                =
                =
                =   /*  "ANL     A,direct"  INSTRUCTION:  */
                =
 388    1       =   ANL$A$DIR:  PROCEDURE;
 389    2       =       DIR$ADDR=FETCH$PROGRAM(PC+1);
 390    2       =       DIR$DATA=FETCH$DIR(DIR$ADDR);
 391    2       =       ACC=ACC AND DIR$DATA;
 392    2       =       PC=PC+2;
 393    2       =   END ANL$A$DIR;
                =
                =
                =   /*  "ANL     A,@Ri"  INSTRUCTION:  */
                =
 394    1       =   ANL$A$IND:  PROCEDURE;
 395    2       =       REG$ADDR=OPCODE AND 00000001B;
 396    2       =       IND$DATA=FETCH$IND(REG$ADDR);
 397    2       =       ACC=ACC AND IND$DATA;
```

```
398   2  =        PC=PC+1;
399   2  =    END ANL$A$IND;
           =
           =
           =    /*  "ANL      A,#data"  INSTRUCTION:  */
           =
400   1  =    ANL$A$IMM:  PROCEDURE;
401   2  =        IMM$DATA=FETCH$PROGRAM(PC+1);
402   2  =        ACC=ACC AND IMM$DATA;
403   2  =        PC=PC+2;
404   2  =    END ANL$A$IMM;
           =
           =
           =    /*  "ANL      direct,A"  INSTRUCTION:  */
           =
405   1  =    ANL$DIR$A:  PROCEDURE;
406   2  =        DIR$ADDR=FETCH$PROGRAM(PC+1);
407   2  =        DIR$DATA=FETCH$DIR$INT(DIR$ADDR);
408   2  =        CALL STORE$DIR(DIR$ADDR,ACC AND DIR$DATA);
409   2  =        PC=PC+2;
410   2  =    END ANL$DIR$A;
           =
           =
           =    /*  "ANL      direct,#data"  INSTRUCTION:  */
           =
411   1  =    ANL$DIR$IMM:  PROCEDURE;
412   2  =        DIR$ADDR=FETCH$PROGRAM(PC+1);
413   2  =        IMM$DATA=FETCH$PROGRAM(PC+2);
414   2  =        DIR$DATA=FETCH$DIR$INT(DIR$ADDR);
415   2  =        CALL STORE$DIR(DIR$ADDR,IMM$DATA AND DIR$DATA);
416   2  =        PC=PC+3;
417   2  =    END ANL$DIR$IMM;
           =
           =
           =    /*  "ANL      C,bit"  INSTRUCTION:  */
           =
418   1  =    ANL$C$BIT:  PROCEDURE;
419   2  =        BIT$ADDR=FETCH$PROGRAM(PC+1);
420   2  =        BIT$DATA=FETCH$BIT(BIT$ADDR);
421   2  =        IF BIT$DATA = 0 THEN PSW=PSW AND 01111111B;
423   2  =        PC=PC+2;
424   2  =    END ANL$C$BIT;
           =
           =
           =    /*  "ANL      C,/bit"  INSTRUCTION:  */
           =
425   1  =    ANL$C$COMP$BIT:  PROCEDURE;
426   2  =        BIT$ADDR=FETCH$PROGRAM(PC+1);
427   2  =        BIT$DATA=FETCH$BIT(BIT$ADDR);
428   2  =        IF BIT$DATA = 1 THEN PSW=PSW AND 01111111B;
430   2  =        PC=PC+2;
431   2  =    END ANL$C$COMP$BIT;
           =
           =
           =    /*  "CJNE     A,direct,rel"  INSTRUCTION:  */
           =
432   1  =    CJNE$A$DIR$REL:  PROCEDURE;
```

```
433   2  =        DIR$ADDR=FETCH$PROGRAM(PC+1);
434   2  =        DIR$DATA=FETCH$DIR(DIR$ADDR);
435   2  =        DISPLACEMENT=FETCH$PROGRAM(PC+2);
436   2  =        IF ACC < DIR$DATA
            =            THEN PSW=(PSW OR  10000000B);
438   2  =            ELSE PSW=(PSW AND 01111111B);
439   2  =        PC=PC+3;
440   2  =        IF ACC <> DIR$DATA
            =            THEN PC=PC+SIGN$EXTENDED(DISPLACEMENT);
442   2  =    END CJNE$A$DIR$REL;
            =
            =
            =    /*  "CJNE    A,#data,rel"  INSTRUCTION:  */
            =
443   1  =    CJNE$A$IMM$REL:  PROCEDURE;
444   2  =        IMM$DATA=FETCH$PROGRAM(PC+1);
445   2  =        DISPLACEMENT=FETCH$PROGRAM(PC+2);
446   2  =        IF ACC < IMM$DATA
            =            THEN PSW=(PSW OR  10000000B);
448   2  =            ELSE PSW=(PSW AND 01111111B);
449   2  =        PC=PC+3;
450   2  =        IF ACC <> IMM$DATA
            =            THEN PC=PC+SIGN$EXTENDED(DISPLACEMENT);
452   2  =    END CJNE$A$IMM$REL;
            =
            =
            =    /*  "CJNE    Rn,#data,rel"  INSTRUCTION:  */
            =
453   1  =    CJNE$REG$IMM$REL:  PROCEDURE;
454   2  =        REG$ADDR=OPCODE AND 00000111B;
455   2  =        REG$DATA=FETCH$REG(REG$ADDR);
456   2  =        IMM$DATA=FETCH$PROGRAM(PC+1);
457   2  =        DISPLACEMENT=FETCH$PROGRAM(PC+2);
458   2  =        IF REG$DATA < IMM$DATA
            =            THEN PSW=(PSW OR  10000000B);
460   2  =            ELSE PSW=(PSW AND 01111111B);
461   2  =        PC=PC+3;
462   2  =        IF REG$DATA <> IMM$DATA
            =            THEN PC=PC+SIGN$EXTENDED(DISPLACEMENT);
464   2  =    END CJNE$REG$IMM$REL;
            =
            =
            =    /*  "CJNE    @Ri,#data,rel"  INSTRUCTION:  */
            =
465   1  =    CJNE$IND$IMM$REL:  PROCEDURE;
466   2  =        REG$ADDR=OPCODE AND 00000001B;
467   2  =        IND$DATA=FETCH$IND(REG$ADDR);
468   2  =        IMM$DATA=FETCH$PROGRAM(PC+1);
469   2  =        DISPLACEMENT=FETCH$PROGRAM(PC+2);
470   2  =        IF IND$DATA < IMM$DATA
            =            THEN PSW=(PSW OR  10000000B);
472   2  =            ELSE PSW=(PSW AND 01111111B);
473   2  =        PC=PC+3;
474   2  =        IF IND$DATA <> IMM$DATA
            =            THEN PC=PC+SIGN$EXTENDED(DISPLACEMENT);
476   2  =    END CJNE$IND$IMM$REL;
            =
```

```
                =
                =   /*  "CLR     A"  INSTRUCTION:  */
                =
 477    1       =   CLR$A:  PROCEDURE;
 478    2       =       ACC=0;
 479    2       =       PC=PC+1;
 480    2       =   END CLR$A;
                =
                =
                =   /*  "CLR     C"  INSTRUCTION:  */
                =
 481    1       =   CLR$C:  PROCEDURE;
 482    2       =       PSW=PSW AND 01111111B;
 483    2       =       PC=PC+1;
 484    2       =   END CLR$C;
                =
                =
                =   /*  "CLR     bit"  INSTRUCTION:  */
                =
 485    1       =   CLR$BIT:  PROCEDURE;
 486    2       =       BIT$ADDR=FETCH$PROGRAM(PC+1);
 487    2       =       CALL STORE$BIT(BIT$ADDR,0);
 488    2       =       PC=PC+2;
 489    2       =   END CLR$BIT;
                =
                =
                =   /*  "CPL     A"  INSTRUCTION:  */
                =
 490    1       =   CPL$A:  PROCEDURE;
 491    2       =       ACC=ACC XOR 11111111B;
 492    2       =       PC=PC+1;
 493    2       =   END CPL$A;
                =
                =
                =   /*  "CPL     C"  INSTRUCTION:  */
                =
 494    1       =   CPL$C:  PROCEDURE;
 495    2       =       PSW=PSW XOR 10000000B;
 496    2       =       PC=PC+1;
 497    2       =   END CPL$C;
                =
                =
                =   /*  "CPL     bit"  INSTRUCTION:  */
                =
 498    1       =   CPL$BIT:  PROCEDURE;
 499    2       =       BIT$ADDR=FETCH$PROGRAM(PC+1);
 500    2       =       BIT$DATA=FETCH$BIT$INT(BIT$ADDR);
 501    2       =       BIT$DATA=BIT$DATA XOR 00000001B;
 502    2       =       CALL STORE$BIT(BIT$ADDR,BIT$DATA);
 503    2       =       PC=PC+2;
 504    2       =   END CPL$BIT;
                =
                =
                =   /*  "DA      A"  INSTRUCTION:  */
                =
 505    1       =   DA$A:  PROCEDURE;
 506    2       =       IF ((ACC AND 0FH) > 09H) OR ((PSW AND 01000000B) <> 0)
```

```
                =              THEN DO;
 508    3       =                  IF ACC >= OFAH THEN PSW=PSW OR 10000000B;
 511    3       =                  ACC=ACC+6;
 512    3       =                  END;
 513    2       =          IF ((ACC AND OFOH) > 90H) OR ((PSW AND 10000000B) <> 0)
                =              THEN DO;
 515    3       =                  IF ACC >= OAOH THEN PSW=PSW OR 10000000B;
                =                  ACC=ACC+60H;
 518    3       =                  END;
 519    2       =          PC=PC+1;
 520    2       =      END DA$A;
                =
                =
                =      /*  "DEC      A"  INSTRUCTION:  */
                =
 521    1       =      DEC$A:  PROCEDURE;
 522    2       =          ACC=ACC-1;
 523    2       =          PC=PC+1;
 524    2       =      END DEC$A;
                =
                =
                =      /*  "DEC      Rn"  INSTRUCTION:  */
                =
 525    1       =      DEC$REG:  PROCEDURE;
 526    2       =          REG$ADDR=OPCODE AND 00000111B;
 527    2       =          REG$DATA=FETCH$REG(REG$ADDR);
 528    2       =          REG$DATA=REG$DATA-1;
 529    2       =          CALL STORE$REG(REG$ADDR,REG$DATA);
 530    2       =          PC=PC+1;
 531    2       =      END DEC$REG;
                =
                =
                =      /*  "DEC      direct"  INSTRUCTION:  */
                =
 532    1       =      DEC$DIR:  PROCEDURE;
 533    2       =          DIR$ADDR=FETCH$PROGRAM(PC+1);
 534    2       =          DIR$DATA=FETCH$DIR$INT(DIR$ADDR);
 535    2       =          DIR$DATA=DIR$DATA-1;
 536    2       =          CALL STORE$DIR(DIR$ADDR,DIR$DATA);
 537    2       =          PC=PC+2;
 538    2       =      END DEC$DIR;
                =
                =
                =      /*  "DEC      @Ri"  INSTRUCTION:  */
                =
 539    1       =      DEC$IND:  PROCEDURE;
 540    2       =          REG$ADDR=OPCODE AND 00000001B;
 541    2       =          IND$DATA=FETCH$IND(REG$ADDR);
 542    2       =          IND$DATA=IND$DATA-1;
 543    2       =          CALL STORE$IND(REG$ADDR,IND$DATA);
 544    2       =          PC=PC+1;
 545    2       =      END DEC$IND;
                =
                =
                =      /*  "DIV      AB"  INSTRUCTION:  */
                =
```

