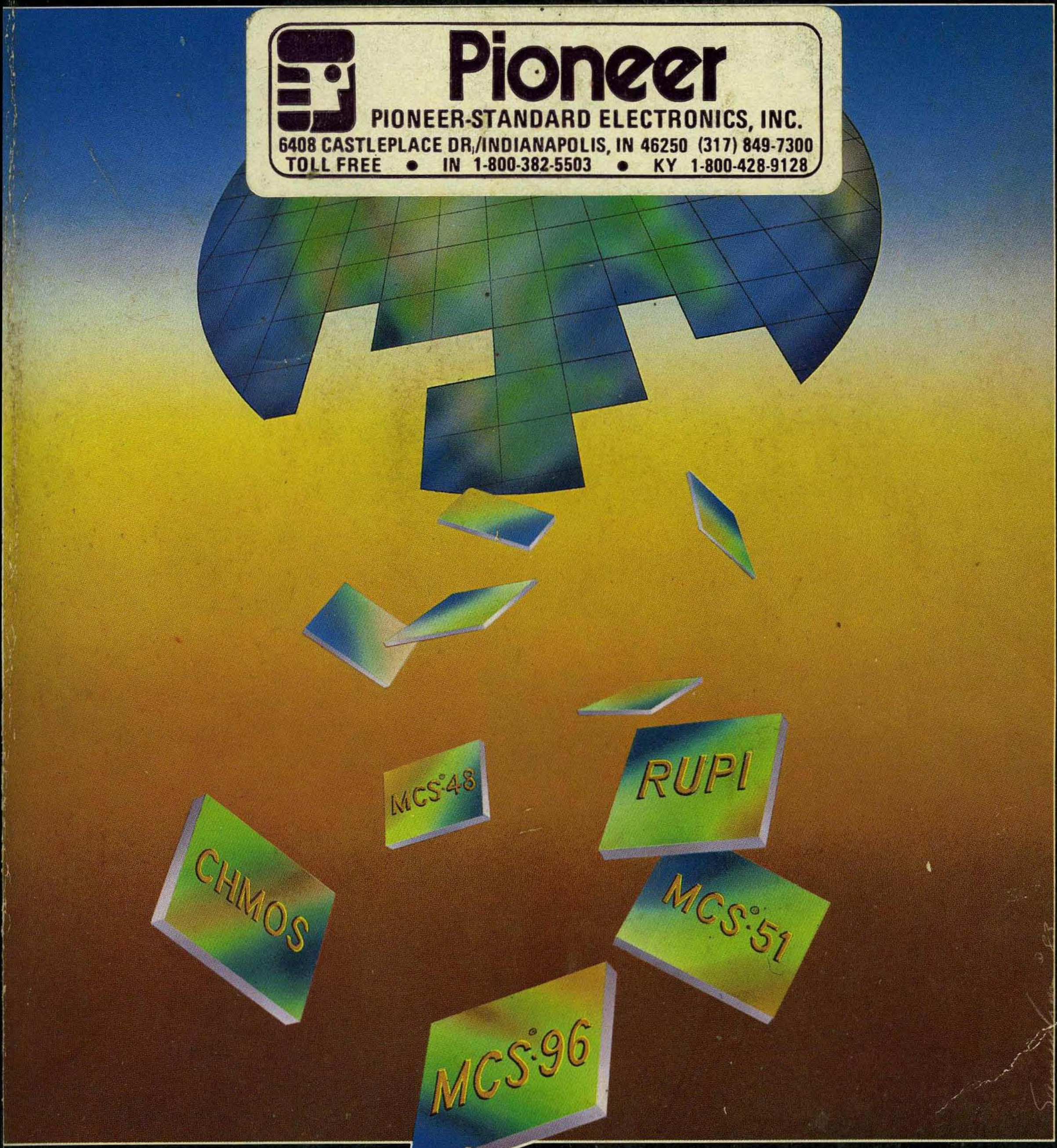
intel

Microcontroller Handbook



Order Number: 210918-002

intel Microcontroller Handbook 1984

LITERATURE

In addition to the product line Handbooks listed below, the INTEL PRODUCT GUIDE (no charge, Order No. 210846) provides an overview of Intel's complete product line and customer services.

Consult the INTEL LITERATURE GUIDE for a complete listing of Intel literature. TO ORDER literature in the United States, write or call the Intel Literature Department, 3065 Bowers Avenue, Santa Clara, CA 95051, (800) 538-1876, or (800) 672-1833 (California only). TO ORDER literature from international locations, contact the nearest Intel sales office or distributor (see listings in the back of most any Intel literature).

1984 HANDBOOKS	U.S. PRICE*
Memory Components Handbook (Order No. 210830) Contains all application notes, article reprints, data sheets, and other design information on RAMs, DRAMs, EPROMs, E^{2}PROMs, Bubble Memories.	**$15.00**
Telecommunication Products Handbook (Order No. 230730) Contains all application notes, article reprints, and data sheets for telecommunication products.	**7.50**
Microcontroller Handbook (Order No. 210918) Contains all application notes, article reprints, data sheets, and design information for the MCS-48, MCS-51 and MCS-96 families.	**15.00**
Microsystem Components Handbook (Order No. 230843) Contains application notes, article reprints, data sheets, technical papers for microprocessors and peripherals. (2 Volumes) (Individual User Manuals are also available on the 8085, 8086, 8088, 186, 286, etc. Consult the Literature Guide for prices and order numbers.)	**20.00**
Military Handbook (Order No. 210461) Contains complete data sheets for all military products. Information on Leadless Chip Carriers and on Quality Assurance is also included.	**10.00**
Development Systems Handbook (Order No. 210940) Contains data sheets on development systems and software, support options, and design kits.	**10.00**
OEM Systems Handbook (Order No. 210941) Contains all data sheets, application notes, and article reprints for OEM boards and systems.	**15.00**
Software Handbook (Order No. 230786) Contains all data sheets, applications notes, and article reprints available directly from Intel, as well as 3rd Party software.	**10.00**

*** Prices are for the U.S. only.**

APPLICATION FOR FREE SUBSCRIPTION TO SOLUTIONS MAGAZINE

Solutions is a bimonthly customer magazine covering all of Intel's new product announcements and offering useful application information. If you wish to receive *SOLUTIONS* regularly, please fill in the requested information and mail it back to:

Intel SC6-714
3065 Bowers Avenue
Santa Clara, CA 95051

See reverse side for European return addresses.

There is no charge—just your continued interest in Intel products.

1. Your principal responsibility (please check one box only).

AA □ General Management
BB □ Engineering Management
AC □ Hardware Design Engineer
AD □ Software Design Engineer
AE □ Technician
AF □ Manufacturing, Reliability or QA
AG □ Marketing or Sales
AH □ Purchasing/Buyer
AJ □ Educator
AK □ Computer Hobbyist
AL □ Data Processing
AM □ Other ____________________

2. Your company's principal business or industry (please check one box only).

AN □ Manufacturer of computers or peripheral equipment
AP □ Manufacturer of control equipment
AQ □ Manufacturer of test/measurement equipment or instruments
AR □ Manufacturer of communication/telecommunication equipment
AS □ Manufacturer of office/business equipment
AT □ System integration
AU □ User of computer equipment
AV □ Research lab/consultant
AW □ Software house
AX □ Government/military agency
AZ □ Sales office or distributor
BA □ Other ____________________

3. Your company's/division's products fall primarily into:

BB □ Industrial **BC** □ Military **BD** □ Consumer

4. Do you own or use an Intel microcomputer development system or design aid?

BE □ Yes **BF** □ No

5. Are products that are purchased to your specifications incorporated into products to be sold or are they used in-house?

BG □ Incorporated into products to be sold
BH □ Used in-house
BJ □ Both

6. Who do you usually buy from?

BK □ Distributor **BL** □ Manufacturer

7. I am interested in the following product areas:

70 □ Microcomputer Components
71 □ Microcomputer Boards and Systems
72 □ Memory Components
73 □ Memory Boards and Systems
74 □ Military Products
75 □ Telecommunications Products

Name ____________________
Title ____________________
Company ____________________
Phone ____________________
Address ____________________
City __________ **State** __________ **Zip** __________ **Country** __________

For a free subscription to the European edition of *SOLUTIONS* please mail this card to one of the regional offices listed below:

Intel Corporation (UK) Ltd
Piper's Way
Swindon, SN3 1RJ
Wiltshire, England

Intel Corporation S.A.R.L.
5 Place de la Balance
Silic 223
94528 Rungis Cedex
France

Intel Corporation Italia Spa
Milanofiori, Palazzo E
20094 Assago (Milano)
Italy

Intel Semiconductor GmbH
Seidlstrasse 27
D-8000 Munchen 2
West Germany

Intel Sweden AB
Box 20092
Archimedesvagen 5
S-16120 Bromma
Sweden

intel®

MICROCONTROLLER HANDBOOK

1984

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

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

Table of Contents

MCS®-96 FAMILY

MCS®-51 FAMILY

CMOS

MCS®-48 FAMILY

THE RUPI™ FAMILY: MICROCONTROLLER WITH ON-CHIP COMMUNICATION CONTROLLER

PACKAGING INFORMATION

Introduction to MCS®-96

1

CHAPTER 1
INTRODUCTION TO MCS®-96

1.0 CONTINUING MICROCONTROLLER EVOLUTION

Beginning with the introduction of the world standard 8048 (MCS®-48) Microcontroller in 1976, Intel has continued to drive the evolution of single chip microcontrollers. In 1980, Intel introduced the 8051 (MCS-51) offering performance levels significantly higher than the 8048. With the advent of the 8051, the microcontroller applications base took a marked vertical leap. These versatile chips are used in applications from keyboards and terminals to controlling automobile engines. The 8051 quickly gained the position of the second generation world standard microcontroller.

Now that the semiconductor process technologies are being pushed to new limits, it has become possible to integrate more than 100,000 transistors onto a single silicon chip. Microcontroller designers at Intel have taken today's process technology achievements and forged a new generation of single chip microcontrollers called the MCS-96. The 8096 (generic part number for MCS-96) offers the highest level of system integration ever achieved on a single chip microcontroller. It uses over 120,000 transistors to implement a high performance 16-bit CPU, 8K bytes of program memory, 232 bytes of data memory and both analog and digital types of I/O features. Figure 1-1 shows the evolution of single chip microcontroller at Intel.

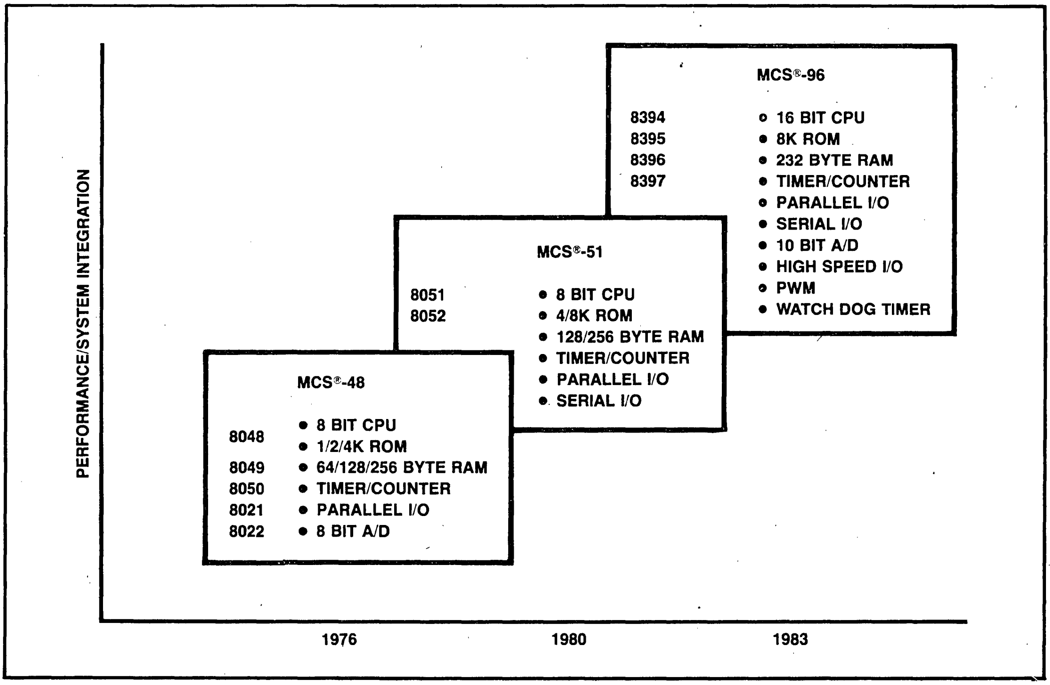


Figure 1-1. Evolution of Microcontrollers at Intel

1.1 INTRODUCTION TO THE MCS®-96

The 8096 consists of a 16-bit powerful CPU tightly coupled with program and data memory along with several I/O features all integrated onto a single piece of silicon. The CPU supports bit, byte, and word operations. 32-bit double words are also supported for a subset of the instruction set. With a 12 MHz input frequency, the 8096 can perform a 16-bit addition in 1.0 μs and 16 × 16 multiply or 32/16 divide in 6.5 μs.

Four high-speed trigger inputs are provided to record the times at which external events occur with a resolution of 2 μs (at 12 MHz crystal frequency). Up to six high-speed pulse generator outputs are provided to trigger external events at preset times. The high speed output unit can simultaneously perform timer functions, up to four such 16-bit software timers can be in operation at once in addition to the two 16-bit hardware timers.

An optional on-chip A/D converter converts up to four (in the 48-pin version) or 8 (in the 68-pin version) analog input channels into 10-bit digital values. Also provided on-chip, is a serial port, a watchdog timer, and a pulse-width modulated output signal. Table 1.1 shows the features and benefits summary for the MCS-96.

The 8096 with its 16-bit CPU and all the I/O features and interface resources on a single piece of silicon represents the highest level of system integration in the world of microcontrollers. It will open up new applications which had to use multiple chip solutions in the past.

Table 1-1 MCS®-96 Features and Benefits Summary

FEATURES	BENEFITS
16-Bit CPU	Efficient machine with higher throughput.
8K Bytes ROM	Large program space for more complex, larger programs.
232 Bytes RAM	Large on-board register file.
Hardware MUL/DIV	Provides good math capability 16 by 16 multiply or 32 by 16 divide in 6.5 μs @ 12 MHz.
6 Addressing Modes	Provides greater flexibility of programming and data manipulation.
High Speed I/O Unit 4 dedicated I/O lines 4 programmable I/O lines	Can measure and generate pulses with high resolution (2 μs @ 12 MHz).
10-Bit A/D Converter	Reads the external analog inputs.
Full Duplex Serial Port	Provides asynchronous serial link to other processors or systems.
Up to 40 I/O Ports	Provides TTL compatible digital data I/O including system expansion with standard 8 or 16-bit peripherals.
Programmable 8 Source Priority Interrupt System	Respond to asynchronous events.
Pulse Width Modulated Output	Provides a programmable pulse train with variable duty cycle. Also used to generate analog output.
Watchdog Timer	Provides ability to recover from software malfunction or hardware upset.
48 Pin (DIP) & 68 Pin (Flatpack, Pin Grid Array) Versions	Offers a variety of package types to choose from to better fit a specific application need for number of I/O's and package size.

1.2. MCS®-96 APPLICATIONS

The MCS-96 products are stand-alone high performance single chip microcontrollers designed for use in sophisticated real-time demanding applications such as industrial control, instrumentation and intelligent computer peripherals. The wide base of applications cut across all industry segments (see table 1.2). With the 16-bit CPU horsepower, high-speed math processing and high-speed I/O, the 8096 is ideal for complex motor control and axis control systems. Examples include three phase, large horsepower AC motors and robotics.

With its 10-bit A/D converter option, the device finds usage in data acquisition systems and closed-loop analog controllers. It permits considerable system integration by combining analog and digital I/O processing in the single chip.

Table 1-2 MCS®-96 Broad Base of Applications

- INDUSTRIAL
  - Motor Control
  - Robotics
  - Discrete and Continuous Process Control
  - Numerical Control
  - Intelligent Transducers
- INSTRUMENTATION
  - Medical Instrumentation
  - Liquid and Gas Chromatographs
  - Oscillioscopes
- CONSUMER
  - Video Recorder
  - Laser Disk Drive
  - High-end Video Games
- GUIDANCE & CONTROL
  - Missile Control
  - Torpedo Guidance Control
  - Intelligent Ammunition
  - Aerospace Guidance Systems
- DATA PROCESSING
  - Plotters
  - Color and B&W Copiers
  - Winchester Disk Drive
  - Tape Drives
  - Impact and Non-Impact Printers
- TELECOMMUNICATIONS
  - Modems
  - Intelligent Line Card Control
- AUTOMOTIVE
  - Ignition Control
  - Transmission Control
  - Anti Skid Braking
  - Emission Control

This chip is ideally suited in the area of instrumentation products such as gas chromatographs, which combine analog processing with high speed number crunching. The same features make it a desirable component for aerospace applications like missile guidance and control.

1.3. MCS®-96 FAMILY DEVELOPMENT SUPPORT TOOLS

The product family is supported by a range of Intel software and hardware development tools. These tools shorten the product development cycle, thus bringing the product to the market sooner.

1.3.1. MCS®-96 Software Development Package

The 8096 software development package provides development system support specifically designed for the MCS-96 family of single chip microcontrollers. The package consists of a symbolic macro assembler ASM-96, Linker/Relocator RL-96 and the librarian LIB-96. Among the high level languages, PLM-96 is offered along with a floating point math package. Additional high level languages are being developed for the MCS-96 product family.

1.3.2. ASM-96 MACRO Assembler

The 8096 macro assembler translates the symbolic assembly language instructions into the machine executable object code. ASM-96 enables the programmer to write the program in a modular fashion. The modular programs divide a rather complex program into smaller functional units, that are easier to code, to debug, and to change. The separate modules can then be linked and located into one program module using the RL-96 utility. This utility combines the selected input object modules into a single output object module. It also allocates memory to input segments and binds the relocatable addresses to absolute addresses. It then produces a print file that consists of a link summary, a symbol table listing and an intermediate cross-reference listing. LIB-96, another utility helps to create, modify, and examine library files. The ASM-96 runs on Intellec Series III or IV.

1.3.3. PL/M-96

The PL/M-96 compiler translates the PL/M-96 language into 8096 relocatable object modules. This allows improved programmer productivity and application reliability. This high level language has been efficiently designed to map into the machine architecture, so as not to trade off higher programmer productivity with inefficient code. Since the language and the compiler are optimized for the 8096 and its application environment, developing software with PL/M-96 is a 'low-risk' project.

1.3.4. Hardware Development Support: iSBE-96

The iSBE-96 is a hardware execution and debug tool for the MCS-96 products. It consists of a monitor/debugger resident in an 8096 system. This development system interfaces with the user's 8096 system via two ribbon cables, one for the 8096 I/O ports, and the other for the memory bus. The iSBE-96 is controlled by an Intellec Series III or other computer system over a serial link. Power for the iSBE-96 can be supplied by plugging it into the MULTIBUS® card slot, or by an external power supply. The iSBE-96 is contained on one standard MULTIBUS board.

The iSBE-96 provides the most often used features for real-time hardware emulation. The user can display and modify memory, set up break points, execute with or without breakpoints and change the memory map. In addition, the user can single step through the system program.

1.3.5. MCS®-96 Workshop

The workshop provides the design engineer or system designer hands-on experience with the MCS-96 family of products. The course includes an explanation of the Intel 8096 architecture, system timing, input/output design. The lab sessions allow the attendees to gain in-depth knowledge of the MCS-96 product family and support tools.

1.3.6. Insite™ Library

The Intel Insite Library contains several application programs. A very useful program contained in the Insite is SIM-96, the software simulator for 8096. It allows software simulations of user's system. The simulator provides the ability to set breakpoints, examine and modify memory, disassemble the object code and single step through the code.

1.4. MCS®-96 FAMILY OF PRODUCTS

Although 8096 is the generic part number often used for the MCS-96 products throughout this manual, the product family consists of eight configurations with eight part numbers including the 8096. This wide variety of products is offered to best meet user's application requirements in terms of number of I/O's and package size. The options include on-board 8K bytes of mask programmed memory, 10-bit A/D converter, and 48 or 68 pin package type.

Table 1-3 summarizes all the current products in the MCS®-96 product family.

Table 1-3 MCS®-96 Family of Products

OPTIONS		68 PIN	48 PIN
DIGITAL I/O	ROMLESS	8096	8094
	ROM	8396	8394
ANALOG AND DIGITAL I/O	ROMLESS	8097	8095
	ROM	8397	8395

The 48 pin version is available in a DIP (dual inline) package.

The 68 pin version comes in two packages, the Plastic Flatpack and the Pin Grid Array.

Architectural Overview 2

CHAPTER 2
ARCHITECTURAL OVERVIEW

2.0. INTRODUCTION

The 8096 can be separated into several sections for the purpose of describing its operation. There is a CPU, a programmable High Speed I/O Unit, an analog to digital converter, a serial port, and a Pulse Width Modulated (PWM) output for digital to analog conversion. In addition to these functional units, there are some sections which support overall operation of the chip such as the clock generator and the back-bias generator. The CPU and the programmable I/O make the 8096 very different from any other microcontroller, let us first examine the CPU.

2.1. CPU OPERATION

The major components of the CPU on the 8096 are the Register File and the RALU. Communication with the outside world is done through either the Special Function Registers (SFRs) or the Memory Controller. The RALU (Register/Arithmetic Logic Unit) does not use an accumulator, it operates directly on the 256-byte register space made up of the Register File and the SFRs. Efficient I/O operations are possible by directly controlling the I/O through the SFRs. The main benefits of this structure are the ability to quickly change context, the absence of accumulator bottleneck, and fast throughput and I/O times.

2.1.1. CPU Buses

A "Control Unit" and two buses connect the Register File and RALU. Figure 2-1 shows the CPU with its major bus connections. The two buses are the "A-Bus" which is 8-bits wide, and the "D-Bus" which is 16-bits wide. The D-Bus transfers data only between the RALU and the Register File or Special Function Registers (SFRs). The A-Bus is used as the address bus for the above transfers or as a multiplexed address/data bus connecting to the "Memory Controller". Any accesses of either the internal ROM or external memory are done through the Memory Controller.

Within the memory controller is a slave program counter (Slave PC) which keeps track of the PC in the CPU. By having most program fetches from memory referenced to

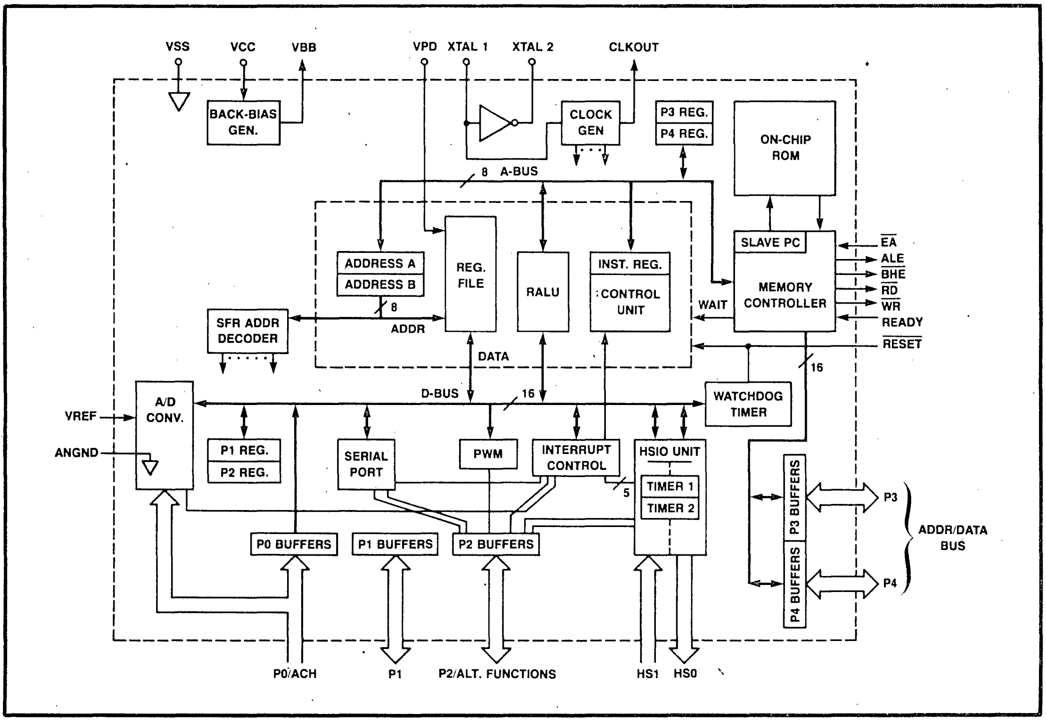


Figure 2-1. Block Diagram (For simplicity, lines connecting port registers to port buffers are not shown.)

the slave PC, the processor saves time as addresses seldom have to be sent to the memory controller. If the address jumps sequence then the slave PC is loaded with a new value and processing continues. Data fetches from memory are also done through the memory controller, but the slave PC is bypassed for this operation.

2.1.2. CPU Register File

The Register File contains 232 bytes of RAM which can be accessed as bytes, words, or double-words. Since each of these locations can be used by the RALU, there are essentially 232 "accumulators". The first word in the Register File is reserved for use as the stack pointer so it can not be used for data when stack manipulations are taking place. Addresses for accessing the Register File and SFRs are temporarily stored in two 8-bit address registers by the CPU hardware.

2.1.3. RALU Control

Instructions to the RALU are taken from the A-Bus and stored temporarily in the instruction register. The Control Unit decodes the instructions and generates the correct sequence of signals to have the RALU perform the desired function. Figure 2-1 shows the instruction register and the control unit.

2.1.4. RALU

Most calculations performed by the 8096 take place in the RALU. The RALU, shown in Figure 2-2, contains a 17-bit ALU, the Program Status Word (PSW), the Program Counter (PC), a loop counter, and three temporary registers. All of the registers are 16-bits or 17-bits (16+ sign extension) wide. Some of the registers have the ability to perform simple operations to off-load the ALU.

A separate incrementer is used for the PC; however, jumps must be handled through the ALU. Two of the temporary registers have their own shift logic. These registers are used for the operations which require logical shifts, including Normalize, Multiply, and Divide. The "Lower Word" register is used only when double-word quantities are being shifted, the "Upper Word" register is used

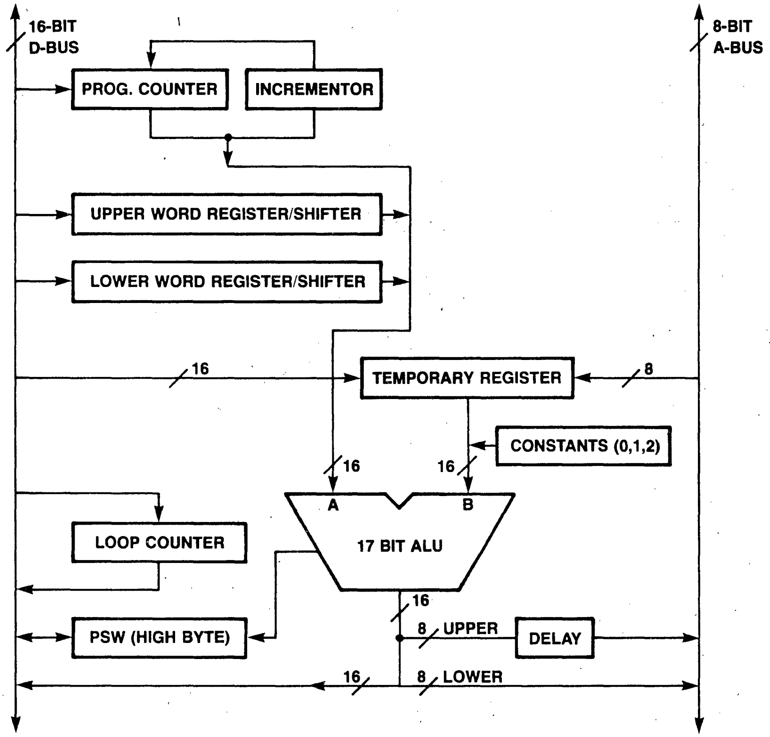


Figure 2-2. RALU Block Diagram

whenever a shift is performed or as a temporary register for many instructions. Repetitive shifts are counted by the 5-bit "Loop Counter".

A temporary register is used to store the second operand of two operand instructions. This includes the multiplier during multiplications and the divisor during divisions. To perform subtractions, the output of this register can be complemented before being placed into the "B" input of the ALU.

The DELAY shown in Figure 2-2 is used to convert the 16-bit bus into an 8-bit bus. This is required as all addresses and instructions are carried on the 8-bit A bus. Several constants, such as 0, 1 and 2 are stored in the RALU for use in speeding up certain calculations. These come in handy when the RALU needs to make a 2's complement number or perform an increment or decrement instruction.

2.2. BASIC TIMING

The 8096 requires an input clock frequency of between 6.0 MHz and 12 MHz to function. This frequency can be applied directly to XTAL1. Alternatively, since XTAL1 and XTAL2 are inputs and outputs of an inverter, it is also possible to use a crystal to generate the clock. A block diagram of the oscillator section is shown in Figure 2-3. Details of the circuit and suggestions for its use can be found in section 4.1.

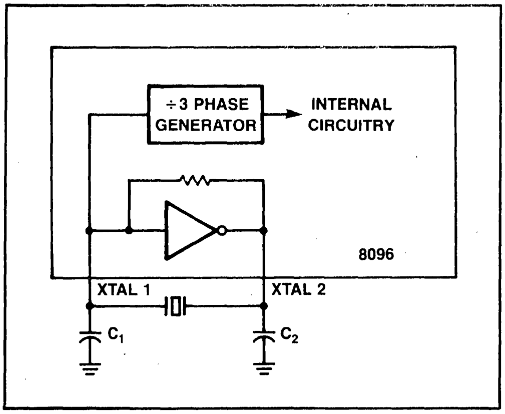


Figure 2-3. Block Diagram of Oscillator

2.2.1. Internal Timings

The crystal or external oscillator frequency is divided by 3 to generate the three internal timing phases as shown in Figure 2-4. Each of the internal phases repeat every 3 oscillator periods: 3 oscillator periods are referred to as one "state time", the basic time measurement for 8096 operations. Most internal operations are synchronized to either Phase A, B or C, each of which have a 33% duty cycle. Phase A is represented externally by CLKOUT, a signal available on the 68-pin part. Phases B and C are not available externally. The relationships of XTAL1, CLKOUT, and Phases A, B, and C are shown in Figure 2-4. It should be noted that propagation delays have not

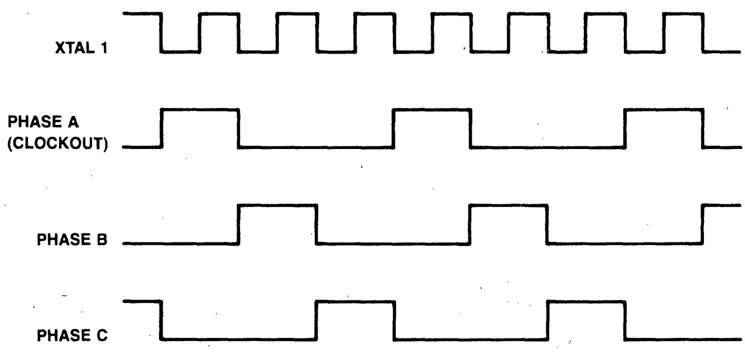


Figure 2-4. Internal Timings Relative to XTAL 1

been taken into account in this diagram. Details on these and other timing relationships can be found in sections 4.1, 4.4 and 4.6.

The RESET line can be used to start the 8096 at an exact time to provide for synchronization of test equipment and multiple chip systems. Use of this feature is fully explained under RESET, sections 2.15 and 4.1.

2.3. MEMORY SPACE

The addressable memory space on the 8096 consists of 64K bytes, most of which is available to the user for program or data memory. Locations which have special purposes are 0000H through 00FFH and 1FFEH through 2010H. All other locations can be used for either program or data storage or for memory mapped peripherals. A memory map is shown in figure 2-5.

2.3.1. Register File

Locations 00H through 0FFH contain the Register File and SFRs. Complete information on this section of memory space can be found in section "RAM SPACE". No code can be executed from this internal RAM section. If an attempt to execute instructions from locations 000H through 0FFH is made, the instructions will be fetched from *external* memory. This section of external memory is reserved for use by Intel development tools. Execution of a nonmaskable interrupt (NMI) will force a call to external location 0000H, therefore, the NMI instruction is also reserved for Intel development tools.

2.3.2. Reserved Memory Spaces

Locations 1FFEH and 1FFFH are reserved for Ports 3 and 4 respectively. This is to allow easy reconstruction of these ports if external memory is used in the system. An example of reconstructing the I/O ports is given in section 4.6.7. If ports 3 and 4 are not going to be reconstructed then these locations can be treated as any other external memory location.

The 9 interrupt vectors are stored in locations 2000H through 2011H. The 9th vector is used by Intel development systems, as explained in section 2.5. Internal locations 2012H through 2077H are reserved for Intel's factory test code. These locations may be used in external memory.

Resetting the 8096 causes instructions to be fetched starting from location 2080H. This location was chosen to allow a system to have up to 8K of RAM continuous with the register file. Further information on reset can be found in section 2.15.

2.3.3. Internal ROM

When a ROM part is ordered, the internal memory locations 2080H through 3FFFH are user specified as are the interrupt vectors in locations 2000H through 2011H.

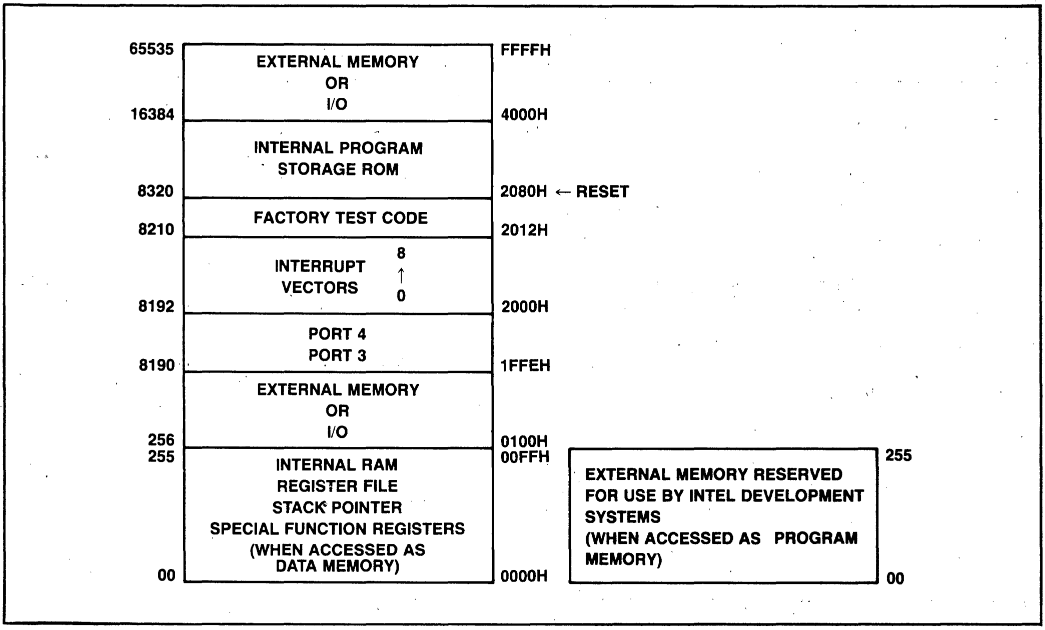


Figure 2-5. Memory Map

Instruction and data fetches from the internal ROM occur only if the part has a ROM, $\overline{EA}$ is tied high, and the address is between 2000H and 3FFFH. At all other times data is accessed from either the internal RAM space or external memory and instructions are fetched from external memory.

2.3.4. Memory Controller

The RALU talks to the memory (except for the locations in the register file and SFR space) through the memory controller which is connected to the RALU by the A-bus and several control lines. Since the A-bus is eight bits wide, the memory controller uses a Slave Program Counter to avoid having to always get the instruction location from the RALU. This slave PC is incremented after each fetch. When a jump or call occurs, the slave PC must be loaded from the A-bus before instruction fetches can continue.

In addition to holding a slave PC, the memory controller contains a 3 byte queue to help speed execution. This queue is transparent to the RALU and to the user unless wait states are forced during external bus cycles. The instruction execution times shown in Tables 3-3 and 3-4 show the normal execution times with no wait states added. Reloading the slave PC and fetching the first byte of the new instruction stream takes 4 state times. This is reflected in the jump taken/not-taken times shown in Table 3-4.

2.3.5. System Bus

External memory is addressed through lines AD0 through AD15 which form a 16-bit multiplexed (address/data) data bus. These lines share pins with I/O ports 3 and 4. The falling edge of the Address Latch Enable (ALE) line is used to provide a clock to a transparent latch (74LS373) in order to demultiplex the bus. A typical circuit and the required timings are shown in section 4.6. Since the 8096's external memory can be addressed as either bytes or words, the decoding is controlled with two lines, Bus High Enable ($\overline{BHE}$) and Addresss/Data Line 0 (AD0). The $\overline{BHE}$ line must be transparently latched, just as the addresses are.

To avoid confusion during the explanation of the memory system it is reasonable to give names to the demultiplexed address/data signals. The address signals will be called MA0 through MA15 (Memory Address), and the data signals will be called MD0 through MD15 (Memory Data).

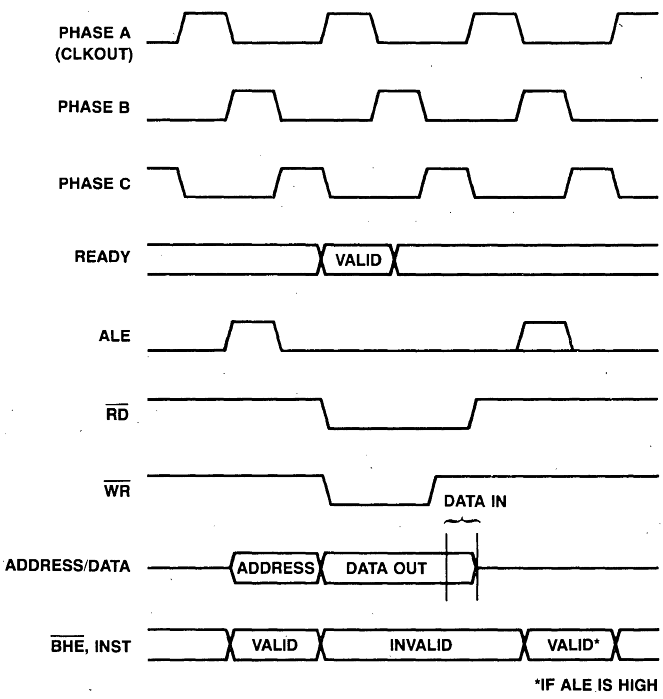


Figure 2-6. External Memory Timings

When $\overline{\mathrm{BHE}}$ is active (low), the memory connected to the high byte of the data bus should be selected. When MA0 is low the memory connected to the low byte of the data bus should be selected. In this way accesses to a 16-bit wide memory can be to the low (even) byte only (MA0-=0, $\overline{\mathrm{BHE}}$=1), to the high (odd) byte only (MA0=1, $\overline{\mathrm{BHE}}$=0), or to both bytes (MA0=0, $\overline{\mathrm{BHE}}$=0). When a memory block is being used only for reads, $\overline{\mathrm{BHE}}$ and MA0 need not be decoded.

Figure 2-6 shows the idealized waveforms related to the following description of external memory manipulations. For exact timing specifications please refer to the latest data sheet. When an external memory fetch begins, the address latch enable (ALE) line rises, the address is put on AD0-AD15 and $\overline{\mathrm{BHE}}$ is set to the required state. ALE then falls, the address is taken off the pins, and the $\overline{\mathrm{RD}}$ (Read) signal goes low. The READY line can be pulled low to hold the processor in this condition for a few extra state times.

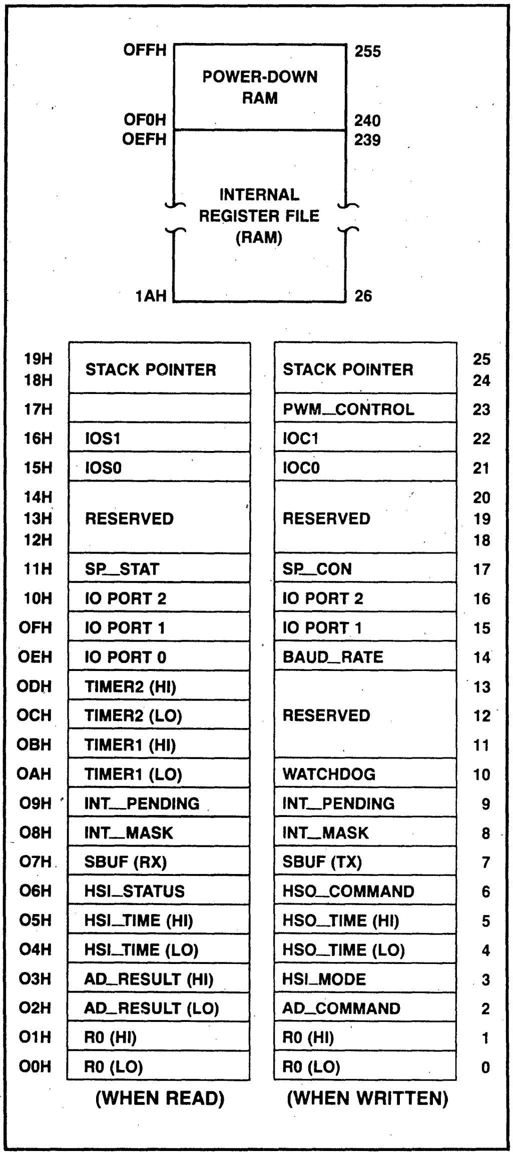
2.3.6. Ready

The READY line can be used to hold the processor in the above condition in order to allow access to slow memories or for DMA purposes. Sampling of the READY line occurs internally during Phase 2, which is the signal that generates CLKOUT. There is a minimum time in which READY must be stable before CLKOUT goes low. If this set-up time is violated while the part is going to the not-ready state, the part may fail to operate predictably.

Since READY is synchronized with CLKOUT, the 8096 will be in a not-ready condition for a period equal to some multiple of CLKOUT, although the READY line can be brought high at any time. There is a maximum time for holding the 8096 in the not-ready condition, typically on the order of 1 μS. The exact time is specified in the data sheet for the particular part and temperature range desired.

The data from the external memory must be on the bus and stable for a minimum of the specified set-up time before the rising edge of $\overline{\mathrm{RD}}$. The rising edge of $\overline{\mathrm{RD}}$ latches the information into the 8096. If the read is for data, the INST pin will be low when the address is valid, if it is for an instruction the INST pin will be high during this time. The 48-lead part does not have the INST pin.

Writing to external memory requires timings that are similar to those required when reading from it. The main difference is that the write ($\overline{\mathrm{WR}}$) signal is used instead of the $\overline{\mathrm{RD}}$ signal. The timings are the same until the falling edge of the $\overline{\mathrm{WR}}$ line. At this point the 8096 removes the address and places the data on the bus. The READY line must be held in the desired state at that time as described above. When the $\overline{\mathrm{WR}}$ line goes high the data should be latched to the external memory. INST is always low during a write, as instructions cannot be written. The exact timing specifications for memory accesses can be found in the data sheet.



19H	STACK POINTER	STACK POINTER	25
18H			24
17H		PWM_CONTROL	23
16H	IOS1	IOC1	22
15H	IOS0	IOC0	21
14H	RESERVED	RESERVED	20
13H			19
12H			18
11H	SP_STAT	SP_CON	17
10H	IO PORT 2	IO PORT 2	16
OFH	IO PORT 1	IO PORT 1	15
OEH	IO PORT 0	BAUD_RATE	14
ODH	TIMER2 (HI)	RESERVED	13
OCH	TIMER2 (LO)		12
OBH	TIMER1 (HI)		11
OAH	TIMER1 (LO)	WATCHDOG	10
O9H	INT_PENDING	INT_PENDING	9
O8H	INT_MASK	INT_MASK	8
O7H	SBUF (RX)	SBUF (TX)	7
O6H	HSI_STATUS	HSO_COMMAND	6
O5H	HSI_TIME (HI)	HSO_TIME (HI)	5
O4H	HSI_TIME (LO)	HSO_TIME (LO)	4
O3H	AD_RESULT (HI)	HSI_MODE	3
O2H	AD_RESULT (LO)	AD_COMMAND	2
O1H	R0 (HI)	R0 (HI)	1
O0H	R0 (LO)	R0 (LO)	0
	(WHEN READ)	(WHEN WRITTEN)	

Figure 2-7. Register File Memory Map

2.4. RAM SPACE

The internal register locations on the 8096 are divided into two groups, a register file and a set of Special Function Registers (SFRs). The RALU can operate on any of these 256 internal register locations. Locations 00H through 17H are used to access the SFRs. Locations 18H and 19H contain the stack pointer. There are no restrictions on the use of the remaining 230 locations except that code cannot be executed from them.

2.4.1. Special Function Registers

All of the I/O on the 8096 is controlled through the SFRs. Many of these registers serve two functions; one if they are read from, the other if they are written to. Figure 2-7 shows the locations and names of these registers. A summary of the capabilities of each of these registers is shown in Figure 2-8, with complete descriptions reserved for later chapters.

Register	Description	Chapter
R0	Zero Register — Always reads as a zero, useful for a base when indexing and as a constant for calculations and compares.	3.2.7
AD_RESULT	A/D Result Hi/Low — Low and high order Results of the A/D converter	2.9.3
AD_COMMAND	A/D Command Register — Controls the A/D	2.9.2
HSI_MODE	HSI Mode Register — Sets the mode of the High Speed Input unit.	2.7.1
HSI_TIME	HSI Time Hi/Lo — Contains the time at which the High Speed Input unit was triggered.	2.7.4
HSO_TIME	HSO Time Hi/Lo — Sets the time for the High Speed Output to execute the command in the Command Register.	2.8.3
HSO_COMMAND	HSO Command Register — Determines what will happen at the time loaded into the HSO Time registers.	2.8.2
HSI_STATUS	HSI Status Registers — Indicates which HSI pins were detected at the time in the HSI Time registers.	2.7.4
SBUF (RX)	Receive buffer for the serial port, holds contents to be outputed.	2.11
SBUF (TX)	Transmit buffer for the serial port, holds the byte just received by the serial port.	2.11
INT_MASK	Interrupt Mask Register — Enables or disables the individual interrupts.	2.5.2 3.6.2
INT_PENDING	Interrupt Pending Register — Indicates when an interrupt signal has occurred on one of the sources.	2.5.2 3.6.2
WATCHDOG	Watchdog Timer Register — Written to periodically to hold off automatic reset every 64K state times.	2.14
TIMER1	Timer 1 Hi/Lo — Timer 1 high and low bytes.	2.6.1 2.7-8
TIMER2	Timer 2 Hi/Lo — Timer 2 high and low bytes.	2.6.2 2.7-8
IOPORT0	Port 0 Register — Levels on pins of port 0.	2.12.1
BAUD_RATE	Register which contains the baud rate, this register is loaded sequentially.	2.11.4
IOPORT1	Port 1 Register — Used to read or write to Port 1.	2.12.2
IOPORT2	Port 2 Register — Used to read or write to Port 2.	2.12.3
SP_STAT	Serial Port Status — Indicates the status of the serial port.	2.11.3
SP_CON	Serial port control — Used to set the mode of the serial port.	2.11.1
IOS0	I/O Status Register 0 — Contains information on the HSO status.	2.13.4
IOS1	I/O Status Register 1 — Contains information on the status of the timers and of the HSI.	2.13.5 3.7.2
IOC0	I/O Control Register 0 — Controls alternate functions of HSI pins, Timer 2 reset sources and Timer 2 clock sources.	2.13.2
IOC1	I/O Control Register 1 — Controls alternate functions of Port 2 pins, timer interrupts and HSI interrupts.	2.13.3
PWM_CONTROL	Pulse Width Modulation Control Register — Sets the duration of the PWM pulse.	2.10 4.3.2

Figure 2-8. SFR Summary

Within the SFR space are several registers labeled as "RESERVED". These registers are reserved for future expansion or test purposes. Reads or writes of these registers may produce unexpected results. For example, writing to location 000CH will set both timers to 0FFFXH, this feature is for use in testing the part and should not be used in programs.

2.4.2. Power Down

The upper 16 RAM locations (0F0H through 0FFH) receive their power from both the VCC pin and the VPD pin. If it is desired to keep the memory in these locations alive during a power down situation, one need only keep voltage on the VPD pin. The current required to keep the RAM alive is approximately 1 milliamp (refer to the data sheet for the exact specification).

To place the 8096 into a power down mode, the RESET pin is pulled low. Two state times later the part will be in reset. This is necessary to prevent the part from writing into RAM as the power goes down. The power may now be removed from the VCC pin, the VPD pin must remain within specifications. The 8096 can remain in this state for any amount of time and the 16 RAM bytes will retain their values.

To bring the 8096 out of power down, RESET is held low while VCC is applied. Two state times after the oscillator and the back bias generator have stabilized (~1 millisecond), the RESET pin can be pulled high. The 8096 will begin to execute code at location 02080H 10 state times after RESET is pulled high. Figure 2-9 shows a timing diagram of the power down sequence. To ensure that the 2 state time minimum reset time (synchronous with CLKOUT) is met, it is recommended that 10 XTAL1 cycles be used. Suggestions for actual hardware connections are given in section 4.1. Reset is discussed in section 2.15.

2.5. INTERRUPT STRUCTURE

2.5.1. Interrupt Sources

Eight interrupt sources are available on the 8096. When enabled, an interrupt occurring on any of these sources will force a call to the location stored in the vector location for that source. The interrupt sources and their respective vector locations are listed in Figure 2-10. In addition to the 8 standard interrupts, there is a TRAP instruction which acts as a software generated interrupt. This instruction is not currently supported by the MCS-96 Assembler and is reserved for use by Intel development systems. Many of the interrupt sources can be activated by several methods, Figure 2-11 shows all of the possible sources for interrupts.

Source	Vector Location		Priority
	(High Byte)	(Low Byte)	
Software	2011H	2010H	Not Applicable
Extint	200FH	200EH	7 (Highest)
Serial Port	200DH	200CH	6
Software Timers	200BH	200AH	5
HSI.0	2009H	2008H	4
High Speed Outputs	2007H	2006H	3
HSI Data Available	2005H	2004H	2
A/D Conversion Complete	2003H	2002H	1
Timer Overflow	2001H	2000H	0 (Lowest)

Figure 2-10. Interrupt Vector Locations

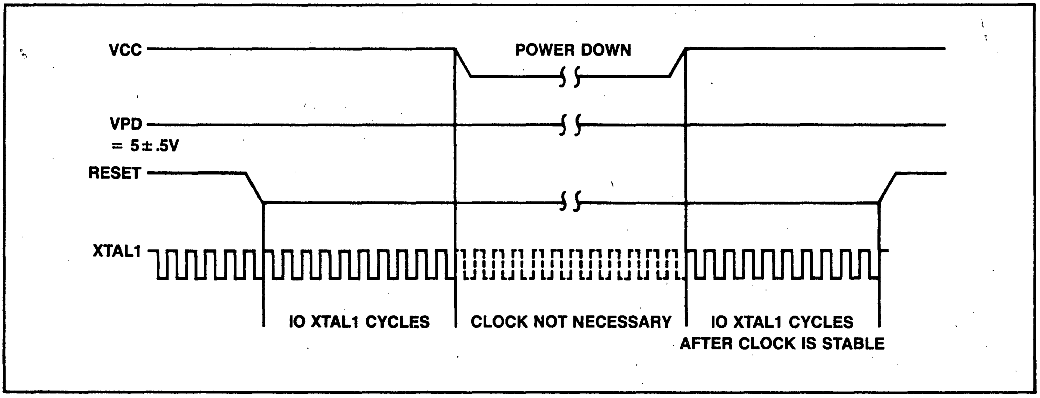


Figure 2-9. Power Down Timing

2.5.2. Interrupt Control

A block diagram of the interrupt system is shown in Figure 2-12. Each of the interrupt sources is tested for a 0 to 1 transition. If this transition occurs, the corresponding bit in the Interrupt Pending Register, located at 0009H, is set. Since this register can be written to, it is possible to generate software interrupts by setting bits within the register, or remove pending interrupts by clearing the bits in this register. The pending register can be set even if the interrupt is disabled.

Caution must be used when writing to the pending register to clear interrupts. If the interrupt has already been acknowledged when the bit is cleared, a 4 state time "partial" interrupt cycle will occur. This is because the 8096 will have to fetch the next instruction of the normal instruction flow, instead of proceeding with the interrupt processing as it was going to. The effect on the program will be essentially that of an extra NOP. This can be prevented by clearing the bits using a 2 operand immediate logical, as the 8096 holds off acknowledging interrupts during these "read/modify/write" instructions.

Enabling and disabling of individual interrupts is done through the Interrupt Mask Register, located at 0008H. If the bit in the mask register is a 1 then the interrupt is enabled, otherwise it is disabled. Even if an interrupt is masked it may still become pending. It may, therefore, be desirable to clear the pending bit before unmasking an interrupt.

The Interrupt Mask Register is also the low byte of the PSW. All of the interrupts may be enabled and disabled simultaneously by using the "EI" (Enable Interrupt) and "DI" (Disable Interrupt) instructions. EI and DI set and clear PSW.9, the interrupt enable bit, they do not effect the contents of the mask register.

2.5.3. Interrupt Priority Programming

The priority encoder looks at all of the interrupts which are *both pending and enabled,* and selects the one with the highest priority. The priorities are shown in Figure 2-10 (7 is highest, 0 is lowest.) The interrupt generator then forces a call to the location in the indicated vector location. This location would be the starting location of the Interrupt Service Routine (ISR).

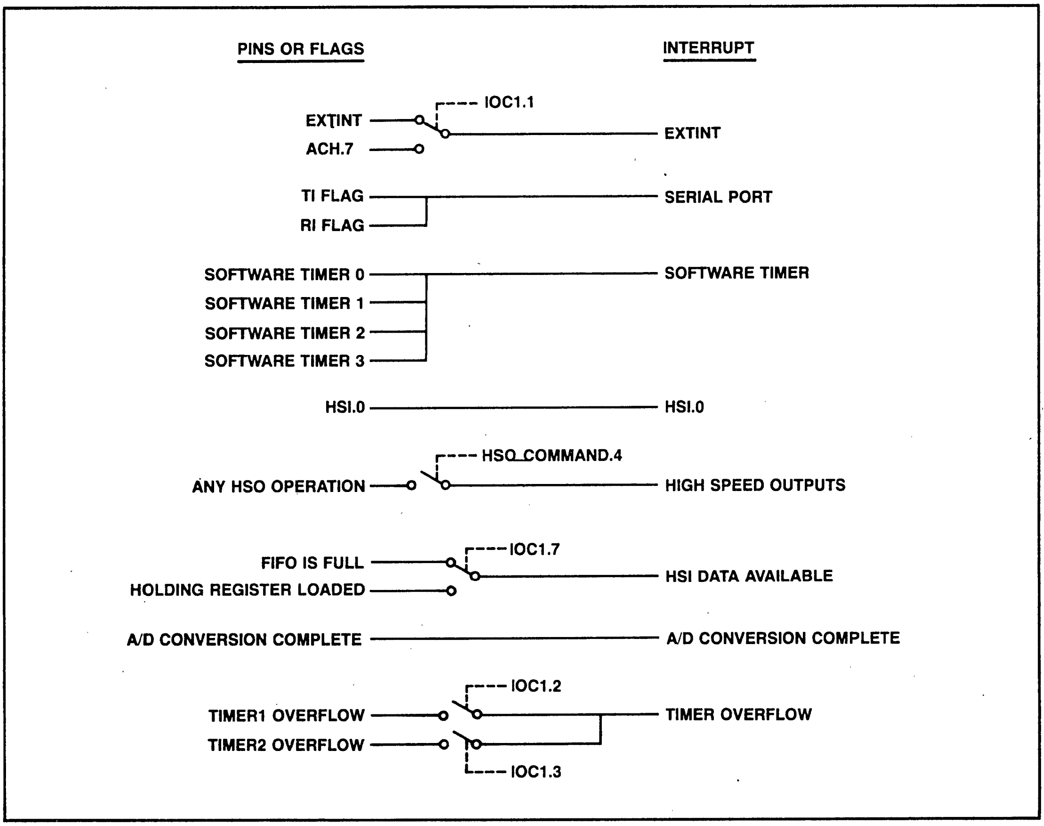


Figure 2-11. All Possible Interrupt Sources

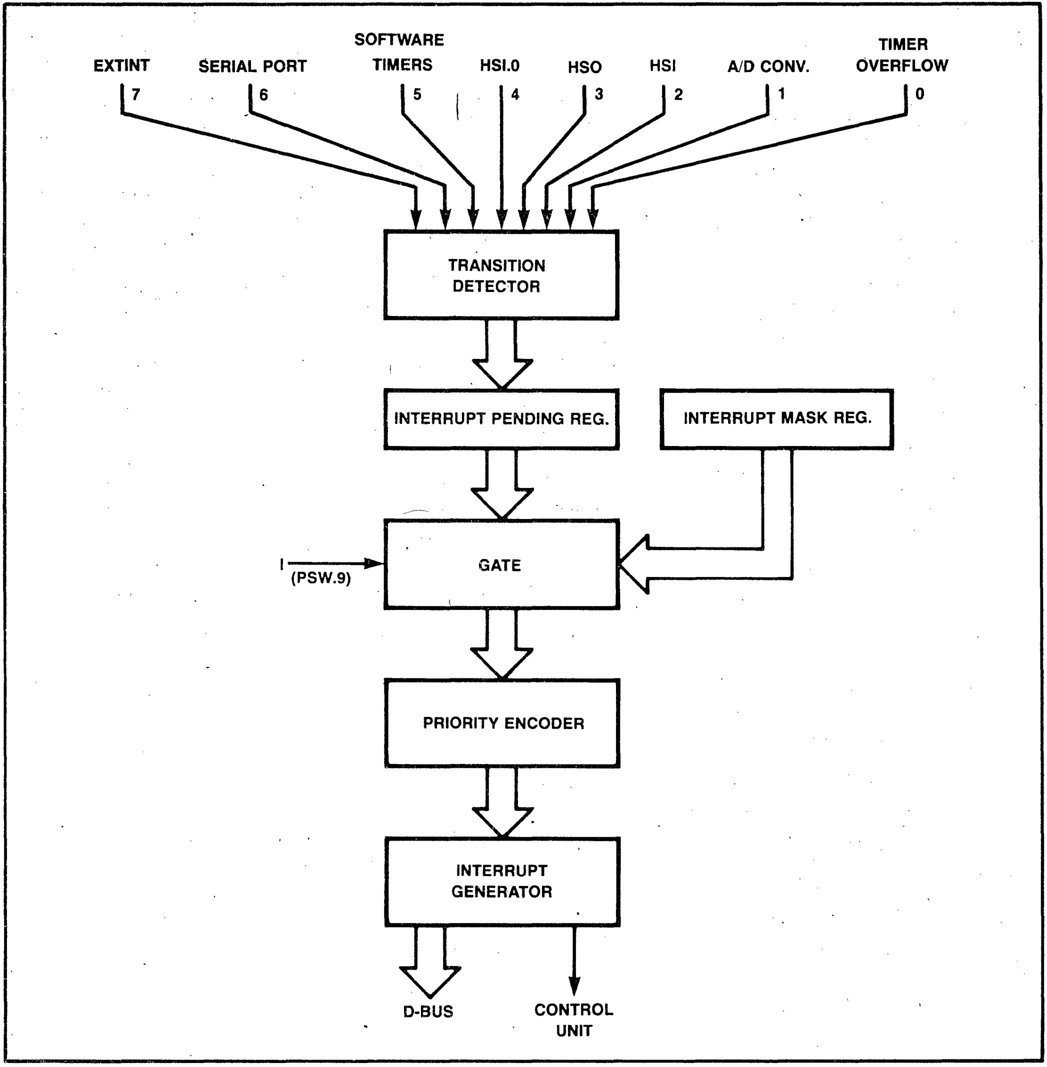


Figure 2-12. Block Diagram of Interrupt System

At least one instruction in the ISR will always be executed before another interrupt can be acknowledged. Usually this instruction is "DI" or "PUSHF" (push flags). The PUSHF instruction pushes the PSW onto the stack and then clears it.

Clearing the PSW disables all interrupts in two ways; by clearing PSW.9, the interrupt enable bit and because the Interrupt Mask Register is located in bits 0 through 7 of the PSW. The interrupts which should be permitted to interrupt this ISR can then be set in the mask register and an "EI" instruction executed.

By selectively determining which interrupts are enabled or disabled within which interrupt service routines, it is possible to configure the interrupt system in any way one would desire. More information on programming the interrupts can be found under software programming of interrupts, section 3.6.

The last two instructions in an ISR are normally a "POPF" (pop flags), which restores the PSW and, therefore, the interrupt mask register, followed by a 'RET', which restores the Program Counter. Execution will then continue from the point at which the call was forced.

2.5.4. Interrupt Timing

Interrupts are not always acknowledged immediately. If the interrupt signal does not occur prior to 4 state-times before the end of an instruction, the interrupt will not be acknowledged until after the next instruction has been executed. This is because an instruction is fetched and prepared for execution a few state times before it is actually executed.

There are 6 instructions which always inhibit interrupts from being acknowledged until after the next instruction has been executed. These instructions are:

EI, DI	— Enable and Disable Interrupts
POPF, PUSHF	— Pop and Push Flags
SIGND	— Prefix to perform signed multiply and divide
TRAP	— Software interrupt

When an interrupt is acknowledged, a call is forced to the location indicated by the specified interrupt vector. This call occurs after the completion of the instruction in process, except as noted above. The procedure of getting the vector and forcing the call requires 21 state times. If the stack is in external RAM an additional 3 state times are required.

The maximum number of state times required from the time an interrupt is generated (not acknowledged) until the 8096 begins executing code at the desired location is the time of the longest instruction, NORML (Normalize — 43 state times), plus the 4 state times prior to the end of the previous instruction, plus the response time (21 to 24 state times). Therefore, the maximum response time is 71 (43+4+24) state times. This does not include the 12 state times required for PUSHF if it is used as the first instruction in the interrupt routine or additional latency caused by having the interrupt masked or disabled.

Interrupt latency time can be reduced by careful selection of instructions in areas of code where interrupts are expected. Using 'EI' followed immediately by a long instruction (e.g. MUL, NORML, etc.) will increase the maximum latency by 4 state times, as an interrupt cannot occur between EI and the instruction following EI. The "DI", "PUSHF", "POPF" and "TRAP" instructions will also cause the same situation. Typically the PUSHF, POPF and TRAP instructions would only effect latency when one interrupt routine is already in process, as these instructions are seldom used at other times.

2.6. TIMERS

Two 16-bit timers are available for use on the 8096. The first is designated "Timer 1", the second, "Timer 2". Timer 1 is used to synchronize events to real time, while Timer 2 can be clocked externally and synchronizes events to external occurences.

2.6.1. Timer 1

Timer 1 is clocked once every eight state times and can be cleared only by executing a reset. The only other way to change its value is by writing to 000CH but this is a test mode which sets both timers to 0FFFXH and should not be used in programs.

2.6.2. Timer 2

Timer 2 can be incremented by transitions (one count each transition, rising *and* falling) on either T2CLK or HSI.1. The multiple functionality of the timer is determined by the state of I/O Control Register 0, bit 7 (IOC0.7). To ensure that all CAM entries are checked each count of Timer 2, the maximum transition speed is limited to once per eight state times. Timer 2 can be cleared by: executing a reset, by setting IOC0.1, by triggering HSO channel 0EH, or by pulling T2RST or HSI.0 high. The HSO and CAM are described in section 2.8. IOC0.3 and IOC0.5 control the resetting of Timer 2. Figure 2-13 shows the different ways of manipulating Timer 2.

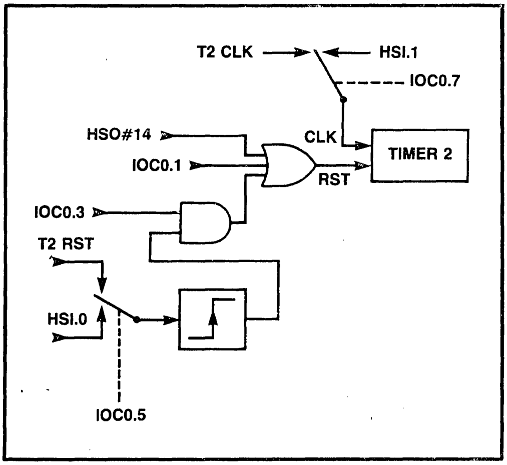


Figure 2-13. Timer 2 Clock and Reset Options

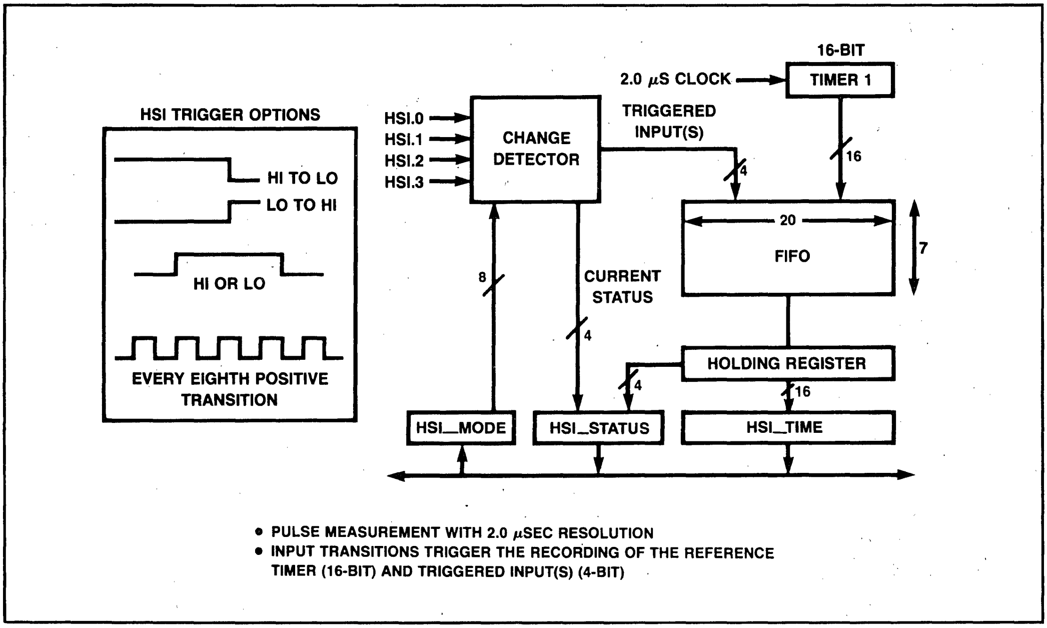
2.6.3. Timer Interrupts

Both Timer 1 and Timer 2 can be used to trigger a timer overflow interrupt and set a flag in the I/O Status Register 1 (IOS1). The interrupts are controlled by IOC1.2 and IOC1.3 respectively. The flags are set in IOS1.6 and IOS1.7, respectively.

Caution must be used when examining the flags, as any access (including Compare and Jump on Bit) of IOS1 clears the whole byte, including the software timer flags. It is, therefore, recommended to write the byte to a temporary register before testing bits. The general enabling and disabling of the timer interrupts are controlled by the Interrupt Mask Register bit 0. In all cases, setting a bit enables a function, while clearing a bit disables it.

2.6.4. Timer Related Sections

The High Speed I/O unit is coupled to the timers in that the HSI records the value on Timer 1 when transitions



- PULSE MEASUREMENT WITH 2.0 μSEC RESOLUTION
- INPUT TRANSITIONS TRIGGER THE RECORDING OF THE REFERENCE TIMER (16-BIT) AND TRIGGERED INPUT(S) (4-BIT)

Figure 2-14. High Speed Input Unit

occur and the HSO causes transitions to occur based on values of either Timer 1 or Timer 2. The Baud rate generator can use the T2CLK pin as input to its counter. A complete listing of the functions of IOS1, IOC0, and IOC1 are in section 2.13.

2.7. HIGH SPEED INPUTS

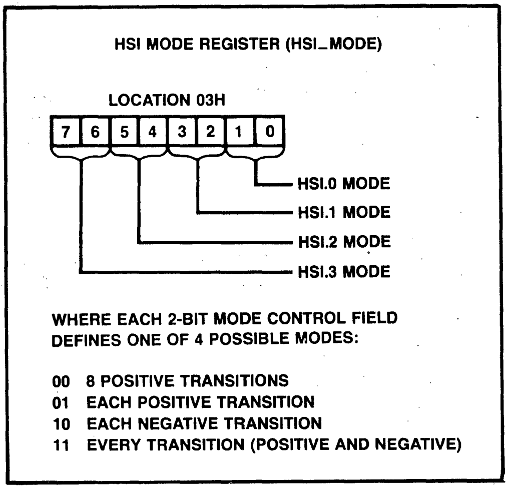
The High Speed Input Unit (HSI), can be used to record the time at which an event occurs with respect to Timer 1. There are 4 lines (HSI.0 through HSI.3) which can be used in this mode and up to a total of 8 events can be recorded. HSI.2 and HSI.3 share pins with HSO.4 and HSO.5. The I/O Control Registers (IOC0 and IOC1) are used to determine the functions of these pins. A block diagram of the HSI unit is shown in Figure 2-14.

2.7.1. HSI Modes

There are 4 possible modes of operation for each of the HSI. The HSI mode register is used to control which pins will look for what type of events. The 8-bit register is set up as shown in Figure 2-15.

High and low levels each need to be held for at least 1 state time to ensure proper operation. The maximum input speed is 1 event every 8 state times except when the 8 transition mode is used, in which case it is 1 transition per state time.

The HSI lines can be individually enabled and disabled using bits in IOC0, at location 0015H. Figure 2-16 shows the bit locations which control the HSI pins. If the pin is disabled, transitions will not be entered in the FIFO.



WHERE EACH 2-BIT MODE CONTROL FIELD DEFINES ONE OF 4 POSSIBLE MODES:

00	8 POSITIVE TRANSITIONS
01	EACH POSITIVE TRANSITION
10	EACH NEGATIVE TRANSITION
11	EVERY TRANSITION (POSITIVE AND NEGATIVE)

Figure 2-15. HSI Mode Register Diagram

2.7.2. HSI FIFO

When an HSI event occurs, a 7x20 FIFO stores the 16 bits of Timer 1 and the 4 bits indicating the state of the

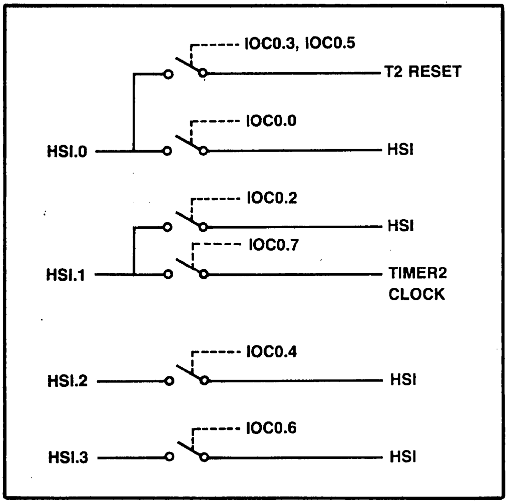


Figure 2-16. IOC0 Control of HSI Pin Functions

4 HSI lines at the recorded timer value. It can take up to 8 state times for this information to reach the holding register. When the FIFO is full, one additional event can be stored by considering the holding register part of the FIFO. If the FIFO and holding register are full any additional events will not be recorded.

2.7.3. HSI Interrupts

Interrupts can be generated from the HSI unit in one of two ways, determined by IOC1.7. If the bit is a 0, then an interrupt will be generated every time a value is loaded into the holding register. If it is a 1, an interrupt will only be generated when the FIFO, (independent of the holding register), has six or seven entries in it. Since all interrupts are rising edge triggered, if IOC1.7 = 1, the processor will not be re-interrupted until the FIFO first contains 5 or less records, then contains six or more.

2.7.4. HSI Status

Bits 6 and 7 of the I/O Status register 1 (IOS1) indicate the status of the HSI FIFO. If bit 6 is a 1, the FIFO contains at least seven entries. If bit 7 is a 1, the FIFO contains at least 1 entry. The FIFO may be read after verifying that it contains valid data. Caution must be used when reading or testing bits in IOS1, as this action clears the entire byte, including the software and hardware timer overflow flags. It is best to store the byte and then test the stored value.

Reading the HSI is done in two steps. First, the HSI Status register is read to obtain the current state of the HSI pins and their state at the recorded time. Second, the HSI Time register is read. Reading the Time register unloads one word of the FIFO, so if the Time register is read before the Status register, the information in the Status register will be lost. The HSI Status register is at location 06H and the HSI Time registers are in locations 04H and 05H.

It should be noted that many of the Status register conditions are changed by a reset, see section 2.15.2. A complete listing of the functions of IOS0, IOS1, and IOC1 can be found in section 2.13.

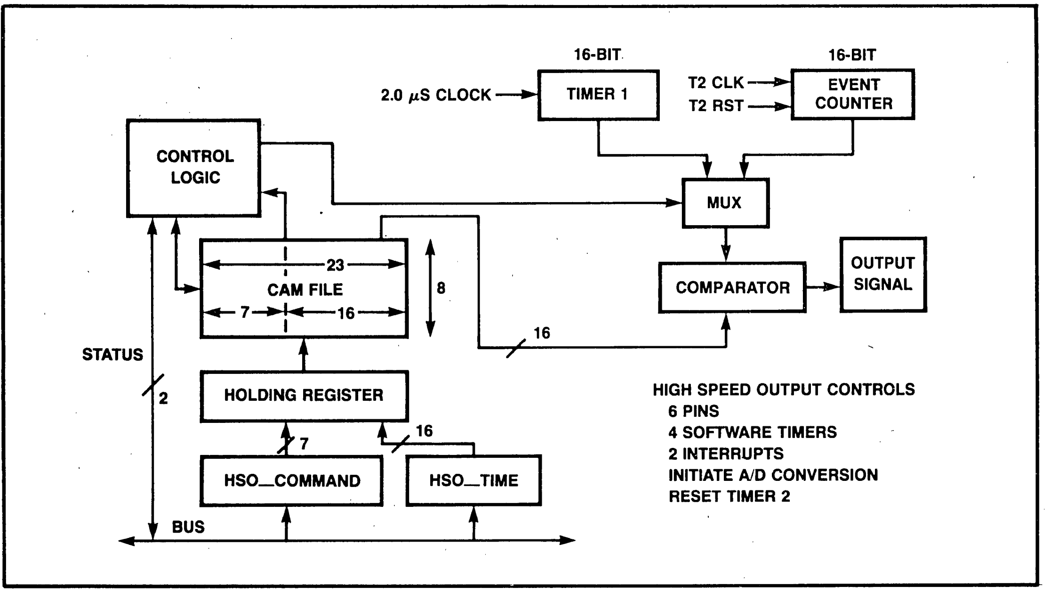


Figure 2-17. High Speed Output Unit

2.8. HIGH SPEED OUTPUTS

The High Speed Output unit (HSO) is used to trigger events at specific times with minimal CPU overhead. These events include: starting an A to D conversion, resetting Timer 2, setting 4 software flags, generating interrupts, and switching up to 6 output lines. Up to 8 events can be pending at any one time.

2.8.1. HSIO Shared Pins

Two of the 6 output lines (HSO.0 through HSO.5) are shared with the High Speed Input (HSI) lines. HSO.4 and HSO.5 are shared with HSI.2 and HSI.3, respectively. Bits 4 and 6 of the I/O Control Register 1 (IOC1) are used to enable HSO.4 and HSO.5 as outputs.

2.8.2. HSIO CAM

A block diagram of the HSO unit is shown in Figure 2-17. The Content Addressable Memory (CAM) file is the center of control. One CAM register is compared with a time value every state time. Therefore, it takes 8 state times to compare all CAM registers with a timer.

Each CAM register is 23 bits wide. Sixteen bits specify the time at which the action is to be carried out and 7 bits specify both the nature of the action and whether Timer 1 or Timer 2 is the reference. The format of the command to the HSO unit shown in Figure 2-18.

To enter a command into the CAM file, write the 7-bit "Command Tag" into location 0006H followed by the time at which the action is to be carried out into word address 0004H. Writing the time value loads the HSO Holding Register with both the time and the last written command tag. The command does not actually enter the CAM file until an empty CAM register becomes available.

Care must be taken when writing the command tag for the HSO. If an interrupt occurs during the time between writing the command tag and loading the time value, and the interrupt service routine writes to the HSO time register, the command tag will be loaded into the CAM with the time value from the interrupt routine. One way of avoiding this problem would be to disable interrupts when writing commands and times to the HSO unit.

2.8.3. HSO Status

Before writing to the HSO, it is desirable to ensure that the Holding Register is empty. If it is not, writing to the HSO will overwrite the value in the Holding Register. I/O Status Register 0 (IOSO) bits 6 and 7 indicate the status of the HSO unit. This register is described in section 2.13.4. If IOS0.6 equals 0, the holding register is empty and at least one CAM register is empty. If IOS0.7 equals 0, the holding register is empty.

One location in the CAM file is checked each state-time. Thus, it takes 8 state-times for the Holding Register to have had access to all 8 CAM registers. Similarly, it takes 8 state-times for the comparator to have had access to all 8 CAM registers. This defines the time-resolution of the HSO unit to be 8 state-times (2.0 μsec, if the oscillator frequency is 12 MHz). Note that the comparator does not look at the holding register, so instructions in the holding register do not execute.

2.8.4. Clearing The HSO

All 8 CAM locations of the HSO are compared before any action is taken. This allows a pending event to be cancelled by simply writing the opposite event to the CAM. However, once an entry is placed in the CAM, it cannot be removed until either the specified timer matches the written value or the chip is reset.

Timer 1 is incremented only once every 8 state-times. When it is being used as the reference timer for an HSO action, the comparator has a chance to look at all 8 CAM registers before Timer 1 changes its value. Following the same reasoning, Timer 2 has been synchronized to allow it to change at a maximum rate of once per 8 state-times. Timer 2 increments on both edges of the input signal.

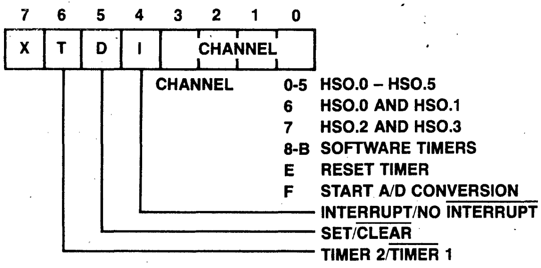


Figure 2-18. HSO Command Tag Format

If Timer 2 is being used as the reference timer, the user must ensure that it is not cleared before all references to it in the CAM have been recognized. If this occurs, the unrecognized references will remain in the CAM, blocking further entries to those CAM registers until the part is reset or the references are recognized.

2.8.5. Software Timers

The HSO can be programmed to generate interrupts at preset times. Up to four such "Software Timers" can be in operation at a time. As each preprogrammed time is reached, the HSO unit generates a Software Timer interrupt. The interrupt service routine can then examine I/O Status register 1 (IOS1) to determine which software timer expired and caused the interrupt. When the HSO resets Timer 2 or starts an A to D conversion, it can also be programmed to generate a software timer interrupt but there is no flag to indicate that this has occurred.

Each read or test of any bit in IOS1 will clear the whole byte. Be certain to save the byte before testing it unless you are only concerned with 1 bit.

A complete listing of the functions of IOS0, IOS1, and IOC1 can be found in section 2.13. The Timers are described in section 2.6 and the HSI is described in section 2.7.

2.9. ANALOG INPUTS

The A to D converter on the 8096 provides a 10-bit result on one of 8 input channels. Conversion is done using successive approximation with a result equal to the ratio of the input voltage divided by the analog supply voltage. If the ratio is 1.00, then the result will be all ones. The A/D converter is available on the 8097, 8397, 8095 and 8395 members of the MCS®-96 family.

2.9.1. A/D Accuracy

Each conversion requires 168 state-times ($42\mu S$ at 12 MHz) independent of the accuracy desired or value of input voltage. The input voltage must be in the range of 0 to VREF, the analog reference and supply voltage. For proper operation, VREF (the reference voltage and analog power supply) must be held at VCC ± 0.3V with $VREF = 5.0 \pm 0.5V$. The A/D result is calculated from the formula:

$$1023 \times (\text{input voltage-AVGND}) / (\text{VREF-AVGND})$$

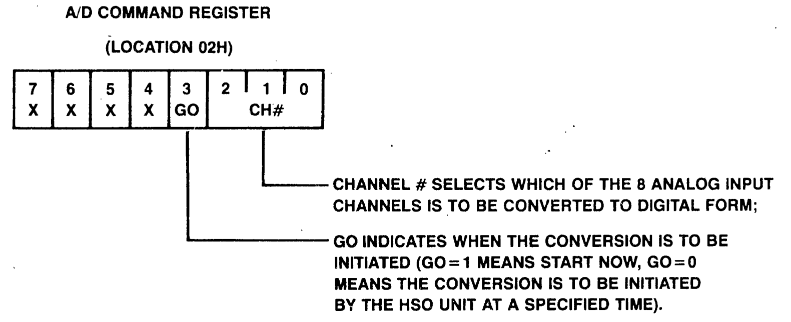
It can be seen from this formula that changes in VREF or AVGND effect the output of the converter. This can be advantageous if a ratiometric sensor is used since these sensors have an output that can be measured as a proportion of VREF.

If high absolute accuracy is needed it may be desirable to use a separate power supply, or power traces, to operate the A/D converter. There is no sample and hold circuit internal to the chip, so the input voltage must be held constant for the entire 168 state times. Examples of connecting the A/D converter to various devices are given in section 4.3.

2.9.2. A/D Commands

Analog signals can be sampled by any one of the 8 analog input pins (ACH0 through ACH7) which are shared with Port 0. ACH7 can also be used as an external interrupt if IOC1.1 is set (see section 2.5). The A/D Command Register, at location 02H, selects which channel is to be converted and whether the conversion should start immediately or when the HSO (Channel #14) triggers it. A to D commands are formatted as shown in Figure 2-19.

The command register is double buffered so it is possible



CHANNEL # SELECTS WHICH OF THE 8 ANALOG INPUT CHANNELS IS TO BE CONVERTED TO DIGITAL FORM;

GO INDICATES WHEN THE CONVERSION IS TO BE INITIATED (GO=1 MEANS START NOW, GO=0 MEANS THE CONVERSION IS TO BE INITIATED BY THE HSO UNIT AT A SPECIFIED TIME).

Figure 2-19. A/D Command Register

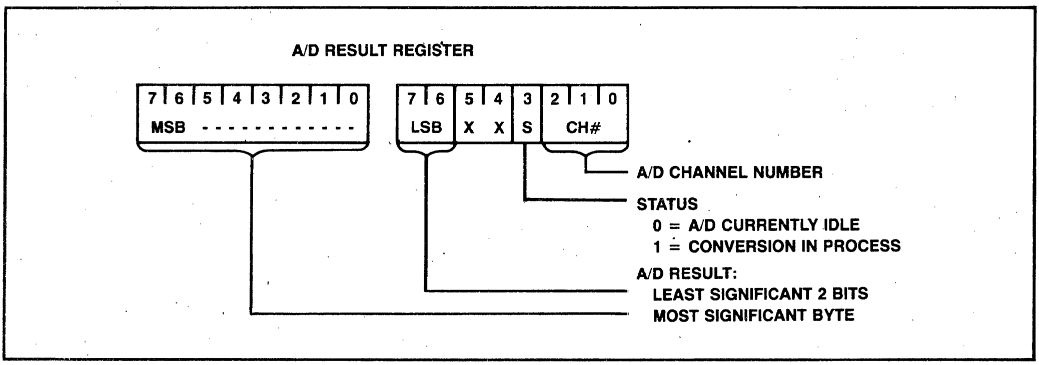


Figure 2-20. A/D Result Register

to write a command to start a conversion triggered by the HSO while one is still in progress. Care must be taken when this is done since if a new conversion is started while one is already in progress, the conversion in progress is cancelled and the new one is started. When a conversion is started, the result register is cleared. For this reason the result register must be read before a new conversion is started or data will be lost.

2.9.3. A/D Results

Results of the analog conversions are read from the A/D Result Register at locations 02H and 03H. Although these addresses are on a word boundary, they must be read as individual bytes. Information in the A/D Result register is formatted as shown in Figure 2-20. Note that the status bit may not be set until 8 state times after the go command. Information on using the HSO is in section 2.8.

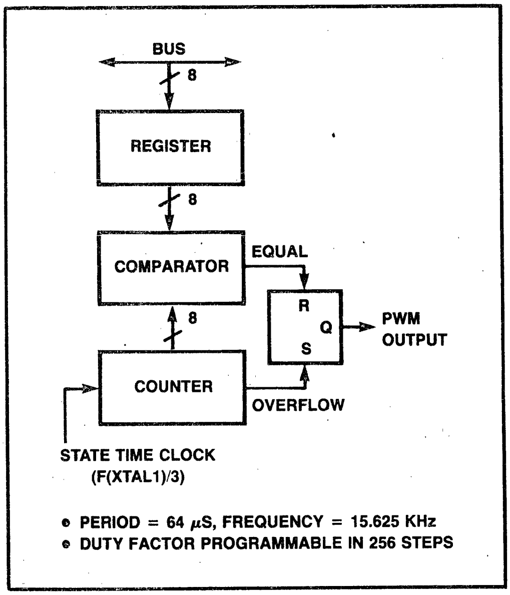

BUS
8
REGISTER
8
COMPARATOR
EQUAL
R
Q
S
PWM OUTPUT
8
COUNTER
OVERFLOW
STATE TIME CLOCK (F(XTAL1)/3)
• PERIOD = 64 μS, FREQUENCY = 15.625 KHz
• DUTY FACTOR PROGRAMMABLE IN 256 STEPS

Figure 2-21. Pulse Width Modulated (D/A) Output

2.10. PULSE WIDTH MODULATION OUTPUT (D/A)

Digital to analog conversion can be done with the pulse width modulation output; a block diagram of the circuit is shown in Figure 2-21. The 8-bit counter is incremented every state time. When it equals 0, the PWM output is set to a one. When the counter matches the value in the PWM register, the output is switched low. When the counter overflows, the output is once again switched high. A typical output waveform is shown in Figure 2-22. Note that when the PWM register equals 00, the output is always low.

The output waveform is a variable duty cycle pulse which repeats every 256 state times (64 μS at 12MHz). Changes in the duty cycle are made by writing to the PWM register at location 17H. There are several types of motors which require a PWM waveform for most efficient operation. Additionally, if this waveform is integrated it will produce a DC level which can be changed in 256 steps by varying the duty cycle.

Details about the hardware required for smooth, accurate D/A conversion can be found in section 4.3.2. Typically, some form of buffer and integrator are needed to obtain the most usefulness from this feature.

The PWM output shares a pin with Port 2, pin 5 so that these two features cannot be used at the same time. IOC1.0 equal to 1 selects the PWM function instead of the standard port function. More information on IOC1 is in section 2.13.3.

2.11. SERIAL PORT

The serial port is compatible with the MCS-51 serial port. It is full duplex, meaning it can transmit and receive simultaneously. It is also receive-buffered, meaning it can commence reception of a second byte before a previously received byte has been read from the receive register. The serial port registers are both accessed at location 07H. A write to this location accesses the transmit register, and a read accesses a physically separate receive register.

The serial port can operate in 4 modes (explained below). Selection of these modes is done through the Serial Port Control register at location 11H. Use of this register will be described after defining the possible modes of operation. In addition to setting the proper mode, it is necessary to enable the TXD output by setting IOC1.5 high before the serial port is used.

2.11.1. Serial Port Modes

MODE 0

Mode 0 is a shift register mode. The 8096 outputs a train of 8 shift pulses to an external shift register to clock 8 bits of data into or out of the register from or to the 8096. Serial data enters and exits the 8096 through RXD. TXD outputs the shift clock. 8 bits are transmitted or received, LSB first. A timing diagram of this mode is shown in Figure 2-23. This mode is useful as an I/O expander in which application external shift registers can be used as additional parallel I/O ports. An example of using the port in this mode is given in section 4.5.

MODE 1

10-bit frames are transmitted through TXD, and received through RXD: a start bit (0), 8 data bits (LSB first), and a stop bit (1). This mode is the one commonly used for CRT terminals. The data frame for Mode 1 is shown in Figure 2-24.

MODE 2

11-bit frames are transmitted through TXD and received through RXD: a start bit (0), 8 data bits (LSB first), a programmable 9th data bit, and a stop bit (1). On transmit, the 9th data bit can be assigned the value of 0 or 1. On receive, the serial port interrupt is not activated unless the received 9th data bit is 1. This mode is commonly used along with mode 3 in a multiprocessor environment.

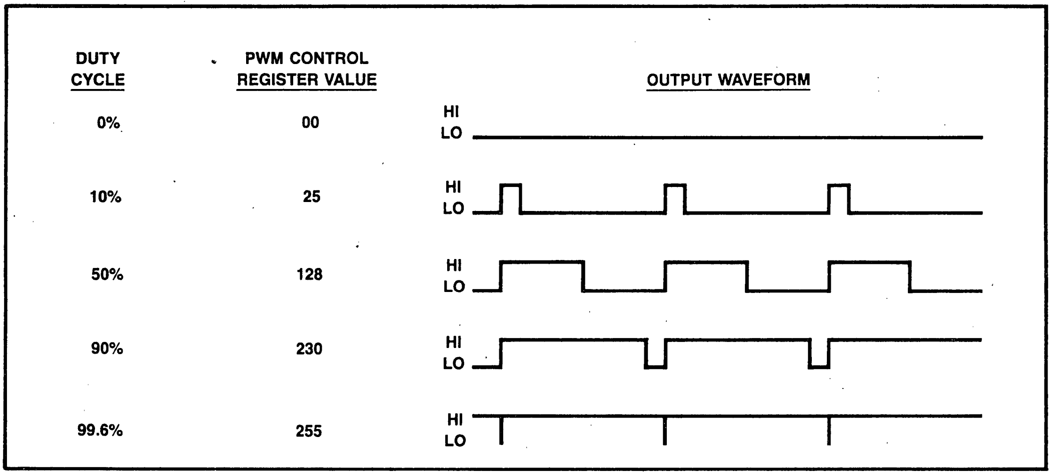


Figure 2-22. Typical PWM Outputs

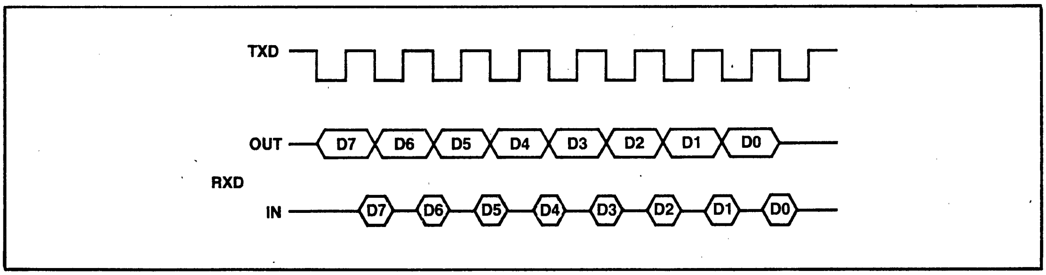


Figure 2-23. Serial Port Mode 0 Timing

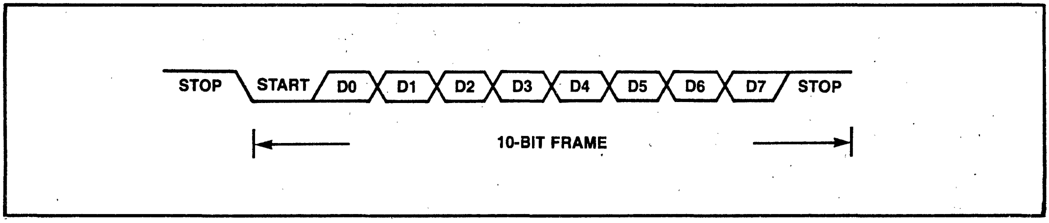


Figure 2-24. Serial Port Frame — Mode 1

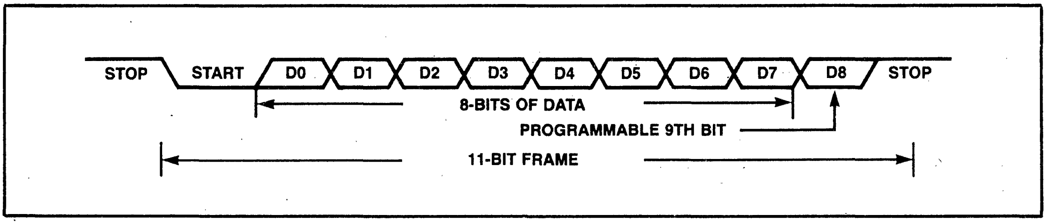


Figure 2-25. Serial Port Frame Modes 2 and 3

MODE 3

11-bit frames are transmitted through TXD and received through RXD: a start bit (0), 8 data bits (LSB first), a programmable 9th data bit, and a stop bit (1). On transmit, the 9th data bit can be assigned the value of 0 or 1. On receive, the received 9th data bit is stored and the serial port interrupt is activated regardless of its value. The data frame for Modes 2 and 3 is shown in Figure 2-25.

2.11.2. Multiprocessor Communications

Mode 2 and 3 are provided for multiprocessor communications. In mode 2 if the received 9th data bit is not 1, the serial port interrupt is not activated. The way to use this feature in multiprocessor systems is described below.

When the master processor wants to transmit a block of data to one of several slaves, it first sends out an address frame which identifies the target slave. An address frame will differ from a data frame in that the 9th data bit is 1 in an address frame and 0 in a data frame. No slave in mode 2 will be interrupted by a data frame. An address frame, however, will interrupt all slaves so that each slave can examine the received byte and see if it is being addressed. The addressed slave switches to mode 3 to receive the coming data frames, while the slaves that were not addressed stay in mode 2 and go on about their business.

2.11.3. Controlling the Serial Port

Control of the Serial Port is done through the Serial Port Control/Status register. The format for the control word is shown in Figure 2-26.

When using Mode 0, it is necessary to set up the port with REN = 0 and then write a one to the REN register before the port will be to properly initialized. In any mode, it is necessary to set IOC1.5 to a 1 to enable the TXD pin. Some examples of the software involved in using the serial port can be found in section 3.8. More information on IOC1 is in section 2.13.3.

2.11.4. Determining Baud Rates

Baud rates in all modes are determined by the contents of a 16-bit register at location 000EH. This register must be loaded sequentially with 2 bytes (least significant byte first). The MSB of this register selects one of two sources for the input frequency to the baud rate generator. If it is a 1, the frequency on the XTAL1 pin is selected, if not, the external frequency from the T2CLK pin is used. It should be noted that the maximum speed of T2CLK is one transition every 2 state times.

The unsigned integer represented by the remaining 15 bits of the baud rate register, plus 1, defines a number B, where B can thus take any value from 1 to 32768. This '1' is not added if the T2CLK is used as the time base. The baud rate for the 4 serial modes is then given by

Mode 0: baud rate = input frequency/(4*B)
Others: baud rate = input frequency/(64*B)

The maximum asynchronous baud rate is 187.5 Kbaud, with a 12 MHz clock.

The unsigned integer must not equal zero if mode 0 is being used.

2.12. I/O PORTS 0, 1, 2, 3, AND 4

There are five 8-bit I/O ports on the 8096. Some of these ports are input only, some output only, some bidirectional and some have alternate functions. Input ports connect to the internal bus through an input buffer. Output ports

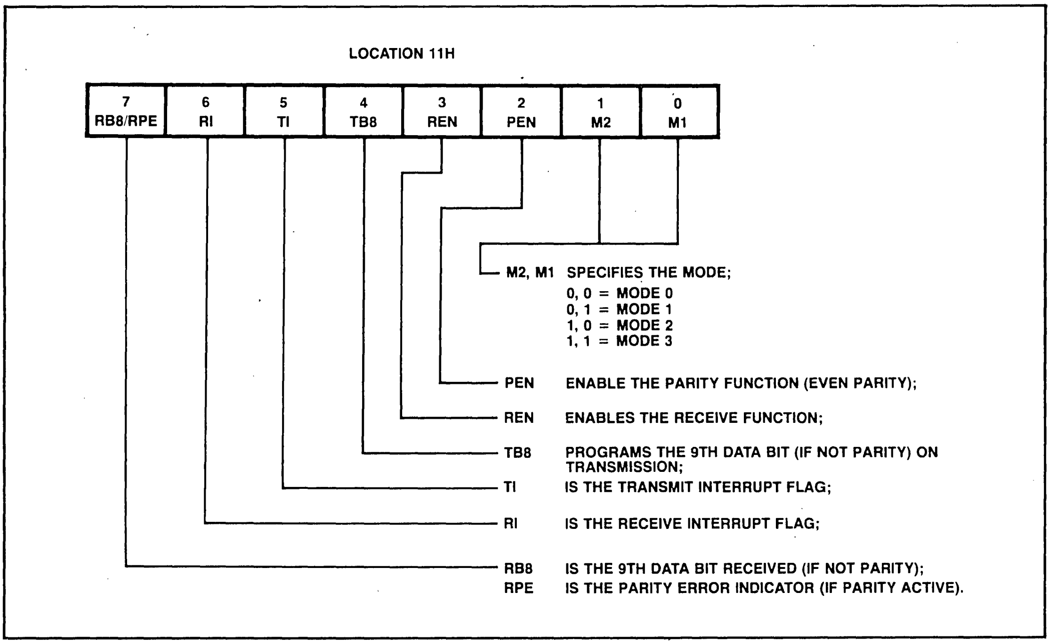

LOCATION 11H
7
RB8/RPE
6
RI
5
TI
4
TB8
3
REN
2
PEN
1
M2
0
M1
M2, M1 SPECIFIES THE MODE;
0, 0 = MODE 0
0, 1 = MODE 1
1, 0 = MODE 2
1, 1 = MODE 3
PEN ENABLE THE PARITY FUNCTION (EVEN PARITY);
REN ENABLES THE RECEIVE FUNCTION;
TB8 PROGRAMS THE 9TH DATA BIT (IF NOT PARITY) ON TRANSMISSION;
TI IS THE TRANSMIT INTERRUPT FLAG;
RI IS THE RECEIVE INTERRUPT FLAG;
RB8 IS THE 9TH DATA BIT RECEIVED (IF NOT PARITY);
RPE IS THE PARITY ERROR INDICATOR (IF PARITY ACTIVE).