```
546   1  =  DIV$AB:  PROCEDURE;
547   2  =      IF B = 0 THEN PSW=PSW OR 00000100B;
549   2  =          ELSE DO;
550   3  =              PSW=PSW AND 11111011B;
551   3  =              DIV$TEMP=ACC / B;
552   3  =              B=ACC MOD B;
553   3  =              ACC=DIV$TEMP;
554   3  =              END;
555   2  =      PSW=PSW AND 01111111B;
556   2  =      PC=PC+1;
557   2  =  END DIV$AB;
         =
         =
         =  /*  "DJNZ    Rn,rel"  INSTRUCTION:  */
         =
558   1  =  DJNZ$REG$REL:  PROCEDURE;
559   2  =      REG$ADDR=OPCODE AND 00000111B;
560   2  =      DISPLACEMENT=FETCH$PROGRAM(PC+1);
561   2  =      REG$DATA=FETCH$REG(REG$ADDR);
562   2  =      REG$DATA=REG$DATA-1;
563   2  =      CALL STORE$REG(REG$ADDR,REG$DATA);
564   2  =      PC=PC+2;
565   2  =      IF REG$DATA <> 0
         =          THEN PC=PC+SIGN$EXTENDED(DISPLACEMENT);
567   2  =  END DJNZ$REG$REL;
         =
         =
         =  /*  "DJNZ    direct,rel"  INSTRUCTION:  */
         =
568   1  =  DJNZ$DIR$REL:  PROCEDURE;
569   2  =      DIR$ADDR=FETCH$PROGRAM(PC+1);
570   2  =      DISPLACEMENT=FETCH$PROGRAM(PC+2);
571   2  =      DIR$DATA=FETCH$DIR$INT(DIR$ADDR);
572   2  =      DIR$DATA=DIR$DATA-1;
573   2  =      CALL STORE$DIR(DIR$ADDR,DIR$DATA);
574   2  =      PC=PC+3;
575   2  =      IF DIR$DATA <> 0
         =          THEN PC=PC+SIGN$EXTENDED(DISPLACEMENT);
577   2  =  END DJNZ$DIR$REL;
         =
         =
         =  /*  "INC     A"  INSTRUCTION:  */
         =
578   1  =  INC$A:  PROCEDURE;
579   2  =      ACC=ACC+1;
580   2  =      PC=PC+1;
581   2  =  END INC$A;
         =
         =
         =  /*  "INC     Rn"  INSTRUCTION:  */
         =
582   1  =  INC$REG:  PROCEDURE;
583   2  =      REG$ADDR=OPCODE AND 00000111B;
584   2  =      REG$DATA=FETCH$REG(REG$ADDR);
585   2  =      REG$DATA=REG$DATA+1;
586   2  =      CALL STORE$REG(REG$ADDR,REG$DATA);
587   2  =      PC=PC+1;
```

```
588   2  =    END INC$REG;
           =
           =
           =    /*  "INC      direct"  INSTRUCTION:   */
           =
589   1  =    INC$DIR:  PROCEDURE;
590   2  =        DIR$ADDR=FETCH$PROGRAM(PC+1);
591   2  =        DIR$DATA=FETCH$DIR$INT(DIR$ADDR);
592   2  =        DIR$DATA=DIR$DATA+1;
593   2  =        CALL STORE$DIR(DIR$ADDR,DIR$DATA);
594   2  =        PC=PC+2;
595   2  =    END INC$DIR;
           =
           =
           =    /*  "INC      @Ri"  INSTRUCTION:   */
           =
596   1  =    INC$IND:  PROCEDURE;
597   2  =        REG$ADDR=OPCODE AND 00000001B;
598   2  =        IND$DATA=FETCH$IND(REG$ADDR);
599   2  =        IND$DATA=IND$DATA+1;
600   2  =        CALL STORE$IND(REG$ADDR,IND$DATA);
601   2  =        PC=PC+1;
602   2  =    END INC$IND;
           =
           =
           =    /*  "INC      DPTR"  INSTRUCTION:   */
           =
603   1  =    INC$DPTR:  PROCEDURE;
604   2  =        DPL=DPL+1;
605   2  =        IF DPL=0 THEN DPH=DPH+1;
607   2  =        PC=PC+1;
608   2  =    END INC$DPTR;
           =
           =
           =    /*  "JB       bit,rel"  INSTRUCTION:   */
           =
609   1  =    JB$BIT$REL:  PROCEDURE;
610   2  =        BIT$ADDR=FETCH$PROGRAM(PC+1);
611   2  =        BIT$DATA=FETCH$BIT(BIT$ADDR);
612   2  =        DISPLACEMENT=FETCH$PROGRAM(PC+2);
613   2  =        PC=PC+3;
614   2  =        IF BIT$DATA=1
           =            THEN PC=PC+SIGN$EXTENDED(DISPLACEMENT);
616   2  =    END JB$BIT$REL;
           =
           =
           =    /*  "JBC      bit,rel"  INSTRUCTION:   */
           =
617   1  =    JBC$BIT$REL:  PROCEDURE;
618   2  =        BIT$ADDR=FETCH$PROGRAM(PC+1);
619   2  =        BIT$DATA=FETCH$BIT$INT(BIT$ADDR);
620   2  =        DISPLACEMENT=FETCH$PROGRAM(PC+2);
621   2  =        PC=PC+3;
622   2  =        CALL STORE$BIT(BIT$ADDR,0);
623   2  =        IF BIT$DATA=1
           =            THEN PC=PC+SIGN$EXTENDED(DISPLACEMENT);
625   2  =    END JBC$BIT$REL;
```

```
                =
                =
                =   /*  "JC      rel"  INSTRUCTION:  */
                =
 626    1       =   JC$REL:  PROCEDURE;
 627    2       =       DISPLACEMENT=FETCH$PROGRAM(PC+1);
 628    2       =       PC=PC+2;
 629    2       =       IF (PSW AND 10000000B) <> 0
                =           THEN PC=PC+SIGN$EXTENDED(DISPLACEMENT);
 631    2       =   END JC$REL;
                =
                =
                =   /*  "JMP     @A+DPTR"  INSTRUCTION:  */
                =
 632    1       =   JMP$ADPTR:  PROCEDURE;
 633    2       =       CODE$ADDR=(DPH*256)+DPL+ACC;
 634    2       =       PC=CODE$ADDR;
 635    2       =   END JMP$ADPTR;
                =
                =
                =   /*  "JNB     bit,rel"  INSTRUCTION:  */
                =
 636    1       =   JNB$BIT$REL:  PROCEDURE;
 637    2       =       BIT$ADDR=FETCH$PROGRAM(PC+1);
 638    2       =       DISPLACEMENT=FETCH$PROGRAM(PC+2);
 639    2       =       BIT$DATA=FETCH$BIT(BIT$ADDR);
 640    2       =       PC=PC+3;
 641    2       =       IF BIT$DATA=0
                =           THEN PC=PC+SIGN$EXTENDED(DISPLACEMENT);
 643    2       =   END JNB$BIT$REL;
                =
                =
                =   /*  "JNC      rel"  INSTRUCTION:  */
                =
 644    1       =   JNC$REL:  PROCEDURE;
 645    2       =       DISPLACEMENT=FETCH$PROGRAM(PC+1);
 646    2       =       PC=PC+2;
 647    2       =       IF (PSW AND 10000000B) = 0
                =           THEN PC=PC+SIGN$EXTENDED(DISPLACEMENT);
 649    2       =   END JNC$REL;
                =
                =
                =   /*  "JNZ      rel"  INSTRUCTION:  */
                =
 650    1       =   JNZ$REL:  PROCEDURE;
 651    2       =       DISPLACEMENT=FETCH$PROGRAM(PC+1);
 652    2       =       PC=PC+2;
 653    2       =       IF ACC <> 0
                =           THEN PC=PC+SIGN$EXTENDED(DISPLACEMENT);
 655    2       =   END JNZ$REL;
                =
                =
                =   /*  "JZ      rel"  INSTRUCTION:  */
                =
 656    1       =   JZ$REL:  PROCEDURE;
 657    2       =       DISPLACEMENT=FETCH$PROGRAM(PC+1);
 658    2       =       PC=PC+2;
```

```
659   2  =          IF ACC = 0
       =              THEN PC=PC+SIGN$EXTENDED(DISPLACEMENT);
661   2  =     END JZ$REL;
       =
       =
       =     /*  "LCALL    addr16"  INSTRUCTION:   */
       =
662   1  =     LCALL$ADDR16:  PROCEDURE;
663   2  =         PAGE$CODE=FETCH$PROGRAM(PC+1);
664   2  =         PAGE$OFFSET=FETCH$PROGRAM(PC+2);
665   2  =         PC=PC+3;
666   2  =         CALL PUSH$STACK(LOW(PC));
667   2  =         CALL PUSH$STACK(HIGH(PC));
668   2  =         PC=(PAGE$CODE * 100H) + PAGE$OFFSET;
669   2  =     END LCALL$ADDR16;
       =
       =
       =     /*  "LJMP     addr16"  INSTRUCTION:   */
       =
670   1  =     LJMP$ADDR16:  PROCEDURE;
671   2  =         PAGE$CODE=FETCH$PROGRAM(PC+1);
672   2  =         PAGE$OFFSET=FETCH$PROGRAM(PC+2);
673   2  =         PC=(PAGE$CODE * 100H) + PAGE$OFFSET;
674   2  =     END LJMP$ADDR16;
       =
       =
       =     /*  "MOV      A,Rn"  INSTRUCTION:   */
       =
675   1  =     MOV$A$REG:  PROCEDURE;
676   2  =         REG$ADDR=OPCODE AND 00000111B;
677   2  =         ACC=FETCH$REG(REG$ADDR);
678   2  =         PC=PC+1;
679   2  =     END MOV$A$REG;
       =
       =
       =     /*  "MOV      A,direct"  INSTRUCTION:   */
       =
680   1  =     MOV$A$DIR:  PROCEDURE;
681   2  =         DIR$ADDR=FETCH$PROGRAM(PC+1);
682   2  =         ACC=FETCH$DIR(DIR$ADDR);
683   2  =         PC=PC+2;
684   2  =     END MOV$A$DIR;
       =
       =
       =     /*  "MOV      A,@Ri"  INSTRUCTION:   */
       =
685   1  =     MOV$A$IND:  PROCEDURE;
686   2  =         REG$ADDR=OPCODE AND 00000001B;
687   2  =         ACC=FETCH$IND(REG$ADDR);
688   2  =         PC=PC+1;
689   2  =     END MOV$A$IND;
       =
       =
       =     /*  "MOV      A,#data"  INSTRUCTION:   */
       =
690   1  =     MOV$A$IMM:  PROCEDURE;
691   2  =         ACC=FETCH$PROGRAM(PC+1);
```

```
692   2  =        PC=PC+2;
693   2  =    END MOV$A$IMM;
             =
             =
             =    /*  "MOV      Rn,A"  INSTRUCTION:  */
             =
694   1  =    MOV$REG$A:  PROCEDURE;
695   2  =        REG$ADDR=OPCODE AND 00000111B;
696   2  =        CALL STORE$REG(REG$ADDR,ACC);
697   2  =        PC=PC+1;
698   2  =    END MOV$REG$A;
             =
             =
             =    /*  "MOV      Rn,direct"  INSTRUCTION:  */
             =
699   1  =    MOV$REG$DIR:  PROCEDURE;
700   2  =        REG$ADDR=OPCODE AND 00000111B;
701   2  =        DIR$ADDR=FETCH$PROGRAM(PC+1);
702   2  =        DIR$DATA=FETCH$DIR(DIR$ADDR);
703   2  =        CALL STORE$REG(REG$ADDR,DIR$DATA);
704   2  =        PC=PC+2;
705   2  =    END MOV$REG$DIR;
             =
             =
             =    /*  "MOV      Rn,#data"  INSTRUCTION:  */
             =
706   1  =    MOV$REG$IMM:  PROCEDURE;
707   2  =        REG$ADDR=OPCODE AND 00000111B;
708   2  =        IMM$DATA=FETCH$PROGRAM(PC+1);
709   2  =        CALL STORE$REG(REG$ADDR,IMM$DATA);
710   2  =        PC=PC+2;
711   2  =    END MOV$REG$IMM;
             =
             =
             =    /*  "MOV      direct,A"  INSTRUCTION:  */
             =
712   1  =    MOV$DIR$A:  PROCEDURE;
713   2  =        DIR$ADDR=FETCH$PROGRAM(PC+1);
714   2  =        CALL STORE$DIR(DIR$ADDR,ACC);
715   2  =        PC=PC+2;
716   2  =    END MOV$DIR$A;
             =
             =
             =    /*  "MOV      direct,Rn"  INSTRUCTION:  */
             =
717   1  =    MOV$DIR$REG:  PROCEDURE;
718   2  =        REG$ADDR=OPCODE AND 00000111B;
719   2  =        REG$DATA=FETCH$REG(REG$ADDR);
720   2  =        DIR$ADDR=FETCH$PROGRAM(PC+1);
721   2  =        CALL STORE$DIR(DIR$ADDR,REG$DATA);
722   2  =        PC=PC+2;
723   2  =    END MOV$DIR$REG;
             =
             =
             =    /*  "MOV      direct,direct"  INSTRUCTION:  */
             =
724   1  =    MOV$DIR$DIR:  PROCEDURE;
```

```
 725   2  =        SOURCE$ADDR=FETCH$PROGRAM(PC+1);
 726   2  =        DEST$ADDR=FETCH$PROGRAM(PC+2);
 727   2  =        DIR$DATA=FETCH$DIR(SOURCE$ADDR);
 728   2  =        CALL STORE$DIR(DEST$ADDR, DIR$DATA);
 729   2  =        PC=PC+3;
 730   2  =    END MOV$DIR$DIR;
              =
              =
              =    /*  "MOV      direct,@Ri"  INSTRUCTION:   */
              =
 731   1  =    MOV$DIR$IND:  PROCEDURE;
 732   2  =        REG$ADDR=OPCODE AND 00000001B;
 733   2  =        IND$DATA=FETCH$IND(REG$ADDR);
 734   2  =        DIR$ADDR=FETCH$PROGRAM(PC+1);
 735   2  =        CALL STORE$DIR(DIR$ADDR, IND$DATA);
 736   2  =        PC=PC+2;
 737   2  =    END MOV$DIR$IND;
              =
              =
              =    /*  "MOV      direct,#data"  INSTRUCTION:   */
              =
 738   1  =    MOV$DIR$IMM:  PROCEDURE;
 739   2  =        DIR$ADDR=FETCH$PROGRAM(PC+1);
 740   2  =        IMM$DATA=FETCH$PROGRAM(PC+2);
 741   2  =        CALL STORE$DIR(DIR$ADDR, IMM$DATA);
 742   2  =        PC=PC+3;
 743   2  =    END MOV$DIR$IMM;
              =
              =
              =    /*  "MOV      @Ri,A"  INSTRUCTION:   */
              =
 744   1  =    MOV$IND$A:  PROCEDURE;
 745   2  =        REG$ADDR=OPCODE AND 00000001B;
 746   2  =        CALL STORE$IND(REG$ADDR, ACC);
 747   2  =        PC=PC+1;
 748   2  =    END MOV$IND$A;
              =
              =
              =    /*  "MOV      @Ri,direct"  INSTRUCTION:   */
              =
 749   1  =    MOV$IND$DIR:  PROCEDURE;
 750   2  =        DIR$ADDR=FETCH$PROGRAM(PC+1);
 751   2  =        DIR$DATA=FETCH$DIR(DIR$ADDR);
 752   2  =        REG$ADDR=OPCODE AND 00000001B;
 753   2  =        CALL STORE$IND(REG$ADDR, DIR$DATA);
 754   2  =        PC=PC+2;
 755   2  =    END MOV$IND$DIR;
              =
              =
              =    /*  "MOV      @Ri,#data"  INSTRUCTION:   */
              =
 756   1  =    MOV$IND$IMM:  PROCEDURE;
 757   2  =        IMM$DATA=FETCH$PROGRAM(PC+1);
 758   2  =        REG$ADDR=OPCODE AND 00000001B;
 759   2  =        CALL STORE$IND(REG$ADDR, IMM$DATA);
 760   2  =        PC=PC+2;
 761   2  =    END MOV$IND$IMM;
```

```
                =
                =
                =   /*  "MOV      C,bit"  INSTRUCTION:  */
                =
 762    1       =   MOV$C$BIT:  PROCEDURE;
 763    2       =       BIT$ADDR=FETCH$PROGRAM(PC+1);
 764    2       =       BIT$DATA=FETCH$BIT(BIT$ADDR);
 765    2       =       IF BIT$DATA=0
                =           THEN PSW=(PSW AND 01111111B);
 767    2       =           ELSE PSW=(PSW OR  10000000B);
 768    2       =       PC=PC+2;
 769    2       =   END MOV$C$BIT;
                =
                =
                =   /*  "MOV      bit,C"  INSTRUCTION:  */
                =
 770    1       =   MOV$BIT$C:  PROCEDURE;
 771    2       =       BIT$ADDR=FETCH$PROGRAM(PC+1);
 772    2       =       BIT$DATA=((PSW AND 10000000B) / 128);
 773    2       =       CALL STORE$BIT(BIT$ADDR,BIT$DATA);
 774    2       =       PC=PC+2;
 775    2       =   END MOV$BIT$C;
                =
                =
                =   /*  "MOV      DPTR,#data16"  INSTRUCTION:  */
                =
 776    1       =   MOV$DPTR$IMM16:  PROCEDURE;
 777    2       =       DPH=FETCH$PROGRAM(PC+1);
 778    2       =       DPL=FETCH$PROGRAM(PC+2);
 779    2       =       PC=PC+3;
 780    2       =   END MOV$DPTR$IMM16;
                =
                =
                =   /*  "MOVC     A,@A+DPTR"  INSTRUCTION:  */
                =
 781    1       =   MOVC$A$ADPTR:  PROCEDURE;
 782    2       =       CODE$ADDR=(DPH*256)+DPL+ACC;
 783    2       =       ACC=FETCH$PROGRAM(CODE$ADDR);
 784    2       =       PC=PC+1;
 785    2       =   END MOVC$A$ADPTR;
                =
                =
                =   /*  "MOVC     A,@A+PC"  INSTRUCTION:  */
                =
 786    1       =   MOVC$A$APC:  PROCEDURE;
 787    2       =       PC=PC+1;
 788    2       =       CODE$ADDR=(PC+ACC);
 789    2       =       ACC=FETCH$PROGRAM(CODE$ADDR);
 790    2       =   END MOVC$A$APC;
                =
                =
                =   /*  "MOVX     A,@Ri"  INSTRUCTION:  */
                =
 791    1       =   MOVX$A$IND:  PROCEDURE;
 792    2       =       REG$ADDR=OPCODE AND 00000001B;
 793    2       =       PAGED$EXTERNAL$ADDR=FETCH$REG(REG$ADDR);
 794    2       =       ACC=FETCH$PAGED$EXTERNAL(PAGED$EXTERNAL$ADDR);
```