Figure 2-26. Serial Port Control/Status Register

connect through an output buffer to an internal register that holds the output bits. Bidirectional ports consist of an internal register, an output buffer, and an input buffer.

When an instruction accesses a bidirectional port as a source register, the question often arises as to whether the value that is brought into the CPU comes from the internal port register or from the port pins through the input buffer. In the 8096, the value always comes from the port pins, never from the internal register.

2.12.1. Port 0

Port 0 is an input only port which shares its pins with the analog inputs to the A/D Converter. One can read Port 0 digitally, and/or, by writing the appropriate control bits to the A/D Command Register, select one of the lines of this port to be the input to the A/D Converter. While a conversion is in process, the impedance of the selected line is lower than normal. See the data sheet for the specific values.

2.12.2. Port 1

Port 1 is a quasi-bidirectional I/O port. "Quasi-bidirectional" means the port pin has a weak internal pullup that is always active and an internal pulldown which can either be on (to output a 0) or off (to output a 1). If the internal pulldown is left off (by writing a 1 to the pin), the pin's logic level can be controlled by an external pulldown which can either be on (to input a 0) or off (to input a 1). From the user's point of view the main distinction is that a quasi-bidirectional input will source current while being externally held low and will pull itself high if left alone.

In parallel with the weak internal pullup, is a much stronger internal pullup that is activated for one state time when the pin is internally driven from 0 to 1. This is done to speed up the 0-to-1 transition time.

2.12.3. Port 2

Port 2 is a multi-functional port. Six of its pins are shared with other functions in the 8096, as shown below.

Port	Function	Alternate Function	Controlled by
P2.0	output	TXD (serial port transmit)	IOC1.5
P2.1	input	RXD (serial port receive)	N/A
P2.2	input	EXTINT (external interrupt)	IOC1.1
P2.3	input	T2CLK (Timer 2 input)	IOC0.7
P2.4	input	T2RST (Timer 2 reset)	IOC0.3
P2.5	output	PWM (pulse-width modulation)	IOC1.0
P2.6	quasi-bidirectional		
P2.7	quasi-bidirectional		

2.12.4. Ports 3 and 4

Ports 3 and 4 have two functions. They are either bidirectional ports with open-drain outputs or System Bus pins which the memory controller uses when it is accessing off-chip memory. Strong internal pullups are used during external memory read or write cycles when the pins are used as address or data outputs. At any other time, the internal pullups are disabled. The port pins and their system bus functions are shown below:

Port Pin	System Bus Function
P3.0	AD0
P3.1	AD1
P3.2	AD2
P3.3	AD3
P3.4	AD4
P3.5	AD5
P3.6	AD6
P3.7	AD7
P4.0	AD8
P4.1	AD9
P4.2	AD10
P4.3	AD11
P4.4	AD12
P4.5	AD13
P4.6	AD14
P4.7	AD15

2.13. STATUS AND CONTROL REGISTERS

2.13.1. I/O Control Registers

There are two I/O Control registers, IOC0 and IOC1. IOC0 controls Timer 2 and the HSI lines. IOC1 controls some pin functions, interrupt sources and 2 HSO pins.

2.13.2. IOC0

IOC0 is located at 0015H. The four HSI lines can be enabled or disabled to the HSI unit by setting or clearing bits in IOC0. Timer 2 functions including clock and reset sources are also determined by IOC0. The control bit locations are shown in Figure 2-27.

2.13.3. IOC1

IOC1 is used to select some pin functions and enable or disable some interrupt sources. Its location is 0016H. Port pin P2.5 can be selected to be the PWM output instead of a standard output. The external interrupt source can be selected to be either EXTINT (same pin as P2.2) or Analog Channel 7 (ACH7, same pin as P0.7). Timer 1 and Timer 2 overflow interrupts can be individually enabled or disabled. The HSI interrupt can be selected to activate either when there is 1 FIFO entry or 7. Port pin P2.0 can be selected to be the TXD output. HSO.4 and HSO.5 can be enabled or disabled to the HSO unit. More information on interrupts is available in section 2.5. The positions of the IOC1 control bits are shown in Figure 2-28.

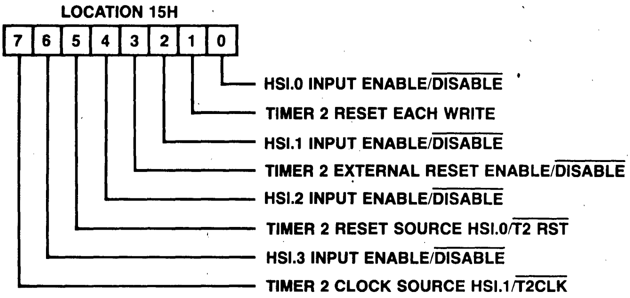

LOCATION 15H
7
6
5
4
3
2
1
0
HSI.0 INPUT ENABLE/DISABLE
TIMER 2 RESET EACH WRITE
HSI.1 INPUT ENABLE/DISABLE
TIMER 2 EXTERNAL RESET ENABLE/DISABLE
HSI.2 INPUT ENABLE/DISABLE
TIMER 2 RESET SOURCE HSI.0/T2 RST
HSI.3 INPUT ENABLE/DISABLE
TIMER 2 CLOCK SOURCE HSI.1/T2CLK

Figure 2-27. I/O Control Register 0 (IOC0)

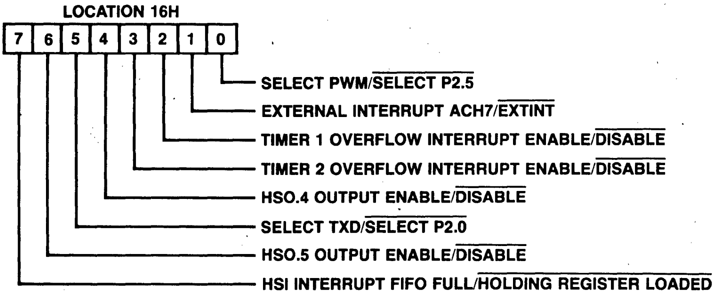

LOCATION 16H
7
6
5
4
3
2
1
0
SELECT PWM/SELECT P2.5
EXTERNAL INTERRUPT ACH7/EXTINT
TIMER 1 OVERFLOW INTERRUPT ENABLE/DISABLE
TIMER 2 OVERFLOW INTERRUPT ENABLE/DISABLE
HSO.4 OUTPUT ENABLE/DISABLE
SELECT TXD/SELECT P2.0
HSO.5 OUTPUT ENABLE/DISABLE
HSI INTERRUPT FIFO FULL/HOLDING REGISTER LOADED

Figure 2-28. I/O Control Register 1 (IOC1)

2.13.4. I/O Status Register 0

There are two I/O Status registers, IOS0 and IOS1. IOS0, located at 0015H, holds the current status of the HSO lines and CAM. The status bits of IOS0 are shown in Figure 2-29.

2.13.5. I/O Status Register 1

IOS1 is located at 0016H. It contains status bits for the timers and the HSI. Every access of this register clears all of the timer overflow and timer expired bits. It is, therefore, important to first store the byte in a temporary location before attempting to test any bit unless only one bit will ever be of importance to the program. The status bits of IOS1 are shown in Figure 2-30.

2.14. WATCHDOG TIMER

This feature is provided as a means of graceful recovery from a software upset. If the software fails to reset the watchdog at least every 64K state times, a hardware reset will be initiated.

The software can be designed so that the watchdog times out if the program does not progress properly. The watchdog will also time-out if the software error was due to ESD (Electrostatic Discharge) or other hardware related problems. This prevents the controller from having a malfunction for longer than 16 mS if a 12 MHz oscillator is used.

The watchdog timer is a 16-bit counter which is incremented every state time. When it overflows it pulls down the $\overline{\text{RESET}}$ pin for at least two state times, resetting the 8096 and any other chips connected to the reset line. To prevent the timer from overflowing and resetting the system, it must be cleared periodically. Clearing the timer is done by writing a "01EH" followed by an "0E1H" to the WDT register at location 000AH.

Use of a large reset capacitor on the $\overline{\text{RESET}}$ pin will increase the length of time required for a watchdog initiated reset. This is because the capacitor will keep the $\overline{\text{RESET}}$ pin from responding immediately to the internal pull-ups and pull-downs. A large capacitor on the $\overline{\text{RESET}}$ pin may also interfere with the reset of other parts connected to the $\overline{\text{RESET}}$ pin. Under some circumstances, it may be desirable to use an open collector circuit. See section 4.1.4.

2.14.1. Disabling The Watchdog

During program development, the watchdog can be disabled by holding the $\overline{\text{RESET}}$ pin at 2.0 to 2.5 volts. Voltages over 2.5 volts on the pin could quickly damage the part. Even at 2.5 volts, using this technique for other than

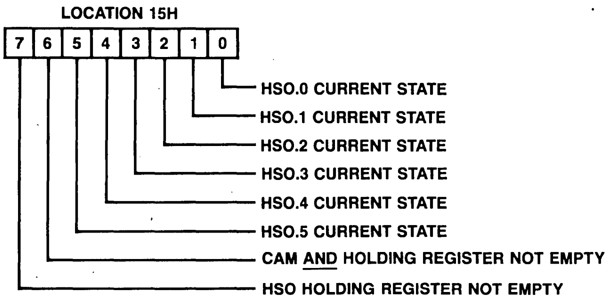


Figure 2-29. HSIO Status Register 0 (IOS0)

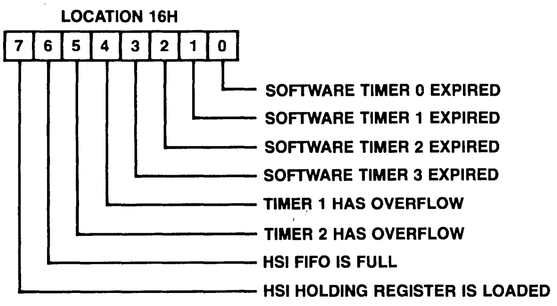


Figure 2-30. HSIO Status Register 1 (IOS1)

debugging purposes is not recommended, as it may effect long term reliability. It is further recommended that any part used in this way for more than a few minutes, not be used in production versions of products.

2.15. RESET

2.15.1. Reset Signal

As with all processors, the 8096 must be reset each time the power is turned on. To complete a reset, the $\overline{\text{RESET}}$ pin must be held active (low) for at least 2 state times after VCC, the oscillator, and the back bias generator have stabilized (~1.0 milliseconds). Then when $\overline{\text{RESET}}$ is brought high again, the 8096 executes a reset sequence that takes 10 state times. (It initializes some registers, clears the PSW and jumps to address 2080H.)

The 8096 can be reset using a capacitor, 1-shot, or any other method capable of providing a pulse of at least 2 state times longer than required for VCC and the oscillator to stabilize.

For best functionality, it is suggested that the reset pin be pulled low with an open collector device. In this way, several reset sources can be wire ored together. Remember, the $\overline{\text{RESET}}$ pin itself can be a reset source (see section 2.14). Details of hardware suggestions for reset can be found in section 4.1.4.

2.15.2. Reset Status

The I/O lines of the 8096 will be in their reset state within 2 state times after reset is low, with VCC and the oscillator stabilized. Prior to that time, the status of the I/O lines is indeterminate. After the 10 state time reset sequence, the Special Function Registers will be set as follows:

SFR	reset value
Port 1	11111111B
Port 2	110XXXX1B
Port 3	11111111B
Port 4	11111111B
PWM Control	00H
Serial Port (Transmit)	undefined
Serial Port (Receive)	undefined
Baud Rate Register	undefined
Serial Control/Status	undefined
A/D Command	undefined
A/D Result	undefined
Interrupt Pending	undefined
Interrupt Mask	00000000B
Timer 1	0000H
Timer 2	0000H
Watchdog Timer	0000H
HSI Mode	11111111B
HSI Status	undefined
IOS0	00000000B
IOS1	00000000B
IOC0	X0X0X0X0B
IOC1	X0X0XXX1B

Other conditions following a reset are:

Register	reset value
HSI FIFO	empty
HSO CAM	empty
HSO lines	00000B
PSW	0000H
Stack Pointer	undefined
Program Counter	2080H

It is important to note that the Stack Pointer and Interrupt Pending Register are undefined, and need to be initialized in software. The Interrupts are disabled by both the mask register and PSW.9 after a reset.

2.15.3. Reset Sync Mode

The $\overline{\text{RESET}}$ line can be used to start the 8096 at an exact state time to provide for synchronization of test equipment and multiple chip systems. $\overline{\text{RESET}}$ is active low. To synchronize parts, $\overline{\text{RESET}}$ is brought high on the rising edge of XTAL1. Complete details on synchronizing parts can be found in section 4.1.5.

It is very possible that parts which start in sync may not stay that way. The best example of this would be when a "jump on I/O bit" is being used to hold the processor in a loop. If the line changes during the time it is being tested, one processor may see it as a one, while the other sees it as a zero. The result is that one processor will do an extra loop, thus putting it several states out of sync with the other.

2.16. PIN DESCRIPTION

VCC

Main supply voltage (5V).

VSS

Digital circuit ground (0V).There are two VSS pins, both must be tied to ground.

VPD

RAM standby supply voltage (5V). This voltage must be present during normal operation. See section 2.4.2 and 4.1.

VREF

Reference voltage and power supply for the analog portion of the A/D converter. Nominally at 5 volts. See section 2.9.1 and 4.1.

ANGND

Reference ground for the A/D converter. Should be held at nominally the same potential as VSS. See section 2.9.1 and 4.1.

VBB

Substrate voltage from the on-chip back-bias generator. This pin should be connected to ANGND through a 0.01

uf capacitor (and not connected to anything else). The capacitor is not required if the A/D convertor is not being used.

XTAL1

Input of the oscillator inverter and input to the internal clock generator. See sections 2.2 and 4.1.

XTAL2

Output of the oscillator inverter. See section 2.2.

CLKOUT

Output of the internal clock generator. The frequency of CLKOUT is ⅓ the oscillator frequency. It has a 33% duty cycle. CLKOUT can drive one TTL input. See section 2.2.

$\overline{\text{RESET}}$

Reset input to the chip, also output to other circuits. Input low for at least 2 state times to reset the chip. $\overline{\text{RESET}}$ has a strong internal pullup. See section 2.15 and 4.1.

$\overline{\text{TEST}}$

Input low enables a factory test mode. The user should tie this pin to VCC for normal operation.

NMI

A low to high transition a vector to *external* memory location 0000H. Reserved for use in Intel Development systems.

INST

Output high while the address is valid during an external read indicates the read is an instruction fetch. See section 2.3.5 and 4.6.

$\overline{\text{EA}}$

Input for memory select (External Access). $\overline{\text{EA}}$ = 1 causes memory accesses to locations 2000H through 3FFFH to be directed to on-chip ROM. $\overline{\text{EA}}$ = 0 causes accesses to these locations to be directed to off-chip memory. $\overline{\text{EA}}$ has an internal pulldown, so it goes to 0 unless driven to 1. See section 2.3.4.

ALE

Address Latch Enable output. ALE is activated only during external memory accesses. It is used to latch the address from the multiplexed address/data bus. See section 2.3.5 and 4.6.

$\overline{\text{RD}}$

Read signal output to external memory. $\overline{\text{RD}}$ is activated only during external memory reads. See section 2.3.5 and 4.6.

$\overline{\text{WR}}$

Write signal output to external memory. $\overline{\text{WR}}$ is activated only during external memory writes. See section 2.3.5 and 4.6.

$\overline{\text{BHE}}$

Bus High Enable signal output to external memory. $\overline{\text{BHE}}$ (0/1) selects/deselects the bank of memory that is connected to the high byte of the data bus. See section 2.3.5 and 4.6.

READY

The READY input is used to lengthen external memory bus cycles up to the time specified in the data sheet. It has a weak internal pullup. See section 2.3.5 and 4.6.

HSI

High impedance inputs to HSI Unit. Four HSI pins are available: HSI.0, HSI.1, HSI.2, and HSI.3. Two of them (HSI.2 and HSI.3) are shared with the HSO Unit. See section 2.7.

HSO

Outputs from HSO Unit. Six HSO pins are available: HSO.0, HSO.1, HSO.2, HSO.3, HSO.4, and HSO.5. Two of them (HSO.4 and HSO.5) are shared with the HSI Unit. All HSO pins are capable of driving one TTL input. See section 2.8.

PORT 0/ANALOG CHANNEL

High impedance input-only port. These pins can be used as digital inputs and/or as analog inputs to the on-chip A/D converter. See sections 2.9 and 2.12.1.

PORT 1

Quasi-bidirectional I/O port. All pins of P1 are capable of driving one LS TTL input. See section 2.1.2.

PORT 2

Multi-functional port. Six of its pins are shared with other functions in the 8096, as shown below.

Port	Function	Alternate Function	Reference section
P2.0	output	TXD (serial port transmit)	2.11.3
P2.1	input	RXD (serial port receive)	2.11.3
P2.2	input	EXTINT (external interrupt)	2.5
P2.3	input	T2CLK (Timer 2 input)	2.6.2
P2.4	input	T2RST (Timer 2 reset)	2.6.2
P2.5	output	PWM (pulse-width modulator)	2.10
P2.6	quasi-bidirectional		
P2.7	quasi-bidirectional		

The multi-functional inputs are high impedance. See section 2.1.3.

PORTS 3 AND 4

8-bit bidirectional I/O ports. These pins are shared the multiplexed address/data bus when accessing external memory, with the Port 3 pins accessing the low byte and Port 4 pins accessing the high byte. They are open drain except when being used as system bus pins. See section 2.3.5.

2.17. PIN LIST

The following is a list of pins in alphabetical order. Where a pin has two names it has been listed under both names, except for the system bus pins, AD0-AD15, which are listed under Port 3 and Port 4.

The Following pins are not bonded out in the 48-pin package:

P1.0 through P1.7, P0.0 through P0.3, P2.3, P2.4, P2.6, P2.7 CLKOUT, INST, NMI, TEST, T2CLK(P2.3), T2RST(P2.4).

Name	68-Pin	48-Pin
ACH0/P0.1	4	—
ACH1/P0.2	5	—
ACH2/P0.3	3	—
ACH3/P0.3	6	—
ACH4/P0.4	67	43
ACH5/P0.5	68	42
ACH6/P0.6	2	40
ACH7/P0.7	1	41
ALE	16	34
ANGND	66	44
$\overline{\text{BHE}}$	37	15
CLKOUT	13	—
$\overline{\text{EA}}$	7	39
EXTINT/P2.2	63	47
HSI.0	54	3
HSI.1	53	4
HSI.2/HSO.4	52	5
HSI.3/HSO.5	51	6
HSO.0	50	7
HSO.1	49	8
HSO.2	44	9
HSO.3	43	10
HSO.4/HSI.2	52	5
HSO.5/HSI.3	51	6
INST	15	—
NMI	7	—
PWM/P2.5	39	13
P0.0/ACH0	4	—
P0.1/ACH1	5	—
P0.2/ACH2	3	—
P0.3/ACH3	6	—
P0.4/ACH4	67	43
P0.5/ACH5	68	42
P0.6/ACH6	2	40
P0.7/ACH7	1	41
P1.0	59	—
P1.1	58	—
P1.2	57	—
P1.3	56	—
P1.4	55	—
P1.5	48	—
P1.6	47	—
P1.7	46	—

Name	68-Pin	48-Pin
P2.0/TXD	60	2
P2.1/RXD	61	1
P2.2/EXTINT	63	47
P2.3/T2CLK	34	—
P2.4/T2RST	36	—
P2.5/PWM	39	13
P2.6	45	—
P2.7	40	—
P3.0/AD0	18	32
P3.1/AD1	19	31
P3.2/AD2	20	30
P3.3/AD3	21	29
P3.4/AD4	22	28
P3.5/AD5	23	27
P3.6/AD6	24	26
P3.7/AD7	25	25
P4.0/AD8	26	24
P4.1/AD9	27	23
P4.2/AD10	28	22
P4.3/AD11	29	21
P4.4/AD12	30	20
P4.5/AD13	31	19
P4.6/AD14	32	18
P4.7/AD15	33	17
$\overline{\text{RD}}$	17	33
READY	35	16
RESET	62	48
RXD/P2.1	61	1
TEST	14	—
TXD/P2.0	60	2
T2CLK/P2.3	34	—
T2RST/P2.4	36	—
VBB	41	12
VCC	9	38
VPD	64	46
VREF	65	45
VSS	10	11
VSS	42	37
$\overline{\text{WR}}$	38	14
XTAL1	11	36
XTAL2	12	35

MCS®-96 Software Design Information 3

CHAPTER 3
MCS®-96 SOFTWARE DESIGN INFORMATION

3.0. INTRODUCTION

This section provides information which will primarily interest those who must write programs to execute in the 8096. Several other sources of information are currently available which will also be of interest:

MCS®-96 MACRO ASSEMBLER USER'S GUIDE
Order Number 122048-001

MCS-96 UTILITIES USER'S GUIDE
Order Number 122049-001

MCS-96 MACRO ASSEMBLER AND UTILITIES POCKET REFERENCE
Order Number 122050-001

Throughout this chapter short segments of code are used to illustrate the operation of the device. For these sections it has been assumed that a set of temporary registers have been predeclared. The names of these registers have been chosen as follows:

AX, BX, CX, and DX are 16 bit registers.

AL is the low byte of AX, AH is the high byte.

BL is the low byte of BX

CL is the low byte of CX

DL is the low byte of DX

These are the same as the names for the general data registers used in 8086. It is important to note, however, that in the 8096, these are not dedicated registers but merely the symbolic names assigned by the programmer to an eight byte region within onboard register file.

3.1. OPERAND TYPES

The MCS®-96 architecture provides support for a variety of data types which are likely to be useful in a control application. In the discussion of these operand types that follows, the names adopted by the PLM-96 programming language will be used where appropriate. To avoid confusion the name of an operand type will be capitalized. A "BYTE" is an unsigned eight bit variable; a "byte" is an eight bit unit of data of any type.

3.1.1. Bytes

BYTES are unsigned 8-bit variables which can take on the values between 0 and 255. Arithmetic and relational operators can be applied to BYTE operands but the result must be interpreted in modulo 256 arithmetic. Logical operations on BYTES are applied bitwise. Bits within BYTES are labeled from 0 to 7 with 0 being the least significant bit. There are no alignment restrictions for BYTES so they may be placed anywhere in the MCS-96 address space.

3.1.2. Words

WORDS are unsigned 16-bit variables which can take on the values between 0 and 65535. Arithmetic and relational operators can be applied to WORD operands but the result must be interpreted modulo 65536. Logical operations on WORDS are applied bitwise. Bits within words are labeled from 0 to 15 with 0 being the least significant bit. WORDS must be aligned at even byte boundaries in the MCS-96 address space. The least significant byte of the WORD is in the even byte address and the most significant byte is in the next higher (odd) address. The address of a word is the address of its least significant byte.

3.1.3. Short-Integers

SHORT-INTEGERS are 8-bit signed variables which can take on the values between −128 and +127. Arithmetic operations which generate results outside of the range of a SHORT-INTEGER will set the overflow indicators in the program status word. The actual numeric result returned will be the same as the equivalent operation on BYTE variables. There are no alignment restrictions on SHORT-INTEGERS so they may be placed anywhere in the MCS-96 address space.

3.1.4. Integers

INTEGERS are 16-bit signed variables which can take on the values between −32,768 and 32,767. Arithmetic operations which generate results outside of the range of an INTEGER will set the overflow indicators in the program status word. The actual numeric result returned will be the same as the equivalent operation on WORD variables. INTEGERS conform to the same alignment and addressing rules as do WORDS.

3.1.5. Bits

BITS are single-bit operands which can take on the Boolean values of true and false. In addition to the normal support for bits as components of BYTE and WORD operands, the 8096 provides for the direct testing of any bit in the internal register file. The MCS-96 architecture requires that bits be addressed as components of BYTES or WORDS, it does not support the direct addressing of bits that can occur in the MCS-51 architecture.

3.1.6. Double-Words

DOUBLE-WORDS are unsigned 32-bit variables which can take on the values between 0 and 4,294,967,295. The MCS-96 architecture provides direct support for this operand type only for shifts and as the dividend in a 32 by 16 divide and the product of a 16 by 16 multiply. For these operations a DOUBLE-WORD variable must reside in the on-board register file of the 8096 and be aligned at an address which is evenly divisible by 4. A DOUBLE-WORD operand is addressed by the address of its least significant byte. DOUBLE-WORD operations which are not directly supported can be easily implemented with two WORD operations. For consistency with INTEL provided software the user should adopt the conventions for addressing DOUBLE-WORD operands which are discussed in section 3.5.

3.1.7. Long-Integers

LONG-INTEGERS are 32-bit signed variables which can take on the values between −2,147,483,648 and 2,147,483,647. The MCS-96 architecture provides direct support for this data type only for shifts and as the dividend in a 32 by 16 divide and the product of a 16 by 16 multiply.

LONG-INTEGERS can also be normalized. For these operations a LONG-INTEGER variable must reside in the onboard register file of the 8096 and be aligned at an address which is evenly divisible by 4. A LONG-INTEGER is addressed by the address of its least significant byte.

LONG-INTEGER operations which are not directly supported can be easily implemented with two INTEGER operations. For consistency with Intel provided software, the user should adopt the conventions for addressing LONG operands which are discussed in section 3.5.

3.2. OPERAND ADDRESSING

Operands are accessed within the address space of the 8096 with one of six basic addressing modes. Some of the details of how these addressing modes work are hidden by the assembly language. If the programmer is to take full advantage of the architecture, it is important that these details be understood. This section will describe the addressing modes as they are handled by the hardware. At the end of this section the addressing modes will be described as they are seen through the assembly language. The six basic addressing modes which will be described are termed register-direct, indirect, indirect with auto-increment, immediate, short-indexed, and long-indexed. Several other useful addressing operations can be achieved by combining these basic addressing modes with specific registers such as the ZERO register or the stack pointer.

3.2.1. Register-direct References

The register-direct mode is used to directly access a register from the 256 byte on-board register file. The register is selected by an 8-bit field within the instruction and register address must conform to the alignment rules for the operand type. Depending on the instruction, up to three registers can take part in the calculation.

Examples

```
ADD    AX,BX,CX      ; AX: = BX + CX
MUL    AX,BX         ; AX: = AX*BX
INCB   CL            ; CL: = CL + 1
```

3.2.2. Indirect References

The indirect mode is used to access an operand by placing its address in a WORD variable in the register file. The calculated address must conform to the alignment rules for the operand type. Note that the indirect address can refer to an operand anywhere within the address space of the 8096, including the register file. The register which contains the indirect address is selected by an eight bit field within the instruction. An instruction can contain only one indirect reference and the remaining operands of the instruction (if any) must be register-direct references.

Examples

```
LD     AX,[AX]     ; AX: = MEM_WORD(AX)
ADDB   AL,BL,[CX]  ; AL: = BL + MEM_BYTE(CX)
POP    [AX]        ; MEM_WORD(AX) : = MEM_WORD(SP); SP: = SP + 2
```

3.2.3. Indirect with Auto-increment References

This addressing mode is the same as the indirect mode except that the WORD variable which contains the indirect address is incremented *after* it is used to address the operand. If the instruction operates on BYTES or SHORT-INTEGERS the indirect address variable will be incremented by one, if the instruction operates on WORDS or INTEGERS the indirect address able will be incremented by two.

Examples

```
LD     AX,[BX]+       ; AX: = MEM_WORD(BX); BX: = BX + 2
ADD    AL,BL,[CX]+    ; AL: = AL + BL + MEM_BYTE(CX); CX: = CX + 1
PUSH   [AX]+          ; SP: = SP − 2;
                      ;   MEM_WORD(SP): = MEM_WORD(AX)
                      ;   AX: = AX + 2
```

3.2.4. Immediate References

This addressing mode allows an operand to be taken directly from a field in the instruction. For operations on BYTE or SHORT-INTEGER operands this field is eight bits wide, for operations on WORD or INTEGER operands the field is 16 bits wide. An instruction can contain only one immediate reference and the remaining operand(s) must be register-direct references.

```
Examples
    ADD   AX,#340  ; AX: = AX + 340
    PUSH  #1234H   ; SP: = SP − 2; MEM_WORD(SP): = 1234H
    DIVB  AX,#10   ; AL: = AX/10; AH: = AX MOD 10
```

3.2.5. Short-indexed References

In this addressing mode an eight bit field in the instruction selects a WORD variable in the register file which is assumed to contain an address. A second eight bit field in the instruction stream is sign-extended and summed with the WORD variable to form the address of the operand which will take part in the calculation. Since the eight bit field is sign-extended the effective address can be up to 128 bytes before the address in the WORD variable and up to 127 bytes after it. An instruction can contain only one short-indexed reference and the remaining operand(s) must be register-direct references.

```
Examples
    LD     AX,12[BX]     ; AX: = MEM_WORD(BX + 12)
    MULB   AX,BL,3[CX]   ; AX: = BL*MEM_BYTE(CX + 3)
```

3.2.6. Long-indexed References

This addressing mode is like the short-indexed mode except that a *16-bit* field is taken from the instruction and added to the WORD variable to form the address of the operand. No sign extension is necessary. An instruction can contain only one long-indexed reference and the remaining operand(s) must to register-direct references.

```
Examples
    AND   AX,BX,TABLE[CX]      ; AX: = BX AND MEM_WORD(TABLE + CX)
    ST    AX,TABLE[BX]         ; MEM_WORD(TABLE + BX) : = AX
    ADDB AL,BL,LOOKUP[CX]      ; AL: = BL + MEM_BYTE(LOOKUP + CX)
```

3.2.7. ZERO Register Addressing

The first two bytes in the register file are fixed at zero by the 8096 hardware. In addition to providing a fixed source of the constant zero for calculations and comparisons, this register can be used as the WORD variable in a long-indexed reference. This combination of register selection and address mode allows any location in memory to be addressed directly.

```
Examples
    ADD   AX,1234[0]   ; AX: = AX + MEM_WORD(1234)
    POP   5678[0]      ; MEM_WORD(5678): = MEM_WORD(SP)
                       ;    SP: = SP + 2
```

3.2.8. Stack Pointer Register Addressing

The system stack pointer in the 8096 can be accessed as register 18H of the internal register file. In addition to providing for convenient manipulation of the stack pointer, this also facilitates the accessing of operands in the stack. The top of the stack, for example, can be accessed by using the stack pointer as the WORD variable in an indirect reference. In a similar fashion, the stack pointer can be used in the short-indexed mode to access data within the stack.

```
Examples
    PUSH   [SP]        ; DUPLICATE TOP_OF_STACK
    LD     AX,2[SP]    ; AX: = NEXT_TO_TOP
```

3.2.9. Assembly Language Addressing Modes

The 8096 assembly language simplifies the choice of addressing modes to be used in several respects:

Direct Addressing. The assembly language will choose between register-direct addressing and long-indexed with the ZERO register depending on where the operand is in memory. The user can simply refer to an operand by its symbolic name; if the operand is in the register file, a register-direct reference will be used, if the operand is elsewhere in memory, a long-indexed reference will be generated.

Immediate Addressing. The assembly language will choose between short and long indexing depending on the value of the index expression. If the value can be expressed in eight bits then short indexing will be used, if it cannot be expressed in eight bits then long indexing will be used.

The use of these features of the assembly language simplifies the programming task and should be used wherever possible.

3.3 PROGRAM STATUS WORD

The program status word (PSW) is a collection of Boolean flags which retain information concerning the state of the user's program. The format of the PSW is shown in figure 3-1. The information in the PSW can be broken down into two basic categories; interrupt control and condition flags. The PSW can be saved in the system stack with a single operation (PUSHF) and restored in a like manner (POPF).

15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Z	N	V	VT	C	—	I	ST	<Interrupt Mask Reg>							

Figure 3-1. PSW Register

3.3.1. Interrupt Flags

The lower eight bits of the PSW are used to individually mask the various sources of interrupt to the 8096. A logical '1' in these bit positions enables the servicing of the corresponding interrupt. These mask bits can be accessed as an eight bit byte (INT_MASK — address 8) in the onboard register file. Bit 9 in the PSW is the global interrupt enable. If this bit is cleared then all interrupts will be locked out except for the Non Maskable Interrupt (NMI). Note that the various interrupts are collected in the INT _ PENDING register even if they are locked out. Execution of the corresponding service routines will procede according to their priority when they become enabled. Further information on the interrupt structure of the 8096 can be found in sections 2.5 and 3.6.

3.3.2. Condition Flags

The remaining bits in the PSW are set as side effects of instruction execution and can be tested by the conditional jump instructions.

Z. The Z (Zero) flag is set to indicate that the operation generated a result equal to zero. For the add-with-carry (ADDC) and subtract-with-borrow (SUBC) operations the Z flag is cleared if the result is non-zero but is never set. These two instructions are normally used in conjunction with the ADD and SUB instructions to perform multiple precision arithmetic. The operation of the Z flag for these instructions leaves it indicating the proper result for the entire multiple precision calculation.

N. The N (Negative) flag is set to indicate that the operation generated a negative result. Note that the N flag will be set to the algebraically correct state even if the calculation overflows.

V. The V (oVerflow) flag is set to indicate that the operation generated a result which is outside the range that can be expressed in the destination data type.

VT. the VT (oVerflow Trap) flag is set whenever the V flag is set but can only be cleared an instruction which explicitly operates on it such as the CLRVT or JVT instructions. The operation of the VT flag allows for the testing for a possible overflow condition at the end of a sequence of related arithmetic operations. This is normally more efficient than testing the V flag after each instruction.

C. The C (Carry) flag is set to indicate the state of the arithmetic carry from the most significant bit of the ALU for an arithmetic operation or the state of the last bit shifted out of the operand for a shift. Arithmetic Borrow after a subtract operation is the complement of the C flag (i.e. if the operation generated a borrow then $C=0$).

ST. The ST (STicky bit) set to indicate that during a right shift a 1 has been shifted first into the C flag and then been shifted out. The ST flag is undefined after a multiply operation. The ST flag can be used along with the C flag to control rounding after a right shift. Consider multiplying two eight bit quantities and then scaling the result down to 12 bits:

```
MULUB     AX,CL,DL        ; AX: = CL*DL
SHR       AX,#4           ; Shift right 4 places
```

If the C flag is set after the shift it indicates that the bits shifted off the end of operand were greater-than or equal-to one half the least significant bit (LSB) of the result. If the C flag is clear after the shift it indicates that the bits shifted off the end of the operand were less than half the LSB of the result. Without the ST flag, the rounding decision must be made on the basis of this information alone. (Normally the result would be rounded up if the C flag is set.) The ST flag allows a finer resolution in the rounding decision:

C ST	Value of the bits shifted off
0 0	Value = 0
0 1	0 < Value < ½ LSB
1 0	Value = ½ LSB
1 1	Value > ½ LSB

Figure 3-2. Rounding Alternatives

Imprecise rounding can be a major source of error in a numerical calculation; use of the ST flag improves the options available to the programmer.

3.4 INSTRUCTION SET

The MCS-96 instruction set contains a full set of arithmetic and logical operations for the 8-bit data types BYTE and SHORT INTEGER and for the 16-bit data types WORD and INTEGER. The DOUBLE-WORD and LONG data types (32 bits) are supported for the products of 16 by 16 multiplies and the dividends of 32 by 16 divides and for shift operations. The remaining operations on 32 bit variables can be implemented by combinations of 16 bit operations. As an example the sequence:

```
ADD         AX,CX
ADDC        BX,DX
```

performs a 32 bit addition, and the sequence

```
SUB         AX,CX
SUBC        BX,DX
```

performs a 32 bit subtraction. Operations on REAL (i.e. floating point) variables are not supported directly by the hardware but are supported by the floating point library for the 8096 (FPAL-96) which implements a single precision subset of the proposed IEEE standard for floating point operations. The performance of this software is significantly improved by the 8096 NORML instruction which normalizes a 32-bit variable and by the existance of the ST flag in the PSW.

In addition to the operations on the various data types, the 8096 supports conversions between these types. LDBZE (load byte zero extended) converts a BYTE to a WORD and LDBSE (load byte sign extended) converts a SHORT-INTEGER into an INTEGER. WORDS can be converted to DOUBLE-WORDS by simply clearing the upper WORD of the DOUBLE-WORD (CLR) and INTEGERS can be converted to LONGS with the EXT (sign extend) instruction.

Tables 3-1 and 3-2 summarize the operation of each of the instructions and Tables 3-3 and 3-4 give the opcode, byte count, and timing information for each of the instructions.

Table 3-1. Instruction Summary

Mnemonic	Oper-ands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
ADD/ADDB	2	D ← D+A	√	√	√	√	√	—	
ADD/ADDB	3	D ← B+A	√	√	√	√	√	—	
ADDC/ADDCB	2	D ← D+A+C	√	√	√	√	√	—	
SUB/SUBB	2	D ← D − A	√	√	√	√	√	—	
SUB/SUBB	3	D ← B − A	√	√	√	√	√	—	
SUBC/SUBCB	2	D ← D − A + C − 1	√	√	√	√	√	—	
CMP/CMPB	2	D − A	√	√	√	√	√	—	
MUL/MULU	2	D, D + 2 ← D * A	—	—	—	—	—	√	2
MUL/MULU	3	D, D + 2 ← B * A	—	—	—	—	—	√	2
MULB/MULUB	2	D, D + 1 ← D * A	—	—	—	—	—	√	3
MULB/MULUB	3	D, D + 1 ← B * A	—	—	—	—	—	√	3
DIV/DIVU	2	D ← (D, D + 2)/A D + 2 remainder	—	—	—	√	√	—	2
DIVB/DIVUB	3	D ← (D, D + 1)/A D + 1 remainder	—	—	—	√	√	—	3
AND/ANDB	2	D ← D and A	√	√	0	0	—	—	
AND/ANDB	3	D ← B and A	√	√	0	0	—	—	
OR/ORB	2	D ← D or A	√	√	0	0	—	—	
XOR/XORB	2	D ← D (excl. or) A	√	√	0	0	—	—	
LD/LDB	2	D ← A	—	—	—	—	—	—	
ST/STB	2	A ← D	—	—	—	—	—	—	
LDBSE	2	D ← A; D + 1 ← SIGN(A)	—	—	—	—	—	—	3,4
LDBZE	2	D ← A; D + 1 ← 0	—	—	—	—	—	—	3,4
PUSH	1	SP ← SP − 2; (SP) A	—	—	—	—	—	—	
POP	1	A ← (SP); SP ← SP + 2	—	—	—	—	—	—	
PUSHF	0	SP ← SP − 2; (SP) ← PSW; PSW ← 0000H	0	0	0	0	0	0	
POPF	0	PSW ← (SP); SP ← SP + 2	√	√	√	√	√	√	
SJMP	1	PC ← PC + 11-bit offset	—	—	—	—	—	—	5
LJMP	1	PC ← PC + 16-bit offset	—	—	—	—	—	—	5
INDJMP	1	PC ← (A)	—	—	—	—	—	—	
SCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 11-bit offset	—	—	—	—	—	—	5
LCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 16-bit offset	—	—	—	—	—	—	5
RET	0	PC ← (SP); SP ← SP + 2	—	—	—	—	—	—	
J(conditional)	1	PC ← PC + 8-bit offset	—	—	—	—	—	—	5
JC		Jump if C = 1	—	—	—	—	—	—	5
JNC		Jump if C = 0	—	—	—	—	—	—	5

Note

1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B, and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.
2. D, D + 2 are consecutive WORDS in memory; D is DOUBLE-WORD aligned.
3. D, D + 1 are consecutive BYTES in memory; D is WORD aligned.
4. Changes a byte to a word.
5. Offset is a 2's complement number.

Table 3-2. Instruction Summary

Mnemonic	Oper- ands	Operation (Note 1)	Flags Z	N	C	V	VT	ST	Notes
JE		Jump if Z = 1	—	—	—	—	—	—	5
JNE		Jump if Z = 0	—	—	—	—	—	—	5
JGE		Jump if N = 1	—	—	—	—	—	—	5
JLT		Jump if N = 0	—	—	—	—	—	—	5
JGT		Jump if N = 0 and Z = 0	—	—	—	—	—	—	5
JLE		Jump if N = 1 or Z = 1	—	—	—	—	—	—	5
JH		Jump if C = 1 and Z = 0	—	—	—	—	—	—	5
JNH		Jump if C = 0 or Z = 1	—	—	—	—	—	—	5
JV		Jump if V = 1	—	—	—	—	—	—	5
JNV		Jump if V = 0	—	—	—	—	—	—	5
JVT		Jump if VT = 1; Clear VT	—	—	—	—	0	—	5
JNVT		Jump if VT = 0; Clear VT	—	—	—	—	0	—	5
JST		Jump if ST = 1	—	—	—	—	—	—	5
JNST		Jump if ST = 0	—	—	—	—	—	—	5
JBS		Jump if ST = 1	—	—	—	—	—	—	5,6
JBC		Jump if Specified Bit = 0	—	—	—	—	—	—	5,6
DJNZ	1	D ← D − 1; if D ≠ then PC ← PC + 8-bit offset	—	—	—	—	—	—	5
DEC/DECB	1	D ← D − 1	√	√	√	√	√	—	
NEG/NEGB	1	D ← 0 − D	√	√	√	√	√	—	
INC/INCB	1	D ← D + 1	√	√	√	√	√	—	
EXT	1	D ← D; D + 2 ← Sign (D)	√	√	0	0	—	—	2
EXTB	1	D ← D; D + 1 ← Sign (D)	√	√	0	0	—	—	3
NOT/NOTB	1	D ← Logical Not (D)	√	√	0	0	—	—	
CLR/CLRB	1	D ← 0	1	0	0	0	—	—	
SHL/SHLB/SHLL	1	C ← msb — — — — — — lsb ← 0	√	√	√	√	√	—	7
SHR/SHRB/SHRL	1	0 → msb — — — — — — lsb → C	√	√	√	0	—	√	7
SHRA/SHRAB/SHRAL	1	msb → msb — — — — — — lsb → C	√	√	√	0	—	√	7
SETC	0	C ← 1	—	—	1	—	—	—	
CLRC	0	C ← 0	—	—	0	—	—	—	
CLRVT	0	VT ← 0	—	—	—	—	0	—	
RST	0	PC ← 2080H	0	0	0	0	0	0	8
DI	0	Disable All Interrupts	—	—	—	—	—	—	
EI	0	Enable All Interrupts	—	—	—	—	—	—	
NOP	0	PC ← PC + 1	—	—	—	—	—	—	
SKIP	0	PC ← PC + 2	—	—	—	—	—	—	
NORML	2	Normalize (See sec 3.13.66)	√	1	—	—	—	—	7

Note

1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.
5. Offset is a 2's complement number.
6. Specified bit is one of the 2048 bits in the register file.
7. The "L" (Long) suffix indicates double-word operation.
8. Initiates a Reset by pulling RESET low. Software should re-initialize all the necessary registers with code starting at 2080H.

Table 3-3. Opcode and State Time Listing

		DIRECT			IMMEDIATE			INDIRECT⊛					INDEXED⊛				
								NORMAL			AUTO-INC.		SHORT			LONG	
MNEMONIC	OPERANDS	OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES
ARITHMETIC INSTRUCTIONS																	
ADD	2	64	3	4	65	4	5	66	3	6/11	3	7/12	67	4	6/11	5	7/12
ADD	3	44	4	5	45	5	6	46	4	7/12	4	8/13	47	5	7/12	6	8/13
ADDB	2	74	3	4	75	3	4	76	3	6/11	3	7/12	77	4	6/11	5	7/12
ADDB	3	54	4	5	55	4	5	56	4	7/12	4	8/13	57	5	7/12	6	8/13
ADDC	2	A4	3	4	A5	4	5	A6	3	6/11	3	7/12	A7	4	6/11	5	7/12
ADDCB	2	B4	3	4	B5	3	4	B6	3	6/11	3	7/12	B7	4	6/11	5	7/12
SUB	2	68	3	4	69	4	5	6A	3	6/11	3	7/12	6B	4	6/11	5	7/12
SUB	3	48	4	5	49	5	6	4A	4	7/12	4	8/13	4B	5	7/12	6	8/13
SUBB	2	78	3	4	79	3	4	7A	3	6/11	3	7/12	7B	4	6/11	5	7/12
SUBB	3	58	4	5	59	4	5	5A	4	7/12	4	8/13	5B	5	7/12	6	8/13
SUBC	2	A8	3	4	A9	4	5	AA	3	6/11	3	7/12	AB	4	6/11	5	7/12
SUBCB	2	B8	3	4	B9	3	4	BA	3	6/11	3	7/12	BB	4	6/11	5	7/12
CMP	2	88	3	4	89	4	5	8A	3	6/11	3	7/12	8B	4	6/11	5	7/12
CMPB	2	98	3	4	99	3	4	9A	3	6/11	3	7/12	9B	4	6/11	5	7/12
MULU	2	6C	3	25	6D	4	26	6E	3	27/32	3	28/33	6F	4	27/32	5	28/33
MULU	3	4C	4	26	4D	5	27	4E	4	28/33	4	29/34	4F	5	28/33	6	29/34
MULUB	2	7C	3	17	7D	3	17	7E	3	19/24	3	20/25	7F	4	19/24	5	20/25
MULUB	3	5C	4	18	5D	4	18	5E	4	20/25	4	21/26	5F	5	20/25	6	21/26
MUL	2	②	4	29	②	5	30	②	4	31/36	4	32/37	②	5	31/36	6	32/37
MUL	3	②	5	30	②	6	31	②	5	32/37	5	33/38	②	6	32/37	7	33/38
MULB	2	②	4	21	②	4	21	②	4	23/28	4	24/29	②	5	23/28	6	24/29
MULB	3	②	5	22	②	5	22	②	5	24/29	5	25/30	②	6	24/29	7	25/30
DIVU	2	8C	3	25	8D	4	26	8E	3	27/32	3	28/33	8F	4	27/32	5	28/33
DIVUB	2	9C	3	17	9D	3	17	9E	3	19/24	3	20/25	9F	4	19/24	5	20/25
DIV	2	②	4	29	②	5	30	②	4	31/36	4	32/37	②	5	31/36	6	32/37
DIVB	2	②	4	21	②	4	21	②	4	23/28	4	24/29	②	5	23/28	6	24/29

Notes:

⊛ Long indexed and Indirect + instructions have identical opocodes with Short indexed and Indirect modes, respectively. The second byte of instructions using any indirect or indexed addressing mode specifies the exact mode used. If the second byte is even, use Indirect or Short Indexed. If it is odd, use Indirect + or Long indexed. In all cases the second byte of the instruction always specifies an even (word) location for the address referenced.

① Number of state times shown for internal/external operands.

② The opcodes for signed multiply and divide are the opcodes for the unsigned functions with an "FE" appended as a prefix.

Table 3-3. Continued

		DIRECT			IMMEDIATE			INDIRECT⊛					INDEXED⊛				
								NORMAL			AUTO-INC.		SHORT			LONG	
MNEMONIC	OPERANDS	OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES
LOGICAL INSTRUCTIONS																	
AND	2	60	3	4	61	4	5	62	3	6/11	3	7/12	63	4	6/11	5	7/12
AND	3	40	4	5	41	5	6	42	4	7/12	4	8/13	43	5	7/12	6	8/13
ANDB	2	70	3	4	71	3	4	72	3	6/11	3	7/12	73	4	6/11	5	7/12
ANDB	3	50	4	5	51	4	5	52	4	7/12	4	8/13	53	5	7/12	6	8/13
OR	2	80	3	4	81	4	5	82	3	6/11	3	7/12	83	4	6/11	5	7/12
ORB	2	90	3	4	91	3	4	92	3	6/11	3	7/12	93	4	6/11	5	7/12
XOR	2	84	3	4	85	4	5	86	3	6/11	3	7/12	87	4	6/11	5	7/12
XORB	2	94	3	4	95	3	4	96	3	6/11	3	7/12	97	4	6/11	5	7/12
DATA TRANSFER INSTRUCTIONS																	
LD	2	A0	3	4	A1	4	5	A2	3	6/11	3	7/12	A3	4	6/11	5	7/12
LDB	2	B0	3	4	B1	3	4	B2	3	6/11	3	7/12	B3	4	6/11	5	7/12
ST	2	C0	3	4	—	—	——	C2	3	7/13	3	8/14	C3	4	7/13	5	8/14
STB	2	C4	3	4	—	—	——	C6	3	7/13	3	8/14	C7	4	7/13	5	8/14
LDBSE	2	BC	3	4	BD	3	4	BE	3	6/11	3	7/12	BF	4	6/11	5	7/12
LDBZE	2	AC	3	4	AD	3	4	AE	3	6/11	3	7/12	AF	4	6/11	5	7/12
STACK OPERATIONS (internal stack)																	
PUSH	1	C8	2	8	C9	3	8	CA	2	11/15	2	12/16	CB	3	11/15	4	12/16
POP	1	CC	2	12	—	—	——	CE	2	14/18	2	14/18	CF	3	14/18	4	14/18
PUSHF	0	F2	1	8													
POPF	0	F3	1	9													
STACK OPERATIONS (external stack)																	
PUSH	1	C8	2	12	C9	3	12	CA	2	15/19	2	16/20	CB	3	15/19	4	16/20
POP	1	CC	2	14	—	—	——	CE	2	16/20	2	16/20	CF	3	16/20	4	16/20
PUSHF	0	F2	1	12													
POPF	0	F3	1	13													

JUMPS AND CALLS							
MNEMONIC	OPCODE	BYTES	STATES	MNEMONIC	OPCODE	BYTES	STATES
LJMP	E7	3	8	LCALL	EF	3	13/16⑤
SJMP	20-27④	2	8	SCALL	28-2F④	2	13/16⑤
INDJMP	E3	2	8	RET	F0	1	12/16⑤

Notes:

① Number of state times shown for internal/external operands.

③ This instruction is generated by the assembler when a branch indirect (BR [Rx]) instruction is reached.

④ The least significant 3 bits of the opcode are concatenated with the following 8 bits to form an 11-bit, 2's complement, offset for the relative call or jump.

⑤ State times for stack located internal/external.

Table 3-4. CONDITIONAL JUMPS

All conditional jumps are 2 byte instructions. They require 8 state times if the jump is taken, 4 if it is not.

MNEMONIC	OPCODE	MNEMONIC	OPCODE	MNEMONIC	OPCODE	MNEMONIC	OPCODE
JC	DB	JE	DF	JGE	D6	JGT	D2
JNC	D3	JNE	D7	JLT	DE	JLE	DA
JH	D9	JV	DD	JVT	DC	JST	D8
JNH	D1	JNV	D5	JNVT	D4	JNST	D0

JUMP ON BIT CLEAR OR BIT SET

These instructions are 3-byte instructions. They require 9 state times if the jump is taken, 5 if it is not.

	BIT NUMBER							
MNEMONIC	0	1	2	3	4	5	6	7
JBC	30	31	32	33	34	35	36	37
JBS	38	39	3A	3B	3C	3D	3E	3F

LOOP CONTROL

DJNZ OPCODE EO; 3 BYTES; 5/9 STATE TIMES (NOT TAKEN/TAKEN)

SINGLE REGISTER INSTRUCTIONS

MNEMONIC	OPCODE	BYTES	STATES	MNEMONIC	OPCODE	BYTES	STATES
DEC	05	2	4	EXT	06	2	4
DECB	15	2	4	EXTB	16	2	4
NEG	03	2	4	NOT	02	2	4
NEGB	13	2	4	NOTB	12	2	4
INC	07	2	4	CLR	01	2	4
INCB	17	2	4	CLRB	11	2	4

SHIFT INSTRUCTIONS

INSTR	WORD		INSTR	BYTE		INSTR	DBL WD		
MNEMONIC	OP	B	MNEMONIC	OP	B	MNEMONIC	OP	B	STATE TIMES
SHL	09	3	SHLB	19	3	SHLL	0D	3	7 + 1 PER SHIFT⑦
SHR	08	3	SHRB	18	3	SHRL	0C	3	7 + 1 PER SHIFT⑦
SHRA	0A	3	SHRAB	1A	3	SHRAL	0E	3	7 + 1 PER SHIFT⑦

SPECIAL CONTROL INSTRUCTIONS

MNEMONIC	OPCODE	BYTES	STATES	MNEMONIC	OPCODE	BYTES	STATES
SETC	F9	1	4	DI	FA	1	4
CLRC	F8	1	4	EI	FB	1	4
CLRVT	FC	1	4	NOP	FD	1	4
RST	FF	1	16	SKIP	00	2	4

NORMALIZE

NORML	0F	3	11 + 1 PER SHIFT

Notes:

⑥ This instruction takes 2 states to pull RST low, then holds it low for 2 states to initiate a reset. The reset takes 12 states, at which time the program restarts at location 2080H.

⑦ Execution will take at least 8 states, even for 0 shift.

3.5. SOFTWARE STANDARDS AND CONVENTIONS

For a software project of any size it is a good idea to modularize the program and to establish standards which control the communication between these modules. The nature of these standards will vary with the needs of the final application. A common component of all of these standards, however, must be the mechanism for passing parameters to procedures and returning results from procedures. In the absence of some overriding consideration which prevents their use, it is suggested that the user conform to the conventions adopted by the PLM-96 programing language for procedure linkage. It is a very usable standard for both the assembly language and PLM-96 environment and it offers compatibility between these environments. Another advantage is that it allows the user access to the same floating point arithmetics library that PLM-96 uses to operate on REAL variables.

3.5.1. Register Utilization

The MCS-96 architecture provides a 256 byte register file. Some of these registers are used to control register-mapped I/O devices and for other special functions such as the ZERO register and the stack pointer. The remaining bytes in the register file, some 230 of them, are available for allocation by the programmer. If these registers are to be used effectively some overall strategy for their allocation must be adopted. PLM-96 adopts the simple and effective strategy of allocating the eight bytes between addresses 1CH and 23H as temporary storage. The starting address of this region is called PLMREG. The remaining area in the register file is treated as a segment of memory which is allocated as required.

3.5.2. Addressing 32-bit Operands

These operands are formed from two adjacent 16-bit words in memory. The least significant word of the double word is always in lower address, even when the data is in the stack (which means that the most significant word must be pushed into the stack first). A double word is addressed by the address of its least significant byte. Note that the hardware supports some operations on double words (e.g. normalize and divide). For these operations the double word must be in the internal register file and must have an address which is evenly divisible by four.

3.5.3. Subroutine Linkage

Parameters are passed to subroutines in the stack. Parameters are pushed into the stack in the order that they are encountered in the scanning of the source text. Eight-bit parameters (BYTES or SHORT-INTEGERS) are pushed into the stack with the high order byte undefined. Thirty-two bit parameters (LONG-INTEGERS, DOUBLE-WORDS, and REALS) are pushed into the stack as two 16 bit values; the most significant half of the parameter is pushed into the stack first.

As an example, consider the following PLM-96 procedure:

```
example_procedure: PROCEDURE (param1,param2,param3);
  DECLARE param1 BYTE,
          param2 DWORD,
          param3 WORD;
```

When this procedure is entered at run time the stack will contain the parameters in the following order:

?????? : param1	
high word of param2	
low word of param2	
param3	
return address	← Stack_pointer

Figure 3-3. Stack Image

If a procedure returns a value to the calling code (as opposed to modifying more global variables) then the result is returned in the variable PLMREG. PLMREG is viewed as either an 8, 16 or 32 bit variable depending on the type of the procedure.

The standard calling convention adopted by PLM-96 has several key features:

a). Procedures can always assume that the eight bytes of register file memory starting at PLMREG can be used as temporaries within the body of the procedure.

b). Code which calls a procedure must assume that the eight bytes of register file memory starting at PLMREG are modified by the procedure.

c). The Program Status Word (PSW-see section 3.3) is not saved and restored by procedures so the calling code must assumed that the condition flags (Z,N,V,VT,C, and ST) are modified by the procedure.

d). Function results from procedures are always returned in the variable PLMREG.

PLM-96 allows the definition of INTERRUPT procedures which are executed when a predefined interrupt occurs. These procedures do not conform to the rules of a normal procedure. Parameters cannot be passed to these procedures and they cannot return results. Since they can execute essentially at any time (hence the term interrupt), these procedures must save the PSW and PLMREG when they are entered and restore these values before they exit.

3.6. USING THE INTERRUPT SYSTEM

Processing interrupts is an integral part of almost any control application. The 8096 allows the program to manage interrupt servicing in an efficient and flexible manner. Software running in the 8096 exerts control over the interrupt hardware at several levels.

3.6.1. Global Lockout

The processing of interrupts can be enabled or disabled by setting or clearing the I bit in the PSW. This is accomplished by the EI (Enable Interrupts) and DI (Disable Interrupts) instructions. Note that the I bit only controls the actual servicing of interrupts; interrupts that occur during periods of lockout will be held in the pending register and serviced on a prioritized basis when the lockout period ends.

3.6.2. Pending Interrupt Register

When the hardware detects one of the eight interrupts it sets the corresponding bit in the pending interrupt register (INT_PENDING-register 09H). This register, which has the same bit layout as the interrupt mask register (see next section), can be read or modified as a byte register. This register can be read to determine which of the interrupts are pending at any given time or modified to either clear pending interrupts or generate interrupts under software control. Any software which modifies the INT_PENDING register should ensure that the entire operation is indivisible. The easiest way of doing this is to use the logical instructions in the two or three operand format, as examples:

```
ANDB    INT_PENDING,#11111101B
ORB     ; Clears the A/D interrupt
        INT_PENDING,#00000010B
        Sets the A/D interrupt
```

If the required modification to INT_PENDING cannot be accomplished with one instruction then a critical region should be established and the INT_PENDING register modified from within this region (see section 3.6.5).

3.6.3. Interrupt Mask Register

Individual interrupts can be enabled or disabled by setting or clearing bits in the interrupt mask register (INT_MASK-register 08H). The format of this register is shown in figure 3-4.

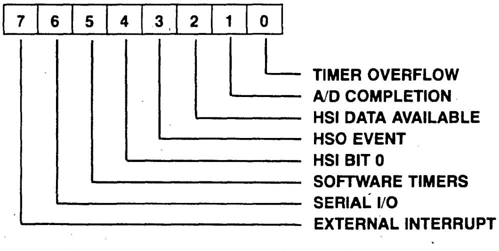


Figure 3-4. Interrupt Mask Register

The INT_MASK register can be read or written as a byte register. A one in any bit position will enable the corresponding interrupt source and a zero will disable the source. The individual masks act like the global lockout in that they only control the servicing of the interrupt; the hardware will save any interrupts that occur in the pending register even if the interrupt mask bit is cleared. The INT_MASK register also can be accessed as the lower eight bits of the PSW so the PUSHF and POPF instructions save and restore the INT_MASK register as well as the global interrupt lockout and the arithmetic flags.

3.6.4. Interrupt Vectors

The 8096 has eight sources of hardware interrupt, each with its own priority and interrupt vector location. Table 3-6 shows the interrupt sources, their priority, and their vector locations. See section 2.5 for a discussion of the various interrupt sources.

Table 3-5. Interrupt Vector Information

Source	Priority	Vector
Timer Overflow	0-Lowest	2000H
A/D Completion	1	2002H
HSI Data Available	2	2004H
HSO Excution	3	2006H
HSI.O	4	2008H
Software timers	5	200AH
Serial I/O	6	200CH
External Interrupt	7-Highest	200EH

The programmer must initialize the interrupt vector table with the starting addresses of the appropriate interrupt service routine. It would be a good idea to vector any interrupts that are not used in the system to an error handling routine.

The priorities given in the table give the *hardware* enforced priorities for these interrupts. This priority controls the order in which pending interrupts are passed to the software via interrupt-calls. The software can implement its own priority structure by controlling the mask register (INT_MASK-register 08H). To see how this is done consider the case of a serial I/O service routine which must run at a priority level which is lower than the HSI data available interrupt but higher than any other source. The "preamble" and exit code for this interrupt service routine would look like this:

```
serial_io_isr:
    PUSHF                       ; Save the PSW
                                  (Includes INT_MASK)
    LD    INT_MASK,#00000100B
    EI                          ; Enable interrupts again
    ;  \
    ;   |
    ;   |
    ;   > Service the interrupt
    ;   |
    ;   |
    ;  /
    POPF                        ; Restore the PSW
    RET
```

Note that location 200CH in the interrupt vector table would have to be loaded with the value of the label serial_io_isr and the interrupt be enabled for this routine to execute.

There is an interesting chain of instruction side-effects which makes this (or any other) 8096 interrupt service routine execute properly:

a). After the hardware decides to process an interrupt it generates and executes a special interrupt-call instruction which pushes the current program counter onto the stack and then loads the program counter with the contents of the vector table entry corresponding to the interrupt. The hardware will not allow another interrupt to be serviced immediately following the interrupt-call. This guarantees that once the interrupt-call starts the first instruction of the interrupt service routine will execute.

b). The PUSHF instruction, which is now guaranteed to execute, saves the PSW in the stack and then clears the PSW. The PSW contains, in addition to the arithmetic flags, the INT_MASK register and the global enable flag (I). The hardware will not allow an interrupt following a PUSHF instruction and by the time the LD instruction starts all of the interrupt enable flags will be cleared. Now there is guaranteed execution of the LD INT_MASK instruction.

c). The LD INT_MASK instruction enables those interrupts that the programmer chooses to allow to interrupt the serial I/O interrupt service routine. In this example only the HSI data available interrupt will be allowed to do this but any interrupt or combination of interrupts could be enabled at this point, even the serial interrupt. It is the loading of the INT_MASK register which allows the software to establish its own priorities for interrupt servicing independently from those that the hardware enforces.

d). The EI instruction reenables the processing of interrupts.

e). The actual interrupt service routine executes within the priority structure established by the software.

f). At the end of the service routine the POPF instruction restores the PSW to its state when the interrupt-call occurred. The hardware will not allow interrupts to be processed following a POPF instruction so the execution of the last instruction (RET) is guaranteed before further interrupts can occur. The reason that this RET instruction must be protected in this fashion is that it is quite likely that the POPF instruction will reenable an interrupt which is already pending. If this interrupt were serviced before the RET instruction, then the return address to the code that was executing when the original interrupt occurred would be left on the stack. While this does not present a problem to the program flow, it could result in a stack overflow if interrupts are occurring at a high frequency.

Notice that the "preamble" and exit code for the interrupt service routine does not include any code for saving or restoring registers. This is because it has been assumed that the interrupt service routine has been allocated its own private set of registers from the on-board register file. The availability of some 230 bytes of register storage makes this quite practical.

3.6.5. Critical Regions

Interrupt service routines must share some data with other routines. Whenever the programmer is coding those sections of code which access these shared pieces of data, great care must be taken to ensure that the integrity of the data is maintained. Consider clearing a bit in the interrupt pending register as part of a non-interrupt routine:

```
LDB     AL,INT_PENDING
ANDB    AL,#bit_mask
STB     AL,INT_PENDING
```

This code works if no other routines are operating concurrently, but will cause occasional but serious problems if used in a concurrent environment. (All programs which make use of interrupts must be considered to be part of a concurrent environment.) To demonstrate this problem, assume that the INT_PENDING register contains 00001111B and bit 3 (HSO event interrupt pending) is to be reset. The code does work for this data pattern but what happens if an HSI interrupt occurs somewhere between the LDB and the STB instructions? Before the LDB instruction INT_PENDING contains 00001111B and after the LDB instruction so does AL. IF the HSI interrupt service routine executes at this point then INT_PENDING will change to 00001011B. The ANDB changes AL to 00000111B and the STB changes INT_PENDING to 00000111B. It should be 00000011B. This code sequence has managed to generate a false HSO interrupt! The same basic process can generate an amazing assortment of problems and headaches. These problems can be avoided by assuring mutual exclusion which basically means that if more than one routine can change a variable, then the programmer must ensure exclusive access to the variable during the entire operation on the variable.

In many cases the instruction set of the 8096 allows the variable to be modified with a single instruction. The code in the above example can be implemented with a single instruction:

```
ANDB    INT_PENDING,#bit_mask
```

Instructions are indivisible so mutual exclusion is ensured in this case. For more complex situations, such a simple solution is not available and the programmer must create what is termed a critical region in which it is safe to modify the variable. One way to do this is to simply disable interrupts with a DI instruction, perform the modification, and then re-enable interrupts with an EI instruction. The problem with this approach is that it leaves the interrupts enabled even if they were not enabled at the start. A better solution is to enter the critical region with a PUSHF instruction which saves the PSW and also clears the interrupt enable flags. The region can then be terminated with a POPF instruction which returns the interrupt enable to the state it was in before the code sequence. It should be noted that some system configurations might require more protection to form a critical region. An example is a system in which more than one processor has access to a common resource such as memory or external I/O devices.

3.7. I/O PROGRAMMING CONSIDERATIONS

The on-board I/O devices are, for the most part, simple to program. There are some areas of potential confusion which need to be addressed:

3.7.1. Programming the I/O Ports

Some of the on-board I/O ports can be used as both input and output pins (e.g. Port 1) When the processor writes to the pins of these ports it actually writes into a register which in turn drives the port pin. When the processor reads these ports, it senses the status of the pin directly. If a port pin is to be used as an input then the software should write a one to that pin, this will cause the low-impedance pull-down device to turn off and leave the pin pulled up with relatively high impedance pull-up device which can be easily driven down by the device driving the input. If some pins of a port are to be used as inputs and some are to be used as outputs the programmer should be careful when writing to the port. Consider using P1.0 as an input and then trying to toggle P1.1 as an output:

```
ORB     IOPORT1,#00000001B    ; Set P1.0 for input
XORB    IOPORT1,#00000010B    ; Complement P1.1
```

The first instruction will work as expected but two problems can occur when the second instruction executes. The first is that even though P1.1 is being driven high by the 8096 it is possible that it is being held low externally. This typically happens when the port pin is used to drive the base of an NPN transistor which in turn drives whatever there is in the outside world which needs to be toggled. The base of the transistor will clamp the port pin to the transistor's Vbe above ground, typically 0.7 volts. The 8096 will input this value as a zero even if a one has been written to the port pin. When this happens the XORB instruction will always write a one to the port pin and it will not toggle. The second problem, which is related to the first one, is that if P1.0 happens to be driven to a zero when Port 1 is read by the XORB instruction then the XORB will write a zero to P1.0 and it will no longer be useable as an input. The first problem can best be solved by the external driver design. A series resistor between the port pin and the base of the transistor often works. The second problem can be solved in the software fairly easily:

```
LDB     AL,IOPORT1
XORB    AL,#010B
ORB     AL,#001B
STB     AL,IOPORT1
```

A software solution to both problems is to keep a byte in RAM as an image of the data to be output to the port; any time the software wants to modify the data on the port it can then modify the image byte and then copy it to the port.

3.7.2. Reading the I/O Status Register 1

This status register contains a collection of status flags which relate to the timer and high speed I/O functions (see section 2.12.5). It can be accessed as register 16H in the on-board register file. The layout of this register is shown in figure 3-5.

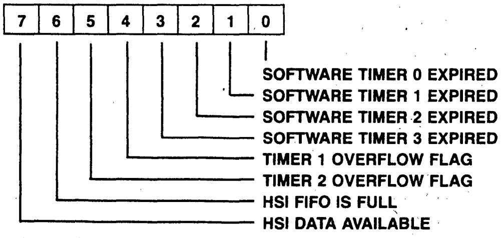


Figure 3-5. I/O Status Register 1

Whenever the processor reads this register all of the time-related flags (bits 5 through 0) are cleared. This applies not only to explicit reads such as:

```
LD      AL,IOS1
```

but also to implicit reads such as:

```
JB      IOS1.3,somewhere_else
```

which jumps to somewhere_else if bit 3 of IOS1 is set. In most cases this situation can best be handled by having a byte in the register file which is used to maintain an image of lower five bits of the register. Any time a hardware timer interrupt or a HSO software timer interrupt occurs the byte can be updated:

```
ORB     IOS1_image,IOS1
```

leaving IOS1_image containing all the flags that were set before plus all the new flags that were read and cleared from IOS1. Any other routine which needs to sample the flags can safely check IOS1_image. Note that if these routines need to clear the flags that they have acted on then the modification of IOS1_image must be done from inside a critical region (see section 3.6.5).

3.7.3. Sending Commands to the HSO Unit

Commands are sent to the HSO unit via a byte and then a word write operation:

```
LDB     HSO_COMMAND,#what_to_do
ADD     HSO_TIME,TIMER1,#when_to_do_it
```

The command is actually accepted when the HSO_TIME register is written. It is important to ensure that this code piece is not interrupted by any interrupt service routine which might also send a command to the HSO unit. If this happens the HSO will know when to do it but not know what to do when it's time to do it. In many systems this becomes a null problem because HSO commands are only issued from one place in the code. If this is not the case then a critical region must be established and the two instructions executed from within this region (see section 3.6.5).

Commands in the holding register will not execute even if their time tag is reached. Commands must be in the CAM for this to occur. Flags are available in IOS0 which indicate the holding register is empty (IOS0.7) or that both the holding register is empty *and* the CAM is not full (IOS0.6). The programmer should carefully decide which of these two flags is the best to use for each application.

It is possible to enter commands into the CAM which never execute. This occurs if TIMER2 has been set up as a variable modulo counter and a command is entered with a time tag referenced to TIMER2 which has a value that TIMER2 never reaches. The inaccessible command will never execute and continue to take up room in the CAM until either the system is reset or the program allows TIMER2 increment up to the value stored in the time tag. Note that commands cannot be flushed from the CAM without being executed but that they can be cancelled. This is accomplished by setting the opposite command in the CAM to execute at the same time tag as the command to be cancelled. If, as an example, a command has been issued to set HSO.1 when TIMER1 = 1234 then entering a second command which clears HSO.1 when TIMER1 = 1234 will result in a no-operation on HSO.1. Both commands will remain in the CAM until TIMER1 = 1234.

3.7.4. High Speed I/O Interrupts

The HSO unit can generate two types of interrupts. The HSO execution interrupt (vector = (2006H)) is generated (if enabled) for HSO commands which operate on one of the six HSO pins. The other HSO interrupt is the Software Timer interrupt (vector = (200AH)) which is generated (if enabled) for any other HSO command (e.g. triggering the A/D, resetting Timer2 or generating a software time delay).

There are also two interrupts associated with the HSI unit. The HSI data available interrupt (vector = (2004H)) is generated if there is data in the HSI FIFO that the program should read. The other HSI related interrupt is the HSI.0 interrupt which occurs whenever High Speed Input pin 0 makes a zero-to-one transition. This interrupt will become pending in the INT _ PENDING register even if the HSI unit is programmed to ignore changes on HSI.0 or look for a one-to-zero transition.

3.7.5. Accessing Register Mapped I/O

The on-board I/O devices such as the serial port or the A/D converter are controlled as register mapped I/O. This allows convenient and efficient I/O processing. The implementation of the current members of the MCS-96 family place some restrictions on how these registers can be accessed. While these restrictions are not severe, the programmer must be aware of them. A complete listing of these registers is shown in figure 2-7 and 2-8. The restrictions are as follows:

a). TIMER0, TIMER1 and HSI _ TIME are *word read only.* They cannot be read as bytes or written to in any format.

b). HSO _ TIME is *word write only.* It cannot be written to as individual bytes or read in any format.

c). R0 (the ZERO register) is byte or word read or write but writing to it will not change its value.

d). All of the other I/O registers can be accessed only as bytes. This applies even to the AD _ RESULT which is logically a word operand.

3.8. EXAMPLE-1 PROGRAMMING THE SERIAL I/O CHANNEL

```
MCS-96 MACRO ASSEMBLER    SERIAL PORT DEMO PROGRAM

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F1:SPX.SRC
OBJECT FILE: :F1:SPX.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB DEBUG

ERR LOC  OBJECT                LINE        SOURCE STATEMENT
                                 1   $TITLE('SERIAL PORT DEMO PROGRAM')
                                 2   $PAGELENGTH(95)
                                 3
                                 4   ;       This program initializes the serial port and echos any
                                 5   ;       character sent to it.
                                 6
                                 7
      000E                       8   BAUD_REG        equu    0EH
      0011                       9   SPCON           equ     11H
      0011                      10   SPSTAT          equ     11H
      0016                      11   IOC1            equ     16H
      0015                      12   IOC0            equ     15H
      0007                      13   SBUF            equ     07H
      0009                      14   INT_PENDING     equ     09H
      0018                      15   SP              equ     18H
                                16
                                17
    0000                        18   rseg
                                19
    0000                        20           CHR:    dsb     1
    0001                        21           TEMP0:  dsb     1
    0002                        22           TEMP1:  dsb     1
    0003                        23           RCV_FLAG:       dsb     1
                                24
                                25
    0000                        26   cseg
                                27
    0000 A1B00018               28           LD      SP, #0B0H
                                29
    0004 B12016                 30           LDB     IOC1, #00100000B            ; Set P2.0 to TXD
                                31
                                32                   ; Baud rate = input frequency / (64*baud_val)
                                33                   ; baud_val  = (input frequency/64) / baud rate
                                34
                                35
      0027                      36   baud_val        equ     39      ; 2400 baud at 6.0 MHz
                                37
      0080                      38   BAUD_HIGH       equ     ((baud_val-1)/256) OR 80H   ; Set MSB to 1
      0026                      39   BAUD_LOW        equ     (baud_val-1) MOD 256
                                40
                                41
    0007 B1260E                 42           LDB     BAUD_REG, #BAUD_LOW
    000A B1800E                 43           LDB     BAUD_REG, #BAUD_HIGH
                                44
    000D B14911                 45           LDB     SPCON, #01001001B       ; Enable receiver, Mode 1
                                46
                                47                           ; The serial port is now initialized
                                48
                                49
    0010 C40007          R      50           STB     SBUF, CHR               ; Clear serial Port
    0013 B12001          R      51           LDB     TEMP0, #00100000B       ; Set TI-temp
                                52
    0016 3609FD                 53   wait:   JBC     INT_PENDING, 6, wait    ; Wait for pending bit to be set
    0019 71BF09                 54           ANDB    INT_PENDING, #10111111B ; Clear pending bit
                                55
    001C 901101          R      56           ORB     TEMP0, SPCON            ; Put SPCON into temp register
                                57                                           ; This is necessary becase reading
                                58                                           ; SPCON clears TI and RI
                                59
    001F                        60   get_byte:
    001F 360109          R      61           JBC     TEMP0, 6, put_byte      ; If RI-temp is not set
    0022 C40007          R      62           STB     SBUF, CHR               ; Store byte
    0025 71BF01          R      63           ANDB    TEMP0, #10111111B       ; CLR RI-temp
    0028 B1FF03          R      64           LDB     RCV_FLAG, #0FFH         ; Set bit-received flag
                                65
    002B                        66   put_byte:
    002B 30030C          R      67           JBC     RCV_FLAG, 0, continue   ; If receive flag is cleared
    002E 350109          R      68           JBC     TEMP0, 5, continue      ; If TI was not set
    0031 B00007          R      69           LDB     SBUF, CHR               ; Send byte
    0034 71DF01          R      70           ANDB    TEMP0, #11011111B       ; CLR TI-temp
    0037 B10003          R      71           LDB     RCV_FLAG, #00           ; Clear bit-received flag
                                72
    003A                        73   continue:
    003A 27DA                   74           BR      wait
                                75
    003C                        76           END

ASSEMBLY COMPLETED,    NO ERROR(S) FOUND.
```

3.9. EXAMPLE-2 GENERATING A PWM WITH THE HSO UNIT

```
MCS-96 MACRO ASSEMBLER    HSO EXAMPLE PROGRAM FOR PWM OUTPUTS

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F1:HSO2X.SRC
OBJECT FILE: :F1:HSO2X.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB DEBUG

ERR LOC  OBJECT                 LINE        SOURCE STATEMENT
                                   1   $TITLE ('HSO EXAMPLE PROGRAM FOR PWM OUTPUTS')
                                   2   $PAGELENGTH(95)
                                   3
                                   4   ; This program will provide 4 PWM outputs on HSO pins 0-3
                                   5   ; The input parameters passed to the program are:
                                   6   ;
                                   7   ;              HSO_ON_N    HSO on time for pin N
                                   8   ;              HSO_OFF_N   HSO off time for pin N
                                   9
                                  10   ;       Where: Times are in timer1 cycles
                                  11   ;              N takes values from 0 to 3
                                  12
                                  13   ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                                  14
                                  15
    0000                          16   dseg
                                  17
    0000                          18           D_STAT:          DSB     1
                                  19           extrn   HSO_ON_0 :word , HSO_OFF_0 :word
                                  20           extrn   HSO_ON_1 :word , HSO_OFF_1 :word
                                  21           extrn   HSO_ON_2 :word , HSO_OFF_2 :word
                                  22           extrn   HSO_ON_3 :word , HSO_OFF_3 :word
                                  23           extrn   HSO_TIME :word , HSO_COMMAND :byte
                                  24           extrn   TIMER1   :word , IOS0       :byte
                                  25           extrn   SP       :word
                                  26
                                  27
    0000                          28   rseg
                                  29
                                  30           public  OLD_STAT
    0000                          31           OLD_STAT:        dsb     1
    0001                          32           NEW_STAT:        dsb     1
                                  33
                                  34
    0000                          35   cseg
                                  36
                                  37           PUBLIC  wait
                                  38
                                  39
    0000 3E00FD            E      40   wait:   JBS     IOS0, 6, wait                  ; Loop until HSO holding register
    0003 FD                       41           NOP                                    ; is empty
    0004 FD                       42           NOP
    0005 C701000000        E      43           STB     IOS0, D_STAT                   ; Load byte to external RAM
                                  44
                                  45                   ; For opperation with interrupts 'store_stat:' would be the
                                  46                   ; entry point of the routine.
                                  47                   ; Note that a DI or PUSHF might have to be added.
                                  48
    000A                          49   store_stat:
    000A 510F0001          E      50           ANDB    NEW_STAT, IOS0, #0FH           ; Store new status of HSO
    000E 980100            R      51           CMPB    OLD_STAT, NEW_STAT
    0011 DFED                     52           JE      wait                           ; If status hasn't changed
    0013 940100            R      53           XORB    OLD_STAT, NEW_STAT
                                  54
                                  55
    0016                          56   check_0:
    0016 300017            R      57           JBC     OLD_STAT, 0, check_1           ; Jump if OLD_STAT(0)=NEW_STAT(0)
    0019 38010B            R      58           JBS     NEW_STAT, 0, set_off_0
                                  59
    001C                          60   set_on_0:
    001C B13000            E      61           LDB     HSO_COMMAND, #00110000B        ; Set HSO for timer1, set pin 0
    001F 470100000000      E      62           ADD     HSO_TIME, TIMER1, HSO_OFF_0    ; Time to set pin = Timer1 value
    0025 2009                     63           BR      check_1                        ;  + Time for pin to be low
                                  64
    0027                          65   set_off_0:
    0027 B11000            E      66           LDB     HSO_COMMAND, #00010000B        ; Set HSO for timer1, clear pin 0
    002A 470100000000      E      67           ADD     HSO_TIME, TIMER1, HSO_ON_0     ; Time to clear pin = Timer1 value
                                  68                                                  ;  + Time for pin to be high
                                  69
    0030                          70   check_1:
    0030 310017            R      71           JBC     OLD_STAT, 1, check_2           ; Jump if OLD_STAT(1)=NEW_STAT(1)
    0033 39010B            R      72           JBS     NEW_STAT, 1, set_off_1
                                  73
    0036                          74   set_on_1:
    0036 B13100            E      75           LDB     HSO_COMMAND, #00110001B        ; Set HSO for timer1, set pin 1
    0039 470100000000      E      76           ADD     HSO_TIME, TIMER1, HSO_OFF_1    ; Time to set pin = Timer1 value
    003F 2009                     77           BR      check_2                        ;  + Time for pin to be low
                                  78
    0041                          79   set_off_1:
    0041 B11100            E      80           LDB     HSO_COMMAND, #00010001B        ; Set HSO for timer1, clear pin 1
    0044 470100000000      E      81           ADD     HSO_TIME, TIMER1, HSO_ON_1     ; Time to clear pin = Timer1 value
                                  82                                                  ;  + Time for pin to be high
                                  83
                                  84   $EJECT
```

```
MCS-96 MACRO ASSEMBLER    HSO EXAMPLE PROGRAM FOR PWM OUTPUTS

ERR LOC  OBJECT                LINE        SOURCE STATEMENT
                                85
    004A                        86   check_2:
    004A 320017         R       87          JBC     OLD_STAT, 2, check_3           ; Jump if OLD_STAT(2)=NEW_STAT(2)
    004D 3A010B         R       88          JBS     NEW_STAT, 2, set_off_2
                                89
    0050                        90   set_on_2:
    0050 B13200         E       91          LDB     HSO_COMMAND, #00110010B        ; Set HSO for timer1, set pin 2
    0053 470100000000   E       92          ADD     HSO_TIME, TIMER1, HSO_OFF_2    ; Time to set pin = Timer1 value
    0059 2009                   93          BR      check_3                        ;  + Time for pin to be low
                                94
    005B                        95   set_off_2:
    005B B11200         E       96          LDB     HSO_COMMAND, #00010010B        ; Set HSO for timer1, clear pin 2
    005E 470100000000   E       97          ADD     HSO_TIME, TIMER1, HSO_ON_2     ; Time to clear pin = Timer1 value
                                98                                                 ;  + Time for pin to be high
                                99
                               100
    0064                       101   check_3:
    0064 330017         R      102          JBC     OLD_STAT, 3, check_done        ; Jump if OLD_STAT(3)=NEW_STAT(3)
    0067 3B010B         R      103          JBS     NEW_STAT, 3, set_off_3
                               104
    006A                       105   set_on_3:
    006A B13300         E      106          LDB     HSO_COMMAND, #00110011B        ; Set HSO for timer1, set pin 3
    006D 470100000000   E      107          ADD     HSO_TIME, TIMER1, HSO_OFF_3    ; Time to set pin = Timer1 value
    0073 2009                  108          BR      check_done                     ;  + Time for pin to be low
                               109
    0075                       110   set_off_3:
    0075 B11300         E      111          LDB     HSO_COMMAND, #00010011B        ; Set HSO for timer1, clear pin 3
    0078 470100000000   E      112          ADD     HSO_TIME, TIMER1, HSO_ON_3     ; Time to clear pin = Timer1 value
                               113                                                 ;  + Time for pin to be high
                               114
                               115
    007E                       116   check_done:
    007E B00100         R      117          LDB     OLD_STAT, NEW_STAT             ; Store current status and
                               118                                                 ; wait for interrupt flag
    0081 F0                    119          RET
                               120
                               121
    0082                       122          END

ASSEMBLY COMPLETED,    NO ERROR(S) FOUND.
```

3.10. EXAMPLE-3 MEASURING PULSES WITH THE HSI UNIT

```
MCS-96 MACRO ASSEMBLER    MEASURING PULSES USING THE HSI UNIT

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F1:PULSEX.SRC
OBJECT FILE: :F1:PULSEX.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB DEBUG

ERR LOC  OBJECT                LINE        SOURCE STATEMENT
                                  1   $TITLE('MEASURING PULSES USING THE HSI UNIT')
                                  2   $PAGELENGTH(95)
                                  3
                                  4   ;       This program measures pulsewidths in TIMER1 cycles
                                  5   ;       and retuns the values in external RAM.
                                  6
                                  7
      0018                        8   SP              equ      18H
      0003                        9   HSI_MODE        equ      03H
      0006                       10   HSI_STATUS      equ      06H
      0004                       11   HSI_TIME        equ      04H
      000A                       12   TIMER1          equ      0AH
      0015                       13   IOC0            equ      15H
      0016                       14   IOS1            equ      16H
                                 15
                                 16
    0000                         17   rseg
                                 18
    0000                         19           HIGH_TIME:      dsw     1
    0002                         20           LOW_TIME:       dsw     1
    0004                         21           PERIOD:         dsw     1
    0006                         22           HI_EDGE:        dsw     1
    0008                         23           LO_EDGE:        dsw     1
                                 24
                                 25
    000A                         26           AX:     dsw     1
      000A                       27           AL      equ     AX      :byte
      000B                       28           AH      equ     (AX+1)  :byte
                                 29
    000C                         30           BX:     dsw     1
      000C                       31           BL      equ     BX      :byte
      000D                       32           BU      equ     (BX+1)  :byte     ; Note that 'BH' is an opcode so it
                                 33                                             ; can't be used as a label
                                 34
    0000                         35   cseg
                                 36
    0000 A1C00018                37           LD      SP, #0C0H
    0004 B10115                  38           LDB     IOC0, #00000001B          ; Enable HSI 0
    0007 B10F03                  39           LDB     HSI_MODE, #00001111B      ; HSI 0 look for either edge
                                 40
    000A 44020004          R     41   wait:   ADD     PERIOD, HIGH_TIME, LOW_TIME
                                 42
    000E 3716F9                  43           JBC     IOS1, 7, wait   ; Wait while no pulse is entered
                                 44
    0011 B0060A            R     45           LDB     AL, HSI_STATUS            ; Load status; Note that reading
                                 46                                             ;   HSI_TIME clears HSI_STATUS
                                 47
    0014 A0040C            R     48           LD      BX, HSI_TIME              ; Load the HSI_TIME
                                 49
    0017 390A09            R     50           JBS     AL, 1, hsi_hi             ; Jump if HSI.0 is high
                                 51
    001A C0080C            R     52   hsi_lo: ST      BX, LO_EDGE
    001D 48060800          R     53           SUB     HIGH_TIME, LO_EDGE, HI_EDGE
    0021 27E7                    54           BR      wait
                                 55
                                 56
    0023 C0060C            R     57   hsi_hi: ST      BX, HI_EDGE
    0026 48080602          R     58           SUB     LOW_TIME, HI_EDGE, LO_EDGE
    002A 27DE                    59           BR      wait
                                 60
    002C                         61           END

ASSEMBLY COMPLETED,    NO ERROR(S) FOUND.
```

3.11. EXAMPLE-4 SCANNING THE A/D CHANNELS

```
MCS-96 MACRO ASSEMBLER    SCANNING THE A TO D CHANNELS

SERIES-III MCS-96 MACRO ASSEMBLER,'V1.0

SOURCE FILE: :F1:ATODX.SRC
OBJECT FILE: :F1:ATODX.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB DEBUG

ERR LOC  OBJECT                LINE        SOURCE STATEMENT
                                  1   $TITLE('SCANNING THE A TO D CHANNELS')
                                  2   $PAGELENGTH(95)
                                  3
                                  4   ;       This program scans A to D lines 0 through 3 and stores the
                                  5   ;       results in RESULT_N
                                  6
      0002                        7   AD_RESULT_LO    equ     02
      0003                        8   AD_RESULT_HI    equ     03
      0002                        9   AD_COMMAND      equ     02
      0018                       10   SP              equ     18H
                                 11
                                 12
    0000                         13   dseg
                                 14
    0000                         15   RESULT_TABLE:
    0000                         16           RESULT_1:       dsw     1
    0002                         17           RESULT_2:       dsw     1
    0004                         18           RESULT_3:       dsw     1
    0006                         19           RESULT_4:       dsw     1
                                 20
    0000                         21   rseg
                                 22
    0000                         23           AX:     dsw     1
      0000                       24           AL      equ     AX      :byte
      0001                       25           AH      equ     (AX+1)  :byte
                                 26
    0002                         27           BX:     dsw     1
      0002                       28           BL      equ     BX      :byte
      0003                       29           BU      equ     (BX+1)  :byte   ; Note that 'BH' is an opcode so it
                                 30                                           ; can't be used as a label
                                 31
                                 32
    0004                         33           DX:     dsw     1
      0004                       34           DL      equ     DX      :byte
      0005                       35           DH      equ     (DX+1)  :byte
                                 36
                                 37
    0000                         38   cseg
                                 39
                                 40
    0000 A1C00018                41   start:  LD      SP, #0C0H       ; Set Stack Pointer
    0004 A00002           R      42           LD      BX, 00H         ; Use the zero register
                                 43
    0007 910802           R      44   next:   ORB     BL, #1000B      ; Start conversion on channel
    000A B00202           R      45           LDB     AD_COMMAND, BL  ; indicated by BL register
    000D 710702           R      46           ANDB    BL, #0111B
                                 47
    0010 FD                      48           NOP             ; Wait for conversion to start
                                 49
    0011 3B02FD                  50   check:  JBS     AD_RESULT_LO, 3, check  ; Wait while A to D is busy
                                 51

    0014 B00200           R      52           LDB     AL, AD_RESULT_LO        ; Load low order result
    0017 B00301           R      53           LDB     AH, AD_RESULT_HI        ; Load high order result
                                 54
    001A 54020204         R      55           ADDB    DL, BL, BL              ; DL=BL*2
    001E AC0404           R      56           LDBZE   DX, DL
    0021 C304000000       R      57           ST      AX, RESULT_TABLE[DX]    ; Store result indexed by BL*2
                                 58
    0026 1702             R      59           INCB    BL
    0028 710302           R      60           ANDB    BL, #03H
                                 61
    002B 27DA                    62           BR      next
                                 63
    002D                         64           END

ASSEMBLY COMPLETED,   NO ERROR(S) FOUND.
```

3.12. EXAMPLE-5 TABLE LOOKUP-AND INTERPOLATION

```
MCS-96 MACRO ASSEMBLER    TABLE LOOKUP AND INTERPOLATION

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F1:INTERX.SRC
OBJECT FILE: :F1:INTERX.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB DEBUG

ERR LOC  OBJECT                LINE        SOURCE STATEMENT
                                 1    $TITLE('TABLE LOOKUP AND INTERPOLATION')
                                 2    $PAGELENGTH(95)
                                 3
                                 4    ;       This program uses a lookup table to generate 12-bit function values
                                 5    ;       using 8-bit input values.  The table is 16 bytes long and 16 bits wide.
                                 6    ;       A linear interpolation is made using the following fomula:
                                 7
                                 8    ;              Table_Step          IN_VAL - TABLE_LOW
                                 9    ;       ---------------------- = ------------------ =
                                10    ;       TABLE_HIGH - TABLE_LOW       OUT - TABLE_LOW
                                11    ;
                                12    ;          16        IN_DIF
                                13    ;       ------- = -------
                                14    ;       TAB_DIF   OUT_DIF
                                15
                                16    ;       Cross Multiplication is used to solve for OUT_DIF
                                17
                                18
                                19
      0018                      20    SP      equ     18H
                                21
                                22
    0000                        23    dseg
                                24
    0000                        25    RESULT_TABLE:
    0000                        26            RESULT:         dsw     1
                                27
                                28
    0000                        29    rseg
                                30
    0000                        31            AX:     dsw     1
      0000                      32            AL      equ     AX      :byte
      0001                      33            AH      equ     (AX+1)  :byte
                                34
    0002                        35            BX:     dsw     1
      0002                      36            BL      equ     BX      :byte
      0003                      37            BU      equ     (BX+1)  :byte      ; Note that 'BH' is an opcode so it
                                38                                               ; can't be used as a label
                                39
    0004                        40            IN_VAL:         dsb     1
    0006                        41            TABLE_LOW:      dsw     1
    0008                        42            TABLE_HIGH:     dsw     1
    000A                        43            IN_DIF:         dsw     1
      000A                      44            IN_DIFB         equ     IN_DIF  :byte
    000C                        45            TAB_DIF:        dsw     1
    000E                        46            OUT:            dsw     1
    0010                        47            OUT_DIF:        dsl     1
                                48
                                49
    0000                        50    cseg
                                51
                                52
    0000 A1C00018               53    start:  LD      SP, #0C0H       ; Set Stack Pointer
                                54
                                55
    0004 B00400          R      56    look:   LDB     AL, IN_VAL
    0007 180300          R      57            SHRB    AL, #3          ; Place 2 times the upper nibble in byte
    000A 71FE00          R      58            ANDB    AL, #11111110B  ; Insure AL is a word address
    000D AC0000          R      59            LDBZE   AX, AL
                                60
    0010 A300420006      R      61            LD      TABLE_LOW, TABLE [AX]    ; TABLE_LOW is table output value
                                62                                             ; of IN_VAL rounded down to the
                                63                                             ; nearest multiple of 10H.
                                64
    0015 A300440008      R      65            LD      TABLE_HIGH, (TABLE+2) [AX]  ; TABLE_HIGH is the table output
                                66                                             ; value of IN_VAL rounded up to the
                                67                                             ; nearest multiple of 10H.
                                68
                                69
    001A 4806080C        R      70            SUB     TAB_DIF, TABLE_HIGH, TABLE_LOW
                                71
    001E 510F040A        R      72            ANDB    IN_DIFB, IN_VAL, #0FH
    0022 BC0A0A          R      73            LDBSE   IN_DIF, IN_DIFB          ; Make input difference into a word
                                74
    0025 FE4C0C0A10      R      75            MUL     OUT_DIF, IN_DIF, TAB_DIF
    002A FE8D100010      R      76            DIV     OUT_DIF, #16
                                77
    002F 4406100E        R      78    labl:   ADD     OUT, OUT_DIF, TABLE_LOW ; Add output difference to output
                                79                                             ; generated with truncated IN_VAL
                                80                                             ; as input
    0033 08040E          R      81            SHR     OUT, #4                  ; Round to 12-bit answer
```

```
MCS-96 MACRO ASSEMBLER    TABLE LOOKUP AND INTERPOLATION                                      10/11/83

ERR LOC  OBJECT                LINE        SOURCE STATEMENT
    0036 D307                   82          JNC     lab2
    0038 070E           R       83          INC     OUT                        ; Round up if Carry = 1
    003A C30100000E     R       84          ST      OUT, RESULT
                                85
    003F 27C3                   86   lab2:  BR      look
                                87
                                88
    0041                        89   cseg
                                90
    0042 000000200034004C       91   table: DCW     0000H, 2000H, 3400H, 4C00H       ; A random non-monotonic
    004A 005D006A00720078       92          DCW     5D00H, 6A00H, 7200H, 7800H       ; function
    0052 007B007D0076006D       93          DCW     7B00H, 7D00H, 7600H, 6D00H
    005A 005D004B00340022       94          DCW     5D00H, 4B00H, 3400H, 2200H
    0062 0010                   95          DCW     1000H
                                96
    0064                        97          END

ASSEMBLY COMPLETED,   NO ERROR(S) FOUND.
```

3.13. DETAILED INSTRUCTION SET DESCRIPTION

This section gives a description of each instruction recognized by the 8096 sorted alphabetically by the mnemonic used in the assembly language for the 8096. Note that the effect on the program counter (PC) is not always shown in the instruction descriptions. All instructions increment the PC by the number of bytes in the instruction. Several acronynms are used in the instruction set descriptions which are defined here:

aa. A two bit field within an opcode which selects the basic addressing mode user. This field is only present in those opcodes which allow address mode options. The encoding of the field is as follows:

aa	Addressing mode
00	Register direct
01	Immediate
10	Indirect
11	Indexed

The selection between indirect and indirect with auto-increment or between short and long indexing is done based on the least significant bit of the instruction byte which follows the opcode. This type selects the 16-bit register which is to take part in the address calculation. Since the 8096 requires that words be aligned on even byte boundaries this bit would be otherwise unused.

breg. A byte register in the internal register file. When confusion could exist as to whether this field refers to a source or a destination register it will be prefixed with an "S" or a "D."

baop. A byte operand which is addressed by any of the address mooes discussed in section 3.2.

bitno. A three bit field within an instruction op-code which selects one of the eight bits in a byte.

wreg. A word register in the internal register file. When confusion could exist as to whether this field refers to a source register or a destination register it will be prefixed with an "S" or a "D."

waop. A word operand which is addressed by any of the address modes discussed in section 3.2.

Ireg. A 32-bit register in the internal register file.

BEA. Extra bytes of code required for the address mode selected.

CEA. Extra state times (cycles) required for the address mode selected.

cadd An address in the program code.

Flag Settings. The modification to the flag setting is shown for each instruction. A checkmark (√) means that the flag is set or cleared as appropriate. A hyphen means that the flag is not modified. A one or zero (1) or (0) indicates that the flag will be in that state after the instruction. An up arrow (↑) indicates that the instruction may set the flag if it is appropriate but will not clear the flag. A down arrow (↓) indicates that the flag can be cleared but not set by the instruction.

Generic Jumps and Calls. The assembler for the 8096 provides for generic jumps and calls. For all of the conditional jump instructions a "B" can be substituted for the "J" and the assembler will generate a code sequence which is logically equivalent but can reach anywhere in the memory. A JH can only jump about 128 locations from the current program counter; a BH can jump anywhere in memory. In a like manner a BR will cause a SJMP or LJMP to be generated as appropriate and a CALL will cause a SCALL or LCALL to be generated. The assembler user guide (see section 3.0) should be consulted for the algorithms used by the assembler to convert these generic instructions into actual machine instructions.

3.13.1. ADD (Two Operands) — ADD WORDS

Operation: The sum of the two word operands is stored into the destination (leftmost) operand.

(DEST) ← (DEST) + (SRC)

Assembly Language Format:

```
        DST     SRC
ADD     wreg,   waop
```

Object Code Format: [011001aa][waop][wreg]

Bytes: 2+BEA
States: 4+CEA

Flags Affected					
Z	N	C	V	VT	ST
√	√	√	√	↑	–

3.13.2. ADD (Three Operands) — ADD WORDS

Operation: The sum of the second and third word operands is stored into the destination (leftmost) operand.

(DEST) ← (SRC1) + (SRC2)

Assembly Language Format:

```
        DST      SRC1     SRC2
ADD     Dwreg,   Swreg,   waop
```

Object Code Format: [010001aa][waop][Swreg][Dwreg]

Bytes: 3+BEA
States: 5+CEA

Flags Affected					
Z	N	C	V	VT	ST
√	√	√	√	↑	–

3.13.3. ADDB (Two Operands) — ADD BYTES

Operation: The sum of the two byte operands is stored into the destination (leftmost) operand.