```
795   2  =        PC=PC+1;
796   2  =    END MOVX$A$IND;
             =
             =
             =    /*  "MOVX     @Ri,A"  INSTRUCTION:   */
             =
797   1  =    MOVX$IND$A:  PROCEDURE;
798   2  =        REG$ADDR=OPCODE AND 00000001B;
799   2  =        PAGED$EXTERNAL$ADDR=FETCH$REG(REG$ADDR);
800   2  =        CALL STORE$PAGED$EXTERNAL(PAGED$EXTERNAL$ADDR,ACC);
801   2  =        PC=PC+1;
802   2  =    END MOVX$IND$A;
             =
             =
             =    /*  "MOVX     A,@DPTR"  INSTRUCTION:   */
             =
803   1  =    MOVX$A$DPTR:  PROCEDURE;
804   2  =        ACC=FETCH$LONG$EXTERNAL((DPH*256)+DPL);
805   2  =        PC=PC+1;
806   2  =    END MOVX$A$DPTR;
             =
             =
             =    /*  "MOVX     @DPTR,A"  INSTRUCTION:   */
             =
807   1  =    MOVX$DPTR$A:  PROCEDURE;
808   2  =        CALL STORE$LONG$EXTERNAL((DPH*256)+DPL,ACC);
809   2  =        PC=PC+1;
810   2  =    END MOVX$DPTR$A;
             =
             =
             =    /*  "MUL      AB"  INSTRUCTION:   */
             =
811   1  =    MUL$AB:  PROCEDURE;
812   2  =        MUL$TEMP=ACC * B;
813   2  =        B=HIGH(MUL$TEMP);
814   2  =        ACC=LOW(MUL$TEMP);
815   2  =        PSW=PSW AND 01111111B;
816   2  =        IF B = 0
             =            THEN PSW=PSW AND 11111011B;
818   2  =            ELSE PSW=PSW OR  00000100B;
819   2  =        PC=PC+1;
820   2  =    END MUL$AB;
             =
             =
             =    /*  "NOP "  INSTRUCTION:   */
             =
821   1  =    NOP:  PROCEDURE;
822   2  =        PC=PC+1;
823   2  =    END NOP;
             =
             =
             =    /*  "ORL      A,Rn"  INSTRUCTION:   */
             =
824   1  =    ORL$A$REG:  PROCEDURE;
825   2  =        REG$ADDR=OPCODE AND 00000111B;
826   2  =        REG$DATA=FETCH$REG(REG$ADDR);
827   2  =        ACC=ACC OR REG$DATA;
```

```
828   2  =        PC=PC+1;
829   2  =    END ORL$A$REG;
           =
           =
           =    /*  "ORL      A,direct"  INSTRUCTION:  */
           =
830   1  =    ORL$A$DIR:  PROCEDURE;
831   2  =        DIR$ADDR=FETCH$PROGRAM(PC+1);
832   2  =        DIR$DATA=FETCH$DIR(DIR$ADDR);
833   2  =        ACC=ACC OR DIR$DATA;
834   2  =        PC=PC+2;
835   2  =    END ORL$A$DIR;
           =
           =
           =    /*  "ORL      A,@Ri"  INSTRUCTION:  */
           =
836   1  =    ORL$A$IND:  PROCEDURE;
837   2  =        REG$ADDR=OPCODE AND 00000001B;
838   2  =        IND$DATA=FETCH$IND(REG$ADDR);
839   2  =        ACC=ACC OR IND$DATA;
840   2  =        PC=PC+1;
841   2  =    END ORL$A$IND;
           =
           =
           =    /*  "ORL      A,#data"  INSTRUCTION:  */
           =
842   1  =    ORL$A$IMM:  PROCEDURE;
843   2  =        IMM$DATA=FETCH$PROGRAM(PC+1);
844   2  =        ACC=ACC OR IMM$DATA;
845   2  =        PC=PC+2;
846   2  =    END ORL$A$IMM;
           =
           =
           =    /*  "ORL      direct,A"  INSTRUCTION:  */
           =
847   1  =    ORL$DIR$A:  PROCEDURE;
848   2  =        DIR$ADDR=FETCH$PROGRAM(PC+1);
849   2  =        DIR$DATA=FETCH$DIR$INT(DIR$ADDR);
850   2  =        CALL STORE$DIR(DIR$ADDR,ACC OR DIR$DATA);
851   2  =        PC=PC+2;
852   2  =    END ORL$DIR$A;
           =
           =
           =    /*  "ORL      direct,#data"  INSTRUCTION:  */
           =
853   1  =    ORL$DIR$IMM:  PROCEDURE;
854   2  =        DIR$ADDR=FETCH$PROGRAM(PC+1);
855   2  =        IMM$DATA=FETCH$PROGRAM(PC+2);
856   2  =        DIR$DATA=FETCH$DIR$INT(DIR$ADDR);
857   2  =        CALL STORE$DIR(DIR$ADDR,IMM$DATA OR DIR$DATA);
858   2  =        PC=PC+3;
859   2  =    END ORL$DIR$IMM;
           =
           =
           =    /*  "ORL      C,bit"  INSTRUCTION:  */
           =
860   1  =    ORL$C$BIT:  PROCEDURE;
```

```
861   2  =        BIT$ADDR=FETCH$PROGRAM(PC+1);
862   2  =        BIT$DATA=FETCH$BIT(BIT$ADDR);
863   2  =        IF BIT$DATA = 1 THEN PSW=PSW OR 10000000B;
865   2  =        PC=PC+2;
866   2  =    END ORL$C$BIT;
              =
              =
              =    /*  "ORL      C,/bit"  INSTRUCTION:   */
              =
867   1  =    ORL$C$COMP$BIT:  PROCEDURE;
868   2  =        BIT$ADDR=FETCH$PROGRAM(PC+1);
869   2  =        BIT$DATA=FETCH$BIT(BIT$ADDR);
870   2  =        IF BIT$DATA = 0 THEN PSW=PSW OR 10000000B;
872   2  =        PC=PC+2;
873   2  =    END ORL$C$COMP$BIT;
              =
              =
              =    /*  "POP      direct"  INSTRUCTION:   */
              =
874   1  =    POP$DIR:  PROCEDURE;
875   2  =        DIR$ADDR=FETCH$PROGRAM(PC+1);
876   2  =        STACK$DATA=POP$STACK;
877   2  =        CALL STORE$DIR(DIR$ADDR,STACK$DATA);
878   2  =        PC=PC+2;
879   2  =    END POP$DIR;
              =
              =
              =    /*  "PUSH     direct"  INSTRUCTION:   */
              =
880   1  =    PUSH$DIR:  PROCEDURE;
881   2  =        DIR$ADDR=FETCH$PROGRAM(PC+1);
882   2  =        DIR$DATA=FETCH$DIR(DIR$ADDR);
883   2  =        CALL PUSH$STACK(DIR$DATA);
884   2  =        PC=PC+2;
885   2  =    END PUSH$DIR;
              =
              =
              =    /*  "RET "  INSTRUCTION:   */
              =
886   1  =    RET:  PROCEDURE;
887   2  =        PAGE$CODE=POP$STACK;
888   2  =        PAGE$OFFSET=POP$STACK;
889   2  =        PC=(PAGE$CODE * 100H) + PAGE$OFFSET;
890   2  =    END RET;
              =
              =
              =    /*  "RETI"  INSTRUCTION:   */
              =
891   1  =    RETI:  PROCEDURE;
892   2  =        PAGE$CODE=POP$STACK;
893   2  =        PAGE$OFFSET=POP$STACK;
894   2  =        PC=(PAGE$CODE * 100H) + PAGE$OFFSET;
              =
              =    /*  RESTORE INTERRUPT SYSTEM TO LEVEL IN EFFECT
              =        BEFORE LAST INTERRUPT RECEIVED  */
              =
895   2  =    END RETI;
```

```
                =
                =
                =   /*  "RL      A"  INSTRUCTION:  */
                =
 896    1       =   RL$A:  PROCEDURE;
 897    2       =       LINK$BIT=(ACC AND 10000000B) / 128;
 898    2       =       ACC=(ACC * 2) + LINK$BIT;
 899    2       =       PC=PC+1;
 900    2       =   END RL$A;
                =
                =
                =   /*  "RLC     A"  INSTRUCTION:  */
                =
 901    1       =   RLC$A:  PROCEDURE;
 902    2       =       LINK$BIT=(ACC AND 10000000B) / 128;
 903    2       =       ACC=(ACC * 2) + ((PSW AND 10000000B) / 128);
 904    2       =       PSW=(PSW AND 01111111B) + (LINK$BIT * 128);
 905    2       =       PC=PC+1;
 906    2       =   END RLC$A;
                =
                =
                =   /*  "RR      A"  INSTRUCTION:  */
                =
 907    1       =   RR$A:  PROCEDURE;
 908    2       =       LINK$BIT=ACC AND 00000001B;
 909    2       =       ACC=(ACC / 2) + (LINK$BIT * 128);
 910    2       =       PC=PC+1;
 911    2       =   END RR$A;
                =
                =
                =   /*  "RRC     A"  INSTRUCTION:  */
                =
 912    1       =   RRC$A:  PROCEDURE;
 913    2       =       LINK$BIT=ACC AND 00000001B;
 914    2       =       ACC=(ACC / 2) + (PSW AND 10000000B);
 915    2       =       PSW=(PSW AND 01111111B)+(LINK$BIT * 128);
 916    2       =       PC=PC+1;
 917    2       =   END RRC$A;
                =
                =
                =   /*  "SETB    C"  INSTRUCTION:  */
                =
 918    1       =   SETB$C:  PROCEDURE;
 919    2       =       PSW=PSW OR 10000000B;
 920    2       =       PC=PC+1;
 921    2       =   END SETB$C;
                =
                =
                =   /*  "SETB    bit"  INSTRUCTION:  */
                =
 922    1       =   SETB$BIT:  PROCEDURE;
 923    2       =       BIT$ADDR=FETCH$PROGRAM(PC+1);
 924    2       =       CALL STORE$BIT(BIT$ADDR,1);
 925    2       =       PC=PC+2;
 926    2       =   END SETB$BIT;
                =
                =
```

```
                =   /*  "SJMP    rel"  INSTRUCTION:  */
                =
 927    1       =   SJMP$REL:  PROCEDURE;
 928    2       =       DISPLACEMENT=FETCH$PROGRAM(PC+1);
 929    2       =       PC=PC+2;
 930    2       =       PC=PC+SIGN$EXTENDED(DISPLACEMENT);
 931    2       =   END SJMP$REL;
                =
                =
                =   /*  "SUBB    A,<src-byte>"  FUNCTION:  */
                =
 932    1       =   SUBB$A:  PROCEDURE(DATA$BYTE);
 933    2       =       DECLARE DATA$BYTE BYTE;
 934    2       =       LINK$BIT=(PSW AND 10000000B) / 128;
 935    2       =       SUB$TEMP=DATA$BYTE;
 936    2       =       SUB$TEMP=SUB$TEMP+LINK$BIT;
 937    2       =       IF (ACC AND 0FH) < (SUB$TEMP AND 0FH)
                =           THEN PSW=PSW OR  01000000B;
 939    2       =           ELSE PSW=PSW AND 10111111B;
 940    2       =       IF (ACC AND 7FH) < (SUB$TEMP AND 7FH)
                =           THEN PSW=PSW OR  00000100B;
 942    2       =           ELSE PSW=PSW AND 11111011B;
 943    2       =       IF ACC < SUB$TEMP
                =           THEN PSW=(PSW OR  10000000B) XOR 00000100B;
 945    2       =           ELSE PSW=(PSW AND 01111111B);
 946    2       =       ACC=ACC-SUB$TEMP;
 947    2       =   END SUBB$A;
                =
                =
                =   /*  "SUBB    A,Rn"  INSTRUCTION:  */
                =
 948    1       =   SUBB$A$REG:  PROCEDURE;
 949    2       =       REG$ADDR=OPCODE AND 00000111B;
 950    2       =       REG$DATA=FETCH$REG(REG$ADDR);
 951    2       =       CALL SUBB$A(REG$DATA);
 952    2       =       PC=PC+1;
 953    2       =   END SUBB$A$REG;
                =
                =
                =   /*  "SUBB    A,direct"  INSTRUCTION:  */
                =
 954    1       =   SUBB$A$DIR:  PROCEDURE;
 955    2       =       DIR$ADDR=FETCH$PROGRAM(PC+1);
 956    2       =       DIR$DATA=FETCH$DIR(DIR$ADDR);
 957    2       =       CALL SUBB$A(DIR$DATA);
 958    2       =       PC=PC+2;
 959    2       =   END SUBB$A$DIR;
                =
                =
                =   /*  "SUBB    A,@Ri"  INSTRUCTION:  */
                =
 960    1       =   SUBB$A$IND:  PROCEDURE;
 961    2       =       REG$ADDR=OPCODE AND 00000001B;
 962    2       =       IND$DATA=FETCH$IND(REG$ADDR);
 963    2       =       CALL SUBB$A(IND$DATA);
 964    2       =       PC=PC+1;
 965    2       =   END SUBB$A$IND;
```

```
                =
                =
                =   /*  "SUBB     A,#data"  INSTRUCTION:   */
                =
 966    1       =   SUBB$A$IMM:  PROCEDURE;
 967    2       =       IMM$DATA=FETCH$PROGRAM(PC+1);
 968    2       =       CALL SUBB$A(IMM$DATA);
 969    2       =       PC=PC+2;
 970    2       =   END SUBB$A$IMM;
                =
                =
                =   /*  "SWAP     A"  INSTRUCTION:   */
                =
 971    1       =   SWAP$A:  PROCEDURE;
 972    2       =       LOW$NIB=ACC AND 00001111B;
 973    2       =       HIGH$NIB=ACC AND 11110000B;
 974    2       =       ACC=(LOW$NIB * 16) + (HIGH$NIB / 16);
 975    2       =       PC=PC+1;
 976    2       =   END SWAP$A;
                =
                =
                =   /*  "XCH      A,Rn"  INSTRUCTION:   */
                =
 977    1       =   XCH$A$REG:  PROCEDURE;
 978    2       =       REG$ADDR=OPCODE AND 00000111B;
 979    2       =       REG$DATA=FETCH$REG(REG$ADDR);
 980    2       =       CALL STORE$REG(REG$ADDR,ACC);
 981    2       =       ACC=REG$DATA;
 982    2       =       PC=PC+1;
 983    2       =   END XCH$A$REG;
                =
                =
                =   /*  "XCH      A,direct"  INSTRUCTION:   */
                =
 984    1       =   XCH$A$DIR:  PROCEDURE;
 985    2       =       DIR$ADDR=FETCH$PROGRAM(PC+1);
 986    2       =       DIR$DATA=FETCH$DIR(DIR$ADDR);
 987    2       =       CALL STORE$DIR(DIR$ADDR,ACC);
 988    2       =       ACC=DIR$DATA;
 989    2       =       PC=PC+2;
 990    2       =   END XCH$A$DIR;
                =
                =
                =   /*  "XCH      A,@Ri"  INSTRUCTION:   */
                =
 991    1       =   XCH$A$IND:  PROCEDURE;
 992    2       =       REG$ADDR=OPCODE AND 00000001B;
 993    2       =       IND$DATA=FETCH$IND(REG$ADDR);
 994    2       =       CALL STORE$IND(REG$ADDR,ACC);
 995    2       =       ACC=IND$DATA;
 996    2       =       PC=PC+1;
 997    2       =   END XCH$A$IND;
                =
                =
                =   /*  "XCHD     A,@Ri"  INSTRUCTION:   */
                =
 998    1       =   XCHD$A$IND:  PROCEDURE;
```

```
 999   2  =          REG$ADDR=OPCODE AND 00000001B;
1000   2  =          IND$DATA=FETCH$IND(REG$ADDR);
1001   2  =          LOW$SOURCE$NIB=IND$DATA AND 00001111B;
1002   2  =          IND$DATA=(IND$DATA AND 11110000B) + (ACC AND 00001111B);
1003   2  =          CALL STORE$IND(REG$ADDR, IND$DATA);
1004   2  =          ACC=(ACC AND 11110000B) + LOW$SOURCE$NIB;
1005   2  =          PC=PC+1;
1006   2  =      END XCHD$A$IND;
               =
               =
               =  /*  "XRL      A,Rn"  INSTRUCTION:   */
               =
1007   1  =      XRL$A$REG:  PROCEDURE;
1008   2  =          REG$ADDR=OPCODE AND 00000111B;
1009   2  =          REG$DATA=FETCH$REG(REG$ADDR);
1010   2  =          ACC=ACC XOR REG$DATA;
1011   2  =          PC=PC+1;
1012   2  =      END XRL$A$REG;
               =
               =
               =  /*  "XRL      A,direct"  INSTRUCTION:   */
               =
1013   1  =      XRL$A$DIR:  PROCEDURE;
1014   2  =          DIR$ADDR=FETCH$PROGRAM(PC+1);
1015   2  =          DIR$DATA=FETCH$DIR(DIR$ADDR);
1016   2  =          ACC=ACC XOR DIR$DATA;
1017   2  =          PC=PC+2;
1018   2  =      END XRL$A$DIR;
               =
               =
               =  /*  "XRL      A,@Ri"  INSTRUCTION:   */
               =
1019   1  =      XRL$A$IND:  PROCEDURE;
1020   2  =          REG$ADDR=OPCODE AND 00000001B;
1021   2  =          IND$DATA=FETCH$IND(REG$ADDR);
1022   2  =          ACC=ACC XOR IND$DATA;
1023   2  =          PC=PC+1;
1024   2  =      END XRL$A$IND;
               =
               =
               =  /*  "XRL      A,#data"  INSTRUCTION:   */
               =
1025   1  =      XRL$A$IMM:  PROCEDURE;
1026   2  =          IMM$DATA=FETCH$PROGRAM(PC+1);
1027   2  =          ACC=ACC XOR IMM$DATA;
1028   2  =          PC=PC+2;
1029   2  =      END XRL$A$IMM;
               =
               =
               =  /*  "XRL      direct,A"  INSTRUCTION:   */
               =
1030   1  =      XRL$DIR$A:  PROCEDURE;
1031   2  =          DIR$ADDR=FETCH$PROGRAM(PC+1);
1032   2  =          DIR$DATA=FETCH$DIR$INT(DIR$ADDR);
1033   2  =          CALL STORE$DIR(DIR$ADDR, ACC XOR DIR$DATA);
1034   2  =          PC=PC+2;
1035   2  =      END XRL$DIR$A;
```

```
                =
                =
                =   /*  "XRL      direct,#data"  INSTRUCTION:   */
                =
1036    1       =   XRL$DIR$IMM:  PROCEDURE;
1037    2       =       DIR$ADDR=FETCH$PROGRAM(PC+1);
1038    2       =       IMM$DATA=FETCH$PROGRAM(PC+2);
1039    2       =       DIR$DATA=FETCH$DIR$INT(DIR$ADDR);
1040    2       =       CALL STORE$DIR(DIR$ADDR, IMM$DATA XOR DIR$DATA);
1041    2       =       PC=PC+3;
1042    2       =   END XRL$DIR$IMM;
                =
```

```
                $EJECT

1043    1       INITIALIZE: PROCEDURE PUBLIC;
1044    2           PC=0000H;
1045    2           P0=0FFH;
1046    2           P1=0FFH;
1047    2           P2=0FFH;
1048    2           P3=0FFH;
1049    2           PSW=00H;
1050    2           SP=07H;
1051    2           DPL=00H;
1052    2           DPH=00H;
1053    2           ACC=00H;
1054    2           B=00H;
1055    2           TL0=00H;
1056    2           TH0=00H;
1057    2           TL1=00H;
1058    2           TH1=00H;
1059    2           TCON=00H;
1060    2           TMOD=00H;
1061    2           SCON=00H;
1062    2           IE=00H;
1063    2           IP=00H;
1064    2           MACH$CYC=0000H;   /*  SIMULATION ELAPSED TIME REGISTER.  */
1065    2       END INITIALIZE;


1066    1       DECLARE EXECUTION$TIME(256) BYTE DATA
                    (1, 2, 2, 1, 1, 1, 1, 1,    1, 1, 1, 1, 1, 1, 1, 1,
                     2, 2, 2, 1, 1, 1, 1, 1,    1, 1, 1, 1, 1, 1, 1, 1,
                     2, 2, 2, 1, 1, 1, 1, 1,    1, 1, 1, 1, 1, 1, 1, 1,
                     2, 2, 2, 1, 1, 1, 1, 1,    1, 1, 1, 1, 1, 1, 1, 1,
                     2, 2, 1, 2, 1, 1, 1, 1,    1, 1, 1, 1, 1, 1, 1, 1,
                     2, 2, 1, 2, 1, 1, 1, 1,    1, 1, 1, 1, 1, 1, 1, 1,
                     2, 2, 1, 2, 1, 1, 1, 1,    1, 1, 1, 1, 1, 1, 1, 1,
                     2, 2, 2, 2, 1, 2, 1, 1,    1, 1, 1, 1, 1, 1, 1, 1,
                     2, 2, 2, 2, 4, 2, 2, 2,    2, 2, 2, 2, 2, 2, 2, 2,
                     2, 2, 2, 2, 1, 1, 1, 1,    1, 1, 1, 1, 1, 1, 1, 1,
                     2, 2, 1, 2, 4, 1, 2, 2,    2, 2, 2, 2, 2, 2, 2, 2,
                     2, 2, 1, 1, 2, 2, 2, 2,    2, 2, 2, 2, 2, 2, 2, 2,
                     2, 2, 1, 1, 1, 1, 1, 1,    1, 1, 1, 1, 1, 1, 1, 1,
                     2, 2, 1, 1, 1, 2, 1, 1,    2, 2, 2, 2, 2, 2, 2, 2,
                     2, 2, 2, 2, 1, 1, 1, 1,    1, 1, 1, 1, 1, 1, 1, 1,
                     2, 2, 2, 2, 1, 1, 1, 1,    1, 1, 1, 1, 1, 1, 1, 1);
```