(DEST) ← (DEST) + (SRC)

Assembly Language Format:

```
      DST    SRC
ADDB  breg,  boap
```

Object Code Format: [011101aa][baop][breg]

Bytes: 2+BEA
States: 4+CEA

Flags Affected

Z	N	C	V	VT	ST
√	√	√	√	↑	–

3.13.4. ADDB (Three Operands) — ADD BYTES

Operation: The sum of the second and third byte operands is stored into the destination (leftmost) operand.

(DEST) ← (SRC1) + (SRC2)

Assembly Language Format:

```
      DST     SRC1    SRC2
ADDB  Dbreg,  Sbreg,  baop
```

Object Code Format: [010101aa][baop][Sbreg][Dbreg]

Bytes: 3+BEA
States: 5+CEA

Flags Affected

Z	N	C	V	VT	ST
√	√	√	√	↑	–

3.13.5. ADDC — ADD WORDS WITH CARRY

Operation: The sum of the two word operands and the carry flag (0 or 1) is stored into the destination (leftmost) operand.

(DEST) ← (DEST) + (SRC) + C

Assembly Language Format:

```
      DST    SRC
ADDC  wreg,  waop
```

Object Code Format: [101001aa][waop][wreg]

Bytes: 2+BEA
States: 4+BEA

Flags Affected

Z	N	C	V	VT	ST
↓	✓	✓	✓	↑	–

3.13.6. ADDCB — ADD BYTES WITH CARRY

Operation: The sum of the two byte operands and the carry flag (0 or 1) is stored into the destination (leftmost) operand.

(DEST) ← (DEST) + (SRC) + C

Assembly Language Format:

```
       DST    SRC
ADDCB  breg,  baop
```

Object Code Format: [101101aa][baop][breg]

Bytes: 2+BEA
States: 4+CEA

Flags Affected

Z	N	C	V	VT	ST
↓	✓	✓	✓	↑	–

3.13.7. AND (Two Operands) — LOGICAL AND WORDS

Operation: The two word operands are ANDed, the result having a 1 only in those bit positions where both operands had a 1, with zeroes in all other bit positions. The result is stored into the destination (leftmost) operand.

(DEST) ← (DEST) AND (SRC)

Assembly Language Format:

```
     DST     SRC
AND  wreg,   waop
```

Object Code Format: [011000aa][waop][wreg]

Bytes: 2+BEA
States: 4+CEA

Flags Affected					
Z	N	C	V	VT	ST
√	√	0	0	–	–

3.13.8. AND (Three Operands) — LOGICAL AND WORDS

Operation: The second and third word operands are ANDed, the result having a 1 only in those bit positions where both operands had a 1, with zeroes in all other bit positions. The result is stored into the destination (leftmost) operand.

(DEST) ← (SRC1) AND (SRC2)

Assembly Language Format:

```
     DST     SRC1    SRC2
AND  Dwreg,  Swreg,  waop
```

Object Code Format: [010000aa][waop][Swreg][Dwreg]

Bytes: 3+BEA
States: 5+CEA

Flags Affected					
Z	N	C	V	VT	ST
√	√	0	0	–	–

3.13.9. ANDB (Two Operands) — LOGICAL AND BYTES

Operation: The two byte operands are ANDed, the result having a 1 only in those bit positions where both operands had a 1, with zeroes in all other bit positions. The result is stored into the destination (leftmost) operand.

(DEST) ← (DEST) AND (SRC)

Assembly Language Format:

```
        DST     SRC
ANDB    breg,   baop
```

Object Code Format: [011100aa][baop][breg]

Bytes: 2+BEA
States: 4+CEA

Flags Affected					
Z	N	C	V	VT	ST
√	√	0	0	–	–

3.13.10. ANDB (Three Operands) — LOGICAL AND BYTES

Operation: The second and third word operands are ANDed, the result having a 1 only in those bit positions where both operands had a 1, with zeroes in all other bit positions. The result is stored into the destination (leftmost) operand.

(DEST) ← (SRC1) AND (SRC2)

Assembly Language Format:

```
        DST      SRC1     SRC2
ANDB    Dbreg,   Sbreg,   baop
```

Object Code Format: [010100aa][baop][Sbreg][Dbreg]

Bytes: 3+BEA
States: 5+CEA

Flags Affected					
Z	N	C	V	VT	ST
√	√	0	0	–	–

3.13.11. BR (Indirect) — BRANCH INDIRECT

Operation: The execution continues at the address specified in the operand word register.

PC ← (DEST)

Assembly Language Format: BR [wreg]

Object Code Format: [11100011][wreg]

Bytes: 2
States: 8

Flags Affected					
Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.12. CLR — CLEAR WORD

Operation: The value of the word operand is set to zero.

(DEST) ← 0

Assembly Language Format: CLR wreg

Object Code Format: [00000001][wreg]

Bytes: 2
States: 4

Flags Affected					
Z	N	C	V	VT	ST
1	0	0	0	–	–

3.13.13. CLRB — CLEAR BYTE

Operation: The value of the byte operand is set to zero.

(DEST) ← 0

Assembly Language Format: CLRB breg

Object Code Format: [00010001] [breg]

Bytes: 2
States: 4

Flags Affected

Z	N	C	V	VT	ST
1	0	0	0	–	–

3.13.14. CLRC — CLEAR CARRY FLAG

Operation: The value of the carry flag is set to zero.

C ← 0

Assembly Language Format: CLRC

Object Code Format: [11111000]

Bytes: 1
States: 4

Flags Affected

Z	N	C	V	VT	ST
–	–	0	–	–	–

3.13.15. CLRVT — CLEAR OVERFLOW TRAP

Operation: The value of the overflow-trap flag is set to zero.

VT ← 0

Assembly Language Format: CLRVT

Object Code Format: [11111100]

Bytes: 1
States: 4

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	0	–

3.13.16. CMP — COMPARE WORDS

Operation: The source (rightmost) word operand is subtracted from the destination (leftmost) word operand. The flags are altered but the operands remain unaffected. The carry flag is set as complement of borrow.

(DEST) — (SRC)

Assembly Language Format:

```
      DST    SRC
CMP   wreg,  waop
```

Object Code Format: [100010aa][waop][wreg]

Bytes: 2+BEA
States: 4+CEA

Flags Affected

Z	N	C	V	VT	ST
√	√	√	√	↑	–

3.13.17. CMPB — COMPARE BYTES

Operation: The source (rightmost) byte operand is subtracted from the destination (leftmost) byte operand. The flags are altered but the operands remain unaffected. The carry flag is set as complement of borrow.

(DEST) — (SRC)

Assembly Language Format:

```
        DST     SRC
CMPB    breg,   baop
```

Object Code Format: [100110aa][baop][breg]

Bytes: 2+BEA
States: 4+CEA

Flags Affected

Z	N	C	V	VT	ST
√	√	√	√	↑	–

3.13.18. DEC — DECREMENT WORD

Operation: The value of the word operand is decremented by one.

(DEST) ← (DEST) — 1

Assembly Language Format: DEC wreg

Object Code Format: [00000101][wreg]

Bytes: 2
States: 4

Flags Affected

Z	N	C	V	VT	ST
√	√	√	√	↑	–

3.13.19. DECB — DECREMENT BYTE

Operation: The value of the byte operand is decremented by one.

(DEST) ← (DEST) — 1

Assembly Language Format: DECB breg

Object Code Format: [00010101][breg]

Bytes: 2
States: 4

Flags Affected

Z	N	C	V	VT	ST
√	√	√	√	↑	–

3.13.20. DI — DISABLE INTERRUPTS

Operation: Interrupts are disabled. Interrupt-calls will not occur after this instruction.

Interrupt Enable (PSW.9) ← 0

Assembly Language Format: DI

Object Code Format: [11111010]

Bytes: 1
States: 4

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.21. DIV — DIVIDE INTEGERS

Operation: This instruction divides the contents of the destination LONG-INTEGER operand by the contents of the INTEGER word operand, using signed arithmetic. The low order word of the destination (i.e., the word with the lower address) will contain the quotient; the high order word will contain the remainder.

(low word DEST) ← (DEST) / (SRC)
(high word DEST) ← (DEST) MOD (SRC)
The above two statements are performed concurrently.

Assembly Language Format:

```
      DST    SRC
DIV   lreg,  waop
```

Object Code Format: [11111110][100011aa][waop][lreq]

Bytes: 2+BEA
States: 29 + CEA

Flags Affected

Z	N	C	V	VT	ST
–	–	–	√	↑	–

3.13.22. DIVB — DIVIDE SHORT-INTEGERS

Operation: This instruction divides the contents of the destination INTEGER operand by the contents of the source SHORT-INTEGER operand, using signed arithmetic. The low order byte of the destination (i.e. the byte with the lower address) will contain the quotient; the high order byte will contain the remainder.

(low byte DEST) ← (DEST) / (SRC)
(high byte DEST) ← (DEST) MOD (SRC)
The above two statements are performed concurrently.

Assembly Language Format:

```
       DST    SRC
DIVB   wreg,  baop
```

Object Code Format: [11111110][100111aa][baop][wreg]

Bytes: 2+BEA
States: 21 + CEA

Flags Affected

Z	N	C	V	VT	ST
–	–	–	√	↑	–

3.13.23. DIVU — DIVIDE WORDS

Operation: This instruction divides the contents of the destination DOUBLE-WORD operand by the contents of the source WORD operand, using unsigned arithmetic. The low order word will contain the quotient; the high order byte will contain the remainder.

(low word DEST) ← (DEST) / (SRC)
(high word DEST) ← MOD (SRC)
The above two statements are performed concurrently.

Assembly Language Format:

```
        DST     SRC
DIVU    lreg,   waop
```

Object Code Format: [100011aa][waop][lreq]

Bytes: 2+BEA
States: 25 + CEA

Flags Affected					
Z	N	C	V	VT	ST
–	–	–	√	↑	–

3.13.24. DIVUB — DIVIDE BYTES

Operation: This instruction divides the contents of the destination WORD operand by the contents of the source BYTE operand, using unsigned arithmetic. The low order byte of the destination (i.e., the byte with the lower address) will contain the quotient; the high order byte will contain the remainder.

(low byte DEST) ← (DEST) / (SRC)
(high byte DEST) ← (DEST) MOD (SRC)
The above two statements are performed concurrently.

Assembly Language Format:

```
        DST     SRC
DIVUB   wreg,   baop
```

Object Code Format: [100111aa][baop][wreg]

Bytes: 2+BEA
States: 17 + CEA

Flags Affected					
Z	N	C	V	VT	ST
–	–	–	√	↑	–

3.13.25. DJNZ — DECREMENT AND JUMP IF NOT ZERO

Operation: The value of the byte operand is decremented by 1. If the result is not equal to 0, the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the result of the decrement is zero then control passes to the next sequential instruction.

```
(COUNT) ← (COUNT) — 1
if (COUNT) < > 0 then
  PC ← PC + disp (sign-extended to 16 bits)
end_if
```

Assembly Language Format: DJNZ breg,cadd

Object Code Format: [11100000][breg][disp]

Bytes: 3
States: Jump Not Taken: 5
Jump Taken: 9

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.26. EI — ENABLE INTERRUPTS

Operation: Interrupts are enabled following the execution of the next statement. Interrupt-calls cannot occur immediately following this instruction.

Interrupt Enable (PSW.9) ← 1

Assembly Language Format: EI

Object Code Format: [11111011]

Bytes: 1
States: 4

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.27. EXT — SIGN EXTEND INTEGER INTO LONG-INTEGER

Operation: The low order word of the operand is sign-extended throughout the high order word of the operand.

```
if (low word DEST)<8000H then
   (high word DEST) ← 0
else
   (high word DEST) ← 0FFFFH
end_if
```

Assembly Language Format: EXT lreg

Object Code Format: [00000110][lreg]

Bytes: 2
States: 4

Flags Affected					
Z	N	C	V	VT	ST
√	√	0	0	–	–

3.13.28. EXTB — SIGN EXTEND SHORT-INTEGER INTO INTEGER

Operation: The low order byte of the operand is sign-extended throughout the high order byte of the operand.

```
if (low byte DEST)<80H then
   (high byte DEST) ← 0
else
   (high byte DEST) ← 0FFH
end_if
```

Assembly Language Format: EXTB wreg

Object Code Format: [00010110][wreg]

Bytes: 2
States: 4

Flags Affected					
Z	N	C	V	VT	ST
√	√	0	0	–	–

3.13.29. INC — INCREMENT WORD

Operation: The value of the word operand is incremented by 1.

(DEST) ← (DEST) + 1

Assembly Language Format: INC wreg

Object Code Format: [00000111][wreg]

Bytes: 2
States: 4

Flags Affected

Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	–

3.13.30. INCB — INCREMENT BYTE

Operation: The value of the byte operand is incremented by 1.

(DEST) ← (DEST) + 1

Assembly Language Format: INCB breg

Object Code Format: [00010111][breg]

Bytes: 2
States: 4

Flags Affected

Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	–

3.13.31. JBC — JUMP IF BIT CLEAR

Operation: The specified bit is tested. If it is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the bit is set (i.e., 1), control passes to the next sequential instruction.

```
if (specified bit) = 0 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JBC breg,bitno,cadd

Object Code Format: [00110bbb][breg][disp]

where bbb is the bit number within the specified register.

Bytes:		3
States:	Jump Not Taken:	5
	Jump Taken:	9

Flags Affected

Z	N	C	V	VT	ST
—	—	—	—	—	—

3.13.32. JBS — JUMP IF BIT SET

Operation: The specified bit is tested. If it is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the bit is clear (i.e., 0), control passes to the next sequential instruction.

```
if (specified bit) = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JBS breg,bitno,cadd

Object Code Format: [00111bbb][breg][disp]

where bbb is the bit number within the specified register.

Bytes: 3
States: Jump Not Taken: 5
Jump Taken: 9

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.33. JC — JUMP IF CARRY FLAG IS SET

Operation: If the carry flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the carry flag is clear (i.e., 0), control passes to the next sequential instruction.

```
if C = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JC cadd

Object Code Format: [11011011][disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.34. JE — JUMP IF EQUAL

Operation: If the zero flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the zero flag is clear (i.e., 0), control passes to the next sequential instruction.

```
if Z = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JE cadd

Object Code Format: [11011111][disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.35. JGE — JUMP IF SIGNED GREATER THAN OR EQUAL

Operation: If the negative flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the negative flag is set (i.e., 1), control passes to the next sequential instruction.

```
if N = 0 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JGE cadd

Object Code Format: [11010110][disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.36. JGT — JUMP IF SIGNED GREATER THAN

Operation: If both the negative flag and the zero flag are clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If either the negative flag or the zero flag are set (i.e., 1,) control passes to the next sequential instruction.

```
if N = 0 AND Z = 0 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JGT cadd

Object Code Format: [11010010][disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.37. JH — JUMP IF HIGHER (UNSIGNED)

Operation: If the carry flag is set (i.e., 1), but the zero flag is not, the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If either the carry flag is clear or the zero flag is set, control passes to the next sequential instruction.

```
if C = 1 and Z = 0 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JH cadd

Object Code Format: [11011001][disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.38. JLE — JUMP IF SIGNED LESS THAN OR EQUAL

Operation: If either the negative flag or the zero flag are set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If both the negative flag and the zero flag are clear (i.e, 0), control passes to the next sequential instruction.

```
if N = 1 OR Z = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JLE cadd

Object Code Format: [11011010][disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected

Z	N	C	V	VT	ST
—	—	—	—	—	—

3.13.39. JLT — JUMP IF SIGNED LESS THAN

Operation: If the negative flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the negative flag is clear (i.e., 0), control passes to the next sequential instruction.

```
if N = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JLT cadd

Object Code Format: [11011110][disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected

Z	N	C	V	VT	ST
—	—	—	—	—	—

3.13.40. JNC — JUMP IF CARRY FLAG IS CLEAR

Operation: If the carry flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the carry flag is set (i.e., 1), control passes to the next sequential instruction.

```
if C = 0 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JNC cadd

Object Code Format: [11010011][disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.41. JNE — JUMP IF NOT EQUAL

Operation: If the zero flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the zero flag is set (i.e., 1), control passes to the next sequential instruction.

```
if Z = 0 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JNE cadd

Object Code Format: [11010111][disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.42. JNH — JUMP IF NOT HIGHER (UNSIGNED)

Operation: If either the carry flag is clear (i.e., 0), or the zero flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the carry flag is set (i.e., 1), or the zero flag is not, control passes to the next sequential instruction.

```
if C = 0 OR Z = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JNH cadd

Object Code Format: [11010001][disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.43. JNST — JUMP IF STICKY BIT IS CLEAR

Operation: If the sticky bit flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the sticky bit flag is set (i.e., 1), control passes to the next sequential instruction.

```
if ST = 0 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JNST cadd

Object Code Format: [11010000][disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.44. JNV — JUMP IF OVERFLOW FLAG IS CLEAR

Operation: If the overflow flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the overflow flag is set (i.e., 1), control passes to next sequential instruction.

```
if V = 0 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JNV cadd

Object Code Format: [11010101][disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.45. JNVT — JUMP IF OVERFLOW TRAP IS CLEAR

Operation: If the overflow trap flag is clear (i.e., 0), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the overflow trap flag is set (i.e., 1), control passes to the next sequential instruction.

```
if VT = 0 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JNVT cadd

Object Code Format: [11010100][disp]

Bytes: 2
States: Jump Not Taken: 4
Jumps Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
–	–	–	–	0	–

3.13.46. JST — JUMP IF STICKY BIT IS SET

Operation: If the sticky bit flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the sticky bit flag is clear (i.e., 0), control passes to the next sequential instruction.

```
if ST = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JST cadd

Object Code Format: [11011000][disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.47. JV — JUMP IF OVERFLOW FLAG IS SET

Operation: If the overflow flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the overflow flag is clear (i.e., 0), control passes to the next sequential instruction.

```
if V = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JV cadd

Object Code Format: [11011101][disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.48. JVT — JUMP IF OVERFLOW TRAP IS SET

Operation: If the overflow flag is set (i.e., 1), the distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −128 to +127. If the overflow trap flag is clear (i.e., 0), control passes to the next sequential instruction.

```
if VT = 1 then
    PC ← PC + disp (sign-extended to 16 bits)
```

Assembly Language Format: JVT cadd

Object Code Format: [11011100][disp]

Bytes: 2
States: Jump Not Taken: 4
Jump Taken: 8

Flags Affected					
Z	N	C	V	VT	ST
–	–	–	–	0	–

3.13.49. LCALL — LONG CALL

Operation: The contents of the program counter (the return address) is pushed onto the stack. Then the distance from the end of this instruction to the target label is added to the program counter, effecting the call. The operand may be any address in the entire address space.

```
SP ← SP − 2
(SP) ← PC
PC ← PC + disp
```

Assembly Language Format: LCALL cadd

Object Code Format: [11101111][disp-low][disp-hi]

Bytes: 3
States: Onchip stack: 13
Onchip stack: 16

Flags Affected					
Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.50. LD — LOAD WORD

Operation: The value of the source (rightmost) word operand is stored into the destination (leftmost) operand.

(DEST) ← (SRC)

Assembly Language Format:

```
     DST     SRC
LD   wreg,   waop
```

Object Code Format: [101000aa][waop][wreg]

Bytes: 2+BEA
States: 4+CEA

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.51. LDB — LOAD BYTE

Operation: The value of the source (rightmost) byte operand is stored into the destination (leftmost) operand.

(DEST) ← (SRC)

Assembly Language Format:

```
     DST     SRC
LDB  breg,   baop
```

Object Code Format: [101100aa][baop][breg]

Bytes: 2+BEA
States: 4+CEA

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.52. LDBSE — LOAD INTEGER WITH SHORT-INTEGER

Operation: The value of the source (rightmost) byte operand is sign-extended and stored into the destination (leftmost) word operand.

```
(low byte DEST) ← (SRC)
if (SRC) < 80H then
   (high byte DEST) ← 0
else
   (high byte DEST) ← 0FFH
end_if
```

Assembly Language Format:

```
         DST     SRC
LDBSE    wreg,   baop
```

Object Code Format: [101111aa][baop][wreg]

Bytes: 2+BEA
States: 4+CEA

Flags Affected

Z	N	C	V	VT	ST
—	—	—	—	—	—

3.13.53. LDBZE — LOAD WORD WITH BYTE

Operation: The value of the source (rightmost) byte operand is zero-extended and stored into the destination (leftmost) word operand.

```
(low byte DEST) ← (SRC)
(high byte DEST) ← 0
```

Assembly Language Format:

```
         DST     SRC
LDBZE    wreg,   baop
```

Object Code Format: [101011aa][baop][wreg]

Bytes: 2+BEA
States: 4+CEA

Flags Affected

Z	N	C	V	VT	ST
—	—	—	—	—	—

3.13.54. LJMP — LONG JUMP

Operation: The distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The operand may be any address in the entire address space.

PC ← PC + disp

Assembly Language Format: LJMP cadd

Object Code Format: [11100111][disp-low][disp-hi]

Bytes: 3
States: 8

Flags Affected					
Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.55. MUL (Two Operands) — MULTIPLY INTEGERS

Operation: The two INTEGER operands are multiplied using signed arithmetic and the 32-bit result is stored into the destination (leftmost) LONG-INTEGER operand. The sticky bit flag is undefined after the instruction is executed.

(DEST) ← (DEST) * (SRC)

Assembly Language Format:

```
     DST    SRC
MUL  lreg,  waop
```

Object Code Format: [11111110][011011aa][waop][lreg]

Bytes: 3 + BEA
States: 29 + CEA

Flags Affected					
Z	N	C	V	VT	ST
–	–	–	–	–	√

3.13.56. MUL (Three Operands) — MULTIPLY INTEGERS

Operation: The second and third INTEGER operands are multiplied using signed arithmetic and the 32-bit result is stored into the destination (leftmost) INTEGER operand. The sticky bit flag is undefined after the instruction is executed.

(DEST) ← (SRC1) * (SRC2)

Assembly Language Format:

```
        DST     SRC1    SRC2
MUL     lreg,   wreg,   waop
```

Object Code Format: [11111110][010011aa][waop][wreg][lreg]

Bytes: 4+BEA
States: 30 + CEA

Flags Affected					
Z	N	C	V	VT	ST
–	–	–	–	–	√

3.13.57. MULB (Two Operands) — MULTIPLY SHORT-INTEGERS

Operation: The two SHORT-INTEGER operands are multiplied using signed arithmetic and the 16-bit result is stored into the destination (leftmost) INTEGER operand. The sticky bit flag is undefined after the instruction is executed.

(DEST) ← (DEST) * (SRC)

Assembly Language Format:

```
        DST     SRC
MULB    wreg,   baop
```

Object Code Format: [11111110][011111aa][baop][wreg]

Bytes: 3+BEA
States: 21 + CEA

Flags Affected					
Z	N	C	V	VT	ST
–	–	–	–	–	√

3.13.58. MULB (Three Operands) — MULTIPLY SHORT-INTEGERS

Operation: The second and third SHORT-INTEGER operands are multiplied using signed arithmetic and the 16-bit result is stored into the destination (leftmost) INTEGER operand. The sticky bit flag is undefined after the instruction is executed.

(DEST) ← (SRC1) * (SRC2)

Assembly Language Format:

```
      DST    SRC1   SRC2
MULB  wreg,  breg   baop
```

Object Code Format: [11111110][010111aa][baop][breg][wreg]

Bytes: 4+BEA
States: 22 + CEA

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	√

3.13.59. MULU (Two Operands) — MULTIPLY WORDS

Operation: The two WORD operands are multiplied using unsigned arithmetic and the 32-bit result is stored into the destination (leftmost) DOUBLE-WORD operand. The sticky bit flag is undefined after the instruction is executed.

(DEST) ← (DEST) * (SRC)

Assembly Language Format:

```
      DST    SRC
MULU  lreg,  waop
```

Object Code Format: [011011aa][waop][lreg]

Bytes: 2+BEA
States: 25 + CEA

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	√

3.13.60. MULU (Three Operands) — MULTIPLY WORDS

Operation: The second and third WORD operands are multiplied using unsigned arithmetic and the 32-bit result is stored into the destination (leftmost) DOUBLE-WORD operand. The sticky bit flag is undefined after the instruction is executed.

(DEST)< — (SRC1) * (SRC2)

Assembly Language Format:

```
        DST    SRC1   SRC2
MULU    lreg,  wreg,  waop
```

Object Code Format: [010011aa][waop][wreg][lreg]

Bytes: 3+BEA
States: 26 + CEA

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	√

3.13.61. MULUB (Two Operands) — MULTIPLY BYTES

Operation: The two BYTE operands are multiplied using unsigned arithmetic and the WORD result is stored into the destination (leftmost) operand. The sticky bit flag is undefined after the instruction is executed.

(DEST) ← (DEST) * (SRC)

Assembly Language Format:

```
         DST    SRC
MULUB    wreg,  baop
```

Object Code Format: [011111aa][baop][wreg]

Bytes: 2+BEA
States: 17 + CEA

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	√

3.13.62. MULUB (Three Operands) — MULTIPLY BYTES

Operation: The second and third BYTE operands are multiplied using unsigned arithmetic and the WORD result is stored into the destination (leftmost) operand. The sticky bit flag is undefined after the instruction is executed.

(DEST) ← (SRC1) * (SRC2)

Assembly Language Format:

```
       DST    SRC1   SRC2
MULUB  wreg,  breg,  baop
```

Object Code Format: [010111aa][baop][breg][wreg]

Bytes: 3+BEA
States: 18 + CEA

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	√

3.13.63. NEG — NEGATE INTEGER

Operation: The value of the INTEGER operand is negated.

(DEST) ← –(DEST)

Assembly Language Format: NEG wreg

Object Code Format: [00000011][wreg]

Bytes: 2
States: 4

Flags Affected

Z	N	C	V	VT	ST
√	√	√	√	↑	–

3.13.64. NEGB — NEGATE SHORT-INTEGER

Operation: The value of the SHORT-INTEGER operand is negated.

(DEST) ← −(DEST)

Assembly Language Format: NEGB breg

Object Code Format: [00010011][breg]

Bytes: 2
States: 4

Flags Affected

Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	–

3.13.65. NOP — NO OPERATION

Operation: Nothing is done. Control passes to the next sequential instruction.

Assembly Language Format: NOP

Object Code Format: [11111101]

Bytes: 1
States: 4

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.66. NORML — NORMALIZE LONG-INTEGER

Operation: The LONG-INTEGER operand is normalized; i.e., it is shifted to the left until its most significant bit is 1. If the most significant bit is still 0 after 31 shifts, the process stops and the zero flag is set. The number of shifts actually performed is stored in the second operand.

```
(COUNT) ← 0
do while (MSB(DEST) = 0) AND ((COUNT) < 31)
    (DEST) ← (DEST) * 2
    (COUNT) ← (COUNT) + 1
end_while
```

Assembly Language Format: NORML lreg,breg

Object Code Format: [00001111][breg][lreg]

Bytes: 3
States: 8 + No. of shifts performed

Flags Affected					
Z	N	C	V	VT	ST
✓	1	0	–	–	–

3.13.67. NOT — COMPLEMENT WORD

Operation: The value of the WORD operand is complemented: each 1 is replaced with a 0, and each 0 with a 1.

(DEST) ← NOT(DEST)

Assembly Language Format: NOT wreg

Object Code Format: [00000010][wreg]

Bytes: 2
States: 4

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	0	0	–	–

3.13.68. NOTB — COMPLEMENT BYTE

Operation: The value of the BYTE operand is complemented: each 1 is replaced with a 0, and each 0 with a 1.

(DEST) ← NOT (DEST)

Assembly Language Format: NOTB breg

Object Code Format: [00010010][breg]

Bytes: 2
States: 4

Flags Affected					
Z	N	C	V	VT	ST
√	√	0	0	–	–

3.13.69. OR — LOGICAL OR WORDS

Operation: The source (rightmost) WORD is ORed with the destination (leftmost) WORD operand. Each bit is set to 1 if the corresponding bit in either the source operand or the destination operand is 1. The result replaces the original destination operand.

(DEST) ← (DEST) OR (SRC)

Assembly Language Format:

```
    DST    SRC
OR  wreg,  waop
```

Object Code Format: [100000aa][waop][wreg]

Bytes: 2+BEA
States: 4+CEA

Flags Affected					
Z	N	C	V	VT	ST
√	√	0	0	–	–

3.13.70. ORB — LOGICAL OR BYTES

Operation: The source (rightmost) BYTE operand is ORed with the destination (leftmost) BYTE operand. Each bit is set to 1 if the corresponding bit in either the source operand or the destination operand was 1. The result replaces the original destination operand.

(DEST) ← (DEST) OR (SRC)

Assembly Language Format: ORB breg,baop

Object Code Format: [100100aa][baop][breg]

Bytes: 2+BEA
States: 4+CEA

Flags Affected

Z	N	C	V	VT	ST
√	√	0	0	–	–

3.13.71. POP — POP WORD

Operation: The word on top of the stack is popped and placed at the destination operand.

(DEST) ← (SP)
SP ← SP + 2

Assembly Language Format: POP waop

Object Code Format: [110011aa][waop]

Bytes: 1+BEA
States: Onchip Stack: 12+CEA
Offchip Stack 14+CEA

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.72. POPF — POP FLAGS

Operation: The word on top of the stack is popped and placed in the PSW. Interrupt calls cannot occur immediately following this instruction.

(PSW) ← (SP)
SP ← SP + 2

Assembly Language Format: POPF

Object Code Format: [11110011]

Bytes:		1
States:	Onchip Stack:	9
	Offchip Stack:	13

Flags Affected

Z	N	C	V	VT	ST
✓	✓	✓	✓	✓	✓

3.13.73. PUSH — PUSH WORD

Operation: The specified operand is pushed onto the stack.

SP ← SP − 2
(SP) ← (DEST)

Assembly Language Format: PUSH waop

Object Code Format: [110010aa][waop]

Bytes:		1+BEA
States:	Onchip Stack:	8+CEA
	Offchip Stack:	12+CEA

Flags Affected

Z	N	C	V	VT	ST
—	—	—	—	—	—

3.13.74. PUSHF — PUSH FLAGS

Operation: The PSW is pushed on top of the stack, and then set to all zeroes. This implies that all interrupts are disabled. Interrupt-calls cannot occur immediately following this instruction.

SP ← SP − 2
(SP) ← PSW
PSW ← 0

Assembly Language Format: PUSHF

Object Code Format: [11110010]

Bytes: 1
States: Onchip Stack: 8
Offchip Stack: 12

Flags Affected					
Z	N	C	V	VT	ST
0	0	0	0	0	0

3.13.75. RET — RETURN FROM SUBROUTINE

Operation: The PC is popped off the top of the stack.

PC ← (SP)
SP ← SP + 2

Assembly Language Format: RET

Object Code Format: [11110000]

Bytes: 1
States: Onchip Stack: 12
Offchip Stack: 16

Flags Affected					
Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.76. RST — RESET SYSTEM

Operation: The PSW is initialized to zero, and the PC is initialized to 2080H. The I/O registers are set to their initial value (see section 2.15.2, "Reset Status"). Executing this instruction will cause a pulse to appear on the reset pin of the 8096.

PSW ← 0
PC ← 2080H

Assembly Language Format: RST

Object Code Format: [11111111]

Bytes: 1
States: 16

Flags Affected					
Z	N	C	V	VT	ST
0	0	0	0	0	0

3.13.77. SCALL — SHORT CALL

Operation: The contents of the program counter (the return address) is pushed onto the stack. Then the distance from the end of this instruction to the target label is added to the program counter, effecting the call. The offset from the end of this instruction to the target label must be in the range of −1024 to +1023 inclusive.

SP ← SP — 2
(SP) ← PC
PC ← PC + disp (sign-extended to 16 bits)

Assembly Language Format: SCALL cadd

Object Code Format: [00101xxx][disp-low]

where xxx holds the three high-order bits of displacement.

Bytes: 2
States: Onchip Stack: 13
Offchip Stack: 16

Flags Affected					
Z	N	C	V	VT	ST
—	—	—	—	—	—

3.13.78. SETC — SET CARRY FLAG

Operation: The carry flag is set.

C ← 1

Assembly Language Format: SETC

Object Code Format: [11111001]

Bytes: 1
States: 4

Flags Affected					
Z	N	C	V	VT	ST
–	–	1	–	–	–

3.13.79. SHL — SHIFT WORD LEFT

Operation: The destination (leftmost) word operand is shifted left as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 inclusive, or as the content of any register, the address of which is 16 to 255. The right bits of the result are filled with zeroes. The last bit shifted out is saved in the carry flag.

```
Temp ← (COUNT)
do while Temp <>0
    C ← High order bit of (DEST)
    (DEST) ← (DEST)*2
    Temp ← Temp — 1
end_while
```

Assembly Language Format:

```
    SHL  wreg,#count
or
    SHL  wreg,breg
```

Object Code Format: [00001001][cnt/breg][wreg]

Bytes: 3
States: 7 + No. of shifts performed
note: 0 place shifts take 8 states.

Flags Affected					
Z	N	C	V	VT	ST
√	√	√	√	↑	–

3.13.80. SHLB — SHIFT BYTE LEFT

Operation: The destination (leftmost) byte operand is shifted left as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 inclusive, or as the content of any register, the address of which is 16 to 255. The right bits of the result are filled with zeroes. The last bit shifted out is saved in the carry flag.

```
Temp ← (COUNT)
do while Temp <> 0
      C ← High order bit of (DEST)
      (DEST) ← (DEST)*2
      Temp ← Temp − 1
end_while
```

Assembly Language Format:

```
     SHLB   breg,#count
or
     SHLB   breg,breg
```

Object Code Format: [00011001][cnt/breg][breg]

Bytes: 3
States: 7 + No. of shifts performed
note: 0 place shifts take 8 states.

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	–

3.13.81. SHLL — SHIFT DOUBLE-WORD LEFT

Operation: The destination (leftmost) double-word operand is shifted left as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 inclusive, or as the content of any register, the address of which is 16 to 255. The right bits of the result are filled with zeroes. The last bit shifted out is saved in the carry flag.

```
Temp ← (COUNT)
do while Temp <> 0
    C ← High order bit of (DEST)
    (DEST) ← (DEST)*2
    Temp ← Temp − 1
end_while
```

Assembly Language Format:

```
    SHLL   lreg,#count
or
    SHLL   lreg,breg
```

Object Code Format: [00001101][cnt/breg][lreg]

Bytes: 3
States: 7 + No. of shifts performed
note: 0 place shifts take 8 states.

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	–

3.13.82. SHR — LOGICAL RIGHT SHIFT WORD

Operation: The destination (leftmost) word operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 inclusive, or as the content of any register, the address of which is 16 to 255. The left bits of the result are filled with zeroes. The last bit shifted out is saved to the carry. The sticky bit flag is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

```
Temp ← (COUNT)
do while Temp <> 0
    C ← Low order bit of (DEST)
    (DEST) ← (DEST) / 2 where / is unsigned division
    Temp ← Temp − 1
end_while
```

Assembly Language Format:

```
    SHR   wreg,#count
or
    SHR   wreg,breg
```

Object Code Format: [00001000][cnt/breg][wreg]

Bytes: 3
States: 7 + No. of shifts performed
note: 0 place shifts take 8 states.

Flags Affected					
Z	N	C	V	VT	ST
√	√	√	0	–	√

3.13.83. SHRA — ARITHMETIC RIGHT SHIFT WORD

Operation: The destination (leftmost) word operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 inclusive, or as the content of any register, the address of which is 16 to 255. If the original high order bit value was 0, zeroes are shifted in. If the value was 1, ones are shifted in. The last bit shifted out is saved in the carry. The sticky bit flag is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

```
Temp ← (COUNT)
do while Temp <> 0
    C ← Low order bit of (DEST)
    (DEST) ← (DEST) / 2 where / is signed division
    Temp ← Temp − 1
end_while
```

Assembly Language Format:

```
    SHRA   wreg,#count
or
    SHRA   wreg,breg
```

Object Code Format: [00001010][cnt/breg][wreg]

Bytes: 3
States: 7 + No. of shifts performed
note: 0 place shifts take 8 states.

Flags Affected

Z	N	C	V	VT	ST
√	√	√	0	–	√

3.13.84. SHRAB — ARITHMETIC RIGHT SHIFT BYTE

Operation: The destination (leftmost) byte operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 inclusive, or as the content of any register, the address of which is 16 to 255. If the original high order bit value was 0, zeroes are shifted in. If that value was 1, ones are shifted in. The last bit shifted out is saved in the carry. The sticky bit flag is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

```
Temp ← (COUNT)
do while Temp <> 0
    C, = Low order bit of (DEST)
    (DEST) ← (DEST) / 2 where /is signed division
    Temp ← Temp − 1
end_while
```

Assembly Language Format:

```
    SHRAB  breg,#count
or
    SHRAB  breg,breg
```

Object Code Format: [00011010][cnt/breg][breg]

Bytes: 3
States: 7 + No. of shifts performed
note: 0 place shifts take 8 states.

Flags Affected

Z	N	C	V	VT	ST
✓	✓	✓	0	–	✓

3.13.85. SHRAL — ARITHMETIC RIGHT SHIFT DOUBLE-WORD

Operation: The destination (leftmost) double-word operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 inclusive, or as the content of any register, the address of which is 16 to 255. If the original high order bit value was 0, zeroes are shifted in. If the value was 1, ones are shifted in. The sticky bit is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

```
Temp ← (COUNT)
do while Temp < > 0
    C ← Low order bit of (DEST)
    (DEST) ← (DEST) / 2 where/is signed division
    Temp ← Temp − 1
end_while
```

Assembly Language Format:

```
    SHRAL   lreg,#count
or
    SHRAL   lreg,breg
```

Object Code Format: [00001110][cnt/breg][lreg]

Bytes: 3
States: 7 + No. of shifts performed
note: 0 place shifts take 8 states.

Flags Affected

Z	N	C	V	VT	ST
√	√	√	0	–	√

3.13.86. SHRB — LOGICAL RIGHT SHIFT BYTE

Operation: The destination (leftmost) byte operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 inclusive, or as the content of any register, the address of which is 16 to 255. The left bits of the result are filled with zeroes. The last bit shifted out is saved in the carry. The sticky bit flag is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

```
Temp ← (COUNT)
do while Temp < > 0
    C ← Low order bit of (DEST)
    (DEST) ← (DEST) / 2 where/is unsigned division
    Temp ← Temp − 1
end_while
```

Assembly Language Format:

```
    SHRB   breg,#count
or
    SHRB   breg,breg
```

Object Code Format: [00011000][cnt/breg][breg]

Bytes: 3
States: 7 + No. of shifts performed
note: 0 place shifts take 8 states.

Flags Affected

Z	N	C	V	VT	ST
√	√	√	0	–	√

3.13.87. SHRL — LOGICAL RIGHT SHIFT DOUBLE-WORD

Operation: The destination (leftmost) double-word operand is shifted right as many times as specified by the count (rightmost) operand. The count may be specified either as an immediate value in the range of 0 to 15 inclusive, or as the content of any register, the address of which is 16 to 255. The left bits of the result are filled with zeroes. The last bit shifted out is saved in the carry. The sticky bit flag is cleared at the beginning of the instruction, and set if at any time during the shift a 1 is shifted first into the carry flag, and a further shift cycle occurs.

```
Temp ← (COUNT)
do while Temp < > 0
    C ← Low order bit of (DEST)
    (DEST) ← (DEST) / 2 where/is unsigned division
    Temp ← Temp − 1
end_while
```

Assembly Language Format:

```
    SHRL   lreg,#count
or
    SHRL   lreg,breg
```

Object Code Format: [00001100][cnt/breg][breg]

Bytes: 3
States: 7 + No. of shifts performed
note: 0 place shifts take 8 states.

Flags Affected

Z	N	C	V	VT	ST
√	√	√	0	–	√

3.13.88. SJMP — SHORT JUMP

Operation: The distance from the end of this instruction to the target label is added to the program counter, effecting the jump. The offset from the end of this instruction to the target label must be in the range of −1024 to +1023 inclusive.

PC ← PC + disp (sign-extended to 16 bits)

Assembly Language Format: SJMP cadd

Object Code Format: [00100xxx][disp-low]

where xxx holds the three high order bits of the displacement.

Bytes: 2
States: 8

Flags Affected

Z	N	C	V	VT	ST
—	—	—	—	—	—

3.13.89. SKIP — TWO BYTE NO-OPERATION

Operation: Nothing is done. This is actually a two-byte NOP where the second byte can be any value, and is simply ignored. Control passes to the next sequential instruction.

Assembly Language Format: SKIP breg

Object Code Format: [00000000][breg]

Bytes: 2
States: 4

Flags Affected

Z	N	C	V	VT	ST
—	—	—	—	—	—

3.13.90. ST — STORE WORD

Operation: The value of the leftmost word operand is stored into the rightmost operand.

(DEST) ← (SRC)

Assembly Language Format:

```
     SRC     DST
ST   wreg,   waop
```

Object Code Format: [110000aa][waop][wreg]

Bytes: 2+BEA
States: 4+CEA

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.91. STB — STORE BYTE

Operation: The value of the leftmost byte operand is stored into the rightmost operand.

(DEST) ← (SRC)

Assembly Language Format:

```
      SRC     DST
STB   breg,   baop
```

Object Code Format: [110001aa][baop][breg]

Bytes: 2+BEA
States: 4+CEA

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.92. SUB (Two Operands) — SUBTRACT WORDS

Operation: The source (rightmost) word operand is subtracted from the destination (leftmost) word operand, and the result is stored in the destination. The carry flag is set as complement of borrow.

(DEST) ← (DEST) — (SRC)

Assembly Language Format:

```
      DST     SRC
SUB   wreg,   waop
```

Object Code Format: [011010aa][waop][wreg]

Bytes: 2+BEA
States: 4+CEA

Flags Affected

Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	–

3.13.93. SUB (Three Operands) — SUBTRACT WORDS

Operation: The source (rightmost) word operand is subtracted from the second word operand, and the result is stored in the destination (the leftmost operand). The carry flag is set as complement of borrow.

(DEST) ← (SRC1) — (SRC2)

Assembly Language Format:

```
      DST     SRC1    SRC2,
SUB   wreg,   wreg,   waop
```

Object Code Format: [010010aa][waop][Swreg][Dwreg]

Bytes: 3+BEA
States: 5+CEA

Flags Affected

Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	–

3.13.94. SUBB (Two Operands) — SUBTRACT BYTES

Operation: The source (rightmost) byte is subtracted from the destination (leftmost) byte operand, and the result is stored in the destination. The carry flag is set as complement of borrow.

(DEST) ← (DEST) — (SRC)

Assembly Language Format:

```
       DST     SRC
SUBB   breg,   baop
```

Object Code Format: [011110aa][baop][breg]

Bytes: 2+BEA
States: 4+CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	–

3.13.95. SUBB (Three Operands) — SUBTRACT BYTES

Operation: The source (rightmost) byte operand is subtracted from the destination (leftmost) byte operand, and the result is stored in the destination. The carry flag is set as complement of borrow.

(DEST) ← (DEST) — (SRC)

Assembly Language Format:

```
       DST     SRC
SUBB   breg,   baop
```

Object Code Format: [010110aa][baop][Sbreg][Dbreg]

Bytes: 3+BEA
States: 5+CEA

Flags Affected					
Z	N	C	V	VT	ST
✓	✓	✓	✓	↑	–

3.13.96. SUBC — SUBTRACT WORDS WITH BORROW

Operation: The source (rightmost) word operand is subtracted from the destination (leftmost) word operand. If the carry flag was clear, 1 is subtracted from the above result. The result replaces the original destination operand. The carry flag is set as complement of borrow.

(DEST) ← (DEST) — (SRC) — (1-C)

Assembly Language Format:

```
        DST     SRC
SUBC    wreg,   waop
```

Object Code Format: [101010aa][waop][wreg]

Bytes: 2+BEA
States: 4+CEA

Flags Affected					
Z	N	C	V	VT	ST
↓	✓	✓	✓	↑	–

3.13.97. SUBCB — SUBTRACT BYTES WITH BORROW

Operation: The source (rightmost) byte operand is subtracted from the destination (leftmost) byte operand. If the carry flag was clear, 1 is subtracted from the above result. The result replaces the original destination operand. The carry flag is set as complement of borrow.

(DEST) ← (DEST) — (SRC) — (1-C)

Assembly Language Format:

```
         DST     SRC
SUBCB    breg,   baop
```

Object Code Format: [101110aa][baop][breg]

Bytes: 2+BEA
States: 4+CEA

Flags Affected					
Z	N	C	V	VT	ST
↓	✓	✓	✓	↑	–

3.13.98. TRAP — SOFTWARE TRAP

Operation: This instruction causes an interrupt-call which is vectored through location 2010H. The operation of this instruction is not effected by the state of the interrupt enable flag in the PSW (I). Interrupt-calls cannot occur immediately following this instruction. This instruction is intended for use by Intel provided development tools. These tools will not support user-application of this instruction.

SP ← SP − 2
(SP) ← PC
PC ← (2010H)

Assembly Language Format: This instruction is not supported by revision 1.0 of the 8096 assembly language.

Object Code Format: [11110111]

Bytes: 1
States: Onchip Stack: 21
Offchip Stack: 24

Flags Affected

Z	N	C	V	VT	ST
–	–	–	–	–	–

3.13.99. XOR — LOGICAL EXCLUSIVE-OR WORDS

Operation: The source (rightmost) word operand is XORed with the destination (leftmost) word operand. Each bit is set to 1 if the corresponding bit in either the source operand or the destination operand was 1, but not both. The result replaces the original destination operand.

(DEST) ← (DEST) XOR (SRC)

Assembly Language Format:

```
       DST     SRC
XOR    wreg,   waop
```

Object Code Format: [100001aa][waop][wreg]

Bytes: 2+BEA
States: 4+CEA

Flags Affected

Z	N	C	V	VT	ST
√	√	0	0	–	–

3.13.100. XORB — LOGICAL EXCLUSIVE-OR BYTES

Operation: The source (rightmost) byte operand is XORed with the destination (leftmost) byte operand. Each bit is set to 1 if the corresponding bit in either the source operand or the destination operand was 1, but not both. The result replaces the original destination operand.

(DEST) ← (DEST) XOR (SRC)

Assembly Language Format:

```
      DST     SRC
XORB  breg,   baop
```

Object Code Format: [100101aa][baop][breg]

Bytes: 2+BEA
States: 4+CEA

Flags Affected					
Z	N	C	V	VT	ST
√	√	0	0	–	–

Hardware Design Information 4

CHAPTER 4
HARDWARE DESIGN INFORMATION

4.0. HARDWARE INTERFACING OVERVIEW

This section of the manual is devoted to the hardware engineer. All of the information you need to connect the correct pin to the correct external circuit is provided. Many of the special function pins have different characteristics which are under software control, therefore, it is necessary to define the system completely before the hardware is wired-up.

Frequently within this section a specification for a current, voltage, or time period is referred to; the values provided are to be used as an approximation only. The exact specification can be found in the latest data sheet for the particular part and temperature range that is being used.

4.1. REQUIRED HARDWARE CONNECTIONS

Although the 8096 is a single-chip microcontroller, it still requires several external connections to make it work. Power must be applied, a clock source provided, and some form of reset circuitry must be present. We will look at each of these areas of circuitry separately. Figure 4-5 shows the connections that are needed for a single-chip system.

4.1.1. Power Supply Information

Power for 8096 flows through 6 pins; one VCC pin, two VSS pins, one VREF (analog VCC), one ANGND (Analog VSS), and one VPD (V Power Down) pin. All six of these pins must be connected to the 8096 for normal operation. The VCC pin, VREF pin and VPD pin should be tied to 5 volts. When the analog to digital converter is being used it may be desirable to connect the VREF pin to a separate power supply, or at least a separate power supply line.

The two VSS pins should be connected together with as short a lead as possible to avoid problems due to voltage drops across the wiring. There should be no measurable voltage difference between VSS1 and VSS2. The 2 VSS pins and the ANGND pin should all be nominally at 0 volts. The maximum current drain of the 8096 is around 200mA, with all lines unloaded.

When the analog converter is being used, clean, stable power must be provided to the analog section of the chip to assure highest accuracy. To achieve this, it may be desirable to separate the analog power supply from the digital power supply. The VREF pin supplies 5 volts to the analog circuitry and the ANGND pin is the ground for this section of the chip. More information on the analog power supply is in section 4.3.1.

4.1.2. Other Needed Connections

Several of the pins on the 8096 are used to configure the mode of operation. In normal operation the following pins should be tied directly to the indicated power supply.

PIN	POWER SUPPLY
NMI	VCC
$\overline{\text{TEST}}$	VCC
$\overline{\text{EA}}$	VCC (to allow internal execution) VSS (to force external execution)

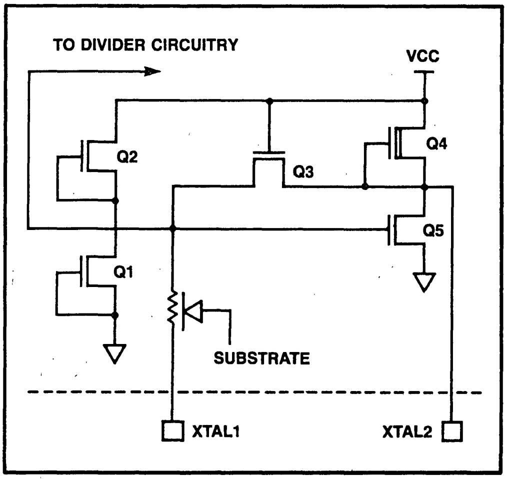


Figure 4-1. 8096 Oscillator Circuit

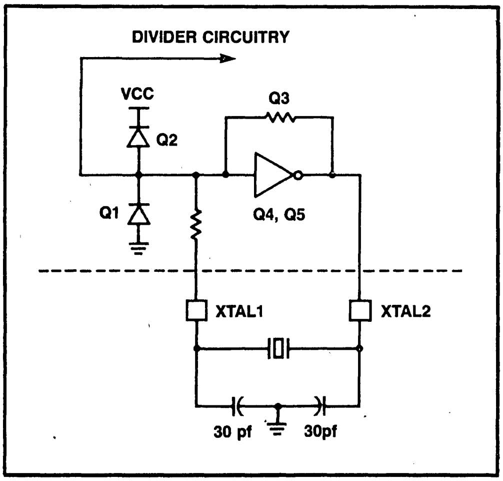


Figure 4-2. Crystal Oscillator Circuit

Although the $\overline{EA}$ pin has an internal pulldown, it is best to tie this pin to the desired level if it is not left completely disconnected. This will prevent induced noise from disturbing the system.

4.1.3. Oscillator Information

The 8096 requires a clock source to operate. This clock can be provided to the chip through the XTAL1 input or the on-chip oscillator can be used. The frequency of operation is from 6.0 MHz to 12 MHz.

The on-chip circuitry for the 8096 oscillator is a single stage linear inverter as shown in Figure 4-1. It is intended for use as a crystal-controlled, positive reactance oscillator with external connections as shown in Figure 4-2. In this application, the crystal is being operated in its fundamental response mode as an inductive reactance in parallel resonance with capacitance external to the crystal.

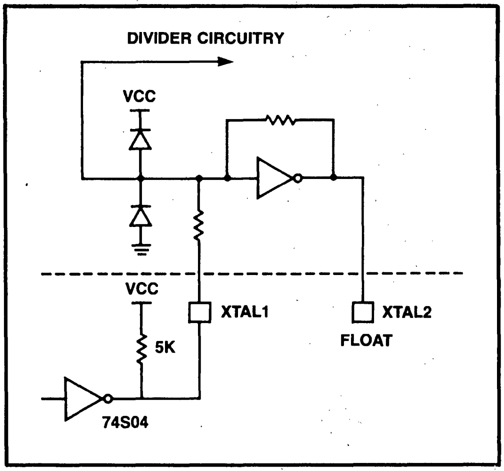


Figure 4-3. External Clock Drive

The crystal specifications and capacitance values (C1 and C2 in Figure 4-2) are not critical. 30 pF can be used in these positions at any frequency with good quality crystals. For 0.5% frequency accuracy, the crystal frequency can be specified at series resonance or for parallel resonance with any load capacitance. (In other words, for that degree of frequency accuracy, the load capacitance simply doesn't matter.) For 0.05% frequency accuracy the crystal frequency should be specified for parallel resonance with 25 pF load capacitance, if C1 and C2 are 30 pF.

A more in-depth discussion of crystal specifications and the selection of values for C1 and C2 can be found in the Intel Application Note, AP-155, "Oscillators for Microcontrollers."

To drive the 8096 with an external clock source, apply the external clock signal to XTAL1 and let XTAL2 float. An example of this circuit is shown in Figure 4-3. The required voltage levels on XTAL1 are specified in the data sheet. The signal on XTAL1 must be clean with good solid levels. It is important that the minimum high and low times are met.

There is no specification on rise and fall times, but they should be reasonably fast (on the order of 30 nanoseconds) to avoid having the XTAL1 pin in the transition range for long periods of time. The longer the signal is in the transition region, the higher the probability that an external noise glitch could be seen by the clock generator circuitry. Noise glitches on the 8096 internal clock lines will cause unreliable operation.

The clock generator provides a 3 phase clock output from the XTAL1 pin input. Figure 4-4 shows the waveforms of the major internal timing signals.

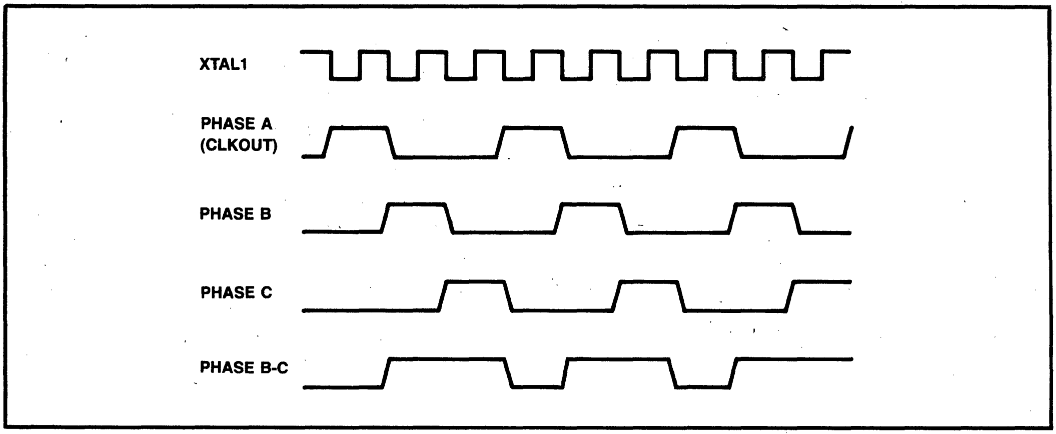


Figure 4-4. Internal Timings

4.1.4. Reset Information

In order for the 8096 to function properly it must be reset. This is done by holding the reset pin low for at least 2 state times after the power supply is within tolerance, the oscillator has stabilized, and the back-bias generator has stabilized. Typically, the back-bias generator requires one millisecond to stabilize.

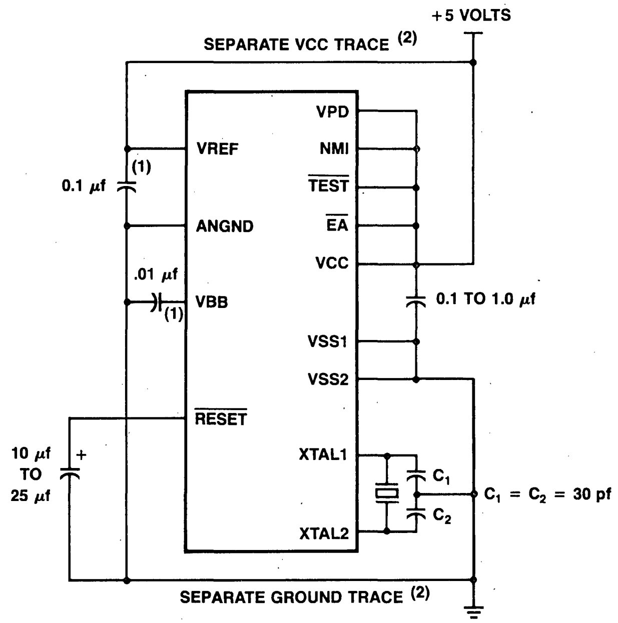
There are several ways of doing this, the simplest being just to connect a capacitor from the reset pin to ground. The capacitor should be on the order of 1 to 2 microfarads for every millisecond of reset time required. This method will only work if the rise time of VCC is fast and the total reset time is less than around 50 milliseconds. It also may not work if the reset pin is to be used to reset other parts on the board. An 8096 with the minimum required connections is shown in Figure 4-5.

The 8096 $\overline{\text{RESET}}$ pin can be used to allow other chips on the board to make use of the watchdog timer or the RST instruction. When this is done the reset hardware should be a one-shot with an open-collector output. The reset pulse going to the other parts may have to be buffered and lengthened with a one-shot, since the $\overline{\text{RESET}}$ low duration is only two state times. If this is done, it is possible that the 8096 will be reset and start running before the other parts on the board are out of reset. The software must account for this possible problem.

A capacitor directly connected to $\overline{\text{RESET}}$ cannot be used to reset the part if the pin is to be used as an output. If a large capacitor is used, the pin will pull down more slowly than normal. It will continue to pull down until the 8096 is reset. It could fall so slowly that it never goes below the internal switch point of the reset signal (1 to 1.5 volts), a voltage which may be above the guaranteed switch point of external circuitry connected to the pin. Several circuit examples are shown in Figure 4-6.

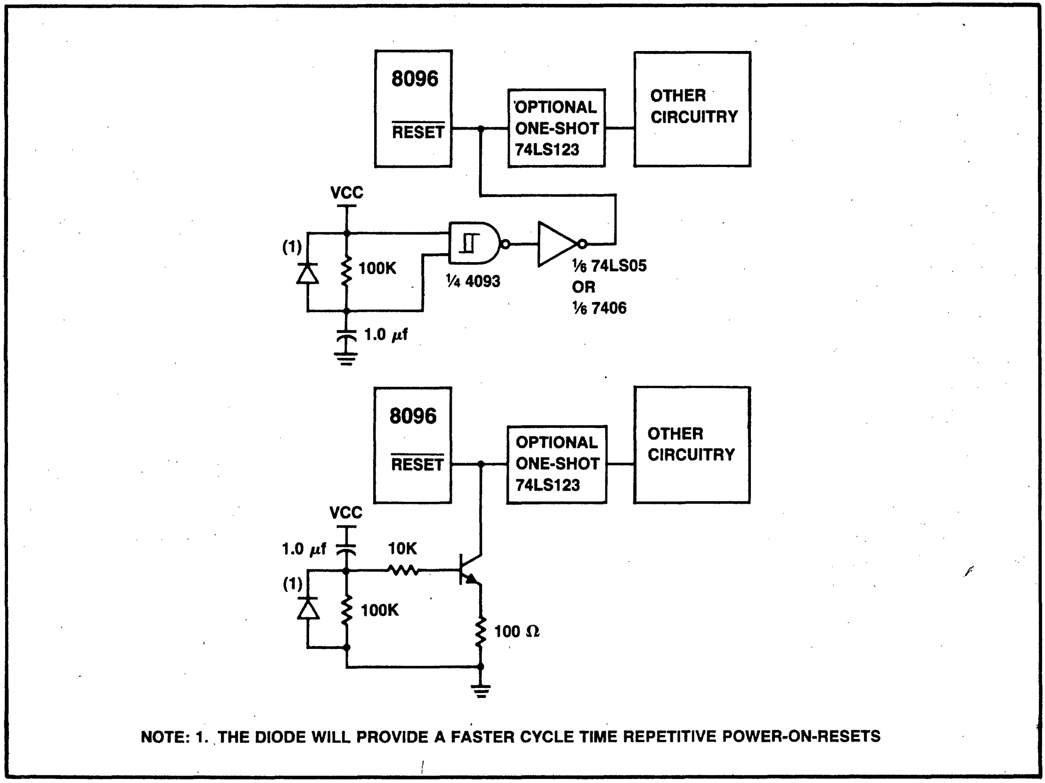
4.1.5. Sync Mode

If $\overline{\text{RESET}}$ is brought high coincident to the falling edge of XTAL1, the part will start executing the 10 state time RST instruction exactly 6 XTAL1 cycles later. This feature can be used to synchronize several MCS-96 devices. A diagram of a typical connection is shown in Figure 4-7. It should be noted that parts that start in sync may not stay that way, due to propagation delays which may cause the synchronized parts to receive signals at slightly different times.



NOTES: 1. THESE CAPACITORS ARE NEEDED ONLY IF A TO D IS USED.
2. VREF & ANGND MAY BE CONNECTED TO THE SAME TRACES AS THE DIGITAL POWER SUPPLY IF THE A TO D IS NOT USED.

Figure 4-5. Minimum Hardware Connections



NOTE: 1. THE DIODE WILL PROVIDE A FASTER CYCLE TIME REPETITIVE POWER-ON-RESETS

Figure 4-6. Multiple Chip Reset Circuits

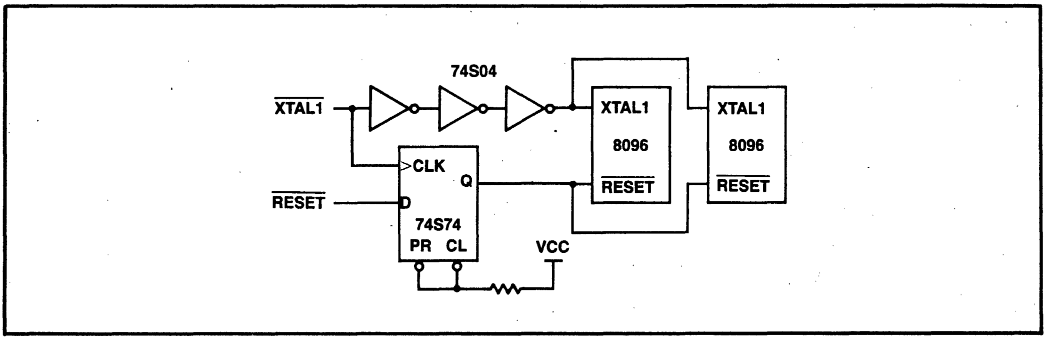


Figure 4-7. Reset Sync Mode

4.1.6. Disabling the Watchdog Timer

The watchdog timer will pull the RESET pin low when it overflows. If the pin is being externally held above the low going threshold, the pull-down transistor will remain on indefinitely. This means that once the watchdog overflows, the part must be reset or RESET must be held high indefinitely. Just resetting the watchdog timer will not reset the flip-flop which keeps the RESET pull-down on.

The pull-down is capable of sinking on the order of 30 milliamps if it is held at 2.0 volts. This amount of current may cause some long term reliability problems due to localized chip heating. For this reason, parts that will be used in production should never have had the watchdog timer over-ridden for more than a second or two.

Whenever the reset pin is being pulled high while the pull-down is on, it should be through a resistor that will limit the voltage on $\overline{\text{RESET}}$ to 2.5 volts and the current through the pin to 40 milliamps. Figure 4-8 shows a circuit which will provide the desired results. Using the LED will provide the additional benefit of having a visual indicator that the part is trying to reset itself, although this circuit only works at room temperature and VCC = 5 Volts.

4.1.7. Power Down Circuitry

Battery backup can be provided on the 8096 with a 1 mA current drain at 5 volts. This mode will hold locations 0F0H through 0FFH valid as long as the power to the VPD pin remains on. The required timings to put the part into power-down and an overview of this mode are given in section 2.4.2.

A 'key' can be written into power-down RAM while the part is running. This key can be checked on reset to determine if it is a start-up from power-down or a complete cold start. In this way the validity of the power-down RAM can be verified. The length of this key determines the probability that this procedure will work, however, there is always a statistical chance that the RAM will power up with a replica of the key.

Under most circumstances, the power-fail indicator which is used to initiate a power-down condition must come from the unfiltered, unregulated section of the power supply. The power supply must have sufficient storage capacity to operate the 8096 until it has completed its reset operation.

4.2. DRIVE AND INTERFACE LEVELS

There are 5 types of I/O lines on the 8096. Of these, 2 are inputs and 3 are outputs. All of the pins of the same type have the same current/voltage characteristics. Some of the control input pins, such as XTAL1 and $\overline{\text{RESET}}$, may have slightly different characteristics. These pins are discussed in section 4.1.

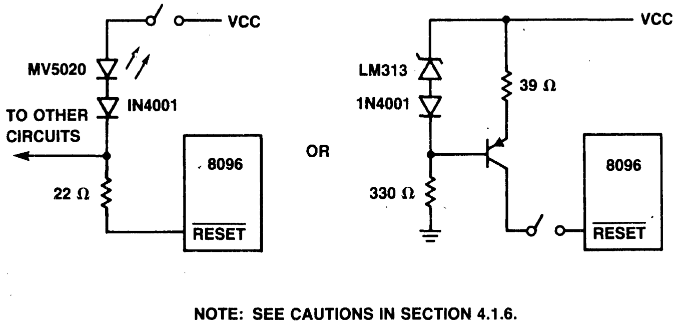
While discussing the characteristics of the I/O pins some approximate current or voltage specifications will be given. The exact specifications are available in the latest version of the 8096 Data Sheet.

4.2.1. Quasi-Bidirectional Ports

The quasi-bidirectional port is both an input and an output port. It has three states, low impedance current sink, low impedance current source, and high impedance current source. As a low impedance current sink, the pin has a specification of sinking up to around .4 milliamps, while staying below 0.45 volts. The pin is placed in this condition by writing a '0' to the SFR (Special Function Register) controlling the pin.

When a '1' is written to the SFR location controlling the pin, a low impedance current source is turned on for one state time, then it is turned off and the depletion pull-up holds the line at a logical '1' state. The low-impedance pull-up is used to shorten the rise time of the pin, and has current source capability on the order of 100 times that of the depletion pull-up. The configuration of a quasi-bidirectional port pin is shown in Figure 4-9.

While the depletion mode pull-up is the only device on, the pin may be used as an input with a leakage of around 100 microamps from 0.45 volts to VCC. It is ideal for use with TTL or CMOS chips and may even be used



NOTE: SEE CAUTIONS IN SECTION 4.1.6.

Figure 4-8. Disabling the WDT

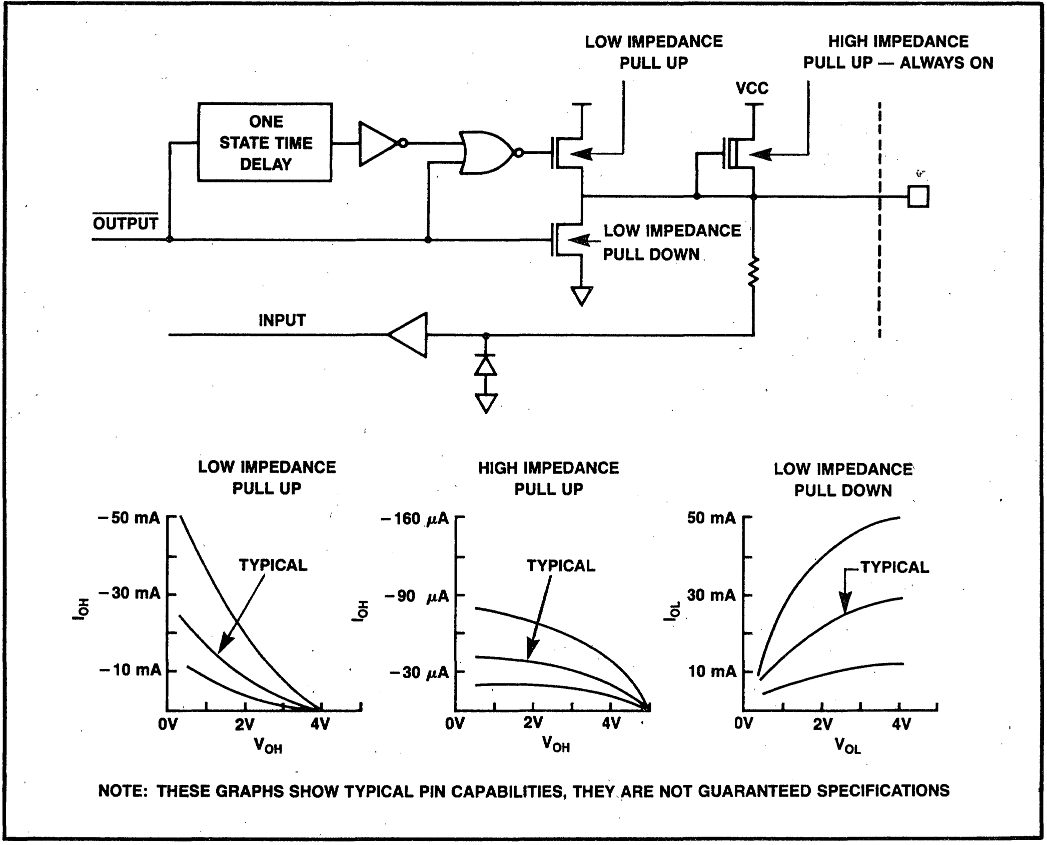


Figure 4-9. Quasi-Bidirectional Port

directly with switches, however if the switch option is used certain precautions should be taken. It is impootant to note that any time the pin is read, the value returned will be the value on the pin, not the value placed in the control register. This could prevent logical operations on these pins while they are being used as inputs.

4.2.2. Quasi-Bidirectional Hardware Connections

When using the quasi-bidirectional ports as inputs tied to switches, series resistors should be used if the ports will be written to internally after the part is initialized. Every time any quasi-bidirectional pin is written from a zero to a one, the low impedance pull-up is turned on. If many of the pins are tied directly to ground, a large current spike will be generated when all of these low impedance devices are turned on at once.

For this reason, a series resistor is recommended to limit the current to a maximum of 0.2 milliamps per pin. If several pins are connected to a common ground through switches, it should be sufficient to limit the current through the common ground to 0.2 milliamps times the maximum number of pins that could be switched to ground. Many switches require a minimum amount of current flow through them to keep them clean. This could cause problems in long term reliability if it is not considered when designing a system.

If a switch is used on a long line connected to a quasi-bidirectional pin, a pull-up resistor is recommended to reduce the possibility of noise glitches. On extremely long lines that are handling slow signals a capacitor may be helpful in addition to the resistor.

4.2.3. Input Ports, Analog and Digital

The high impedance input ports on the 8096 have an input leakage of a few microamps and are predominantly capacitive loads on the order of 10 pf. The Port 0 pins have an additional function when the A to D converter is being used. These pins are the input to the A to D converter, and as such, are required to provide current to the comparator when a conversion is in process. This means that the input characteristics of a pin will change if a conversion is being done on that pin. See section 4.3.1.

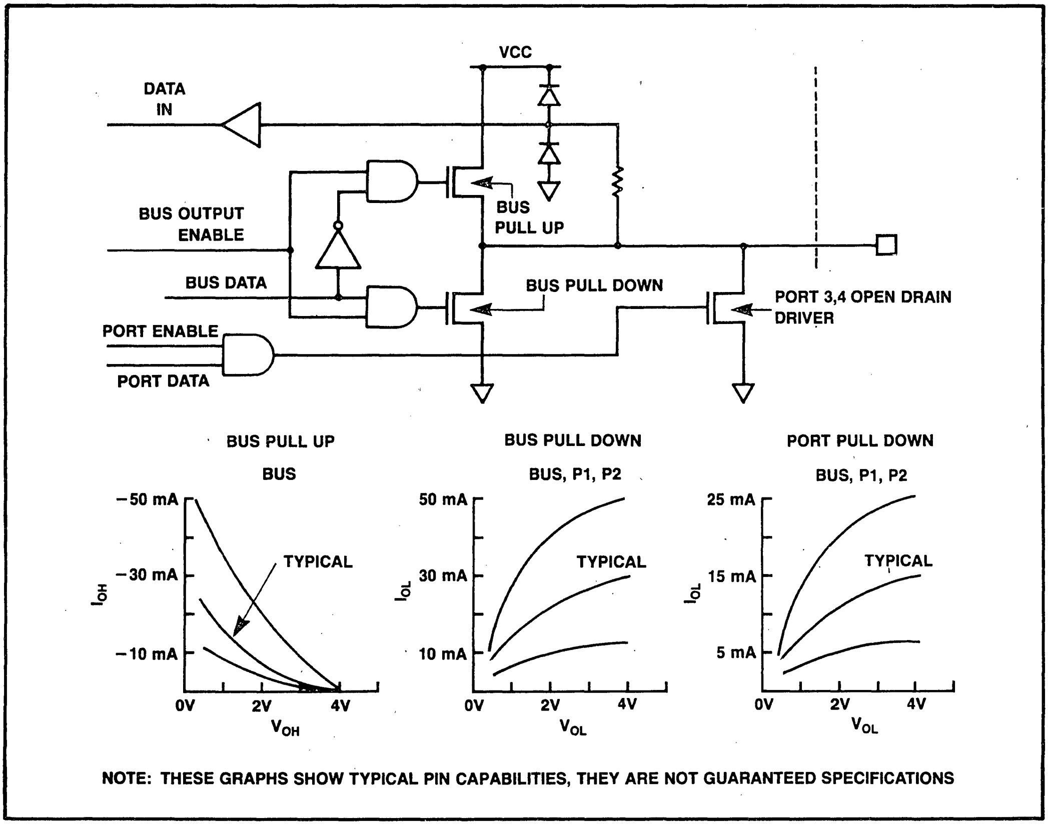
4.2.4. Open Drain Ports

Ports 3 and 4 on the 8096 are open drain ports. There is no pull-up when these pins are used as I/O ports. These pins have different characteristics when used as bus pins as described in the next section. A diagram of the output buffers connected to ports 3 and 4 and the Bus pins is shown in Figure 4-10.

When Ports 3 and 4 are to be used as inputs, or as Bus pins, they must first be written with a '1', this will put the ports in a high impedance mode. When they are used as outputs, a pull-up resistor must be used externally. The sink capability of these pins is on the order of 0.4 milliamps so the total pull-up current to the pin must be less than this. A 15k pull-up resistor will source a maximum of 0.33 milliamps, so it would be a reasonable value to choose if no other circuits with pullups were connected to the pin.

4.2.5. HSO Pins, Control Outputs and Bus Pins

The control outputs and HSO pins have output buffers with the same output characteristics as those of the bus pins. Included in the category of control outputs are; TXD, RXD (in mode 0), PWM, CLKOUT, ALE, $\overline{BHE}$, $\overline{RD}$, and $\overline{WR}$. The bus pins have 3 states; output high, output low, and high impedance input. As a high output, the pins are specified to source around 200 µA to 2.4 volts, but the pins can source on the order of ten times that value in order to provide fast rise times. When used as a low output, the pins can sink around 2 mA at .45 volts, and considerably more as the voltage increases. When in the high impedance state, the pin acts as a capacitive load with a few microamps of leakage. Figure 4-10 shows the internal configuration of a bus pin.



NOTE: THESE GRAPHS SHOW TYPICAL PIN CAPABILITIES, THEY ARE NOT GUARANTEED SPECIFICATIONS

Figure 4-10. Bus and Port 3 and 4 Pins

4.3. ANALOG INTERFACE

Interfacing the 8096 to analog signal can be done in several ways. If the 8096 needs to measure an analog signal the A to D converter can be used. Creation of analog outputs can be done with either the PWM output or the HSO unit.

4.3.1. Analog Inputs

The 8096 can have 8 analog inputs and can convert one input at a time into a digital value. Each conversion takes 42 microseconds with a 12 MHz signal on XTAL1. The input signal is applied to one of the Port 0/Analog Channel inputs. Since there is no sample and hold on the A to D, the input signal must remain constant over the sampling period.

Whan a conversion takes place, the 8096 compares the external signal to that of its internal D to A. Based on the result of the comparison it adjusts the D to A and compares again. Each comparison takes 8 state times and requires the input to the comparator to be charged up. 20 comparisons are made during a conversion, two times for each bit of resolution. An additional 8 states are used to load and store values. The total number of state times required is 168 for a 10-bit conversion. Attempting to do other than a 10-bit conversion is not recommended.

Since the capacitance of the comparator input is around 0.5pf, the sample and hold circuit must be able to charge a 10pf (20*0.5pf) capacitor without a significant voltage change. To keep the effect of the sample and hold circuit below ± ½ lsb on a 10-bit converter, the voltage on the sample and hold circuit may vary no more than 0.05% (1/2048).

The effective capacitance of the sample and hold must, therefore, be at least 20000pf or 0.02 uf. If there is external leakage on the capacitor, its value must be increased to compensate for the leakage. At 10μA leakage; 2.5 mV (5/2048) will be lost from a 0.17 uf capacitor in 42 μS. The capacitor connected externally to the pin should, therefore, be at least 0.2 uf for best results. If the external signal changes slowly relative to 42 μS, then a larger capacitor will work well and also filter out unwanted noise.

The converter is a 10-bit, successive approximation, ratiometric converter, so the numerical value obtained from the conversion will be:

1023 * (VIN-ANGND) / (VREF-ANGND)

It can be seen that the power supply levels strongly influence the absolute accuracy of the conversion. For this reason, it is recommended that the ANGND pin be tied to a clean ground, as close to the power supply as possible. VREF should be well regulated and used only for the A to D converter. If ratiometric information is derived, VREF can be connected to VCC, but this should be done at the power supply not at the chip. It needs to be able to source around 15 milliamps. Bypass capacitors should be used between VREF and ANGND. ANGND should be within about a tenth of a volt of VSS and VREF should be within a few tenths of a volt of VCC. A 0.01 uf capacitor should be connected between the ANGND and

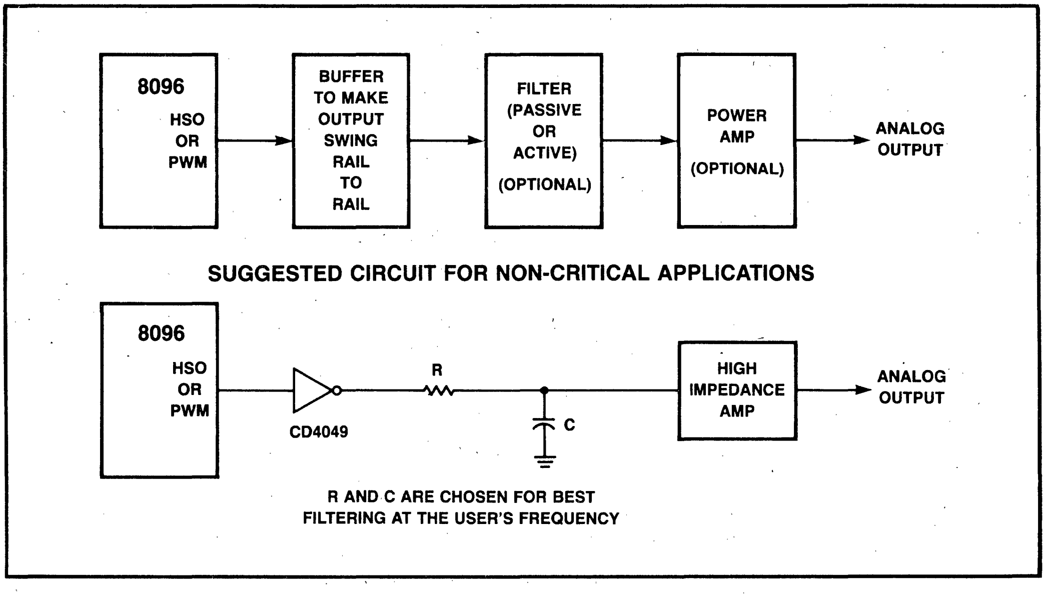


Figure 4-11. D/A Buffer Block Diagram

VBB pins to reduce the noise on VBB and provide the highest possible accuracy. Figure 4-5 shows all of these connections.

4.3.2. Analog Output Suggestions

Analog outputs can be generated by two methods, either by using the PWM output or the HSO. Either device will generate a rectangular pulse training that varies in duty cycle and (for the HSO only) period. If a smooth analog signal is desired as an output, the rectangular waveform must be filtered.

In most cases this filtering is best done after the signal is buffered to make it swing from 0 to 5 volts since both of the outputs are guaranteed only to TTL levels. A block diagram of the type of circuit needed is shown in Figure 4-11. By proper selection of components, accounting for temperature and power supply drift, a highly accurate 8-bit D to A converter can be made using either the HSO or the PWM output. If the HSO is used the accuracy could be theoretically extended to 16-bits, however the temperature and noise related problems would be extremely hard to handle.

When driving some circuits it may be desirable to use unfiltered Pulse Width Modulation. This is particularly true for motor drive circuits. The PWM output can be used to generate these waveforms if a fixed period on the order of 64 uS is acceptable. If this is not the case then the HSO unit can be used. The HSO can generate a variable waveform with a duty cycle variable in up to 65536 steps and a period of up to 131 milliseconds. Both of these outputs produce TTL levels.

4.4. I/O TIMINGS

The I/O pins on the 8096 are sampled and changed at specific times within an instruction cycle. The timings shown in this section are idealized; no propagation delay factors have been taken into account. Designing a system that depends on an I/O pin to change within a window of less than 50 nanoseconds using the information in this section is not recommended.

4.4.1. HSO Outputs

Changes in the HSO lines are synchronized to Timer 1. All of the HSO lines due to change at a certain value of a timer will change just prior to the incrementing of Timer 1. This corresponds to an internal change during Phase C, every eight state times. From an external perspective the HSO pin should change around the rising edge of CLKOUT and be stable by its falling edge.

Timer 2 is synchronized to increment no faster than Timer 1, so there will always be at least one incrementing of Timer 1 while Timer 2 is at a specific value.

4.4.2. HSI Input Sampling

The HSI pins are sampled internally once each state time. Any value on these pins must remain stable for at least 1 full state time to guarantee that it is recognized. This restriction applies even if the divide by eight mode is being used. If two events occur on the same pin within the same 8 state time window, only one of the events will be recorded. The 8 state time window, (ie. the amount of time during which Timer 1 remains constant), is stable to within about 20 nanoseconds. The window starts roughly around the rising edge of CLKOUT, however this timing is very approximate due to the amount of internal circuitry involved.

4.4.3. Standard I/O Port Pins

Port 0 is different from the other digital ports in that it is actually part of the A to D converter. The port is sampled once every 8 state times, the same frequency at which the comparator is charged-up during an A to D conversion. This 8 state times counter is not synchronized with Timer 1. If this port is used the input signal on the pin must be stable 8 state times prior to reading the SFR.

Port 1 and Port 2 have quasi-bidirectional I/O pins. When used as inputs the data on these pins must be stable one state time prior to reading the SFR. This timing is also valid for the input-only pins of Port 2. When used as outputs, the quasi-bidirectional pins will change state shortly after CLKOUT falls. If the change was from '0' to a '1' the low impedance pull-up will remain on for one state time after the change.

Ports 3 and 4 are addressed as off-chip memory-mapped I/O. The port pins will change state shortly after the rising edge of CLKOUT. When these pins are used as Ports 3 and 4 they are open drain, their structure is different when they are used as part of the bus.

4.5. SERIAL PORT TIMINGS

The serial port on the 8096 was designed to be compatible with the 8051 serial port. Since the 8051 uses a divide by 2 clock and the 8096 uses a divide by 3, the serial port on the 8096 had to be provided with its own clock circuit to maximize its compatibility with the 8051 at high baud rates. This means that the serial port itself does not know about state times. There is circuitry which is synchronized to the serial port and to the rest of the 8096 so that information can be passed back and forth.

The baud rate generator is clocked by either XTAL1 or T2CLK, because T2CLK needs to be synchronized to the XTAL1 signal its speed must be limited to ¹⁄₁₆ that of XTAL1. The serial port will not function during the time between the consecutive writes to the baud rate register. Section 2.11.4 discusses programming the baud rate generator.

4.5.1. Mode 0

Mode 0 is the shift register mode. The TXD pin sends out a clock train, while the RXD pin transmits or receives the data. Figure 4-12 shows the waveforms and timing. Note that the port starts functioning when a '1' is written to the REN (Receiver Enable) bit in the serial port control register. If REN is already high, clearing the RI flag will start a reception.

In this mode the serial port can be used to expand the I/O capability of the 8096 by simply adding shift registers.

A schematic of a typical circuit is shown in Figure 4-13. This circuit inverts the data coming in, so it must be re-inverted in software. The enable and latch connections to the shift registers can be driven by decoders, rather than directly from the low speed I/O ports, if the software and hardware are properly designed.

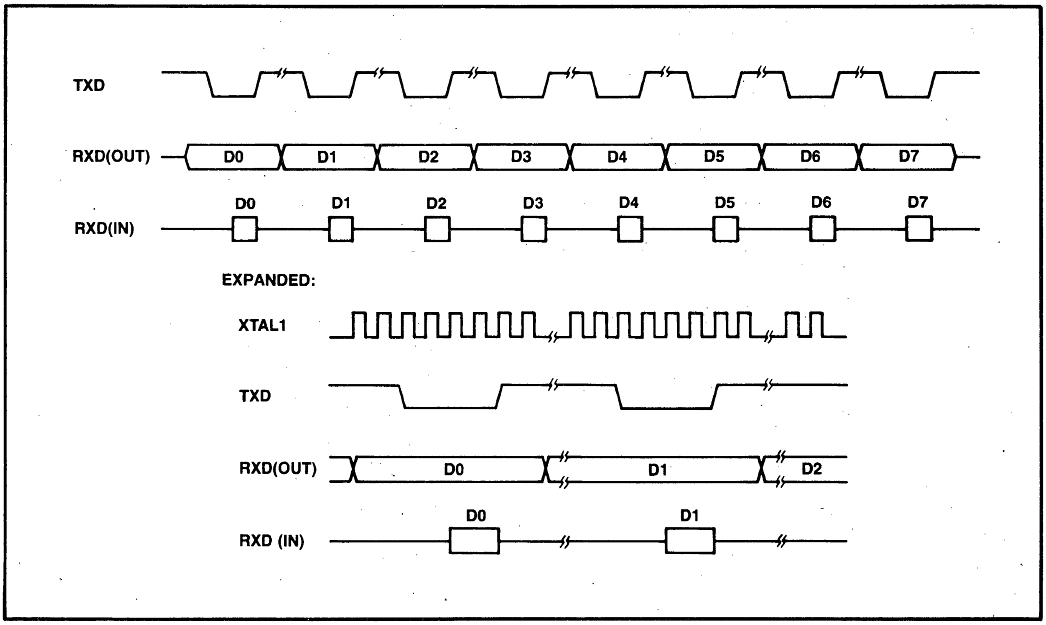


Figure 4-12. Serial Port Timings in Mode 0

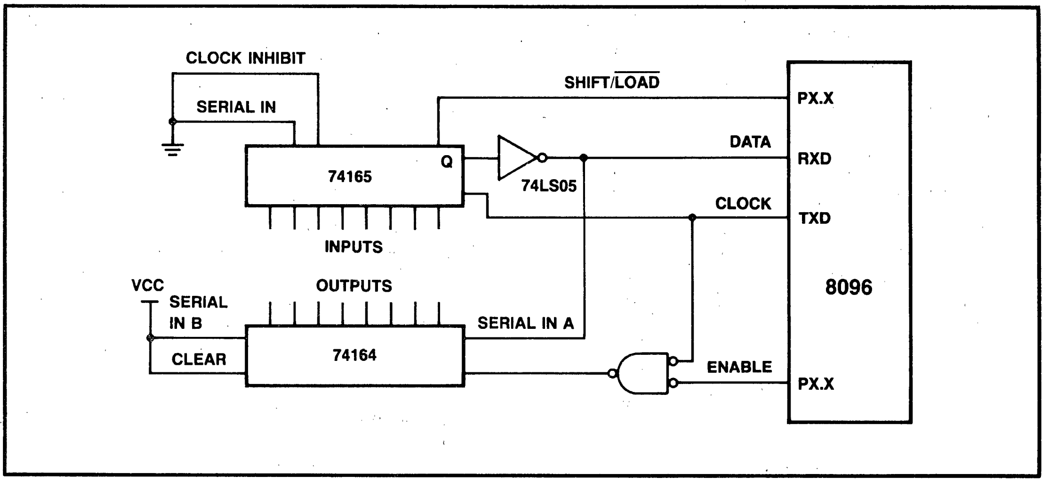


Figure 4-13. Mode 0 Serial Port Example

4.5.2. Mode 1 Timings

Mode 1 operation of the serial port makes use of 10-bit data packages, a start bit, 8 data bits and a stop bit. The transmit and receive functions are controlled by separate shift clocks. The transmit shift clock starts when the baud rate generator is initialized, the receive shift clock is reset when a '1 to 0' transition (start bit) is received. The transmit clock may therefore not be in sync with the receive clock, although they will both be at the same frequency.

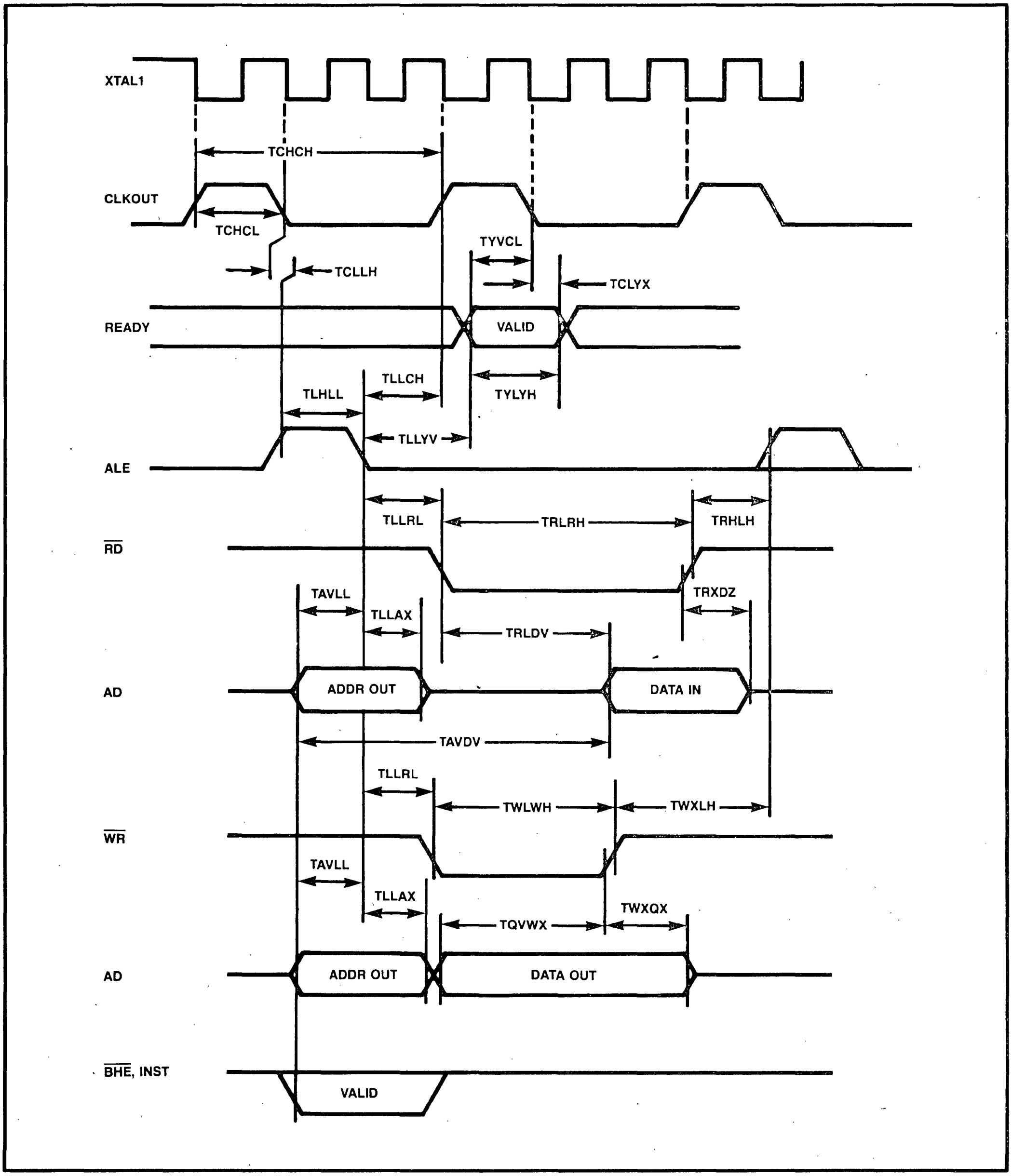


Figure 4-14. Bus Signal Timings

The TI (Transmit Interrupt) and RI (Receive Interrupt) flags are set to indicate when operations are complete. TI is set when the last data bit of the message has been sent, not when the stop bit is sent. If an attempt to send another byte is made before the stop bit is sent the port will hold off transmission until the stop bit is complete. RI is set when 8 data bits are received, not when the stop bit is received. Note that when the serial port status register is read both TI and RI are cleared.

Caution should be used when using the serial port to connect more than two devices in half-duplex, (ie. one wire for transmit *and* receive). If the receiving processor does not wait for one bit time after RI is set before starting to transmit, the stop bit on the link could be squashed. This could cause a problem for other devices listening on the link.

4.5.3. Mode 2 and 3 Timings

Modes 2 and 3 operate in a manner similar to that of mode 1. The only difference is that the data is now made up of 9 bits, so 11-bit packages are transmitted and received. This means that TI and RI will be set on the 9th data bit

Tosc — Oscillator Period, one cycle time on XTAL1.

Timings The Memory System Must Meet

TYVCL — READY valid to CLKOUT low: This is a fixed time of around 40 ns, if it is violated the part could malfunction, necessitating a chip reset.

TLLYV — ALE low to READY VALID: This spec is just a more convenient form of TYVCL to use.

TYLYH — READY low to READY high: Maximum time the part can be in the not-ready state. If it is exceeded, the 8096 dynamic nodes which hold the current instruction may 'forget' how to finish the instruction.

TAVDV — ADDRESS valid to DATA valid: Maximum time that the memory has to output valid data after the 8096 outputs a valid address. Nominally, a maximum of 5 Tosc periods.

TRLDV — $\overline{READ}$ low to DATA valid: Maximum time that the memory has to output data after $\overline{READ}$ goes low. Nominally, a maximum of 3 Tosc periods.

TRXDZ — $\overline{READ}$ not low to DATA float: Time after $\overline{READ}$ is no longer low until the memory must float the bus. The memory signal can be removed as soon as $\overline{READ}$ is not low, and must be removed within the specified maximum time.
Nominally, a maximum of 1 Tosc period.

Timings the 8096 Will Provide

TCHCH — CLKOUT high to CLKOUT high: The period of CLKOUT and the duration of one state time. Always 3 Tosc average, but individual periods could vary by a few nanoseconds.

TCHCL — CLKOUT high to CLKOUT low: Nominally 1 Tosc period.

TCLLH — CLKOUT low to ALE high: A help in deriving other timings, typically plus or minus 5 to 10 ns.

TLLCH — ALE low to CLKOUT high: Used to derive other timings, nominally 1 Tosc period.

TLHLL — ALE high to ALE low: ALE pulse width. Useful in determining ALE rising edge to ADDRESS valid time. Nominally 1 Tosc period.

TAVLL — ADDRESS valid to ALE low: Length of time ADDRESS is valid before ALE falls. Important timing for address latch circuitry. Nominally 1 Tosc period.

TLLAX — ALE low to ADDRESS invalid: Length of time ADDRESS is valid after ALE falls. Important timing for address latch circuitry. Nominally 1 Tosc period.

TLLRL — ALE low to $\overline{READ}$ or $\overline{WRITE}$ low: Length of time after ALE falls before $\overline{RD}$ or $\overline{WR}$ fall. Could be needed to ensure that proper memory decoding takes place before it is output enabled. Nominally 1 Tosc period.

TRLRH — $\overline{READ}$ low to $\overline{READ}$ high: $\overline{RD}$ pulse width, nominally 1 Tosc period.

TRHLH — $\overline{READ}$ high to ALE high: Time between $\overline{RD}$ going inactive and next ALE, also used to calculate time between $\overline{RD}$ inactive and next ADDRESS valid. Nominally 1 Tosc period.

TWLWH — $\overline{WRITE}$ low to $\overline{WRITE}$ high: Write pulse width, nominally 2 Tosc periods.

TQVWX — OUTPUT valid to $\overline{WRITE}$ not low: time that the OUTPUT data is valid before $\overline{WR}$ starts to go high. Nominally 2 Tosc periods.

TWXQX — $\overline{WRITE}$ not low to OUTPUT not valid: Time that the OUTPUT data is valid after $\overline{WR}$ starts to rise. Nominally 1 Tosc period.

TWXLH — $\overline{WRITE}$ not low to ALE high: Time between write starting to rise and next ALE, also used to calculate the time between $\overline{WR}$ starting to rise and next ADDRESS valid. Nominally 2 Tosc periods.

Figure 4-15. Timing Specification Explanations

rather than the 8th. The 9th bit can be used for parity or multiple processor communications (see section 2.11).

4.6. BUS TIMING AND MEMORY INTERFACE

4.6.1. Bus Functionality

The 8096 has a multiplexed (address/data) 16 bit bus. There are control lines provided to demultiplex the bus (ALE), indicate reads or writes ($\overline{RD}$, $\overline{WR}$), indicate if the access is for an instruction (INST), and separate the bus into high and low bytes ($\overline{BHE}$, AD0). Section 2.3.5 contains an overview of the bus operation.

4.6.2. Timing Specifications

Figure 4-14 shows the timing of the bus signals and data lines. Since this is a new part, the exact timing specifications are subject to change, please refer to the latest 8096 data sheet to ensure that your system is designed to the proper specifications. The major timing specifications are described in Figure 4-15.

4.6.3. READY Line Usage

When the processor has to address a memory location that cannot respond within the standard specifications it is necessary to use the READY line to generate wait states. When the READY line is held low the processor waits in a loop for the line to come high. There is a maximum time that the READY line can be held low without risking a processor malfunction due to dynamic nodes that have not been refreshed during the wait states. This time is shown as TYLYH in the data sheet.

In most cases the READY line is brought low after the address is decoded and it is determined that a wait state is needed. It is very likely that some addresses, such as those addressing memory mapped peripherals, would need wait states, and others would not. The READY line must be stable within the TLLYV specification after ALE falls (or the TYVCL before CLKOUT falls) or the processor could lock-up. There is no requirement as to when READY may go high, as long as the maximum READY low time (TYLYH) is not violated.

4.6.4. INST Line Usage

The INST (Instruction) line is high during the output of an address that is for an instruction stream fetch. It is low during the same time for any other memory access. At any other time it is not valid. This pin is not present on the 48-pin versions. The INST signal can be used with a logic analyzer to debug a system. In this way it is possible to determine if the fetch was for instructions or data, making the task of tracing the program much easier.

4.6.5. Address Decoding

The multiplexed bus of the 8096 must be demultiplexed before it can be used. This ca be done with 2 74LS373 transparent latches. As explained in section 2.3.5, the latched address signal will be referred to as MA0 through MA15. (Memory Address), and the data lines will be called MD0 through MD15, (Memory Data).

Since the 8096 can make accesses to memory for either bytes or words it is necessary to have a way of determining the type of access desired. The $\overline{BHE}$ and MA0 lines are used for this purpose. $\overline{BHE}$ must be latched, as it is valid only when the address is valid. The memory system is typically set up as 32K by 16, instead of 64K by 8. When the $\overline{BHE}$ line is low, the upper byte is enabled. When MA0 is low, the lower byte is enabled when MA0 is high $\overline{BHE}$ will be low, and the upper byte is enabled.

When external RAM and EPROM are both used in the system the control logic can be simplified a little to some of the addresses. The 8096 will always output BHE to indicate if a read is of the high byte or the low byte, but it discards the byte it is not going to use. It is therefore possible to use the $\overline{BHE}$ and MA0 lines only to control

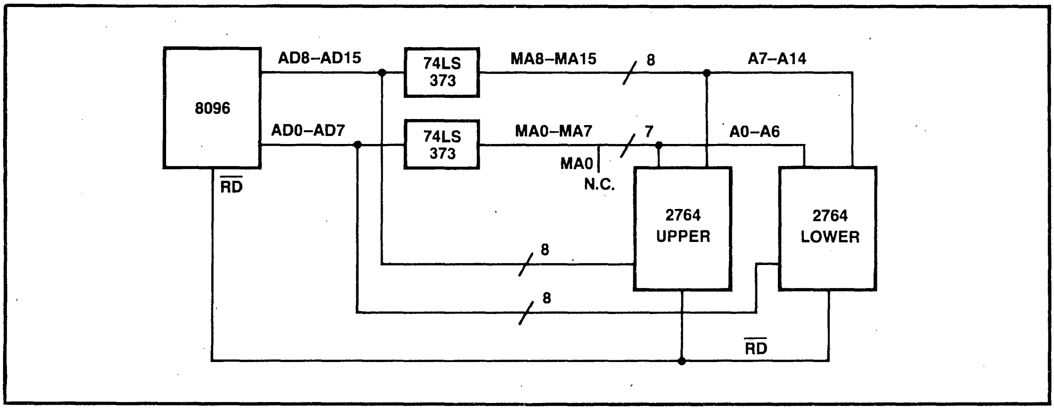


Figure 4-16. Memory Wiring Example—EPROM Only System

memory writes, and to ignore these lines during memory reads. Figure 4-16 and 4-17 show block diagrams of two memory systems, an external EPROM only system and a RAM/ROM system.

4.6.6. System Verification Example

To verify that a system such as the one in Figure 4-17 will work with the 8096, it is necessary to check all of the timing parameters. Let us examine this system one parameter at a time using the proposed 8096 specifications. These specifications are subject to change, refer to the latest 8096 data sheet for the current specifications.

The timings of signals that the processor and memory use are effected by the latch and buffer circuitry. The timings of the signal provided by the processor are delayed by various amounts of time. Similarly, the signals coming back from the memory are also delayed. The calculations involved in verifying this system follow:

Address Valid Delay — 20 nanoseconds

The address lines are delayed by passing them through the 74LS373s, this delay is specified at 18ns after Address is valid or 30ns after ALE is high. Since the signal may be limited by either the ALE timing or the Address timing, these two cases must be considered.

If Limited by ALE:

Minimum ALE pulse width =	Tosc-10 (TLHLL)
Minimum Addr set-up to ALE falling =	Tosc-20 (TAVLL)

Therefore ALE could occur 10 ns after Address valid.

Total delay from 8096 Address stable to MA (Memory Address) stable would be:

ALE delay from address	−10
74LS373 clock to output	30
	20 nanoseconds

If Limited by Address Valid:

74LS373 Data Valid to Data Output = 18 nanoseconds

In the worst case, the delay in Address valid is controlled by ALE and has a value of 20 nanoseconds.

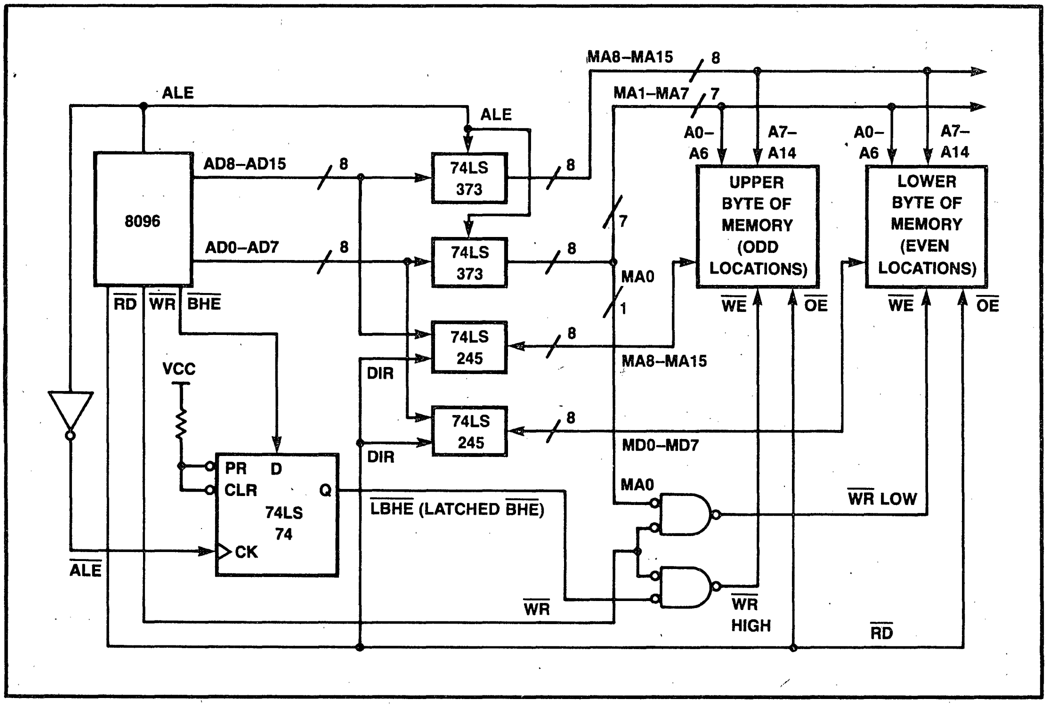


Figure 4-17. RAM/ROM Memory System

Delay of Data Transfer to/from Processor — 12 nanoseconds

The $\overline{\text{RD}}$ low to Data valid specification (TRLDV) is 3 Tosc-50, (200 ns at 12 MHz). The 74LS245 is enabled by $\overline{\text{RD}}$ and has a delay of 40 ns from enable. The enable delay is clearly not a problem.

The 74LS245 is enabled except during a read, so there is no enable delay to consider for write operaions.

The Data In to Data Out delay of the 74LS245 is 12 ns.

Delay of $\overline{\text{WR}}$ signal to memory — 15 nanoseconds

Latched $\overline{\text{BHE}}$ is delayed by the inverter on ALE and the 74LS74.

74LS04 delay (Output low to high)	= 22
74LS74 delay (Clock to Output)	= 40
Delay of Latched $\overline{\text{BHE}}$ from ALE falling	= 62 nanoseconds

The 74LS74 requires data valid for 20 ns prior to the clock, the 8096 will have $\overline{\text{BHE}}$ stable Tosc-20 ns (TAVLL, 63 ns at 12 MHz) prior to ALE falling. There is no problem here.

MA0 is valid prior to ALE falling, since the 20 ns Address Delay is less than TAVLL.

$\overline{\text{WR}}$ will fall no sooner than Tosc-20 ns (TLLRL, 63 ns at 12 MHz) after ALE goes low. It will therefore be valid just after the Latched $\overline{\text{BHE}}$ is valid, so it is the controlling signal.

$\overline{\text{WR}}$ High and $\overline{\text{WR}}$ Low are valid 15 ns after MA0, Latched $\overline{\text{BHE}}$ and $\overline{\text{WR}}$ are valid. Since $\overline{\text{WR}}$ is the last signal to go valid, the delay of $\overline{\text{WR}}$ (High and Low) to memory is 15 ns.

Delay Summary —	Address Delay	= 20 ns
	Data Delay	= 12 ns
	$\overline{\text{WR}}$ Delay	= 15 ns
	$\overline{\text{RD}}$ Delay	= 0 ns

Characteristics of a 12 MHz 8096 system with latches:

Required by system:

Address valid to Data in;

TAVDV	:	386.6 ns maximum
Address Delay	:	− 20.0 ns maximum
Data Delay	:	− 12.0 ns maximum
		354.6 ns maximum

Read low to Data in;

TRLDV	:	200.0 ns maximum
$\overline{\text{RD}}$ Delay	:	− 00.0 ns maximum
Data Delay	:	− 12.0 ns maximum
		188.0 ns maximum

Provided by System:

Address valid to Control;

TLLRL	:	63.3 ns minimum
TAVLL	:	63.3 ns minimum
Address Delay	:	− 20.0 ns maximum
$\overline{\text{WR}}$ Delay	:	− 00.0 ns minimum (no spec)
		101.6 ns minimum

Write Pulse Width;

TWLWH	:	151.6 ns minimum
Rising $\overline{\text{WR}}$ Delay	:	− 15.0 ns maximum
Falling $\overline{\text{WR}}$ Delay	:	00.0 ns minimum (no spec)
		146.6 ns minimum

Data Setup to $\overline{\text{WR}}$ rising;

TQVWX	:	136.6 ns minimum
Data Delay	:	− 12.0 ns maximum
$\overline{\text{WR}}$ Delay	:	00.0 ns minimum (no spec)
		124.6 ns minimum

Data Hold after $\overline{\text{WR}}$;

TWXQX	:	58.3 ns minimum
Data Delay	:	0.0 ns minimum (no spec)
$\overline{\text{WR}}$ Delay	:	− 15.0 ns maximum
		43.3 ns minimum

The two memory devices which are expected to be used most often with the 8096 are the 2764 EPROM and the 2128 RAM. The system verification for the 2764 is simple.

2764 Tac

(Address valid to Output) < Address valid to Data in
250 ns < 354 ns O.K.

2764 Toe

(Output Enable to Output) < $\overline{\text{Read}}$ low to Data in
100 ns < 188 ns O.K.

These calculations assume no address decoder delays and no delays on the $\overline{\text{RD}}$ (OE) line. If there are delays in these signals the delays must be added to the 2764's timing.

The read calculations for the 2128 are similar to those for the 2764.

2128-20 Tac < Address valid to Data in
200 ns < 354 ns O.K.

2128-20 Toe < $\overline{\text{Read}}$ low to Data in
65 ns < 188 ns O.K.

The write calculations are a little more involved, but still straight-forward.

2128 Twp (Write Pulse)	<	Write Pulse Width
100 ns	<	146 ns O.K.
2128 Tds (Data Setup)	<	Data Setup to $\overline{WR}$ rising
65 ns	<	124 ns O.K.
2128 Tdh (Data Hold)	<	Data Hold after $\overline{WR}$
0 ns	<	43 ns

All of the above calculations have been done assuming that no components are in the circuit except for those shown in Figure 4-17. If additional components are added, as may be needed for address decoding or memory bank switching, the calculations must be updated to reflect the actual circuit.

4.6.7. I/O Port Reconstruction

When a single-ship system is being designed using a multiple chip system as a prototype, it may be necessary to reconstruct I/O ports 3 and 4 using a memory-mapped I/O technique. The circuit shown in Figure 4-18 provides this function on the iSBE-96 emulator board. It can be attached to any 8096 system which has the required address decoding and bus demultiplexing.

The output circuitry is basically just a latch that operates when 1FFEH or 1FFFH are placed on the MA lines. The inverters surrounding the latch create an open-collector output to emulate the open-drain output found on the 8096. The 'reset' line is used to set the ports to all 1's when the 8096 is reset. It should be noted that the voltage and current characteristics of this port will differ from those of the 8096, but the basic functionality will be the same.

The input circuitry is just a bus transceiver that is addressed at 1FFEH or 1FFFH. If the ports are going to be used for either input or output, but not both, some of the circuitry can be eliminated.

4.7. NOISE PROTECTION TIPS

Designing controllers differs from designing other computer equipment in the area of noise protection. A microcontroller circuit under the hood of a car, in a photocopier, CRT terminal, or a high speed printer is subject

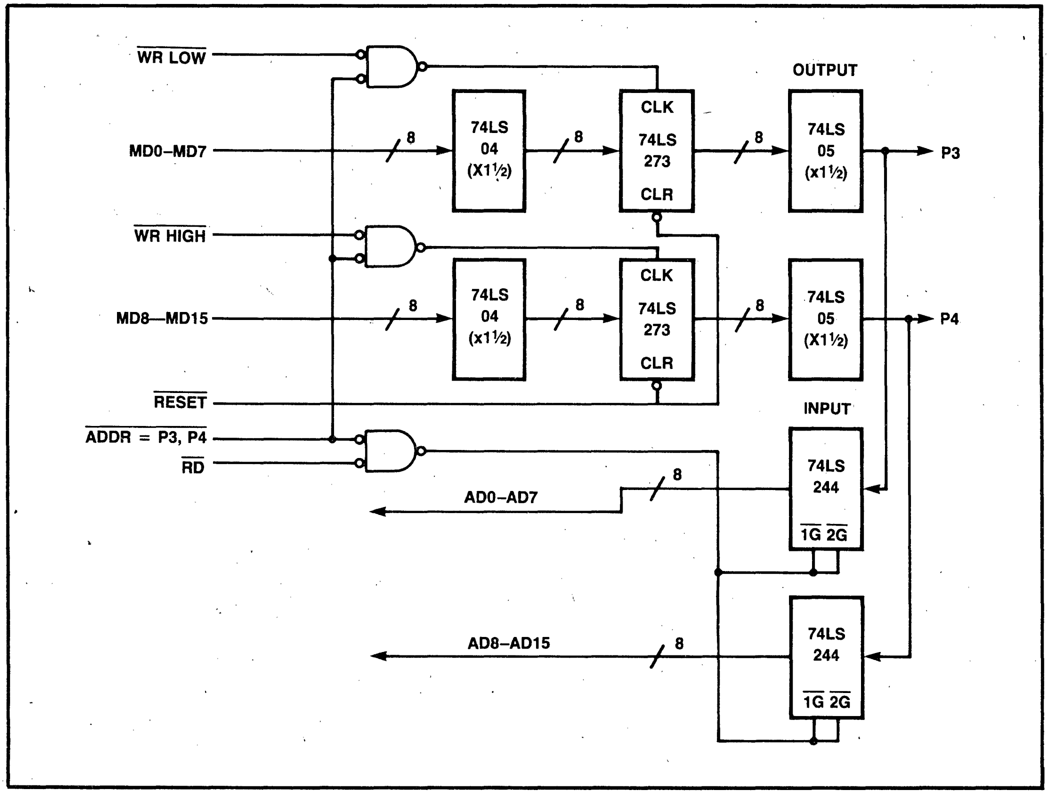


Figure 4-18. I/O Port Reconstruction

to many types of electrical noise. Noise can get to the processor directly through the power supply, or it can be induced onto the board by electromagnetic fields. It is also possible for the pc board to find itself in the path of electrostatic discharges. Glitches and noise on the pc board can cause the processor to act unpredictably, usually by changing either the memory locations or the program counter.

There are both hardware and software solutions to noise problems, but the best solution is good design practice and a few ounces of prevention. The 8096 has a watchdog timer which will reset the part if it fails to execute the software properly. The software should be set up to take advantage of this feature.

It is also recommended that unused areas of code be filled with NOPs and periodic jumps to an error routine or RST (reset chip) instructions. This is particularly important in the code around lookup tables, since if lookup tables are executed all sorts of bad things can happen. Wherever space allows, each table should be surrounded by 7 NOPs (the longest 8096 instruction has 7 bytes) and a RST or jump to error routine instruction. This will help to ensure a speedy recovery should the processor have a glitch in the program flow.

Many hardware solutions exist for keeping pc board noise to a minimum. Ground planes, gridded ground and VCC structures, bypass capacitors, transient absorbers and power busses with built-in capacitors can all be of great help. It is much easier to design a board with these features than to try to retrofit them later. Proper pc board layout is probably the single most important and, unfortunately, least understood aspect of project design. Minimizing loop areas and inductance, as well as providing clean grounds are very important. More information on protecting against noise can be found in the Intel Application Note AP-125, ''Designing Microcontroller Systems For Noisy Environments.''

4.8. PACKAGING PINOUTS AND ENVIRONMENT

The MCS-96 family of products is offered in many versions. They are available in 48-pin or 68-pin packages, with or without ROM, and with or without an A to D converter. A summary of the available options is shown in Figure 4-19.

The 48-pin versions are available in a 48-pin DIP (Dual In-Line) package, in either ceramic or plastic.

The 68-pin versions are available in a ceramic pin grid array, and a plastic flatpack. A plastic pin grid array will be available in the near future.

	ROMLESS		WITH ROM	
	68-pin	48-pin	68-pin	48-pin
Without A to D	8096	8094	8396	8394
With A to D	8097	8095	8397	8395

Figure 4-19. The MCS®-96 Family of Products

MCS®-96 Data Sheets

ADVANCE INFORMATION

8094/8095/8096/8097
8394/8395/8396/8397
16-BIT MICROCONTROLLERS

- **839X: an 809X with 8K Bytes of On-chip ROM**

- **High Speed Pulse I/O**
- **10-bit A/D Converter**
- **8 Interrupt Sources**
- **Pulse-Width Modulated Output**
- **Four 16-bit Software Timers**
- **232 Byte Register File**
- **Memory-to-Memory Architecture**
- **Full Duplex Serial Port**
- **Five 8-bit I/O Ports**
- **Watchdog Timer**

The MCS-96 family of 16-bit microcontrollers consists of 8 members, all of which are designed for high-speed control functions.

The CPU supports bit, byte, and word operations. 32-bit double-words are supported for a subset of the instruction set. With a 12 MHz input frequency the 8096 can do a 16-bit addition in 1.0 μsec and a 16 x 16-bit multiply or 32/16-bit divide in 6.5 μsec. Instruction execution times average 1 to 2 μsec in typical applications.

Four high-speed trigger inputs are provided to record the times at which external events occur. Six high-speed pulse generator outputs are provided to trigger external events at preset times. The high-speed output unit can simultaneously perform timer functions. Up to four such 16-bit Software Timers can be in operation at once.

An on-chip A/D Converter converts up to 4 (in the 48-pin version) or 8 (in the 68-pin version) analog input channels to 10-bit digital values. This feature is only available on the 8095, 8395, 8097 and 8397.

Also provided on-chip are a serial port, a watchdog timer, and a pulse-width modulated output signal.

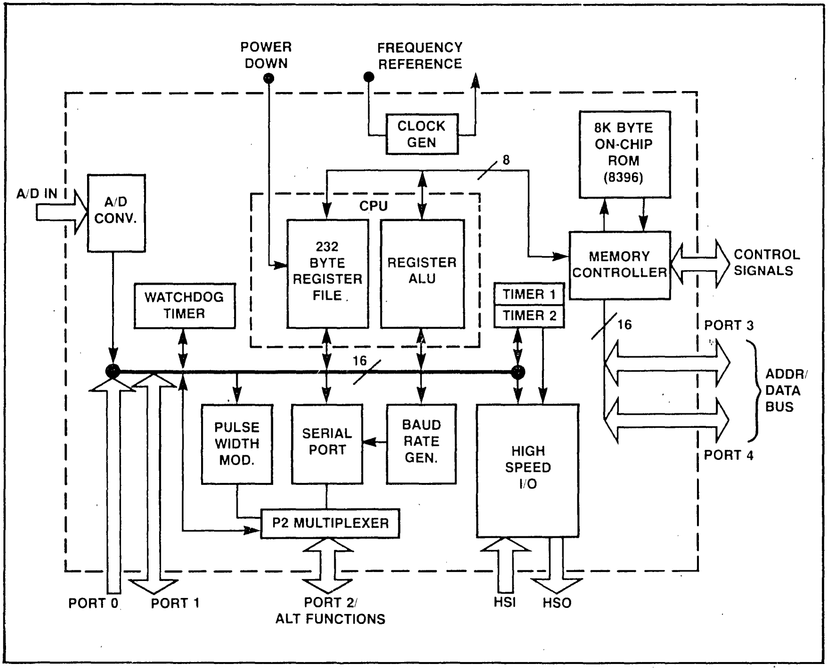


Figure 1. Block Diagram (For simplicity, lines connecting port registers to port buffers are not shown.)

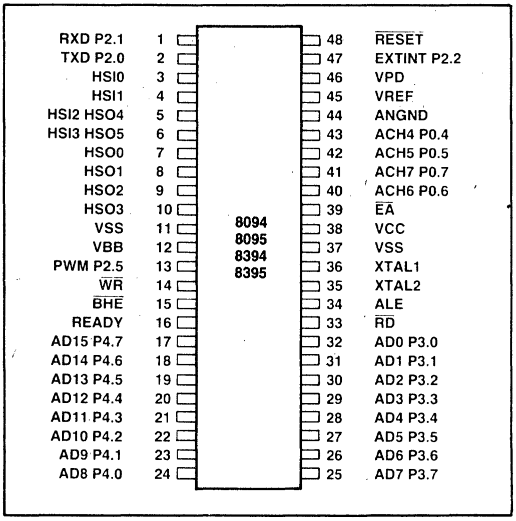


Figure 2. 48-Pin Package

Figure 1 shows a block diagram of the MCS-96 parts, generally referred to as the 8096. The 8096 is available in 48-pin and 68-pin packages, with and without A/D, and with and without on-chip ROM. The MCS-96 numbering system is shown below:

OPTIONS		68 PIN	48 PIN
DIGITAL I/O	ROMLESS	8096	8094
	ROM	8396	8394
ANALOG AND DIGITAL I/O	ROMLESS	8097	8095
	ROM	8397	8395

Figures 2 and 3 show the pinouts for the 48- and 68-pin packages.

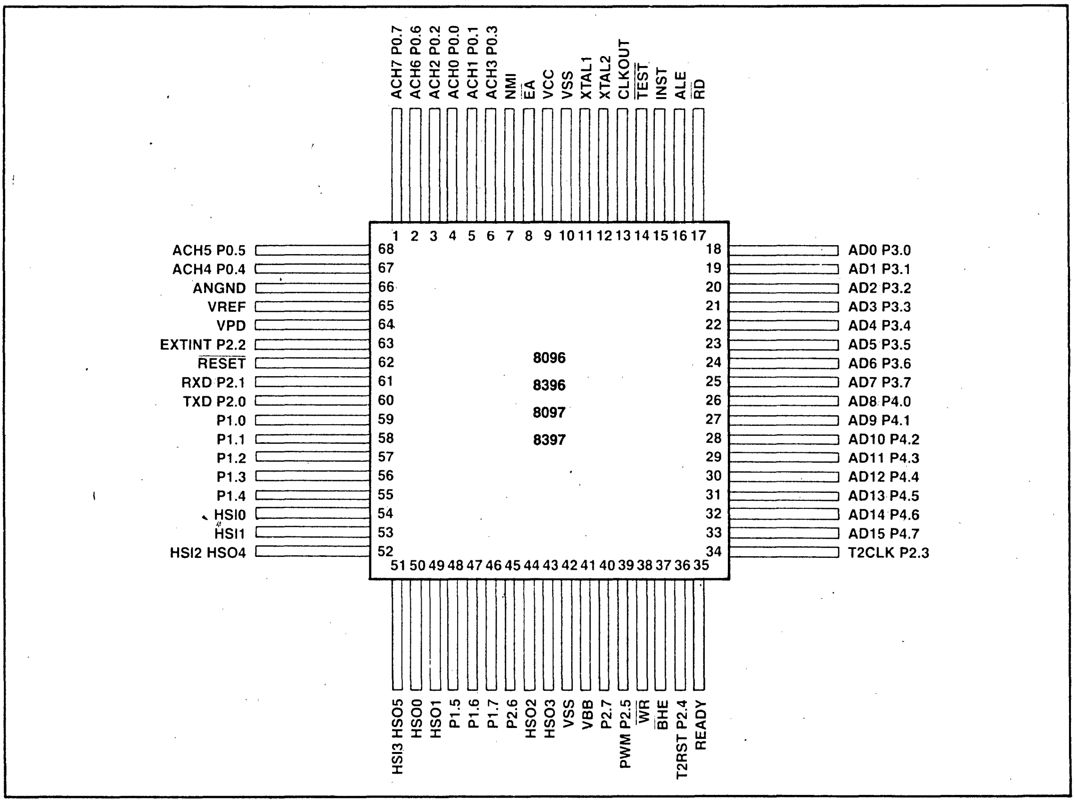


Figure 3. 68-Pin Package

Instruction Summary

Mnemonic	Oper-ands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
ADD/ADDB	2	D ← D+A	✓	✓	✓	✓	✓	—	
ADD/ADDB	3	D ← B+A	✓	✓	✓	✓	✓	—	
ADDC/ADDCB	2	D ← D+A+C	✓	✓	✓	✓	✓	—	
SUB/SUBB	2	D ← D − A	✓	✓	✓	✓	✓	—	
SUB/SUBB	3	D ← B − A	✓	✓	✓	✓	✓	—	
SUBC/SUBCB	2	D ← D − A + C − 1	✓	✓	✓	✓	✓	—	
CMP/CMPB	2	D − A	✓	✓	✓	✓	✓	—	
MUL/MULU	2	D, D + 2 ← D * A	—	—	—	—	—	✓	2
MUL/MULU	3	D, D + 2 ← B * A	—	—	—	—	—	✓	2
MULB/MULUB	2	D, D + 1 ← D * A	—	—	—	—	—	✓	3
MULB/MULUB	3	D, D + 1 ← B * A	—	—	—	—	—	✓	3
DIV/DIVU	2	D ← (D, D + 2)/A D + 2 remainder	—	—	—	✓	✓	—	2
DIVB/DIVUB	3	D ← (D, D + 1)/A D + 1 remainder	—	—	—	✓	✓	—	3
AND/ANDB	2	D ← D and A	✓	✓	0	0	—	—	
AND/ANDB	3	D ← B and A	✓	✓	0	0	—	—	
OR/ORB	2	D ← D or A	✓	✓	0	0	—	—	
XOR/XORB	2	D ← D (excl. or) A	✓	✓	0	0	—	—	
LD/LDB	2	D ← A	—	—	—	—	—	—	
ST/STB	2	A ← D	—	—	—	—	—	—	
LDBSE	2	D ← A; D + 1 ← SIGN(A)	—	—	—	—	—	—	3,4
LDBZE	2	D ← A; D + 1 ← 0	—	—	—	—	—	—	3,4
PUSH	1	SP ← SP − 2; (SP) A	—	—	—	—	—	—	
POP	1	A ← (SP); SP ← SP + 2	—	—	—	—	—	—	
PUSHF	0	SP ← SP − 2; (SP) ← PSW; PSW ← 0000H	0	0	0	0	0	0	
POPF	0	PSW ← (SP); SP ← SP + 2	✓	✓	✓	✓	✓	✓	
SJMP	1	PC ← PC + 11-bit offset	—	—	—	—	—	—	5
LJMP	1	PC ← PC + 16-bit offset	—	—	—	—	—	—	5
INDJMP	1	PC ← (A)	—	—	—	—	—	—	
SCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 11-bit offset	—	—	—	—	—	—	5
LCALL	1	SP ← SP − 2; (SP) ← PC; PC ← PC + 16-bit offset	—	—	—	—	—	—	5
RET	0	PC ← (SP); SP ← SP + 2	—	—	—	—	—	—	
J(conditional)	1	PC ← PC + 8-bit offset	—	—	—	—	—	—	5
JC		Jump if C = 1	—	—	—	—	—	—	5
JNC		Jump if C = 0	—	—	—	—	—	—	5

Note

1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B, and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.
2. D, D + 2 are consecutive WORDS in memory; D is DOUBLE-WORD aligned.
3. D, D + 1 are consecutive BYTES in memory; D is WORD aligned.
4. Changes a byte to a word.
5. Offset is a 2's complement number.

Instruction Summary (continued)

Mnemonic	Oper-ands	Operation (Note 1)	Flags						Notes
			Z	N	C	V	VT	ST	
JE		Jump if Z = 1	—	—	—	—	—	—	5
JNE		Jump if Z = 0	—	—	—	—	—	—	5
JGE		Jump if N = 1	—	—	—	—	—	—	5
JLT		Jump if N = 0	—	—	—	—	—	—	5
JGT		Jump if N = 0 and Z = 0	—	—	—	—	—	—	5
JLE		Jump if N = 1 or Z = 1	—	—	—	—	—	—	5
JH		Jump if C = 1 and Z = 0	—	—	—	—	—	—	5
JNH		Jump if C = 0 or Z = 1	—	—	—	—	—	—	5
JV		Jump if V = 1	—	—	—	—	—	—	5
JNV		Jump if V = 0	—	—	—	—	—	—	5
JVT		Jump if VT = 1; Clear VT	—	—	—	—	0	—	5
JNVT		Jump if VT = 0; Clear VT	—	—	—	—	0	—	5
JST		Jump if ST = 1	—	—	—	—	—	—	5
JNST		Jump if ST = 0	—	—	—	—	—	—	5
JBS		Jump if ST = 1	—	—	—	—	—	—	5,6
JBC		Jump if Specified Bit = 0	—	—	—	—	—	—	5,6
DJNZ	1	D ← D − 1; if D ≠ then PC ← PC + 8-bit offset	—	—	—	—	—	—	5
DEC/DECB	1	D ← D − 1	√	√	√	√	√	—	
NEG/NEGB	1	D ← 0 − D	√	√	√	√	√	—	
INC/INCB	1	D ← D + 1	√	√	√	√	√	—	
EXT	1	D ← D; D + 2 ← Sign (D)	√	√	0	0	—	—	2
EXTB	1	D ← D; D + 1 ← Sign (D)	√	√	0	0	—	—	3
NOT/NOTB	1	D ← Logical Not (D)	√	√	0	0	—	—	
CLR/CLRB	1	D ← 0	1	0	0	0	—	—	
SHL/SHLB/SHLL	1	C ← msb — — — — — — lsb ← 0	√	√	√	√	√	—	7
SHR/SHRB/SHRL	1	0 → msb — — — — — — lsb → C	√	√	√	0	—	√	7
SHRA/SHRAB/SHRAL	1	msb → msb — — — — — — lsb → C	√	√	√	0	—	√	7
SETC	0	C ← 1	—	—	1	—	—	—	
CLRC	0	C ← 0	—	—	0	—	—	—	
CLRVT	0	VT ← 0	—	—	—	—	0	—	
RST	0	PC ← 2080H	0	0	0	0	0	0	8
DI	0	Disable All Interrupts	—	—	—	—	—	—	
EI	0	Enable All Interrupts	—	—	—	—	—	—	
NOP	0	PC ← PC + 1	—	—	—	—	—	—	
SKIP	0	PC ← PC + 2	—	—	—	—	—	—	
NORML	2	Normalize	√	1	—	—	—	—	7

Note

1. If the mnemonic ends in "B", a byte operation is performed, otherwise a word operation is done. Operands D, B and A must conform to the alignment rules for the required operand type. D and B are locations in the register file; A can be located anywhere in memory.
5. Offset is a 2's complement number.
6. Specified bit is one of the 2048 bits in the register file.
7. The "L" (Long) suffix indicates double-word operation.
8. Initiates a Reset by pulling RESET low. Software should re-initialize all the necessary registers with code starting at 2080H.

Opcode and State Time Listing

		DIRECT			IMMEDIATE			INDIRECT⊛					INDEXED⊛				
								NORMAL			AUTO-INC.		SHORT			LONG	
MNEMONIC	OPERANDS	OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES
ARITHMETIC INSTRUCTIONS																	
ADD	2	64	3	4	65	4	5	66	3	6/11	3	7/12	67	4	6/11	5	7/12
ADD	3	44	4	5	45	5	6	46	4	7/12	4	8/13	47	5	7/12	6	8/13
ADDB	2	74	3	4	75	3	4	76	3	6/11	3	7/12	77	4	6/11	5	7/12
ADDB	3	54	4	5	55	4	5	56	4	7/12	4	8/13	57	5	7/12	6	8/13
ADDC	2	A4	3	4	A5	4	5	A6	3	6/11	3	7/12	A7	4	6/11	5	7/12
ADDCB	2	B4	3	4	B5	3	4	B6	3	6/11	3	7/12	B7	4	6/11	5	7/12
SUB	2	68	3	4	69	4	5	6A	3	6/11	3	7/12	6B	4	6/11	5	7/12
SUB	3	48	4	5	49	5	6	4A	4	7/12	4	8/13	4B	5	7/12	6	8/13
SUBB	2	78	3	4	79	3	4	7A	3	6/11	3	7/12	7B	4	6/11	5	7/12
SUBB	3	58	4	5	59	4	5	5A	4	7/12	4	8/13	5B	5	7/12	6	8/13
SUBC	2	A8	3	4	A9	4	5	AA	3	6/11	3	7/12	AB	4	6/11	5	7/12
SUBCB	2	B8	3	4	B9	3	4	BA	3	6/11	3	7/12	BB	4	6/11	5	7/12
CMP	2	88	3	4	89	4	5	8A	3	6/11	3	7/12	8B	4	6/11	5	7/12
CMPB	2	98	3	4	99	3	4	9A	3	6/11	3	7/12	9B	4	6/11	5	7/12
MULU	2	6C	3	25	6D	4	26	6E	3	27/32	3	28/33	6F	4	27/32	5	28/33
MULU	3	4C	4	26	4D	5	27	4E	4	28/33	4	29/34	4F	5	28/33	6	29/34
MULUB	2	7C	3	17	7D	3	17	7E	3	19/24	3	20/25	7F	4	19/24	5	20/25
MULUB	3	5C	4	18	5D	4	18	5E	4	20/25	4	21/26	5F	5	20/25	6	21/26
MUL	2	②	4	29	②	5	30	②	4	31/36	4	32/37	②	5	31/36	6	32/37
MUL	3	②	5	30	②	6	31	②	5	32/37	5	33/38	②	6	32/37	7	33/38
MULB	2	②	4	21	②	4	21	②	4	23/28	4	24/29	②	5	23/28	6	24/29
MULB	3	②	5	22	②	5	22	②	5	24/29	5	25/30	②	6	24/29	7	25/30
DIVU	2	8C	3	25	8D	4	26	8E	3	27/32	3	28/33	8F	4	27/32	5	28/33
DIVUB	2	9C	3	17	9D	3	17	9E	3	19/24	3	20/25	9F	4	19/24	5	20/25
DIV	2	②	4	29	②	5	30	②	4	31/36	4	32/37	②	5	31/36	6	32/37
DIVB	2	②	4	21	②	4	21	②	4	23/28	4	24/29	②	5	23/28	6	24/29

Notes:

⊛ Long indexed and Indirect + instructions have identical opocodes with Short indexed and Indirect modes, respectively. The second byte of instructions using any indirect or indexed addressing mode specifies the exact mode used. If the second byte is even, use Indirect or Short Indexed. If it is odd, use Indirect + or Long indexed. In all cases the second byte of the instruction always specifies an even (word) location for the address referenced.

① Number of state times shown for internal/external operands.

② The opcodes for signed multiply and divide are the opcodes for the unsigned functions with an "FE" appended as a prefix.

Opcode and State Time Listing (Continued)

		DIRECT			IMMEDIATE			INDIRECT③					INDEXED③				
								NORMAL			AUTO-INC.		SHORT			LONG	
MNEMONIC	OPERANDS	OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES	OPCODE	BYTES	STATE① TIMES	BYTES	STATE① TIMES
LOGICAL INSTRUCTIONS																	
AND	2	60	3	4	61	4	5	62	3	6/11	3	7/12	63	4	6/11	5	7/12
AND	3	40	4	5	41	5	6	42	4	7/12	4	8/13	43	5	7/12	6	8/13
ANDB	2	70	3	4	71	3	4	72	3	6/11	3	7/12	73	4	6/11	5	7/12
ANDB	3	50	4	5	51	4	5	52	4	7/12	4	8/13	53	5	7/12	6	8/13
OR	2	80	3	4	81	4	5	82	3	6/11	3	7/12	83	4	6/11	5	7/12
ORB	2	90	3	4	91	3	4	92	3	6/11	3	7/12	93	4	6/11	5	7/12
XOR	2	84	3	4	85	4	5	86	3	6/11	3	7/12	87	4	6/11	5	7/12
XORB	2	94	3	4	95	3	4	96	3	6/11	3	7/12	97	4	6/11	5	7/12
DATA TRANSFER INSTRUCTIONS																	
LD	2	A0	3	4	A1	4	5	A2	3	6/11	3	7/12	A3	4	6/11	5	7/12
LDB	2	B0	3	4	B1	3	4	B2	3	6/11	3	7/12	B3	4	6/11	5	7/12
ST	2	C0	3	4	—	—	——	C2	3	7/13	3	8/14	C3	4	7/13	5	8/14
STB	2	C4	3	4	—	—	——	C6	3	7/13	3	8/14	C7	4	7/13	5	8/14
LDBSE	2	BC	3	4	BD	3	4	BE	3	6/11	3	7/12	BF	4	6/11	5	7/12
LDBZE	2	AC	3	4	AD	3	4	AE	3	6/11	3	7/12	AF	4	6/11	5	7/12
STACK OPERATIONS (internal stack)																	
PUSH	1	C8	2	8	C9	3	8	CA	2	11/15	2	12/16	CB	3	11/15	4	12/16
POP	1	CC	2	12	—	—	——	CE	2	14/18	2	14/18	CF	3	14/18	4	14/18
PUSHF	0	F2	1	8													
POPF	0	F3	1	9													
STACK OPERATIONS (external stack)																	
PUSH	1	C8	2	12	C9	3	12	CA	2	15/19	2	16/20	CB	3	15/19	4	16/20
POP	1	CC	2	14	—	—	——	CE	2	16/20	2	16/20	CF	3	16/20	4	16/20
PUSHF	0	F2	1	12													
POPF	0	F3	1	13													

JUMPS AND CALLS							
MNEMONIC	**OPCODE**	**BYTES**	**STATES**	**MNEMONIC**	**OPCODE**	**BYTES**	**STATES**
LJMP	E7	3	8	LCALL	EF	3	13/16⑤
SJMP	20-27④	2	8	SCALL	28-2F④	2	13/16⑤
INDJMP	E3	2	8	RET	F0	1	12/16⑤

Notes:

① Number of state times shown for internal/external operands.

③ This instruction is generated by the assembler when a branch indirect (BR [Rx]) instruction is reached.

④ The least significant 3 bits of the opcode are concatenated with the following 8 bits to form an 11-bit, 2's complement, offset for the relative call or jump.

⑤ State times for stack located internal/external.

CONDITIONAL JUMPS

All conditional jumps are 2 byte instructions. They require 8 state times if the jump is taken, 4 if it is not.

MNEMONIC	OPCODE	MNEMONIC	OPCODE	MNEMONIC	OPCODE	MNEMONIC	OPCODE
JC	DB	JE	DF	JGE	D6	JGT	D2
JNC	D3	JNE	D7	JLT	DE	JLE	DA
JH	D9	JV	DD	JVT	DC	JST	D8
JNH	D1	JNV	D5	JNVT	D4	JNST	D0

JUMP ON BIT CLEAR OR BIT SET

These instructions are 3-byte instructions. They require 9 state times if the jump is taken, 5 if it is not.

	BIT NUMBER							
MNEMONIC	0	1	2	3	4	5	6	7
JBC	30	31	32	33	34	35	36	37
JBS	38	39	3A	3B	3C	3D	3E	3F

LOOP CONTROL

DJNZ OPCODE EO; 3 BYTES; 5/9 STATE TIMES (NOT TAKEN/TAKEN)

SINGLE REGISTER INSTRUCTIONS

MNEMONIC	OPCODE	BYTES	STATES	MNEMONIC	OPCODE	BYTES	STATES
DEC	05	2	4	EXT	06	2	4
DECB	15	2	4	EXTB	16	2	4
NEG	03	2	4	NOT	02	2	4
NEGB	13	2	4	NOTB	12	2	4
INC	07	2	4	CLR	01	2	4
INCB	17	2	4	CLRB	11	2	4

SHIFT INSTRUCTIONS

INSTR	WORD		INSTR	BYTE		INSTR	DBL WD		
MNEMONIC	OP	B	MNEMONIC	OP	B	MNEMONIC	OP	B	STATE TIMES
SHL	09	3	SHLB	19	3	SHLL	0D	3	7 + 1 PER SHIFT⑦
SHR	08	3	SHRB	18	3	SHRL	0C	3	7 + 1 PER SHIFT⑦
SHRA	0A	3	SHRAB	1A	3	SHRAL	0E	3	7 + 1 PER SHIFT⑦

SPECIAL CONTROL INSTRUCTIONS

MNEMONIC	OPCODE	BYTES	STATES	MNEMONIC	OPCODE	BYTES	STATES
SETC	F9	1	4	DI	FA	1	4
CLRC	F8	1	4	EI	FB	1	4
CLRVT	FC	1	4	NOP	FD	1	4
RST	FF	1	16	SKIP	00	2	4

NORMALIZE

NORML	0F	3	11 + 1 PER SHIFT

Notes:

⑥ This instruction takes 2 states to pull RST low, then holds it low for 2 states to initiate a reset. The reset takes 12 states, at which time the program restarts at location 2080H.

⑦ Execution will take at least 8 states, even for 0 shift.

ELECTRICAL CHARACTERISTICS

ABSOLUTE MAXIMUM RATINGS

Ambient Temperature Under Bias	0°C to +70°C
Storage Temperature	-40°C to +150°C
Voltage from Any Pin to VSS or ANGND	-0.3V to +7.0V
Average Output Current from Any Pin	10 mA
Power Dissipation	1.5 Watts

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

OPERATING CONDITIONS

Symbol	Parameter	Min	Max	Units
TA	Ambient Temperature Under Bias	0	+70	C
VCC	Digital Supply Voltage	4.50	5.50	V
VREF	Analog Supply Voltage	4.5 VCC−0.3	5.5 VCC+0.3	V V
fOSC	Oscillator Frequency	6.0	12	MHz
VPD	Power-Down Supply Voltage	4.50	5.50	V

VBB should be connected to ANGND through a 0.01 μF capacitor. ANGND and VSS should be nominally at the same potential.

DC CHARACTERISTICS

Symbol	Parameter	Min	Max	Units	Test Conditions
VIL	Input Low Voltage	-0.3	+0.8	V	
VIH	Input High Voltage	2.0	VCC+0.5	V	
VOL	Output Low Voltage		0.45	V	See Note 1.
VOH	Output High Voltage	2.4		V	See Note 2.
ICC	VCC Supply Current		200	mA	All outputs disconnected.
IPD	VPD Supply Current		1	mA	Normal operation and Power-Down.
IREF	VREF Supply Current		15	mA	
ILI	Input Leakage Current to all pins of HSI, P0, P3, P4, and to P2.1, P2.2, P2.3, and P2.4		±10	μA	Vin = 0 to VCC See Note 3
IIH	Input High Current to $\overline{EA}$		100	μA	VIH = 2.4V
IIL	Input Low Current to all pins of P1, and to P2.6, P2.7, and READY		-100	μA	VIL = 0.45V
IIL1	Input Low Current to $\overline{RESET}$		-2	mA	VIL = 0.45V
Cs	Pin Capacitance (Any Pin to VSS)		10	pF	fTEST = 1MHz

NOTES:

1. IOL = 0.36 mA for all pins of P1, for P2.6 and P2.7, and for all pins of P3 and P4 when used as ports.
   IOL = 2.0 mA for TXD, RXD (in serial port mode 0), PWM, CLKOUT, ALE, $\overline{BHE}$, $\overline{RD}$, $\overline{WR}$, and all pins of HSO and P3 and P4 when used as external memory bus (AD0-AD15).

2. IOH = −20 μA for all pins of P1, for P2.6 ad P2.7.
   IOH = −200 μA for TXD, RXD (in serial port mode 0), PWM, CLKOUT, ALE, $\overline{BHE}$, $\overline{WR}$, and all pins of HSO and P3 and P4 when used as external memory bus (AD0-AD15).
   P3 and P4, when used as ports, have open-drain outputs.

3. Analog Conversion not in process.

A/D CONVERTER SPECIFICATIONS

A/D Converter operation is verified only on the 8097, 8397, 8095, 8395.

The absolute conversion accuracy is dependent on the accuracy of VREF. The specifications given below assume adherence to the Operating Conditions section of these data sheets. Testing is done at VREF = 5.120 volts.

Resolution	±0.001 VREF
Accuracy	±0.004 VREF
Differential nonlinearity	±0.002 VREF max
Integral nonlinearity	±0.004 VREF max
Channel-to-chanel matching	±1 LSB
Crosstalk (DC to 100kHz)	−60dB max

AC CHARACTERISTICS

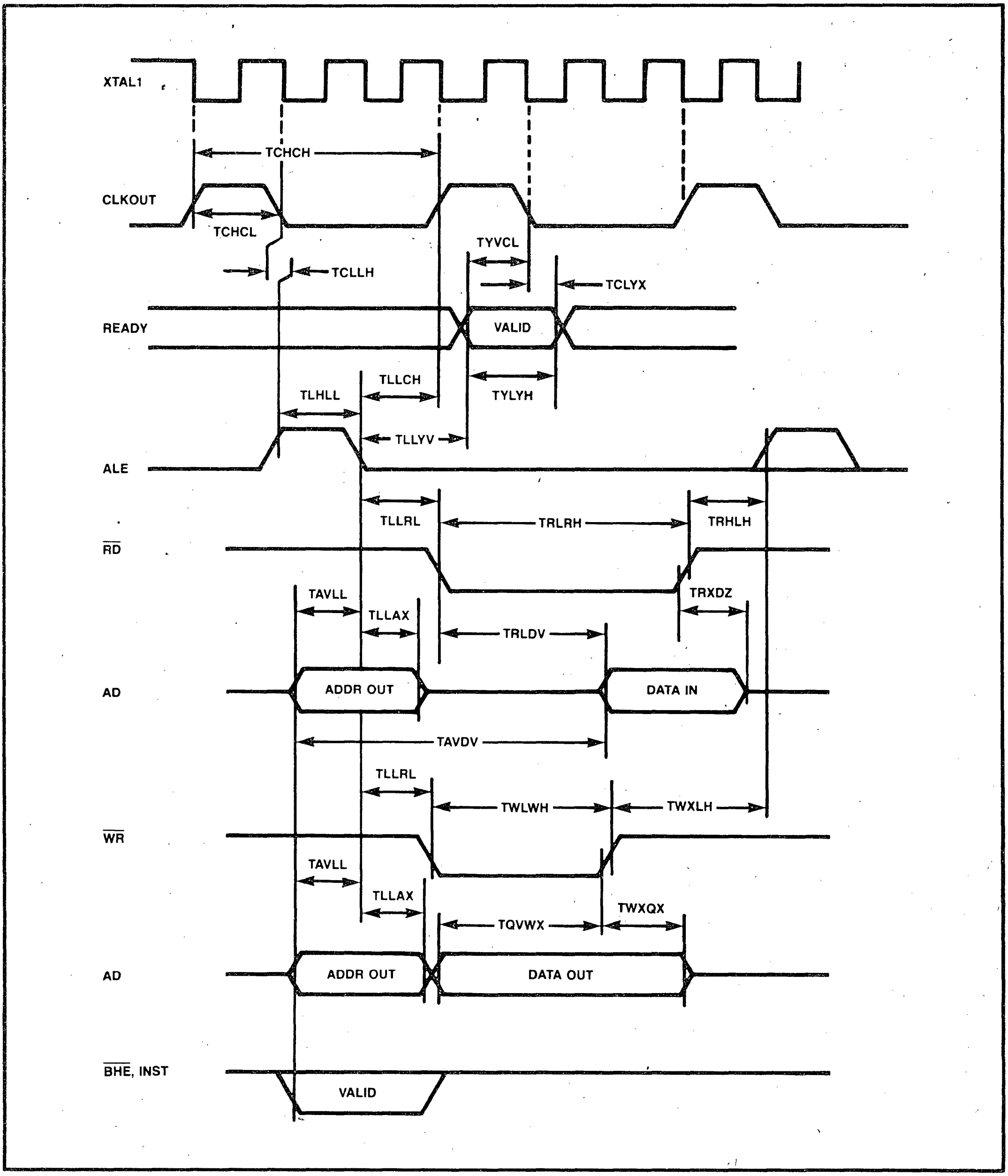
Test Conditions; Oscillator Frequency = 12MHz
Load Capacitance on Output Pins = 80pF
Other Operating Conditions as previously described.

Timing Requirements (Other system components must meet these specs.)

Symbol	Parameter	Min	Max	Units
TYVCL	READY Setup to CLKOUT Edge	50		nsec
TCLYX	READY Hold after CLKOUT Edge	0		nsec
TLLYV	End of ALE to READY Setup		2Tosc-50	nsec
TYLYH	Non-ready Time		1000	nsec
TAVDV	Address Valid to Input Data Valid		5Tosc-70	nsec
TRLDV	$\overline{RD}$ Active to Input Data Valid		3Tosc-50	nsec
TRXDZ	End of $\overline{RD}$ to Input Data Float	0	Tosc-20	nsec

Timing Responses (MCS-96 parts meet these specs.)

Symbol	Parameter	Min	Max	Units
FXTAL	Oscillator Frequency	6.0	12.0	MHz
TOSC	1/Oscillator Frequency	160	83	nsec
TCHCH	CLKOUT Period	3Tosc	3Tosc	nsec
TCHCL	CLKOUT High Time	Tosc-20	Tosc +10	nsec
TCLLH	CLKOUT Low to ALE High	-5	+20	nsec
TLLCH	ALE Low to CLKOUT High	Tosc-20	Tosc +20	nsec
TLHLL	ALE Pulse Width	Tosc-10	Tosc	nsec
TAVLL	Address Valid to End of ALE	Tosc-40	Tosc-15	nsec
TLLRL	End of ALE to $\overline{RD}$ or $\overline{WR}$ Active	Tosc-20		
TLLAX	End of ALE to Address Invalid	Tosc-20		nsec
TWLWH	$\overline{WR}$ Pulse Width	2Tosc-15		nsec
TQVWX	Output Data Valid to End of $\overline{WR}$	2Tosc-30		nsec
TWXQX	Output Data Hold after $\overline{WR}$	Tosc-25		nsec
TWXLH	End of $\overline{WR}$ to Next ALE	2Tosc-30		nsec
TRLRH	$\overline{RD}$ Pulse Width	3Tosc-30		nsec
TRHLH	End of $\overline{RD}$ to Next ALE	Tosc-25		nsec


XTAL1
TCHCH
CLKOUT
TCHCL
TYVCL
TCLLH
TCLYX
READY
VALID
TLLCH
TLHLL
TYLYH
TLLYV
ALE
TLLRL
TRLRH
TRHLH
RD
TAVLL
TRXDZ
TLLAX
TRLDV
AD
ADDR OUT
DATA IN
TAVDV
TLLRL
TWLWH
TWXLH
WR
TAVLL
TLLAX
TWXQX
TQVWX
AD
ADDR OUT
DATA OUT
BHE, INST
VALID

MCS®-51 Architecture

6

CHAPTER 6
MCS®-51 ARCHITECTURE

6.0 INTRODUCTION

The MCS®-51 family of 8-bit microcontrollers consists of the devices listed in Table 1, all of which are based on the MCS-51 architecture shown in Figure 6-1. The original 8051 was built in HMOS I technology. The HMOS II version, which is the device currently in production, is called the 8051AH. The term "8051," however, is still often used to generically refer to all of the MCS-51 family members. This is the case throughout this manual, except where specifically stated otherwise. Also for brevity, the term "8052" is used to refer to both the 8052 and the 8032, unless otherwise noted.

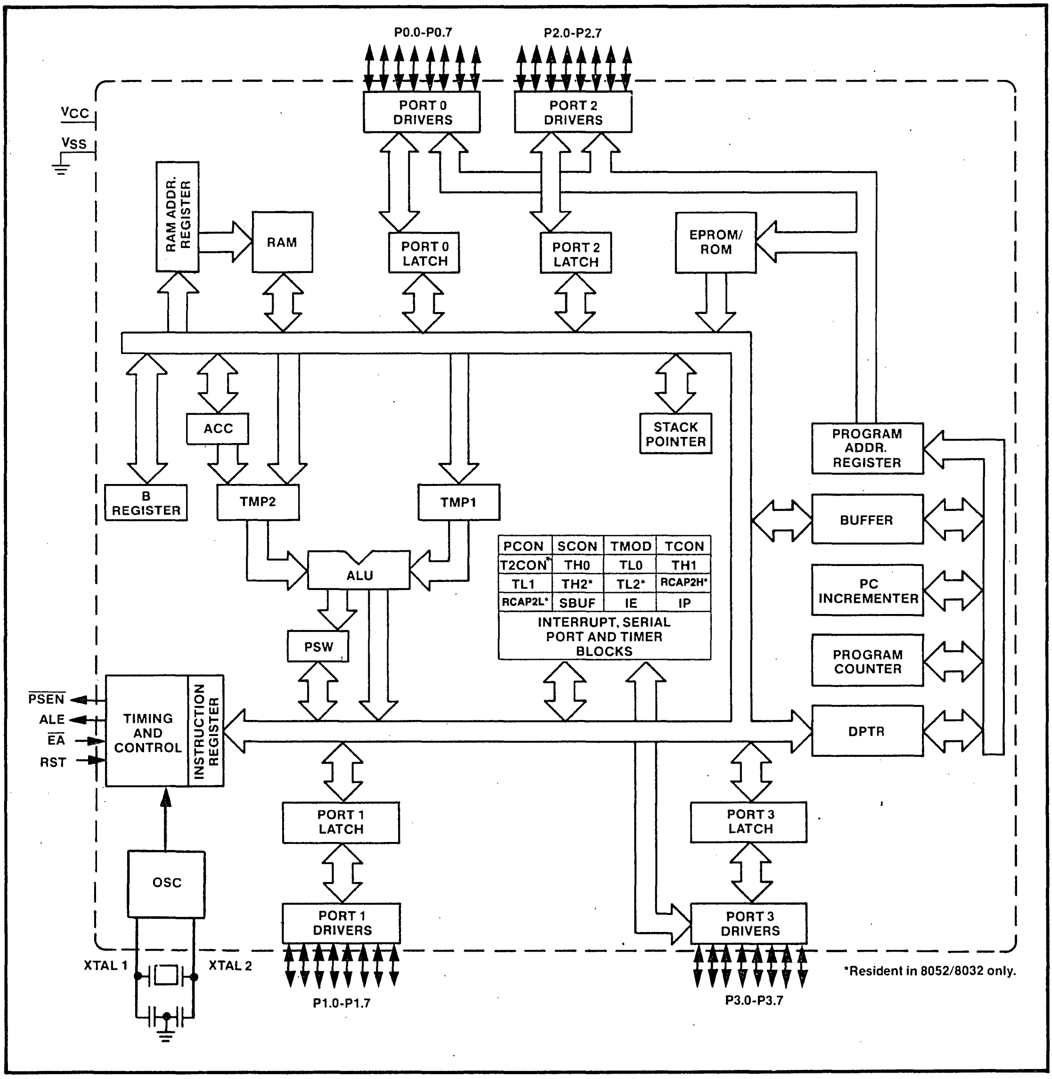


Figure 6-1. MCS-51 Architectural Block Diagram

The newest MCS-51 members, the 8032 and 8052, have more on-chip memory and an additional 16-bit timer/counter. The new timer can be used as a timer, a counter, or to generate baud rates for the serial port. As a timer/counter, it operates in either a 16-bit auto-reload mode or a 16-bit "capture" mode. This new feature is described in Section 6.6.2.

Pinouts are shown in the individual data sheets and on the inside back cover of this handbook.

Table 1. MCS®-51 Family Members

PART	TECHNOLOGY	ON-CHIP PROGRAM MEMORY	ON-CHIP DATA MEMORY
8051AH	HMOS II	4K — ROM	128
8031AH	HMOS II	NONE	128
8751H	HMOS I	4K — EPROM	128
80C51	CHMOS	4K — ROM	128
80C31	CHMOS	NONE	128
8052	HMOS II	8K — ROM	256
8032	HMOS II	NONE	256

The major MCS®-51 features are:

- 8-Bit CPU
- On-Chip oscillator and clock circuitry
- 32 I/O lines
- 64K address space for external data memory
- 64K address space for external program memory
- Two 16-bit timer/counters (three on 8032/8052)
- A five-source interrupt structure (six sources on 8032/8052) with two priority levels
- Full duplex serial port
- Boolean processor

6.1 MEMORY ORGANIZATION

The 8051 has separate address spaces for Program Memory and Data Memory. The Program Memory can be up to 64K bytes long. The lower 4K (8K for 8052) may reside on-chip. The Data Memory can consist of up to 64K bytes of off-chip RAM, in addition to which, it includes 128 bytes of on-chip RAM (256 bytes for the 8052), plus a number of "SFRs" (Special Function Registers) as listed below.

Symbol	Name	Address
*ACC	Accumulator	0E0H
*B	B Register	0F0H
*PSW	Program Status Word	0D0H
SP	Stack Pointer	81H
DPTR	Data Pointer (consisting of DPH and DPL	83H 82H
*P0	Port 0	80H
*P1	Port 1	90H
*P2	Port 2	0A0H
*P3	Port 3	0B0H
*IP	Interrupt Priority Control	0B8H
*IE	Interrupt Enable Control	0A8H
TMOD	Timer/Counter Mode Control	89H
+*T2CON	Timer/Counter Control	88H
TCON	Timer/Counter 2 Control	0C8H
TH0	Timer/Counter 0 (high byte)	8CH
TL0	Timer/Counter 0 (low byte)	8AH
TH1	Timer/Counter 1 (high byte)	8DH
TL1	Timer/Counter 1 (low byte)	8BH
+TH2	Timer/Counter 2 (high byte)	0CDH
+TL2	Timer/Counter 2 (low byte)	0CCH
+RCAP2H	Timer/Counter 2 Capture Register (high byte)	0CBH
+RCAP2L	Timer/Counter 2 Capture Register (low byte)	0CAH
*SCON	Serial Control	98H
SBUF	Serial Data Buff	99H
PCON	Power Control	97H

The SFRs marked with an asterisk (*) are both bit- and byte-addressable. The SFRs marked with a plus sign (+) are present in the 8052 only. The functions of the SFRs are described as follows.

ACCUMULATOR

ACC is the Accumulator register. The mnemonics for accumulator-specific instructions, however, refer to the accumulator simply as A.

B REGISTER

The B register is used during multiply and divide operations. For other instructions it can be treated as another scratch pad register.

PROGRAM STATUS WORD

The PSW register contains program status information as detailed in Figure 6-2.

STACK POINTER

The Stack Pointer register is 8 bits wide. It is incremented before data is stored during PUSH and CALL executions. While the stack may reside anywhere in on-chip RAM, the Stack Pointer is initialized to 07H after a reset. This causes the stack to begin at location 08H.

DATA POINTER

The Data Pointer (DPTR) consists of a high byte (DPH) and a low byte (DPL). Its intended function is to hold a 16-bit address. It may be manipulated as a 16-bit register or as two independent 8-bit registers.

PORTS 0 to 3

P0, P1, P2 and P3 are the SFR latches of Ports 0, 1, 2 and 3, respectively.

SERIAL DATA BUFFER

The Serial Data Buffer is actually two separate registers, a transmit buffer and a receive buffer register. When data is moved to SBUF, it goes to the transmit buffer where it is held for serial transmission. (Moving a byte to SBUF is what initiates the transmission.) When data is moved from SBUF, it comes from the receive buffer.

TIMER REGISTERS

Register pairs (TH0, TL0), (TH1, TL1), and (TH2, TL2) are the 16-bit counting registers for Timer/Counters 0, 1, and 2, respectively.

CAPTURE REGISTERS

The register pair (RCAP2H, RCAP2L) are the capture registers for the Timer 2 ''capture mode.'' In this mode, in response to a transition at the 8052's T2EX pin, TH2 and TL2 are copied into RCAP2H and RCAP2L. Timer 2 also has a 16-bit auto-reload mode, and RCAP2H and RCAP2L hold the reload value for this mode. More about Timer 2's features in Section 6.6.2.

CONTROL REGISTERS

Special Function Registers IP, IE, TMOD, TCON, T2CON, SCON, and PCON contain control and status bits for the interrupt system, the timer/counters, and the serial port. They are described in later sections.

6.2 OSCILLATOR AND CLOCK CIRCUIT

XTAL1 and XTAL2 are the input and output of a single-stage on-chip inverter, which can be configured with off-chip components as a Pierce oscillator, as shown in Figure 6-3. The on-chip circuitry, and selection of off-chip components to configure the oscillator are discussed in Section

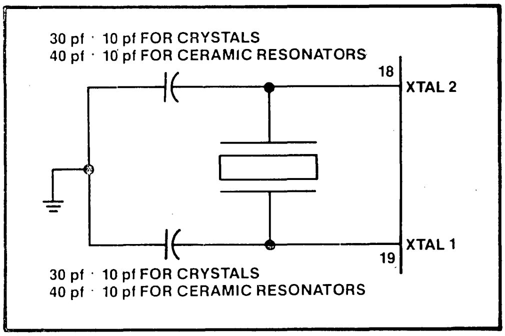

30 pf ± 10 pf FOR CRYSTALS
40 pf ± 10 pf FOR CERAMIC RESONATORS
18
XTAL 2
XTAL 1
19
30 pf ± 10 pf FOR CRYSTALS
40 pf ± 10 pf FOR CERAMIC RESONATORS

Figure 6-3. Crystal/Ceramic Resonator Oscillator

(MSB)							(LSB)
CY	AC	F0	RS1	RS0	OV	—	P

Symbol	Position	Name and Significance
CY	PSW.7	Carry flag.
AC	PSW.6	Auxiliary Carry flag. (For BCD operations.)
F0	PSW.5	Flag 0 (Available to the user for general purposes.)
RS1	PSW.4	Register bank Select control bits 1 & 0. Set/cleared by software to determine working register bank (see Note).
RS0	PSW.3	

Symbol	Position	Name and Significance
OV	PSW.2	Overflow flag.
—	PSW.1	(reserved)
P	PSW.0	Parity flag. Set/cleared by hardware each instruction cycle to indicate an odd/even number of "one" bits in the accumulator, i.e., even parity.

Note— the contents of (RS1, RS0) enable the working register banks as follows:

(0.0)—Bank 0	(00H-07H)
(0.1)—Bank 1	(08H-0FH)
(1.0)—Bank 2	(10H-17H)
(1.1)—Bank 3	(18H-1FH)

Figure 6-2. PSW: Program Status Word Register

6.13. A more detailed discussion will be found in Application Note AP-155, "Oscillators for Microcontrollers," which is included in this manual.

The oscillator, in any case, drives the internal clock generator. The clock generator provides the internal clocking signals to the chip. The internal clocking signals are at half the oscillator frequency, and define the internal phases, states, and machine cycles, which are described in the next section.

6.3 CPU TIMING

A machine cycle consists of 6 states (12 oscillator periods). Each state is divided into a Phase 1 half, during which the Phase 1 clock is active, and a Phase 2 half,

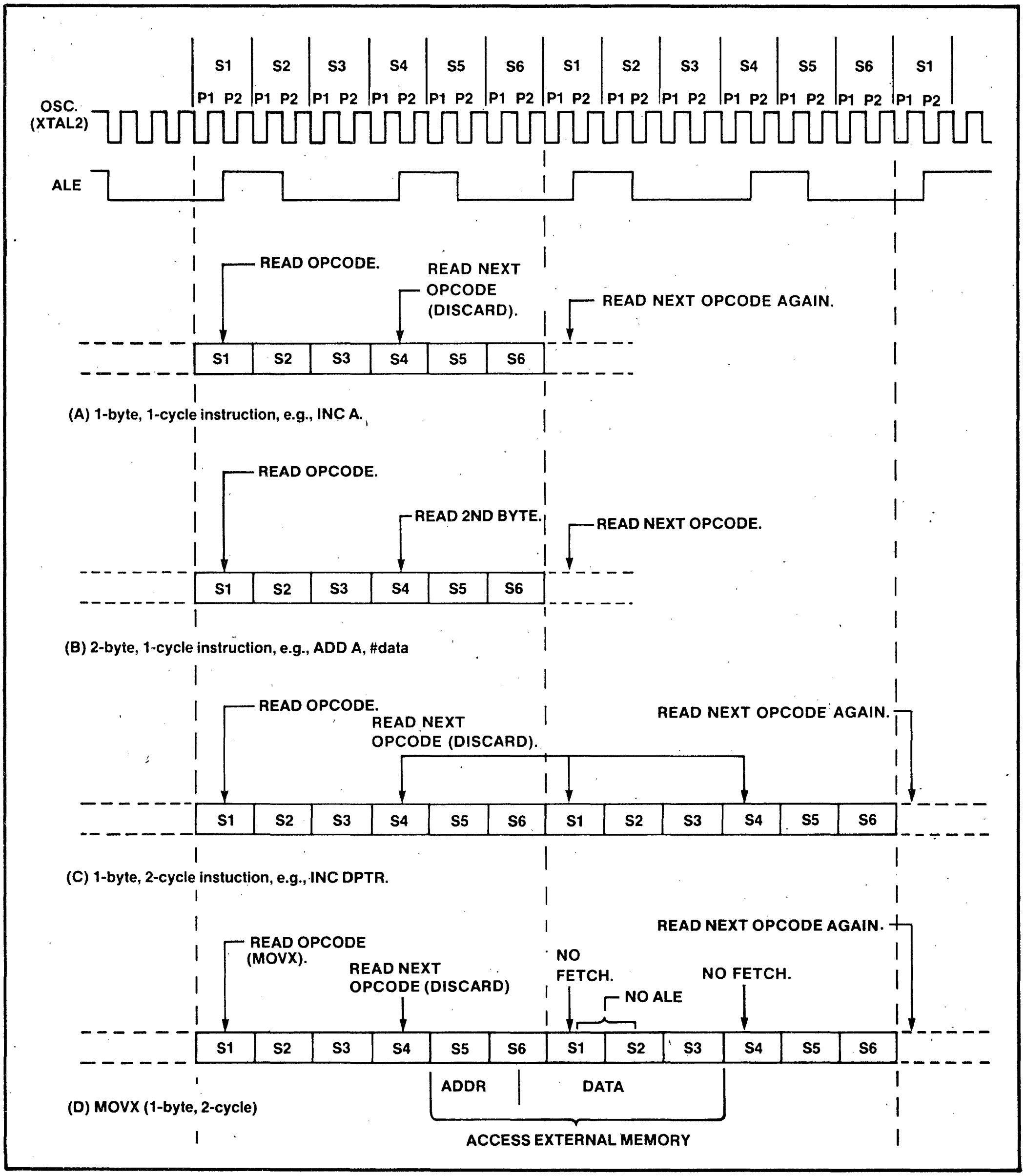

S1
S2
S3
S4
S5
S6
P1 P2
OSC.
(XTAL2)
ALE
READ OPCODE.
READ NEXT OPCODE (DISCARD).
READ NEXT OPCODE AGAIN.
(A) 1-byte, 1-cycle instruction, e.g., INC A.
READ OPCODE.
READ 2ND BYTE.
READ NEXT OPCODE.
(B) 2-byte, 1-cycle instruction, e.g., ADD A, #data
READ OPCODE.
READ NEXT OPCODE (DISCARD).
READ NEXT OPCODE AGAIN.
(C) 1-byte, 2-cycle instuction, e.g., INC DPTR.
READ OPCODE (MOVX).
READ NEXT OPCODE (DISCARD)
NO FETCH.
NO ALE
NO FETCH.
READ NEXT OPCODE AGAIN.
ADDR
DATA
ACCESS EXTERNAL MEMORY
(D) MOVX (1-byte, 2-cycle)

Figure 6-4. 8051 Fetch/Execute Sequences during which the Phase 2 clock is active. Thus, a machine cycle consists of 12 oscillator periods, numbered S1P1 (State 1, Phase 1), through S6P2 (State 6, Phase 2). Each phase lasts for one oscillator period. Each state lasts for two oscillator periods. Typically, arithmetic and logical operations take place during Phase 1 and internal register-to-register transfers take place during Phase 2.

The diagrams in Figure 6-4 show the fetch/execute timing referenced to the internal states and phases. Since these internal clock signals are not user accessible, the XTAL2 oscillator signal and the ALE (Address Latch Enable) signal are shown for external reference. ALE is normally activated twice during each machine cycle: once during S1P2 and S2P1, and again during S4P2 and S5P1.

Execution of a one-cycle instruction begins at S1P2, when the opcode is latched into the Instruction Register. If it is a two-byte instruction, the second byte is read during S4 of the same machine cycle. If it is a one-byte instruction, there is still a fetch at S4, but the byte read (which would be the next opcode), is ignored, and the Program Counter is not incremented. In any case, execution is complete at the end of S6P2. Figures 6-4A and 6-4B show the timing for a 1-byte, 1-cycle instruction and for a 2-byte, 1-cycle instruction.

Most 8051 instructions execute in one cycle. MUL (multiply) and DIV (divide) are the only instructions that take more than two cycles to complete. They take four cycles.

Normally, two code bytes are fetched from Program Memory during every machine cycle. The only exception to this is when a MOVX instruction is executed. MOVX is a 1-byte 2-cycle instruction that accesses external Data Memory. During a MOVX, two fetches are skipped while the external Data Memory is being addressed and strobed. Figures 6-4C and 6-4D show the timing for a normal 1-byte, 2-cycle instruction and for a MOVX instruction.

6.4 PORT STRUCTURES AND OPERATION

All four ports in the 8051 are bidirectional. Each consists of a latch (Special Function Registers P0 through P3), an output driver, and an input buffer.

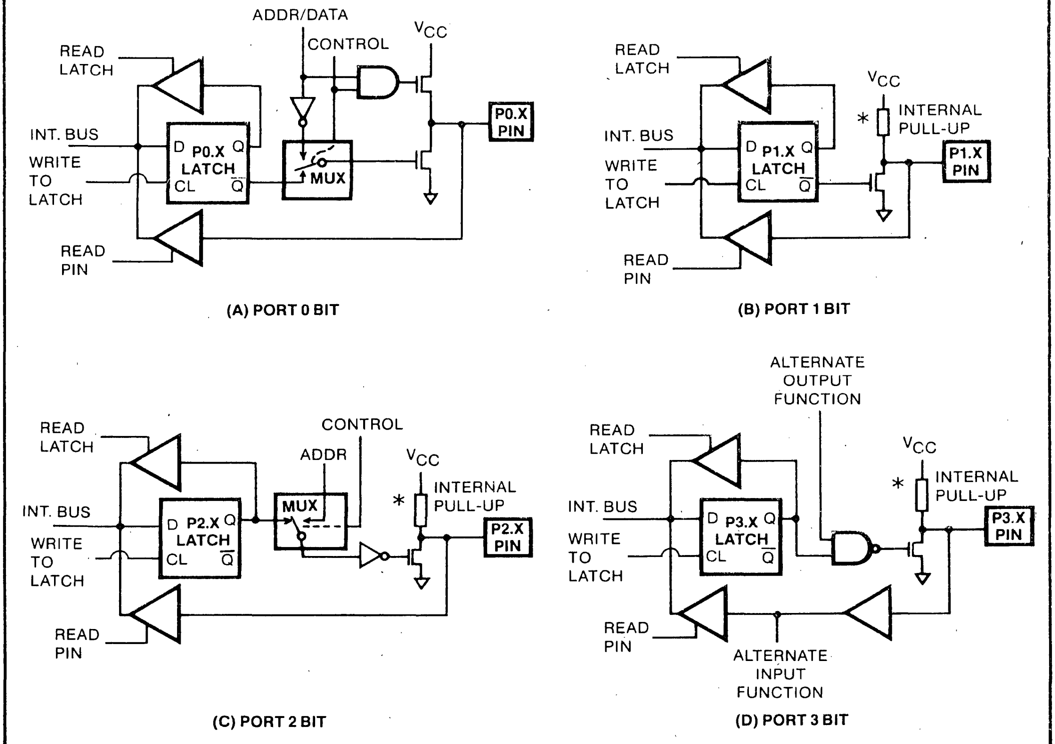


Figure 6-5. 8051 Port Bit Latches and I/O Buffers

* See Figure 6-6 for details of the internal pullup.

The output drivers of Ports 0 and 2, and the input buffers of Port 0, are used in accesses to external memory. In this application, Port 0 outputs the low byte of the external memory address, time-multiplexed with the byte being written or read. Port 2 outputs the high byte of the external memory address when the address is 16 bits wide. Otherwise the Port 2 pins continue to emit the P2 SFR content.

All the Port 3 pins, and (in the 8052) two Port 1 pins are multifunctional. They are not only port pins, but also serve the functions of various special features as listed below:

PORT PIN	**ALTERNATE FUNCTION**
***P1.0**	**T2 (Timer/Counter 2 external input)**
***P1.1**	**T2EX (Timer/Counter 2 capture/reload trigger)**
P3.0	**RXD (serial input port)**
P3.1	**TXD (serial output port)**
P3.2	**$\overline{\text{INT0}}$ (external interrupt)**
P3.3	**$\overline{\text{INT1}}$ (external interrupt)**
P3.4	**T0 (Timer/Counter 0 external input)**
P3.5	**T1 (Timer/Counter 1 external input)**
P3.6	**$\overline{\text{WR}}$ (external Data memory write strobe)**
P3.7	**$\overline{\text{RD}}$ (external Data memory read strobe)**

*P1.0 and P1.1 serve these alternate functions only on the 8052.

The alternate functions can only be activated if the corresponding bit latch in the port SFR contains a 1. Otherwise the port pin is stuck at 0.

6.4.1 I/O Configurations

Figure 6-5 shows a functional diagram of a typical bit latch and I/O buffer in each of the four ports. The bit latch (one bit in the port's SFR) is represented as a Type D flip-flop, which will clock in a value from the internal bus in response to a "write to latch" signal from the CPU. The Q output of the flip-flop is placed on the internal bus in response to a "read latch" signal from the CPU. The level of the port pin itself is placed on the internal bus in response to a "read pin" signal from the CPU. Some instructions that read a port activate the "read latch" signal, and others activate the "read pin" signal. More about that in Section 6.4.4.

As shown in Figure 6-5, the output drivers of Ports 0 and 2 are switchable to an internal ADDR and ADDR/DATA bus by an internal CONTROL signal for use in external memory accesses. During external memory accesses, the P2 SFR remains unchanged, but the P0 SFR gets 1s written to it.

Also shown in Figure 6-5, is that if a P3 bit latch contains a 1, then the output level is controlled by the signal labeled "alternate output function." The actual P3.X pin level is always available to the pin's alternate input function, if any.

Ports 1, 2, and 3 have internal pull-ups. Port 0 has open-drain outputs. Each I/O line can be independently used as an input or an output. (Ports 0 and 2 may not be used as general purpose I/O when being used as the ADDR/DATA BUS). To be used as an input, the port bit latch must contain a 1, which turns off the output driver FET. Then, for Ports 1, 2, and 3, the pin is pulled high by the internal pull-up, but can be pulled low by an external source.

Port 0 differs in not having internal pullups. The pullup FET in the P0 output driver (see Figure 6-5A) is used only when the Port is emitting 1s during external memory accesses. Otherwise the pullup FET is off. Consequently P0 lines that are being used as output port lines are open drain. Writing a 1 to the bit latch leaves both output FETs off, so the pin floats. In that condition it can be used as a high-impedance input.

Because Ports 1, 2, and 3 have fixed internal pullups they are sometimes called "quasi-bidirectional" ports. When configured as inputs they pull high and will source current (IIL, in the data sheets) when externally pulled low. Port 0, on the other hand, is considered "true" bidirectional, because when configured as an input it floats.

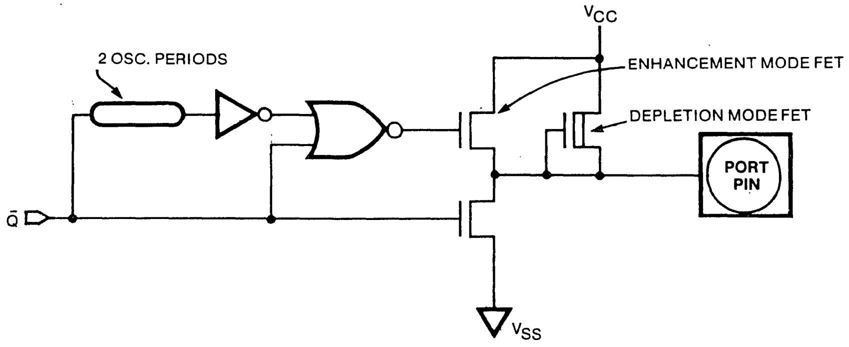
All the port latches in the 8051 have 1s written to them by the reset function. If a 0 is subsequently written to a port latch, it can be reconfigured as an input by writing a 1 to it.

6.4.2 Writing to a Port

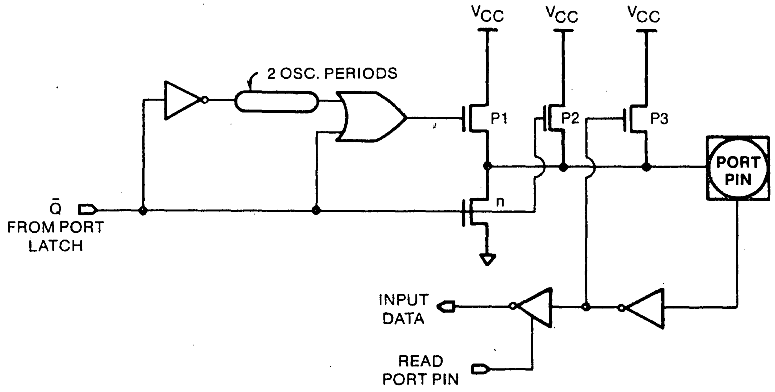
In the execution of an instruction that changes the value in a port latch, the new value arrives at the latch during S6P2 of the final cycle of the instruction. However, port latches are in fact sampled by their output buffers only during Phase 1 of any clock period. (During Phase 2 the output buffer holds the value it saw during the previous Phase 1). Consequently, the new value in the port latch won't actually appear at the output pin until the next Phase 1, which will be at S1P1 of the next machine cycle.

If the change requires a 0-to-1 transition in Port 1, 2, or 3, an additional pull-up is turned on during S1P1 and S1P2 of the cycle in which the transition occurs. This is done to increase the transition speed. The extra pull-up can source about 100 times the current that the normal pull-up can. It should be noted that the internal pull-ups are field-effect transistors, not linear resistors. The pull-up arrangements are shown in Figure 6-6.

In HMOS versions of the 8051, the fixed part of the pull-up is a depletion-mode transistor with the gate wired to the source. This transistor will allow the pin to source



A. HMOS Configuration. The enhancement mode transistor is turned on for 2 osc. periods after Q makes a 1-to-0 transition.



B. CHMOS Configuration. pFET 1 is turned on for 2 osc. periods after Q makes a 1-to-0 transition. During this time, pFET 1 also turns on pFET 3 through the inverter to form a latch which holds the 1. pFET 2 is also on.

Figure 6-6. Ports 1, 2 and 3 HMOS and CHMOS Internal Pull-up Configurations

about 0.25 mA when shorted to ground. In parallel with the fixed pull-up is an enhancement-mode transistor, which is activated during S1 whenever the port bit does a 0-to-1 transition. During this interval, if the port pin is shorted to ground, this extra transistor will allow the pin to source an additional 30 mA.

In the CHMOS versions, the pull-up consists of three pFETs. It should be noted that an n-channel FET (nFET) is turned on when a logical 1 is applied to its gate, and is turned off when a logical 0 is applied to its gate. A p-channel FET (pFET) is the opposite: it is on when its gate sees a 0, and off when its gate sees a 1.

pFET 1 in Figure 6-6B is the transistor that is turned on for 2 oscillator periods after a 0-to-1 transition in the port latch. While it's on, it turns on pFET 3 (a weak pull-up), through the inverter. This inverter and pFET form a latch which hold the 1.

Note that if the pin is emitting a 1, a negative glitch on the pin from some external source can turn off pFET 3, causing the pin to go into a float state, pFET 2 is a very weak pull-up which is on whenever the nFET is off, in traditional CMOS style. It's only about 1/10 the strength of pFET3. Its function is to restore a 1 to the pin in the event the pin *had* a 1 and lost it to a glitch.

6.4.3 Port Loading and Interfacing

The output buffers of Ports 1, 2, and 3 can each drive 4 LS TTL inputs. These ports on HMOS versions can be driven in a normal manner by any TTL or NMOS circuit. Both HMOS and CHMOS versions can be driven by open-collector and open-drain outputs, without the need for external pull-ups.

Port 0 output buffers can each drive 8 LS TTL inputs. They do, however, require external pull-ups to drive NMOS inputs, except when being used as the ADDRESS/DATA bus.

6.4.4 Read-Modify-Write Feature

Some instructions that read a port read the latch and others read the pin. Which ones do which? The instructions that read the latch rather than the pin are the ones that read a value, possibly change it, and then rewrite it to the latch. These are called "read-modify-write" instructions. The instructions listed below are read-modify-write instructions. When the destination operand is a port, or a port bit, these instructions read the latch rather than the pin:

ANL	**(logical AND, e.g., ANL P1,A)**
ORL	**(logical OR, e.g., ORL P2,A)**
XRL	**(logical EX-OR, e.g., XRL P3,A)**
JBC	**(jump if bit = 1 and clear bit, e.g., JBC P1.1, LABEL)**
CPL	**(complement bit, e.g., CPL P3.0)**
INC	**(increment, e.g., INC P2)**
DEC	**(decrement, e.g., DEC P2)**
DJNZ	**(decrement and jump if not zero, e.g., DJNZ P3, LABEL)**
MOV PX.Y,C	**(move carry bit to bit Y of Port X)**
CLR PX.Y	**(clear bit Y of Port X)**
SET PX.Y	**(set bit Y of Port X)**

It is not obvious that the last three instructions in this list are read-modify-write instructions, but they are. They read the port byte, all 8 bits, modify the addressed bit, then write the new byte back to the latch.

The reason that read-modify-write instructions are directed to the latch rather than the pin is to avoid a possible misinterpretation of the voltage level at the pin. For example, a port bit might be used to drive the base of a transistor. When a 1 is written to the bit, the transistor is turned on. If the CPU then reads the same port bit at the pin rather than the latch, it will read the base voltage of the transistor and interpret it as a 0. Reading the latch rather than the pin will return the correct value of 1.

6.5 ACCESSING EXTERNAL MEMORY

Accesses to external memory are of two types: accesses to external Program Memory and accesses to external Data Memory. Accesses to external Program Memory use signal $\overline{\text{PSEN}}$ (program store enable) as the read strobe. Accesses to external Data Memory use $\overline{\text{RD}}$ or $\overline{\text{WR}}$ (alternate functions of P3.7 and P3.6) to strobe the memory.

Fetches from external Program Memory always use a 16-bit address. Accesses to external Data Memory can use either a 16-bit address (MOVX @DPTR) or an 8-bit address (MOVX @Ri).

Whenever a 16-bit address is used, the high byte of the address comes out on Port 2, where it is held for the duration of the read or write cycle. During this time the Port 2 latch (the Special Function Register) does not have to contain 1s, and the contents of the Port 2 SFR are not modified. If the external memory cycle is not immediately followed by another external memory cycle, the undisturbed contents of the Port 2 SFR will reappear in the next cycle.

If an 8-bit address is being used (MOVX @Ri), the contents of the Port 2 SFR remain at the Port 2 pins throughout the external memory cycle. This will facilitate paging.

In any case, the low byte of the address is time-multiplexed with the data byte on Port 0. The ADDR/DATA signal drives both FETs in the Port 0 output buffers. Thus, in this application the Port 0 pins are not open-drain outputs, and do not require external pull-ups. Signal ALE (address latch enable) should be used to capture the address byte into an external latch. The address byte is valid at the negative transition of ALE. Then, in a write cycle, the data byte to be written appears on Port 0 just before $\overline{\text{WR}}$ is activated, and remains there until after $\overline{\text{WR}}$ is deactivated. In a read cycle, the incoming byte is accepted at Port 0 just before the read strobe is deactivated.

During any access to external memory, the CPU writes 0FFH to the Port 0 latch (the Special Function Register), thus obliterating whatever information the Port 0 SFR may have been holding.

External Program Memory is accessed under two conditions:

1) Whenever signal $\overline{\text{EA}}$ is active; or
2) Whenever the program counter (PC) contains a number that is larger than 0FFFH (1FFFH for the 8052).

This requires that the ROMless versions have $\overline{\text{EA}}$ wired low to enable the lower 4K (8K for the 8032) program bytes to be fetched from external memory.

When the CPU is executing out of external Program Memory, all 8 bits of Port 2 are dedicated to an output function and may not be used for general purpose I/O. During external program fetches they output the high byte of the PC, and during accesses to external Data Memory they output either DPH or the Port 2 SFR (depending on whether the external Data Memory access is a MOVX @DPTR or a MOVX @Ri).

6.5.1 PSEN

The read strobe for external fetches is PSEN. PSEN is not activated for internal fetches. When the CPU is accessing external Program Memory, PSEN is activated twice every cycle (except during a MOVX instruction) whether or not the byte fetched is actually needed for the current instruction. When PSEN is activated its timing is not the same as RD. A complete RD cycle, including activation and deactivation of ALE and RD, takes 12 oscillator periods. A complete PSEN cycle, including activation and deactivation of ALE and PSEN, takes 6 oscillator periods. The execution sequence for these two types of read cycles are shown in Figure 6-7 for comparison.

6.5.2 ALE

The main function of ALE is to provide a properly timed signal to latch the low byte of an address from P0 to an external latch during fetches from external Program Memory. For that purpose ALE is activatd twice every machine cycle. This activation takes place even when the cycle involves no external fetch. The only time an ALE pulse doesn't come out is during an access to external Data Memory. The first ALE of the second cycle of a MOVX instruction is missing (see Figure 6-7). Consequently, in any system that does not use external Data Memory, ALE is activated at a constant rate of 1/6 the oscillator frequency, and can be used for external clocking or timing purposes.

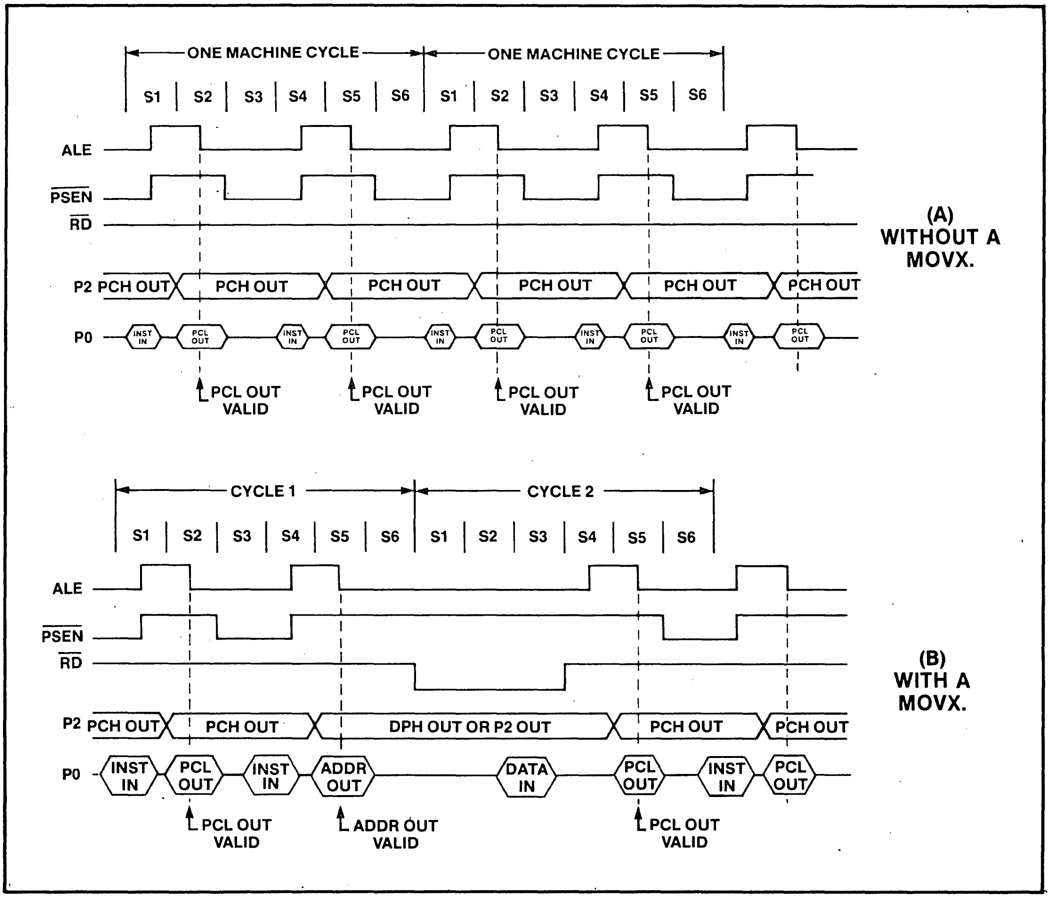


Figure 6-7. External Program Memory Execution

6.5.3 Overlapping External Program and Data Memory Spaces

In some applications it is desirable to execute a program from the same physical memory that is being used to store data. In the 8051, the external Program and Data Memory spaces can be combined by ANDing $\overline{\text{PSEN}}$ and $\overline{\text{RD}}$. A positive-logic AND of these two signals produces an active-low read strobe that can be used for the combined physical memory. Since the $\overline{\text{PSEN}}$ cycle is faster than the $\overline{\text{RD}}$ cycle, the external memory needs to be fast enough to accommodate the $\overline{\text{PSEN}}$ cycle.

6.6 TIMER/COUNTERS

The 8051 has two 16-bit timer/counter registers: Timer 0 and Timer 1. The 8052 has these two plus one more: Timer 2. All three can be configured to operate either as timers or event counters.

In the "timer" function, the register is incremented every machine cycle. Thus, one can think of it as counting machine cycles. Since a machine cycle consists of 12 oscillator periods, the count rate is 1/12 of the oscillator frequency.

In the "counter" function, the register is incremented in response to a 1-to-0 transition at its corresponding external input pin, T0, T1 or (in the 8052) T2. In this function, the external input is sampled during S5P2 of every machine cycle. When the samples show a high in one cycle and a low in the next cycle, the count is incremented. The new count value appears in the register during S3P1 of the cycle following the one in which the transition was detected. Since it takes 2 machine cycles (24 oscillator periods) to recognize a 1-to-0 transition, the maximum count rate is 1/24 of the oscillator frequency. There are no restrictions on the duty cycle of the external input signal, but to ensure that a given level is sampled at least once before it changes, it should be held for at least one full machine cycle.

In addition to the "timer" or "counter" selection, Timer 0 and Timer 1 have four operating modes from which to select. Timer 2, in the 8052, has three modes of operation: "capture," "auto-reload" and "baud rate generator."

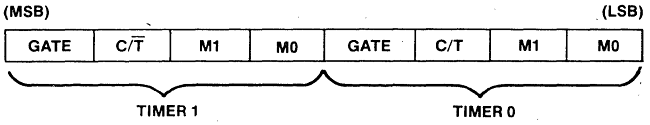
6.6.1 Timer 0 and Timer 1

These timer/counters are present in both the 8051 and the 8052. The "timer" or "counter" function is selected by control bits C/$\overline{\text{T}}$ in the Special Function Register TMOD (Figure 6-8). These two timer/counters have four operating modes, which are selected by bit-pairs (M1, M0) in TMOD. Modes 0, 1, and 2 are the same for both timer/counters. Mode 3 is different. The four operating modes are described below.

MODE 0

Putting either Timer into mode 0 makes it look like an 8048 Timer, which is an 8-bit counter with a divide-by-32 prescaler. Figure 6-9 shows the mode 0 operation as it applies to Timer 1.

In this mode, the timer register is configured as a 13-bit register. As the count rolls over from all 1s to all 0s, it sets the timer interrupt flag TF1. The counted input is enabled to the Timer when TR1 = 1 and either GATE = 0 or $\overline{\text{INT1}}$ = 1. (Setting GATE = 1 allows the Timer to be controlled by external input $\overline{\text{INT1}}$, to facilitate pulse width measurements.) TR1 is a control bit in the Special Function Register TCON (Figure 6-10). GATE is in TMOD.



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

C/$\overline{\text{T}}$ 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 6-8. TMOD: Timer/Counter Mode Control Register

The 13-bit register consists of all 8 bits of TH1 and the lower 5 bits of TL1. The upper 3 bits of TL1 are indeterminate and should be ingored. Setting the run flag (TR1) does not clear the registers.

Mode 0 operation is the same for Timer 0 as for Timer 1. Substitute TR0, TF0 and $\overline{\text{INT0}}$ for the corresponding Timer 1 signals in Figure 6-9. There are two different GATE bits, one for Timer 1 (TMOD.7) and one for Timer 0 (TMOD.3).

MODE 1

Mode 1 is the same as Mode 0, except that the Timer register is being run with all 16 bits.

MODE 2

Mode 2 configures the timer register as an 8-bit counter (TL1) with automatic reload, as shown in Figure 6-11. Overflow from TL1 not only sets TF1, but also reloads TL1 with the contents of TH1, which is preset by software. The reload leaves TH1 unchanged.

Mode 2 operation is the same for Timer/Counter 0.

MODE 3

Timer 1 in Mode 3 simply holds its count. The effect is the same as setting TR1 = 0.

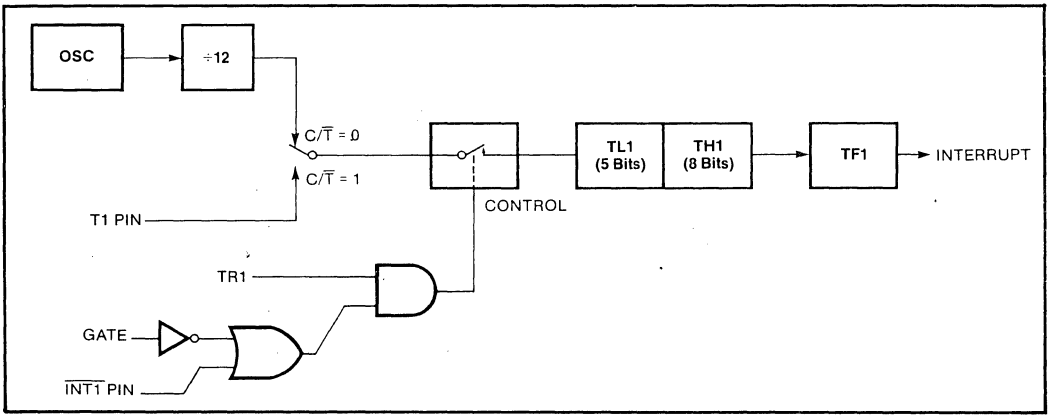

OSC
÷12
C/$\overline{\text{T}}$ = 0
C/$\overline{\text{T}}$ = 1
T1 PIN
CONTROL
TL1 (5 Bits)
TH1 (8 Bits)
TF1
INTERRUPT
TR1
GATE
$\overline{\text{INT1}}$ PIN

Figure 6-9. Timer/Counter 1 Mode 0: 13-bit Counter

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

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

Figure 6-10. TCON: Timer/Counter Control Register

Timer 0 in Mode 3 establishes TL0 and TH0 as two separate counters. The logic for Mode 3 on Timer 0 is shown in Figure 6-12. TL0 uses the Timer 0 control bits: C/$\overline{T}$, GATE, TR0, $\overline{INT0}$, and TF0. TH0 is locked into a timer function (counting machine cycles) and takes over the use of TR1 and TF1 from Timer 1. Thus, TH0 now controls the "Timer 1" interrupt.

Mode 3 is provided for applications requiring an extra 8-bit timer or counter. With Timer 0 in Mode 3, an 8051 can look like it has three timer/counters, and an 8052, like it has four. When Timer 0 is in Mode 3, Timer 1 can be turned on and off by switching it out of and into its own Mode 3, or can still be used by the serial port as a baud rate generator, or in fact, in any application not requiring an interrupt.

6.6.2 Timer 2

Timer 2 is a 16-bit timer/counter which is present only in the 8052. Like Timers 0 and 1, it can operate either as a timer or as an event counter. This is selected by bit C/$\overline{T2}$ in the Special Function Register T2CON (Figure 6-13). It has three operating modes: "capture," "auto-re-

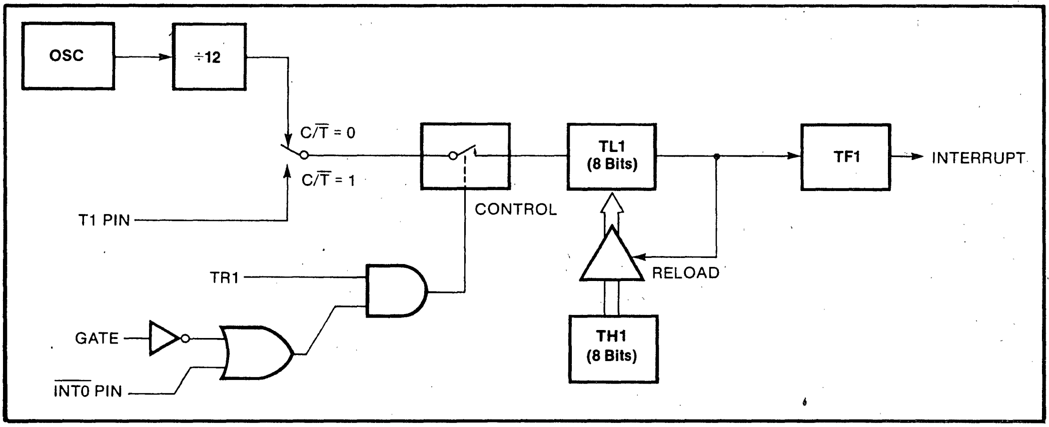


Figure 6-11. Timer/Counter 1 Mode 2: 8-bit Auto-reload

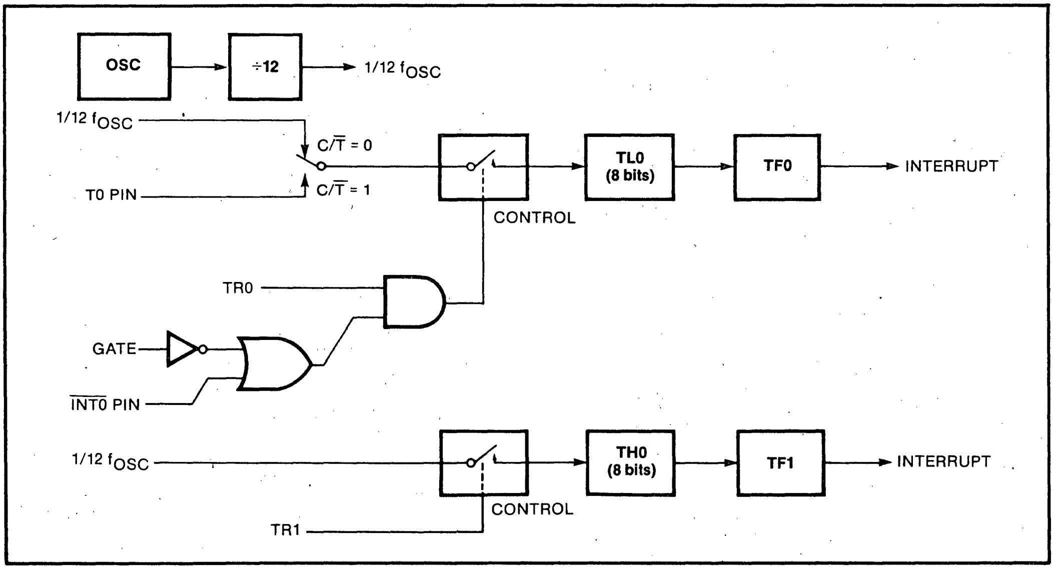


Figure 6-12. Timer/Counter 0 Mode 3: Two 8-bit Counters

(MSB)							(LSB)
TF2	EXF2	RCLK	TCLK	EXEN2	TR2	$C/\overline{T2}$	$CP/\overline{RL2}$

Symbol	Position	Name and Significance
TF2	T2CON.7	Timer 2 overflow flag set by a Timer 2 overflow and must be cleared by software. TF2 will not be set when either RCLK = 1 or TCLK = 1.
EXF2	T2CON.6	Timer 2 external flag set when either a capture or reload is caused by a negative transition on T2EX and EXEN2 = 1. When Timer 2 interrupt is enabled, EXF2 = 1 will cause the CPU to vector to the Timer 2 interrupt routine. EXF2 must be cleared by software.
RCLK	T2CON.5	Receive clock flag. When set, causes the serial port to use Timer 2 overflow pulses for its receive clock in modes 1 and 3. RCLK = 0 causes Timer 1 overflow to be used for the receive clock.
TCLK	T2CON.4	Transmit clock flag. When set, causes the serial port to use Timer 2 overflow pulses for its transmit clock in modes 1 and 3. TCLK = 0 causes Timer 1 overflows to be used for the transmit clock.
EXEN2	T2CON.3	Timer 2 external enable flag. When set, allows a capture or reload to occur as a result of a negative transition on T2EX if Timer 2 is not being used to clock the serial port. EXEN2 = 0 causes Timer 2 to ignore events at T2EX.
TR2	T2CON.2	Start/stop control for Timer 2. A logic 1 starts the timer.
$C/\overline{T2}$	T2CON.1	Timer or counter select. (Timer 2) 0 = Internal timer (OSC/12) 1 = External event counter (falling edge triggered).
$CP/\overline{RL2}$	T2CON.0	Capture/Reload flag. When set, captures will occur on negative transitions at T2EX if EXEN2 = 1. When cleared, auto reloads will occur either with Timer 2 overflows or negative transitions at T2EX when EXEN2 = 1. When either RCLK = 1 or TCLK = 1, this bit is ignored and the timer is forced to auto-reload on Timer 2 overflow.

Figure 6-13. T2CON: Timer/Counter 2 Control Register

load" and "baud rate generator," which are selected by bits in T2CON as shown in Table 2.

Table 2. Timer 2 Operating Modes

RCLK+TCLK	$CP/\overline{RL2}$	TR2	MODE
0	0	1	16-bit auto-reload
0	1	1	16-bit capture
1	X	1	baud rate generator
X	X	0	(off)

In the capture mode there are two options which are selected by bit EXEN2 in T2CON. If EXEN2 = 0, then Timer 2 is a 16-bit timer or counter which upon overflowing sets bit TF2, the Timer 2 overflow bit, which can be used to generate an interrupt. If EXEN2 = 1, then Timer 2 still does the above, but with the added feature that a 1-to-0 transition at external input T2EX causes the current value in the Timer 2 registers, TL2 and TH2, to be captured into registers RCAP2L and RCAP2H, respectively. (RCAP2L and RCAP2H are new Special Function Registers in the 8052.) In addition, the transition at T2EX causes bit EXF2 in T2CON to be set, and EXF2, like TF2, can generate an interrupt.

The capture mode is illustrated in Figure 6-14.

In the auto-reload mode there are again two options, which are selected by bit EXEN2 in T2CON. If EXEN2 = 0, then when Timer 2 rolls over it not only sets TF2 but also causes the Timer 2 registers to be reloaded with the 16-bit value in registers RCAP2L and RCAP2H, which are preset by software. If EXEN2 = 1, then Timer 2 still does the above, but with the added feature that a 1-to-0 transition at external input T2EX will also trigger the 16-bit reload and set EXF2.

The auto-reload mode is illustrated in Figure 6-15.

The baud rate generator mode is selected by RCLK = 1 and/or TCLK = 1. It will be described in conjunction with the serial port.

6.7 SERIAL INTERFACE

The serial port is full duplex, meaning it can transmit and receive simultaneously. It is also receive-buffered, meaning it can commence reception of a second byte before a previously received byte has been read from the receive register. (However, if the first byte still hasn't been read by the time reception of the second byte is complete, one of the bytes will be lost). The serial port receive and transmit registers are both accessed at Special Function Register SBUF. Writing to SBUF loads the transmit reg-

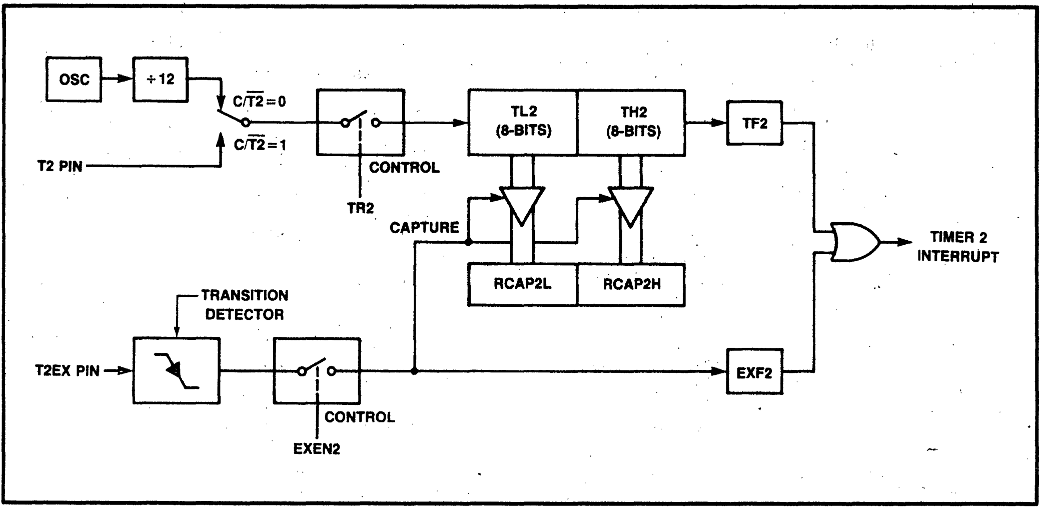


Figure 6-14. Timer 2 in Capture Mode

ister, and reading SBUF accesses a physically separate receive register.

The serial port can operate in 4 modes:

Mode 0: Serial data enters and exits through RXD. TXD outputs the shift clock. 8 bits are transmitted/received: 8 data bits (LSB first). The baud rate is fixed at 1/12 the oscillator frequency.

Mode 1: 10 bits are transmitted (through TXD) or received (through RXD): a start bit (0), 8 data bits (LSB first), and a stop bit (1). On receive, the stop bit goes into RB8 in Special Function Register SCON. The baud rate is variable.

Mode 2: 11 bits are transmitted (through TXD) or received (through RXD): a start bit (0), 8 data bits (LSB first), a programmable 9th data bit, and a stop bit (1). On transmit, the 9th data bit (TB8 in SCON) can be assigned the value of 0 or 1. Or, for example, the parity bit (P, in the PSW) could be moved into TB8. On receive, the 9th data bit goes into RB8 in Special Function Register SCON, while the stop bit is ignored. The baud rate is programmable to either 1/32 or 1/64 the oscillator frequency.

Mode 3: 11 bits are transmitted (through TXD) or received (through RXD): a start bit (0), 8 data bits (LSB first), a programmable 9th data bit and a stop bit (1). In fact, Mode 3 is the same as Mode 2 in all respects except the baud rate. The baud rate in Mode 3 is variable.

In all four modes, transmission is initiated by any instruction that uses SBUF as a destination register. Reception is initiated in Mode 0 by the condition RI = 0 and REN = 1. Reception is initiated in the other modes by the incoming start bit if REN = 1.

6.7.1 Multiprocessor Communications

Modes 2 and 3 have a special provision for multiprocessor communications. In these modes, 9 data bits are received. The 9th one goes into RB8. Then comes a stop bit. The port can be programmed such that when the stop bit is received, the serial port interrupt will be activated only if RB8 = 1. This feature is enabled by setting bit SM2 in SCON. A way to use this feature in multiprocessor systems is as follows.

When the master processor wants to transmit a block of data to one of several slaves, it first sends out an address byte which identifies the target slave. An address byte differs from a data byte in that the 9th bit is 1 in an address byte and 0 in a data byte. With SM2 = 1, no slave will be interrupted by a data byte. An address byte, however, will interrupt all slaves, so that each slave can examine the received byte and see if it is being addressed. The addressed slave will clear its SM2 bit and prepare to receive the data bytes that will be coming. The slaves that weren't being addressed leave their SM2s set and go on about their business, ignoring the coming data bytes.

SM2 has no effect in Mode 0, and in Mode 1 can be used to check the validity of the stop bit. In a Mode 1 reception, if SM2 = 1, the receive interrupt will not be activated unless a valid stop bit is received.

6.7.2 Serial Port Control Register

The serial port control and status register is the Special Function Register SCON, shown in Figure 6-16. This register contains not only the mode selection bits, but also the 9th data bit for transmit and receive (TB8 and RB8), and the serial port interrupt bits (TI and RI).

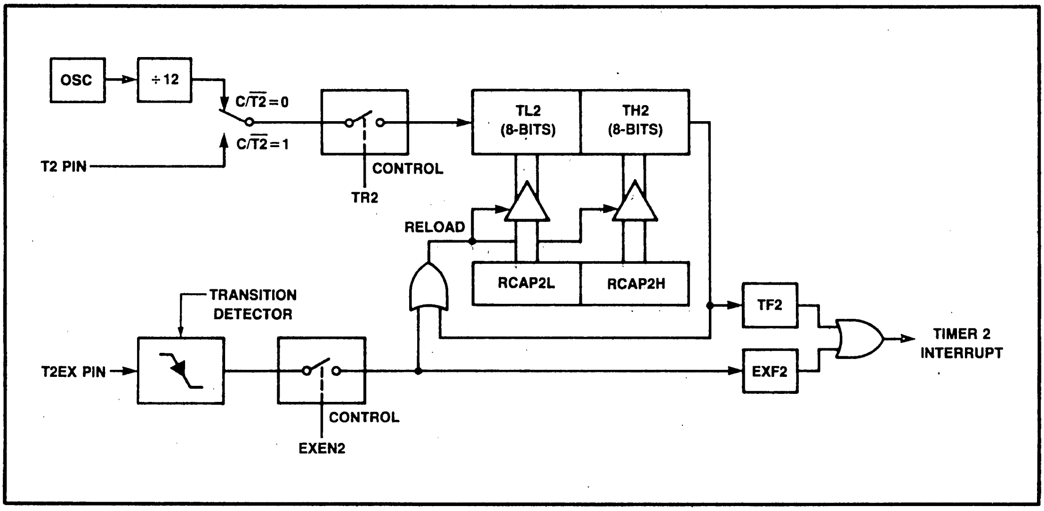


Figure 6-15. Timer 2 in Auto-Reload Mode

(MSB) (LSB)

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

where SM0, SM1 specify the serial port mode, as follows:

SM0	SM1	Mode	Description	Baud Rate
0	0	0	shift register	$f_{osc.}/12$
0	1	1	8-bit UART	variable
1	0	2	9-bit UART	$f_{osc.}/64$ or $f_{osc.}/32$
1	1	3	9-bit UART	variable

- SM2 enables the multiprocessor communication feature in modes 2 and 3. In mode 2 or 3, if SM2 is set to 1 then RI will not be activated if the received 9th data bit (RB8) is 0. In mode 1, if SM2 = 1 then RI will not be activated if a valid stop bit was not received. In mode 0, SM2 should be 0.
- REN enables serial reception. Set by software to enable reception. Clear by software to disable reception.
- TB8 is the 9th data bit that will be transmitted in modes 2 and 3. Set or clear by software as desired.
- RB8 in modes 2 and 3, is the 9th data bit that was received. In mode 1, if SM2 = 0, RB8 is the stop bit that was received. In mode 0, RB8 is not used.
- TI is transmit interrupt flag. Set by hardware at the end of the 8th bit time in mode 0, or at the beginning of the stop bit in the other modes, in any serial transmission. Must be cleared by software.
- RI is receive interrupt flag. Set by hardware at the end of the 8th bit time in mode 0, or halfway through the stop bit time in the other modes, in any serial reception (except see SM2). Must be cleared by software.

Figure 6-16. SCON: Serial Port Control Register

6.7.3 Baud Rates

The baud rate in Mode 0 is fixed:

$$\text{Mode 0 Baud Rate} = \frac{\text{Oscillator Frequency}}{12}$$

The baud rate in Mode 2 depends on the value of bit SMOD in Special Function Register PCON. If SMOD = 0 (which is its value on reset), the baud rate is 1/64 the oscillator frequency. If SMOD = 1, the baud rate is 1/32 the oscillator frequency.

$$\text{Mode 2 Baud Rate} = \frac{2^{\text{SMOD}}}{64} \text{ x (Oscillator Frequency)}$$

In the 8051, the baud rates in Modes 1 and 3 are determined by the Timer 1 overflow rate. In the 8052, these baud rates can be determined by Timer 1, or by Timer 2, or by both (one for transmit and the other for receive).

Using Timer 1 to Generate Baud Rates

When Timer 1 is used as the baud rate generator, the baud rates in Modes 1 and 3 are determined by the Timer 1 overflow rate and the value of SMOD as follows:

$$\text{Modes 1, 3 Baud Rate} = \frac{2^{SMOD}}{32} \times \text{(Timer 1 Overflow Rate)}$$

The Timer 1 interrupt should be disabled in this application. The Timer itself can be configured for either "timer" or "counter" operation, and in any of its 3 running modes. In the most typical applications, it is configured for "timer" operation, in the auto-reload mode (high nibble of TMOD = 0010B). In that case, the baud rate is given by the formula

$$\text{Modes 1, 3 Baud Rate} = \frac{2^{SMOD}}{32} \times \frac{\text{Oscillator Frequency}}{12x[256-(TH1)]}$$

One can achieve very low baud rates with Timer 1 by leaving the Timer 1 interrupt enabled, and configuring the Timer to run as a 16-bit timer (high nibble of TMOD = 0001B), and using the Timer 1 interrupt to do a 16-bit software reload.

Figure 6-17 lists various commonly used baud rates and how they can be obtained from Timer 1.

BAUD RATE	f_{OSC}	SMOD	TIMER 1		
			$C/\overline{T}$	MODE	RELOAD VALUE
MODE 0 MAX: 1MHZ	12 MHZ	X	X	X	X
MODE 2 MAX: 375K	12 MHZ	1	X	X	X
MODES 1,3: 62.5K	12 MHZ	1	0	2	FFH
19.2K	11.059 MHZ	1	0	2	FDH
9.6K	11.059 MHZ	0	0	2	FDH
4.8K	11.059 MHZ	0	0	2	FAH
2.4K	11.059 MHZ	0	0	2	F4H
1.2K	11.059 MHZ	0	0	2	E8H
137.5	11.986 MHZ	0	0	2	1DH
110	6 MHZ	0	0	2	72H
110	12 MHZ	0	0	1	FEEBH

Figure 6-17. Timer 1 Generated Commonly Used Baud Rates

Using Timer 2 to Generate Baud Rates

In the 8052, Timer 2 is selected as the baud rate generator by setting TCLK and/or RCLK in T2CON (Figure 6-13). Note then the baud rates for transmit and receive can be simultaneously different. Setting RCLK and/or TCLK puts Timer 2 into its baud rate generator mode, as shown in Figure 6-18.

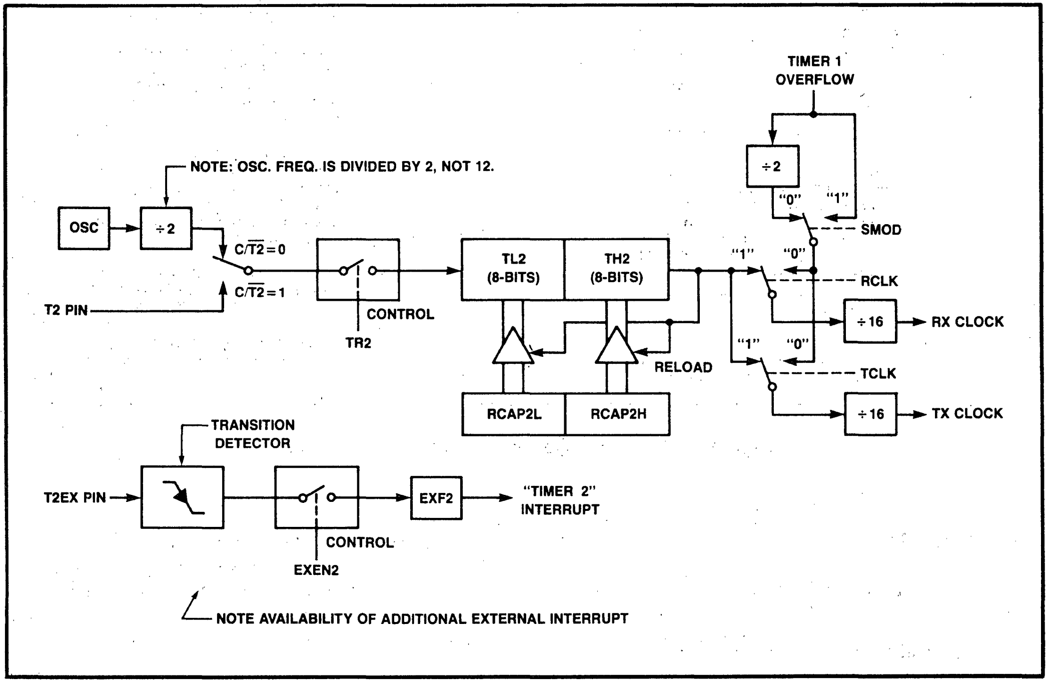


Figure 6-18. Timer 2 in Baud Rate Generator Mode

The baud rate generator mode is similar to the auto-reload mode, in that a rollover in TH2 causes the Timer 2 registers to be reloaded with the 16-bit value in registers RCAP2H and RCAP2L, which are preset by software.

Now, the baud rates in Modes 1 and 3 are determined by Timer 2's overflow rate as follows:

$$\text{Modes 1, 3 Baud Rate} = \frac{\text{Timer 2 Overflow Rate}}{16}$$

The Timer can be configured for either "timer" or "counter" operation. In the most typical applications, it is configured for "timer" operation ($C/\overline{T2} = 0$). "Timer" operation is a little different for Timer 2 when it's being used as a baud rate generator. Normally as a timer it would increment every machine cycle (thus at 1/12 the oscillator frequency). As a baud rate generator, however, it increments every state time (thus at 1/2 the oscillator frequency). In that case the baud rate is given by the formula

$$\begin{array}{l}\text{Modes 1, 3}\\ \text{Baud Rate}\end{array} = \frac{\text{Oscillator Frequency}}{32x[65536-(\text{RCAP2H, RCAP2L})]}$$

where (RCAP2H, RCAP2L) is the content of RCAP2H and RCAP2L taken as a 16-bit unsigned integer.

Timer 2 as a baud rate generator is shown in Figure 6-18. This Figure is valid only if RCLK + TCLK = 1 in T2CON. Note that a rollover in TH2 does not set TF2, and will not generate an interrupt. Therefore, the Timer 2 interrupt does not have to be disabled when Timer 2 is in the baud rate generator mode. Note too, that if EXEN2 is set, a 1-to-0 transition in T2EX will set EXF2 but will not cause a reload from (RCAP2H, RCAP2L) to (TH2, TL2). Thus when Timer 2 is in use as a baud rate generator, T2EX can be used as an extra external interrupt, if desired.

It should be noted that when Timer 2 is running (TR2 = 1) in "timer" function in the baud rate generator mode, one should not try to read or write TH2 or TL2. Under these conditions the Timer is being incremented every state time, and the results of a read or write may not be accurate. The RCAP registers may be read, but shouldn't be written to, because a write might overlap a reload and cause write and/or reload errors. Turn the Timer off (clear TR2) before accessing the Timer 2 or RCAP registers, in this case.

6.7.4 More About Mode 0

Serial data enters and exits through RXD. TXD outputs the shift clock. 8 bits are transmitted/received: 8 data bits (LSB first). The baud rate is fixed at 1/12 the oscillator frequency.

Figure 6-19 shows a simplified functional diagram of the serial port in mode 0, and associated timing.

Transmission is initiated by any instruction that uses SBUF as a destination register. The "write to SBUF" signal at S6P2 also loads a 1 into the 9th bit position of the transmit shift register and tells the TX Control block to commence a transmission. The internal timing is such that one full machine cycle will elapse between "write to SBUF," and activation of SEND.

SEND enables the output of the shift register to the alternate output function line of P3.0, and also enables SHIFT CLOCK to the alternate output function line of P3.1. SHIFT CLOCK is low during S3, S4, and S5 of every machine cycle, and high during S6, S1 and S2. At S6P2 of every machine cycle in which SEND is active, the contents of the transmit shift register are shifted to the right one position.

As data bits shift out to the right, zeros come in from the left. When the MSB of the data byte is at the output position of the shift register, then the 1 that was initially loaded into the 9th position, is just to the left of the MSB, and all positions to the left of that contain zeros. This condition flags the TX Control block to do one last shift and then deactivate SEND and set T1. Both of these actions occur at S1P1 of the 10th machine cycle after "write to SBUF."

Reception is initiated by the condition REN = 1 and RI = 0. At S6P2 of the next machine cycle, the RX Control unit writes the bits 11111110 to the receive shift register, and in the next clock phase activates RECEIVE.

RECEIVE enables SHIFT CLOCK to the alternate output function line of P3.1. SHIFT CLOCK makes transitions at S3P1 and S6P1 of egvery machine cycle. At S6P2 of every machine cycle in which RECEIVE is active, the contents of the receive shift register are shifted to the left one position. The value that comes in from the right is the value that was sampled at the P3.0 pin at S5P2 of the same machine cycle.

As data bits come in from the right, 1s shift out to the left. When the 0 that was initially loaded into the rightmost position arrives at the leftmost position in the shift register, it flags the RX Control block to do one last shift and load SBUF. At S1P1 of the 10th machine cycle after the write to SCON that cleared RI, RECEIVE is cleared and RI is set.

6.7.5 More About Mode 1

Ten bits are transmitted (through TXD), or received (through RXD): a start bit (0), 8 data bits (LSB first), and a stop bit (1). On receive, the stop bit goes into RB8 in SCON. In the 8051 the baud rate is determined by the Timer 1 overflow rate. In the 8052 it is determined either by the Timer 1 overflow rate, or the Timer 2 overflow rate, or both (one for transmit and the other for receive).

Figure 6-20 shows a simplified functional diagram of the serial port in Mode 1, and associated timings for transmit and receive.

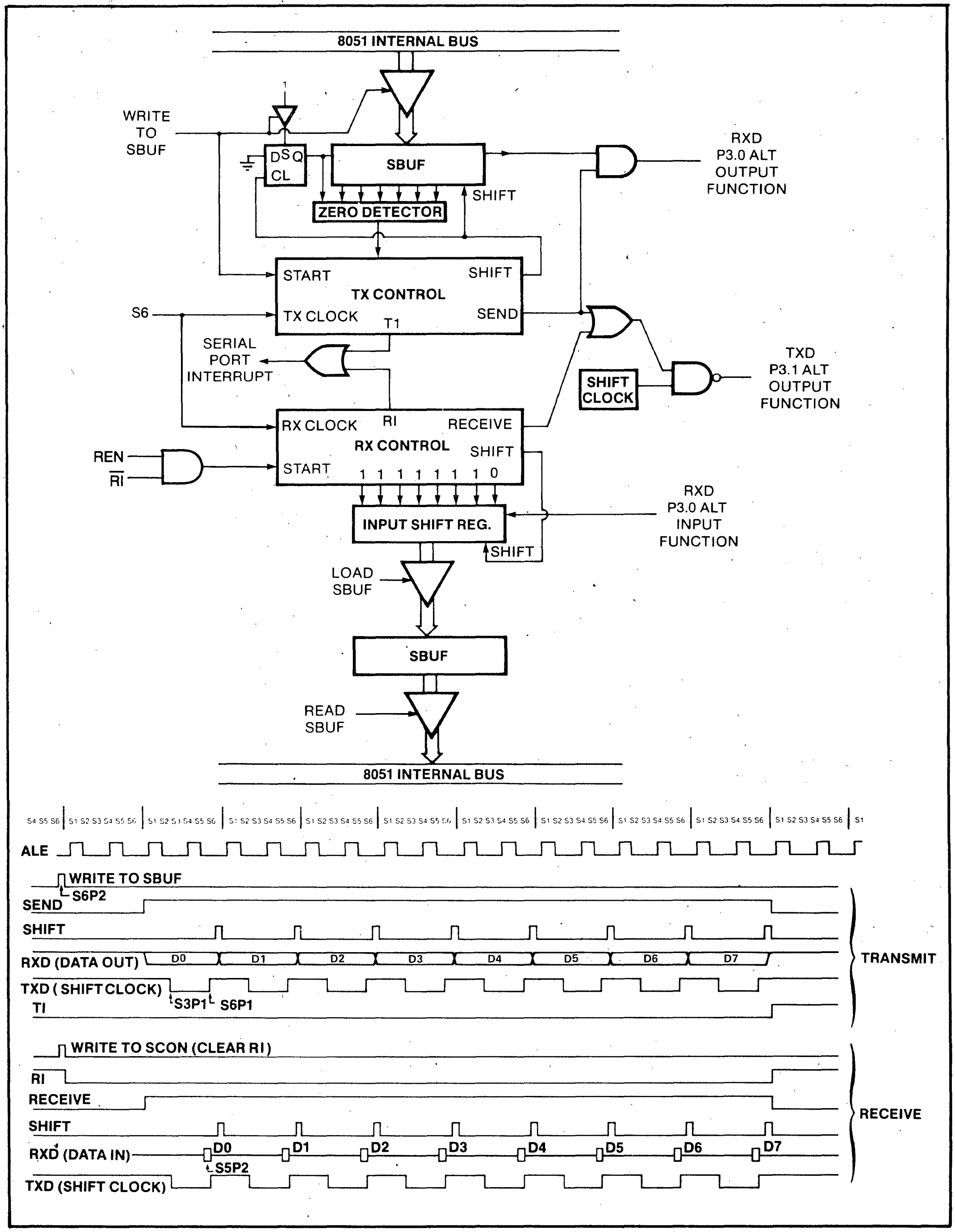


Figure 6-19. Serial Port Mode 0

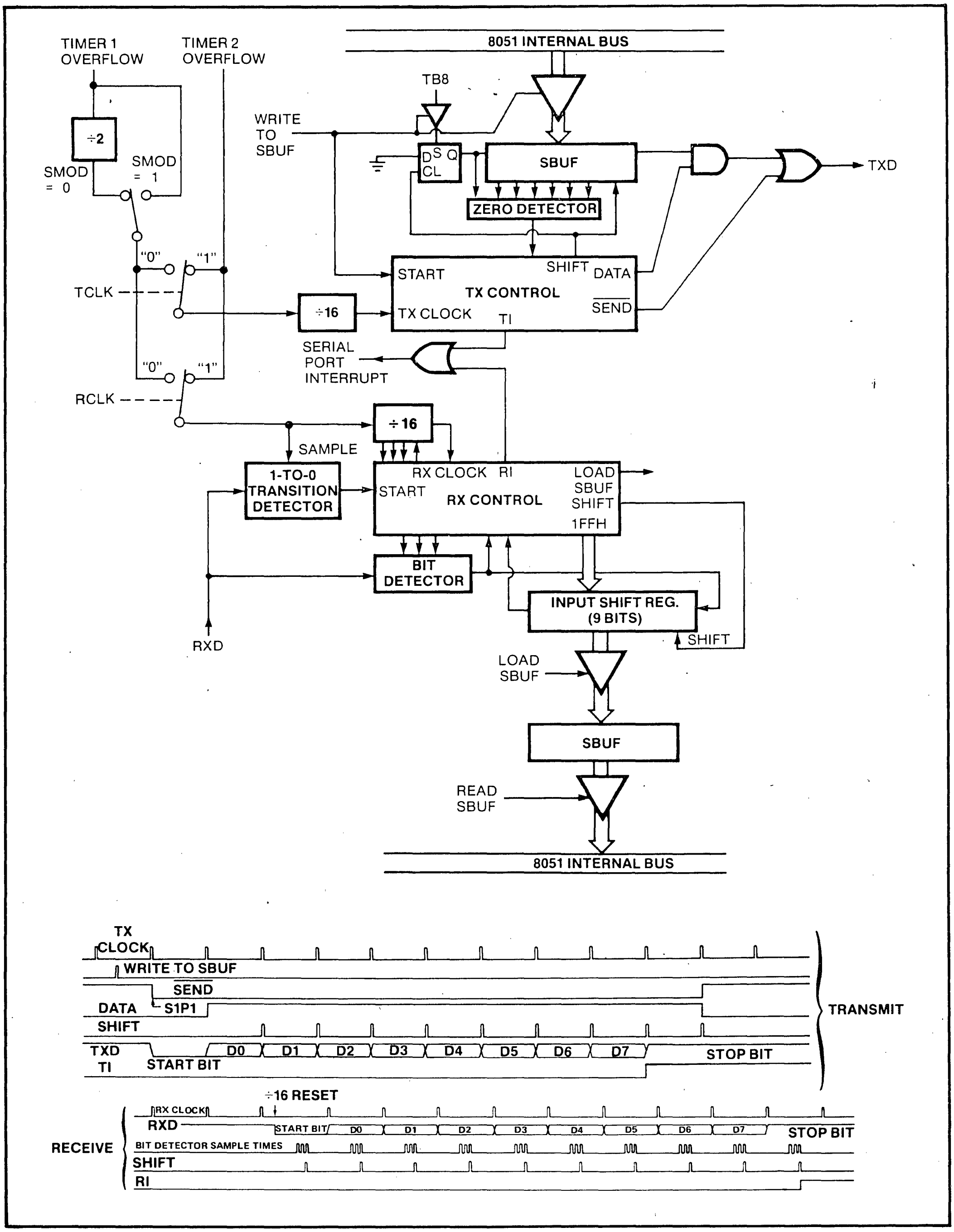


Figure 6-20. Serial Port Mode 1
TCLK, RCLK, and Timer 2 are present in the 8052/8032 only.

Transmission is initiated by any instruction that uses SBUF as a destination register. The "write to SBUF" signal also loads a 1 into the 9th bit position of the transmit shift register and flags the TX Control unit that a transmission is requested. Transmission actually commences at S1P1 of the machine cycle following the next rollover in the divide-by-16 counter. (Thus, the bit times are synchronized to the divide-by-16 counter, not to the "write to SBUF" signal).

The transmission begins with activation of $\overline{\text{SEND}}$, which puts the start bit at TXD. One bit time later, DATA is activated, which enables the output bit of the transmit shift register to TXD. The first shift pulse occurs one bit time after that.

As data bits shift out to the right, zeros are clocked in from the left. When the MSB of the data byte is at the output position of the shift register, then the 1 that was initially loaded into the 9th position is just to the left of the MSB, and all positions to the left of that contain zeroes. This condition flags the TX Control unit to do one last shift and then deactivate $\overline{\text{SEND}}$ and set T1. This occurs at the 10th divide-by-16 rollover after "write to SBUF."

Reception is initiated by a detected 1-to-0 transition at RXD. For this purpose RXD is sampled at a rate of 16 times whatever baud rate has been established. When a transition is detected, the divide-by-16 counter is immediately reset, and 1FFH is written into the input shift register. Resetting the divide-by-16 counter aligns its rollovers with the boundaries of the incoming bit times.

The 16 states of the counter divide each bit time into 16ths. At the 7th, 8th, and 9th counter states of each bit time, the bit detector samples the value of RXD. The value accepted is the value that was seen in at least 2 of the 3 samples. This is done for noise rejection. If the value accepted during the first bit time is not 0, the receive circuits are reset and the unit goes back to looking for another 1-to-0 transition. This is to provide rejection of false start bits. If the start bit proves valid, it is shifted into the input shift register, and reception of the rest of the frame will proceed.

As data bits come in from the right, 1s shift out to the left. When the start bit arrives at the leftmost position in the shift register, (which in mode 1 is a 9-bit register), it flags the RX Control block to do one last shift, load SBUF and RB8, and set RI. The signal to load SBUF and RB8, and to set RI, will be generated if, and only if, the following conditions are met at the time the final shift pulse is generated.

1) RI = 0, and
2) Either SM2 = 0, or the received stop bit = 1

If either of these two conditions are not met, the received frame is irretrievably lost. If both conditions are met, the stop bit goes into RB8, the 8 data bits go into SBUF, and RI is activated. At this time, whether the above conditions are met or not, the unit goes back to looking for a 1-to-0 transition in RXD.

6.7.6 More About Modes 2 and 3

Eleven bits are transmitted (through TXD), or received (through RXD): a start bit (0), 8 data bits (LSB first), a programmable 9th data bit, and a stop bit (1). On transmit, the 9th data bit (TD8) can be assigned the value of 0 or 1. On receive, the 9th data bit goes into RB8 in SCON. The baud rate is programmable to either 1/32 or 1/64 the oscillator frequency in mode 2. Mode 3 may have a variable baud rate generated from either Timer 1 or 2 depending on the state of TCLK and RCLK.

Figures 6-21 A and B show a functional diagram of the serial port in modes 2 and 3. The receive portion is exactly the same as in mode 1. The transmit portion differs from mode 1 only in the 9th bit of the transmit shift register.

Transmission is initiated by any instruction that uses SBUF as a destination register. The "write to SBUF" signal also loads TB8 into the 9th bit position of the transmit shift register and flags the TX Control unit that a transmission is requested. Transmission commences at S1P1 of the machine cycle following the next rollover in the divide-by-16 counter. (Thus, the bit times are synchronized to the divide-by-16 counter, not to the "write to SBUF" signal.)

The transmission begins with activation of SEND, which puts the start bit at TXD. One bit time later, DATA is activated, which enables the output bit of the transmit shift register to TXD. The first shift pulse occurs one bit time after that. The first shift clocks a 1 (the stop bit) into the 9th bit position of the shift register. Thereafter, only zeroes are clocked in. Thus, as data bits shift out to the right, zeroes are clocked in from the left. When TB8 is at the output position of the shift register, then the stop bit is just to the left of TB8, and all positions to the left of that contain zeroes. This condition flags the TX Control unit to do one last shift and then deactivate $\overline{\text{SEND}}$ and set T1. This occurs at the 11th divide-by-16 rollover after "write to SBUF."

Reception is initiated by a detected 1-to-0 transition at RXD. For this purpose RXD is sampled at a rate of 16 times whatever baud rate has been established. When a transition is detected, the divide-by-16 counter is immediately reset, and 1FFH is written to the input shift register.

At the 7th, 8th and 9th counter states of each bit time, the bit detector samples the value of RXD. The value accepted is the value that was seen in at least 2 of the 3 samples. If the value accepted during the first bit time is not 0, the receive circuits are reset and the unit goes back to looking for another 1-to-0 transition. If the start bit proves valid, it is shifted into the input shift register, and reception of the rest of the frame will proceed.

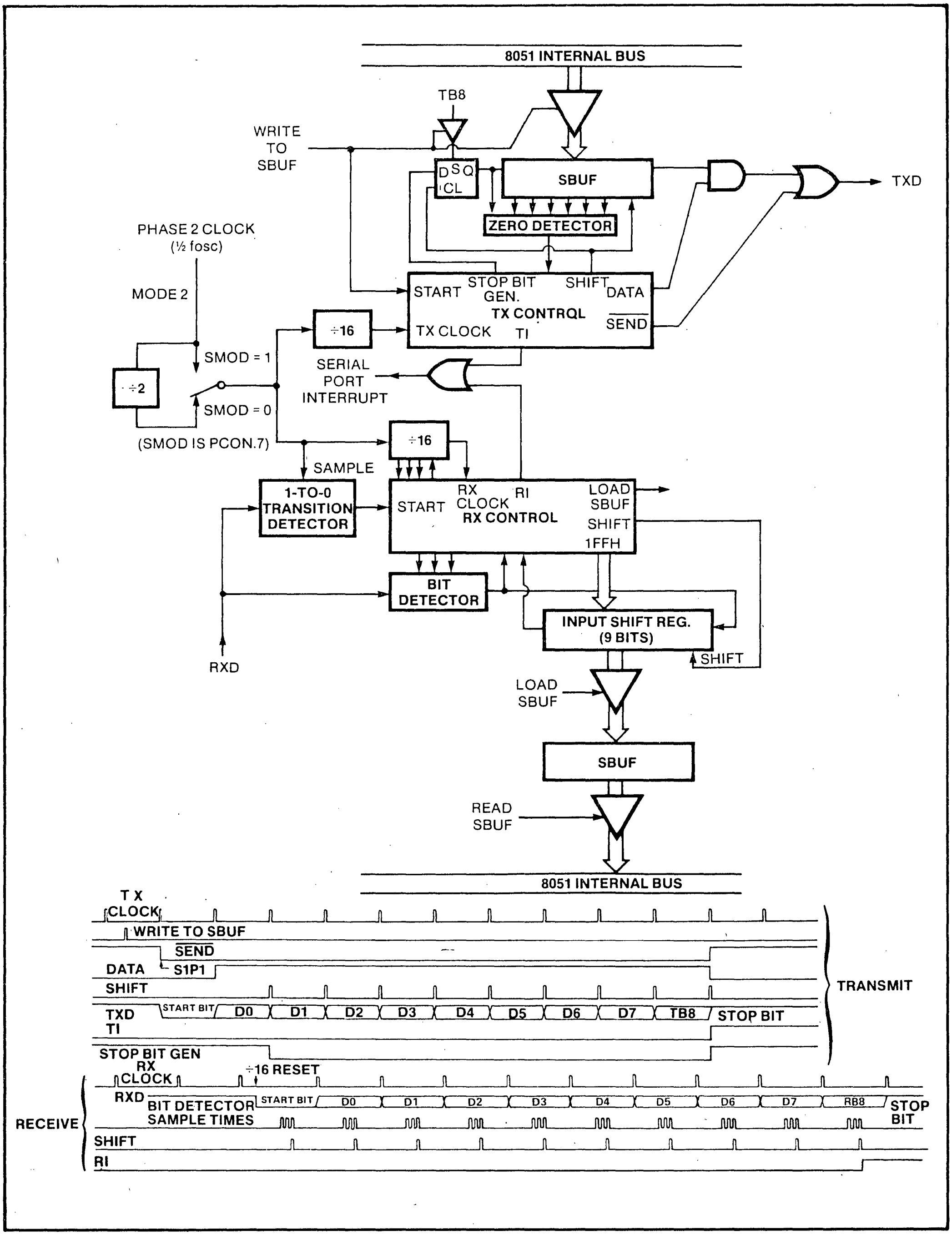


Figure 6-21A. Serial Port Mode 2

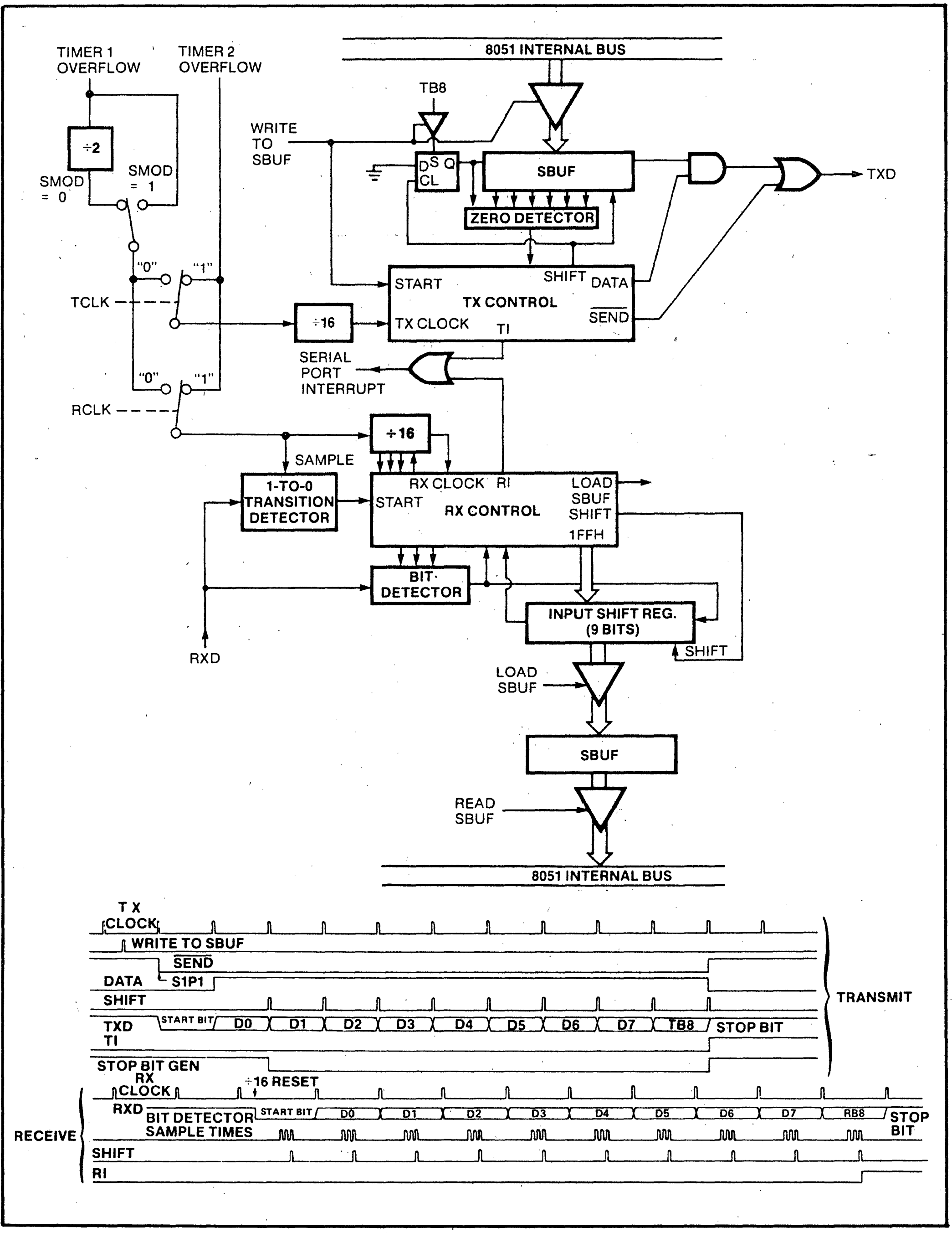


Figure 6-21B. Serial Port Mode 3
TCLK, RCLK, and Timer 2 are present in the 8052/8032 only.

As data bits come in from the right, 1s shift out to the left. When the start bit arrives at the leftmost position in the shift register (which in modes 2 and 3 is a 9-bit register), it flags the RX Control block to do one last shift, load SBUF and RB8, and set RI. The signal to load SBUF and RB8, and to set RI, will be generated if, and only if, the following conditions are met at the time the final shift pulse is generated:

1) RI = 0, and
2) Either SM2 = 0 or the received 9th data bit = 1

If either of these conditions are not met, the received frame is irretrievably lost, and RI is not set. IF both conditions are met, the received 9th data bit goes into RB8, and the first 8 data bits go into SBUF. One bit time later, whether the above conditions were met or not, the unit goes back to looking for a 1-to-0 transition at the RXD input.

Note that the value of the received stop bit is irrelevant to SBUF, RB8, or RI.

6.8 INTERRUPTS

The 8051 provides 5 interrupt sources. The 8052 provides 6. These are shown in Figure 6-22.

The External Interrupts $\overline{\text{INT0}}$ and $\overline{\text{INT1}}$ can each be either level-activated or transition-activated, depending on bits IT0 and IT1 in Register TCON. The flags that actually generate these interrupts are bits IE0 and IE1 in TCON. When an external interrupt is generated, the flag that generated it is cleared by the hardware when the service routine is vectored to only if the interrupt was transition-activated. If the interrupt was level-activated, then the external requesting source is what controls the request flag, rather than the on-chip hardware.

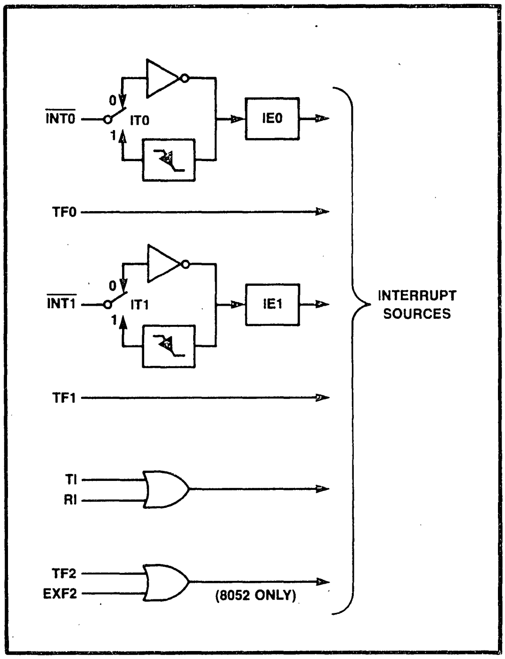


Figure 6-22. MCS-51 Interrupt Sources

The Timer 0 and Timer 1 Interrupts are generated by TF0 and TF1, which are set by a rollover in their respective timer/counter registers (except see Section 6.6.1 for Timer 0 in mode 3). When a timer interrupt is generated, the flag that generated it is cleared by the on-chip hardware when the service routine is vectored to.

The Serial Port Interrupt is generated by the logical OR of RI and TI. Neither of these flags is cleared by hardware when the service routine is vectored to. In fact, the service routine will normally have to determine whether it was RI or TI that generated the interrupt, and the bit will have to be cleared in software.

In the 8052, the Timer 2 Interrupt is generated by the logical OR of TF2 and EXF2. Neither of these flags is cleared by hardware when the service routine is vectored to. In fact, the service routine may have to determine whether it was TF2 or EXF2 that generated the interrupt, and the bit will have to be cleared in software.

All of the bits that generate interrupts can be set or cleared by software, with the same result as though it had been set or cleared by hardware. That is, interrupts can be generated or pending interrupts can be canceled in software.

(MSB)							(LSB)
EA	X	ET2	ES	ET1	EX1	ET0	EX0

Symbol	Position	Function
EA	IE.7	disables all interrupts. If EA = 0, no interrupt will be acknowledged. If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.
—	IE.6	reserved
ET2	IE.5	enables or disables the Timer 2 overflow or capture interrupt. If ET2 = 0, the Timer 2 interrupt is disabled.
ES	IE.4	enables or disables the Serial Port interrupt. If ES = 0, the Serial Port Interrupt is disabled.
ET1	IE.3	enables or disables the Timer 1 Overflow interrupt. If ET1 = 0, the Timer 1 interrupt is disabled.
EX1	IE.2	enables or disables External Interrupt 1. If EX1 = 0, External Interrupt 1 is disabled.
ET0	IE.1	enables or disables the Timer 0 Overflow interrupt. If ET0 = 0, the Timer 0 Interrupt is disabled.
EX0	IE.0	enables or disables External Interrupt 0. If EX0 = 0, External Interrupt 0 is disabled.

Figure 6-23. IE: Interrupt Enable Register

Each of these interrupt sources can be individually enabled or disabled by setting or clearing a bit in Special Function Register IE (Figure 6-23). Note that IE contains also a global disable bit, EA, which disables all interrupts at once.

6.8.1 Priority Level Structure

Each interrupt source can also be individually programmed to one of two priority levels by setting or clearing a bit in Special Function Register IP (Figure 6-24). A low-priority interrupt can itself be interrupted by a high-priority interrupt, but not by another low-priority interrupt. A high-priority interrupt can't be interrupted by any other interrupt source.

(MSB)							(LSB)
X	X	PT2	PS	PT1	PX1	PT0	PX0

Symbol	Position	Function
—	IP.7	reserved
—	IP.6	reserved
PT2	IP.5	defines the Timer 2 interrupt priority level. PT2 = 1 programs it to the higher priority level.
PS	IP.4	defines the Serial Port interrupt priority level. PS = 1 programs it to the higher priority level.
PT1	IP.3	defines the Timer 1 interrupt priority level. PT1 = 1 programs it to the higher priority level.
PX1	IP.2	defines the External Interrupt 1 priority level. PX1 = 1 programs it to the higher priority level.
PT0	IP.1	defines the Timer 0 interrupt priority level. PT0 = 1 programs it to the higher priority level.
PX0	IP.0	defines the External Interrupt 0 priority level. PX0 = 1 programs it to the higher priority level.

Figure 6-24. IP: Interrupt Priority Register

If two requests of different priority levels are received simultaneously, the request of higher priority level is serviced. If requests of the **same** priority level are received simultaneously, an internal polling sequence determines which request is serviced. Thus within each priority level there is a second priority structure determined by the polling sequence, as follows:

	SOURCE	PRIORITY WITHIN LEVEL
1.	IE0	(highest)
2.	TF0	
3.	IE1	
4.	TF1	
5.	RI + TI	
6.	TF2 + EXF2	(lowest)

Note that the "priority within level" structure is only used to resolve *simultaneous requests of the same priority level.*

6.8.2 How Interrupts Are Handled

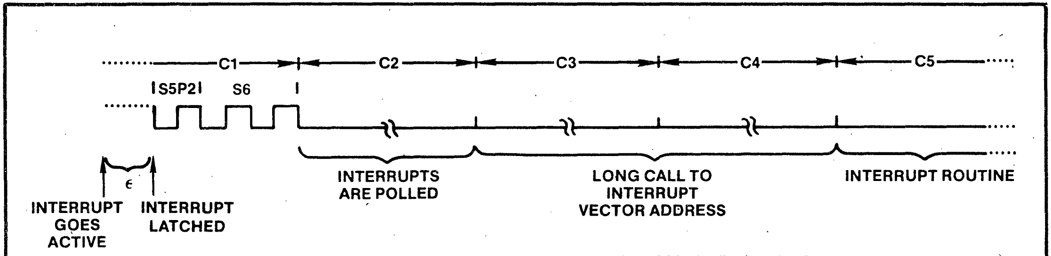
The interrupt flags are sampled at S5P2 of every machine cycle. The samples are polled during the following machine cycle. If one of the flags was in a set condition at S5P2 of the preceding cycle, the polling cycle will find it and the interrupt system will generate an LCALL to the appropriate service routine, provided this hardware-generated LCALL is not blocked by any of the following conditions:

1. An interrupt of equal or higher priority level is already in progress.

2. The current (polling) cycle is not the final cycle in the execution of the instruction in progress.

3. The instruction in progress is RETI or any access to the IE or IP registers.

Any of these three conditions will block the generation of the LCALL to the interrupt service routine. Condition 2 ensures that the instruction in progress will be completed



This is the fastest possible response when C2 is the final cycle of an instruction other than RETI or an access to IE or IP.

Figure 6-25. Interrupt Response Timing Diagram

before vectoring to any service routine. Condition 3 ensures that if the instruction in progress is RETI or any access to IE or IP, then at least *one more* instruction will be executed before any interrupt is vectored to.

The polling cycle is repeated with each machine cycle, and the values polled are the values that were present at S5P2 of the previous machine cycle. Note then that if an interrupt flag is active but not being responded to for one of the above conditions, if the flag is not *still* active when the blocking condition is removed, the denied interrupt will not be serviced. In other words, the fact that the interrupt flag was once active but not serviced is not remembered. Every polling cycle is new.

The polling cycle/LCALL sequence is illustrated in Figure 6-25.

Note that if an interrupt of higher priority level goes active prior to S5P2 of the machine cycle labeled C3 in Figure 6-25, then in accordance with the above rules it will be vectored to during C5 and C6, without any instruction of the lower priority routine having been executed.

Thus the processor acknowledges an interrupt request by executing a hardware-generated LCALL to the appropriate servicing routine. In some cases it also clears the flag that generated the interrupt, and in other cases it doesn't. It never clears the Serial Port or Timer 2 flags. This has to be done in the user's software. It clears an external interrupt flag (IE0 or IE1) only if it was transition-activated. The hardware-generated LCALL pushes the contents of the Program Counter onto the stack (but it does not save the PSW) and reloads the PC with an address that depends on the source of the interrupt being vectored to, as shown below.

SOURCE	VECTOR ADDRESS
IE0	0003H
TF0	000BH
IE1	0013H
TF1	001BH
RI+TI	0023H
TF2+EXF2	002BH

Execution proceeds from that location until the RETI instruction is encountered. The RETI instruction informs the processor that this interrupt routine is no longer in progress, then pops the top two bytes from the stack and reloads the Program Counter. Execution of the interrupted program continues from where it left off.

Note that a simple RET instruction would also have returned execution to the interrupted program, but it would have left the interrupt control system thinking an interrupt was still in progress.

6.8.3 External Interrupts

The external sources can be programmed to be level-activated or transition-activated by setting or clearing bit IT1 or IT0 in Register TCON. If ITx = 0, external interrupt x is triggered by a detected low at the $\overline{\text{INTx}}$ pin. If ITx = 1, external interrupt x is edge-triggered. In this mode if successive samples of the $\overline{\text{INTx}}$ pin show a high in one cycle and a low in the next cycle, interrupt request flag IEx in TCON is set. Flag bit IEx then requests the interrupt.

Since the external interrupt pins are sampled once each machine cycle, an input high or low should hold for at least 12 oscillator periods to ensure sampling. If the external interrupt is transition-activated, the external source has to hold the request pin high for at least one cycle, and then hold it low for at least one cycle to ensure that the transition is seen so that interrupt request flag IEx will be set. IEx will be automatically cleared by the CPU when the service routine is called.

If the external interrupt is level-activated, the external source has to hold the request active until the requested interrupt is actually generated. Then it has to deactivate the request before the interrupt service routine is completed, or else another interrupt will be generated.

6.8.4 Response Time

The $\overline{\text{INT0}}$ and $\overline{\text{INT1}}$ levels are inverted and latched into IE0 and IE1 at S5P2 of every machine cycle. The values are not actually polled by the circuitry until the next machine cycle. If a request is active and conditions are right for it to be acknowledged, a hardware subroutine call to the requested service routine will be the next instruction to be executed. The call itself takes two cycles. Thus, a minimum of three complete machine cycles elapse between activation of an external interrupt request and the beginning of execution of the first instruction of the service routine. Figure 6-25 shows interrupt response timings.

A longer response time would result if the request is blocked by one of the 3 previously listed conditions. If an interrupt of equal or higher priority level is already in progress, the additional wait time obviously depends on the nature of the other interrupt's service routine. If the instruction in progress is not in its final cycle, the additional wait time cannot be more than 3 cycles, since the longest instructions (MUL and DIV) are only 4 cycles long, and if the instruction in progress is RETI or an access to IE or IP, the additional wait time cannot be more than 5 cycles (a maximum of one more cycle to complete the instruction in progress, plus 4 cycles to complete the next instruction if the instruction is MUL or DIV).

Thus, in a single-interrupt system, the response time is always more than 3 cycles and less than 8 cycles.

6.9 SINGLE-STEP OPERATION

The 8051 interrupt structure allows single-step execution with very little software overhead. As previously noted, an interrupt request will not be responded to while an interrupt of equal priority level is still in progress, nor will it be responded to after RETI until at least one other instruction has been executed. Thus, once an interrupt routine has been entered, it cannot be re-entered until at least once instruction of the interrupted program is executed. One way to use this feature for single-step operation is to program one of the external interrupts (say, $\overline{\text{INT0}}$) to be level-activated. The service routine for the interrupt will terminate with the following code:

```
JNB     P3.2,$     ;WAIT HERE TILL INTO
                   GOES HIGH
JB      P3.2,$     ;NOW WAIT HERE TILL
                   IT GOES LOW
RETI               :GO BACK AND
                   EXECUTE ONE
                   INSTRUCTION
```

Now if the $\overline{\text{INT0}}$ pin, which is also the P3.2 pin, is held normally low, the CPU will go right into the External Interrupt 0 routine and stay there until $\overline{\text{INT0}}$ is pulsed (from low to high to low). Then it will execute RETI, go back to the task program, execute one instruction, and immediately re-enter the External Interrupt 0 routine to await the next pulsing of P3.2. One step of the task program is executed each time P3.2 is pulsed.

6.10 RESET

The reset input is the RST pin, which is the input to a Schmitt Trigger.

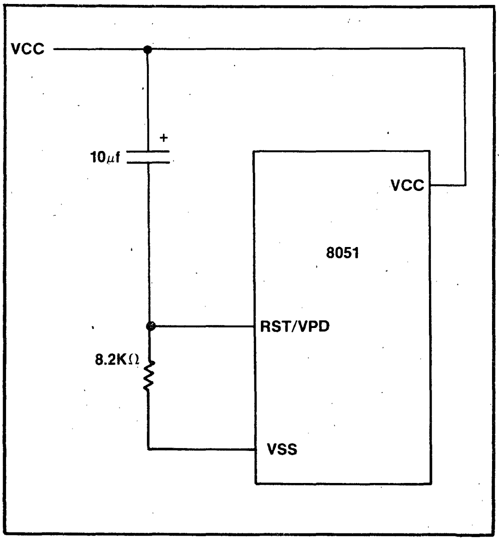


Figure 6-26. Power on Reset Circuit

A reset is accomplished by holding the RST pin high for at least two machine cycles (24 oscillator periods), *while the oscillator is running*. The CPU responds by executing an internal reset. It also configures the ALE and $\overline{\text{PSEN}}$ pins as inputs. (They are quasi-bidirectional). The internal reset is executed during the second cycle in which RST is high and is repeated every cycle until RST goes low. It leaves the internal registers as follows:

REGISTER	CONTENT
PC	0000H
ACC	00H
B	00H
PSW	00H
SP	07H
DPTR	0000H
P0–P3	0FFH
IP	(XX000000)
IE	(0X000000)
TMOD	00H
TCON	00H
T2CON	00H
TH0	00H
TL0	00H
TH1	00H
TL1	00H
TH2	00H
TL2	00H
RCAP2H	00H
RCAP2L	00H
SCON	00H
SBUF	Indeterminate
PCON	(0XXX0000)

The internal RAM is not affected by reset. When VCC is turned on, the RAM content is indeterminate unless the part is returning from a reduced power mode of operation.

POWER-ON RESET

An automatic reset can be obtained when VCC is turned on by connecting the RST pin to VCC through a 10 µf capacitor and to VSS through an 8.2KΩ resistor, providing the VCC risetime does not exceed a millisecond and the oscillator start-up time does not exceed 10 milliseconds. This power-on reset circuit is shown in Figure 6-26. When power comes on, the current drawn by RST commences to charge the capacitor. The voltage at RST is the difference between VCC and the capacitor voltage, and decreases from VCC as the cap charges. The larger the capacitor, the more slowly VRST decreases. VRST must remain above the lower threshhold of the Schmitt Trigger long enough to effect a complete reset. The time required is the oscillator start-up time, plus 2 machine cycles.

6.11 POWER-SAVING MODES OF OPERATION

For applications where power consumption is a critical factor, both the HMOS and CHMOS versions provide reduced power modes of operation. For the CHMOS version of the 8051 the reduced power modes, Idle and Power Down, are standard features. In the HMOS versions a reduced power mode is available, but not as a standard feature. The local sales office will provide ordering information for users requiring this feature.

6.11.1 HMOS Power Down Mode

The power down mode in the HMOS devices allows one to reduce VCC to zero while saving the on-chip RAM through a backup supply connected to the RST pin. To use the feature, the user's system, upon detecting that a power failure is imminent, would interrupt the processor in some manner to transfer relevant data to the on-chip RAM and enable the backup power supply to the RST pin before VCC falls below its operating limit. When power returns, the backup supply needs to stay on long enough to accomplish a reset, and then can be removed so that normal operation can be resumed.

6.11.2 CHMOS Power Reduction Modes

CHMOS versions have two power-reducing modes, Idle and Power Down. The input through which backup power is supplied during these operations is VCC. Figure 6-27 shows the internal circuitry which implements these features. In the Idle mode (IDL = 1), the oscillator continues to run and the Interrupt, Serial Port, and Timer blocks continue to be clocked, but the clock signal is gated off to the CPU. In Power Down (PD = 1), the oscillator is frozen. The Idle and Power Down modes are activated by setting bits in Special Function Register PCON. The address of this register is 87H. Figure 6-28 details its contents.

(MSB) (LSB)

SMOD	—	—	—	GF1	GF0	PD	IDL

Symbol	Position	Name and Function
SMOD	PCON.7	Double Baud rate bit. When set to a 1, the baud rate is doubled when the serial port is being used in either modes 1, 2 or 3.
—	PCON.6	(Reserved)
—	PCON.5	(Reserved)
—	PCON.4	(Reserved)
GF1	PCON.3	General-purpose flag bit.
GF0	PCON.2	General-purpose flag bit.
PD	PCON.1	Power Down bit. Setting this bit activates power down operation.
IDL	PCON.0	Idle mode bit. Setting this bit activates idle mode operation.

If 1s are written to PD and IDL at the same time, PD takes precedence. The reset value of PCON is (0XXX0000).

Figure 6-28. PCON: Power Control Register

IDLE MODE

An instruction that sets PCON.0 causes that to be the last instruction executed before going into the Idle mode. In the Idle mode, the internal clock signal is gated off to the CPU, but not to the Interrupt, Timer, and Serial Port functions. The CPU status is preserved in its entirety: the Stack Pointer, Program Counter, Program Status Word, Accumulator, and all other registers maintain their data during Idle. The port pins hold the logical states they had at the time Idle was activated. ALE and $\overline{\text{PSEN}}$ go to their inactive levels.

There are two ways to terminate the Idle. Activation of any enabled interrupt will cause PCON.0 to be cleared by hardware, terminating the Idle mode. The interrupt will be serviced, and following RETI the next instruction to be executed will be the one following the instruction that put the device into Idle.

The flag bits GF0 and GF1 can be used to give an indication if an interrupt occurred during normal operation or

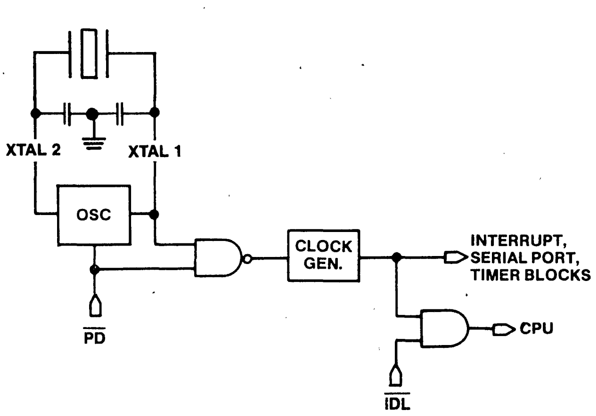


Figure 6-27. Idle and Power Down Hardware

during an Idle. For example, an instruction that activates Idle can also set one or both flag bits. When Idle is terminated by an interrupt, the interrupt service routine can examine the flag bits.

The other way of terminating the Idle mode is with a hardware reset. Since the clock oscillator is still running, the hardware reset needs to be held active for only two machine cycles (24 oscillator periods) to complete the reset.

POWER DOWN MODE

An instruction that sets PCON.1 causes that to be the last instruction executed before going into the Power Down mode. In the Power Down mode, the on-chip oscillator is stopped. With the clock frozen, all functions are stopped, but the on-chip RAM and Special Function Registers are held. The port pins output the values held by their respective SFRs. ALE and $\overline{\text{PSEN}}$ output lows.

The only exit from Power Down is a hardware reset. Reset redefines all the SFRs, but does not change the on-chip RAM.

In the Power down mode of operation, VCC can be reduced to minimize power consumption. Care must be taken, however, to ensure that VCC is not reduced before the Power Down mode is invoked, and that VCC is restored to its normal operating level, before the Power Down mode is terminated. The reset that terminates Power Down also frees the oscillator. The reset should not be activated before VCC is restored to its normal operating level, and must be held active long enough to allow the oscillator to restart and stabilize (normally less than 10 msec).

6.12 8751H

The 8751H is the EPROM member of the MCS-51 family. This means that the on-chip Program Memory can be electrically programmed, and can be erased by exposure to ultraviolet light. The 8751H also has a provision for denying external access to the on-chip Program Memory, in order to protect its contents against software piracy.

6.12.1 Programming the EPROM

To be programmed, the 8751H must be running with a 4 to 6 MHz oscillator. (The reason the oscillator needs to be running is that the internal bus is being used to transfer address and program data to appropriate internal registers.) The address of an EPROM location to be programmed is applied to Port 1 and pins P2.0–P2.3 of Port 2, while the data byte is applied to Port 0. Pins P2.4–P2.6 and $\overline{\text{PSEN}}$ should be held low, and P2.7 and RST high. (These are all TTL levels except RST, which requires 2.5V for a logic high.) $\overline{\text{EA}}$/VPP is held normally high, and is pulsed to +21V. While $\overline{\text{EA}}$/VPP is at 21V, the ALE/$\overline{\text{PROG}}$ pin, which is normally being held high, is pulsed low for 50 msec. Then $\overline{\text{EA}}$/VPP is returned to high. This setup is shown in Figure 6.29. Detailed timing specifications are provided in the 8751H data sheet.

Note: The $\overline{\text{EA}}$ pin must not be allowed to go above the maximum specified VPP level of 21.5V for any amount of time. Even a narrow glitch above that voltage level can cause permanent damage to the device. The VPP source should be well regulated and free of glitches.

6.12.2 Program Verification

If the program security bit has not been programmed, the on-chip Program Memory can be read out for verification

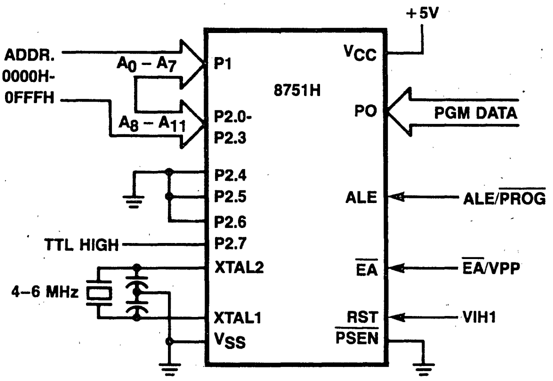


Figure 6-29. Programming the 8751H

purposes, if desired, either during or after the programming operation. The required setup, which is shown in Figure 6.30, is the same as for programming the EPROM except that pin P2.7 is held at TTL low (or used as an active-low read strobe). The address of the Program Memory location to be read is applied to Port 1 and pins P2.0–P2.3. The other Port 2 pins and PSEN are held low. ALE, EA, and RST are held high. The contents of the addressed location will come out on Port 0. External pullups are required on Port 0 for this operation.

6.12.3 Program Memory Security

The 8751H contains a security bit, which, once programmed, denies electrical access by any external means to the on-chip Program Memory. The setup and procedure for programming the security bit are the same as for normal programming, except that pin P2.6 is held at TTL high. The setup is shown in Figure 6.31. Port 0, Port 1, and pins P2.0–P2.3 of Port 2 may be in any state.

Once the security bit has been programmed, it can be deactivated only by full erasure of the Program Memory. While it is programmed, the internal Program Memory cannot be read out, the device cannot be further programmed, and it *cannot execute external program memory*. Erasing the EPROM, thus deactivating the security bit, restores the device's full functionality. It can then be re-programmed.

6.12.4 Erasure Characteristics

Erasure of the 8751H Program Memory begins to occur when the chip is exposed to light with wavelengths shorter

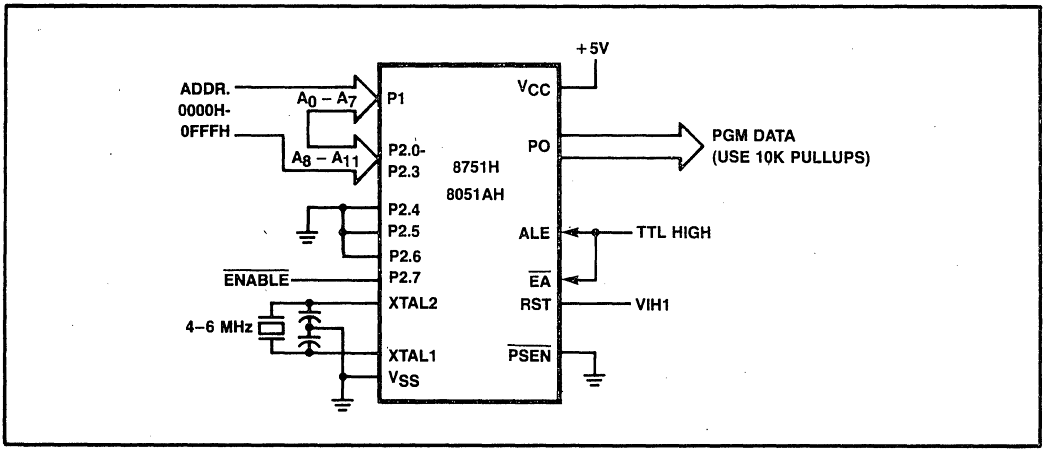


Figure 6-30. Program Verification in the 8751H and 8051AH

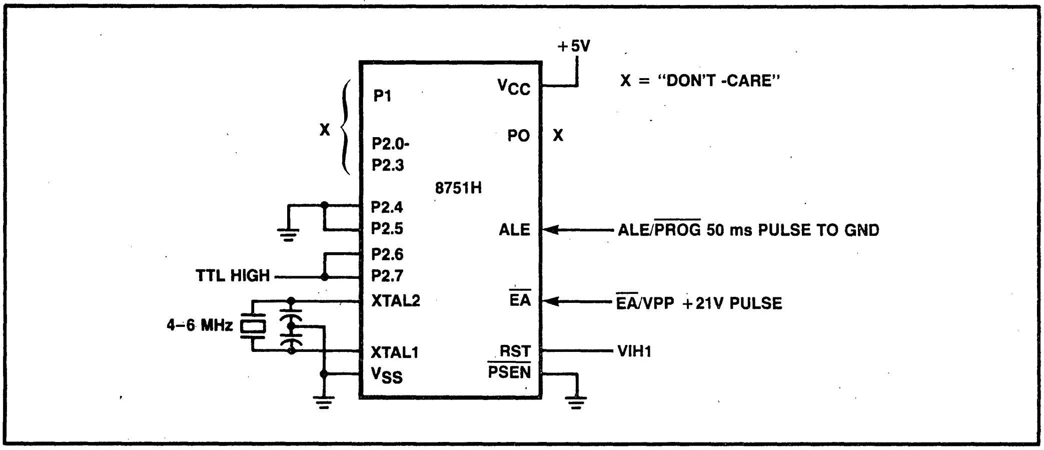


Figure 6-31. Programming the Security Bit in the 8751H

than approximately 4,000 Angstroms. Since sunlight and fluorescent lighting have wavelengths in this range, exposure to these light sources over an extended time (about 1 week in sunlight, or 3 years in room-level fluorescent lighting) could cause inadvertent erasure. If an application subjects the 8751H to this type of exposure, it is suggested that an opaque label be placed over the window.

The recommended erasure procedure is exposure to ultraviolet light (at 2537 Angstroms) to an integrated dose of at least 15 W/cm^2. Exposing the 8751H to an ultraviolet lamp of 12,000 μW/cm^2 rating for 20 to 30 minutes, at a distance of about 1 inch, should be sufficient.

Erasure leaves the array in an all 1s state.

6.13 MORE ABOUT THE ON-CHIP OSCILLATOR

6.13.1 HMOS Versions

The on-chip oscillator circuitry for the HMOS (HMOS-I and HMOS-II) members of the MCS-51 family is a single stage linear inverter (figure 6-32), intended for use as a crystal-controlled, positive reactance oscillator (Figure 6-33). In this application the crystal is operated in its fundamental response mode as an inductive reactance in parallel resonance with capacitance external to the crystal.

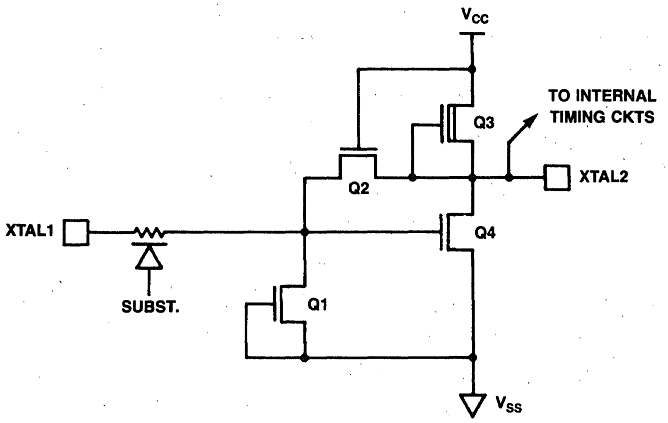


Figure 6-32. On-Chip Oscillator Circuitry in the HMOS Versions of the MCS-51 Family

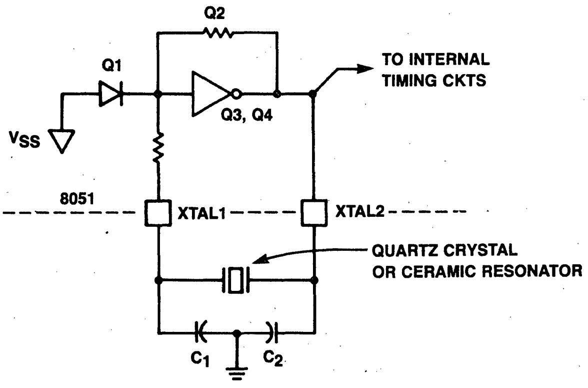


Figure 6-33. Using the HMOS On-Chip Oscillator

The crystal specifications and capacitance values (C1 and C2 in Figure 6-33) are not critical. 30 pF can be used in these positions at any frequency with good quality crystals. A ceramic resonator can be used in place of the crystal in cost-sensitive applications. When a ceramic resonator is used, C1 and C2 are normally selected to be of somewhat higher values, typically, 47 pF. The manufacturer of the ceramic resonator should be consulted for recommendations on the values of these capacitors.

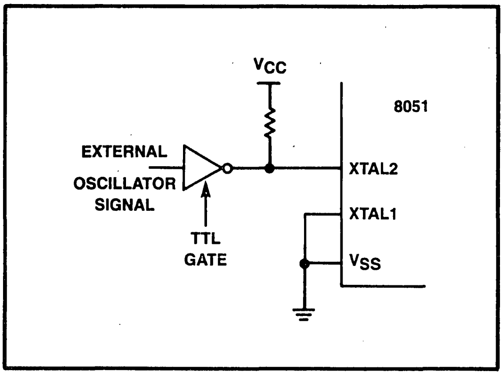


Figure 6-34. Driving the HMOS MCS-51 Parts with an External Clock Source

A more in-depth discussion of crystal specifications, ceramic resonators, and the selection of values for C1 and C2 can be found in Application Note AP-155, "Oscillators for Microcontrollers," which is included in this manual.

To drive the HMOS parts with an external clock source, apply the external clock signal to XTAL2, and ground XTAL1, as shown in Figure 6-34. A pull-up resistor is suggested because the logic levels at XTAL2 are not TTL.

6.13.2 CHMOS

The on-chip oscillator circuitry for the 80C51, shown in Figure 6-35, consists of a single stage linear inverter intended for use as a crystal-controlled, positive reactance oscillator in the same manner as the HMOS parts. However, there are some important differences.

One difference is that the 80C51 is able to turn off its oscillator under software control (by writing a 1 to the PD bit in PCON). Another difference is that in the 80C51 the internal clocking circuitry is driven by the signal at XTAL1, whereas in the HMOS versions it is by the signal at XTAL2.

The feedback resistor Rf in Figure 6-35 consists of paralleled n- and p-channel FETs controlled by the PD bit, such that Rf is opened when PD = 1. The diodes D1 and D2, which act as clamps to VCC and VSS, are parasitic to the Rf FETs.

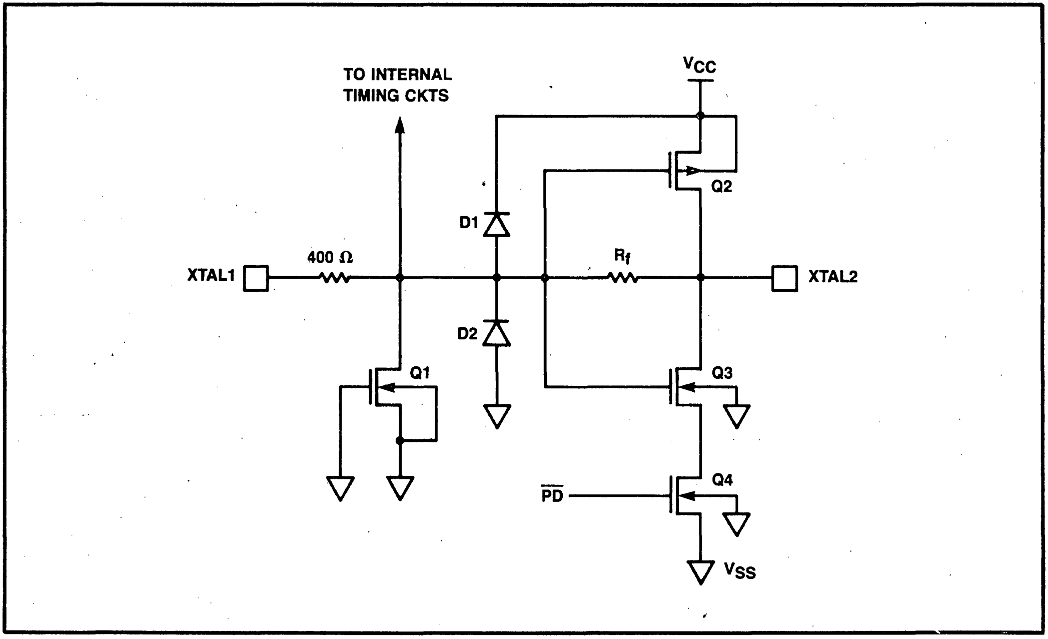


Figure 6-35. On-Chip Oscillator Circuitry in the CHMOS Versions of the MCS-51 Family

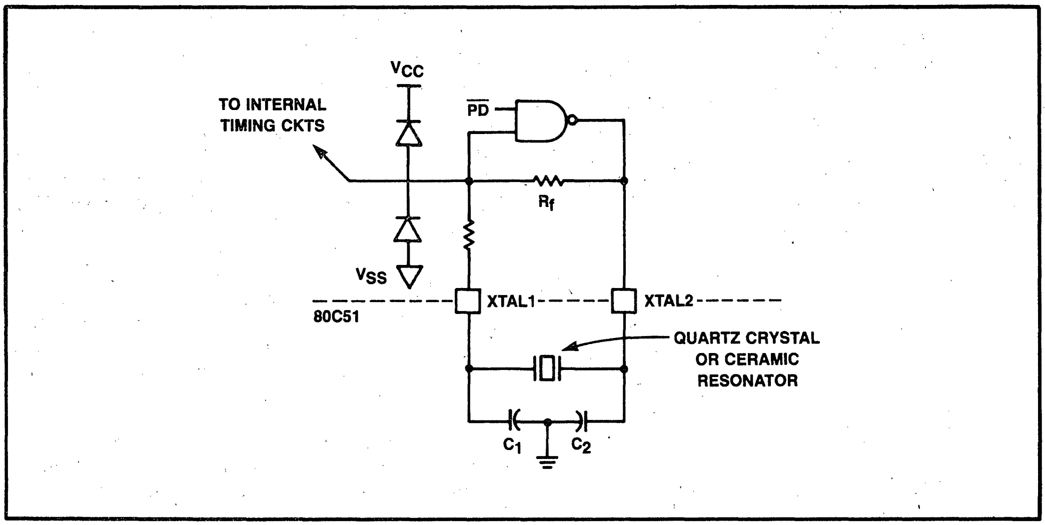


Figure 6-36. Using the CHMOS On-Chip Oscillator

The oscillator can be used with the same external components as the HMOS versions, as shown in Figure 6-36. Typically, C1 = C2 = 30 pF when the feedback element is a quartz crystal, and C1 = C2 = 47 pF when a ceramic resonator is used.

To drive the CHMOS parts with an external clock source, apply the external clock signal to XTAL1, and leave XTAL2 float, as shown in Figure 6-37.

6.14 MCS-51 PIN DESCRIPTIONS

VCC: Supply voltage.

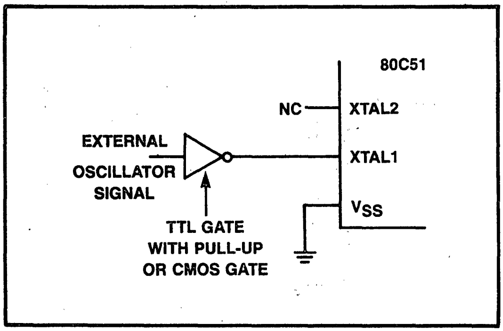

80C51
NC
XTAL2
EXTERNAL OSCILLATOR SIGNAL
XTAL1
VSS
TTL GATE WITH PULL-UP OR CMOS GATE

Figure 6-37. Driving the CHMOS MCS-51 Parts with an External Clock Source

VSS: Circuit ground potential.

Port 0: Port 0 is an 8-bit open drain bidirectional I/O port. As an open drain output port it can sink 8 LS TTL loads. Port 0 pins that have 1s written to them float, and in that state will function as high-impedance inputs. Port 0 is also the multiplexed low-order address and data bus during accesses to external memory. In this application it uses strong internal pullups when emitting 1s. Port 0 also emits code bytes during program verification. In that application, external pullups are required.

Port 1: Port 1 is an 8-bit bidirectional I/O port with internal pullups. The port 1 output buffers can sink/source 4 LS TTL loads. Port 1 pins that have 1s written to them are pulled high by the internal pullups, and in that state can be used as inputs. As inputs, Port 1 pins that are externally being pulled low will source current (IIL, on the data sheet) because of the internal pullups.

In the 8052, pins P1.0 and P1.1 also serve the alternate functions of T2 and T2EX. T2 is the Timer 2 external input. T2EX is the input through which a Timer 2 "capture" is triggered.

Port 2: Port 2 is an 8-bit bidirectional I/O port with internal pullups. The Port 2 output buffers can sink/source 4 LS TTL loads. Port 2 emits the high-order address byte during accesses to external memory that use 16-bit addresses. In this application it uses the strong internal pullups when emitting 1s. Port 2 also receives the high-order address and control bits during 8751H programming and verification, and during program verification in the 8051AH.

Port 3: Port 3 is an 8-bit bidirectional I/O port with internal pullups. It also serves the functions of various special features of the MCS-51 Family, as listed below:

PORT PIN	ALTERNATE FUNCTION
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	$\overline{\text{INT0}}$ (external interrupt 0)
P3.3	$\overline{\text{INT1}}$ (external interrupt 1)
P3.4	T0 (Timer 0 external input)
P3.5	T1 (Timer 1 external input)
P3.6	$\overline{\text{WR}}$ (external data memory write strobe)
P3.7	$\overline{\text{RD}}$ (external data memory read strobe)

The Port 3 output buffers can source/sink 4 LS TTL loads.

RST: Reset input. A high on this pin for two machine cycles while the oscillator is running resets the device.

ALE/$\overline{\text{PROG}}$: Address Latch Enable output pulse for latching the low byte of the address during accesses to external memory. ALE is emitted at a constant rate of 1/6 of the oscillator frequency, for external timing or clocking purposes, even when there are no accesses to external memory. (However, one ALE pulse is skipped during each acces to external **Data** Memory) This pin is also the program pulse input ($\overline{\text{PROG}}$) during EPROM programming.

$\overline{\text{PSEN}}$: Program Store Enable is the read strobe to external Program Memory. When the device is executing out of external Program Memory, $\overline{\text{PSEN}}$ is activated twice each machine cycle (except that two $\overline{\text{PSEN}}$ activations are skipped during accesses to external **Data** Memory). $\overline{\text{PSEN}}$ is not activated when the device is executing out of internal Program Memory.

$\overline{\text{EA}}$/VPP: When $\overline{\text{EA}}$ is held high the CPU executes out of internal Program Memory (unless the Program Counter exceeds 0FFFH in the 8051AH, or 1FFFH in the 8052). Holding $\overline{\text{EA}}$ low forces the CPU to execute out of external memory regardless of the Program Counter value. In the 8031AH and 8032, $\overline{\text{EA}}$ must be externally wired low. In the 8751H, this pin also receives the 21V programming supply voltage (VPP) during EPROM programming.

XTAL1: Input to the inverting oscillator amplifier.

XTAL2: Output from the inverting oscillator amplifier.

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

7

CHAPTER 7
MCS®-51 MEMORY ORGANIZATION, ADDRESSING MODES AND BOOLEAN PROCESSOR

7.0 INTRODUCTION

The MCS®-51 architecture provides on-chip memory as well as off-chip memory expansion capabilities. Several addressing mechanisms are incorporated to allow for an optimal instruction set.

7.1 MEMORY ORGANIZATION

The 8051 has three basic memory address spaces:

- 64K-byte Program Memory;
- 64K-byte External Data Program Memory; and
- 256-byte (384 bytes for the 8032/8052 Internal Data Memory

Figure 7-1 shows MCS®-51 memory maps.

PROGRAM MEMORY ADDRESS SPACE

The 64K-byte Program Memory space consists of an internal and an external memory portion. If the $\overline{EA}$ pin is held high, the 8051 executes out of internal program memory unless the address exceeds 0FFFH (1FFFH for the 8052). Locations 1000H through 0FFFFH (2000H through 0FFFFH for the 8052), are then fetched from external Program memory. If the $\overline{EA}$ pin is held low, the 8051 fetches all instructions from external Program Memory. In either case, the 16-bit Program Counter is the addressing mechanism.

Locations 00 through 23H (00 through 2BH for the 8032/8052) in Program Memory are used by interrupt service routines as indicated in Table 1.

DATA MEMORY ADDRESS SPACE

The Data Memory address space consists of an internal and an external memory space. External Data Memory is accessed when a MOVX instruction is executed.

Internal Data Memory is divided into three physically separate and distinct blocks: the lower 128 bytes of RAM; the upper 128 bytes of RAM (accessable in the 8032/8052 only); and the 128-byte Special Function Register (SFR) area. While the upper RAM area and the SFR area share the same address locations, they are accessed through different addressing modes. These modes are discussed in a later section.

Figure 7-2 shows a mapping of Internal Data Memory. Four 8-Register Banks occupy locations 0 through 31 in the lower RAM area. Only one of these banks may be enabled at a time (through a two-bit field in the PSW). The next sixteen bytes, locations 32 through 47 contain 128 bit addressable locations. Figure 7-3 shows the RAM bit addresses. The SFR area also has bit addressable locations. They are shown in Figure 7-4.

Note that reading from unused locations in Internal Data Memory will yield random data.

Table 1.

Source	Address
External Interrupt 0	0003H
Timer 0 Overflow	000BH
External Interrupt 1	0013H
Timer 1 Overflow	001BH
Serial Port	0023H
Timer 2 Overflow/TZEX Negative Transition	002BH

7.2 ADDRESSING MODES

The 8051 uses five addressing modes:

- Register;
- Direct;
- Register Indirect;
- Immediate; and
- Base-Register plus Index-Register Indirect.

Table 2 summarizes which memory spaces may be accessed by each of the addressing modes.

Table 2. Addressing Method and Associated Memory Spaces

Register Addressing

— R0-R7
— ACC, B, CY(bit), DPTR

Direct Addressing

— Lower 128 bytes of internal RAM
— Special Function Registers

Register Indirect Addressing

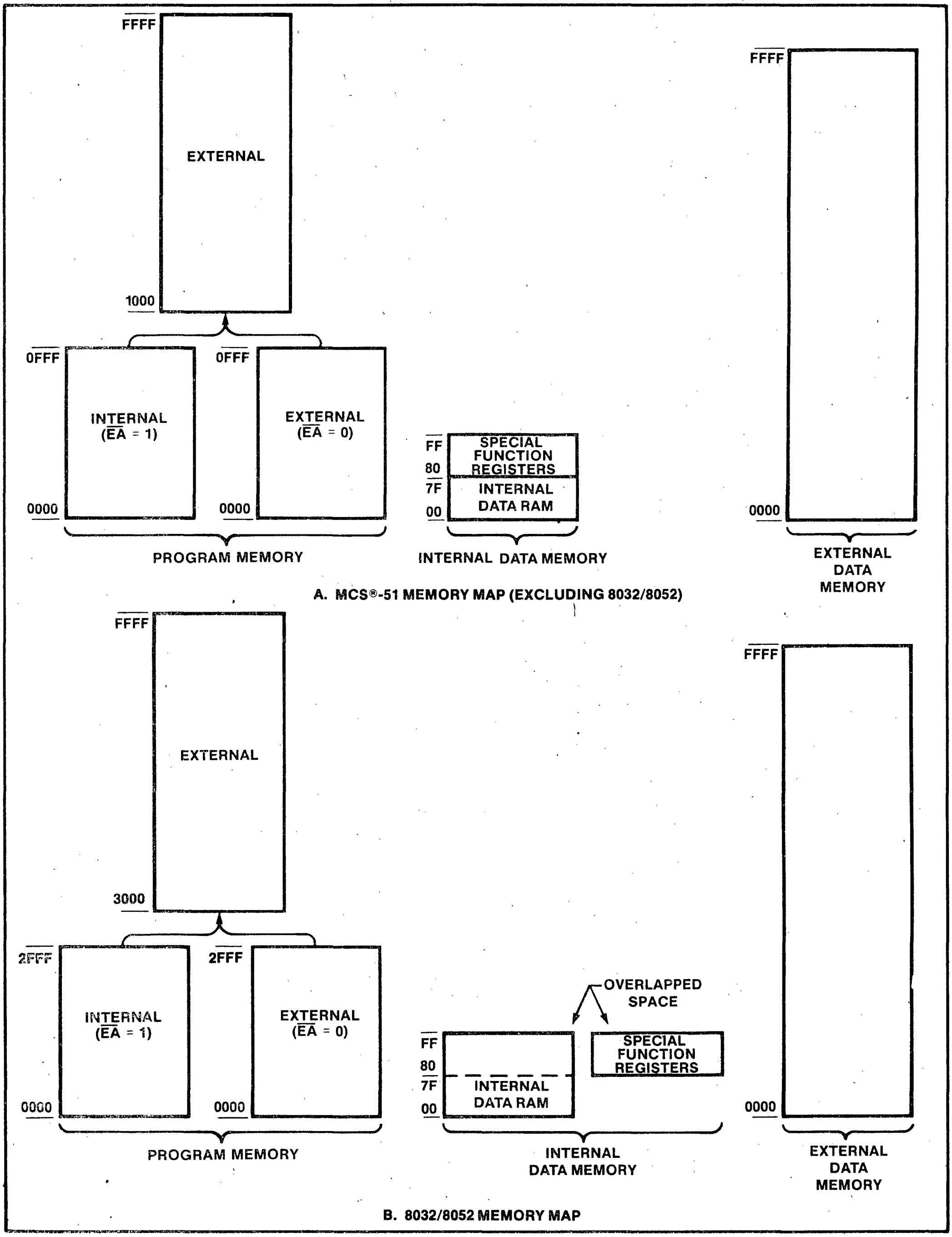
— Internal RAM (@R1, @R0, SP)
— External Data Memory (@R1, R0, @DPTR)

Immediate Addressing

— Program Memory

Base-Register plus Index-Register Indirect Addressing

— Program Memory (@DPTR + A, @ PC + A)



A. MCS®-51 MEMORY MAP (EXCLUDING 8032/8052)

B. 8032/8052 MEMORY MAP

Figure 7-1. MCS®-51 Memory Maps

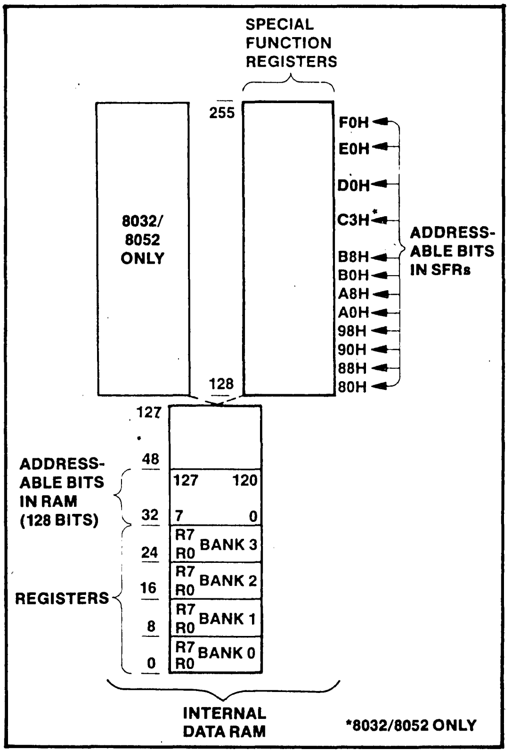


Figure 7-2. Internal Data Memory Address Space

REGISTER ADDRESSING

Register Addressing accesses the eight working registers (R0-R7) of the selected Register Bank. The least significant three bits of the instruction op code indicate which register is to be used. ACC, B, DPTR and CY, the Boolean Processor accumulator, can also be addressed as registers.

DIRECT ADDRESSING

Direct Addressing is the only method of accessing the Special Function Registers. The lower 128 bytes of Internal RAM are also directly addressable.

REGISTER-INDIRECT ADDRESSING

Register-Indirect Addressing uses the contents of either R0 or R1 (in the selected Register Bank) as a pointer to locations in a 256-byte block: the lower 128 bytes of internal RAM; the upper 128 bytes of internal RAM (8032/8052 only); or the lower 256 bytes of external Data Memory. Note that the Special Function Registers are not accessable by this method. Access to the full 64K external Data Memory address space is accomplished by using the 16-bit Data Pointer.

RAM Byte	(MSB)							(LSB)	
7FH									127
2FH	7F	7E	7D	7C	7B	7A	79	78	47
2EH	77	76	75	74	73	72	71	70	46
2DH	6F	6E	6D	6C	6B	6A	69	68	45
2CH	67	66	65	64	63	62	61	60	44
2BH	5F	5E	5D	5C	5B	5A	59	58	43
2AH	57	56	55	54	53	52	51	50	42
29H	4F	4E	4D	4C	4B	4A	49	48	41
28H	47	46	45	44	43	42	41	40	40
27H	3F	3E	3D	3C	3B	3A	39	38	39
26H	37	36	35	34	33	32	31	30	38
25H	2F	2E	2D	2C	2B	2A	29	28	37
24H	27	26	25	24	23	22	21	20	36
23H	1F	1E	1D	1C	1B	1A	19	18	35
22H	17	16	15	14	13	12	11	10	34
21H	0F	0E	0D	0C	0B	0A	09	08	33
20H	07	06	05	04	03	02	01	00	32
1FH									31
	Bank 3								
18H									24
17H									23
	Bank 2								
10H									16
0FH									15
	Bank 1								
08H									8
07H									7
	Bank 0								
00H									0

Figure 7-3. Special Function Bit Addressable Locations

Execution of PUSH and POP instructions also use Register-Indirect addressing. The Stack Pointer may reside anywhere in Internal RAM.

IMMEDIATE ADDRESSING

Immediate Addressing allows constants to be part of the op code instruction in Program Memory.

BASE-REGISTER PLUS INDEX REGISTER-INDIRECT ADDRESSING

Base-Register plus Index Register-Indirect Addressing allows a byte to be accessed from Program Memory via an indirect move from the location whose address is the sum of a base register (DPTR or PC) and index register, ACC. This mode facilitates look-up-table accesses.

7.3 BOOLEAN PROCESSOR

The Boolean Processor is an integrated bit processor within the 8051. It has its own instruction set, accumulator (the carry flag), and bit addressable RAM and I/O.

The bit-manipulation instructions allow a bit to be set, cleared, complimented, jump-if-set, jump-if-not-set, jump-if-set-then-cleared and moved to/from the carry. Addressable bits, or their compliments, may be logically ANDed or ORed with the contents of the carry flag. The result is returned to the carry register.

Direct Byte Address	Bit Addresses (MSB)							(LSB)	Hardware Register Symbol
240	F7	F6	F5	F4	F3	F2	F1	F0	B
224	E7	E6	E5	E4	E3	E2	E1	E0	ACC
	CY	AC	F0	RS1	RS0	OV		P	
208	D7	D6	D5	D4	D3	D2	D1	D0	PSW
	TF2	EXF2	RCLK	TCLK	EXEN2	TR2	$C/\overline{T2}$	$CP/\overline{RL2}$	
200	CF	CE	CD	CC	CB	CA	C9	C8	T2CON
			PT2	PS	PT1	PX1	PT0	PX0	
184	—	—	BD	BC	BB	BA	B9	B8	IP
176	B7	B6	B5	B4	B3	B2	B1	B0	P3
	EA		ET2	ES	ET1	EX1	ET0	EX0	
168	AF	—	AD	AC	AB	AA	A9	A8	IE
160	A7	A6	A5	A4	A3	A2	A1	A0	P2
	SM0	SM1	SM2	REN	TB8	RB8	TI	RI	
152	9F	9E	9D	9C	9B	9A	99	98	SCON
144	97	96	95	94	93	92	91	90	P1
	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	
136	8F	8E	8D	8C	8B	8A	89	88	TCON
128	87	86	85	84	83	82	81	80	P0

Figure 7-4. Special Function Register Bit Address

MCS®-51 Instruction Set 8

CHAPTER 8
MCS®-51 INSTRUCTION SET

8.0 INTRODUCTION

The MCS®-51 instruction set includes 111 instructions, 49 of which are single-byte, 45 two-byte and 17 three byte. The instruction op code format consists of a function mnemonic followed by a "destination, source" operand field. This field specifies the data type and addressing method(s) to be used.

8.1 FUNCTIONAL OVERVIEW

The MCS-51 instruction set is divided into four functional groups:

- Data Transfer
- Arithmetic
- Logic
- Control Transfer

8.1.1 Data Transfer

Data transfer operations are divided into three classes:

- General Purpose
- Accumulator-Specific
- Address-Object

None of these operations affect the PSW flag settings except a POP or MOV directly to the PSW.

GENERAL-PURPOSE TRANSFERS

- MOV performs a bit or a byte transfer from the source operand to the destination operand.

- PUSH increments the SP register and then transfers a byte from the source operand to the stack location currently addressed by SP.

- POP transfer a byte operand from the stack location addressed by SP to the destination operand and then decrements SP.

ACCUMULATOR SPECIFIC TRANSFERS

- XCH exchanges the byte source operand with register A (accumulator).

- XCHD exchanges the low-order nibble of the byte source operand with the low-order nibble of A.

- MOVX performs a byte move between the External Data Memory and the accumulator. The external address can be specified by the DPTR register (16-bit) or the R1 or R0 register (8-bit).

- MOVC moves a byte from Program memory to the accumulator. The operand in A is used as an index into a 256-byte table pointed to by the base register (DPTR or PC). The byte operand accessed is transferred to the accumulator.

ADDRESS-OBJECT TRANSFER

- MOV DPTR, #data loads 16-bits of immediate data into a pair of destination registers, DPH and DPL.

8.1.2 Arithmetic

The 8051 has four basic mathematical operations. Only 8-bit operations using unsigned arithmetic are supported directly. The overflow flag, however, permits the addition and subtraction operation to serve for both unsigned and signed binary integers. Arithmetic can also be performed directly on packed decimal (BCD) representations.

ADDITION

- INC (increment) adds one to the source operand and puts the result in the operand.

- ADD adds A to the source operand and returns the result to A.

- ADDC (add with Carry) adds A and the source operand, then adds one (1) if CY is set, and puts the result in A.

- DA (decimal-add-adjust for BCD addition) corrects the sum which results from the binary addition of two two-digit decimal operands. The packed decimal sum formed by DA is returned to A. CY is set if the BCD result is greater than 99; otherwise, it is cleared.

SUBTRACTION

- SUBB (subtract with borrow) subtracts the second source operand from the first operand (the accumulator), subtracts one (1) if CY is set and returns the result to A.

- DEC (decrement) subtracts one (1) from the source operand and returns the result to the operand.

MULTIPLICATION

- MUL performs an unsigned multiplication of the A register by the B register, returning a double-byte result. A receives the low-order byte, B receives the high-order byte. OV is cleared if the top half of the result is zero and is set if it is non-zero. CY is cleared. AC is unaffected.

DIVISION

- DIV performs an unsigned division of the A register by the B register and returns the integer quotient to A and returns the fractional remainder to the B register. Division by zero leaves indeterminate data in registers A and B and sets OV; otherwise OV is cleared. CY is cleared. AC is unaffected.

Unless otherwise stated in the above descriptions, the flags of PSW are affected as follows:

- CY is set if the operation causes a carry to or from the resulting high-order bit. Otherwise CY is cleared.
- AC is set if the operation results in a carry from the low-order four bits of the result (during addition), or a borrow from the high-order bits to the low-order bits (during subtraction); otherwise AC is cleared.
- OV is set if the operation results in a carry to the high-order bit of the result but not a carry from the high-order bit, or vice versa; otherwise OV is cleared. OV is used in two's-complement arithmetic, because it is set when the signed result cannot be represented in 8 bits.
- P is set if the modulo 2 sum of the eight bits in the accumulator is 1 (odd parity); otherwise P is cleared (even parity). When a value is written to the PSW register, the P bit remains unchanged, as it always reflects the parity of A.

8.1.3 Logic

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

SINGLE-OPERAND OPERATIONS

- CLR sets A or any directly addressable bit to zero (0).
- SETB sets any directly addressable bit to one (1).
- CPL is used to compliment the contents of the A register without affecting any flags, or any directly addressable bit location.
- RL, RLC, RR, RRC, SWAP are the five rotate operations that can be performed on A. RL, rotate left, RR, rotate right, RLC, rotate left through C, RRC, rotate right through C, and SWAP, rotate left four. For RLC and RRC the CY flag becomes equal to the last bit rotated out. SWAP rotates A left four places to exchange bits 3 through 0 with bits 7 through 4.

TWO-OPERAND OPERATIONS

- ANL performs bitwise logical and of with two source operands (for both bit and byte operands) and returns the result to the location of the first operand.
- ORL performs bitwise logical or of two source operands (for both bit and byte operands) and returns the result of the location of the first operand.
- XRL performs bitwise logical or of two source operands (byte operands) and returns the result to the location of the first operand.

8.1.4 Control Transfer

There are three classes of control transfer operations: unconditional calls, returns and jumps; conditional jumps; and interrupts. All control transfer operations cause, some upon a specific condition, the program execution to continue at a non-sequential location in program memory.

UNCONDITIONAL CALLS, RETURNS AND JUMPS

Unconditional calls, returns and jumps transfer control from the current value of the Program Counter to the target address. Both direct and indirect transfers are supported.

- ACALL and LCALL push the address of the next instruction onto the stack and then transfer control to the target address. ACALL is a 2-byte instruction used when the target address is in the current 2K page. LCALL is a 3-byte instruction that addresses the full 64K program space. In ACALL, immediate data (i.e. an 11 bit address field) is concatenated to the five most significant bits of the PC (which is pointing to the next instruction). If ACALL is in the last 2 bytes of a 2K page then the call will be made to the next page since the PC will have been incremented to the next instruction prior to execution.

- RET transfers control to the return address saved on the stack by a previous call operation and decrements the SP register by two (2) to adjust the SP for the popped address.
- AJMP, LJMP and SJMP transfer control to the target operand. The operation of AJMP and LJMP are analogous to ACALL and LCALL. The SJMP (short jump) instruction provides for transfers within a 256 byte range centered about the starting address of the next instruction (-128 to +127).
- JMP @A+DPTR performs a jump relative to the DPTR register. The operand in A is used as the offset (0-255) to the address in the DPTR register. Thus, the effective destination for a jump can be anywhere in the Program Memory space.

CONDITIONAL JUMPS

Conditional jumps perform a jump contingent upon a specific condition. The destination will be within a 256-byte range centered about the starting address of the next instruction (-128 to +127).

- JZ performs a jump if the accumulator is zero.
- JNZ performs a jump if the accumulator is not zero.
- JC performs a jump if the carry flag is set.
- JNC performs a jump if the carry flag is not set.
- JB performs a jump if the Direct Addressed bit is set.
- JNB performs a jump if the Direct Addressed bit is not set.
- JBC performs a jump if the Direct Addressed bit is set and then clears the Direct Addressed bit.
- CJNE compares the first operand to the second operand and performs a jump if they are not equal. CY is set if the first operand is less than the second operand; otherwise it is cleared. Comparisons can be made between A directly addressable bytes in Internal Data Memory or between an immediate value and either A, a register in the selected Register Bank, or a Register-Indirect addressed byte of Internal RAM.
- DJNZ decrements the source operand and returns the result to the operand. A jump is performed if the result is not zero. The source operand of the DJNZ instruction may be any byte in the Internal Data Memory. Either Direct or Register Addressing may be used to address the source operand.

INTERRUPT RETURNS

- RETI transfers control as does RET, but additionally enables interrupts of the current priority level.

8.2 INSTRUCTION DEFINITIONS

Each of the 51 basic MCS-51 operations, ordered alphabetically according to the operation mnemonic are described beginning page 8-8.

A brief example of how the instruction might be used is given as well as its effect on the PSW flags. The number of bytes and machine cycles required, the binary machine-language encoding, and a symbolic description or restatement of the function is also provided.

Note: Only the carry, auxiliary-carry, and overflow flags are discussed. The parity bit is computed after every instruction cycle that alters the accumulator. Similarly, instructions which alter directly addressed registers could affect the other status flags if the instruction is applied to the PSW. Status flags can also be modified by bit-manipulation.

For details on the MCS-51 assembler, ASM51, refer to the MCS-51 Macro Assembler User's Guide, publication number 9800937.

Table 8-1 summarized the MCS-51 instruction set.

Table 8-1. 8051 Instruction Set Summary

Interrupt Response Time: To finish execution of current instruction, respond to the interrupt request, push the PC and to vector to the first instruction of the interrupt service program requires 38 to 81 oscillator periods (3 to 7μs @ 12 MHz).

INSTRUCTIONS THAT AFFECT FLAG SETTINGS[1]

INSTRUCTION	FLAG			INSTRUCTION	FLAG		
	C	OV	AC		C	OV	AC
ADD	X	X	X	CLR C	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.

direct —8-bit internal data location's address. This could be an Internal Data RAM location (0-127) or a SFR [i.e., I/O port, control register, status register, etc. (128-255)].

@Ri —8-bit internal data RAM location (0-255) addressed indirectly through register R1 or R0.

#data —8-bit constant included in instruction.

#data 16 —16-bit constant included in instruction

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

addr 11 —11-bit destination address. Used by ACALL & AJMP. The branch will be within the same 2K-byte page of program memory as the first byte of the following instruction.

rel —Signed (two's complement) 8-bit offset byte. Used by SJMP and all conditional jumps. Range is -128 to +127 bytes relative to first byte of the following instruction.

bit —Direct Addressed bit in Internal Data RAM or Special Function Register.

* —New operation not provided by 8048AH/8049AH.

ARITHMETIC OPERATIONS

Mnemonic		Description	Byte	Oscillator Period
ADD	A,Rn	Add register to Accumulator	1	12
ADD	A,direct	Add direct byte to Accumulator	2	12
ADD	A,@Ri	Add indirect RAM to Accumulator	1	12
ADD	A,#data	Add immediate data to Accumulator	2	12
ADDC	A,Rn	Add register to Accumulator with Carry	1	12
ADDC	A,direct	Add direct byte to Accumulator with Carry	2	12
ADDC	A,@Ri	Add indirect RAM to Accumulator with Carry	1	12
ADDC	A,#data	Add immediate data to Acc with Carry	2	12
SUBB	A,Rn	Subtract register from Acc with borrow	1	12
SUBB	A,direct	Subtract direct byte from Acc with borrow	2	12

ARITHMETIC OPERATIONS Cont.

Mnemonic		Description	Byte	Oscillator Period
SUBB	A,@Ri	Subtract indirect RAM from Acc with borrow	1	12
SUBB	A,#data	Subtract immediate data from Acc with borrow	2	12
INC	A	Increment Accumulator	1	12
INC	Rn	Increment register	1	12
INC	direct	Increment direct byte	2	12
INC	@Ri	Increment indirect RAM	1	12
DEC	A	Decrement Accumulator	1	12
DEC	Rn	Decrement Register	1	12
DEC	direct	Decrement direct byte	2	12
DEC	@Ri	Decrement indirect RAM	1	12
INC	DPTR	Increment Data Pointer	1	24
MUL	AB	Multiply A & B	1	48
DIV	AB	Divide A by B	1	48
DA	A	Decimal Adjust Accumulator	1	12

Table 8-1. 8051 Instruction Set Summary (Continued)

LOGICAL OPERATIONS

Mnemonic		Description	Byte	Oscillator Period
ANL	A,Rn	AND register to Accumulator	1	12
ANL	A,direct	AND direct byte to Accumulator	2	12
ANL	A,@Ri	AND indirect RAM to Accumulator	1	12
ANL	A,#data	AND immediate data to Accumulator	2	12
ANL	direct,A	AND Accumulator to direct byte	2	12
ANL	direct,#data	AND immediate data to direct byte	3	24
ORL	A,Rn	OR register to Accumulator	1	12
ORL	A,direct	OR direct byte to Accumulator	2	12
ORL	A,@Ri	OR indirect RAM to Accumulator	1	12
ORL	A,#data	OR immediate data to Accumulator	2	12
ORL	direct,A	OR Accumulator to direct byte	2	12
ORL	direct,#data	OR immediate data to direct byte	3	24
XRL	A,Rn	Exclusive-OR register to Accumulator	1	12
XRL	A,direct	Exclusive-OR direct byte to Accumulator	2	12

LOGICAL OPERATIONS Cont.

Mnemonic		Description	Byte	Oscillator Period
XRL	A,@Ri	Exclusive-OR indirect RAM to Accumulator	1	12
XRL	A,#data	Exclusive-OR immediate data to Accumulator	2	12
XRL	direct,A	Exclusive-OR Accumulator to direct byte	2	12
XRL	direct,#data	Exclusive-OR immediate data to direct byte	3	24
CLR	A	Clear Accumulator	1	12
CPL	A	Complement Accumulator	1	12
RL	A	Rotate Accumulator Left	1	12
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 8-1. 8051 Instruction Set Summary (Continued)

DATA TRANSFER

Mnemonic		Description	Byte	Oscillator Period
MOV	A,Rn	Move register to Accumulator	1	12
MOV	A,direct	Move direct byte to Accumulator	2	12
MOV	A,@Ri	Move indirect RAM to Accumulator	1	12
MOV	A,#data	Move immediate data to Accumulator	2	12
MOV	Rn,A	Move Accumulator to register	1	12
MOV	Rn,direct	Move direct byte to register	2	24
MOV	Rn,#data	Move immediate data to register	2	12
MOV	direct,A	Move Accumulator to direct byte	2	12
MOV	direct,Rn	Move register to direct byte	2	24
MOV	direct,direct	Move direct byte to direct	3	24
MOV	direct,@Ri	Move indirect RAM to direct byte	2	24
MOV	direct,#data	Move immediate data to direct byte	3	24
MOV	@Ri,A	Move Accumulator to indirect RAM	1	12
MOV	@Ri,direct	Move direct byte to indirect RAM	2	24
MOV	@Ri,#data	Move immediate data to indirect RAM	2	12

DATA TRANSFER Cont.

Mnemonic		Description	Byte	Oscillator Period
MOV	DPTR,#data16	Load Data Pointer with a 16-bit constant	3	24
MOVC	A,@A+DPTR	Move Code byte relative to DPTR to Acc	1	24
MOVC	A,@A+PC	Move Code byte relative to PC to Acc	1	24
MOVX	A,@Ri	Move External RAM (8-bit addr) to Acc	1	24
MOVX	A,@DPTR	Move External RAM (16-bit addr) to Acc	1	24
MOVX	@Ri,A	Move Acc to External RAM (8-bit addr)	1	24
MOVX	@DPTR,A	Move Acc to External RAM (16-bit addr)	1	24
PUSH	direct	Push direct byte onto stack	2	24
POP	direct	Pop direct byte from stack	2	24
XCH	A,Rn	Exchange register with Accumulator	1	12
XCH	A,direct	Exchange direct byte with Accumulator	2	12
XCH	A,@Ri	Exchange indirect RAM with Accumulator	1	12
XCHD	A,@Ri	Exchange low-order Digit indirect RAM with Acc	1	12

Table 8-1. 8051 Instruction Set Summary (Continued)

BOOLEAN VARIABLE MANIPULATION

				Oscillator
Mnemonic		Description	Byte	Period
CLR	C	Clear Carry	1	12
CLR	bit	Clear direct bit	2	12
SETB	C	Set Carry	1	12
SETB	bit	Set direct bit	2	12
CPL	C	Complement Carry	1	12
CPL	bit	Complement direct bit	2	12
ANL	C,bit	AND direct bit to Carry	2	24
ANL	C,/bit	AND complement of direct bit to Carry	2	24
ORL	C,bit	OR direct bit to Carry	2	24
ORL	C,/bit	OR complement of direct bit to Carry	2	24
MOV	C,bit	Move direct bit to Carry	2	12
MOV	bit,C	Move Carry to direct bit	2	24
JC	rel	Jump if Carry is set	2	24
JNC	rel	Jump if Carry not set	2	24
JB	bit,rel	Jump if direct Bit is set	3	24
JNB	bit,rel	Jump if direct Bit is Not set	3	24
JBC	bit,rel	Jump if direct Bit is set & clear bit	3	24

PROGRAM BRANCHING

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

PROGRAM BRANCHING Cont.

				Oscillator
Mnemonic		Description	Byte	Period
RETI		Return from interrupt	1	24
AJMP	addr11	Absolute Jump	2	24
LJMP	addr16	Long Jump	3	24
SJMP	rel	Short Jump (relative addr)	2	24
JMP	@A+DPTR	Jump indirect relative to the DPTR	1	24
JZ	rel	Jump if Accumulator is Zero	2	24
JNZ	rel	Jump if Accumulator is Not Zero	2	24
CJNE	A,direct,rel	Compare direct byte to Acc and Jump if Not Equal	3	24
CJNE	A,#data,rel	Compare immediate to Acc and Jump if Not Equal	3	24
CJNE	Rn,#data,rel	Compare immediate to register and Jump If Not Equal	3	24
CJNE	@Ri,#data,rel	Compare immediate to indirect and Jump if Not Equal	3	24
DJNZ	Rn,rel	Decrement register and Jump if Not Zero	3	24
DJNZ	direct,rel	Decrement direct byte and Jump if Not Zero	3	24
NOP		No Operation	1	12

ACALL addr11

Function: Absolute Call

Description: ACALL unconditionally calls a subroutine located at the indicated address. The instruction increments the PC twice to obtain the address of the following instruction, then pushes the 16-bit result onto the stack (low-order byte first) and increments the stack pointer twice. The destination address is obtained by successively concatenating the five high-order bits of the incremented PC, 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

$(PC) \leftarrow (PC) + 2$
$(SP) \leftarrow (SP) + 1$
$((SP)) \leftarrow (PC_{7-0})$
$(SP) \leftarrow (SP) + 1$
$((SP)) \leftarrow (PC_{15-8})$
$(PC_{10-0}) \leftarrow$ 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

$(A) \leftarrow (A) + (C) + (\text{direct})$

ADDC A,@Ri

Bytes: 1

Cycles: 1

Encoding:

0	0	1	1	0	1	1	i

Operation: ADDC

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

ADDC A,#data

Bytes: 2

Cycles: 1

Encoding:

0	0	1	1	0	1	0	0	immediate data

Operation: ADDC

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

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

$(PC) \leftarrow (PC) + 2$

$(PC_{10\text{-}0}) \leftarrow$ 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

$(A) \leftarrow (A) \wedge (Rn)$

ANL A,direct

Bytes: 2

Cycles: 1

Encoding:

0 1 0 1	0 1 0 1	direct address

Operation: ANL
(A) ← (A) ∧ (direct)

ANL A,@Ri

Bytes: 1

Cycles: 1

Encoding:

0 1 0 1	0 1 1 i

Operation: ANL
(A) ← (A) ∧ ((Ri))

ANL A,#data

Bytes: 2

Cycles: 1

Encoding:

0 1 0 1	0 1 0 0	immediate data

Operation: ANL
(A) ← (A) ∧ #data

ANL direct,A

Bytes: 2

Cycles: 1

Encoding:

0 1 0 1	0 0 1 0	direct address

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

ANL direct,#data

Bytes: 3

Cycles: 2

Encoding:

0 1 0 1	0 0 1 1	direct address	immediate data

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

$(C) \leftarrow (C) \wedge (bit)$

ANL C,/bit

Bytes: 2

Cycles: 2

Encoding:

1 0 1 1	0 0 0 0	bit address

Operation: ANL

$(C) \leftarrow (C) \wedge \neg (bit)$

CJNE <dest-byte>,<src-byte>, rel

Function: Compare and Jump if Not Equal.

Description: CJNE compares the magnitudes of the first two operands, and branches if their values are not equal. The branch destination is computed by adding the signed relative-displacement in the last instruction byte to the PC, after incrementing the PC to the start of the next instruction. The carry flag is set if the unsigned integer value of <dest-byte> is less than the unsigned integer value of <src-byte>; otherwise, the carry is cleared. Neither operand is affected.

The first two operands allow four addressing mode combinations: the accumulator may be compared with any directly addressed byte or immediate data, and any indirect RAM location or working register can be compared with an immediate constant.

Example: The accumulator contains 34H. Register 7 contains 56H. The first instruction in the sequence,

```
           CJNE   R7,#60H, NOT_EQ
;          . . .  . . . . .            ; R7 = 60H.
NOT_EQ:    JC     REQ_LOW              ; IF R7 < 60H.
;          . . .  . . . . .            ; R7 > 60H.
```

sets the carry flag and branches to the instruction at label NOT_EQ. By testing the carry flag, this instruction determines whether R7 is greater or less than 60H.

If the data being presented to port 1 is also 34H, then the instruction,

```
WAIT:  CJNE   A,P1,WAIT
```

clears the carry flag and continues with the next instruction in sequence, since the accumulator does equal the data read from P1. (If some other value was being input on P1, the program will loop at this point until the P1 data changes to 34H.)

CJNE A,direct,rel

Bytes: 3

Cycles: 2

Encoding:

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

Operation:

```
CJNE
(PC) ← (PC) + 3
 IF (direct) < (A)
   THEN  (PC) ← (PC) + rel and (C) ← 0
              OR
 IF (direct) < (A)
   THEN (PC) ← (PC) + rel and (C) ← 1
```

CJNE A,#data,rel

Bytes: 3

Cycles: 2

Encoding:

1 0 1 1	0 1 0 0	immediate data	rel. address

Operation:

```
CJNE
(PC) ← (PC) + 3
 IF #data < (A)
    THEN (PC) ← (PC) + rel and (C) ← 0
          OR
 IF #data > (A)
    THEN (PC) ← (PC) + rel and (C) ← 1
```

CJNE Rn,#data,rel

Bytes: 3

Cycles: 2

Encoding:

1 0 1 1	1 r r r	immediate data	rel. address

Operation:

```
CJNE
(PC) ← (PC) + 3
 IF #data < (Rn)
    THEN (PC) ← (PC) + rel and (C) ← 0
          OR
 IF #data > (Rn)
    THEN (PC) ← (PC) + rel and (C) ← 1
```

CJNE @Ri,#data,rel

Bytes: 3

Cycles: 2

Encoding:

1 0 1 1	0 1 1 i	immediate data	rel. address

Operation:

```
CJNE
(PC) ← (PC) + 3
 IF #data < ((Ri))
    THEN (PC) ← (PC) + rel and (C) ← 1
          OR
 IF #data > ((Ri))
    THEN (PC) ← (PC) + rel and (C) ← 0
```

CLR A

Function: Clear Accumulator

Description: The accumulator is cleared (all bits set to zero). No flags are affected.

Example: The accumulator contains 5CH (01011100B). The instruction,

CLR A

will leave the accumulator set to 00H (00000000B).

Bytes: 1

Cycles: 1

Encoding:

1	1	1	0	0	1	0	0

Operation: CLR
(A)←0

CLR bit

Function: Clear bit

Description: The indicated bit is cleared (reset to zero). No other flags are affected. CLR can operate on the carry flag or any directly addressable bit.

Example: Port 1 has previously been written with 5DH (01011101B). The instruction,

CLR P1.2

will leave the port set to 59H (01011001B).

CLR C

Bytes: 1

Cycles: 1

Encoding:

1	1	0	0	0	0	1	1

Operation: CLR
(C)←0

CLR bit

Bytes: 2

Cycles: 1

Encoding:

1	1	0	0	0	0	1	0	bit address

Operation: CLR
(bit)←0

CPL A

Function: Complement Accumulator

Description: Each bit of the accumulator is logically complemented (one's complement). Bits which previously contained a one are changed to zero and vice-versa. No flags are affected.

Example: The accumulator contains 5CH (01011100B). The instruction,

```
CPL     A
```

will leave the accumulator set to 0A3H (10100011B).

Bytes: 1

Cycles: 1

Encoding:

1 1 1 1	0 1 0 0

Operation: CPL

$(A) \leftarrow \neg (A)$

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

$(C) \leftarrow \neg (C)$

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  [[$(A_{3-0}) > 9$] ∨ [$(AC) = 1$]]
    THEN  $(A_{3-0}) \leftarrow (A_{3-0}) + 6$
              AND
IF  [[$(A_{7-4}) > 9$] ∨ [$(C) = 1$]]
    THEN  $(A_{7-4}) \leftarrow (A_{7-4}) + 6$
```

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

$$\begin{matrix}(A)_{15-8}\\(B)_{7-0}\end{matrix} \leftarrow (A) / (B)$$

DJNZ <byte>,<rel-addr>

Function: Decrement and Jump if Not Zero

Description: DJNZ decrements the location indicated by 1, and branches to the address indicated by the second operand if the resulting value is not zero. An original value of 00H will underflow to 0FFH. No flags are affected. The branch destination would be computed by adding the signed relative-displacement value in the last instruction byte to the PC, after incrementing the PC to the first byte of the following instruction.

The location decremented may be a register or directly addressed byte.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

Example: Internal RAM locations 40H, 50H, and 60H contain the values 01H, 70H, and 15H, respectively. The instruction sequence,

```
DJNZ    40H,LABEL_1
DJNZ    50H,LABEL_2
DJNZ    60H,LABEL_3
```

will cause a jump to the instruction at label LABEL_2 with the values 00H, 6FH, and 15H in the three RAM locations. The first jump was *not* taken because the result was zero.

This instruction provides a simple way of executing a program loop a given number of times, or for adding a moderate time delay (from 2 to 512 machine cycles) with a single instruction. The instruction sequence,

```
          MOV   R2,#8
TOGGLE:   CPL   P1.7
          DJNZ  R2,TOGGLE
```

will toggle P1.7 eight times, causing four output pulses to appear at bit 7 of output port 1. Each pulse will last three machine cycles; two for DJNZ and one to alter the pin.

DJNZ Rn,rel

Bytes: 2
Cycles: 2

Encoding:

1 1 0 1	1 r r r	direct address

Operation:

```
DJNZ
(PC) ← (PC) + 2
(Rn) ← (Rn) − 1
IF (Rn) > 0 or (Rn) < 0
    THEN
        (PC) ← (PC) + rel
```

DJNZ direct,rel

Bytes: 3
Cycles: 2

Encoding:

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

Operation:

```
DJNZ
(PC) ← (PC) + 2
(direct) ← (direct) − 1
IF (direct) > 0 or (direct) < 0
    THEN
        (PC) ← (PC) + rel
```

INC <byte>

Function: Increment

Description: INC increments the indicated variable by 1. An original value of 0FFH will overflow to 00H. No flags are affected. Three addressing modes are allowed: register, direct, or register-indirect.

Note: When this instruction is used to modify an output port, the value used as the original port data will be read from the output data latch, *not* the input pins.

Example: Register 0 contains 7EH (011111110B). Internal RAM locations 7EH and 7FH contain 0FFH and 40H, respectively. The instruction sequence,

```
INC   @R0
INC   R0
INC   @R0
```

will leave register 0 set to 7FH and internal RAM locations 7EH and 7FH holding (respectively) 00H and 41H.

INC A

Bytes: 1

Cycles: 1

Encoding:

0 0 0 0	0 1 0 0

Operation: INC

(A) ← (A) + 1

INC Rn

Bytes: 1

Cycles: 1

Encoding:

0 0 0 0	1 r r r

Operation: INC

(Rn) ← (Rn) + 1

INC direct

Bytes: 2

Cycles: 1

Encoding:

0 0 0 0	0 1 0 1	direct address

Operation: INC

(direct) ← (direct) + 1

INC @Ri

Bytes: 1

Cycles: 1

Encoding:

0 0 0 0	0 1 1 i

Operation: INC

((Ri))←((Ri)) + 1

INC DPTR

Function: Increment Data Pointer

Description: Increment the 16-bit data pointer by 1. A 16-bit increment (modulo 2^{16}) is performed; an overflow of the low-order byte of the data pointer (DPL) from 0FFH to 00H will increment the high-order byte (DPH). No flags are affected.

This is the only 16-bit register which can be incremented.

Example: Registers DPH and DPL contain 12H and 0FEH, respectively. The instruction sequence,

```
INC    DPTR
INC    DPTR
INC    DPTR
```

will change DPH and DPL to 13H and 01H.

Bytes: 1

Cycles: 2

Encoding:

1 0 1 0	0 0 1 1

Operation: INC

(DPTR)←(DPTR) + 1

JB bit,rel

Function: Jump if Bit set

Description: If the indicated bit is a one, jump to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. *The bit tested is not modified.* No flags are affected.

Example: The data present at input port 1 is 11001010B. The accumulator holds 56 (01010110B). The instruction sequence,

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

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

Bytes: 3

Cycles: 2

Encoding:

0 0 1 0	0 0 0 0	bit address	rel. address

Operation:

```
JB
(PC) ← (PC) + 3
IF (bit) = 1
    THEN
        (PC) ← (PC) + rel
```

JBC bit,rel

Function: Jump if Bit is set and Clear bit

Description: If the indicated bit is one, branch to the address indicated; otherwise proceed with the next instruction. *In either case, clear the designated bit.* The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. No flags are affected.

Note: When this instruction is used to test an output pin, the value used as the original data will be read from the output data latch, *not* the input pin.

Example: The accumulator holds 56H (01010110B). The instruction sequence,

```
JBC    ACC.3,LABEL1
JBC    ACC.2,LABEL2
```

will cause program execution to continue at the instruction identified by the label LABEL2, with the accumulator modified to 52H (01010010B).

Bytes: 3

Cycles: 2

Encoding:

0 0 0 1	0 0 0 0	bit address	rel. address

Operation:

```
JBC
(PC) ← (PC) + 3
IF (bit) = 1
    THEN
        (bit) ← 0
        (PC) ← (PC) + rel
```

JC rel

Function: Jump if Carry is set

Description: If the carry flag is set, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. No flags are affected.

Example: The carry flag is cleared. The instruction sequence,

```
JC    LABEL1
CPL   C
JC    LABEL2
```

will set the carry and cause program execution to continue at the instruction identified by the label LABEL2.

Bytes: 2

Cycles: 2

Encoding:

0 1 0 0	0 0 0 0	rel. address

Operation:

```
JC
(PC) ← (PC) + 2
IF (C) = 1
    THEN
        (PC) ← (PC) + rel
```

JMP @A+DPTR

Function: Jump indirect

Description: Add the eight-bit unsigned contents of the accumulator with the sixteen-bit data pointer, and load the resulting sum to the program counter. This will be the address for subsequent instruction fetches. Sixteen-bit addition is performed (modulo 2^{16}): a carry-out from the low-order eight bits propagates through the higher-order bits. Neither the accumulator nor the data pointer is altered. No flags are affected.

Example: An even number from 0 to 6 is in the accumulator. The following sequence of instructions will branch to one of four AJMP instructions in a jump table starting at JMP_TBL:

```
            MOV     DPTR,#JMP_TBL
            JMP     @A+DPTR
JMP_TBL:    AJMP    LABEL0
            AJMP    LABEL1
            AJMP    LABEL2
            AJMP    LABEL3
```

If the accumulator equals 04H when starting this sequence, execution will jump to label LABEL2. Remember that AJMP is a two-byte instruction, so the jump instructions start at every other address.

Bytes: 1

Cycles: 2

Encoding:

0 1 1 1	0 0 1 1

Operation: JMP

(PC)←(A) + (DPTR)

JNB bit,rel

Function: Jump if Bit Not set

Description: If the indicated bit is a zero, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the third instruction byte to the PC, after incrementing the PC to the first byte of the next instruction. *The bit tested is not modified.* No flags are affected.

Example: The data present at input port 1 is 11001010B. The accumulator holds 56H (01010110B). The instruction sequence,

```
JNB    P1.3,LABEL1
JNB    ACC.3,LABEL2
```

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

Bytes: 3

Cycles: 2

Encoding:

0 0 1 1	0 0 0 0	bit address	rel. address

Operation:

```
JNB
(PC)←(PC) + 3
IF (bit) = 0
   THEN (PC)← (PC) + rel.
```

JNC rel

Function: Jump if Carry not set

Description: If the carry flag is a zero, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice to point to the next instruction. The carry flag is not modified.

Example: The carry flag is set. The instruction sequence,

```
JNC  LABEL1
CPL  C
JNC  LABEL2
```

will clear the carry and cause program execution to continue at the instruction identified by the label LABEL2.

Bytes: 2

Cycles: 2

Encoding:

0 1 0 1	0 0 0 0		rel. address

Operation:

```
JNC
(PC)←(PC) + 2
IF (C) = 0
    THEN (PC)←(PC) + rel
```

JNZ rel

Function: Jump if accumulator Not Zero

Description: If any bit of the accumulator is a one, branch to the indicated address; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The accumulator is not modified. No flags are affected.

Example: The accumulator originally holds 00H. The instruction sequence,

```
JNZ  LABEL1
INC  A
JNZ  LABEL2
```

will set the accumulator to 01H and continue at label LABEL2.

Bytes: 2

Cycles: 2

Encoding:

0 1 1 1	0 0 0 0		rel. address

Operation:

```
JNZ
(PC)← (PC) + 2
IF (A) ≠0
    THEN (PC)←(PC) + rel
```

JZ rel

Function: Jump if Accumulator Zero

Description: If all bits of the accumulator are zero, branch to the address indicated; otherwise proceed with the next instruction. The branch destination is computed by adding the signed relative-displacement in the second instruction byte to the PC, after incrementing the PC twice. The accumulator is not modified. No flags are affected.

Example: The accumulator originally contains 01H. The instruction sequence,

```
JZ     LABEL1
DEC    A
JZ     LABEL2
```

will change the accumulator to 00H and cause program execution to continue at the instruction identified by the label LABEL2.

Bytes: 2

Cycles: 2

Encoding:

0 1 1 0	0 0 0 0	rel. address

Operation:

```
JZ
(PC)◄—(PC) + 2
IF (A) = 0
    THEN (PC)◄—(PC) + rel
```

LCALL addr16

Function: Long Call

Description: LCALL calls a subroutine located at the indicated address. The instruction adds three to the program counter to generate the address of the next instruction and then pushes the 16-bit result onto the stack (low byte first), incrementing the stack pointer by two. The high-order and low-order bytes of the PC are then loaded, respectively, with the second and third bytes of the LCALL instruction. Program execution continues with the instruction at this address. The subroutine may therefore begin anywhere in the full 64K-byte program memory address space. No flags are affected.

Example: Initially the stack pointer equals 07H. The label "SUBRTN" is assigned to program memory location 1234H. After executing the instruction,

LCALL SUBRTN

at location 0123H, the stack pointer will contain 09H, internal RAM locations 08H and 09H will contain 26H and 01H, and the PC will contain 1235H.

Bytes: 3

Cycles: 2

Encoding:

0 0 0 1	0 0 1 0	addr15 - addr8	addr7 - addr0

Operation: LCALL

$(PC) \leftarrow (PC) + 3$

$(SP) \leftarrow (SP) + 1$

$((SP)) \leftarrow (PC_{7-0})$

$(SP) \leftarrow (SP) + 1$

$((SP)) \leftarrow (PC_{15-8})$

$(PC) \leftarrow addr_{15-0}$

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

$(PC) \leftarrow addr_{15-0}$

MOV <dest-byte>,<src-byte>

Function: Move byte variable

Description: The byte variable indicated by the second operand is copied into the location specified by the first operand. The source byte is not affected. No other register or flag is affected.

This is by far the most flexible operation. Fifteen combinations of source and destination addressing modes are allowed.

Example: Internal RAM location 30H holds 40H. The value of RAM location 40H is 10H. The data present at input port 1 is 11001010B (0CAH).

```
MOV  R0,#30H          ;R0<= 30H
MOV  A,@R0            ;A <= 40H
MOV  R1,A             ;R1 <= 40H
MOV  R,@R1            ;B <= 10H
MOV  @R1,P1           ;RAM (40H) <= 0CAH
MOV  P2, P1           ;P2 #0CAH
```

leaves the value 30H in register 0, 40H in both the accumulator and register 1, 10H in register B, and 0CAH (11001010B) both in RAM location 40H and output on port 2.

MOV A,Rn

Bytes: 1

Cycles: 1

Encoding:

1 1 1 0	1 r r r

Operation: MOV
(A)← (Rn)

*MOV A,direct

Bytes: 2

Cycles: 1

Encoding:

1 1 1 0	0 1 0 1	direct address

Operation: MOV
(A)← (direct)

***MOV A,ACC is not a valid instruction.**

MOV A,@Ri

Bytes: 1
Cycles: 1

Encoding:

1 1 1 0	0 1 1 i

Operation: MOV
(A) ← ((Ri))

MOV A,#data

Bytes: 2
Cycles: 1

Encoding:

0 1 1 1	0 1 0 0	immediate data

Operation: MOV
(A) ← #data

MOV Rn,A

Bytes: 1
Cycles: 1

Encoding:

1 1 1 1	1 r r r

Operation: MOV
(Rn) ← (A)

MOV Rn,direct

Bytes: 2
Cycles: 2

Encoding:

1 0 1 0	1 r r r	direct addr.

Operation: MOV
(Rn) ← (direct)

MOV Rn,#data

Bytes: 2
Cycles: 1

Encoding:

0 1 1 1	1 r r r	immediate data

Operation: MOV
(Rn) ← #data

MOV direct,A

Bytes: 2

Cycles: 1

Encoding:

1 1 1 1	0 1 0 1		direct address

Operation: MOV
(direct) ← (A)

MOV direct,Rn

Bytes: 2

Cycles: 2

Encoding:

1 0 0 0	1 r r r		direct address

Operation: MOV
(direct) ← (Rn)

MOV direct,direct

Bytes: 3

Cycles: 2

Encoding:

1 0 0 0	0 1 0 1		dir. addr. (src)		dir. addr. (dest)

Operation: MOV
(direct) ← (direct)

MOV direct,@Ri

Bytes: 2

Cycles: 2

Encoding:

1 0 0 0	0 1 1 i		direct addr.

Operation: MOV
(direct) ← ((Ri))

MOV direct,#data

Bytes: 3

Cycles: 2

Encoding:

0 1 1 1	0 1 0 1		direct address		immediate data

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
$(DPTR) \leftarrow \#data_{15\text{-}0}$
$DPH \ \square \ DPL \leftarrow \#data_{15\text{-}8} \ \square \ \#data_{7\text{-}0}$

MOVC A,@A + <base-reg>

Function: Move Code byte

Description: The MOVC instructions load the accumulator with a code byte, or constant from program memory. The address of the byte fetched is the sum of the original unsigned eight-bit accumulator contents and the contents of a sixteen-bit base register, which may be either the data pointer or the PC. In the latter case, the PC is incremented to the address of the following instruction before being added with the accumulator: otherwise the base register is not altered. Sixteen-bit addition is performed so a carry-out from the low-order eight bits may propagate through higher-order bits. No flags are affected.

Example: A value between 0 and 3 is in the accumulator. The following instructions will translate the value in the accumulator to one of four values defined by the DB (define byte) directive.

```
REL_PC:  INC       A
         MOVC      A,@A+PC
         RET
         DB        66H
         DB        77H
         DB        88H
         DB        99H
```

If the subroutine is called with the accumulator equal to 01H, it will return with 77H in the accumulator. The INC A before the MOVC instruction is needed to "get around" the RET instruction above the table. If several bytes of code separated the MOVC from the table, the corresponding number would be added to the accumulator instead.

MOVC A,@A + DPTR

Bytes: 1

Cycles: 2

Encoding:

1 0 0 1	0 0 1 1

Operation: MOVC

(A) ← ((A) + (DPTR))

MOVC A,@A+PC

Bytes: 1

Cycles: 2

Encoding:

1	0	0	0	0	0	1	1

Operation: MOVC
(PC)←(PC) + 1
(A)←((A) + (PC))

MOVX <dest-byte>,<src-byte>

Function: Move External

Description: The MOVX instructions transfer data between the accumulator and a byte of external data memory, hence the "X" appended to MOV. There are two types of instructions, differing in whether they provide an eight-bit or sixteen-bit indirect address to the external data RAM.

In the first type, the contents of R0 or R1 in the current register bank provide an eight-bit address multiplexed with data on P0. Eight bits are sufficient for external I/O expansion decoding or a relatively small RAM array. For somewhat larger arrays, any output port pins can be used to output higher-order address bits. These pins would be controlled by an output instruction preceding the MOVX.

In the second type of MOVX instruction, the data pointer generates a sixteen-bit address. P2 outputs the high-order eight address bits (the contents of DPH) while P0 multiplexes the low-order eight bits (DPL) with data. The P2 Special Function Register retains its previous contents while the P2 output buffers are emitting the contents of DPH. This form is faster and more efficient when accessing very large data arrays (up to 64K bytes), since no additional instructions are needed to set up the output ports.

It is possible in some situations to mix the two MOVX types. A large RAM array with its high-order address lines driven by P2 can be addressed via the data pointer, or with code to output high-order address bits to P2 followed by a MOVX instruction using R0 or R1.

Example: An external 256 byte RAM using multiplexed address/data lines (e.g., an Intel® 8155 RAM/I/O/Timer) is connected to the 8051 Port 0. Port 3 provides control lines for the external RAM. Ports 1 and 2 are used for normal I/O. Registers 0 and 1 contain 12H and 34H. Location 34H of the external RAM holds the value 56H. The instruction sequence,

```
MOVX    A,@R1
MOVX    @R0,A
```

copies the value 56H into both the accumulator and external RAM location 12H.

MOVX A,@Ri

Bytes: 1
Cycles: 2

Encoding:

1 1 1 0	0 0 1 i

Operation: MOVX
(A)◄— ((Ri))

MOVX A,@DPTR

Bytes: 1
Cycles: 2

Encoding:

1 1 1 0	0 0 0 0

Operation: MOVX
(A)◄— ((DPTR))

MOVX @Ri,A

Bytes: 1
Cycles: 2

Encoding:

1 1 1 1	0 0 1 i

Operation: MOVX
((Ri))◄— (A)

MOVX @DPTR,A

Bytes: 1
Cycles: 2

Encoding:

1 1 1 1	0 0 0 0

Operation: MOVX
(DPTR)◄— (A)

MUL AB

Function: Multiply

Description: MUL AB multiplies the unsigned eight-bit integers in the accumulator and register B. The low-order byte of the sixteen-bit product is left in the accumulator, and the high-order byte in B. If the product is greater than 255 (0FFH) the overflow flag is set; otherwise it is cleared. The carry flag is always cleared.

Example: Originally the accumulator holds the value 80 (50H). Register B holds the value 160 (0A0H). The instruction,

```
MUL     AB
```

will give the product 12,800 (3200H), so B is changed to 32H (00110010B) and the accumulator is cleared. The overflow flag is set, carry is cleared.

Bytes: 1

Cycles: 4

Encoding:

1 0 1 0	0 1 0 0

Operation: MUL

$(A)_{7-0} \leftarrow (A) \text{ X } (B)$

$(B)_{15-8}$

NOP

Function: No Operation

Description: Execution continues at the following instruction. Other than the PC, no registers or flags are affected.

Example: It is desired to produce a low-going output pulse on bit 7 of port 2 lasting exactly 5 cycles. A simple SETB/CLR sequence would generate a one-cycle pulse, so four additional cycles must be inserted. This may be done (assuming no interrupts are enabled) with the instruction sequence,

```
CLR     P2.7
NOP
NOP
NOP
NOP
SETB    P2.7
```

Bytes: 1

Cycles: 1

Encoding:

0 0 0 0	0 0 0 0

Operation: NOP

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

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:

0	1	0	0	0	1	0	1	direct address

Operation: ORL
(A) ← (A) ∨ (direct)

ORL A,@Ri

Bytes: 1

Cycles: 1

Encoding:

0	1	0	0	0	1	1	i

Operation: ORL
(A) ← (A) ∨ ((Ri))

ORL A,#data

Bytes: 2

Cycles: 1

Encoding:

0	1	0	0	0	1	0	0	immediate data

Operation: ORL
(A) ← (A) ∨ #data

ORL direct,A

Bytes: 2

Cycles: 1

Encoding:

0	1	0	0	0	0	1	0	direct address

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

ORL direct,#data

Bytes: 3

Cycles: 2

Encoding:

0	1	0	0	0	0	1	1	direct addr.	immediate data

Operation: ORL
(direct) ← (direct) ∨ #data

ORL C, <src-bit>

Function: Logical-OR for bit variables

Description Set the carry flag if the Boolean value is a logical 1; 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
$(C) \leftarrow (C) \vee (bit)$

ORL C,/bit

Bytes: 2

Cycles: 2

Encoding:

1 0 1 0	0 0 0 0	bit address

Operation: ORL
$(C) \leftarrow (C) \vee (\overline{bit})$

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

$(A_n) \leftarrow (A_{n+1})$ $n = 0-6$

$(A7) \leftarrow (A0)$

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

$(A_n) \leftarrow (A_{n+1})$ $n = 0-6$

$(A7) \leftarrow (C)$

$(C) \leftarrow (A0)$

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

(A3-0) ⇄ (A7-4), (A7-4) ← (A3-0)

XCH A,<byte>

Function: Exchange Accumulator with byte variable

Description: XCH loads the accumulator with the contents of the indicated variable, at the same time writing the original accumulator contents to the indicated variable. The source/destination operand can use register, direct, or register-indirect addressing.

Example: R0 contains the address 20H. The accumulator holds the value 3FH (00111111B). Internal RAM location 20H holds the value 75H (01110101B). The instruction,

XCH A,@R0

will leave RAM location 20H holding the values 3FH (00111111B) and 75H (01110101B) in the accumulator.

XCH A,Rn

Bytes: 1

Cycles: 1

Encoding:

1 1 0 0	1 r r r

Operation: XCH
$(A) \rightleftarrows (R_n)$

XCH A,direct

Bytes: 2

Cycles: 1

Encoding:

1 1 0 0	0 1 0 1		direct address

Operation: XCH
$(A) \rightleftarrows (\text{direct})$

XCH A,@Ri

Bytes: 1

Cycles: 1

Encoding:

1 1 0 0	0 1 1 i

Operation: XCH
$(A) \rightleftarrows ((R_i))$

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

MCS®-51 Application Examples

9

CHAPTER 9
MCS®-51 APPLICATION EXAMPLES

Chapter 9 is divided into three sections:

- 8051 Programming Techniques
- Peripheral Interfacing Techniques
- Peripheral Interface Examples

The first section has 8051 software examples for common routines in controller applications. Some of the routines are multiple-precision arithmetic and table look-up techniques.

Peripheral Interfacing Techniques include routines for handling the 8051's I/O ports, serial channel and timer/counters. Discussed in this section is I/O port reconfiguration, software delay timing, and transmitting serial port character strings along with other routines.

The Peripheral Interface examples consist of block diagram schematics.

9.0 8051 PROGRAMMING TECHNIQUES

9.0.1 Radix Conversion Routines

The divide instruction can be used to convert a number from one radix to another. BINBCD is a short subroutine to convert an eight-bit unsigned binary integer in the accumulator (between 0 & 255) to a three-digit (two byte) BCD representation. The hundred's digit is returned in one variable (HUND) and the ten's and one's digits returned as packed BCD in another (TENONE).

```
;
; BINBCD  CONVERT 8-BIT BINARY VARIABLE IN ACCUMULATOR
;         TO 3-DIGIT PACKED BCD FORMAT.
;         HUNDREDS' PLACE LEFT IN VARIABLE 'HUND',
          TENS' AND ONES' PLACES IN 'TENONE'.
;
HUND      DATA   21H
TENONE    DATA   22H
;
BINBCD:   MOV    B,#100            ;DIVIDED BY 100 TO
          DIV    AB                ;DETERMINE NUMBER OF HUNDREDS
          MOV    HUND,A
          MOV    A,#10             ;DIVIDE REMAINDER BY TEN TO
          XCH    A,B               ;DETERMINE NUMBER OF TENS LEFT
          DIV    AB                ;TEN'S DIGIT IN ACC, REMAINDER IS
                                   ;ONE'S DIGIT
          SWAP   A
          ADD    A,B               ;PACK BCD DIGITS IN ACC
          MOV    TENONE,A
          RET
;
```

The divide instruction can also separate data in the accumulator into sub-fields. For example, dividing packed BCD data by 16 will separate the two nibbles, leaving the high-order digit in the accumulator and the low-order digit (remainder) in B. Each is right-justified, so the digits can be processed individually. This example receives two packed BCD digits in the accumulator, separates the digits, computes their product, and returns the product in packed BCD format in the accumulator.

```
;
;MULBCD  UNPACK TWO BCD DIGITS RECEIVED IN ACCUMULATOR,
;        FIND THEIR PRODUCT, AND RETURN PRODUCT
;        IN PACKED BCD FORMAT IN ACCUMULATOR
;
MULBCD: MOV   B,#10H           ;DIVIDE INPUT BY 16
        DIV   AB               ;A & B HOLD SEPARATED DIGITS
                               ;(EACH RIGHT JUSTIFIED IN REGISTER).
        MUL   AB               ;A HOLDS PRODUCT IN BINARY FORMAT (0-
                               ;99 (DECIMAL) = 0 - 63H)
        MOV   B,#10            ;DIVIDE PRODUCT BY 10
        DIV   AB               ;A HOLDS NUMBER OF TENS, B HOLDS
                               ;REMAINDER
        SWAP  A
        ORL   A,B              ;PACK DIGITS
        RET
```

9.0.2 Multiple Precision Arithmetic

The ADDC and SUBB instructions incorporate the previous state of the carry (borrow) flag to allow multiple-precision calculations by repeating the operation with successively higher-order operand bytes. If the input data for a multiple-precision operation is an unsigned string of integers, the carry flag will be set upon completion if an overflow (for ADDC) or underflow (for SUBB) occurs. With two's complement signed data, the most significant bit of the original input data's most significant byte indicates the sign of the string, so the overflow flag (OV) will indicate if overflow or underflow occurred.

```
;
;SUBSTR  SUBTRACT STRING INDICATED BY R1
;        FROM STRING INDICATED BY R0 TO
;        PRECISION INDICATED BY R2.
;        CHECK FOR SIGNED UNDERFLOW WHEN DONE.
;
SUBSTR: CLR   C                ;BORROW = 0.
```

```
SUBS1:   MOV    A,@R0             ;LOAD MINUEND BYTE
         SUBB   A,@R1             ;SUBTRACT SUBTRAHEND BYTE
         MOV    @R0,A             ;STORE DIFFERENCE BYTE
         INC    R0                ;BUMP POINTERS TO NEXT PLACE
         INC    R1
         DJNZ   R2,SUBS1          ;LOOP UNTIL DONE
;
;        WHEN DONE, TEST IF OVERFLOW OCCURRED
;        ON LAST ITERATION OF LOOP.
;
         JNB    OV,OV_OK
;        ...    ........          (OVERFLOW RECOVERY ROUTINE)
         OV_OK: RET               ;RETURN
```

9.0.3 Table Look-Up Sequences

The two versions of the MOVC instructions are used as part of a three-step sequence to access look-up tables in ROM. To use the DPTR version, load the Data Pointer with the starting address of a look-up table; load the accumulator with (or compute) the index of the entry desired; and execute MOVC A,@A+DPTR. The data pointer may be loaded with a constant for short tables, or to allow more complicated data structures, and tables with more than 256 entries, the values for DPH and DPL may be computed or modified with the standard arithmetic instruction set.

The PC-based version is used with smaller, "local" tables, and has the advantage of not affecting the data pointer. This makes it useful in interrupt routines or other situations where the DPTR contents might be significant. Again, a look-up sequence takes three steps: load the accumulator with the index; compensate for the offset from the look-up instruction's address to the start of the table by adding that offset to the accumulator; then execute the MOVC A,@A+PC instruction.

As a non-trivial situation where this instruction would be used, consider applications which store large multi-dimensional look-up tables of dot matrix patterns, non-linear calibration parameters, and so on in the linear (one-dimensional) program memory. To retrieve data from the tables, variables representing matrix indices must be converted to the desired entry's memory address. For a matrix of dimensions (MDIMEN x NDIMEN) starting at address BASE and respective indices INDEXI and INDEXJ, the address of element (INDEXI, INDEXJ) is determined by the formula,

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

The subroutine MATRIX1 can access an entry in any array with less than 255 elements (e.g., an 11 x 21 array with 231 elements). The table entries are defined using the Data Byte ("DB") directive, and will be contained in the assembly object code as part of the accessing subroutine itself.

To handle the more general case, subroutine MATRX2 allows tables to be unlimited in size, by combining the MUL instruction, double-precision addition, and the data pointer-based version of MOVC. The only restriction is that each index be between 0 and 255.

```
;
;MATRX1  LOAD CONSTANT READ FROM TWO DIMENSIONAL LOOK-UP
;        TABLE IN PROGRAM MEMORY INTO ACCUMULATOR
;        USING LOCAL TABLE LOOK-UP INSTRUCTION, 'MOVC A,@A+PC'.
;        THE TOTAL NUMBER OF TABLE ENTRIES IS ASSUMED TO
;        BE SMALL, I.E., LESS THAN ABOUT 255 ENTRIES.
;        TABLE USED IN THIS EXAMPLE IS 11 x 21.
;        DESIRED ENTRY ADDRESS IS GIVEN BY THE FORMULA,
;
;        [(BASE ADDRESS) = (21 X INDEXI) = (INDEXJ)]
;
INDEXI   EQU   R6              ;FIRST COORDINATE OF ENTRY (0-10).
INDEXJ   DATA  23H             ;SECOND COORDINATE OF ENTRY (0-20).
;
MATRX1:  MOV   A,INDEXI
         MOV   B, #21
         MUL   AB              ;(21 X INDEXI)
         ADD   A,INDEXJ        ;ADD IN OFFSET WITHIN ROW
;
;        ALLOW FOR INSTRUCTION BYTE BETWEEN "MOVC" AND
;        ENTRY (0,0).
;
         INC   A
         MOVC  A,@A+PC
         RET
BASE1:   DB    1               ;(entry 0,0)
         DB    2               ;(entry 0,1)
;        ..    ......
         DB    21              ;(entry 0,20)
         DB    22              ;(entry 1,0)
;        ..    ......
         DB    42              ;(entry 1,20)
;        ..    ......
;        ..    ......
;        DB    231             ;(entry 10,20)
```

```
MATRX2: MOV    A,INDEXI          ;LOAD FIRST COORDINATE
        MOV    B,#NDIMEN
        MUL    AB                :INDEXI X NDIMEN
        ADD    A,#LOW(BASE2)     ;ADD IN 16-BIT BASE ADDRESS
        MOV    DPL,A
        MOV    A,B
        ADDC   A,#HIGH(BASE2)
        MOV    DPH,A             ;DPTR=(BASE ADDR) + (INDEXI + NDIMEN)
        MOV    A,INDEXJ
        MOVC   A,@A+DPTR         ;ADD INDEXJ AND FETCH BYTE
        RET
        ...    .....
BASE2:  DB     0                 ;(entry 0,0)
        DB     0                 ;(entry 0,1)
;       ...    ......
        DB     0                 ;(entry 0, NDIMEN-1)
        DB     0                 ;(entry 1,0)
;       ...    ......
        DB     0                 ;(entry 1, NDIMEN-1)
;       ...    ......
;       ...    ......
        DB     0                 ;(entry MDIMEN-1, NDIMEN-1)
```

9.0.4 Saving CPU Status during Interrupts

When the 8051 hardware recognizes an interrupt request, program control branches automatically to the corresponding service routine, by forcing the CPU to process a Long CALL (LCALL) instruction to the appropriate address. The return address is stored on the top of the stack. After completing the service routine, an RETI instruction returns the processor to the background program at the point from which it was interrupted.

Interrupt service routines must not change any variable or hardware registers modified by the main program, or else the program may not resume correctly. (Such a change might look like a spontaneous random error. An example of this will be given later in this section, in the second method of I/O port reconfiguration.) Resources used or altered by the service routine (Accumulator, PSW, etc.) must be saved and restored to their previous value before returning from the service routine. PUSH and POP provide an efficient and convenient way to save such registers on the stack.

```
;
LOC_TMP  EQU    $                  ;REMEMBER  LOCATION  COUNTER
;
         ORG    0003H              ;STARTING ADDRESS FOR INTERRUPT ROUTINE

         LJMP   SERVER             ;JUMP TO ACTUAL SERVICE ROUTINE LOCATE
                                   ;ELSEWHERE
;        ...    ......
         ORG    LOC__TMP           ;RESTORE  LOCATION  COUNTER
SERVER:  PUSH   PSW
         PUSH   ACC                ;SAVE  ACCUMULATOR (NOTE DIRECT ADDRESS
                                   ;NOTATION)
         PUSH   B                  ;SAVE  B  REGISTER
         PUSH   DPL                :SAVE  DATA  POINTER
         PUSH   DPH                ;
         MOV    PSW,#00001000B     ;SELECT  REGISTER  BANK  1
;        ...    ......
;        ...    ......
         POP    DPH                ;RESTORE  REGISTERS  IN  REVERSE  ORDER
         POP    DPL
         POP    B
         POP    ACC
         POP    PSW                ;RESTORE  PSW  AND  RE-SELECT  ORIGINAL
                                   ;REGISTER  BANK
         RETI                      ;RETURN  TO  MAIN  PROGRAM  AND  RESTORE
                                   ;INTERRUPT  LOGIC
```

If the SP register held 1FH when the interrupt was detected, then while the service routine was in progress the stack would hold the registers shown in Figure 9-1; SP would contain 26H. This is the most general case; if the service routine doesn't alter the B-register and data pointer, for example, the instructions saving and restoring those registers could be omitted.

9.0.5 Passing Parameters on the Stack

The stack may also pass parameters to and from subroutines. The subroutine can indirectly address the parameters derived from the contents of the stack pointer, or simply pop the stack into registers before processing.

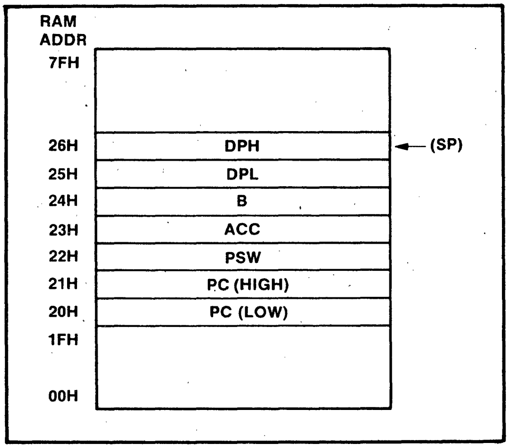


Figure 9-1.

```
HEXASC: MOV   R0,SP
        DEC   R0          ;ACCESS LOCATION PARAMETER PUSHED
                           INTO
        DEC   R0
        XCH   A,@R0       ;READ INPUT PARAMETER AND SAVE AC-
                           CUMULATOR
        ANL   A,#0FH      ;MASK ALL BUT LOW-ORDER 4 BITS
        ADD   A,#2        ;ALLOW FOR OFFSET FROM MOVC TO TABLE
        MOVC  A,@A+PC     ;READ LOOK-UP TABLE ENTRY
        XCH   A,@R0       ;PASS BACK TRANSLATED VALUE AND
                          ;RESTORE ACCUMULATOR
        RET               ;RETURN TO BACKGROUND PROGRAM
ASCTBL: DB    '0'         ;ASCII CODE FOR 00H
        DB    '1'         ;ASCII CODE FOR 01H
        DB    '2'         ;ASCII CODE FOR 02H
        DB    '3'         ;ASCII CODE FOR 03H
        DB    '4'         ;ASCII CODE FOR 04H
        DB    '5'         ;ASCII CODE FOR 05H
        DB    '6'         ;ASCII CODE FOR 06H
        DB    '7'         ;ASCII CODE FOR 07H
        DB    '8'         ;ASCII CODE FOR 08H
        DB    '9'         ;ASCII CODE FOR 09H
        DB    'A'         ;ASCII CODE FOR 0AH
        DB    'B'         ;ASCII CODE FOR 0BH
        DB    'C'         ;ASCII CODE FOR 0CH
        DB    'D'         ;ASCII CODE FOR 0DH
        DB    'E'         ;ASCII CODE FOR 0EH
        DB    'F'         ;ASCII CODE FOR 0FH
```

One advantage here is simplicity. Variables need not be allocated for specific parameters, a potentially large number of parameters may be passed, and different calling programs may use different techniques for determining or handling the variables.

For example, the subroutine HEXASC converts a hexadecimal value to ASCII code for its low-order digit. It first reads a parameter stored on the stack by calling program, then uses the low-order bits to access a local 16-entry look-up table holding ASCII codes, stores the appropriate code back in the stack and then returns. The accumulator contents are left unchanged.

The background program may reach this subroutine with several different calling sequences, all of which PUSH a value before calling the routine and POP the result to any destination register or port later. There is even the option of leaving a value on the stack if it won't be needed until later. The example below converts the three-digit BCD value computed in the Radix Conversion example above to a three-character string, calling a subroutine SP_OUT to output an eight-bit code in the accumulator.

```
;       ...     ......
        PUSH    HUND
        CALL    HEXASC          ;CONVERT HUNDREDS DIGIT
        POP     ACC
        CALL    SP_OUT          ;TRANSMIT HUNDREDS CHARACTER
        PUSH    TENONE
        CALL    HEXASC          ;CONVERT ONE'S PLACE DIGIT
                                ;BUT LEAVE ON STACK!
        MOV     A, TENONE
        SWAP    A               ;RIGHT-JUSTIFY TEN'S PLACE
        PUSH    ACC             ;CONVERT TEN'S PLACE DIGIT
        CALL    HEXASC
        POP     ACC
        CALL    SP_OUT          ;TRANSMIT TEN'S PLACE CHARACTER
        POP     ACC
        CALL    SP_OUT          ;TRANSMIT ONE'S PLACE CHARACTER
        ...     ......
```

9.0.6 N-Way Branching

There are several different means for branching to sections of code determined or selected at run time. (The single destination addresses incorporated into conditional and unconditional jumps are, of course, fixed at assembly time.) Each has advantages for different applications.

In a typical N-way branch situation, the potential destinations are generally known at assembly time. One of a number of small routines is selected according to the value of an index variable determined while the program is running. The most efficient way to solve this problem is with the MOVC and an indirect jump instruction, using a short table of offset values in ROM to indicate the relative starting addresses of the several routines.

JMP @A+DPTR is an instruction which performs an indirect jump to an address determined during program execution. The instruction adds the eight-bit unsigned accumulator contents with the contents of the sixteen-bit data pointer, just like MOVC A,@A+DPTR. The resulting sum is loaded into the program counter and is used as the address for subsequent instruction fetches. Again, a sixteen-bit addition is performed: a carry-out from the low-order eight-bits may propagate through the higher-order bits. In this case, neither the accumulator contents nor the data pointer is altered.

The example subroutine below reads a byte of RAM into the accumulator from one of four alternate address spaces, as selected by the contents of the variable MEMSEL. The address of the byte to be read is determined by the contents of R0 (and optionally R1). It might find use in a printing terminal application, where four different model printers all use the same ROM code but use different types (and sizes) of buffer memory for different speeds and options.

```
;
MEMSEL  EQU    R3
;
JUMP_4: MOV    A,MEMSEL
        MOV    DPTR,#JMPTBL
        MOVC   A,@A+DPTR
        JMP    @A+DPTR
JMPTBL: DB     MEMSP0-JMPTBL
        DB     MEMSP1-JMPTBL
        DB     MEMSP2-JMPTBL
        DB     MEMSP3-JMPTBL
MEMSP0: MOV    A,@R0             ;READ FROM INTERNAL RAM
        RET
MEMSP1: MOVX   A,@R0             ;READ 256 BYTE EXTERNAL RAM
        RET
MEMSP2: MOV    DPL,R0            ;READ 64K BYTE EXTERNAL RAM
        MOV    DPH,R1
        MOVX   A,@DPTR
        RET
MEMSP3: MOV    A,R1              ;READ 4K BYTE EXTERNAL RAM
        ANL    A,#07H
        ANL    P1,#11111000B
        ORL    P1,A
        MOVX   A,@R0
        RET
```

To use this approach, the size of the jump table plus the length of the alternate routines must be less than 256 bytes. The jump table and routines may be located anywhere in program memory and are independent of 256-byte program memory pages.

For applications where up to 128 destinations must be selected, all residing in the same 2K page of program memory, the following technique may be used. In the printing terminal example, this sequence could process 128 different codes for ASCII characters arriving via the 8051 serial port.

```
;
OPTION    EQU     R3
;         ...     ......
;         ...     ......
JMP128:   MOV     A,OPTION
          RL      A                  ;MULTIPLY BY 2 FOR 2-BYTE JUMP TABLE
          MOV     DPTR,#INSTBL       ;FIRST ENTRY IN JUMP TABLE
          JMP     @A+DPTR            ;JUMP INTO JUMP TABLE
;         ...     ......             ;
INSTBL:   AJMP    PROC00             ;128 CONSECUTIVE
          AJMP    PROC01             ;AJMP INSTRUCTIONS
          AJMP    PROC02
;         ...     ......
;         ...     ......
          AJMP    PROC7E
          AJMP    PROC7F
```

The destinations in the jump table (PROC00-PROC7F) are not all necessarily unique routines. A large number of special control codes could each be processed with their own unique routine, with the remaining printing characters all causing a branch to a common routine for entering the character into the output queue.

9.0.7 Computing Branch Destinations at Run Time

In some rare situations, 128 options are insufficient, the destination routines may cross a 2K page boundary, or a branch destination is not known at assembly time (for whatever reason), and therefore cannot be easily included in the assembled code. These situations can all be handled by computing the destination address at run-time with standard arithmetic or table look-up instructions, then performing an indirect branch to that address. There are two simple ways to execute this last step, assuming the 16-bit destination address has already been computed. The first is to load the address into the DPH and DPL registers, clear the accumulator and branch using the JMP @A+DPTR instruction; the second is to push the destination address onto the stack, low-order byte first (so as to mimic a call instruction) then pop that address into the PC by performing a return instruction. This also adjusts the stack pointer to its previous value. The code segment below illustrates the latter possibility.

```
;
RTEMP     EQU     R7
;         ...     ......
JMP256:   MOV     DPTR,#ADRTBL       ;FIRST ADDRESS TABLE ENTRY
          MOV     A,OPTION           ;LOAD INDEX INTO TABLE
          CLR     C
          RLC     A                  ;MULTIPLY BY 2 FOR 2-BYTE JUMP TABLE
          JNC     LOW128
          INC     DPH                ;FIX BASE IF INDEX>127
```

```
LOW128: MOV     RTEMP,A             ;SAVE ADJUSTED ACC FOR SECOND READ
        INC     A                   ;READ LOWORDER BYTE FIRST
        MOVC    A,@A+DPTR           ;GET LOW-ORDER BYTE FROM TABLE
        PUSH    ACC
        MOV     A,RTEMP             ;RELOAD ADJUSTED ACC
        MOVC    A,@A+DPTR           ;GET HIGH-ORDERED BYTE FROM TABLE
        PUSH    ACC
;
;       THE TWO ACC PUSHES HAVE PRODUCED
;       A "RETURN ADDRESS" ON THE STACK WHICH CORRESPONDS
;       TO THE DESIRED STARTING ADDRESS.
;       IT MAY BE REACHED BY POPPING THE STACK
;       INTO THE PC.
        RET
;       ...     ......
;       ...     ......
;       ...     ......
ADRTBL: DW      PROC00              ;UP TO 256 CONSECUTIVE DATA
        DW      PROC01              ;WORDS INDICATING STARTING ADDRESSES
;       ...     ......
;       ...     ......
        DW      PROCFF
;
```

9.0.8 In-Line-Code Parameter Passing

Parameters can be passed by loading appropriate registers with values before calling the subroutine. This technique is inefficient if a lot of the parameters are constants, since each would require a separate register to carry it, and a separate instruction to load the register each time the routine is called.

If the routine is called frequently, a more code-efficient way to transfer constants, is "in-line-code" parameter-passing. The constants are actually part of the program code, immediately following the call instruction. The subroutine determines where to find them from the return address on the stack, and then reads the parameters it needs from program memory.

For example, assume a utility named ADDBCD adds a 16-bit packed-BCD constant with a two-byte BCD variable in internal RAM and stores the sum in a different two-byte buffer. The utility must be given the constant and both buffer addresses. Rather than using four working registers to carry this information, all four bytes could be inserted into program memory each time the utility is called. Specifically, the calling sequence below invokes the utility to add 1234 (decimal) with the string at internal RAM address 56H, and store the sum in a buffer at location 78H.

The ADDBCD subroutine determines at what point the call was made by popping the return address from the stack into the data pointer high- and low-order bytes. A MOVC instruction then reads the parameters from program memory as they are needed. When done, ADDBCD resumes execution by jumping to the instruction following the last parameter.

```
         ...     ......
         CALL    ADDBCD
         DW      1234H            ;BCD CONSTANT
         DB      56H              ;SOURCE STRING ADDRESS
         DB      78H              ;DESTINATION STRING ADDRESS
;        ...     ......           ;CONTINUATION OF PROGRAM
;
;
ADDBCD:  POP     DPH              ;POP RETURN ADDRESS INTO DPTR
         POP     DPL
         MOV     A,#2             ;INDEX FOR SOURCE STRING PARAMETER
         MOVC    A,@A+DPTR        ;GET SOURCE STRING LOCATION
         MOV     R0,A
         MOV     A,#3             ;INDEX FOR DESTINATION STRING
                                  ;PARAMETER
         MOVC    A,@A+DPTR        ;GET DESTINATION ADDRESS
         MOV     R1,A
         MOV     A,#1             ;INDEX FOR 16-BIT CONSTANT LOW BYTE
         MOVC    A,@A+DPTR        ;GET LOW-ORDER VALUE
         ADD     A,@R0            ;COMPUTE LOW-ORDER BYTE OF SUM
         DA      A                ;DECIMAL ADJUST FOR ADDITION
         MOV     @R1,A            ;SAVE IN BUFFER
         INC     R0
         INC     R1
         CLR     A                ;INDEX FOR HIGH-BYTE = 0
         MOVC    A,@A+DPTR        ;GET HIGH-ORDER CONSTANT
         ADDC    A,@R0
         DA      A                ;DECIMAL ADJUST FOR ADDITION
         MOV     @R1,A            ;SAVE IN BUFFER
         MOV     A,#4             ;INDEX FOR CONTINUATION OF PROGRAM
         JMP     @A+DPTR          ;JUMP BACK INTO MAIN PROGRAM
```

This example illustrates several points:

1) The "subroutine" does not end with a normal return statement; instead, an indirect jump relative to the data pointer returns execution to the first instruction following the parameter list. The two initial POP instructions correct the stack pointer contents.

2) Either an ACALL or LCALL works with the subroutine, since each pushes the address of the *next* instruction or data byte onto the stack. The call may be made from anywhere in the full 8051 address space, since the MOVC instruction access all 64K bytes.

3) The parameters passed to the utility can be listed in whatever order is most convenient, which may not be that in which they're used. The utility has essentially "random access" to the paramater list, by loading the appropriate constant into the accumulator before each MOVC instruction.

4) Other than the data pointer, the whole calling and processing sequence only affects the accumulator, PSW and pointer registers. The utility could have pushed these registers onto the stack (after popping the parameter list starting address), and popped before returning.

Passing parameters through in-line-code can be used in conjunction with other variable passing techniques.

The utility can also get input variables from working registers or from the stack, and return output variables to registers or to the stack.

9.1 PERIPHERAL INTERFACING TECHNIQUES

9.1.1 I/O Port Reconfiguration (First Approach)

I/O ports must often transmit or receive parallel data in formats other than as eight-bit bytes. For example, if an application requires three five-bit latched output ports (called X, Y, and Z), these "virtual" ports could be mapped onto the pins of "physical" ports 1 and 2 (see example at bottom of page).

This pin assignment leaves P2.7 free for use as a test pin, input data pin, or control output through software.

Notice that the bits of port Z are reversed. The highest-order port Z pin corresponds to pin P2.2, and the lowest-order pin of port Z is P2.6, due to P.C. board layout considerations. When connecting an 8051 to an immediately adjacent keyboard column decoder or another device with weighted inputs, the corresponding pins may not be aligned. The interconnections must be "scrambled" to compensate either with interwoven circuit board traces or through software (as shown on the next page).

	PORT "Z"					PORT "Y"					PORT "X"				
—	PZ0	PZ1	PZ2	PZ3	PZ4	PY4	PY3	PY2	PY1	PY0	PX4	PX3	PX2	PX1	PX0
P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	P2.0	P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	P1.0

```
PX_MAP  DATA   20H
PY_MAP  DATA   21H
PZ_MAP  DATA   22H
;       ...    ....
OUT_PX: ANL    A,#00011111B          ;CLEAR BITS ACC.7 - ACC. 5
        MOV    PX_MAP,A              ;SAVE DATA IN MAP BYTE
        ACALL  OUT_P1                ;UPDATE PORT 1 OUTPUT LATCH
        RET
;       ....   ....
OUT_PY: MOV    PY_MAP,A              ;SAVE IN MAP BYTE
        ACALL  OUT_P1                ;UPDATE PORT 1
        ACALL  OUT_P2                ;AND PORT 2 OUTPUT LATCHES
        RET
;       ....   ....
OUT_PZ: MOV    PZ_MAP,A              ;SAVE DATA IN MAP BYTE
        ACALL  OUT_P2                ;UPDATE PORT 2.
        RET
;       ....   ....
;       ....   ....
OUT_P1: MOV    A,PY_MAP              ;OUTPUT ALL P1 BITS
        SWAP   A
        RL     A                     ;SHIFT PY_MAP LEFT 5 BITS
        ANL    A,#11100000B          ;MASK OUT GARBAGE
        ORL    A,PX_MAP              ;INCLUDE PX_MAP BITS
        MOV    P1,A
        RET
;       ....   ....
OUT_P2: MOV    C,PZ_MAP.0            ;LOAD CY WITH P2.6 BIT
        RLC    A                     ;AND SHIFT INTO ACC.
        MOV    C,PZ_MAP.1            ;LOAD CY WITH P2.5 BIT
        RLC    A                     ;AND SHIFT INTO ACC.
        MOV    C,PZ_MAP.2            ;LOAD CY WITH P2.4 BIT
        RLC    A                     ;AND SHIFT INTO ACC.
        MOV    C,PZ_MAP.3            ;LOAD CY WITH P2.3 BIT
        RLC    A                     ;AND SHIFT INTO ACC.
        MOV    C,PZ_MAP.4            ;LOAD CY WITH P2.2 BIT
        RLC    A                     ;AND SHIFT INTO ACC.
        MOV    C,PZ_MAP.4            ;LOAD CY WITH P2.1 BIT
        RLC    A                     ;AND SHIFT INTO ACC.
        MOV    C,PZ_MAP.3            ;LOAD CY WITH P2.0 BIT
        RLC    A                     ;AND SHIFT INTO ACC.
        SETB   ACC.7                 ;(ASSUMING INPUT ON P2.7)
        MOV    P2.A
        RET
```

Writing to the virtual ports must not affect any other pins. Since the virtual output algorithms are non-trivial, a subroutine is needed for each port: OUT_PX, OUT_PY and OUT_PZ. Each is called with data to output right-justified in the accumulator, and any data in bits ACC.7-ACC.5 is insignificant. Each subroutine saves the data in a "map" variable for the virtual port, then calls other subroutines which use the data in the various map bytes to compute and output the eight-bit pattern needed for each physical port affected. The two level structure of the above subroutines can be modified somewhat if code efficiency and execution speed are critical: incorporate the code shown as subroutines OUT_P1 and OUT_P2 directly into the code for OUT_PX and OUT_PZ, in place of the corresponding ACALL instructions. OUT_PY would not be changed, but now the destinations for its ACALL instructions would be alternate entry points in OUT_PX and OUT_PZ, instead of isolated subroutines.

9.1.2 I/O Port Configuration (Second Approach)

A trickier situation arises if two sections of code which write to the same port or register, or call virtual output routines like those above, need to be executed at different interrupt levels. For example, suppose the background program wants to rewrite Port X (using the port associations in the previous example), and has computed the bit pattern needed for P1. An interrupt is detected just before the MOV P1,A instruction, and the service routine tries to write to Port Y. The service routine would correctly update P1 and P2, but upon returning to the background program P1 is immediately *re*-written with the data computed *before* the interrupt! Now pins P2.1 and P2.0 indicate (correctly) data written to port Y in the interrupt routine, but the earlier data written to P1.7-P1.5 is no longer valid. The same sort of confusion could arise if a high-level interrupt disrupted such an output sequence.

One solution is to disable interrupts around any section of code which must not be interrupted (called "critical section") but this would adversely affect interrupt latency. Another is to have interrupt routines set or clear a flag ("semaphore") when a common resource is altered — a rather complex and elaborate system.

An easier way to ensure that any instruction which writes the port X field of P1 does not change the port Y field pins from their state *at the beginning of that instruction*, is shown next. A number of 8051 operations read, modify, and write the output port latches all in one instruction. These are the arithmetic and logical instructions (INC, DEC, ANL, ORL, etc.), where an addressed byte is both the destination variable and one of the source operands. Using these instructions, instead of data moves, eliminates the critical section problem entirely.

```
OUT_PX: ANL     P1,#11100000B     ;CLEAR BITS P1.4 - P1.0
        ORL     P1,A              ;SET P1 PIN FOR EACH ACC BIT SET
        RET
;       ...     ......
;       ...     ......
OUT_PY: MOV     B#20H
        MUL     AB                ;SHIFT B A LEFT 5 BITS.
        ANL     P1,#00011111B     ;CLEAR PY FIELD OF PORT 1
        ORL     P1,A              ;SET PY PITS ON PORT 1
        MOV     A,B               ;LOAD 2 BITS SHIFTED INTO B
        ANL     P2,#11111100B     ;AND UPDATE P2
        ORL     P2,A
        RET
;       ...     ......
;       ...     ......
OUT_PZ: RRC     A                 ;MOVE ORIGINAL ACC.0 INTO CY
        MOV     P2.6,C            ;AND STORE TO PIN P2.6.
        RRC     A                 ;MOVE ORIGINAL ACC.1 INTO CY
        MOV     P2.5,C            ;AND STORE TO PIN P2.5.
        RRC     A                 ;MOVE ORIGINAL ACC.2 INTO CY
        MOV     P2.4,C            ;AND STORE TO PIN P2.4.
        RRC     A                 ;MOVE ORIGINAL ACC.3 INTO CY
        MOV     P2.3,C            ;AND STORE TO PIN P2.3.
        RRC     A                 ;MOVE ORIGINAL ACC.4 INTO CY
        MOV     P2.2,C            ;AND STORE TO PIN P2.2.
        RET
```

9.1.3 8243 Interfacing

The 8051's quasi-bidirectional port structure lets each I/O pin input data, output data, or serve as a test pin or output strobe under software control. An example of these modes operating in conjunction is the host-processor interface expected by an 8243 I/O expander. Even though the 8051 does not include 8048-type instructions for interfacing with an 8243, the parts can be interconnected and the protocol may be emulated with simple software; see Figure 9-2.

```
;
;IN8243   INPUT DATA FROM AN 8243 I/O EXPANDER
;         CONNECTED TO P23-P20.
;         P25 & P24 MIMIC CS & PROG.
;         P27-P26 USED AS INPUTS. CODE FOR
;         PORT TO BE READ IN ACC.1-ACC.0
;
```

```
PROG      BIT     P2.4                ;SYMBOLIC PIN DESCRIPTION
;
IN8243:   ORL     A,#11010000B        ;SET PROG AND PINS USED AS INPUT
          MOV     P2,A                ;OUTPUT PORT CODE AND OPERATION CODE
          CLR     PROG                ;LOWER PROG TO LATCH ADDRESS
          ORL     P2,#00001111B       ;SET LOW ORDER PINS FOR INPUT
          MOV     A,P2                ;READ IN PORT DATA
          ORL     P2,#00110000B       ;SET PROG AND CS HIGH
```

9.1.4 Software Delay Timing

Many 8051 applications involve exact control over output timing. A software-generated output strobe, for instance, might have to be *exactly* 50µsec. wide. The DJNZ operation can insert an one instruction software delay into a piece of code, adding a moderate delay of two instruction cycles per iteration. For example, two instructions can add a 49-µsec software delay loop to code to generate a pulse on the WR pin.

```
CLR       WR
MOV       R2,#24
DJNZ      R2,$
SETB      WR
```

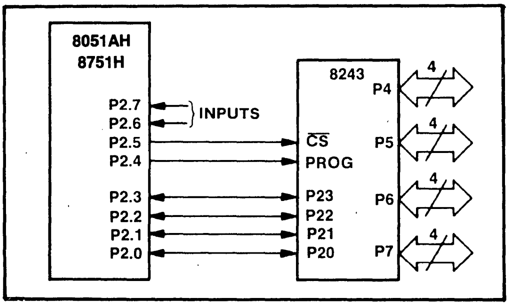


Figure 9-2. Connecting an 8051 with an 8243 I/O Expander

The dollar sign in this example is a special character meaning "the address of this instruction." It can be used to eliminate instruction labels on nearby source lines.

9.1.5 Serial Port and Timer Configuration

Configuring the 8051's Serial Port for a given data rate and protocol requires essentially three short sections of software. On power-up or hardware reset the serial port and timer control words must be initialized to the appropriate values. Additional software is also needed in the transmit routine to load the serial port data register and in the receive routine to unload the data as it arrives.

To choose one arbitrary example, assume the 8051 should communicate with a standard CRT operating at 2400 baud (bits per second). Each character is transmitted as seven data bits, odd parity, and one stop bit. The resulting character rate is 2400 baud/9 bits, approximately 265 characters per second.

For the sake of clarity, the transmit and receive subroutines here are driven by simple-minded software status polling code rather than interrupts. The serial port must be initialized to 8-bit UART mode (SM0, SM1 = 01), enabled to receive all messages (SM2 = 0, REN = 1). The flag indicating that the transmit register is free for more data will be artificially set in order to let the output software know the output register is available. All this can be set up with instruction at label SPINIT.

Timer 1 will be used in auto-reload mode as a baud rate generator. To achieve a data rate of 2400 baud, the timer must divide the 1MHz internal clock by

$$\frac{1 \times 10^6}{(32)(2400)}$$

which equals 13 (actually, 13.02) instruction cycles. The timer must reload the value −13, or 0F3H, as shown by the code at label TIINIT. (ASM51 will accept both the signed decimal or hexadecimal representations.)

```
;
;           INITIALIZE SERIAL PORT
;           FOR 8-BIT UART MODE
;           & SET TRANSMIT READY FLAG.
SPINIT:     MOV    SCON,#01010010B
;           INITIALIZE TIMER 1 FOR
;           AUTO-RELOAD AT 32 X 2400HZ
;           (T0 USED AS GATED 16-BIT COUNTER.)
;TIINIT:    MOV    TCON,#1101001B
            MOV    TH1,#13
            SETB   TR1
;           ...    ......
;
```

9.1.6 Simple Serial I/O Drivers

SP_OUT is a simple subroutine to transmit the character passed to it in the accumulator. First it must compute the parity bit, insert it into the data byte, wait until the transmitter is available, output the character and then return.

SP_IN is an equally simple routine which waits until a character is received, sets the carry flag if there is an odd-parity error, and returns the masked seven-bit code in the accumulator.

```
;
;SP_OUT ADD ODD PARITY TO ACC AND
;       TRANSMIT WHEN SERIAL PORT READY
;
SP_OUT: MOV    C,P
        CPL    C
        MOV    ACC.7,C
        JNB    TI,$
        CLR    TI
        MOV    SBUF,A
        RET
;
;
```

```
;SP_IN      INPUT NEXT CHARACTER FROM SERIAL PORT.
:           SET CARRY IF ODD-PARITY ERROR
;
SP_IN:      JNB     RI,$
            CLR     RI
            MOV     A,SBUF
            MOV     C,P
            CPL     C
            ANL     A,#7FH
            RET
```

9.1.7 Transmitting Serial Port Character Strings

Any application which transmits characters through a serial port to an ASCII output device will on occasion need to output "canned" messages, including error messages, diagnostics, or operator instructions. These character strings are most easily defined with in-line data bytes defined with the DB directive.

```
CR          EQU     0DH                     ;ASCII CARRIAGE RET
LF          EQU     0AH                     ;ASCII  LINE-FEED
ESC         EQU     1BH                     ;ASCII ESCAPE CODE
;           ...     ......
            CALL    XSTRING
            DB      CR,LF                   ;NEW LINE
            DB      'INTEL  DELIVERS'       ;MESSAGE
            DB      ESC                     ;ESCAPE CHARACTER
;
;           (CONTINUATION OF PROGRAM)
;
;           ...     ......
XSTRING:    POP     DPH                     ;LOAD DPTR WITH FIRST CHARACTER
            POP     DPL
XSTR_1:     CLR     A                       ;(ZERO OFFSET)
            MOVC    A,@A+DPTR               ;FETCH FIRST CHARACTER OF STRING
XSTR_2:     JNB     TI,$                    ;WAIT UNTIL TRANSMITTER READY
            CLR     TI                      ;MARK AS NOT READY
            MOV     SBUF,A                  ;OUTPUT NEXT CHARACTER
            INC     DPTR                    ;BUMP POINTER
            CLR     A
            MOVC    A,@A+DPTR               ;GET NEXT OUTPUT CHARACTER
            CJNE    A,#ESC,XSTR_2           ;LOOP UNTIL ESCAPE READ
            MOV     A,#1
            JMP     @A+DPTR                 ;RETURN TO CODE AFTER ESCAPE
```

9.1.8 Recognizing and Processing Special Cases

Before operating on the data it receives, a subroutine might give "special handling" to certain input values. Consider a word processing device which receives ASCII characters through the 8051 serial port and drives a thermal hard-copy printer. A standard routine translates most printing characters to bit patterns, but certain control characters ([DEL], [CR], [LF], [BEL], [ESC], or [SP]) must invoke corresponding special routines. Any other character with an ASCII code less than 20H should be translated into the [NUL] value, 00H, and processed with the printing characters. The CJNE operation provides essentially a one-instruction CASE statement.

```
;
CHAR     EQU    R7                  ;CHARACTER CODE VARIABLE
;
INTERP:  CJNE   CHAR,#7FH, INTP_1   ;SKIP UNLESS RUBOUT
;        ...    . . . . .           (SPECIAL ROUTINE FOR RUBOUT CODE)
         RET
INTP_1:  CJNE   CHAR,#07H,INTP_2    ;SKIP UNLESS BELL
;        ...    . . . . .           (SPECIAL ROUTINE FOR BELL CODE)
         RET
INTP_2:  CJNE   CHAR,#0AH,INTP_3    ;SKIP UNLESS LFEED
;        ...    . . . . .           (SPECIAL ROUTINE FOR LFEED CODE)
         RET
INTP_3:  CJNE   CHAR,#0DH,INTP_4    ;SKIP UNLESS RETURN
;        ...    . . . . .           (SPECIAL ROUTINE FOR RETURN CODE)
         RET
INTP_4:  CJNE   CHAR,#1BH,INTP_5    ;SKIP UNLESS ESCAPE
;        ...    . . . . .           (SPECIAL ROUTINE FOR ESCAPE CODE)
         RET
INTP_5:  CJNE   CHAR,#20H,INTP_6    ;SKIP UNLESS SPACE
;        ...    . . . . .           (SPECIAL ROUTINE FOR SPACE CODE)
         RET
INTP_6:  JC     PRINTC              ;JUMP IF CODE 20H
         MOV    CHAR,#0             ;REPLACE CONTROL CHARACTER WITH
                                    ;NULL CODE
PRINTC:                             ;PROCESS STANDARD PRINTING
;        ...    . . . . .           ;CHARACTER
         RET
;
```

9.1.9 Synchronizing Timer Overflows

8051 timer overflows automatically generate an internal interrupt request, which will vector program execution to the appropriate interrupt service routine if interrupts are enabled and no other service routines are in progress at the time. However, it is not predictable *exactly* how long it will take to reach the service routine. The service routine call takes two instruction cycles, but 1, 2, or 4 additional cycles may be needed to complete the instruction in progress. If the background program ever disables interrupts, the response latency could further increase by a few instruction cycles. (Critical sections generally involve simple instruction sequences – rarely multiplies or divides.) Interrupt response delay is generally negligible, but certain time-critical applications must take the exact delay into account. For example, generating interrupts with timer 1 every millisecond (1000 instruction cycles) or so would normally call for reloading it with the value -1000 (0FC30H). But if the interrupt interval (average over time) must be accurate to 1 instruction cycle, the 16-bit value reload into the timer must be computed, taking into account when the timer actually overflowed.

This simply requires reading the appropriate timer, which has been incremented each cycle since the overflow occurred. A sequence like the one below can stop the timer, compute how much time should elapse before the next interrupt, and reload and restart the timer. The double-precision calculation shown here compensates for any amount of timer overrun within the maximum interval. Note that it also takes into account that the timer is stopped for seven instruction cycles in the process. All interrupts are disabled, so a higher priority request will not be able to disrupt the time-critical code section.

```
;       ...     . . . . .
        CLR     EA                      ;DISABLE ALL INTERRUPTS
        CLR     TR1                     ;STOP TIMER 1
        MOV     A,#LOW(—1000+7)         ;LOAD LOW-ORDER DESIRED COUNT
        ADD     A,TL1                   ;CORRECT FOR TIMER OVERRUN
        MOV     TL1,A                   ;RELOAD LOW-ORDER BYTE.
        MOV     A,#HIGH(—1000+7)        ;REPEAT FOR HIGH-ORDER BYTE.
        ADDC    A,TH1
        MOV     TH1,A
        SETB    TR1                     ;RESTART TIMER
;       ...     . . . . .
```

9.1.10 Reading a Timer/Counter "On-the-Fly"

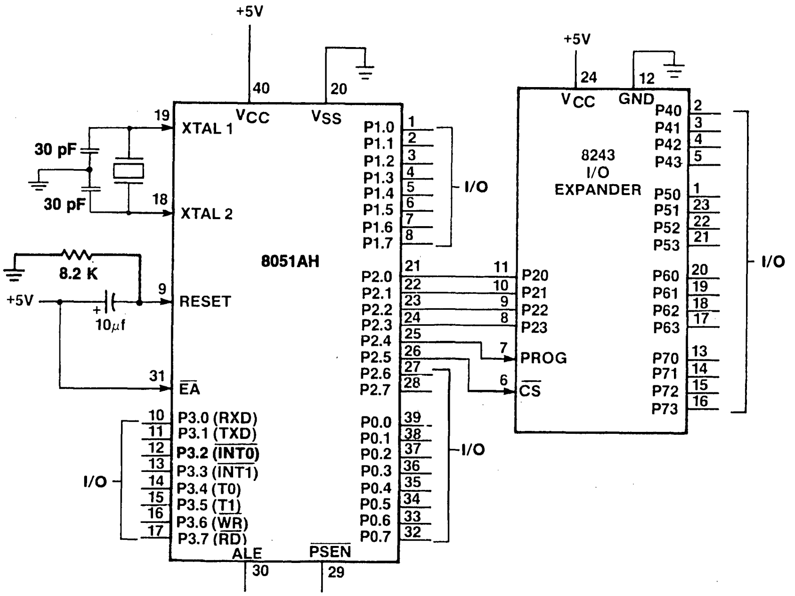
The preceding example simply stopped the timer before changing its contents. This is normally done when reloading a timer so that the time at which the timer is started (i.e. the "run" flag is set) can be exactly controlled. There are situations, though, when it is desired to read the current count without disrupting the timing process. The 8051 timer/counter registers can all be read or written while they are running, but a few precautions must be taken.

Suppose the subroutine RDTIME should return in [R1], [R0] a sixteen-bit value indicating the count in timer 0. The instant at which the count was sampled is not as critical as the fact that the value returned must have been valid at some point while the routine was in progress. There is a potential problem that between reading the two halves, a low-order register overflow might increment the high-order register, and the two data bytes returned would be "out of phase." The solution is to read the high-order byte first, then the low-order byte, and then confirm that the high-order byte has not changed. If it has, repeat the whole process.

```
RDTIME: MOV   A,TH0              ;SAMPLE TIMER0 (HIGH)
        MOV   R0,TL0             ;SAMPLE TIMER0 (LOW)
        CJNE  A,TH0,RDTIME       ;REPEAT IF NECESSARY
        MOV   R1,A               ;STORE VALID READ
        RET
```

9.2 PERIPHERAL INTERFACE EXAMPLES

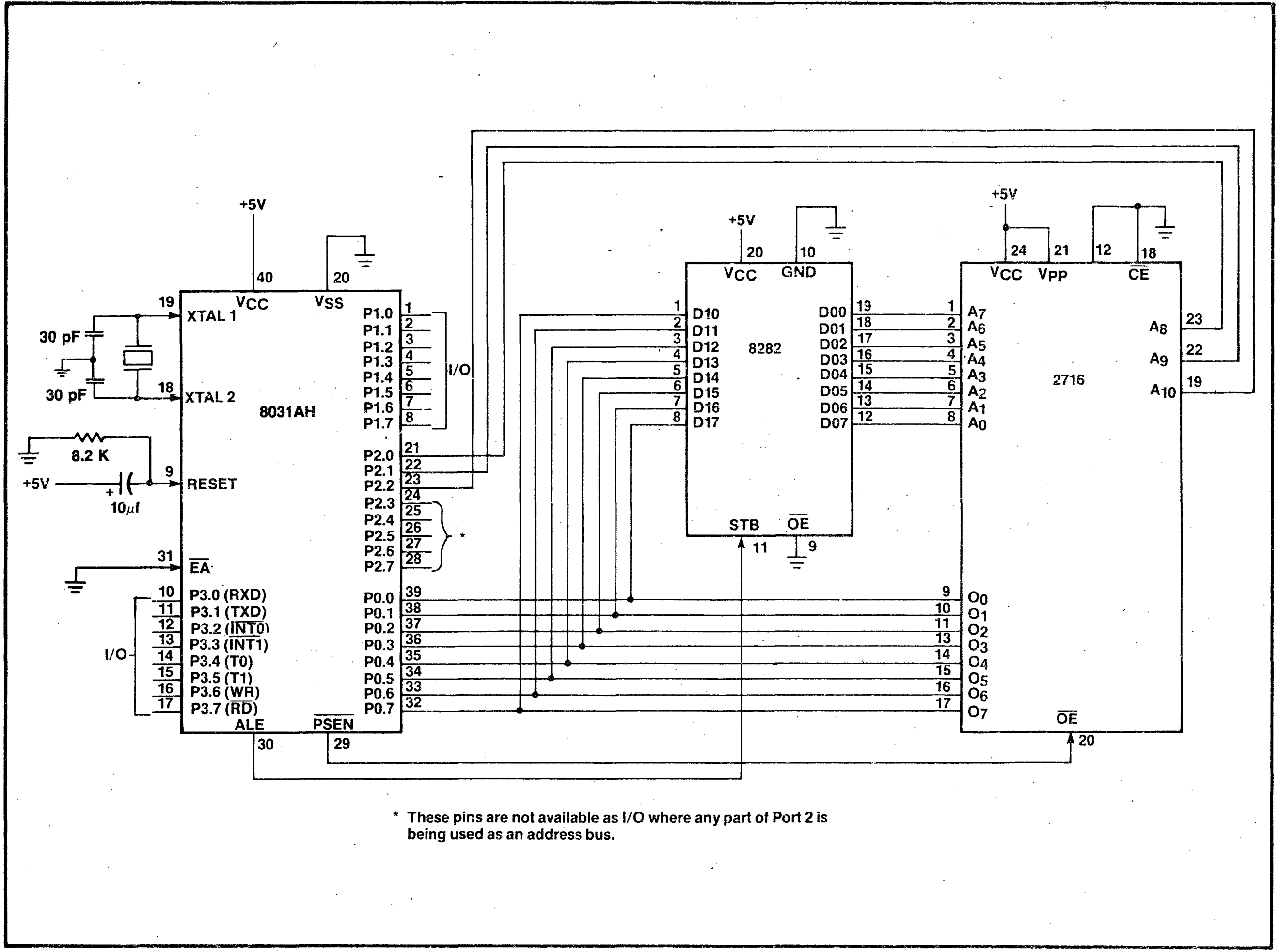
This section should give the designer an insight into connecting external peripherals and memories to the MCS-51 family.



The following software driver is required to interface to the 8243.