```
                $EJECT

1067    1       STEP:  PROCEDURE (NEXT$INSTRUCTION) ADDRESS PUBLIC;
1068    2           DECLARE NEXT$INSTRUCTION ADDRESS;
1069    2           PC=NEXT$INSTRUCTION;
1070    2           OPCODE=USER$CODE(PC);
1071    2           DO CASE OPCODE;

                /*  INSTRUCTIONS CORRESPONDING TO ROW 0 OF OPCODE MAP
                    (FORM 0XH):  */
1072    3               CALL NOP;
1073    3               CALL AJMP$ADDR11;
1074    3               CALL LJMP$ADDR16;
1075    3               CALL RR$A;
1076    3               CALL INC$A;
1077    3               CALL INC$DIR;
1078    3               CALL INC$IND;
1079    3               CALL INC$IND;
1080    3               CALL INC$REG;
1081    3               CALL INC$REG;
1082    3               CALL INC$REG;
1083    3               CALL INC$REG;
1084    3               CALL INC$REG;
1085    3               CALL INC$REG;
1086    3               CALL INC$REG;
1087    3               CALL INC$REG;


                /*  INSTRUCTIONS CORRESPONDING TO ROW 1 OF OPCODE MAP
                    (FORM 1XH):  */
1088    3               CALL JBC$BIT$REL;
1089    3               CALL ACALL$ADDR11;
1090    3               CALL LCALL$ADDR16;
1091    3               CALL RRC$A;
1092    3               CALL DEC$A;
1093    3               CALL DEC$DIR;
1094    3               CALL DEC$IND;
1095    3               CALL DEC$IND;
1096    3               CALL DEC$REG;
1097    3               CALL DEC$REG;
1098    3               CALL DEC$REG;
1099    3               CALL DEC$REG;
1100    3               CALL DEC$REG;
1101    3               CALL DEC$REG;
1102    3               CALL DEC$REG;
1103    3               CALL DEC$REG;
```

```
                $EJECT

                /*  INSTRUCTIONS CORRESPONDING TO ROW 2 OF OPCODE MAP
                    (FORM 2XH):  */
1104    3               CALL JB$BIT$REL;
1105    3               CALL AJMP$ADDR11;
1106    3               CALL RET;
1107    3               CALL RL$A;
1108    3               CALL ADD$A$IMM;
1109    3               CALL ADD$A$DIR;
1110    3               CALL ADD$A$IND;
1111    3               CALL ADD$A$IND;
1112    3               CALL ADD$A$REG;
1113    3               CALL ADD$A$REG;
1114    3               CALL ADD$A$REG;
1115    3               CALL ADD$A$REG;
1116    3               CALL ADD$A$REG;
1117    3               CALL ADD$A$REG;
1118    3               CALL ADD$A$REG;
1119    3               CALL ADD$A$REG;

                /*  INSTRUCTIONS CORRESPONDING TO ROW 3 OF OPCODE MAP
                    (FORM 3XH):  */
1120    3               CALL JNB$BIT$REL;
1121    3               CALL ACALL$ADDR11;
1122    3               CALL RETI;
1123    3               CALL RLC$A;
1124    3               CALL ADDC$A$IMM;
1125    3               CALL ADDC$A$DIR;
1126    3               CALL ADDC$A$IND;
1127    3               CALL ADDC$A$IND;
1128    3               CALL ADDC$A$REG;
1129    3               CALL ADDC$A$REG;
1130    3               CALL ADDC$A$REG;
1131    3               CALL ADDC$A$REG;
1132    3               CALL ADDC$A$REG;
1133    3               CALL ADDC$A$REG;
1134    3               CALL ADDC$A$REG;
1135    3               CALL ADDC$A$REG;
```

```
                $EJECT

                /*  INSTRUCTIONS CORRESPONDING TO ROW 4 OF OPCODE MAP
                    (FORM 4XH):  */
1136    3               CALL JC$REL;
1137    3               CALL AJMP$ADDR11;
1138    3               CALL ORL$DIR$A;
1139    3               CALL ORL$DIR$IMM;
1140    3               CALL ORL$A$IMM;
1141    3               CALL ORL$A$DIR;
1142    3               CALL ORL$A$IND;
1143    3               CALL ORL$A$IND;
1144    3               CALL ORL$A$REG;
1145    3               CALL ORL$A$REG;
1146    3               CALL ORL$A$REG;
1147    3               CALL ORL$A$REG;
1148    3               CALL ORL$A$REG;
1149    3               CALL ORL$A$REG;
1150    3               CALL ORL$A$REG;
1151    3               CALL ORL$A$REG;

                /*  INSTRUCTIONS CORRESPONDING TO ROW 5 OF OPCODE MAP
                    (FORM 5XH):  */
1152    3               CALL JNC$REL;
1153    3               CALL ACALL$ADDR11;
1154    3               CALL ANL$DIR$A;
1155    3               CALL ANL$DIR$IMM;
1156    3               CALL ANL$A$IMM;
1157    3               CALL ANL$A$DIR;
1158    3               CALL ANL$A$IND;
1159    3               CALL ANL$A$IND;
1160    3               CALL ANL$A$REG;
1161    3               CALL ANL$A$REG;
1162    3               CALL ANL$A$REG;
1163    3               CALL ANL$A$REG;
1164    3               CALL ANL$A$REG;
1165    3               CALL ANL$A$REG;
1166    3               CALL ANL$A$REG;
1167    3               CALL ANL$A$REG;
```

```
                $EJECT


                /*  INSTRUCTIONS CORRESPONDING TO ROW 6 OF OPCODE MAP
                    (FORM 6XH):  */
1168    3               CALL JZ$REL;
1169    3               CALL AJMP$ADDR11;
1170    3               CALL XRL$DIR$A;
1171    3               CALL XRL$DIR$IMM;
1172    3               CALL XRL$A$IMM;
1173    3               CALL XRL$A$DIR;
1174    3               CALL XRL$A$IND;
1175    3               CALL XRL$A$IND;
1176    3               CALL XRL$A$REG;
1177    3               CALL XRL$A$REG;
1178    3               CALL XRL$A$REG;
1179    3               CALL XRL$A$REG;
1180    3               CALL XRL$A$REG;
1181    3               CALL XRL$A$REG;
1182    3               CALL XRL$A$REG;
1183    3               CALL XRL$A$REG;


                /*  INSTRUCTIONS CORRESPONDING TO ROW 7 OF OPCODE MAP
                    (FORM 7XH):  */
1184    3               CALL JNZ$REL;
1185    3               CALL ACALL$ADDR11;
1186    3               CALL ORL$C$BIT;
1187    3               CALL JMP$ADPTR;
1188    3               CALL MOV$A$IMM;
1189    3               CALL MOV$DIR$IMM;
1190    3               CALL MOV$IND$IMM;
1191    3               CALL MOV$IND$IMM;
1192    3               CALL MOV$REG$IMM;
1193    3               CALL MOV$REG$IMM;
1194    3               CALL MOV$REG$IMM;
1195    3               CALL MOV$REG$IMM;
1196    3               CALL MOV$REG$IMM;
1197    3               CALL MOV$REG$IMM;
1198    3               CALL MOV$REG$IMM;
1199    3               CALL MOV$REG$IMM;
```

```
                $EJECT


                /*  INSTRUCTIONS CORRESPONDING TO ROW 8 OF OPCODE MAP
                    (FORM 8XH):  */
1200    3               CALL SJMP$REL;
1201    3               CALL AJMP$ADDR11;
1202    3               CALL ANL$C$BIT;
1203    3               CALL MOVC$A$APC;
1204    3               CALL DIV$AB;
1205    3               CALL MOV$DIR$DIR;
1206    3               CALL MOV$DIR$IND;
1207    3               CALL MOV$DIR$IND;
1208    3               CALL MOV$DIR$REG;
1209    3               CALL MOV$DIR$REG;
1210    3               CALL MOV$DIR$REG;
1211    3               CALL MOV$DIR$REG;
1212    3               CALL MOV$DIR$REG;
1213    3               CALL MOV$DIR$REG;
1214    3               CALL MOV$DIR$REG;
1215    3               CALL MOV$DIR$REG;


                /*  INSTRUCTIONS CORRESPONDING TO ROW 9 OF OPCODE MAP
                    (FORM 9XH):  */
1216    3               CALL MOV$DPTR$IMM16;
1217    3               CALL ACALL$ADDR11;
1218    3               CALL MOV$BIT$C;
1219    3               CALL MOVC$A$ADPTR;
1220    3               CALL SUBB$A$IMM;
1221    3               CALL SUBB$A$DIR;
1222    3               CALL SUBB$A$IND;
1223    3               CALL SUBB$A$IND;
1224    3               CALL SUBB$A$REG;
1225    3               CALL SUBB$A$REG;
1226    3               CALL SUBB$A$REG;
1227    3               CALL SUBB$A$REG;
1228    3               CALL SUBB$A$REG;
1229    3               CALL SUBB$A$REG;
1230    3               CALL SUBB$A$REG;
1231    3               CALL SUBB$A$REG;
```

```
                $EJECT

                /*  INSTRUCTIONS CORRESPONDING TO ROW A OF OPCODE MAP
                    (FORM AXH):  */
1232    3               CALL ORL$C$COMP$BIT;
1233    3               CALL AJMP$ADDR11;
1234    3               CALL MOV$C$BIT;
1235    3               CALL INC$DPTR;
1236    3               CALL MUL$AB;
1237    3               CALL NOP;
1238    3               CALL MOV$IND$DIR;
1239    3               CALL MOV$IND$DIR;
1240    3               CALL MOV$REG$DIR;
1241    3               CALL MOV$REG$DIR;
1242    3               CALL MOV$REG$DIR;
1243    3               CALL MOV$REG$DIR;
1244    3               CALL MOV$REG$DIR;
1245    3               CALL MOV$REG$DIR;
1246    3               CALL MOV$REG$DIR;
1247    3               CALL MOV$REG$DIR;