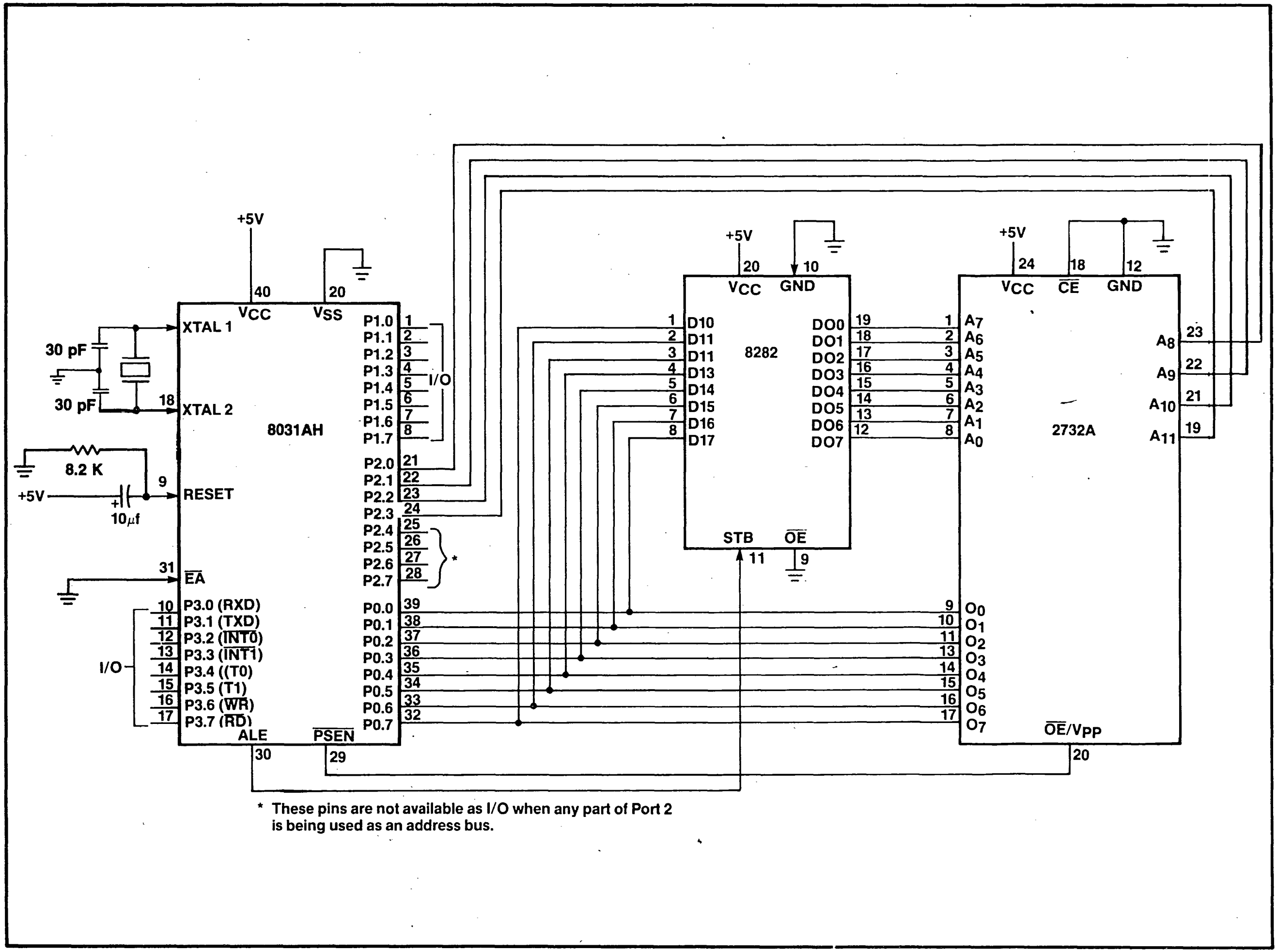
```
        Mixing Parallel Output, Input, and
        Control Strobes on Port 2.

IN8243  INPUT DATA FROM AN 8243 I/O EXPANDER
        CONNECT TO P23-P20
        P25 & P24 MIMIC CS/ & PROG
        P27-P26 USED AS INPUTS
        PORT TO BE READ IN ACC

IN8243  ORL     A#11010000B
        MOV     P2,A    ;OUTPUT INSTRUCTION CODE
        CLR     P2,4    :FALLING EDGE OF PROG
        ORL     P2,#00001111B    ;SET FOR INPUT
        MOV     A,P2    ;READ INPUT DATA
        SETB    P2.4    ;RETURN PROG HIGH
```

Figure 9-3. I/O Expansion Using an 8243



* These pins are not available as I/O where any part of Port 2 is being used as an address bus.

Figure 9-4. External Program Memory Using a 2716



* These pins are not available as I/O when any part of Port 2 is being used as an address bus.

Figure 9-5. External Program Memory Using a 2732

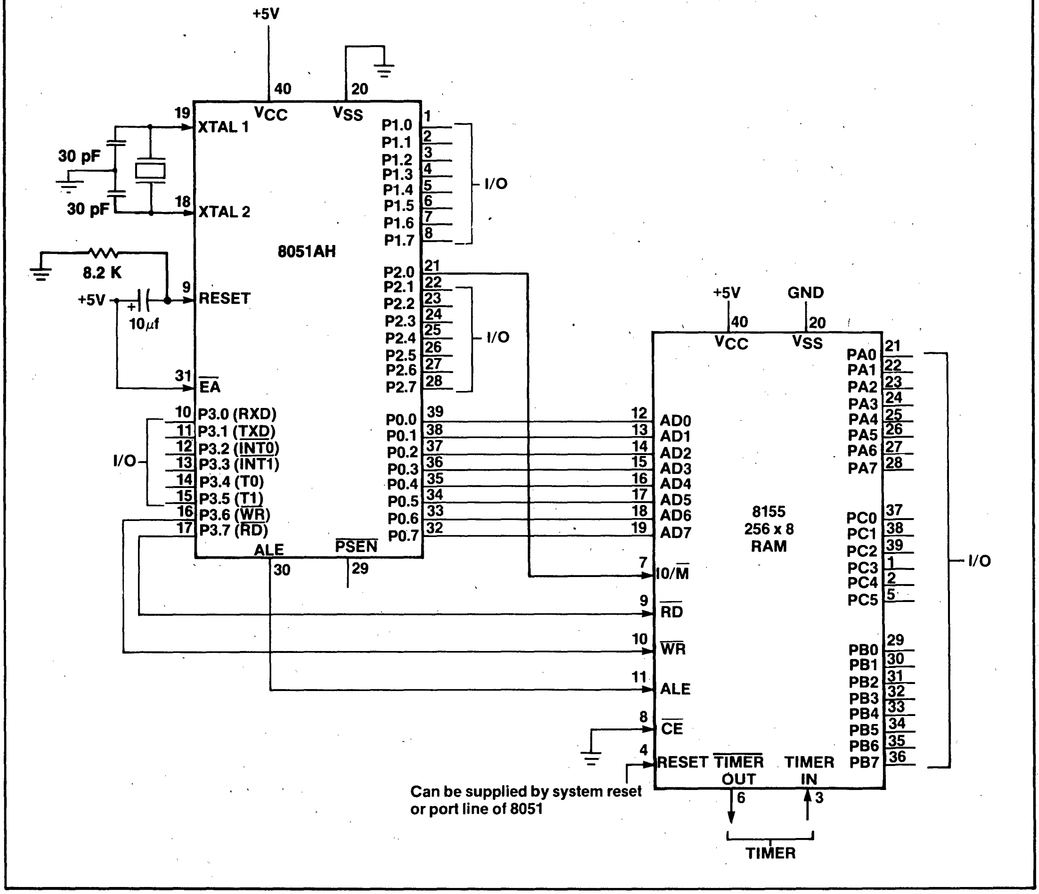


Figure 9-6. Adding a Data Memory and I/O Expander

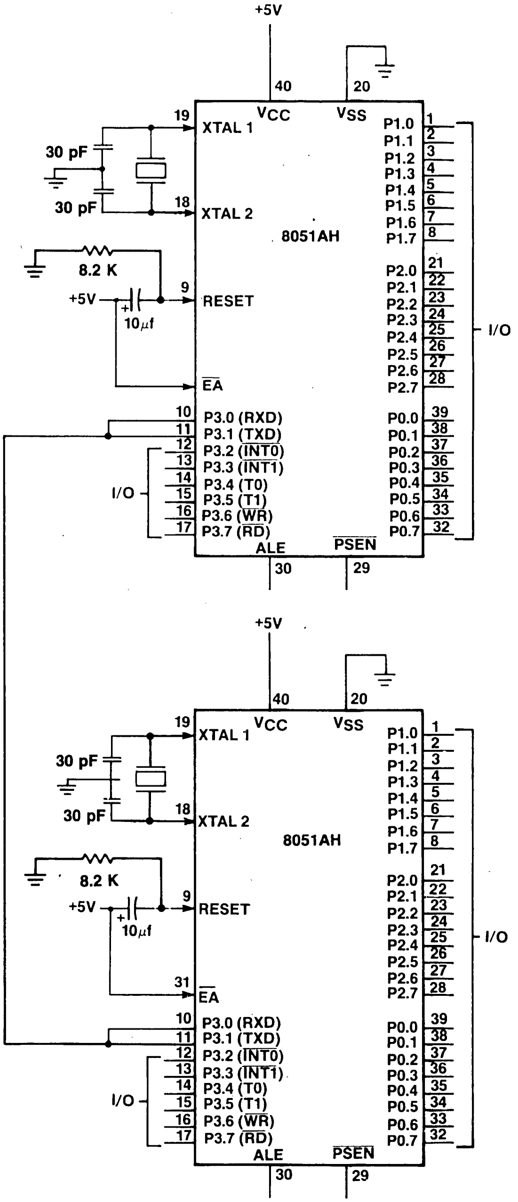


Figure 9-7. Multiple 8051's Using Half-Duplex Serial Communication

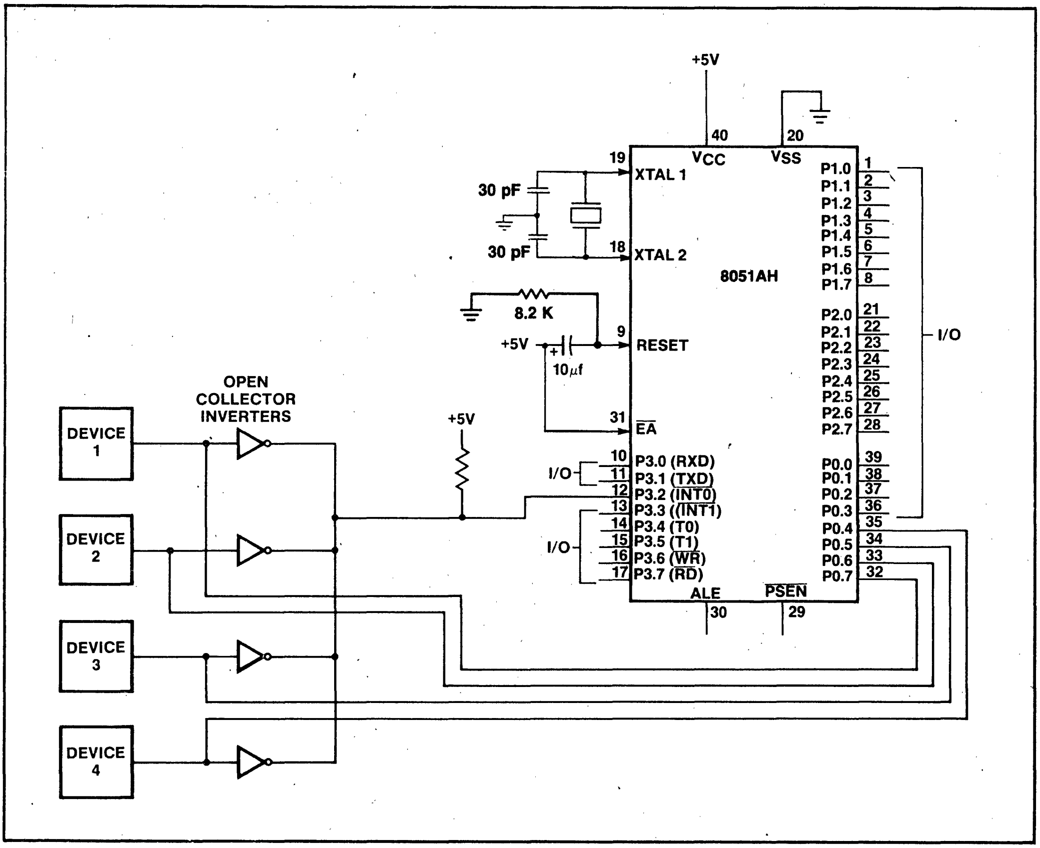


Figure 9-8. Multiple Interrupt Sources

Article Reprints 10

Putting a new wrinkle in a CMOS process has led to a pair of high-caliber microcontrollers that limit power dissipation but not speed.

Microcontrollers go CMOS without slowing down

Long considered a low-speed, complex technology compared with NMOS, CMOS is on the rise as it prepares to compete successfully with NMOS for new, high-speed microprocessor applications. Using its HMOS-II technology, Intel has developed high-performance CMOS (C-HMOS) logic and applied it to two standard NMOS microcontrollers, the 8049 and 8051.

The two new members of the MCS-48 and MCS-51 families—the 80C49 and 80C51, respectively—are out to erase the low-performance image of previous CMOS devices while minimizing price differences and chip sizes compared with their n-channel counterparts. C-HMOS technology borrows many of the design and performance characteristics of high-performance MOS (HMOS). However, developing C-HMOS called for the creation of an n-well CMOS process (Fig. 1). That had not been done previously because CMOS technology grew naturally out of the older p-channel MOS technology.

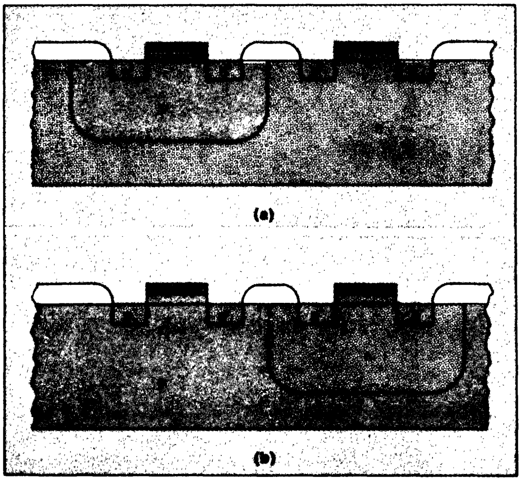
N-well CMOS offers two important processing advantages: First, the substrate remains the same as in HMOS. Thus n-channel transistors in C-HMOS require no new process characterization. Second, the only new processing necessary are the n well and a p-channel transistor.

Because they retain the design rules of HMOS-II, C-HMOS devices realize the high speed and superior packing density of that process. In addition, C-HMOS, like CMOS, dissipates less power than any other semiconductor technology. One of the best uses of the combined characteristics is in a microcontroller that must operate at low input power levels, and the number of such applications is growing. They include battery-powered equipment and systems that must keep working when normal ac power is removed. In fact, this class of applications would not exist if not for CMOS.

Microcontrollers are an ideal vehicle for the first use of C-HMOS technology for several reasons. For one, since they are generally stand-alone devices, designers can construct a complete one-chip CMOS microcontroller system. For another, they are widely used in battery-powered systems, in which CMOS devices are mandatory. Also, since they contain RAM, ROM, and random logic on a single chip, designers gain experience with the major forms of CMOS hardware contained on a single device. As for the 80C49 and 80C51 in particular, designers already familiar with the 8049 and 8051 can gain the benefits of CMOS with minor software modifications.

In developing the 80C49 and 80C51, the architectural differences between the NMOS and C-HMOS

John Katausky, Technical Marketing Manager
George Leach, Design Engineering Manager
Intel Corp., Microcontroller Operation
5000 W. Williams Field Rd., Chandler, AZ 85224



1. In the standard CMOS process, a p well resides in an n-type substrate (a). C-HMOS technology reverses this process, placing an n well in a p-type substrate (b).

Reprinted from ELECTRONIC DESIGN - February 18, 1982

versions were minimized. As a result, designers need not relearn instruction sets, development tools, and existing hardware designs. Existing software can even be patched with idle instructions to reduce significantly the part's power consumption in a given application.

The 80C49 leads the way

The 80C49 is pin– and instruction-set–compatible with the industry-standard 8049 microcontroller (Fig. 2a). It contains 2 kbytes of ROM, 128 bytes of RAM, and an 8-bit timer-counter. Other features include three 8-bit I/O ports, two test inputs, and an output structure geared to easy expansion of the I/O. To cover a range of controller applications, two companion devices are in the offing. One is the 80C48 (1 kbyte of ROM, 64 bytes of RAM) and the other is the 80C50 (4 kbytes of ROM, 256 bytes of RAM). Both will use the same CPU as the 80C49 and include the same power-down and idle features.

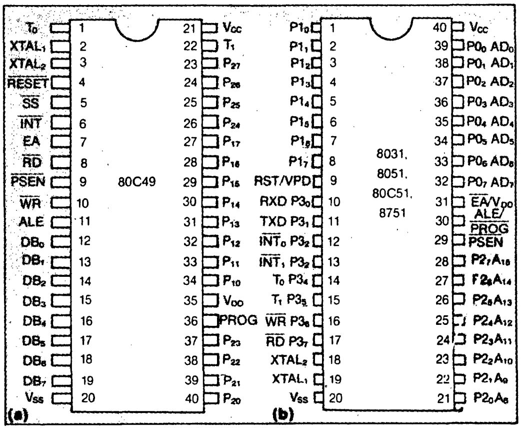
The 80C49 operates off 4.0 to 6.0 V. Between 4.5 to 5.5 V, its inputs and outputs match those of TTL circuits. When interfaced with CMOS circuitry, both inputs and outputs reach CMOS levels for the logic 1 and 0 states.

The operating frequency in CMOS devices is highly dependent on supply voltage. In the case of 80C49, above 4.5 V it can run at 11 MHz, but below that the clock rate must be reduced. For example, at the maximum supply voltage of 6 V and an operating frequency of 11 MHz, the chip draws a maximum current of 15 mA; with a 4-V power supply and a frequency of 1 MHz, however, the current drain is just 1 mA (see the table).

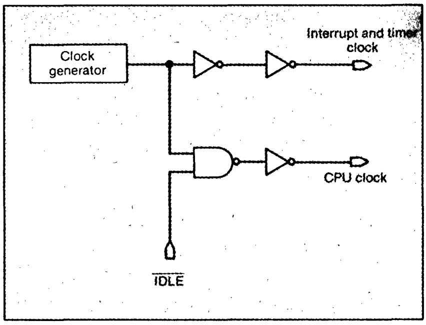
To reduce power consumption further, the 80C49 incorporates a very flexible idle mode. "Idle" is a new instruction added to the 80C49's architecture. Its function is to stop all CPU activity and allow only the timer-counter and the interrupt circuitry to remain active. In hardware, that is accomplished using two sets of clocks, one for the CPU and the other for the timer-counter and interrupt circuitry. Figure 3 shows the logical implementation of the dual clock. With this setup, the CPU is easily disabled by selectively stopping its clock.

During idle, the counter-timer clock continues to run and the entire 80C49 draws but a fraction of its normal active current. The user can monitor events or real-time functions while holding the current drain to just 500 μA at 6 V and 11 MHz.

Either the timer-counter interrupt or an external interrupt terminates the idle mode. Interrupts are then handled as though they were received under normal operating conditions. Moreover, a simple flag, implemented by the user, can indicate if the interrupt occurred during the idle condition or during normal operation. A hardware reset also terminates the idle mode.



2. The first high-performance CMOS microcontroller, the 80C49, is compatible pin for pin and electrically with the NMOS 8049 (a), but contains features that allow it to dissipate less power than its NMOS cousin. Like members of the MCS-48 family, the C-HMOS 80C51 executes programs out of its internal memory or from an external memory. When operating with an external memory, the chip's pins assume the functions shown in the outside columi 'h).

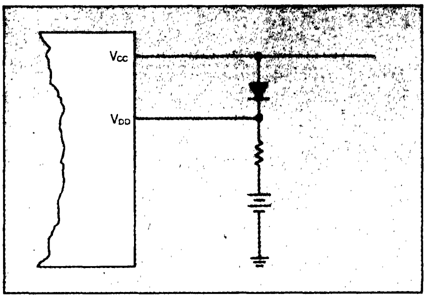


3. The 80C49's low-power idle mode is implemented using dual clocks, allowing the timer-counter and interrupts to remain active while the controller draws just 500 μA operating from a 6-V power supply at an 11-MHz clock rate.

The timer-counter remains active during the idle mode to allow the idle instruction to be as useful as possible. For example, the absence of an active timer-counter forces the user to implement an external hardware interrupt to end the idle mode, a less than ideal solution in almost all applications. With an active timer-counter, the user can operate the 80C49 in a programmable duty-cycle mode, which allows the ratio of CPU active time to idle time to be determined by software.

Holding down power

Power consumption is reduced to an absolute minimum by placing the 80C49 in a hardware power-down mode (Fig. 4). During power-down, only the on-board RAM is kept active. Power-down is activated by turning off the V_{CC} supply and reducing the V_{DD} supply to its minimum value of 2 V. That cuts the current drain to only 10 μA.



4. Absolute minimum power is drawn during the 80C49's power-down mode. A fail-safe power-down mode circuit can be designed with just a germanium diode, a resistor, and two or three nickel-cadmium cells.

For some applications, it is useful for the designer to know whether a power-up condition occurred from a cold start or from the power-down mode. That is made possible by software and is accomplished by inserting a 1- or 2-byte "key" in the RAM before entering the power-down mode. When the device resets and comes out of power-down, the key bytes are examined, enabling the CPU to determine from what condition the reset occurred.

Higher performance with the 80C51

Specifically targeted at high-end 8-bit microprocessor applications requiring low power consumption, the 80C51 has all of the 8051's architectural features, including its enhanced CPU and I/O functions. The on-chip program memory size remains 4 kbytes, with a full 64-kbyte external range. The 128-byte on-chip RAM also is externally expandable—to 64 kbytes. Also, on board are two 16-bit timer-counters, a full duplex serial port, and a 1-bit Boolean processor for control functions. Figure 2b shows the 80C51's pinout for both the expanded and the port mode.

Electrically similar to the 80C49, the 80C51 has a V_{CC} range of 4 to 6 V and is TTL-compatible between 4.5 and 5.5 V. Above 4.5 V, the chip operates at clock rates of up to 15 MHz (see the table).

The current drain for the 80C49/80C51 (mA)

V_{CC}	At 1 MHz	At 6 MHz	At 11/12 MHz*	At 15 MHz
4 V	1/1.3	5.5/8	—/16	—/—
5 V	1.2/1.6	7/10	12.5/20	—/26
6 V	1.5/2.0	8.5/12.5	15/25	—/31

*11 MHz for the 80C49, 12 MHz for the 80C51.

Like the 80C49, the 80C51 offers an idle mode for lower power dissipation. This mode is initiated by setting a bit in a new on-chip register, called the PCON register, that resides in previously unused chip area. When the bit is set, the CPU is disabled, but the timer-counter, interrupts, and serial port continue to function normally. A reset or any enabled interrupt terminates the idle mode.

If an interrupt ends the idle mode, all register data remain unchanged and the interrupt is serviced. Idle is implemented in much the same way as on the 80C49, that is, by splitting the on-chip clocks into two signals, one for the CPU and the other for the active idle circuitry. Using this technique, current drain in the idle mode is about 1 mA at a supply voltage of 6 V and an operating frequency of 15 MHz.

Other bits of the PCON registers can be set as flags to determine whether an interrupt has been used for its normal function or to terminate the idle mode. These flags permit an interrupt to do double duty, instead of having to dedicate a particular interrupt specifically to this mode.

How the idle mode works

Connecting a number of slave 80C51s to a common serial link serves as an example of how to use the 80C51's idle mode. These slave processors would be controlled by a master computer or a master 80C51. Each slave processor would contain a unique code byte that would be stored in on-chip memory. To conserve power, each of the slave 80C51s could be in the idle mode. When the master transmitted a code byte, all of the slaves would receive a serial-port interrupt terminating the mode. Then each slave would compare the transmitted code byte with its internal code byte. The 80C51 whose internal code matched the transmitted code could then carry on the task it was programmed for, while the other slaves would return to the idle mode.

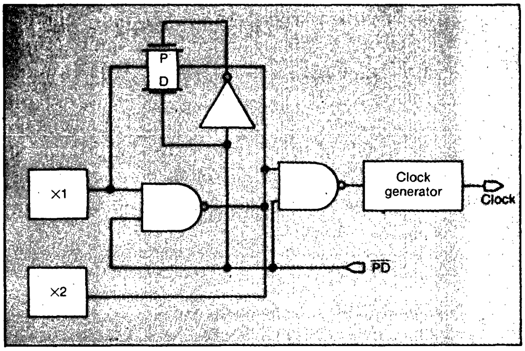
A new power-down mode is incorporated on the

80C51 (Fig. 5), but unlike that on the 80C49, it is achieved through software. As with idle, power-down is entered by setting a bit in the PCON register. That disables the clock oscillator, freezing the clocks and stopping all functions. Therefore all vital hardware-register data must be stored in the on-chip RAM before starting the power-down mode. While powered down, V_{CC} can be lowered to 2 V—at that level, total current drain is only 50 μA. The only way to terminate power-down is through a hardware reset. As with the 80C49, software flags can be used to determine the state the processor was in when an interrupt or reset occurred.

C-HMOS for other kinds of circuits

Although the first products are single-chip microcontrollers, C-HMOS technology can be used for any product fabricated in HMOS and for some made with other technologies.

In building CMOS and C-HMOS circuits, p-channel and n-channel devices are created. Thus n-type and p-type impurities are routinely doped into the substrate. It is therefore possible to fabricate npn and pnp bipolar transistors on the same substrate. When high-drive capability is needed, a designer can create an npn device. In fact, in an experimental ECL-compatible RAM memory, designers have used an npn transistor to provide the ECL output levels.



5. The 80C51's power-down mode is activated by a bit in the new on-board PCON register. Setting the register's $\overline{PD}$ bit turns off the chip's oscillator. This bit actually enables a gate that turns off the oscillator.

The ability to fabricate n-channel, p-channel, and bipolar devices on the same substrate gives designers considerable flexibility. It should, for example, be easier in the future to combine analog and digital functions on the same chip.

Experiments with memories

C-HMOS has been successfully demonstrated in laboratory memory devices. One is a 4-k static RAM that is functionally identical to the 2147. The chip exhibits the same address and data access times as its HMOS counterpart but consumes one-fifth as much power.

Another experimental 4-k static RAM is compatible with 10K series ECL memories. Here, the memory array was fabricated with six-transistor CMOS devices, and special conversion circuitry was fitted to the front and back ends. These converters change the ECL inputs to MOS inputs and the MOS outputs to ECL outputs. The resulting performance rivals that of the chip's bipolar equivalents. More importantly, the circuit may pave the way for 16-k, and larger, ECL-compatible memories. At those levels, working with bipolar technology becomes very difficult, if not impossible. Thus it may be that higher-capacity ECL-compatible memories will have to be fabricated in an MOS technology.

Lots of room to shrink

Both HMOS and C-HMOS will no doubt undergo scaling down in the future. Interestingly, C-HMOS may have some unique advantages as sizes shrink. For example, p-channel transistors are typically used as load devices. At channel lengths of 2 μm or more, these devices have, typically, one-fourth the gain of n-channel transistors, because of the dominance of carrier mobility. However, as device dimensions shrink, saturation velocity becomes more important than carrier mobility. Therefore, as lengths are scaled down below 2 μm, the gain disparity should improve to about a 2:1 ratio, and the p-channel transistors will be even better load devices.

With scaling down and 5-V operation, it is likely that hot-electron trapping will occur and cause a shift in reference levels. To reduce that effect, it will probably be necessary to lower the power-supply levels commensurately. That is no problem, because with CMOS or C-HMOS technology the reference point will naturally assume the half-voltage level between the supply rails.□

MCS®-51 Data Sheets

11

PRELIMINARY

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

- 8031AH — Control-Oriented CPU with RAM and I/O
- 8051AH — An 8031AH with Factory Mask-Programmable ROM
- Fabricated with Intel's HMOS II Process

- 4K × 8 ROM (8051AH only)
- 128 × 8 RAM
- 32 I/O Lines (Four 8-Bit Ports)
- Two 16-Bit Timer/Counters
- Programmable Full-Duplex Serial Channel
- 128K Accessible External Memory
- Boolean Processor
- 4 μs Multiply and Divide
- 218 User Bit-Addressable Locations
- 100 mA Typical Supply Current

The 8031AH/8051AH is Intel's HMOS II version of the high performance 8-bit 8031/8051 microcomputer. While the 8031AH/8051AH features the same powerful architecture and instruction set as its HMOS I predecessor, it offers the additional benefit of lower power supply current.

The 8031AH/8051AH provides a cost-effective solution for those controller applications requiring up to 64K bytes of program and/or 64K bytes of data storage. Specifically, the 8031AH contains 128 bytes of read/write data memory; 32 I/O lines configured as four 8-bit parallel ports; two 16-bit timer/counters; a five-source, two-priority level, nested interrupt structure; a programmable serial I/O port; and an on-chip oscillator with clock circuitry. The 8051AH has all of these 8031AH features plus 4K bytes of nonvolatile read only program memory. Both microcomputers can use standard TTL compatible memories and most byte-oriented MCS®-80 and MCS-85 peripherals for additional I/O and memory capabilities.

The 8031AH/8051AH microcomputer, characteristic of the entire MCS-51 family, is efficient in both control and computational type applications. This results from extensive BCD/binary arithmetic and bit-handling facilities. The 8031AH/8051AH also makes efficient use of its program memory space with an instruction set consisting of 44% one-byte, 41% two-byte, and 15% three-byte instructions. At 12MHz CPU operation, over half of the instructions execute in just 1.0μs, while the longest instructions, multiply and divide, require only 4μs.

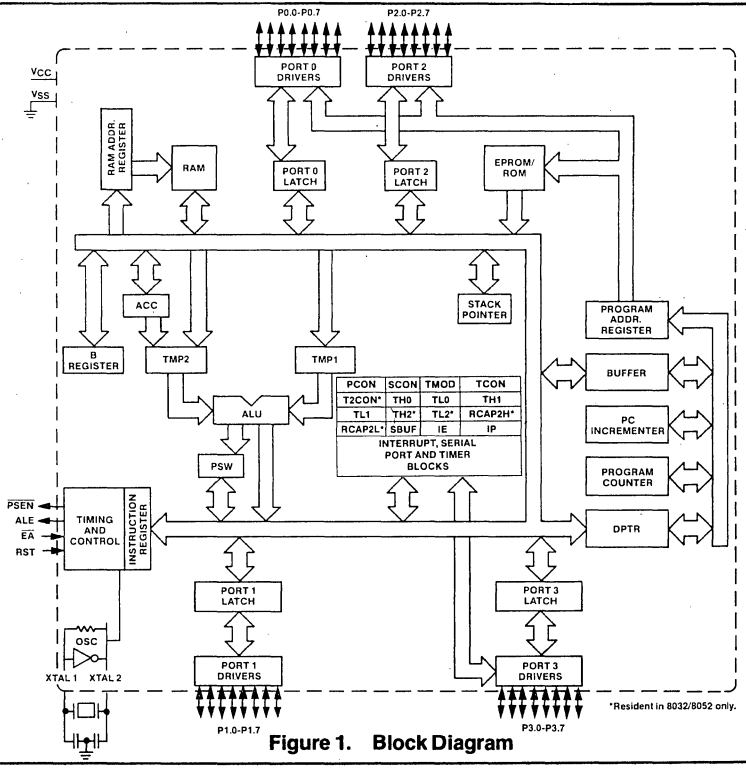


Figure 1. Block Diagram

8031AH/8051AH PIN DESCRIPTIONS

V_{SS}

Circuit ground potential.

V_{CC}

5V power supply input for normal operation and program verification.

Port 0

Port 0 is an 8-bit open drain bidirectional I/O port. It is also the multiplexed low-order address and data bus when using external memory. It is used for data output during program verification. Port 0 can sink (and in bus operations can source) eight LS TTL loads.

Port 1

Port 1 is an 8-bit quasi-bidirectional I/O port. It is also used for the low-order address byte during program verification. Port 1 can sink/source four LS TTL loads.

Port 2

Port 2 is an 8-bit quasi-bidirectional I/O port. It also emits the high-order address byte when accessing external memory. It is used for the high-order address and the control signals during program verification. Port 2 can sink/source four LS TTL loads.

Port 3

Port 3 is an 8-bit bidirectional I/O port with internal pullups. It also serves the functions of various special features of the MCS-51 Family, as listed below:

Port Pin	Alternate Function
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	INT0 (external interrupt)
P3.3	$\overline{\text{INT1}}$ (external interrupt)
P3.4	T0 (Timer/counter 0 external input)
P3.5	T1 (Timer/counter 1 external input)
P3.6	$\overline{\text{WR}}$ (external Data Memory write strobe)
P3.7	$\overline{\text{RD}}$ (external Data Memory read strobe)

The output latch corresponding to a secondary function must be programmed to a one (1) for that function to operate. Port 3 can sink/source four LS TTL loads.

RST

A high on this pin for two machine cycles while the oscillator is running resets the device. A small external pulldown resistor (≈8.2kΩ) from RST to V_{SS} permits power-on reset when a capacitor (≈10 μf) is also connected from this pin to V_{CC}.

ALE

Address Latch Enable output for latching the low byte of the address during accesses to external memory. ALE is activated at a constant rate of 1/6 the oscillator frequency except during an external data memory access at which time one ALE pulse is skipped. ALE can sink/source 8 LS TTL inputs.

$\overline{\text{PSEN}}$

The Program Store Enable output is a control signal that enables the external Program Memory to the bus during external fetch operations. It is activated every six oscillator periods except during external data memory access. $\overline{\text{PSEN}}$ remains high during internal program execution.

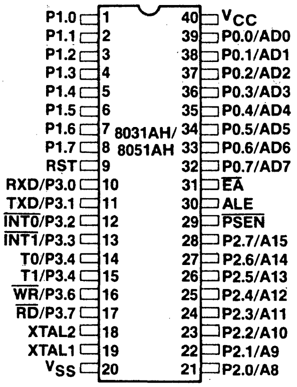


Figure 2. Pin Configuration

$\overline{EA}$

When held at a TTL high level, the 8051AH executes instructions from the internal ROM when the PC is less than 4096. When held at a TTL low level, the 8031AH/8051AH fetches all instructions from external Program Memory. Do not float $\overline{EA}$ during normal operation.

XTAL1

Input to the inverting amplifier that forms part of the oscillator. This pin should be connected to ground when an external oscillator is used.

XTAL2

Output of the inverting amplifier that forms part of the oscillator, and input to the internal clock generator. XTAL2 receives the oscillator signal when an external oscillator is used.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias ...	0°C to 70°C
Storage Temperature	-65°C to +150°C
Voltage on Any Pin With Respect to Ground (V_{SS})	-0.5V to +7V
Power Dissipation	1 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

DC CHARACTERISTICS (T_A = 0° C to 70° C; V_{CC} = 4.5V to 5.5V; V_{SS} = 0V)

Symbol	Parameter	Min	Max	Unit	Test Conditions
VIL	Input Low Voltage	-0.5	0.8	V	
VIH	Input High Voltage (Except RST and XTAL2)	2.0	V_{CC} + 0.5	V	
VIH1	Input High Voltage to RST For Reset, XTAL2	2.5	V_{CC} + 0.5	V	XTAL1 to V_{SS}
VOL	Output Low Voltage Ports 1, 2, 3 (Note 1)		0.45	V	IOL = 1.6mA
VOL1	Output Low Voltage Port 0, ALE, $\overline{PSEN}$ (Note 1)		0.45	V	IOL = 3.2mA
VOH	Output High Voltage Ports 1, 2, 3	2.4		V	IOH = -80μA
VOH1	Output High Voltage Port 0, ALE, $\overline{PSEN}$	2.4		V	IOH = -400μA
IIL	Logical 0 Input Current Ports 1, 2, 3		-800	μA	Vin = 0.45V
IIL2	Logical 0 Input Current for XTAL 2		-2.5	mA	XTAL1 = V_{SS}, Vin = 0.45V
ILI	Input Leakage Current To Port 0, $\overline{EA}$		±10	μA	0.45V < Vin < V_{CC}
IIH1	Input High Current to RST/V_{PD} For Reset		500	μA	Vin < V_{CC} - 1.5V
ICC	Power Supply Current		125	mA	All outputs disconnected
CIO	Capacitance of I/O Buffer		10	pF	f_c = 1MHz, T_A = 25°C

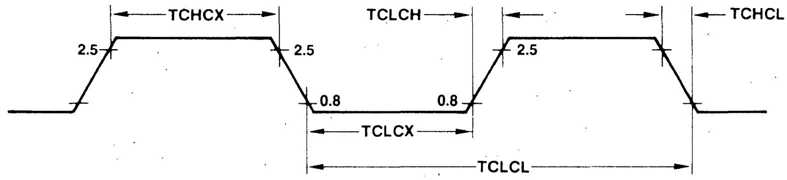
See page 4 for Notes.

Note 1: VOL is degraded when the 8031AH/8051AH rapidly discharges external capacitance. This AC noise is most pronounced during emission of address data. When using external memory, locate the latch or buffer as close to the 8031AH/8051AH as possible.

Datum	Emitting Ports	Degraded I/O Lines	VOL (peak) (max)
Address	P2, P0	P1, P3	0.8V
Write Data	P0	P1, P3, ALE	0.8V

EXTERNAL CLOCK DRIVE CHARACTERISTICS (XTAL2)

Symbol	Parameter	Variable Clock freq = 3.5 MHz to 12 MHz		Unit
		Min	Max	
TCLCL	Oscillator Period	83.3	286	ns
TCHCX	High Time	20		ns
TCLCX	Low Time	20		ns
TCLCH	Rise Time		20	ns
TCHCL	Fall Time		20	ns



AC CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5V ±10%, V_{SS} = 0V, C_L for Port 0, ALE and $\overline{\text{PSEN}}$ Outputs = 100 pF; C_L for all other outputs = 80 pF)

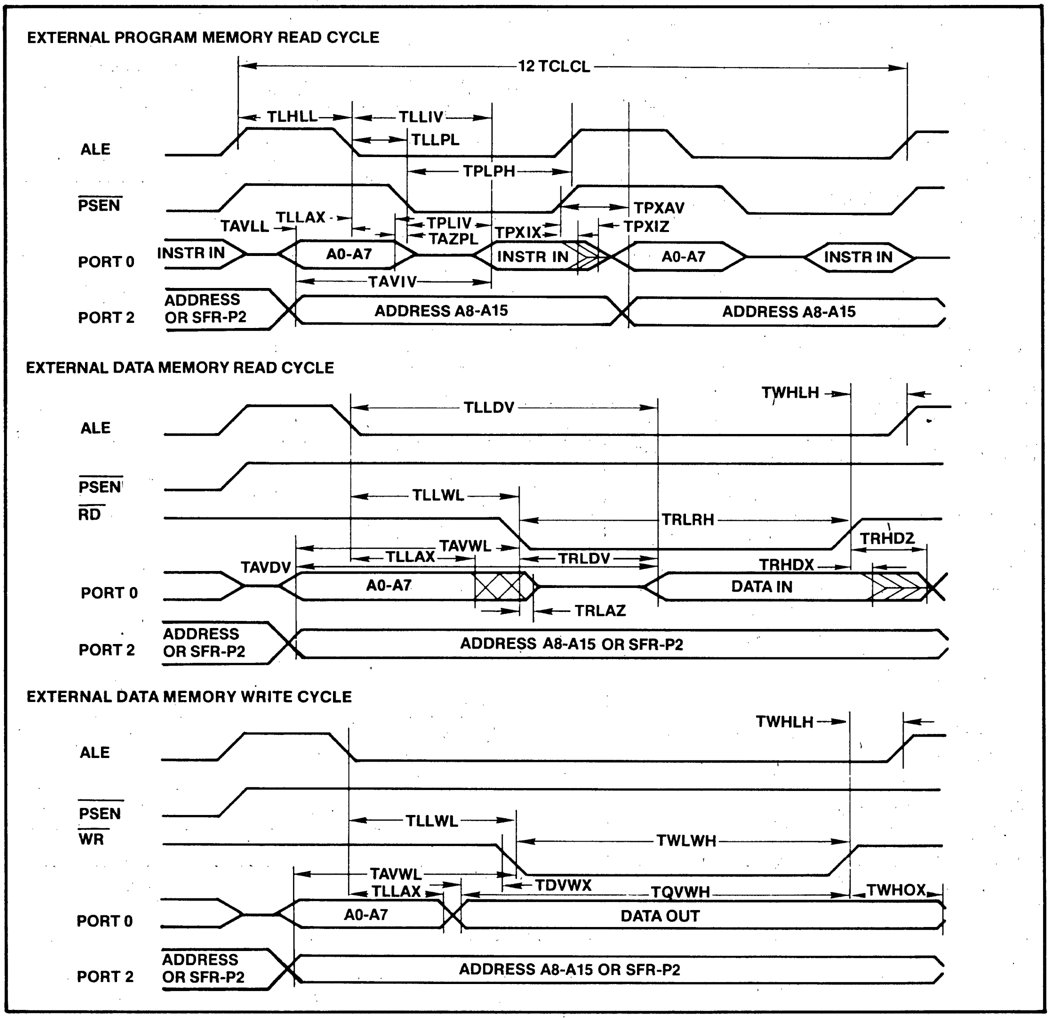
EXTERNAL PROGRAM MEMORY CHARACTERISTICS

		12 MHz Clock			Variable Clock 1/TCLCL = 3.5 MHz to 12 MHz		
Symbol	**Parameter**	**Min**	**Max**	**Unit**	**Min**	**Max**	**Unit**
TLHLL	ALE Pulse Width	127		ns	2TCLCL-40		ns
TAVLL	Address Setup to ALE	43		ns	TCLCL-40		ns
TLLAX	Address Hold After ALE	48		ns	TCLCL-35		ns
TLLIV	ALE to Valid Instr In		233	ns		4TCLCL-100	ns
TLLPL	ALE To $\overline{\text{PSEN}}$	58		ns	TCLCL-25		ns
TPLPH	$\overline{\text{PSEN}}$ Pulse Width	215		ns	3TCLCL-35		ns
TPLIV	$\overline{\text{PSEN}}$ To Valid Instr In		125	ns		3TCLCL-125	ns
TPXIX	Input Instr Hold After $\overline{\text{PSEN}}$	0		ns	0		ns
TPXIZ	Input Instr Float After $\overline{\text{PSEN}}$		63	ns		TCLCL-20	ns
TPXAV	Address Valid After $\overline{\text{PSEN}}$	75		ns	TCLCL-8		ns
TAVIV	Address To Valid Instr In		302	ns		5TCLCL-115	ns
TAZPL	Address Float To $\overline{\text{PSEN}}$	0		ns	0		ns

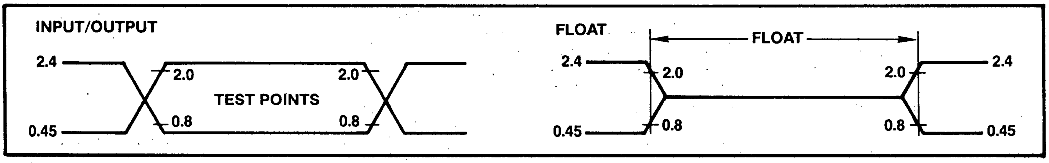
EXTERNAL DATA MEMORY CHARACTERISTICS

		12 MHz Clock			Variable Clock 1/TCLCL = 3.5 MHz to 12 MHz		
Symbol	**Parameter**	**Min**	**Max**	**Unit**	**Min**	**Max**	**Unit**
TRLRH	$\overline{\text{RD}}$ Pulse Width	400		ns	6TCLCL-100		ns
TWLWH	$\overline{\text{WR}}$ Pulse Width	400		ns	6TCLCL-100		ns
TLLAX	Address Hold After ALE	48		ns	TCLCL-35		
TRLDV	$\overline{\text{RD}}$ To Valid Data In		250	ns		5TCLCL-165	ns
TRHDX	Data Hold After $\overline{\text{RD}}$	0		ns	0		ns
TRHDZ	Data Float After $\overline{\text{RD}}$		97	ns		2TCLCL-70	ns
TLLDV	ALE To Valid Data In		517	ns		8TCLCL-150	ns
TAVDV	Address To Valid Data In		585	ns		9TCLCL-165	ns
TLLWL	ALE To $\overline{\text{WR}}$ or $\overline{\text{RD}}$	200	300	ns	3TCLCL-50	3TCLCL + 50	ns
TAVWL	Address To $\overline{\text{WR}}$ or $\overline{\text{RD}}$	203		ns	4TCLCL-130		ns
TWHLH	$\overline{\text{WR}}$ or $\overline{\text{RD}}$ High To ALE High	43	123	ns	TCLCL-40	TCLCL + 40	ns
TDVWX	Data Valid To $\overline{\text{WR}}$ Transition	23		ns	TCLCL-60		ns
TQVWH	Data Setup Before $\overline{\text{WR}}$	433		ns	7TCLCL-150		ns
TWHQX	Data Hold After $\overline{\text{WR}}$	33		ns	TCLCL-50		ns
TRLAZ	Address Float After $\overline{\text{RD}}$		0	ns		0	ns

AC TIMING DIAGRAMS

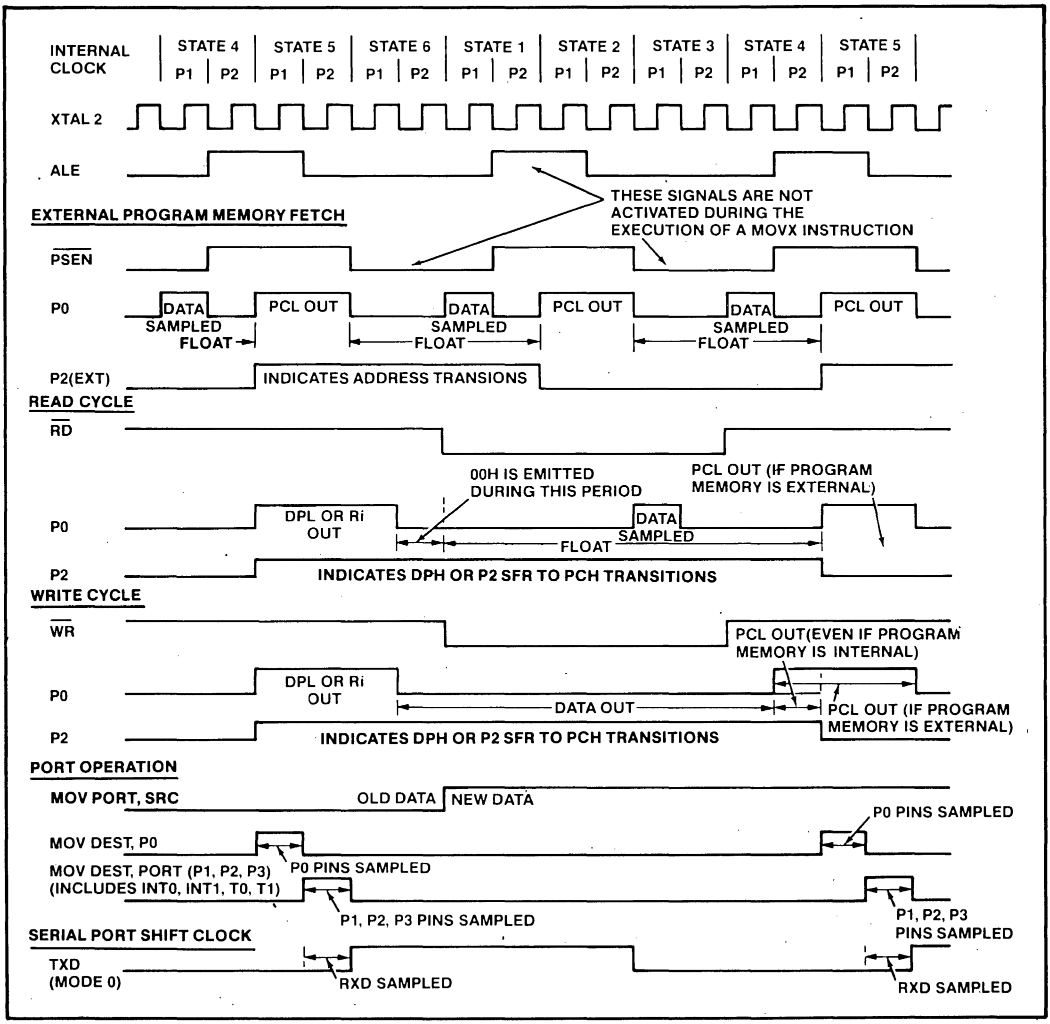


AC TESTING INPUT/OUTPUT, FLOAT WAVEFORMS



AC inputs during testing are driven at 2.4V for a logic "1" and 0.45V for a logic "0". Timing measurements are made at 2.0V for a logic "1" and 0.8V for a logic "0". For timing purposes, the float state is defined as the point at which a P0 pin sinks 3.2mA or sources 400 μA at the voltage test levels.

CLOCK WAVEFORMS


INTERNAL CLOCK
STATE 4
STATE 5
STATE 6
STATE 1
STATE 2
STATE 3
STATE 4
STATE 5
P1
P2
XTAL 2
ALE
THESE SIGNALS ARE NOT ACTIVATED DURING THE EXECUTION OF A MOVX INSTRUCTION
EXTERNAL PROGRAM MEMORY FETCH
PSEN
P0
DATA
SAMPLED
FLOAT
PCL OUT
P2(EXT)
INDICATES ADDRESS TRANSIONS
READ CYCLE
RD
00H IS EMITTED DURING THIS PERIOD
PCL OUT (IF PROGRAM MEMORY IS EXTERNAL)
P0
DPL OR Ri OUT
DATA SAMPLED
FLOAT
P2
INDICATES DPH OR P2 SFR TO PCH TRANSITIONS
WRITE CYCLE
WR
PCL OUT(EVEN IF PROGRAM MEMORY IS INTERNAL)
P0
DPL OR Ri OUT
DATA OUT
PCL OUT (IF PROGRAM MEMORY IS EXTERNAL)
P2
INDICATES DPH OR P2 SFR TO PCH TRANSITIONS
PORT OPERATION
MOV PORT, SRC
OLD DATA
NEW DATA
P0 PINS SAMPLED
MOV DEST, P0
MOV DEST, PORT (P1, P2, P3) (INCLUDES INT0, INT1, T0, T1)
P0 PINS SAMPLED
P1, P2, P3 PINS SAMPLED
P1, P2, P3 PINS SAMPLED
SERIAL PORT SHIFT CLOCK
TXD (MODE 0)
RXD SAMPLED
RXD SAMPLED

This diagram indicates when signals are clocked internally. The time it takes the signals to propagate to the pins, however, ranges from 25 to 125 ns. This propagation delay is dependent on variables such as temperature and pin loading. Propagation also varies from output to output and component to component. Typically though, (T_A = 25° C, fully loaded) RD and WR propagation delays are approximately 50 ns. The other signals are typically 85 ns. Propagation delays are incorporated in the AC specifications.

Table 1. MCS®-51 Instruction Set Description

ARITHMETIC OPERATIONS

Mnemonic		Description	Byte	Cyc
ADD	A,Rn	Add register to Accumulator	1	1
ADD	A,direct	Add direct byte to Accumulator	2	1
ADD	A,@Ri	Add indirect RAM to Accumulator	1	1
ADD	A,#data	Add immediate data to Accumulator	2	1
ADDC	A,Rn	Add register to Accumulator with Carry	1	1
ADDC	A,direct	Add direct byte to A with Carry flag	2	1
ADDC	A,@Ri	Add indirect RAM to A with Carry flag	1	1
ADDC	A,#data	Add immediate data to A with Carry flag	2	1
SUBB	A,Rn	Subtract register from A with Borrow	1	1
SUBB	A,direct	Subtract direct byte from A with Borrow	2	1
SUBB	A,@Ri	Subtract indirect RAM from A with Borrow	1	1
SUBB	A,#data	Subtract immed data from A with Borrow	2	1
INC	A	Increment Accumulator	1	1
INC	Rn	Increment register	1	1
INC	direct	Increment direct byte	2	1
INC	@Ri	Increment indirect RAM	1	1
INC	DPTR	Increment Data Pointer	1	2
DEC	A	Decrement Accumulator	1	1
DEC	Rn	Decrement register	1	1
DEC	direct	Decrement direct byte	2	1
DEC	@Ri	Decrement indirect RAM	1	1
MUL	AB	Multiply A & B	1	4
DIV	AB	Divide A by B	1	4
DA	A	Decimal Adjust Accumulator	1	1

LOGICAL OPERATIONS

Mnemonic		Destination	Byte	Cyc
ANL	A,Rn	AND register to Accumulator	1	1
ANL	A,direct	AND direct byte to Accumulator	2	1
ANL	A,@Ri	AND indirect RAM to Accumulator	1	1
ANL	A,#data	AND immediate data to Accumulator	2	1
ANL	direct,A	AND Accumulator to direct byte	2	1
ANL	direct,#data	AND immediate data to direct byte	3	2
ORL	A,Rn	OR register to Accumulator	1	1
ORL	A,direct	OR direct byte to Accumulator	2	1

LOGICAL OPERATIONS (CONTINUED)

Mnemonic		Destination	Byte	Cyc
ORL	A,@Ri	OR indirect RAM to Accumulator	1	1
ORL	A,#data	OR immediate data to Accumulator	2	1
ORL	direct,A	OR Accumulator to direct byte	2	1
ORL	direct,#data	OR immediate data to direct byte	3	2
XRL	A,Rn	Exclusive-OR register to Accumulator	1	1
XRL	A,direct	Exclusive-OR direct byte to Accumulator	2	1
XRL	A,@Ri	Exclusive-OR indirect RAM to A	1	1
XRL	A,#data	Exclusive-OR immediate data to A	2	1
XRL	direct,A	Exclusive-OR Accumulator to direct byte	2	1
XRL	direct,#data	Exclusive-OR immediate data to direct	3	2
CLR	A	Clear Accumulator	1	1
CPL	A	Complement Accumulator	1	1
RL	A	Rotate Accumulator Left	1	1
RLC	A	Rotate A Left through the Carry flag	1	1
RR	A	Rotate Accumulator Right	1	1
RRC	A	Rotate A Right through Carry flag	1	1
SWAP	A	Swap nibbles within the Accumulator	1	1

DATA TRANSFER

Mnemonic		Description	Byte	Cyc
MOV	A,Rn	Move register to Accumulator	1	1
MOV	A,direct	Move direct byte to Accumulator	2	1
MOV	A,@Ri	Move indirect RAM to Accumulator	1	1
MOV	A,#data	Mov immediate data to Accumulator	2	1
MOV	Rn,A	Move Accumulator to register	1	1
MOV	Rn,direct	Move direct byte to register	2	2
MOV	Rn,#data	Move immediate data to register	2	1
MOV	direct,A	Move Accumulator to direct byte	2	1
MOV	direct,Rn	Move register to direct byte	2	2
MOV	direct,direct	Move direct byte to direct	3	2
MOV	direct,@Ri	Move indirect RAM to direct byte	2	2

Table 1. (Cont.)

DATA TRANSFER (CONTINUED)

Mnemonic		Description	Byte	Cyc
MOV	direct,#data	Move immediate data to direct byte	3	2
MOV	@Ri,A	Move Accumulator to indirect RAM	1	1
MOV	@Ri,direct	Move direct byte to indirect RAM	2	2
MOV	@Ri,#data	Move immediate data to indirect RAM	2	1
MOV	DPTR,#data16	Load Data Pointer with a 16-bit constant	3	2
MOVC	A,@A+DPTR	Move Code byte relative to DPTR to A	1	2
MOVC	A,@A+PC	Move Code byte relative to PC to A	1	2
MOVX	A,@Ri	Move External RAM (8-bit addr) to A	1	2
MOVX	A,@DPTR	Move External RAM (16-bit addr) to A	1	2
MOVX	@Ri,A	Move A to External RAM (8-bit addr)	1	2
MOVX	@DPTR,A	Move A to External RAM (16-bit addr)	1	2
PUSH	direct	Push direct byte onto stack	2	2
POP	direct	Pop direct byte from stack	2	2
XCH	A,Rn	Exchange register with Accumulator	1	1
XCH	A,direct	Exchange direct byte with Accumulator	2	1
XCH	A,@Ri	Exchange indirect RAM with A	1	1
XCHD	A,@Ri	Exchange low-order Digit ind RAM w A	1	1

BOOLEAN VARIABLE MANIPULATION

Mnemonic		Description	Byte	Cyc
CLR	C	Clear Carry flag	1	1
CLR	bit	Clear direct bit	2	1
SETB	C	Set Carry flag	1	1
SETB	bit	Set direct Bit	2	1
CPL	C	Complement Carry flag	1	1
CPL	bit	Complement direct bit	2	1
ANL	C,bit	AND direct bit to Carry flag	2	2
ANL	C,1 bit	AND complement of direct bit to Carry	2	2
ORL	C/bit	OR direct bit to Carry flag	2	2
ORL	C,1 bit	OR complement of direct bit to Carry	2	2
MOV	C/bit	Move direct bit to Carry flag	2	1
MOV	bit,C	Move Carry flag to direct bit	2	2

PROGRAM AND MACHINE CONTROL

Mnemonic		Description	Byte	Cyc
ACALL	addr11	Absolute Subroutine Call	2	2
LCALL	addr16	Long Subroutine Call	3	2
RET		Return from subroutine	1	2
RETI		Return from interrupt	1	2
AJMP	addr11	Absolute Jump	2	2
LJMP	addr16	Long Jump	3	2
SJMP	rel	Short Jump (relative addr)	2	2
JMP	@A+DPTR	Jump indirect relative to the DPTR	1	2
JZ	rel	Jump if Accumulator is Zero	2	2
JNZ	rel	Jump if Accumulator is Not Zero	2	2
JC	rel	Jump if Carry flag is set	2	2
JNC	rel	Jump if No Carry flag	2	2
JB	bit,rel	Jump if direct Bit set	3	2
JNB	bit,rel	Jump if direct Bit Not set	3	2
JBC	bit,rel	Jump if direct Bit is set & Clear bit	3	2
CJNE	A,direct,rel	Compare direct to A & Jump if Not Equal	3	2
CJNE	A,#data,rel	Comp, immed, to A & Jump if Not Equal	3	2
CJNE	Rn,#data,rel	Comp, immed, to reg & Jump if Not Equal	3	2
CJNE	@Ri,#data,rel	Comp, immed, to ind, & Jump if Not Equal	3	2
DJNZ	Rn,rel	Decrement register & Jump if Not Zero	2	2
DJNZ	direct,rel	Decrement direct & Jump if Not Zero	3	2
NOP		No operation	1	1

Notes on data addressing modes:

Rn —Working register R0-R7

direct —128 internal RAM locations, any I/O port, control or status register

@Ri —Indirect internal RAM location addressed by register R0 or R1

#data —8-bit constant included in instruction

#data16 —16-bit constant included as bytes 2 & 3 of instruction

bit —128 software flags, any I/O pin, control or status bit

Notes on program addressing modes:

addr16 —Destination address for LCALL & LJMP may be anywhere within the 64-K program memory address space

Addr11 —Destination address for ACALL & AJMP will be within the same 2-K page of program memory as the first byte of the following instruction

rel —SJMP and all conditional jumps include an 8-bit offset byte, Range is +127-128 bytes relative to first byte of the following instruction

Table 2. Instruction Opcodes in Hexadecimal Order

Hex Code	Number of Bytes	Mnemonic	Operands
00	1	NOP	
01	2	AJMP	code addr
02	3	LJMP	code addr
03	1	RR	A
04	1	INC	A
05	2	INC	data addr
06	1	INC	@R0
07	1	INC	@R1
08	1	INC	R0
09	1	INC	R1
0A	1	INC	R2
0B	1	INC	R3
0C	1	INC	R4
0D	1	INC	R5
0E	1	INC	R6
0F	1	INC	R7
10	3	JBC	bit addr, code addr
11	2	ACALL	code addr
12	3	LCALL	code addr
13	1	RRC	A
14	1	DEC	A
15	2	DEC	data addr
16	1	DEC	@R0
17	1	DEC	@R1
18	1	DEC	R0
19	1	DEC	R1
1A	1	DEC	R2
1B	1	DEC	R3
1C	1	DEC	R4
1D	1	DEC	R5
1E	1	DEC	R6
1F	1	DEC	R7
20	3	JB	bit addr, code addr
21	2	AJMP	code addr
22	1	RET	
23	1	RL	A
24	2	ADD	A,#data
25	2	ADD	A,data addr
26	1	ADD	A,@R0
27	1	ADD	A,@R1
28	1	ADD	A,R0
29	1	ADD	A,R1
2A	1	ADD	A,R2
2B	1	ADD	A,R3
2C	1	ADD	A,R4
2D	1	ADD	A,R5
2E	1	ADD	A,R6
2F	1	ADD	A,R7
30	3	JNB	bit addr, code addr
31	2	ACALL	code addr
32	1	RETI	
33	1	RLC	A
34	2	ADDC	A,#data
35	2	ADDC	A,data addr
36	1	ADDC	A,@R0
37	1	ADDC	A,@R1
38	1	ADDC	A,R0
39	1	ADDC	A,R1
3A	1	ADDC	A,R2
3B	1	ADDC	A,R3
3C	1	ADDC	A,R4
3D	1	ADDC	A,R5
3E	1	ADDC	A,R6
3F	1	ADDC	A,R7
40	2	JC	code addr
41	2	AJMP	code addr
42	2	ORL	data addr,A
43	3	ORL	data addr,#data
44	2	ORL	A,#data
45	2	ORL	A,data addr
46	1	ORL	A,@R0
47	1	ORL	A,@R1
48	1	ORL	A,R0
49	1	ORL	A,R1
4A	1	ORL	A,R2
4B	1	ORL	A,R3
4C	1	ORL	A,R4
4D	1	ORL	A,R5
4E	1	ORL	A,R6
4F	1	ORL	A,R7
50	2	JNC	code addr
51	2	ACALL	code addr
52	2	ANL	data addr,A
53	3	ANL	data addr,#data
54	2	ANL	A,#data
55	2	ANL	A,data addr
56	1	ANL	A,@R0
57	1	ANL	A@R1
58	1	ANL	A,R0
59	1	ANL	A,R1
5A	1	ANL	A,R2
5B	1	ANL	A,R3
5C	1	ANL	A,R4
5D	1	ANL	A,R5
5E	1	ANL	A,R6
5F	1	ANL	A,R7
60	2	JZ	code addr
61	2	AJMP	code addr
62	2	XRL	data addr.A
63	3	XRL	data addr,#data
64	2	XRL	A,#data
65	2	XRL	A,data addr

Table 2. (Cont.)

Hex Code	Number of Bytes	Mnemonic	Operands
66	1	XRL	A,@R0
67	1	XRL	A,@R1
68	1	XRL	A,R0
69	1	XRL	A,R1
6A	1	XRL	A,R2
6B	1	XRL	A,R3
6C	1	XRL	A,R4
6D	1	XRL	A,R5
6E	1	XRL	A,R6
6F	1	XRL	A,R7
70	2	JNZ	code addr
71	2	ACALL	code addr
72	2	ORL	C,bit addr
73	1	JMP	@A+DPTR
74	2	MOV	A,#data
75	3	MOV	data addr,#data
76	2	MOV	@R0,#data
77	2	MOV	@R1,#data
78	2	MOV	R0,#data
79	2	MOV	R1,#data
7A	2	MOV	R2,#data
7B	2	MOV	R3,#data
7C	2	MOV	R4,#data
7D	2	MOV	R5,#data
7E	2	MOV	R6,#data
7F	2	MOV	R7,#data
80	2	SJMP	code addr
81	2	AJMP	code addr
82	2	ANL	C,bit addr
83	1	MOVC	A,@A+PC
84	1	DIV	AB
85	3	MOV	data addr, data addr
86	2	MOV	data addr,@R0
87	2	MOV	data addr,@R1
88	2	MOV	data addr,R0
89	2	MOV	data addr,R1
8A	2	MOV	data addr,R2
8B	2	MOV	data addr,R3
8C	2	MOV	data addr,R4
8D	2	MOV	data addr,R5
8E	2	MOV	data addr,R6
8F	2	MOV	data addr,R7
90	3	MOV	DPTR,#data
91	2	ACALL	code addr
92	2	MOV	bit addr,C
93	1	MOVC	A,@A+DPTR
94	2	SUBB	A,#data
95	2	SUBB	A,data addr
96	1	SUBB	A,@R0
97	1	SUBB	A,@R1
98	1	SUBB	A,R0

Hex Code	Number of Bytes	Mnemonic	Operands
99	1	SUBB	A,R1
9A	1	SUBB	A,R2
9B	1	SUBB	A,R3
9C	1	SUBB	A,R4
9D	1	SUBB	A,R5
9E	1	SUBB	A,R6
9F	1	SUBB	A,R7
A0	2	ORL	C,/bit addr
A1	2	AJMP	code addr
A2	2	MOV	C,bit addr
A3	1	INC	DPTR
A4	1	MUL	AB
A5		reserved	
A6	2	MOV	@R0, data addr
A7	2	MOV	@R1,data addr
A8	2	MOV	R0,data addr
A9	2	MOV	R1,data addr
AA	2	MOV	R2,data addr
AB	2	MOV	R3,data addr
AC	2	MOV	R4,data addr
AD	2	MOV	R5,data addr
AE	2	MOV	R6,data addr
AF	2	MOV	R7,data addr
B0	2	ANL	C,/bit addr
B1	2	ACALL	code addr
B2	2	CPL	bit addr
B3	1	CPL	C
B4	3	CJNE	A,#data,code addr
B5	3	CJNE	A,data addr,code addr
B6	3	CJNE	@R0,#data,code addr
B7	3	CJNE	@R1,#data,code addr
B8	3	CJNE	R0,#data,code addr
B9	3	CJNE	R1,#data,code addr
BA	3	CJNE	R2,#data,code addr
BB	3	CJNE	R3,#data,code addr
BC	3	CJNE	R4,#data,code addr
BD	3	CJNE	R5,#data,code addr
BE	3	CJNE	R6,#data,code addr
BF	3	CJNE	R7,#data,code addr
C0	2	PUSH	data addr
C1	2	AJMP	code addr
C2	2	CLR	bit addr
C3	1	CLR	C
C4	1	SWAP	A
C5	2	XCH	A,data addr
C6	1	XCH	A,@R0
C7	1	XCH	A,@R1
C8	1	XCH	A,R0
C9	1	XCH	A,R1
CA	1	XCH	A,R2
CB	1	XCH	A,R3

Table 2. (Cont.)

Hex Code	Number of Bytes	Mnemonic	Operands
CC	1	XCH	A,R4
CD	1	XCH	A,R5
CE	1	XCH	A,R6
CF	1	XCH	A,R7
D0	2	POP	data addr
D1	2	ACALL	code addr
D2	2	SETB	bit addr
D3	1	SETB	C
D4	1	DA	A
D5	3	DJNZ	data addr,code addr
D6	1	XCHD	A,@R0
D7	1	XCHD	A,@R1
D8	2	DJNZ	R0,code addr
D9	2	DJNZ	R1,code addr
DA	2	DJNZ	R2,code addr
DB	2	DJNZ	R3,code addr
DC	2	DJNZ	R4,code addr
DD	2	DJNZ	R5,code addr
DE	2	DJNZ	R6,code addr
DF	2	DJNZ	R7,code addr
E0	1	MOVX	A,@DPTR
E1	2	AJMP	code addr
E2	1	MOVX	A,@R0
E3	1	MOVX	A,@R1
E4	1	CLR	A
E5	2	MOV	A,data addr

Hex Code	Number of Bytes	Mnemonic	Operands
E6	1	MOV	A,@R0
E7	1	MOV	A,@R1
E8	1	MOV	A,R0
E9	1	MOV	A,R1
EA	1	MOV	A,R2
EB	1	MOV	A,R3
EC	1	MOV	A,R4
ED	1	MOV	A,R5
EE	1	MOV	A,R6
EF	1	MOV	A,R7
F0	1	MOVX	@DPTR,A
F1	2	ACALL	code addr
F2	1	MOVX	@R0,A
F3	1	MOVX	@R1,A
F4	1	CPL	A
F5	2	MOV	data addr,A
F6	1	MOV	@R0,A
F7	1	MOV	@R1,A
F8	1	MOV	R0,A
F9	1	MOV	R1,A
FA	1	MOV	R2,A
FB	1	MOV	R3,A
FC	1	MOV	R4,A
FD	1	MOV	R5,A
FE	1	MOV	R6,A
FF	1	MOV	R7,A

intel®

PRELIMINARY

8032AH/8052AH
SINGLE-COMPONENT 8-BIT MICROCOMPUTER

- **8032AH—Control-Oriented CPU with RAM and I/O**
- **8052AH—An 8032AH with Factory Mask-Programmable ROM**

- **8K x 8 ROM (8052AH only)**
- **256 x 8 RAM**
- **32 I/O lines (Four 8-Bit Ports)**
- **Three 16-Bit Timer/Counters**
- **Programmable Full-Duplex Serial Channel**
- **Variable Transmit/Receive Baud Rate Capability**
- **Timer 2 Capture Capability**
- **128K Accessible External Memory**
- **Boolean Processor**
- **218 User Bit-Addressable Locations**
- **Upward Compatible with 8031AH/8051AH**

The 8032AH/8052AH is the highest performance member of Intel's MCS®-51 family of 8-bit microcomputers. It is fabricated with Intel's highly reliable +5 depletion-load, N-channel, silicon gate HMOS-II technology, and like the other MCS-51 members is packaged in a 40-pin DIP.

The 8032AH contains 256 bytes of read/write data memory; 32 I/O lines configured as four 8-bit ports; three 16-bit timer/counters; a six-source, two-priority level, nested interrupt structure; a programmable serial I/O port; and an on-chip oscillator with clock circuitry. The 8052AH has all of these features plus 8K bytes of nonvolatile read-only program memory. Both microcomputers have memory expansion capabilities of up to 64K bytes of data storage and 64K bytes of program memory that may be realized with standard TTL compatible memories and/or byte-oriented MCS-80 and MCS-85 peripherals.

The 8032AH/8052AH microcomputer, characteristic of the entire MCS-51 family, is efficient at both computational and control-oriented tasks. This results from its extensive BCD/binary arithmetic and bit-handling facilities. Efficient use of program memory is also achieved by using the familiar compact instruction set of the 8031/8051. Forty-four percent of the instructions are one-byte, 41% two-byte and 15% three-byte. The majority of the instructions execute in just 1.0 μs at 12 MHz operation. The longest instructions, multiply and divide, require only 4 μs at 12 MHz.

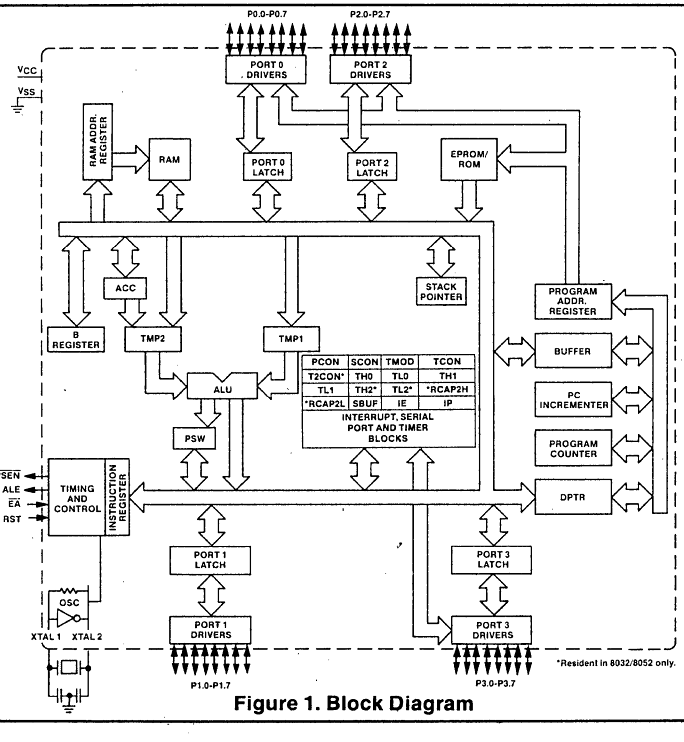


Figure 1. Block Diagram

8032AH/8052AH PIN DESCRIPTIONS

V_{SS}

Circuit ground potential.

V_{CC}

+5V power supply during operation and program verification.

Port 0

Port 0 is an 8-bit open drain bidirectional I/O port. It is also the multiplexed low-order address and data bus when using external memory. It is used for data output during program verification. Port 0 can sink (and in bus operations can source) eight LS TTL loads.

Port 1

Port 1 is an 8-bit quasi-bidirectional I/O port. Pins P1.0 and P1.1 also correspond to the special functions T2, Timer 2 counter trigger input, and T2EX, external input to Timer 2. The output latch on these two special function pins must be programmed to a one (1) for that function to operate. Port 1 is also used for the low-order address byte during program verification. Port 1 can sink/source four LS TTL loads.

Port 2

Port 2 is an 8-bit quasi-bidirectional I/O port. It also emits the high-order address byte when accessing external memory. It is used for the high-order address and the control signals during program verification. Port 2 can sink/source four LS TTL loads.

Port 3

Port 3 is an 8-bit quasi-bidirectional I/O port. Each of the P3 pins also correspond to special functions as listed below:

Port Pin	Alternate Function
P3.0	RXD (serial input/output port)
P3.1	TXD (serial output port)
P3.2	$\overline{\text{INT0}}$ (external interrupt 0 input)
P3.3	$\overline{\text{INT1}}$ (external interrupt 1 input)
P3.4	T0 (Timer/Counter 0 external input)
P3.5	T1 (Timer/Counter 1 external input)
P3.6	$\overline{\text{WR}}$ (external Data Memory write strobe)
P3.7	$\overline{\text{RD}}$ (external Data Memory read strobe)

The output latch corresponding to a secondary function must be programmed to a one (1) for that function to operate. Port 3 can sink/source four LS TTL loads.

RST

A high on this pin for two machine cycles while the oscillator is running resets the device. A small external pulldown resistor (≈8.2kΩ) from RST to V_{SS} permits power-on reset when a capacitor (≈10 μf) is also connected from this pin to V_{CC}.

ALE

Provides Address Latch Enable output used for latching the address into external memory during normal operation. It is activated every six oscillator periods except during an external data memory access.

$\overline{\text{PSEN}}$

The Program Store Enable output is a control signal that enables the external Program Memory to the bus during external fetch operations. It is activated every six oscillator periods, except during external data memory accesses. Remains high during internal program execution.

$\overline{\text{EA}}$

When held at a TTL high level, the 8052AH executes instructions from the internal ROM when the PC is less than 8192. When held at a TTL low level, the 8032AH/8052AH fetches all instructions from external Program Memory.

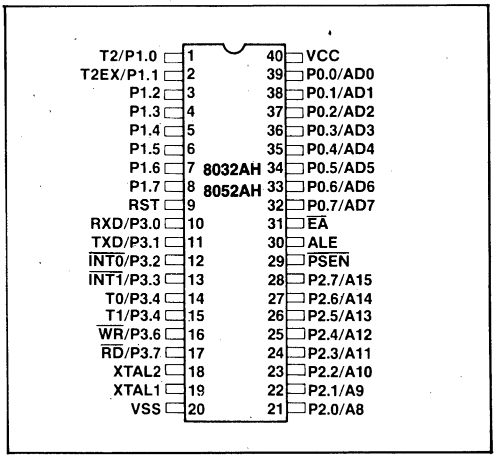


Figure 2. Pin Configuration

XTAL1

Input to the inverting amplifier that forms the oscillator.

XTAL2

Output of the inverting amplifier that forms the oscillator, and input to the internal clock generator. Receives the external oscillator signal when an external oscillator is used.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias ... 0°C to 70°C

Storage Temperature .... -65°C to +150°C

Voltage on Any Pin With
Respect to Ground (V_{SS}) .... -0.5V to +7V

Power Dissipation .... 2 Watts

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

DC CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 4.5V to 5.5V, V_{SS} = 0V)

Symbol	Parameter	Min	Max	Unit	Test Conditions
VIL	Input Low Voltage	-0.5	0.8	V	
VIH	Input High Voltage (Except RST and XTAL2)	2.0	V_{CC} + 0.5	V	
VIH1	Input High Voltage to RST for Reset, XTAL2	2.5	V_{CC} + 0.5	V	XTAL1 to V_{SS}
VOL	Output Low Voltage Ports 1, 2, 3 (Note 1)		0.45	V	IOL = 1.6mA
VOL1	Output Low Voltage Port 0, ALE,PSEN (Note 1)		0.45	V	IOL = 3.2mA
VOH	Output High Voltage Ports 1, 2, 3	2.4		V	IOH = -80μA
VOH1	Output High Voltage Port 0, ALE, $\overline{\text{PSEN}}$	2.4		V	IOH = -400μA
IIL	Logical 0 Input Current Ports 1, 2, 3		-800	μA	Vin = 0.45V
IIL2	Logical 0 Input Current XTAL2		-2.5	mA	XTAL1 at V_{SS}, Vin=0.45V
ILI	Input Leakage Current To Port 0, EA		±10	μA	0.45V < Vin < V_{CC}
IIH1	Input High Current to RST/VPD For Reset		500	μA	Vin = V_{CC} - 1.5V
ICC	Power Supply Current		175	mA	All outputs disconnected; $\overline{\text{EA}}$ = V_{CC}
CIO	Capacitance of I/O Buffer		10	pF	f_C = 1MHz, T_A = 25°C

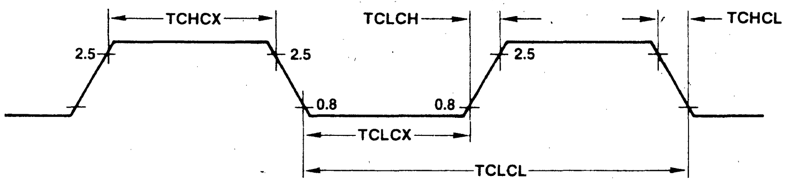
See page 4 for Notes.

Note 1: Vol is degraded when the 8032AH/8052AH rapidly discharges external capacitance. This AC noise is most pronounced during emission of address data. When using external memory, locate the latch or buffer as close to the 8032AH/8052AH as possible.

Datum	Emitting Ports	Degraded I/O Lines	VOL (peak) (max)
Address	P2, P0	P1, P3	0.8V
Write Data	P0	P1, P3, ALE	0.8V

EXTERNAL CLOCK DRIVE CHARACTERISTICS (XTAL2)

		Variable Clock f = 3.5 MHz to 12 MHz		
Symbol	Parameter	Min	Max	Unit
TCLCL	Oscillator Period	83.3	286	ns
TCHCX	High Time	20		ns
TCLCX	Low Time	20		ns
TCLCH	Rise Time		20	ns
TCHCL	Fall Time		20	ns



AC CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5V ±10%, V_{SS} = 0V, C_L for Port 0, ALE and $\overline{\text{PSEN}}$ Outputs = 100 pF, C_L for all other outputs = 80 pF)

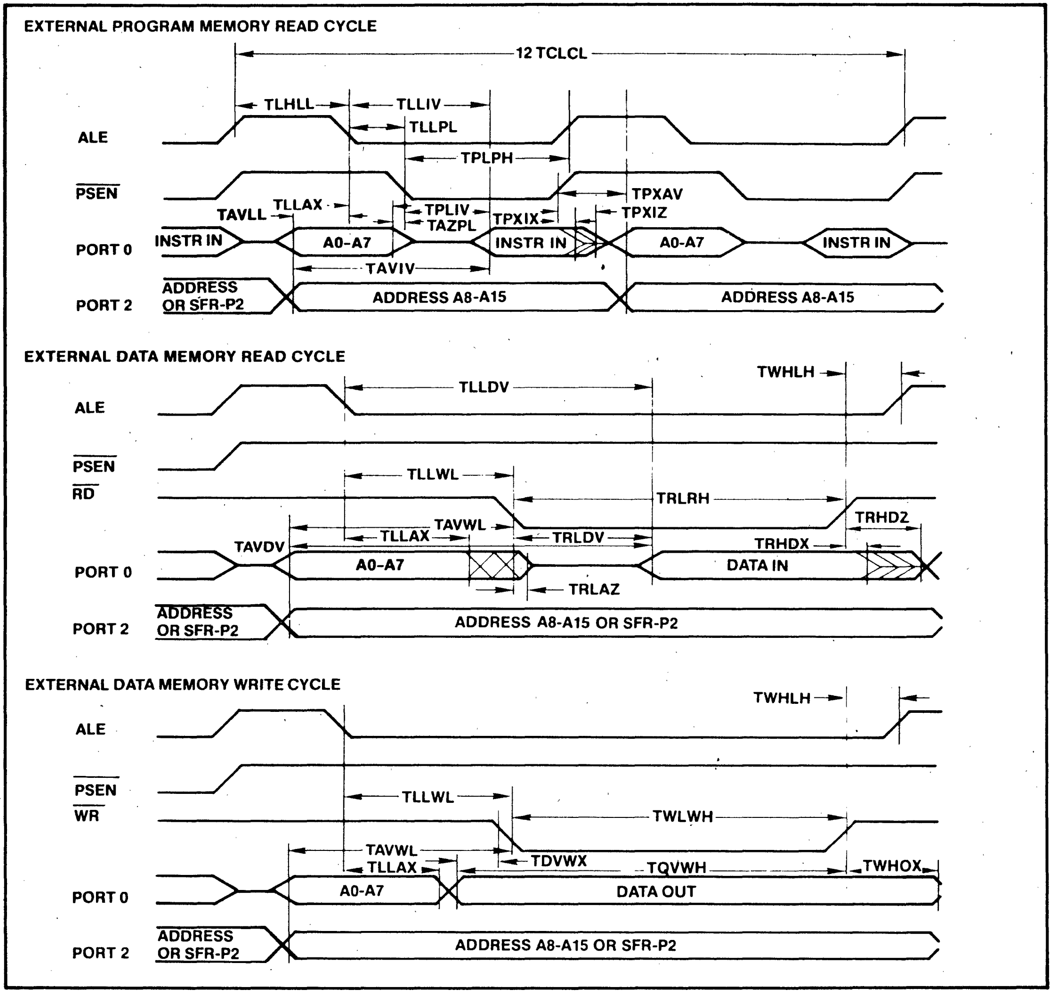
PROGRAM MEMORY CHARACTERISTICS

		12 MHz Clock			Variable Clock 1/TCLCL = 3.5 MHz to 12 MHz		
Symbol	**Parameter**	**Min**	**Max**	**Unit**	**Min**	**Max**	**Unit**
TLHLL	ALE Pulse Width	127		ns	2TCLCL-40		ns
TAVLL	Address Setup to ALE	43		ns	TCLCL-40		ns
TLLAX	Address Hold After ALE	48		ns	TCLCL-35		ns
TLLIV	ALE to Valid Instr In		233	ns		4TCLCL-100	ns
TLLPL	ALE To $\overline{\text{PSEN}}$	58		ns	TCLCL-25		ns
TPLPH	$\overline{\text{PSEN}}$ Pulse Width	215		ns	3TCLCL-35		ns
TPLIV	$\overline{\text{PSEN}}$ To Valid Instr In		125	ns		3TCLCL-125	ns
TPXIX	Input Instr Hold After $\overline{\text{PSEN}}$	0		ns	0		ns
TPXIZ	Input Instr Float After $\overline{\text{PSEN}}$		63	ns		TCLCL-20	ns
TPXAV	Address Valid After $\overline{\text{PSEN}}$	75		ns	TCLCL-8		ns
TAVIV	Address To Valid Instr In		302	ns		5TCLCL-115	ns
TAZPL	Address Float To $\overline{\text{PSEN}}$	0		ns	0		ns

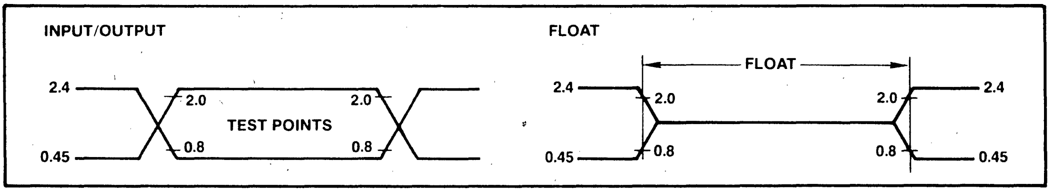
EXTERNAL DATA MEMORY CHARACTERISTICS

		12 MHz Clock			Variable Clock 1/TCLCL = 3.5 MHz to 12 MHz		
Symbol	**Parameter**	**Min**	**Max**	**Unit**	**Min**	**Max**	**Unit**
TRLRH	$\overline{\text{RD}}$ Pulse Width	400		ns	6TCLCL-100		ns
TWLWH	$\overline{\text{WR}}$ Pulse Width	400		ns	6TCLCL-100		ns
TLLAX	Address Hold After ALE	48		ns	TCLCL-35		
TRLDV	$\overline{\text{RD}}$ To Valid Data In		250	ns		5TCLCL-165	ns
TRHDX	Data Hold After $\overline{\text{RD}}$	0		ns	0		ns
TRHDZ	Data Float After $\overline{\text{RD}}$		97	ns		2TCLCL-70	ns
TLLDV	ALE To Valid Data In		517	ns		8TCLCL-150	ns
TAVDV	Address To Valid Data In		585	ns		9TCLCL-165	ns
TLLWL	ALE To $\overline{\text{WR}}$ or $\overline{\text{RD}}$	200	300	ns	3TCLCL-50	3TCLCL + 50	ns
TAVWL	Address To $\overline{\text{WR}}$ or $\overline{\text{RD}}$	203		ns	4TCLCL-130		ns
TWHLH	$\overline{\text{WR}}$ or $\overline{\text{RD}}$ High To ALE High	43	123	ns	TCLCL-40	TCLCL + 40	ns
TDVWX	Data Valid To $\overline{\text{WR}}$ Transition	23		ns	TCLCL-60		ns
TQVWH	Data Setup Before $\overline{\text{WR}}$	433		ns	7TCLCL-150		ns
TWHQX	Data Hold After $\overline{\text{WR}}$	33		ns	TCLCL-50		ns
TRLAZ	Address Float After $\overline{\text{RD}}$		0	ns		0	ns

AC TIMING DIAGRAMS

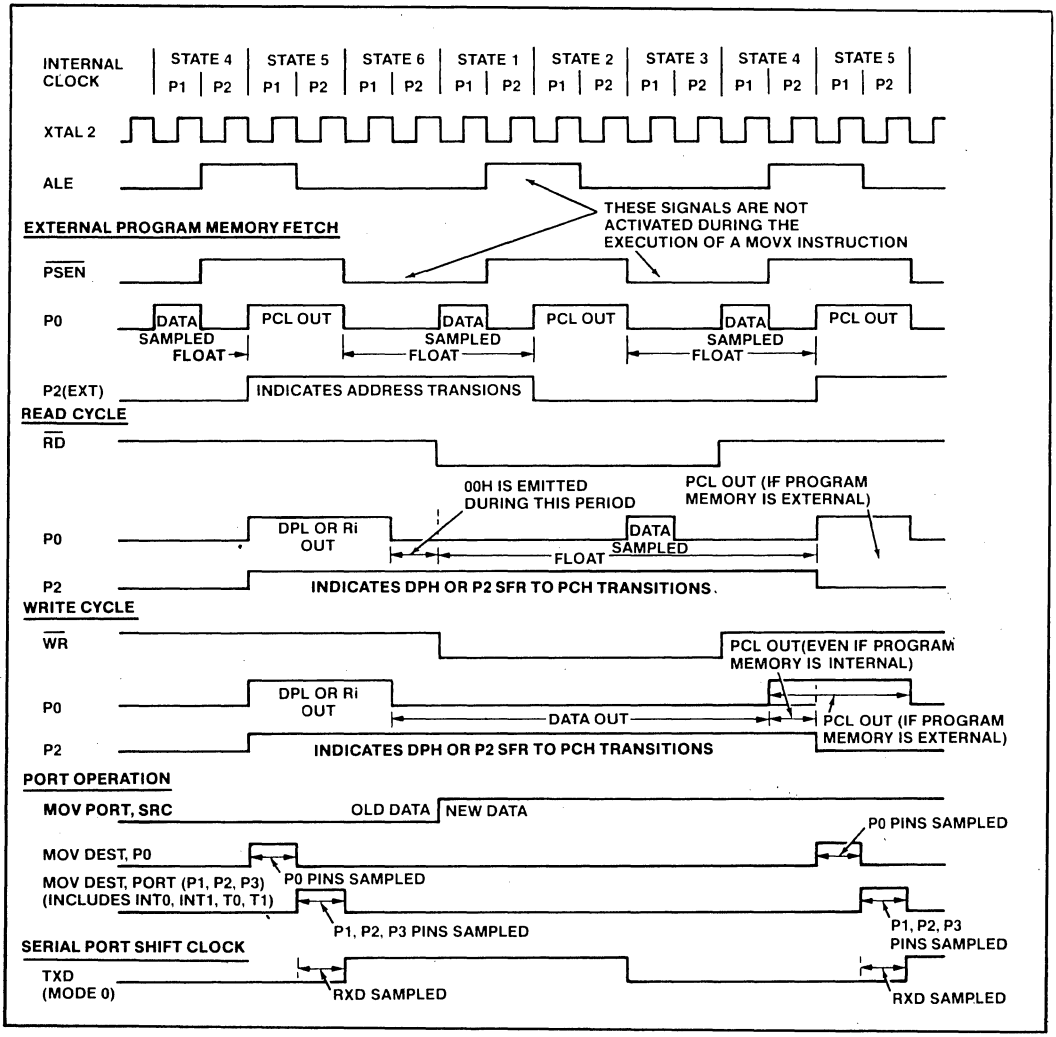


AC TESTING INPUT/OUTPUT, FLOAT WAVEFORMS



AC inputs during testing are driven at 2.4V for a logic "1" and 0.45V for a logic "0". Timing measurements are made at 2.0V for a logic "1" and 0.8V for a logic "0". For timing purposes, the float state is defined as the point at which a P0 pin sinks 2.4mA or sources 400 μA at the voltage test levels.

CLOCK WAVEFORMS



This diagram indicates when signals are clocked internally. The time it takes the signals to propagate to the pins, however, ranges from 25 to 125 ns. This propagation delay is dependent on variables such as temperature and pin loading. Propagation also varies from output to output and component to component. Typically though, (T_A = 25° C, fully loaded) RD and WR propagation delays are approximately 50 ns. The other signals are typically 85 ns. Propagation delays are incorporated in the AC specifications.

Table 1. MCS®-51 Instruction Set Description

ARITHMETIC OPERATIONS

Mnemonic		Description	Byte	Cyc
ADD	A,Rn	Add register to Accumulator	1	1
ADD	A,direct	Add direct byte to Accumulator	2	1
ADD	A,@Ri	Add indirect RAM to Accumulator	1	1
ADD	A,#data	Add immediate data to Accumulator	2	1
ADDC	A,Rn	Add register to Accumulator with Carry	1	1
ADDC	A,direct	Add direct byte to A with Carry flag	2	1
ADDC	A,@Ri	Add indirect RAM to A with Carry flag	1	1
ADDC	A,#data	Add immediate data to A with Carry flag	2	1
SUBB	A,Rn	Subtract register from A with Borrow	1	1
SUBB	A,direct	Subtract direct byte from A with Borrow	2	1
SUBB	A,@Ri	Subtract indirect RAM from A with Borrow	1	1
SUBB	A,#data	Subtract immed data from A with Borrow	2	1
INC	A	Increment Accumulator	1	1
INC	Rn	Increment register	1	1
INC	direct	Increment direct byte	2	1
INC	@Ri	Increment indirect RAM	1	1
INC	DPTR	Increment Data Pointer	1	2
DEC	A	Decrement Accumulator	1	1
DEC	Rn	Decrement register	1	1
DEC	direct	Decrement direct byte	2	1
DEC	@Ri	Decrement indirect RAM	1	1
MUL	AB	Multiply A & B	1	4
DIV	AB	Divide A by B	1	4
DA	A	Decimal Adjust Accumulator	1	1

LOGICAL OPERATIONS

Mnemonic		Destination	Byte	Cyc
ANL	A,Rn	AND register to Accumulator	1	1
ANL	A,direct	AND direct byte to Accumulator	2	1
ANL	A,@Ri	AND indirect RAM to Accumulator	1	1
ANL	A,#data	AND immediate data to Accumulator	2	1
ANL	direct,A	AND Accumulator to direct byte	2	1
ANL	direct,#data	AND immediate data to direct byte	3	2
ORL	A,Rn	OR register to Accumulator	1	1
ORL	A,direct	OR direct byte to Accumulator	2	1

LOGICAL OPERATIONS (CONTINUED)

Mnemonic		Destination	Byte	Cyc
ORL	A,@Ri	OR indirect RAM to Accumulator	1	1
ORL	A,#data	OR immediate data to Accumulator	2	1
ORL	direct,A	OR Accumulator to direct byte	2	1
ORL	direct,#data	OR immediate data to direct byte	3	2
XRL	A,Rn	Exclusive-OR register to Accumulator	1	1
XRL	A,direct	Exclusive-OR direct byte to Accumulator	2	1
XRL	A,@Ri	Exclusive-OR indirect RAM to A	1	1
XRL	A,#data	Exclusive-OR immediate data to A	2	1
XRL	direct,A	Exclusive-OR Accumulator to direct byte	2	1
XRL	direct,#data	Exclusive-OR immediate data to direct	3	2
CLR	A	Clear Accumulator	1	1
CPL	A	Complement Accumulator	1	1
RL	A	Rotate Accumulator Left	1	1
RLC	A	Rotate A Left through the Carry flag	1	1
RR	A	Rotate Accumulator Right	1	1
RRC	A	Rotate A Right through Carry flag	1	1
SWAP	A	Swap nibbles within the Accumulator	1	1

DATA TRANSFER

Mnemonic		Description	Byte	Cyc
MOV	A,Rn	Move register to Accumulator	1	1
MOV	A,direct	Move direct byte to Accumulator	2	1
MOV	A,@Ri	Move indirect RAM to Accumulator	1	1
MOV	A,#data	Mov immediate data to Accumulator	2	1
MOV	Rn,A	Move Accumulator to register	1	1
MOV	Rn,direct	Move direct byte to register	2	2
MOV	Rn,#data	Move immediate data to register	2	1
MOV	direct,A	Move Accumulator to direct byte	2	1
MOV	direct,Rn	Move register to direct byte	2	2
MOV	direct,direct	Move direct byte to direct	3	2
MOV	direct,@Ri	Move indirect RAM to direct byte	2	2

Table 1. (Cont.)

DATA TRANSFER (CONTINUED)

Mnemonic		Description	Byte	Cyc
MOV	direct,#data	Move immediate data to direct byte	3	2
MOV	@Ri,A	Move Accumulator to indirect RAM	1	1
MOV	@Ri,direct	Move direct byte to indirect RAM	2	2
MOV	@Ri,#data	Move immediate data to indirect RAM	2	1
MOV	DPTR,#data16	Load Data Pointer with a 16-bit constant	3	2
MOVC	A,@A+DPTR	Move Code byte relative to DPTR to A	1	2
MOVC	A,@A+PC	Move Code byte relative to PC to A	1	2
MOVX	A,@Ri	Move External RAM (8-bit addr) to A	1	2
MOVX	A,@DPTR	Move External RAM (16-bit addr) to A	1	2
MOVX	@Ri,A	Move A to External RAM (8-bit addr)	1	2
MOVX	@DPTR,A	Move A to External RAM (16-bit addr)	1	2
PUSH	direct	Push direct byte onto stack	2	2
POP	direct	Pop direct byte from stack	2	2
XCH	A,Rn	Exchange register with Accumulator	1	1
XCH	A,direct	Exchange direct byte with Accumulator	2	1
XCH	A,@Ri	Exchange indirect RAM with A	1	1
XCHD	A,@Ri	Exchange low-order Digit ind RAM w A	1	1

BOOLEAN VARIABLE MANIPULATION

Mnemonic		Description	Byte	Cyc
CLR	C	Clear Carry flag	1	1
CLR	bit	Clear direct bit	2	1
SETB	C	Set Carry flag	1	1
SETB	bit	Set direct Bit	2	1
CPL	C	Complement Carry flag	1	1
CPL	bit	Complement direct bit	2	1
ANL	C,bit	AND direct bit to Carry flag	2	2
ANL	C,1 bit	AND complement of direct bit to Carry	2	2
ORL	C/bit	OR direct bit to Carry flag	2	2
ORL	C,1 bit	OR complement of direct bit to Carry	2	2
MOV	C/bit	Move direct bit to Carry flag	2	1
MOV	bit,C	Move Carry flag to direct bit	2	2

PROGRAM AND MACHINE CONTROL

Mnemonic		Description	Byte	Cyc
ACALL	addr11	Absolute Subroutine Call	2	2
LCALL	addr16	Long Subroutine Call	3	2
RET		Return from subroutine	1	2
RETI		Return from interrupt	1	2
AJMP	addr11	Absolute Jump	2	2
LJMP	addr16	Long Jump	3	2
SJMP	rel	Short Jump (relative addr)	2	2
JMP	@A+DPTR	Jump indirect relative to the DPTR	1	2
JZ	rel	Jump if Accumulator is Zero	2	2
JNZ	rel	Jump if Accumulator is Not Zero	2	2
JC	rel	Jump if Carry flag is set	2	2
JNC	rel	Jump if No Carry flag	2	2
JB	bit,rel	Jump if direct Bit set	3	2
JNB	bit,rel	Jump if direct Bit Not set	3	2
JBC	bit,rel	Jump if direct Bit is set & Clear bit	3	2
CJNE	A,direct,rel	Compare direct to A & Jump if Not Equal	3	2
CJNE	A,#data,rel	Comp, immed, to A & Jump if Not Equal	3	2
CJNE	Rn,#data,rel	Comp, immed, to reg & Jump if Not Equal	3	2
CJNE	@Ri,#data,rel	Comp, immed, to ind, & Jump if Not Equal	3	2
DJNZ	Rn,rel	Decrement register & Jump if Not Zero	2	2
DJNZ	direct,rel	Decrement direct & Jump if Not Zero	3	2
NOP		No operation	1	1

Notes on data addressing modes:

Rn —Working register R0–R7
direct —128 internal RAM locations, any I/O port, control or status register
@Ri —Indirect internal RAM location addressed by register R0 or R1
#data —8-bit constant included in instruction
#data16 —16-bit constant included as bytes 2 & 3 of instruction
bit —128 software flags, any I/O pin, control or status bit

Notes on program addressing modes:

addr16 —Destination address for LCALL & LJMP may be anywhere within the 64-K program memory address space
Addr11 —Destination address for ACALL & AJMP will be within the same 2-K page of program memory as the first byte of the following instruction
rel —SJMP and all conditional jumps include an 8-bit offset byte, Range is +127–128 bytes relative to first byte of the following instruction

Table 2. Instruction Opcodes in Hexadecimal Order

Hex Code	Number of Bytes	Mnemonic	Operands
00	1	NOP	
01	2	AJMP	code addr
02	3	LJMP	code addr
03	1	RR	A
04	1	INC	A
05	2	INC	data addr
06	1	INC	@R0
07	1	INC	@R1
08	1	INC	R0
09	1	INC	R1
0A	1	INC	R2
0B	1	INC	R3
0C	1	INC	R4
0D	1	INC	R5
0E	1	INC	R6
0F	1	INC	R7
10	3	JBC	bit addr, code addr
11	2	ACALL	code addr
12	3	LCALL	code addr
13	1	RRC	A
14	1	DEC	A
15	2	DEC	data addr
16	1	DEC	@R0
17	1	DEC	@R1
18	1	DEC	R0
19	1	DEC	R1
1A	1	DEC	R2
1B	1	DEC	R3
1C	1	DEC	R4
1D	1	DEC	R5
1E	1	DEC	R6
1F	1	DEC	R7
20	3	JB	bit addr, code addr
21	2	AJMP	code addr
22	1	RET	
23	1	RL	A
24	2	ADD	A,#data
25	2	ADD	A,data addr
26	1	ADD	A,@R0
27	1	ADD	A,@R1
28	1	ADD	A,R0
29	1	ADD	A,R1
2A	1	ADD	A,R2
2B	1	ADD	A,R3
2C	1	ADD	A,R4
2D	1	ADD	A,R5
2E	1	ADD	A,R6
2F	1	ADD	A,R7
30	3	JNB	bit addr, code addr
31	2	ACALL	code addr
32	1	RETI	
33	1	RLC	A
34	2	ADDC	A,#data
35	2	ADDC	A,data addr
36	1	ADDC	A,@R0
37	1	ADDC	A,@R1
38	1	ADDC	A,R0
39	1	ADDC	A,R1
3A	1	ADDC	A,R2
3B	1	ADDC	A,R3
3C	1	ADDC	A,R4
3D	1	ADDC	A,R5
3E	1	ADDC	A,R6
3F	1	ADDC	A,R7
40	2	JC	code addr
41	2	AJMP	code addr
42	2	ORL	data addr,A
43	3	ORL	data addr,#data
44	2	ORL	A,#data
45	2	ORL	A,data addr
46	1	ORL	A,@R0
47	1	ORL	A,@R1
48	1	ORL	A,R0
49	1	ORL	A,R1
4A	1	ORL	A,R2
4B	1	ORL	A,R3
4C	1	ORL	A,R4
4D	1	ORL	A,R5
4E	1	ORL	A,R6
4F	1	ORL	A,R7
50	2	JNC	code addr
51	2	ACALL	code addr
52	2	ANL	data addr,A
53	3	ANL	data addr,#data
54	2	ANL	A,#data
55	2	ANL	A,data addr
56	1	ANL	A,@R0
57	1	ANL	A@R1
58	1	ANL	A,R0
59	1	ANL	A,R1
5A	1	ANL	A,R2
5B	1	ANL	A,R3
5C	1	ANL	A,R4
5D	1	ANL	A,R5
5E	1	ANL	A,R6
5F	1	ANL	A,R7
60	2	JZ	code addr
61	2	AJMP	code addr
62	2	XRL	data addr.A
63	3	XRL	data addr,#data
64	2	XRL	A,#data
65	2	XRL	A,data addr

Table 2. (Cont.)