                /*  INSTRUCTIONS CORRESPONDING TO ROW B OF OPCODE MAP
                    (FORM BXH):  */
1248    3               CALL ANL$C$COMP$BIT;
1249    3               CALL ACALL$ADDR11;
1250    3               CALL CPL$BIT;
1251    3               CALL CPL$C;
1252    3               CALL CJNE$A$IMM$REL;
1253    3               CALL CJNE$A$DIR$REL;
1254    3               CALL CJNE$IND$IMM$REL;
1255    3               CALL CJNE$IND$IMM$REL;
1256    3               CALL CJNE$REG$IMM$REL;
1257    3               CALL CJNE$REG$IMM$REL;
1258    3               CALL CJNE$REG$IMM$REL;
1259    3               CALL CJNE$REG$IMM$REL;
1260    3               CALL CJNE$REG$IMM$REL;
1261    3               CALL CJNE$REG$IMM$REL;
1262    3               CALL CJNE$REG$IMM$REL;
1263    3               CALL CJNE$REG$IMM$REL;
```

```
                $EJECT

                /*  INSTRUCTIONS CORRESPONDING TO ROW C OF OPCODE MAP
                    (FORM CXH):  */
1264    3               CALL PUSH$DIR;
1265    3               CALL AJMP$ADDR11;
1266    3               CALL CLR$BIT;
1267    3               CALL CLR$C;
1268    3               CALL SWAP$A;
1269    3               CALL XCH$A$DIR;
1270    3               CALL XCH$A$IND;
1271    3               CALL XCH$A$IND;
1272    3               CALL XCH$A$REG;
1273    3               CALL XCH$A$REG;
1274    3               CALL XCH$A$REG;
1275    3               CALL XCH$A$REG;
1276    3               CALL XCH$A$REG;
1277    3               CALL XCH$A$REG;
1278    3               CALL XCH$A$REG;
1279    3               CALL XCH$A$REG;

                /*  INSTRUCTIONS CORRESPONDING TO ROW D OF OPCODE MAP
                    (FORM DXH):  */
1280    3               CALL POP$DIR;
1281    3               CALL ACALL$ADDR11;
1282    3               CALL SETB$BIT;
1283    3               CALL SETB$C;
1284    3               CALL DA$A;
1285    3               CALL DJNZ$DIR$REL;
1286    3               CALL XCHD$A$IND;
1287    3               CALL XCHD$A$IND;
1288    3               CALL DJNZ$REG$REL;
1289    3               CALL DJNZ$REG$REL;
1290    3               CALL DJNZ$REG$REL;
1291    3               CALL DJNZ$REG$REL;
1292    3               CALL DJNZ$REG$REL;
1293    3               CALL DJNZ$REG$REL;
1294    3               CALL DJNZ$REG$REL;
1295    3               CALL DJNZ$REG$REL;
```

```
                $EJECT


                /*  INSTRUCTIONS CORRESPONDING TO ROW E OF OPCODE MAP
                    (FORM EXH):  */
1296    3               CALL MOVX$A$DPTR;
1297    3               CALL AJMP$ADDR11;
1298    3               CALL MOVX$A$IND;
1299    3               CALL MOVX$A$IND;
1300    3               CALL CLR$A;
1301    3               CALL MOV$A$DIR;
1302    3               CALL MOV$A$IND;
1303    3               CALL MOV$A$IND;
1304    3               CALL MOV$A$REG;
1305    3               CALL MOV$A$REG;
1306    3               CALL MOV$A$REG;
1307    3               CALL MOV$A$REG;
1308    3               CALL MOV$A$REG;
1309    3               CALL MOV$A$REG;
1310    3               CALL MOV$A$REG;
1311    3               CALL MOV$A$REG;


                /*  INSTRUCTIONS CORRESPONDING TO ROW F OF OPCODE MAP
                    (FORM FXH):  */
1312    3               CALL MOVX$DPTR$A;
1313    3               CALL ACALL$ADDR11;
1314    3               CALL MOVX$IND$A;
1315    3               CALL MOVX$IND$A;
1316    3               CALL CPL$A;
1317    3               CALL MOV$DIR$A;
1318    3               CALL MOV$IND$A;
1319    3               CALL MOV$IND$A;
1320    3               CALL MOV$REG$A;
1321    3               CALL MOV$REG$A;
1322    3               CALL MOV$REG$A;
1323    3               CALL MOV$REG$A;
1324    3               CALL MOV$REG$A;
1325    3               CALL MOV$REG$A;
1326    3               CALL MOV$REG$A;
1327    3               CALL MOV$REG$A;

1328    3               END;

1329    2           PSW=(PSW AND 11111110B) + PARITY$STATE(ACC);
1330    2           MACH$CYC=MACH$CYC + EXECUTION$TIME(OPCODE);
1331    2           RETURN PC;
1332    2       END STEP;


1333    1       END SIM51;
```

U.S. AND CANADIAN SALES OFFICES

3065 Bowers Avenue
Santa Clara, California 95051
Tel: (408) 987-8080
TWX: 910-338-0026
TELEX: 34-6372

ALABAMA

Intel Corp.
303 Williams Avenue, S.W.
Suite 1422
Huntsville 35801
Tel: (205) 533-9300

ARIZONA

Intel Corp.
10210 N. 25th Avenue, Suite 11
Phoenix 85021
Tel: (602) 869-4980

BFA
4426 North Saddle Bag Trail
Scottsdale 85251
Tel: (602) 994-5400

CALIFORNIA

Intel Corp.
7670 Opportunity Rd.
Suite 135
San Diego 92111
Tel: (714) 268-3563

Intel Corp.*
2000 East 4th Street
Suite 100
Santa Ana 92705
Tel: (714) 835-9642
TWX: 910-595-1114

Intel Corp.*
5530 Corbin Ave.
Suite 120
Tarzana 91356
Tel: (213) 708-0333
TWX: 910-495-2045

Intel Corp.*
3375 Scott Blvd.
Santa Clara 95051
Tel: (408) 987-8086
TWX: 910-339-9279
910-338-0255

Earle Associates, Inc.
4617 Ruffner Street
Suite 202
San Diego 92111
Tel: (714) 278-5441

Mac-I
P.O. Box 1420
Cupertino 95014
Tel: (408) 257-9880

Mac-I
558 Valley Way
Calaveras Business Park
Milpitas 95035
Tel: (408) 946-8885

Mac-I
P.O. Box 8763
Fountain Valley 92708
Tel: (714) 839-3341

Mac-I
1321 Centinela Avenue
Suite 1
Santa Monica 90404
Tel: (213) 829-4797

Mac-I
20121 Ventura Blvd., Suite 240E
Woodland Hills 91364
Tel: (213) 347-5900

COLORADO

Intel Corp.*
650 S. Cherry Street
Suite 720
Denver 80222
Tel: (303) 321-8086
TWX: 910-931-2289

CONNECTICUT

Intel Corp.
Peacock Alley
36 Padanaram Road
Danbury 06810
Tel: (203) 792-8366
TWX: 710-456-1199

EMC Corp.
48 Purnell Place
Manchester 06040
Tel: (203) 646-8085

FLORIDA

Intel Corp.
1100 Cleveland Street
Suite 900
Clearwater 33515
(813) 446-4334

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 Holcomb Bridge Rd.
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.
9101 Wesleyan Road
Suite 204
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.
9393 W. 110th St., Ste. 265
Overland Park 66210
Tel: (913) 642-8080

MARYLAND

Intel Corp.*
7257 Parkway Drive
Hanover 21076
Tel: (301) 796-7500
TWX: 710-862-1944

Mesa Inc.
16201 Industrial Dr.
Gaithersburg 20760
Tel: (301) 948-4350

MASSACHUSETTS

Intel Corp.*
27 Industrial Ave.
Chelmsford 01824
Tel: (617) 256-1800
TWX: 710-343-6333

EMC Corp.
381 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.
7401 Metro Blvd.
Suite 355
Edina 55435
Tel: (612) 835-6722
TWX: 910-576-2867

MISSOURI

Intel Corp.
502 Earth City Plaza
Suite 121
Earth City 63045
Tel: (314) 291-1990

NEW JERSEY

Intel Corp.*
Raritan Plaza
2nd Floor
Raritan Center
Edison 08837
Tel: (201) 225-3000
TWX: 710-480-6238

NEW MEXICO

BFA Corporation
P.O. Box 1237
Las Cruces 88004
Tel: (505) 523-0601
TWX: 910-983-0543

BFA Corporation
3705 Westerfield, N.E.
Albuquerque 87111
Tel: (505) 292-1212
TWX: 910-989-1157

NEW YORK

Intel Corp.*
300 Motor Pkwy.
Hauppauge 11787
Tel: (516) 231-3300
TWX: 510-227-6236

Intel Corp.
80 Washington St.
Poughkeepsie 12601
Tel: (914) 473-2303
TWX: 510-248-0060

Intel Corp.*
2255 Lyell Avenue
Lower Floor East Suite
Rochester 14606
Tel: (716) 254-6120
TWX: 510-253-7391

Measurement Technology, Inc.
159 Northern Boulevard
Great Neck 11021
Tel: (516) 482-3500

T-Squared
4054 Newcourt Avenue
Syracuse 13206
Tel: (315) 463-8592
TWX: 710-541-0554

T-Squared
2 E. Main
Victor 14564
Tel: (716) 924-9101
TWX: 510-254-8542

NORTH CAROLINA

Intel Corp.
2306 W. Meadowview Rd.
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 Blvd.
Cleveland 44122
Tel: (216) 464-2736
TWX: 810-427-9298

OREGON

Intel Corp.
10700 S.W. Beaverton
Hillsdale Highway
Suite 324
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.*
2925 L.B.J. Freeway
Suite 175
Dallas 75234
Tel: (214) 241-9521
TWX: 910-860-5617

Intel Corp.*
6420 Richmond Ave.
Suite 280
Houston 77057
Tel: (713) 784-3400
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.
3519 Lexington Dr.
Bountiful, UT 84010
Tel: (801) 292-2164

VIRGINIA

Intel Corp.
1501 Santa Rosa Road
Suite C-7
Richmond, VA 23288
Tel: (804) 282-5668

WASHINGTON

Intel Corp.
Suite 114, Bldg. 3
1603 116th Ave. N.E.
Bellevue 98005
Tel: (206) 453-8086
TWX: 910-443-3002

WISCONSIN

Intel Corp.
150 S. Sunnyslope Rd.
Brookfield 53005
Tel: (414) 784-9060

CANADA

Intel Semiconductor Corp.*
Suite 233, Bell Mews
39 Highway 7, Bells Corners
Ottawa, Ontario K2H 8R2
Tel: (613) 829-9714
TELEX: 053-4115

Intel Semiconductor Corp.
50 Galaxy Blvd.
Unit 12
Rexdale, Ontario
M9W 4Y5
Tel: (416) 675-2105
TELEX: 06983574

Multilek, Inc.*
15 Grenfell Crescent
Ottawa, Ontario K2G 0G3
Tel: (613) 226-2365
TELEX: 053-4585

Multilek, Inc.*
Toronto
Tel: 1-800-267-1070

Multilek, Inc.
Montreal
Tel: 1-800-267-1070

*Field Application Location

INTERNATIONAL SALES AND MARKETING OFFICES

3065 Bowers Avenue
Santa Clara, California 95051
Tel: (408) 987-8080
TWX: 910-338-0026
TELEX: 34-6372

INTERNATIONAL DISTRIBUTORS/REPRESENTATIVES

ARGENTINA

Micro Sistemas S. A.
9 De Julio 561
Cordoba
Tel: 54-51-32-880
TELEX: Agezaar 51999

AUSTRALIA

A.J.F. Systems & Components Pty. Ltd.
310 Queen Street
Melbourne
Victoria 3000
Tel: 679-702
TELEX: AA 31261

Warburton Franki
Corporate Headquarters
372 Eastern Valley Way
Chatswood, New South Wales 2067
Tel: 407-3261
TELEX: AA 21299

AUSTRIA

Bacher Elektronische Geraete GmbH
Rotenmulgasse 26
A 1120 Vienna
Tel: 43222 83.56.46
TELEX: 131532

Rekirsch Elektronik Geraete GmbH
Lichtensteinstrasse 97.16
A 1090 Vienna
Tel: (222) 347646
TELEX: 134759

BELGIUM

Inelco Belgium S.A.
Ave. des Croix de Guerre 94
B 1120 Brussels
Tel: 32 (02) 216 01 60
TELEX: 25441

BRAZIL

Icotron S.A.
0511 Av. Mutinga 3650
6 Andar
Pirituba Sao Paulo
Tel: 261-0211
TELEX: 1122274/ICOTBR

CHILE

DIN
Av. Vic. MacKenna 204
Casilla 6055
Santiago
Tel: 227 564
TELEX: 3520003

CHINA

C.M. Technologies
525 University Avenue
Suite A-40
Palo Alto, CA 94301
Tel: (415) 326-9150

COLOMBIA

International Computer Machines
Carrera 7 No. 72-34
Apdo. Aereo 19403
Bogota 1
Tel: 211-7282
TELEX: 41314 INCO
or 44865 Aserco
or 44891 BMS

CYPRUS

Cyprus Eltrom Electronics
P.O. Box 5393
Nicosia
Tel: 21-27982

DENMARK

ITT Multi Komponent
Fabrysparken 31
DK-2600 Gloskrup
Tel: 45-2-45 66 45
TX: 33355

Scandinavian Semiconductor
Supply A/S
Nannasgade 18
DK-2200 Copenhagen
Tel: 45 (01) 83 50 90
TELEX: 19037

FINLAND

Oy Fintronic AB
Melkonkatu 24 A
SF-00210
Helsinki 21
Tel: 358-692 6022
TELEX: 124 224 Ftron SF

FRANCE

Celdis S.A.*
53, Rue Charles Frerot
F-94250 Gentilly
Tel: 33 (1) 546.13.13
TELEX: 200 485

Feutrier
Rue des Trois Glorieuses
F-42270 St. Priest-en-Jarez
Tel: 33 (77) 74 67 33
TELEX: 300 0 21

Metrologie*
La Tour d' Asnieres
1, Avenue Laurent Cely
92606-Asnieres
Tel: 33-1-853.17.21
TELEX: 611 448

Tekelec Airtronic*
Cite des Bruyeres
Rue Carle Vernet
F-92310 Sevres
Tel: 33 (1) 534 75 35
TELEX: 204552

GERMANY

Electronic 2000 Vertriebs GmbH
Neumarkter Strasse 75
D-8000 Munich 80
Tel: 49 (89) 434061
TELEX: 522561

Jermyn GmbH
Postfach 1180
Schulstrasse 48
D-6277 Camberg
Tel: 49 (6434) 231
TELEX: 484426

Kontron Elektronik GmbH
Breslauerstrasse 2
8057 Eching B
D-8000 Munich
Tel: 49 (89) 319 011
TELEX: 522122

Neye Enatechnik GmbH
Schillerstrasse 14
D-2085 Quickborn-Hamburg
Tel: 49 (4106) 6121
TELEX: 213590

GREECE

American Technical Enterprises
P.O. Box 156
Athens
Tel: 30-1-8811271
TX: 30-1-8219470

HONG KONG

Schmidt & Co.
Wing on Center, 28th Floor
Connaught Road
Hong Kong
Tel: 5-455-644
TELEX: 74766 Schmc Hx

INDIA

Micronic Devices
104/108C, Nirmal Industrial Estate
Sion (E)
Bombay 400022, India
Tel: 486-170
TELEX: 011-5947 MDEV IN

ISRAEL

Eastronics Ltd.*
11 Rozanis Street
P.O. Box 39300
Tel Aviv 61390
Tel: 972-3-47 51 51
TELEX: 33638

ITALY

Eledra 3S S.P.A.*
Viale Elvezia, 18
I 20154 Milano
Tel: 39-2-34.97.51
TELEX: 332332

JAPAN

Asahi Electronics Co. Ltd.
KMM Bldg. Room 407
2-14-1 Asano, Kokura
Kita-Ku, Kitakyushu City 802
Tel: (093) 511-6471
TELEX: AECKY 7126-16

Hamilton-Avnet Electronics Japan Ltd.
YU and YOU Bldg. 1-4 Horidome-Cho
Nihonbashi Chuo-Ku, Tokyo 103
Tel: (03) 662-9911
TELEX: 2523774

Ryoyo Electric Corp.
Konwa Bldg.
1-12-22, Tsukiji
Chuo-Ku, Tokyo 104
Tel: (03) 543-7711

Tokyo Electron Ltd.
Shin Juku, Nomura Bldg.
26-2 Nishi-Shin Juku-Ichome
Shin Juku-Ku, Tokyo 160
Tel. (03) 343-4411
TELEX: 232-2220 LABTEL J

KOREA

Koram Digital
Room 909 Woonam Bldg.
7, 1-KA Bongre-Dong
Chung-Ku Seoul
Tel: 238-123
TELEX: K23542 HANSINT

MEXICO

Proveedora Electronica, S.A. (Proesa)
Prol. Moctezuma Ote. 24
Col. Romero de Terreros
Apdo. Postal 21-139
Mexico 21, D.F.
Tel: 554-8300
TELEX: 017-72402 SAULME

NETHERLANDS

Inelco Nether. Comp. Sys. BV
Turfstekerstraat 63
P.O. Box 360
NL Aalsmeer 1430
Tel: (2977) 28855
TELEX: 14693

Koning & Hartman
Koperwerf 30
P.O. Box 43220
2544 EN's Gravenhage
Tel: 31 (70) 210.101
TELEX: 31528

NEW ZEALAND

W. K. McLean Ltd.
P.O. Box 18-065
Glenn Innes, Auckland, 6
Tel: 587-037
TELEX: NZ2763 KOSFY

NORWAY

Nordisk Elektronic (Norge) A/S
Postoffice Box 122
Smedsvingen 4
1364 Hvalstad
Tel: 02 78 62 10
TELEX: 16963

PORTUGAL

Ditram
Componentes E Electronica LDA
Av. Miguel Bombarda, 133
P1000 Lisboa
Tel: 351 (19) 545313
TELEX: 14182 Brieks-P

SINGAPORE

General Engineers Associates
Blk 3, 1003-1008, 10th Floor
P.S.A. Multi-Storey Complex
Telok Blangah/Pasir Panjang
Singapore 5
Tel: 271-3163
TELEX: RS23987 GENERCO

SOUTH AFRICA

Electronic Building Elements
P.O. Box 4609
Hazelwood, Pretoria 0001
Tel: 011-27-12-46-9221
TELEX: 30181SA

SPAIN

Interface S.A.
Ronda San Pedro 22, 3°
Barcelona 10
Tel: 34-3 301 78 51
TWX: 51508

ITT SESA
Miguel Angel 16-6
Madrid 10
Tel: 34-1-410.23.54
TELEX: 27707/27461

SWEDEN

AB Gosta Backstrom
Box 12009
Alstrahergatan 22
S-10221 Stockholm 12
Tel: 46 (8) 541 080
TELEX: 10135

Nordisk Electronik AB
Box 27301
S-10254 Stockholm
Tel: 46 (8) 635040
TELEX: 10547

SWITZERLAND

Industrade AG
Gemsenstrasse 2
Postcheck 80 - 21190
CH-8021 Zurich
Tel: 41 (1) 363 22 30
TELEX: 56788

TAIWAN

Taiwan Automation Co.*
3d Floor #75, Section 4
Nanking East Road
Taipei
Tel: 771-0940
TELEX: 11942 TAIAUTO

TURKEY

Turkelek Electronics
Apapurk Boulevard 169
Ankara
Tel: 189483

UNITED KINGDOM

Comway Microsystems Ltd.
Market Street
UK-Bracknell, Berkshire
Tel: 44 (344) 51654
TELEX: 847201

M.E.D.L.
East Lane Road
North Wembley
Middlesex HA9 7PP
Tel: 44 (01) 904-9303/908-4111
TELEX: 28817

Jermyn Industries (Mogul)
Vestry Estate
Sevenoaks, Kent
Tel: (0732) 501.44
TELEX: 95142

Rapid Recall, Ltd.
Rapid House/Denmark St
High Wycombe
Bucks, England HP11 2ER
Tel: 44 494 26 271
TELEX: 849439

Bytech Ltd.
Sutton Park Avenue
Reading, Berkshire 61A2
Tel: (0734) 61 031
TELEX: 848215

VENEZUELA

Componentes y Circuitos
Electronicos TTLCA C.A.
Apartado 3223
Caracas 101
Tel: 718-100
TELEX: 21795 TELETIPOS

*Field Application Location

Intel Corporation
3065 Bowers Avenue
Santa Clara, CA 95051

Intel International
Rue du Moulin à Papier 51, Boite 1,
B-1160 Brussels, Belgium

Intel Japan K.K.
Flower Hill-Shinmachi East Bldg.
1-23-9, Shinmachi, Setagayu-ku
Tokyo 154, Japan

Printed in U.S.A./C219/0681/30K/RRD