Hex Code	Number of Bytes	Mnemonic	Operands
66	1	XRL	A,@R0
67	1	XRL	A,@R1
68	1	XRL	A,R0
69	1	XRL	A,R1
6A	1	XRL	A,R2
6B	1	XRL	A,R3
6C	1	XRL	A,R4
6D	1	XRL	A,R5
6E	1	XRL	A,R6
6F	1	XRL	A,R7
70	2	JNZ	code addr
71	2	ACALL	code addr
72	2	ORL	C,bit addr
73	1	JMP	@A+DPTR
74	2	MOV	A,#data
75	3	MOV	data addr,#data
76	2	MOV	@R0,#data
77	2	MOV	@R1,#data
78	2	MOV	R0,#data
79	2	MOV	R1,#data
7A	2	MOV	R2,#data
7B	2	MOV	R3,#data
7C	2	MOV	R4,#data
7D	2	MOV	R5,#data
7E	2	MOV	R6,#data
7F	2	MOV	R7,#data
80	2	SJMP	code addr
81	2	AJMP	code addr
82	2	ANL	C,bit addr
83	1	MOVC	A,@A+PC
84	1	DIV	AB
85	3	MOV	data addr, data addr
86	2	MOV	data addr,@R0
87	2	MOV	data addr,@R1
88	2	MOV	data addr,R0
89	2	MOV	data addr,R1
8A	2	MOV	data addr,R2
8B	2	MOV	data addr,R3
8C	2	MOV	data addr,R4
8D	2	MOV	data addr,R5
8E	2	MOV	data addr,R6
8F	2	MOV	data addr,R7
90	3	MOV	DPTR,#data
91	2	ACALL	code addr
92	2	MOV	bit addr,C
93	1	MOVC	A,@A+DPTR
94	2	SUBB	A,#data
95	2	SUBB	A,data addr
96	1	SUBB	A,@R0
97	1	SUBB	A,@R1
98	1	SUBB	A,R0

Hex Code	Number of Bytes	Mnemonic	Operands
99	1	SUBB	A,R1
9A	1	SUBB	A,R2
9B	1	SUBB	A,R3
9C	1	SUBB	A,R4
9D	1	SUBB	A,R5
9E	1	SUBB	A,R6
9F	1	SUBB	A,R7
A0	2	ORL	C,/bit addr
A1	2	AJMP	code addr
A2	2	MOV	C,bit addr
A3	1	INC	DPTR
A4	1	MUL	AB
A5		reserved	
A6	2	MOV	@R0, data addr
A7	2	MOV	@R1,data addr
A8	2	MOV	R0,data addr
A9	2	MOV	R1,data addr
AA	2	MOV	R2,data addr
AB	2	MOV	R3,data addr
AC	2	MOV	R4,data addr
AD	2	MOV	R5,data addr
AE	2	MOV	R6,data addr
AF	2	MOV	R7,data addr
B0	2	ANL	C,/bit addr
B1	2	ACALL	code addr
B2	2	CPL	bit addr
B3	1	CPL	C
B4	3	CJNE	A,#data,code addr
B5	3	CJNE	A,data addr,code addr
B6	3	CJNE	@R0,#data,code addr
B7	3	CJNE	@R1,#data,code addr
B8	3	CJNE	R0,#data,code addr
B9	3	CJNE	R1,#data,code addr
BA	3	CJNE	R2,#data,code addr
BB	3	CJNE	R3,#data,code addr
BC	3	CJNE	R4,#data,code addr
BD	3	CJNE	R5,#data,code addr
BE	3	CJNE	R6,#data,code addr
BF	3	CJNE	R7,#data,code addr
C0	2	PUSH	data addr
C1	2	AJMP	code addr
C2	2	CLR	bit addr
C3	1	CLR	C
C4	1	SWAP	A
C5	2	XCH	A,data addr
C6	1	XCH	A,@R0
C7	1	XCH	A,@R1
C8	1	XCH	A,R0
C9	1	XCH	A,R1
CA	1	XCH	A,R2
CB	1	XCH	A,R3

Table 2. (Cont.)

Hex Code	Number of Bytes	Mnemonic	Operands
CC	1	XCH	A,R4
CD	1	XCH	A,R5
CE	1	XCH	A,R6
CF	1	XCH	A,R7
D0	2	POP	data addr
D1	2	ACALL	code addr
D2	2	SETB	bit addr
D3	1	SETB	C
D4	1	DA	A
D5	3	DJNZ	data addr,code addr
D6	1	XCHD	A,@R0
D7	1	XCHD	A,@R1
D8	2	DJNZ	R0,code addr
D9	2	DJNZ	R1,code addr
DA	2	DJNZ	R2,code addr
DB	2	DJNZ	R3,code addr
DC	2	DJNZ	R4,code addr
DD	2	DJNZ	R5,code addr
DE	2	DJNZ	R6,code addr
DF	2	DJNZ	R7,code addr
E0	1	MOVX	A,@DPTR
E1	2	AJMP	code addr
E2	1	MOVX	A,@R0
E3	1	MOVX	A,@R1
E4	1	CLR	A
E5	2	MOV	A,data addr
E6	1	MOV	A,@R0
E7	1	MOV	A,@R1
E8	1	MOV	A,R0
E9	1	MOV	A,R1
EA	1	MOV	A,R2
EB	1	MOV	A,R3
EC	1	MOV	A,R4
ED	1	MOV	A,R5
EE	1	MOV	A,R6
EF	1	MOV	A,R7
F0	1	MOVX	@DPTR,A
F1	2	ACALL	code addr
F2	1	MOVX	@R0,A
F3	1	MOVX	@R1,A
F4	1	CPL	A
F5	2	MOV	data addr,A
F6	1	MOV	@R0,A
F7	1	MOV	@R1,A
F8	1	MOV	R0,A
F9	1	MOV	R1,A
FA	1	MOV	R2,A
FB	1	MOV	R3,A
FC	1	MOV	R4,A
FD	1	MOV	R5,A
FE	1	MOV	R6,A
FF	1	MOV	R7,A

intel®

ADVANCED INFORMATION

8751H
8751H-11/8751H-10/8751H-8
SINGLE-COMPONENT 8-BIT MICROCOMPUTERS

- **8751H: 12 MHz Operation**
- **8751H-8: 8 MHz Operation**

- **Program Memory Security**
- **4K × 8 EPROM**
- **128 × 8 RAM**
- **32 I/O Lines (Four 8-Bit Ports)**
- **Two 16-Bit Timer/Counters**
- **Programmable Full-Duplex Serial Channel**
- **128K Accessible External Memory**
- **Boolean Processor**
- **4 μs Multiply and Divide**
- **218 User Bit-Addressable Locations**

The 8751H is a pin compatible EPROM version of the 8051AH. Intel's advanced +5 volt, depletion load, N-channel, HMOS technology allows the 8751H to remain fully compatible with its 8751/8751-8 predecessor in addition to incorporating a new program memory security feature. This allows the 8751H to be a full-speed MCS®-51 prototyping tool and provides for an effective single component solution for highly sensitive controller applications requiring code modification flexibility.

Specifically, the 8751H features: 4K byte program memory space; 32 I/O lines; two 16-bit timer/event counters; a 5-source; 2-level interrupt structure; a full duplex serial channel; a Boolean processor; and on-chip oscillator and clock circuitry. Standard TTL and most byte-oriented MCS-80 and MCS-85 peripherals can be used for I/O and memory expansion.

The 8751H is available in a hermetically sealed, ceramic, 40-lead dual in-line package which includes a window that allows for EPROM erasure when exposed to ultraviolet light (see Erasure Characteristics). During normal operation, ambient light may adversely affect the functionality of the chip. Therefore, applications which expose the 8751H to ambient light may require an opaque label over the window.

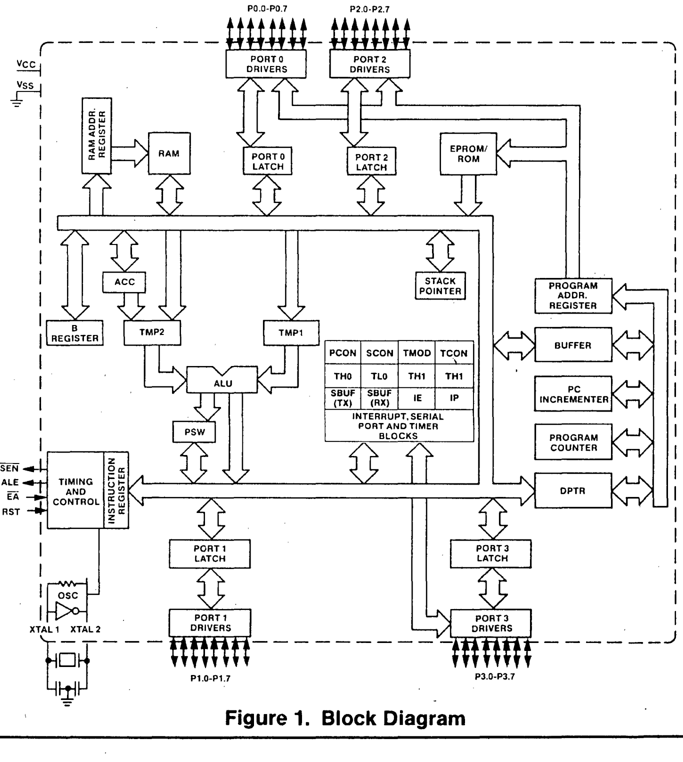


Figure 1. Block Diagram

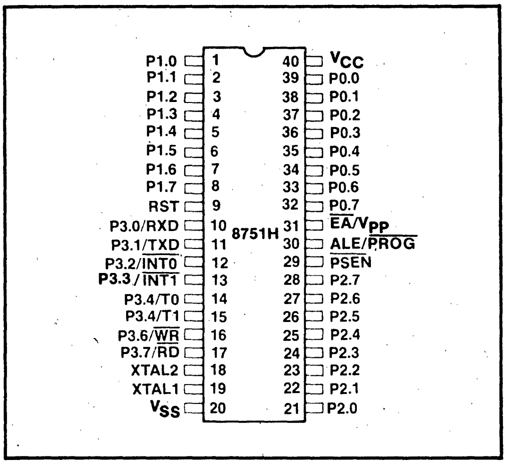


Figure 2. Pin Configuration

8751H PIN DESCRIPTIONS

V_{SS}

Circuit ground potential.

V_{CC}

Supply voltage during programming, verification, and normal operation.

Port 0

Port 0 is an 8-bit open drain bidirectional I/O port. It is also the multiplexed low-order address and data bus during accesses to external memory. It also receives the instruction bytes during EPROM programming, and outputs instruction bytes during program verification. (External pullups are required during program verification.) Port 0 can sink (and in bus operations can source) eight LS TTL inputs.

Port 1

Port 1 is an 8-bit bidirectional I/O port with internal pullups. It receives the low-order address byte during EPROM programming and program verification. Port 1 can sink/source four LS TTL inputs.

Port 2

Port 2 is an 8-bit bidirectional I/O port with internal pullups. It emits the high-order address byte during accesses to external memory. It also receives the high-order address bits during EPROM programming and program verification. Port 2 can sink/source four LS TTL inputs.

Port 3

Port 3 is an 8-bit bidirectional I/O port with internal pullups. It also serves the functions of various special features of the MCS®-51 Family, as listed below:

Port Pin	Alternate Function
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	$\overline{\text{INT0}}$ (external interrupt)
P3.3	$\overline{\text{INT1}}$ (external interrupt)
P3.4	T0 (Timer/counter 0 external input)
P3.5	T1 (Timer/counter 1 external input)
P3.6	$\overline{\text{WR}}$ (external Data Memory write strobe)
P3.7	$\overline{\text{RD}}$ (external Data Memory read strobe)

Port 3 can sink/source four LS TTL inputs.

RST

A high on this pin for two machine cycles while the oscillator is running resets the device. A small external pulldown resistor (≈8.2kΩ) from RST to V_{SS} permits power-on reset when a capacitor (≈10 µf) is also connected from this pin to V_{CC}.

ALE/$\overline{\text{PROG}}$

Address Latch Enable output for latching the low byte of the address during accesses to external memory. ALE is activated at a constant rate of 1/6 the oscillator frequency except during an external data memory access at which time one ALE pulse is skipped. ALE can sink/source 8 LS TTL inputs. This pin is also the program pulse input ($\overline{\text{PROG}}$) during EPROM programming.

$\overline{\text{PSEN}}$

Program Store Enable output is the read strobe to external Program Memory. $\overline{\text{PSEN}}$ is activated twice each machine cycle during fetches from external Program Memory. (However, even when executing out of external Program Memory two activations of $\overline{\text{PSEN}}$ are skipped during each access to external

Data Memory.) $\overline{\text{PSEN}}$ is not activated during fetches from internal Program Memory. $\overline{\text{PSEN}}$ can sink/source 8 LS TTL inputs.

$\overline{\text{EA}}$/VPP

When $\overline{\text{EA}}$ is held high, the 8751H executes out of internal Program Memory (unless the Program Counter exceeds 0FFFH). When $\overline{\text{EA}}$ is held low, the 8751H executes only out of external Program Memory. This pin also receives the 21V programming supply voltage (VPP) during EPROM programming. This pin should not be floated during normal operation.

XTAL1

Input to the inverting amplifier that forms the oscillator. XTAL1 should be grounded when an external oscillator is used.

XTAL2

Output of the inverting amplifier that forms the oscillator, and input to the internal clock generator. Receives the external oscillator signal when an external oscillator is used.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias	0°C to 70°C
Storage Temperature	-65°C to +150°C
Voltage On Any Pin to V_{SS} (Except V_{PP})	-0.5V to +7V
Voltage from V_{PP} to V_{SS}	21.5V
Power Dissipation	2 W

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS: (T_A = 0°C to 70°C; V_{CC} = 4.5V to 5.5V, V_{SS} = 0V)

Symbol	Parameter	Min	Max	Unit	Test Conditions
VIL	Input Low Voltage	-0.5	0.8	V	
VIH	Input High Voltage (Except XTAL2, RST)	2.0	V_{CC} + 0.5	V	
VIH1	Input High Voltage to XTAL2, RST	2.5	V_{CC} + 0.5	V	XTAL1 = V_{SS}
VOL	Output Low Voltage Ports 1, 2, 3 (Note 1)		0.45	V	IOL = 1.6mA
VOL1	Output Low Voltage Port 0, ALE, $\overline{\text{PSEN}}$ (Note 1)		0.60 0.45	V V	IOL = 3.2mA IOL = 2.4mA
VOH	Output High Voltage Ports 1, 2, 3	2.4		V	IOH = -80μA
VOH1	Output High Voltage Port 0 (in External Bus Mode), ALE, $\overline{\text{PSEN}}$	2.4		V	IOH = -400μA
IIL	Logical 0 Input Current P1, P2, P3		500	μA	Vin = 0.45V
IIL1	Logical 0 Input Current to $\overline{\text{EA}}$/V_{PP}		-15	mA	Vin = 0.45V
IIL2	Logical 0 Input Current to XTAL2		-2.5	mA	XTAL1 = V_{SS} Vin = 0.45V
ILI	Input Leakage Current to Port 0		100	μA	0.45 < Vin < V_{CC}
IIH	Logical Input Current to $\overline{\text{EA}}$/V_{PP}		500	μA	Vin = 2.4V
IIH1	Input Current to RST/V_{PD} to activate reset		500	μA	Vin < (V_{CC} - 1.5V)
ICC	Power Supply Current		250	mA	All outputs disconnected, $\overline{\text{EA}}$ = V_{CC}
CIO	Capacitance of I/O Buffers		10	pF	fc = 1 MHz T_A = 25°C

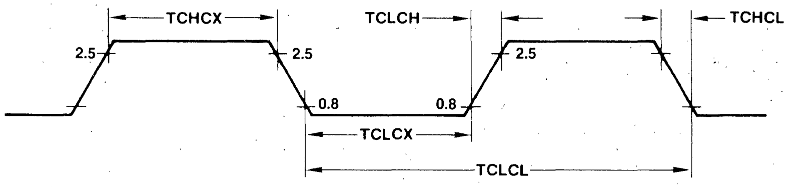
See page 4 for Notes.

NOTE: VOL is degraded when the 8751H rapidly discharges external capacitance. This AC noise is most pronounced during emission of address data. When using external memory, locate the latch or buffer as close to the 8751H as possible.

Datum	Emitting Ports	Degraded I/O Lines	VOL (peak) (max)
Address	P2, P0	P1, P3	0.8V
Write Data	P0	P1, P3, ALE	0.8V

EXTERNAL CLOCK DRIVE CHARACTERISTICS (XTAL2)

Symbol	Parameter	Min	Max	Units
TCLCL	Oscillator Period: 8751H	83.3	286	ns
TCHCX	High Time	20		ns
TCLCX	Low Time	20		ns
TCLCH	Rise Time		20	ns
TCHCL	Fall Time		20	ns



AC CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 4.5V to 5.5V, V_{SS} = 0V, Load Capacitance for Port 0, ALE, and $\overline{\text{PSEN}}$ = 100 pF; Load Capacitance for all other outputs = 80 pF)

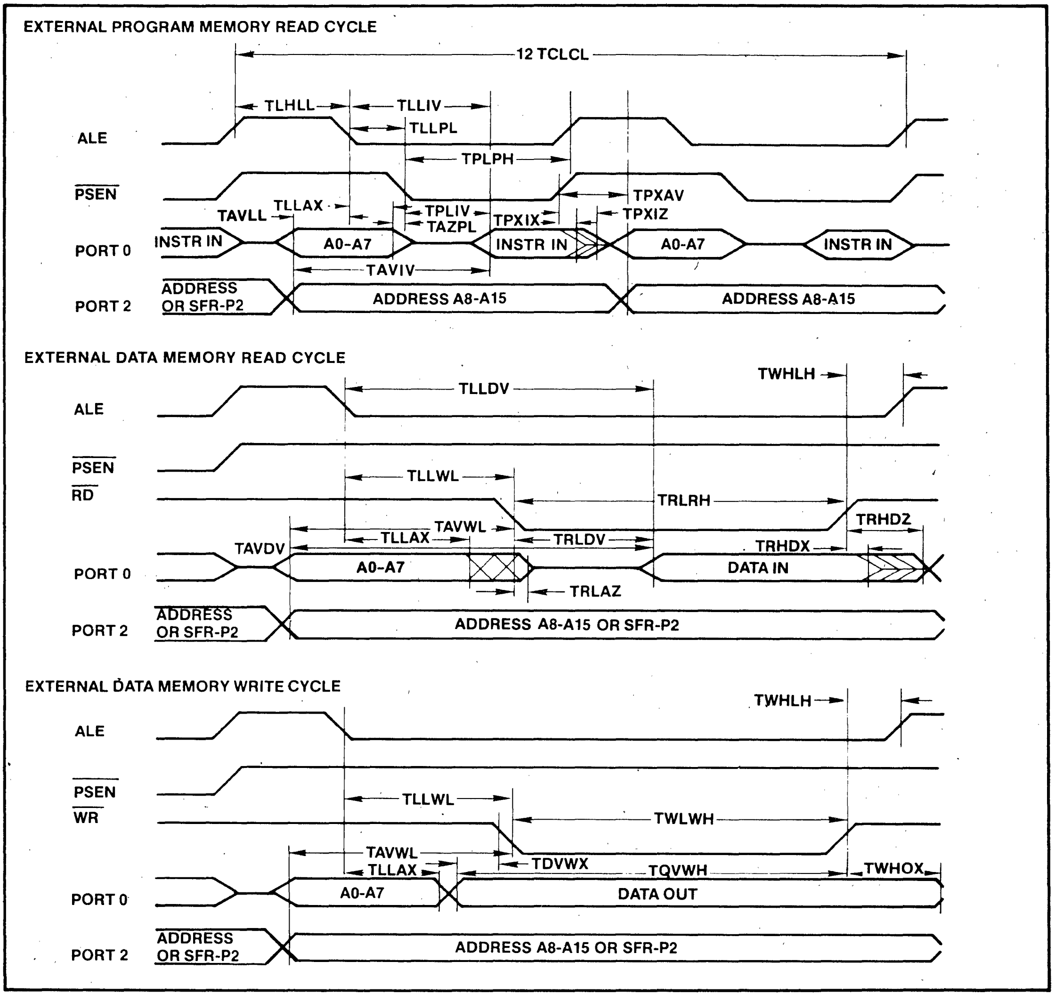
EXTERNAL PROGRAM MEMORY CHARACTERISTICS

Symbol	Parameter	8751H 12 MHz Osc		Variable Oscillator		Units
		Min	Max	Min	Max	
TCLCL	Oscillator Period			83.3	833.3	ns
TLHLL	ALE Pulse Width	127		2TCLCL-40		ns
TAVLL	Address Valid to ALE	53		TCLCL-30		ns
TLLAX	Address Hold after ALE	48		TCLCL-35		ns
TLLIV	ALE to Valid Instr In		183		4TCLCL-150	ns
TLLPL	ALE to $\overline{\text{PSEN}}$	58		TCLCL-25		ns
TPLPH	$\overline{\text{PSEN}}$ Pulse Width	190		3TCLCL-60		ns
TPLIV	$\overline{\text{PSEN}}$ to Valid Instr In		100		3TCLCL-150	ns
TPXIX	Input Instr Hold after $\overline{\text{PSEN}}$	0		0		ns
TPXIZ	Input Instr Float after $\overline{\text{PSEN}}$		63		TCLCL-20	ns
TPXAV	$\overline{\text{PSEN}}$ to Address Valid	75		TCLCL-8		ns
TAVIV	Address to Valid Instr In		267		5TCLCL-150	ns
TAZPL	Address Float to $\overline{\text{PSEN}}$	0		0		ns

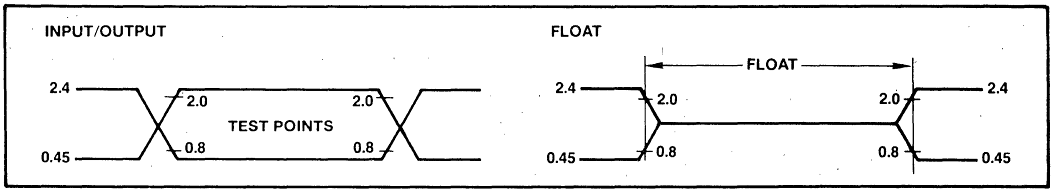
EXTERNAL DATA MEMORY CHARACTERISTICS

Symbol	Parameter	8751H 12 MHz Osc		Variable Oscillator		Units
		Min	Max	Min	Max	
TRLRH	$\overline{\text{RD}}$ Pulse Width	400		6TCLCL-100		ns
TWLWH	$\overline{\text{WR}}$ Pulse Width	400		6TCLCL-100		ns
TLLAX	Address Hold After ALE	48		TCLCL-35		ns
TRLDV	$\overline{\text{RD}}$ to Valid Data In		252		5TCLCL-165	ns
TRHDX	Data Hold after $\overline{\text{RD}}$	0		0		ns
TRHDZ	Data Float after $\overline{\text{RD}}$		97		2TCLCL-70	ns
TLLDV	ALE to Valid Data In		517		8TCLCL-150	ns
TAVDV	Address to Valid Data In		585		9TCLCL-165	ns
TLLWL	ALE to $\overline{\text{WR}}$ or $\overline{\text{RD}}$	200	300	3TCLCL-50	3TCLCL+50	ns
TAVWL	Address to $\overline{\text{WR}}$ or $\overline{\text{RD}}$	203		4TCLCL-130		ns
TQVWX	Data Valid to $\overline{\text{WR}}$ Transition	13		TCLCL-70		ns
TQVWH	Data Setup to $\overline{\text{WR}}$ High	433		7TCLCL-150		ns
TWHQX	Data Held after $\overline{\text{WR}}$	33		TCLCL-50		ns
TRLAZ	$\overline{\text{RD}}$ Low to Address Float		0		0	ns
TWHLH	$\overline{\text{RD}}$ or WR High to ALE High	33	133	TCLCL-50	TCLCL+50	ns

AC TIMING DIAGRAMS

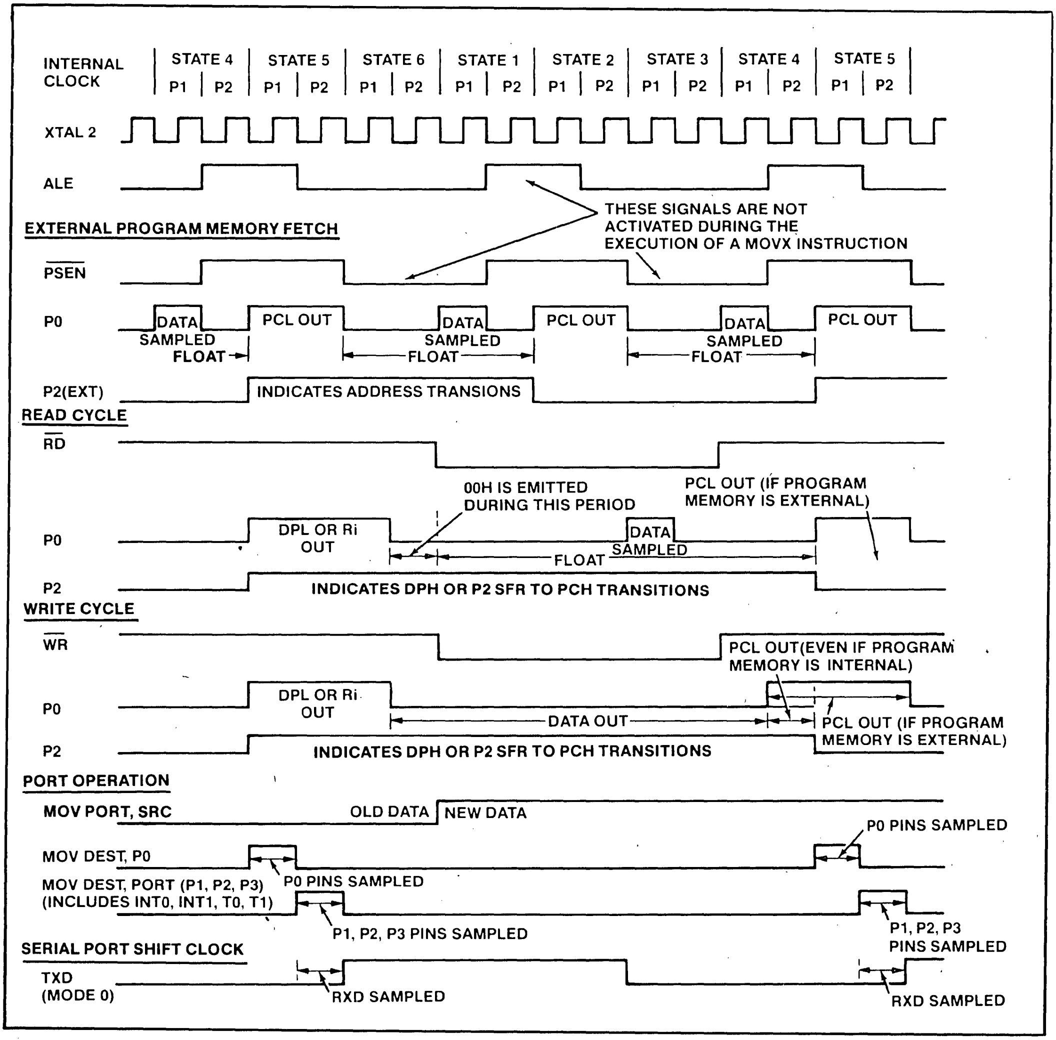


AC TESTING INPUT/OUTPUT, FLOAT WAVEFORMS



AC inputs during testing are driven at 2.4V for a logic "1" and 0.45V for a logic "0". Timing measurements are made at 2.0V for a logic "1" and 0.8V for a logic "0". For timing purposes, the float state is defined as the point at which a P0 pin sinks 2.4mA or sources 400 μA at the voltage test levels.

CLOCK WAVEFORMS


INTERNAL CLOCK
STATE 4 STATE 5 STATE 6 STATE 1 STATE 2 STATE 3 STATE 4 STATE 5
P1 P2
XTAL 2
ALE
THESE SIGNALS ARE NOT ACTIVATED DURING THE EXECUTION OF A MOVX INSTRUCTION
EXTERNAL PROGRAM MEMORY FETCH
PSEN
P0
DATA SAMPLED
PCL OUT
FLOAT
P2(EXT)
INDICATES ADDRESS TRANSIONS
READ CYCLE
RD
00H IS EMITTED DURING THIS PERIOD
PCL OUT (IF PROGRAM MEMORY IS EXTERNAL)
P0
DPL OR Ri OUT
DATA SAMPLED
FLOAT
P2
INDICATES DPH OR P2 SFR TO PCH TRANSITIONS
WRITE CYCLE
WR
PCL OUT(EVEN IF PROGRAM MEMORY IS INTERNAL)
P0
DPL OR Ri OUT
DATA OUT
PCL OUT (IF PROGRAM MEMORY IS EXTERNAL)
P2
INDICATES DPH OR P2 SFR TO PCH TRANSITIONS
PORT OPERATION
MOV PORT, SRC
OLD DATA
NEW DATA
MOV DEST, P0
P0 PINS SAMPLED
MOV DEST, PORT (P1, P2, P3) (INCLUDES INT0, INT1, T0, T1)
P0 PINS SAMPLED
P1, P2, P3 PINS SAMPLED
P1, P2, P3 PINS SAMPLED
SERIAL PORT SHIFT CLOCK
TXD (MODE 0)
RXD SAMPLED
RXD SAMPLED

This diagram indicates when signals are clocked internally. The time it takes the signals to propagate to the pins, however, ranges from 25 to 125 ns. This propagation delay is dependent on variables such as temperature and pin loading. Propagation also varies from output to output and component to component. Typically though, (T_A = 25° C, fully loaded) RD and WR propagation delays are approximately 50 ns. The other signals are typically 85 ns. Propagation delays are incorporated in the AC specifications.

8751H EPROM CHARACTERISTICS

Erasure Characteristics

Erasure of the 8751H Program Memory begins to occur when the chip is exposed to light with wavelengths shorter than approximately 4,000 Ångstroms. Since sunlight and fluorescent lighting have wavelengths in this range, constant exposure to these light sources over an extended period of time (about 1 week in sunlight, or 3 years in room-level fluorescent lighting) could cause unintentional erasure. If an application subjects the 8751H to this type of exposure, it is suggested that an opaque label be placed over the window.

The recommended erasure procedure is exposure to ultraviolet light (at 2537 Ångstroms) to an integrated dose of at least 15 W-sec. Exposing the 8751H to an ultraviolet lamp of 12,000 μW/cm^2 rating for 20 to 30 minutes, at a distance of about 1 inch, should be sufficient.

Erasure leaves the array in an all 1s state.

Programming the EPROM

To be programmed, the 8751H must be running with a 4 to 6 MHz oscillator. (The reason the oscillator needs to be running is that the internal bus is being used to transfer address and program data to appropriate registers.) The address of an EPROM location to be programmed is applied to Port 1 and pins P2.0-P2.3 of Port 2, while the data byte is applied to Port 0. Pins P2.4-P2.6 and $\overline{\text{PSEN}}$ should be held low, and P2.7 and RST high. (These are all TTL levels except RST, which requires 2.5V for high.) $\overline{\text{EA}}$/VPP is held normally high, and is pulsed to +21V. While $\overline{\text{EA}}$/VPP is at 21V, the ALE/$\overline{\text{PROG}}$ pin, which is normally being held high, is pulsed low for 50 msec. Then $\overline{\text{EA}}$/VPP is returned to high. This is illustrated in Figure 3. detailed timing specifications are provided in the EPROM Programming and Verification Characteristics section of this data sheet.

Program Memory Security

The program memory security feature is developed around a "security bit" in the 8751H EPROM array. Once this "hidden bit" is programmed, electrical access to the contents of the entire program memory array becomes impossible. Activation of this feature is accomplished by programming the 8751H as described in "Programming the EPROM" with the exception that P2.6 is held at a TTL high rather than a TTL low. In addition, Port 1 and P2.0-P2.3 may be in any state. Figure 4 illustrates the security bit programming configuration. Deactivating the security feature, which again allows programmability of the EPROM, is accomplished by exposing the EPROM to ultraviolet light. This exposure, as described in "Erasure Characteristics," erases the entire EPROM array. Therefore, attempted retrieval of "protected code" results in its destruction.

Program Verification

Program Memory may be read only when the "security feature" has not been activated. Refer to Figure 5 for Program Verification setup. To read the Program Memory, the following procedure can be used. The unit must be running with a 4 to 6 MHz oscillator. The address of a Program Memory location to be read is applied to Port 1 and pins P2.0-P2.3 of Port 2. Pins P2.4-P2.6 and $\overline{\text{PSEN}}$ are held at TTL low, while the ALE/$\overline{\text{PROG}}$, RST, and $\overline{\text{EA}}$/VPP pins are held at TTL high. (These are all TTL levels except RST, which requires 2.5V for high.) Port 0 will be the data output lines. P2.7 can be used as a read strobe. While P2.7 is held high, the Port 0 pins float. When P2.7 is strobed low, the contents of the addressed location will appear at Port 0. External pullups (e.g., 10K) are required on Port 0 during program verification.

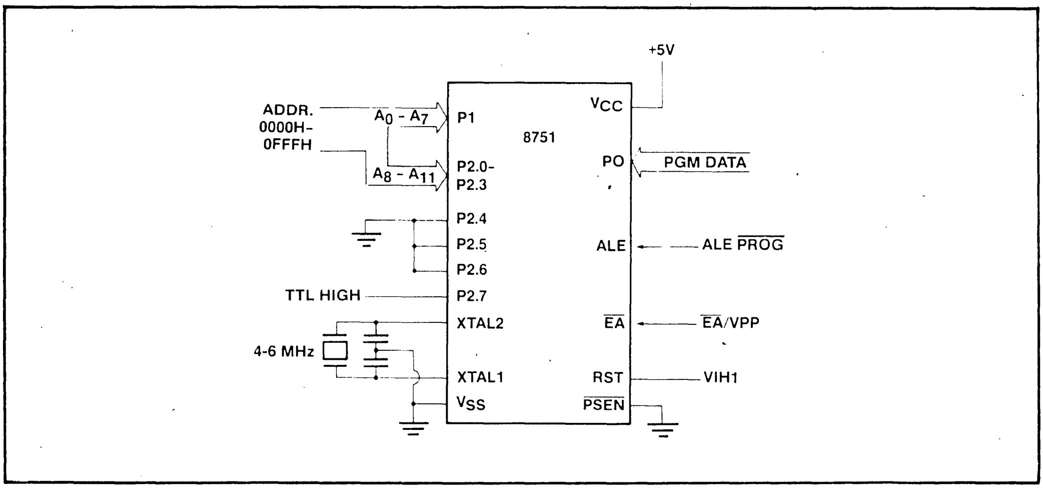


Figure 3. Programming Configuration

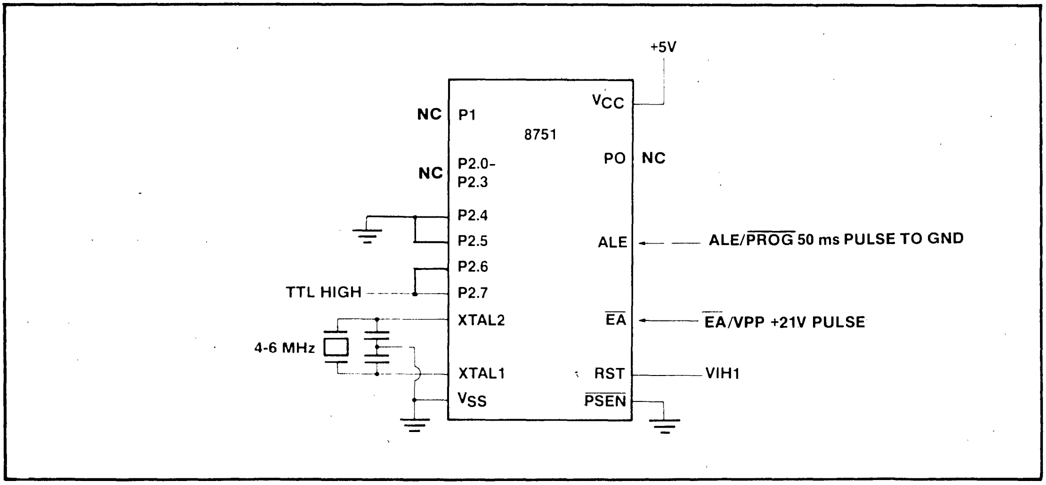


Figure 4. Security Bit Programming Configuration

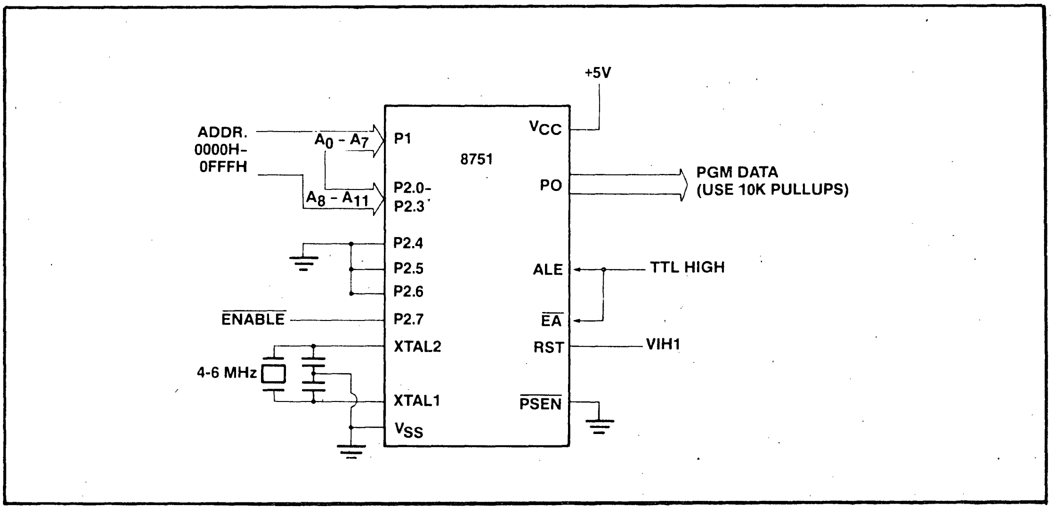
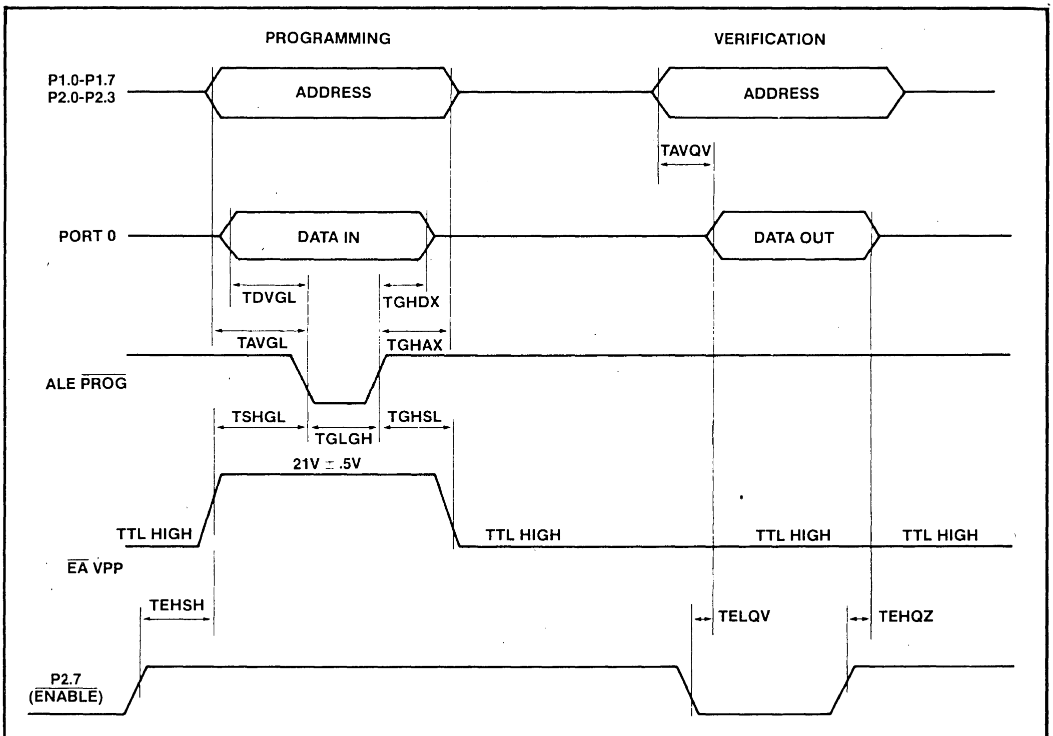


Figure 5. Program Verification Configuration

EPROM PROGRAMMING, SECURITY BIT PROGRAMMING AND VERIFICATION CHARACTERISTICS
(TA = 21°C to 27°C, V_{CC} = 4.5V to 5.5V, V_{SS} = 0V)

Symbol	Parameter	Min	Max	Units
V_{PP}	Programming Supply Voltage	20.5	21.5	V
IPP	Programming Current		30	mA
1/TCLCL	Oscillator Frequency	4	6	MHz
TAVGL	Address Setup to $\overline{\text{PROG}}$	48TCLCL		
TGHAX	Address Hold after $\overline{\text{PROG}}$	48TCLCL		
TDVGL	Data Setup to $\overline{\text{PROG}}$	48TCLCL		
TGHDX	Data Hold after $\overline{\text{PROG}}$	48TCLCL		
TEHSH	$\overline{\text{ENABLE}}$ High to V_{PP}	48TCLCL		
TSHGL	V_{PP} Setup to $\overline{\text{PROG}}$	10		μsec
TGHSL	V_{PP} Hold after $\overline{\text{PROG}}$	10		μsec
TGLGH	$\overline{\text{PROG}}$ Width	45	55	msec
TAVQV	Address to Data Valid		48TCLCL	
TELQV	$\overline{\text{ENABLE}}$ to Data Valid		48TCLCL	
TEHQZ	Data Float after $\overline{\text{ENABLE}}$	0	48TCLCL	

EPROM PROGRAMMING, SECURITY BIT PROGRAMMING AND VERIFICATION WAVEFORMS



FOR PROGRAMMING CONDITIONS SEE FIGURE 3.
FOR SECURITY BIT PROGRAMMING CONDITIONS SEE FIGURE 4.
FOR VERIFICATION CONDITIONS SEE FIGURE 5.

Table 1. MCS®-51 Instruction Set Description

ARITHMETIC OPERATIONS

Mnemonic		Description	Byte	Cyc
ADD	A,Rn	Add register to Accumulator	1	1
ADD	A,direct	Add direct byte to Accumulator	2	1
ADD	A,@Ri	Add indirect RAM to Accumulator	1	1
ADD	A,#data	Add immediate data to Accumulator	2	1
ADDC	A,Rn	Add register to Accumulator with Carry	1	1
ADDC	A,direct	Add direct byte to A with Carry flag	2	1
ADDC	A,@Ri	Add indirect RAM to A with Carry flag	1	1
ADDC	A,#data	Add immediate data to A with Carry flag	2	1
SUBB	A,Rn	Subtract register from A with Borrow	1	1
SUBB	A,direct	Subtract direct byte from A with Borrow	2	1
SUBB	A,@Ri	Subtract indirect RAM from A with Borrow	1	1
SUBB	A,#data	Subtract immed data from A with Borrow	2	1
INC	A	Increment Accumulator	1	1
INC	Rn	Increment register	1	1
INC	direct	Increment direct byte	2	1
INC	@Ri	Increment indirect RAM	1	1
INC	DPTR	Increment Data Pointer	1	2
DEC	A	Decrement Accumulator	1	1
DEC	Rn	Decrement register	1	1
DEC	direct	Decrement direct byte	2	1
DEC	@Ri	Decrement indirect RAM	1	1
MUL	AB	Multiply A & B	1	4
DIV	AB	Divide A by B	1	4
DA	A	Decimal Adjust Accumulator	1	1

LOGICAL OPERATIONS

Mnemonic		Destination	Byte	Cyc
ANL	A,Rn	AND register to Accumulator	1	1
ANL	A,direct	AND direct byte to Accumulator	2	1
ANL	A,@Ri	AND indirect RAM to Accumulator	1	1
ANL	A,#data	AND immediate data to Accumulator	2	1
ANL	direct,A	AND Accumulator to direct byte	2	1
ANL	direct,#data	AND immediate data to direct byte	3	2
ORL	A,Rn	OR register to Accumulator	1	1
ORL	A,direct	OR direct byte to Accumulator	2	1

LOGICAL OPERATIONS (CONTINUED)

Mnemonic		Destination	Byte	Cyc
ORL	A,@Ri	OR indirect RAM to Accumulator	1	1
ORL	A,#data	OR immediate data to Accumulator	2	1
ORL	direct,A	OR Accumulator to direct byte	2	1
ORL	direct,#data	OR immediate data to direct byte	3	2
XRL	A,Rn	Exclusive-OR register to Accumulator	1	1
XRL	A,direct	Exclusive-OR direct byte to Accumulator	2	1
XRL	A,@Ri	Exclusive-OR indirect RAM to A	1	1
XRL	A,#data	Exclusive-OR immediate data to A	2	1
XRL	direct,A	Exclusive-OR Accumulator to direct byte	2	1
XRL	direct,#data	Exclusive-OR immediate data to direct	3	2
CLR	A	Clear Accumulator	1	1
CPL	A	Complement Accumulator	1	1
RL	A	Rotate Accumulator Left	1	1
RLC	A	Rotate A Left through the Carry flag	1	1
RR	A	Rotate Accumulator Right	1	1
RRC	A	Rotate A Right through Carry flag	1	1
SWAP	A	Swap nibbles within the Accumulator	1	1

DATA TRANSFER

Mnemonic		Description	Byte	Cyc
MOV	A,Rn	Move register to Accumulator	1	1
MOV	A,direct	Move direct byte to Accumulator	2	1
MOV	A,@Ri	Move indirect RAM to Accumulator	1	1
MOV	A,#data	Mov immediate data to Accumulator	2	1
MOV	Rn,A	Move Accumulator to register	1	1
MOV	Rn,direct	Move direct byte to register	2	2
MOV	Rn,#data	Move immediate data to register	2	1
MOV	direct,A	Move Accumulator to direct byte	2	1
MOV	direct,Rn	Move register to direct byte	2	2
MOV	direct,direct	Move direct byte to direct	3	2
MOV	direct,@Ri	Move indirect RAM to direct byte	2	2

Table 1. (Cont.)

DATA TRANSFER (CONTINUED)

Mnemonic		Description	Byte	Cyc
MOV	direct,#data	Move immediate data to direct byte	3	2
MOV	@Ri,A	Move Accumulator to indirect RAM	1	1
MOV	@Ri,direct	Move direct byte to indirect RAM	2	2
MOV	@Ri,#data	Move immediate data to indirect RAM	2	1
MOV	DPTR,#data16	Load Data Pointer with a 16-bit constant	3	2
MOVC	A,@A+DPTR	Move Code byte relative to DPTR to A	1	2
MOVC	A,@A+PC	Move Code byte relative to PC to A	1	2
MOVX	A,@Ri	Move External RAM (8-bit addr) to A	1	2
MOVX	A,@DPTR	Move External RAM (16-bit addr) to A	1	2
MOVX	@Ri,A	Move A to External RAM (8-bit addr)	1	2
MOVX	@DPTR,A	Move A to External RAM (16-bit addr)	1	2
PUSH	direct	Push direct byte onto stack	2	2
POP	direct	Pop direct byte from stack	2	2
XCH	A,Rn	Exchange register with Accumulator	1	1
XCH	A,direct	Exchange direct byte with Accumulator	2	1
XCH	A,@Ri	Exchange indirect RAM with A	1	1
XCHD	A,@Ri	Exchange low-order Digit ind RAM w A	1	1

BOOLEAN VARIABLE MANIPULATION

Mnemonic		Description	Byte	Cyc
CLR	C	Clear Carry flag	1	1
CLR	bit	Clear direct bit	2	1
SETB	C	Set Carry flag	1	1
SETB	bit	Set direct Bit	2	1
CPL	C	Complement Carry flag	1	1
CPL	bit	Complement direct bit	2	1
ANL	C,bit	AND direct bit to Carry flag	2	2
ANL	C,/bit	AND complement of direct bit to Carry	2	2
ORL	C/bit	OR direct bit to Carry flag	2	2
ORL	C,/bit	OR complement of direct bit to Carry	2	2
MOV	C,/bit	Move direct bit to Carry flag	2	1
MOV	bit,C	Move Carry flag to direct bit	2	2

PROGRAM AND MACHINE CONTROL

Mnemonic		Description	Byte	Cyc
ACALL	addr11	Absolute Subroutine Call	2	2
LCALL	addr16	Long Subroutine Call	3	2
RET		Return from subroutine	1	2
RETI		Return from interrupt	1	2
AJMP	addr11	Absolute Jump	2	2
LJMP	addr16	Long Jump	3	2
SJMP	rel	Short Jump (relative addr)	2	2
JMP	@A+DPTR	Jump indirect relative to the DPTR	1	2
JZ	rel	Jump if Accumulator is Zero	2	2
JNZ	rel	Jump if Accumulator is Not Zero	2	2
JC	rel	Jump if Carry flag is set	2	2
JNC	rel	Jump if No Carry flag	2	2
JB	bit,rel	Jump if direct Bit set	3	2
JNB	bit,rel	Jump if direct Bit Not set	3	2
JBC	bit,rel	Jump if direct Bit is set & Clear bit	3	2
CJNE	A,direct,rel	Compare direct to A & Jump if Not Equal	3	2
CJNE	A,#data,rel	Comp, immed; to A & Jump if Not Equal	3	2
CJNE	Rn,#data,rel	Comp, immed, to reg & Jump if Not Equal	3	2
CJNE	@Ri,#data,rel	Comp, immed, to ind, & Jump if Not Equal	3	2
DJNZ	Rn,rel	Decrement register & Jump if Not Zero	2	2
DJNZ	direct,rel	Decrement direct & Jump if Not Zero	3	2
NOP		No operation	1	1

Notes on data addressing modes:

Rn —Working register R0–R7

direct —128 internal RAM locations, any I/O port, control or status register

@Ri —Indirect internal RAM location addressed by register R0 or R1

#data —8-bit constant included in instruction

#data16 —16-bit constant included as bytes 2 & 3 of instruction

bit —128 software flags, any I/O pin, control or status bit

Notes on program addressing modes:

addr16 —Destination address for LCALL & LJMP may be anywhere within the 64-K program memory address space

Addr11 —Destination address for ACALL & AJMP will be within the same 2-K page of program memory as the first byte of the following instruction

rel —SJMP and all conditional jumps include an 8-bit offset byte, Range is +127–128 bytes relative to first byte of the following instruction

Table 2. Instruction Opcodes in Hexadecimal Order

Hex Code	Number of Bytes	Mnemonic	Operands
00	1	NOP	
01	2	AJMP	code addr
02	3	LJMP	code addr
03	1	RR	A
04	1	INC	A
05	2	INC	data addr
06	1	INC	@R0
07	1	INC	@R1
08	1	INC	R0
09	1	INC	R1
0A	1	INC	R2
0B	1	INC	R3
0C	1	INC	R4
0D	1	INC	R5
0E	1	INC	R6
0F	1	INC	R7
10	3	JBC	bit addr, code addr
11	2	ACALL	code addr
12	3	LCALL	code addr
13	1	RRC	A
14	1	DEC	A
15	2	DEC	data addr
16	1	DEC	@R0
17	1	DEC	@R1
18	1	DEC	R0
19	1	DEC	R1
1A	1	DEC	R2
1B	1	DEC	R3
1C	1	DEC	R4
1D	1	DEC	R5
1E	1	DEC	R6
1F	1	DEC	R7
20	3	JB	bit addr, code addr
21	2	AJMP	code addr
22	1	RET	
23	1	RL	A
24	2	ADD	A,#data
25	2	ADD	A,data addr
26	1	ADD	A,@R0
27	1	ADD	A,@R1
28	1	ADD	A,R0
29	1	ADD	A,R1
2A	1	ADD	A,R2
2B	1	ADD	A,R3
2C	1	ADD	A,R4
2D	1	ADD	A,R5
2E	1	ADD	A,R6
2F	1	ADD	A,R7
30	3	JNB	bit addr, code addr
31	2	ACALL	code addr
32	1	RETI	
33	1	RLC	A
34	2	ADDC	A,#data
35	2	ADDC	A,data addr
36	1	ADDC	A,@R0
37	1	ADDC	A,@R1
38	1	ADDC	A,R0
39	1	ADDC	A,R1
3A	1	ADDC	A,R2
3B	1	ADDC	A,R3
3C	1	ADDC	A,R4
3D	1	ADDC	A,R5
3E	1	ADDC	A,R6
3F	1	ADDC	A,R7
40	2	JC	code addr
41	2	AJMP	code addr
42	2	ORL	data addr,A
43	3	ORL	data addr,#data
44	2	ORL	A,#data
45	2	ORL	A,data addr
46	1	ORL	A,@R0
47	1	ORL	A,@R1
48	1	ORL	A,R0
49	1	ORL	A,R1
4A	1	ORL	A,R2
4B	1	ORL	A,R3
4C	1	ORL	A,R4
4D	1	ORL	A,R5
4E	1	ORL	A,R6
4F	1	ORL	A,R7
50	2	JNC	code addr
51	2	ACALL	code addr
52	2	ANL	data addr,A
53	3	ANL	data addr,#data
54	2	ANL	A,#data
55	2	ANL	A,data addr
56	1	ANL	A,@R0
57	1	ANL	A@R1
58	1	ANL	A,R0
59	1	ANL	A,R1
5A	1	ANL	A,R2
5B	1	ANL	A,R3
5C	1	ANL	A,R4
5D	1	ANL	A,R5
5E	1	ANL	A,R6
5F	1	ANL	A,R7
60	2	JZ	code addr
61	2	AJMP	code addr
62	2	XRL	data addr.A
63	3	XRL	data addr,#data
64	2	XRL	A,#data
65	2	XRL	A,data addr

Table 2. (Cont.)

Hex Code	Number of Bytes	Mnemonic	Operands
66	1	XRL	A,@R0
67	1	XRL	A,@R1
68	1	XRL	A,R0
69	1	XRL	A,R1
6A	1	XRL	A,R2
6B	1	XRL	A,R3
6C	1	XRL	A,R4
6D	1	XRL	A,R5
6E	1	XRL	A,R6
6F	1	XRL	A,R7
70	2	JNZ	code addr
71	2	ACALL	code addr
72	2	ORL	C,bit addr
73	1	JMP	@A+DPTR
74	2	MOV	A,#data
75	3	MOV	data addr,#data
76	2	MOV	@R0,#data
77	2	MOV	@R1,#data
78	2	MOV	R0,#data
79	2	MOV	R1,#data
7A	2	MOV	R2,#data
7B	2	MOV	R3,#data
7C	2	MOV	R4,#data
7D	2	MOV	R5,#data
7E	2	MOV	R6,#data
7F	2	MOV	R7,#data
80	2	SJMP	code addr
81	2	AJMP	code addr
82	2	ANL	C,bit addr
83	1	MOVC	A,@A+PC
84	1	DIV	AB
85	3	MOV	data addr, data addr
86	2	MOV	data addr,@R0
87	2	MOV	data addr,@R1
88	2	MOV	data addr,R0
89	2	MOV	data addr,R1
8A	2	MOV	data addr,R2
8B	2	MOV	data addr,R3
8C	2	MOV	data addr,R4
8D	2	MOV	data addr,R5
8E	2	MOV	data addr,R6
8F	2	MOV	data addr,R7
90	3	MOV	DPTR,#data
91	2	ACALL	code addr
92	2	MOV	bit addr,C
93	1	MOVC	A,@A+DPTR
94	2	SUBB	A,#data
95	2	SUBB	A,data addr
96	1	SUBB	A,@R0
97	1	SUBB	A,@R1
98	1	SUBB	A,R0
99	1	SUBB	A,R1
9A	1	SUBB	A,R2
9B	1	SUBB	A,R3
9C	1	SUBB	A,R4
9D	1	SUBB	A,R5
9E	1	SUBB	A,R6
9F	1	SUBB	A,R7
A0	2	ORL	C,/bit addr
A1	2	AJMP	code addr
A2	2	MOV	C,bit addr
A3	1	INC	DPTR
A4	1	MUL	AB
A5		reserved	
A6	2	MOV	@R0,data adr
A7	2	MOV	@R1,data addr
A8	2	MOV	R0,data addr
A9	2	MOV	R1,data addr
AA	2	MOV	R2,data addr
AB	2	MOV	R3,data addr
AC	2	MOV	R4,data addr
AD	2	MOV	R5,data addr
AE	2	MOV	R6,data addr
AF	2	MOV	R7,data addr
B0	2	ANL	C,/bit addr
B1	2	ACALL	code addr
B2	2	CPL	bit addr
B3	1	CPL	C
B4	3	CJNE	A,#data,code addr
B5	3	CJNE	A,data addr,code addr
B6	3	CJNE	@R0,#data,code addr
B7	3	CJNE	@R1,#data,code addr
B8	3	CJNE	R0,#data,code addr
B9	3	CJNE	R1,#data,code addr
BA	3	CJNE	R2,#data,code addr
BB	3	CJNE	R3,#data,code addr
BC	3	CJNE	R4,#data,code addr
BD	3	CJNE	R5,#data,code addr
BE	3	CJNE	R6,#data,code addr
BF	3	CJNE	R7,#data,code addr
C0	2	PUSH	data addr
C1	2	AJMP	code addr
C2	2	CLR	bit addr
C3	1	CLR	C
C4	1	SWAP	A
C5	2	XCH	A,data addr
C6	1	XCH	A,@R0
C7	1	XCH	A,@R1
C8	1	XCH	A,R0
C9	1	XCH	A,R1
CA	1	XCH	A,R2
CB	1	XCH	A,R3

Table 2. (Cont.)

Hex Code	Number of Bytes	Mnemonic	Operands
CC	1	XCH	A,R4
CD	1	XCH	A,R5
CE	1	XCH	A,R6
CF	1	XCH	A,R7
D0	2	POP	data addr
D1	2	ACALL	code addr
D2	2	SETB	bit addr
D3	1	SETB	C
D4	1	DA	A
D5	3	DJNZ	data addr,code addr
D6	1	XCHD	A,@R0
D7	1	XCHD	A,@R1
D8	2	DJNZ	R0,code addr
D9	2	DJNZ	R1,code addr
DA	2	DJNZ	R2,code addr
DB	2	DJNZ	R3,code addr
DC	2	DJNZ	R4,code addr
DD	2	DJNZ	R5,code addr
DE	2	DJNZ	R6,code addr
DF	2	DJNZ	R7,code addr
E0	1	MOVX	A,@DPTR
E1	2	AJMP	code addr
E2	1	MOVX	A,@R0
E3	1	MOVX	A,@R1
E4	1	CLR	A
E5	2	MOV	A,data addr

Hex Code	Number of Bytes	Mnemonic	Operands
E6	1	MOV	A,@R0
E7	1	MOV	A,@R1
E8	1	MOV	A,R0
E9	1	MOV	A,R1
EA	1	MOV	A,R2
EB	1	MOV	A,R3
EC	1	MOV	A,R4
ED	1	MOV	A,R5
EE	1	MOV	A,R6
EF	1	MOV	A.R7
F0	1	MOVX	@DPTR,A
F1	2	ACALL	code addr
F2	1	MOVX	@R0,A
F3	1	MOVX	@R1,A
F4	1	CPL	A
F5	2	MOV	data addr,A
F6	1	MOV	@R0,A
F7	1	MOV	@R1,A
F8	1	MOV	R0,A
F9	1	MOV	R1,A
FA	1	MOV	R2,A
FB	1	MOV	R3,A
FC	1	MOV	R4,A
FD	1	MOV	R5,A
FE	1	MOV	R6,A
FF	1	MOV	R7,A

Design Considerations When Using CHMOS

12

CHAPTER 12
DESIGN CONSIDERATIONS WHEN USING CHMOS

12.0 WHAT IS CHMOS?

CHMOS is Intel's n-well CMOS process which is based on the highly developed HMOS-II technology. There are three other types of CMOS processes: p-well, twin-tub, and silicon on sapphire (SOS). All four CMOS structures are discussed in the accompanying article reprint, ''Inside CMOS Technology.'' SOS and twin-tub offer superior performance, but are very costly to manufacture. The n-well technology offers about the same performance as p-well and has been chosen for Intel's microcontrollers because it is more readily adapted to the currently used and well understood HMOS-II technology. This CMOS technology also offers a known path to higher performance products in the future.

Because CMOS tends to have lower gate density and a higher gate count than an NMOS circuit of the same functionality, the ability to scale down the transistor size in CMOS processes is essential to improving the price/performance ratio. The penalty paid for the size reduction, however, is a departure from the traditional CMOS supply voltage range of 3 to 18 volts. CHMOS will be limited to a maximum of 6 volts VCC.

Further reduction in CHMOS die size is accomplished by using dynamic nodes at appropriate points in the circuit, whereas, traditional CMOS is fully static. This reduces the gate count, and therefore, the die size — achieving lower cost. However, the use of dynamic nodes imposes a minimum clock frequency requirement on the CHMOS part.

12.1 NOISE IMMUNITY

CMOS noise immunity is greatly over stated. Noise immunity has been described as the amount of noise that can be induced at the input of a gate that will not change the logic state of the output. Noise margin, on the other hand, is the DC levels that will be applied to an input from another output (Voh, Vol) and the trip point of that input.

On the surface, CMOS would seem ideal for use in noisy environments. Output voltages are rail-to-rail, and input switch points are approximately 50% of VCC. There is one thing wrong with this analysis — CMOS has high impedance inputs. High impedance inputs need only voltage and very little current (nano Amps) to switch its output logic state. TTL, on the other hand, needs voltage and at least 20 μA (for an LS device) to switch its output logic state. Because it is a lot more difficult to induce noise in the form of current than in the form of voltage, CMOS tends to be more noise sensitive than TTL.

However, NMOS also has high impedance inputs and has input leakages typically less than 1 μA. Because NMOS output voltage swings are considerably less than CMOS, CMOS has the advantage when inputs and outputs are noisy. Another advantage of CMOS over NMOS is the p-channel pullup instead of a depletion pullup. The p-channel pullup is able to charge up the stray capacitance faster thus signal rise times are significantly improved.

In conclusion, don't fool yourself into thinking that CMOS eliminates the need to be concerned about noise problems. Time is well spent following good design practices and layout techniques from the earliest phases of a project.

12.2 LATCH-UP

CHMOS is not immune to traditional CMOS latch-up, but the latch-up threat is highly overrated. Latch-up is usually the result of unforeseen operating conditions, such as an unexpected power-up sequence, inadvertent removal and re-application of a supply voltage, or ''hot-socketing'' the part (plugging a chip into its socket while the system is active). An energetic voltage spike on VCC or the I/O lines might also trigger latch-up, but a little ringing on the data lines isn't going to cause any problems.

It is helpful to understand the mechanisms involved in the latch-up phenomenon. Figure 12-1A shows the circuit diagram of a typical CMOS output stage. Figure 12-1B shows a plan view of how this stage might look in n-well CMOS (with the gate electrodes of the FETs stripped away, since they are not germane to the discussion). There are two parasitic bipolar transistors in this structure, one pnp and the other npn. The n-well forms the base region of the pnp transistor, which is connected to VCC through the distributed resistance of the n-well. The source and drain of the p-channel pullup FET are dual emitters to the pnp transistor. The p-type substrate is the collector of this transistor and also serves as the base of the npn transistor which is connected to VSS through the distributed resistance of the substrate. The collector of this transistor is the n-well, and the drain and source of the pulldown n-channel FET are dual emitters. These parasitic transistors are shown in Figure 12-1C.

Any pullup FET that shares the same n-well region acts as additional multiple emitters to the parasitic pnp transistor. Much worse, ALL pulldown FETs and input protection devices on the IC act as additional collectors to the parasitic npn transistor. This has several implications. Latch-up is not limited to output stages, inputs only pins and internal gates can also be the cause. So when latch-up does occur, it's hard to tell which device caused it.

In normal operation, both of these parasitic bipolar junction transistors are in an off state, and do not hamper the operation of the FETs. However, if either parasitic junction transistor should turn on, its collector current might turn the other parasitic junction transistor on, and an SCR type effect rapidly ensues which is called latch-up.

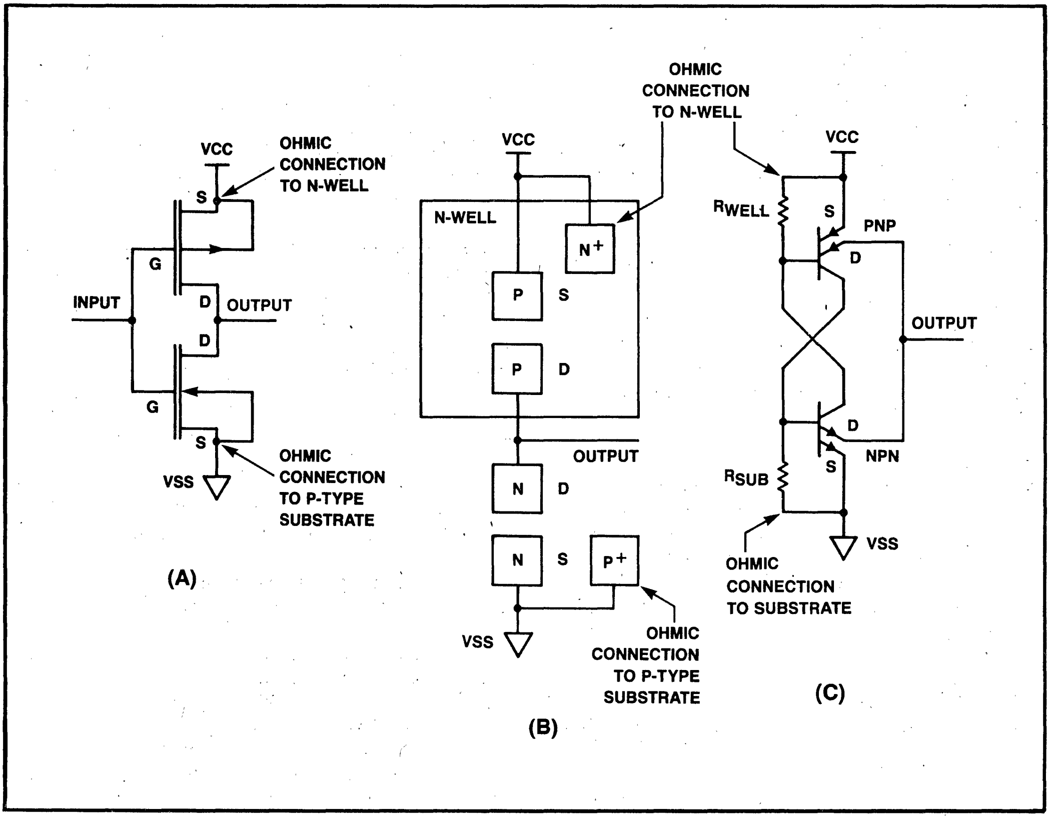

VCC
OHMIC CONNECTION TO N-WELL
S
G
INPUT
D
OUTPUT
D
G
S
VSS
OHMIC CONNECTION TO P-TYPE SUBSTRATE
(A)
VCC
N-WELL
N+
P
S
P
D
OUTPUT
N
D
N
S
P+
VSS
OHMIC CONNECTION TO N-WELL
OHMIC CONNECTION TO P-TYPE SUBSTRATE
(B)
VCC
RWELL
S
PNP
D
OUTPUT
D
NPN
S
RSUB
VSS
OHMIC CONNECTION TO SUBSTRATE
(C)

Figure 12-1. CHMOS Inverter

What might turn on one of the parasitic junction transistors? Look at Figure 12-1C. The base of the parasitic pnp is at VCC, and the base of the npn is at VSS. If the output line swings a diode drop above VCC, or below VSS, it forward-biases the "D" emitter of either the pnp or the npn transistor.

Suppose it is the pnp emitter that is forward-biased by the output line swinging a diode drop above VCC. Some of the current that enters that emitter now exits the parasitic device as collector current. It flows down to the juncture of Rsub and the base of the npn transistor. If Rsub is low, the current is shunted through it to VSS, and the npn transistor stays off. If Rsub is high, some of the current crosses the base-emitter junction of the npn, and that transistor will likely turn on.

It is important to note that the designer of the integrated circuit has a certain amount of control over the turn-on probabilities. Grounded guard rings placed around the npn emitters will have the effect of reducing the value of Rsub. Reducing the value of Rsub has the effect of increasing the amount of current that has to enter the pnp D emitter in order to turn the npn transistor on. The CHMOS, the npn transistor will not turn on if the current entering the pnp D emitter is less than 10 mA.

A similar sequence of events can occur when the output line swings a diode drop below VSS. In this case it is the npn transistor that is in danger of turning on the pnp, depending on the value of Rwell and how much current is involved. Again, the IC designer can reduce the value of Rwell by the judicious placement of guard rings, connected to VCC, around the pnp emitters. When the output line swings a diode drop below VSS, if the current that exits the output pin does not exceed 10 mA, there will be no latch-up.

12.3 POWER SUPPLY CONSIDERATIONS

The power supply, as viewed by the microcontroller, should be low in inductance because of the peak currents associated with CHMOS switching characteristics. Note that the repetition rate of these peak currents increases with the clock frequency. Bypass capacitors of approximately 0.1uF should be used with proper PCB layout to ensure a low inductance, high peak current power source. Low inductance capacitors are also available that fit under the package. These capacitors are also advantages for HMOS in noisy environments (See Application Note AP125 ''Designing Microcontroller Systems For Electrically Noisy Environments'' in this manual for detailed discussion).

Power supply glitches must be filtered to ensure that the maximum voltage rating of the device is not violated. Violation may induce an SCR effect between VCC/VDD and Vss (latch-up).

The polarity of the power supply must never be reversed. The n-well is connected to VCC and the p-type substrate is connected to ground so normally that pn junction is reversed biased. If VCC or VDD are ever more negative than −0.5 volts with respect to Vss, the n-well/substrate pn junction will be forward biased and short VCC/VDD to ground.

When the microcontroller is powered separately from the surrounding circuitry, the microcontroller should always be powered up before any input signal is applied. The reverse is true when powering down, input signals first then the microcontroller.

When separate VCC and VDD power supplies are used for the 80C49, VDD and VCC must track each other within 1.5 volts (except during power down) and also maintain the 5v 20% specification. This ensures that the CPU, powered by VCC, and the RAM, powered by VDD, have proper voltage levels to communicate. If VCC and VDD cannot be powered up simultaneously, VDD should be applied first.

12.4 MINIMIZING POWER CONSUMPTION

The reason CMOS parts draw considerably less current than an NMOS part is that there is no direct path between VCC and ground (refer to figure 10-1A). When the p-channel pullup FET is on the n-channel pulldown FET is off and the output line gets pulled up. The opposite is true, when the n-channel FET is on the p-channel FET is off and the output goes low. There is leakage associated with either the n-channel and p-channel FETs in the off state. The sum of all the leakages from every inverter on the chip is what is called the ''quiescent current.'' This current is in the order of microAmps and is the lesser of the two currents that add up to the total ICC.

When the inverter is switching in either direction, there is a moment in time when both FETs are on creating a low resistance path between VCC and ground. The capacitance on the output line (pin capacitance, trace capacitance, and input capacitance to the next stage) is also charged or discharged which also increases current consumption. This ''dynamic current'' is a couple of orders of magnitude higher than the quiescent current.

Power supply voltage also comes into play with both types of currents. If VCC is high, the leakage across the off FET is increased. Output capacitance now has to be charged up to a higher level. So a higher VCC also contributes to making ICC larger.

Frequency of switch can also affect the amount of current used. If the inverter stage is switched very slowly, then the average current is equal to the quiescent current. As the frequency of switching is increased, the dynamic current contributes more and more to the total ICC until the dynamic current totally swamps the quiescent current.

Now that we are aware of what components make up the total ICC, we can work on ways of minimizing it. VCC can play a big roll in minimizing power. If the whole purpose in going to CMOS is to minimize heat dissipation and the power is drawn from the 110 volt AC socket, tweak your power supply to the minimum voltage that the system will run. On the other hand if batteries are used you must consider types of batteries available, lifetime of the battery, and voltage drop off at the end of its life to determine what voltage to start at. Remember the lower the voltage the lower the current draw.

The application also has to be analyzed for what kind of response time is needed to determine how fast the microcontroller needs to run. If the end application is a direct human interface, then the response time can be relatively slow and the processor can run at minimum speed. But if real time decisions need to be made on evaluating incoming data, then the microcontroller needs all the time it can get. The idea is to run the microcontroller as slow as you can and still get the task accomplished.

Intel has added idle mode and power down features to help further manage your power consumption. Idle mode in the 80C51/80C31 stops the clock to the CPU while keeping the oscillator, RAM, interrupts, timer/counter, and serial port alive. Stopping the clock to the CPU decreases the current in the CPU from dynamic to mostly quiescent. Idle mode consumes approximately 1/10th the operating current.

Idle mode allows the microcontroller to minimize its current when no processing needs to be done. The live interrupts, timer/counter, and the serial port can wake the CPU and since the oscillator is running the response time is quick.

Data on the ports are left in the state they were in when idle was invoked. If careful attention is given to the logic state of the port, total system current can be reduced. The state of the port for reduced current is dependent on what the port is driving. If it is a transistor, the transistor should be put into its off state. If the port pin is driving another CMOS gate, the loading of the port pin would be minimal and a choice of the logic state may be made on what is

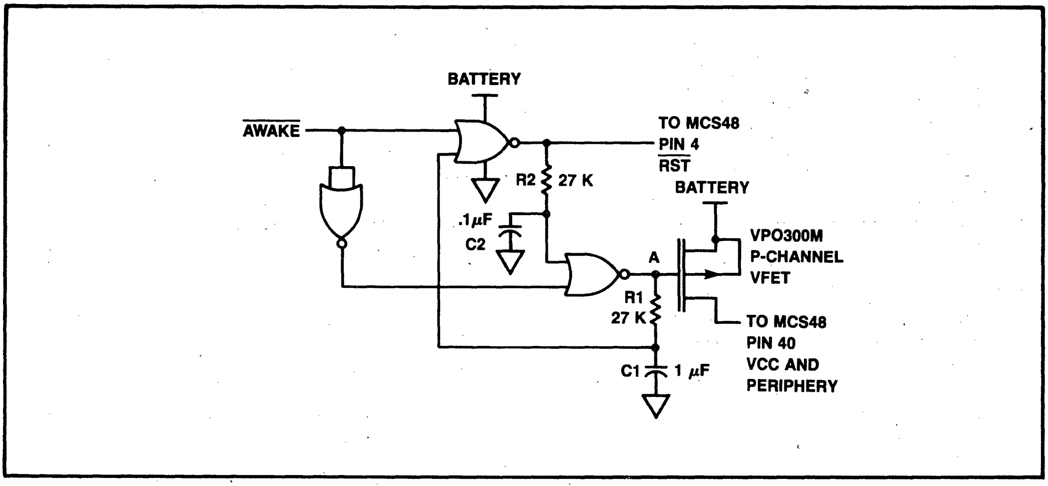


Figure 12-2. MCS®-48 Power Down Circuit

on the outputs of the CMOS gate. If TTL is being driven, the reduced current would be when the port pin is high. A little thought in this area can go a long way in minimizing system current.

Power down removes all internal clocks to decrease the current to totally quiescent current while the contents of the RAM are saved. This mode is useful in hand held applications where data needs to be saved between uses. On the 80C51, the port pins are left in the same state they were in when the power down mode was called. The same thought processes that were needed for reduced current in the idle mode are valid for power down also.

Figure 12-2 shows a simple circuit that uses one quad NOR CMOS integrated circuit and some external resistors and capacitors to power down the CHMOS MCS48 family under the control of one input. To activate the operation of the microcontroller, the AWAKE/ signal is pulled low and in turn, node A goes low turning on the p-channel VFET. This allows VCC to the microcontroller to be pulled to the power supply minus Vsd. Node A being low brings RESET high after a time determined by the R1C1 time constant. The RC has been chosen to allow 10 mS between VCC going high and RESET going high to allow the oscillator time to stabilize.

Powering down is accomplished by AWAKE/ going high which pulls RESET low. The time delay R2C2 allows the reset signal enough time to signal the microcontroller to save the RAM before VCC is shut off.

When idle mode and power down are used in conjunction with slow operating speed it can reduce your power needs to a minimum.

Unused input only pins on the MCS48 family such as SS/, TO, and T1 should not be allowed to float. The inputs would float about the trip point of the input buffer. This switching of the input buffer can waste up to .5mA per input. Tie all unused input only pins high or low.

When the CHMOS units are being used with external program or data memory, PortO (Data Bus on the MCS48 products) is left floating when in idle mode. PortO also floats on the 80C51 when in the power down mode. The same condition as described above can happen with the input gate on those pins. Without tying these pins high or low at least .5 mA x 8 or 4 mA could be wasted. Tie the PortO pins high or low through a 500KΩ or 1MΩ resistor. A little extra board space is minimal when considering the extra power savings.

The quasi-bidirectional pins have internal pullups so the inputs on these pins never float.

12.5 CHMOS I/O PORT STRUCTURE

The CHMOS I/O ports have similar drive capability to their HMOS counterparts, but the differences must be noted.

12.5.1 As An Output Pin

The I/O port structure is implemented as shown in Figure 12-3.

As an output pin latched to a low (0) state, pullups P1, P2, and P3 are in an off state while the pulldown N1 is on. This configuration uses little current, since there is no path between VCC and ground in the output buffer stage.

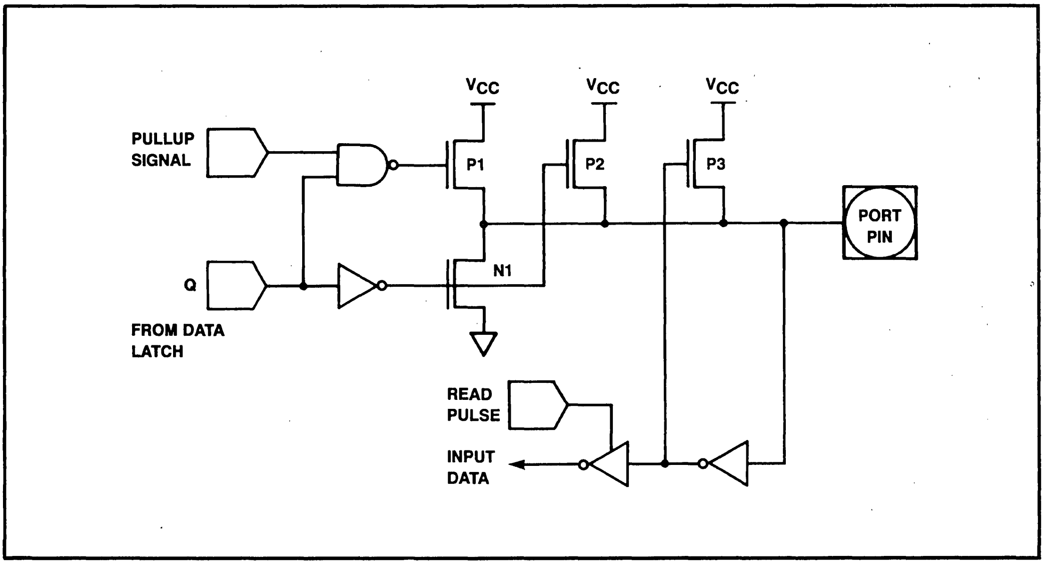
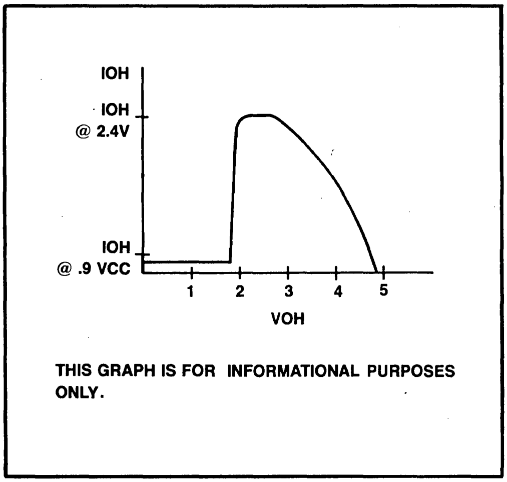


Figure 12-3. CHMOS Quasi-Bidirectional I/O Port Structure



THIS GRAPH IS FOR INFORMATIONAL PURPOSES ONLY.

Figure 12-4. CHMOS I/O Current Characteristics

When the output pin is to be latched in the high (1) state, the data line turns N1 off and turns on the weak (5uA) pullup P2. At the same time P1 (a strong pullup) turns on for one state time pulling the output up very quickly. Once the output voltage is above approximately 2 volts, P3 turns on to supply the source current. P3 and the inverter form a latch. This latch along with the support of P2 keeps the output high. Figure 12-4 shows the VOH vs IOH curve for this output structure when P1 is off.

12.5.2 As An Input Pin

To use the I/O port as an input, a one must be written to the pin first, leaving the pin in a high state. When the input goes below 2 volts, P3 turns off to avoid any high sink currents from being presented to the input device. Note when returning back to a one, P2 is the only internal pullup that is on. This will result in a long rise time if P2 is the only pullup.

12.5.3 Interfacing Between CHMOS and Other Logic Families

Interfacing Intel's CHMOS to other logic families is very simple and straight forward. When VCC is kept within 10% of 5 volts all inputs (except those noted in the data sheets) and outputs are TTL compatible. CMOS compatibility is achieved as long as VCC is kept within 20% of 5 volts.

When driving a high current load, the output current (Ioh) must be limited to keep the output voltage (Voh) at a minimum of 2 volts. If the voltage is pulled below 2 volts the output current will be dropped to approximately 5 μA. See Figure 12-4.

Inside CMOS Technology

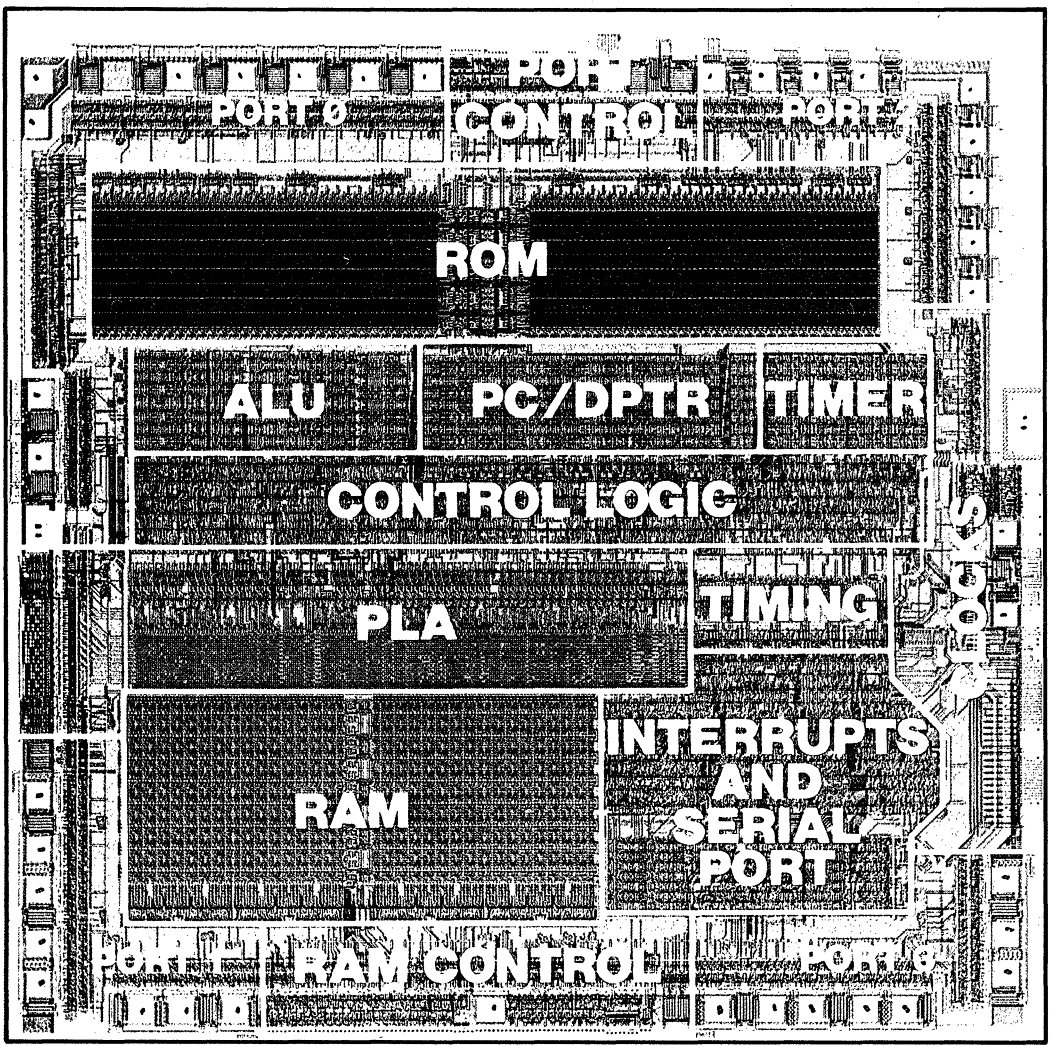


Photo 1: *The die for the 80C51, with the functions of the various sections identified.*

How CMOS devices are manufactured and a look at three of them

by Martin B. Pawloski, Tony Moroyan, and Joe Altnether

Order Number: 230860-001

CMOS (complementary metal-oxide semiconductor) has often been called the ideal technology. It has low power dissipation, high immunity to power-supply noise, symmetric switching characteristics, and a large supply-voltage tolerance. But CMOS has rarely been used for advanced VLSI (very-large-scale-integration) microcomputer designs. Because of the complexity of the CMOS process, the ICs (integrated circuits) produced have traditionally had a relatively poor price/performance ratio.

As a result, CMOS was used only in applications that required low power and were neither performance conscious (such as in calculators and watches) nor cost conscious (many military applications, for example). Suddenly, however, all major semiconductor companies have announced either advanced CMOS products or the intention of designing their next generation of high-performance microprocessors using CMOS technology.

What has happened to make CMOS both affordable and high performance? For one thing, the dominant VLSI technology, NMOS (n-channel metal-oxide semiconductor), is rapidly approaching the process complexity of standard CMOS. It is not unusual nowadays for NMOS technology to have up to four transistor types with different operating characteristics. Much of the complexity of this process is added simply to help VLSI designers keep the operating power of their circuits under control.

Second, CMOS circuit designers are being more selective in the use of static CMOS logic. Critically placed dynamic logic, creative circuit design, and use of modes that offer varying degrees of power consumption are all tricks designers are using to maintain the advantages of CMOS.

Finally, aggressive reduction in CMOS transistor size is being used to bring CMOS performance in line with that of NMOS. As a matter of fact, many manufacturers are developing CMOS as a derivative of their advanced NMOS processes. This not only improves CMOS performance levels but also boosts reliability and reduces development costs.

The Evolution of LSI

Early LSI circuits were built with p-channel MOS transistors, which permitted high-circuit densities yet were relatively slow and difficult to interface to normal integrated circuits, such as TTL (transistor-transistor logic). As an example, the 1103-type 1K by 1-bit dynamic RAM (random-access read/write memory), circa 1971, required its inputs (address, controls, and data) to swing between 1 and 15 volts (V) although its output was measured in millivolts — hardly TTL compatible! About 1974, NMOS came to the rescue. It provided faster speed, and most of its inputs and outputs were TTL compatible.

Low power requirements are a major advantage of designing a system that uses CMOS.

NMOS was more difficult to manufacture than PMOS because contaminants would vary the thresholds of the n-channel transistors, causing deviations in speed and performance. But this problem was quickly overcome through ultraclean processing rooms, and NMOS became the workhorse technology because it cost less to manufacture, was easy to use, and had good speed-power characteristics. And NMOS technology had potential for greater improvement of its speed-power characteristics through scaling (or shrinking) of the silicon devices. The result of this scaling was HMOS (high-speed NMOS), which accomplished three objectives: increased speed, reduced power, and increased density.

Over the past 10 years, the reduction in transistor size has, at the device level, increased memory density by a factor of 64, increased speed by a factor of 3, and reduced power consumption by a factor of 100. However, the scaling cannot continue ad infinitum because of resolution limitations of the photolithographic equipment used to make the circuits as well as breakdown mechanisms within the devices. More important, even before these limitations are reached, heat dissipation will prohibit major enhancements with NMOS. Heat generation increases exponentially with transistor count, and, at densities approaching 150,000 transistors per integrated circuit, special cooling measures are required. This heat can accelerate failure mechanisms within the silicon, reducing device and system reliability. To hurdle this barrier, low-power devices must be used.

The Importance of Power Consumption

The development of NMOS was spurred on by the semiconductor industry's drive to produce high-volume, large-capacity memory devices, for which high density, rather than low power consumption, was the primary concern. As VLSI began to emerge, however, power dissipation became a limiting factor in continued increases in NMOS packing densities. Thus, the semiconductor industry turned to CMOS as a potential alternative.

CMOS achieves its low power dissipation through the use of both p- and n-channel transistors (hence the name "complementary"). Essentially, no DC power is dissipated in either logical state, and AC power occurs only during the relatively short switching period. Because most circuitry in a complex design is active only 10 to 20 percent of the time, CMOS achieves a dramatic reduction in power dissipation compared with NMOS, which continually dissipates DC power whenever an operating voltage is applied.

Low power requirements are a major advantage of designing a system that uses CMOS. Reducing power requirements has a domino effect that often substantially reduces the cost of the end product:

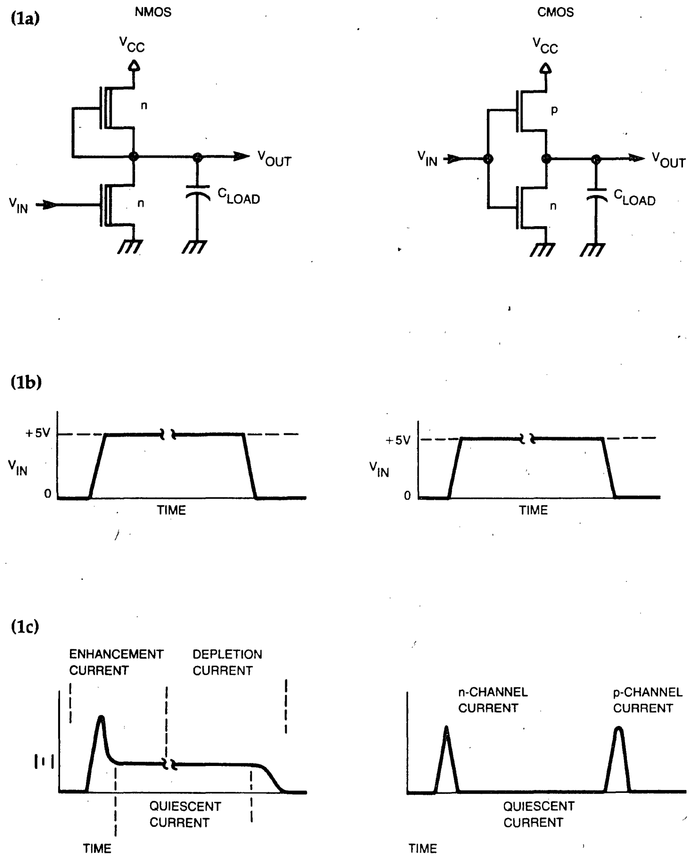


Figure 1: *A comparison of NMOS and CMOS technologies. Figure 1a shows the schematic diagrams of an inverter as implemented in both NMOS and CMOS. A hypothetical input waveform and the resulting transistor currents are shown in 1b and 1c.*

- low power allows smaller, lower-cost power supplies to be used .
- power distribution in the system is simplified
- cooling fans can be eliminated
- printed-circuit boards can be packed more densely and can thus become smaller

With smaller power supplies, denser circuit boards, and no fans, smaller cabinets can be used, resulting in savings in chassis and enclosure costs. Also, power fail-safe and hand-held use become possible if battery operation is feasible.

Basic CMOS Operation

To truly understand the promises (and problems) facing both the CMOS VLSI digital designer and the CMOS systems designer, one must first understand some CMOS fundamentals.

Figure 1 compares the circuit diagrams and current characteristics of both an NMOS and a CMOS inverter. The NMOS inverter uses an n-channel depletion-mode transistor as the pull-up device (which drives the output line high) and an n-channel enhancement-mode transistor as the pull-down device (which drives the output line low). The pull-up transistor is used as a load; its operation approximates that of a constant current source. The pull-down transistor is used as the switching device; when active, it discharges the load, and when inactive it lets the pull-up charge the load. MOS loads are primarily capacitive and include the parasitic capacitances of the inverter itself, interconnect capacitances, and the thin-oxide capacitances of all the gates the inverter is driving.

Let's note several characteristics of an NMOS inverter. When the pull-down device is turned on, it not only has to sink the current from the capacitive load, but it also has to sink the current supplied by the pull-up load device. Even in the quiescent state this current component from the pull-up device still exists. Because logic gates spend most of their time in the quiescent state, this quiescent current accounts for up to 90 percent of the total power dissipated in NMOS VLSI designs; the remaining 10 percent is switching or *dynamic* power.

A second related characteristic is that the inverter's output voltage in the low state, $V_{OL,}$ is dependent on the ratio of the impedances of the pull-down and pull-up devices. This ratio affects the noise margin and switching speed and is generally around 4:1. Such a ratio results in a V_{OL} on the order of 0.2 V to 0.3 V. It also causes asymmetric switching characteristics: the fall time of the inverter is significantly faster than its rise time.

The CMOS inverter uses a p-channel enhancement-mode transistor as the pull-up device and an n-channel enhancement-mode transistor as the pull-down device. In a CMOS inverter, both the pull-up and pull-down transistors are used as switching devices. When the input changes from low to high, the p-channel device shuts off and the n-channel transistor *discharges* the load. When the input changes from

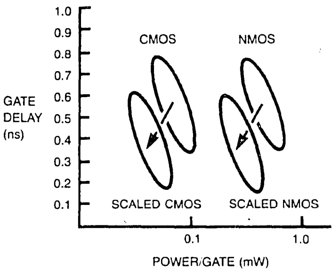


Figure 2: *The speed versus power consumption characteristics of NMOS and CMOS technologies. Note the advantages gained by scaling (reducing the size) of the integrated components.*

high to low, the n-channel device shuts off and the p-channel transistor *charges* the load. While almost all current from the CMOS inverter is used to charge or discharge the load, a small current component does not flow through the load. This is a result of the fact that both the p-channel and n-channel transistors are on for a short period of time during the input voltage transition. This current component is typically less than 10 percent of the total inverter current, though it depends greatly on the rise and fall times of the input signal.

With no quiescent power component, a CMOS inverter's dynamic power dissipation represents only a small fraction of an equivalent NMOS inverter's power dissipation. Also, the CMOS inverter is a "ratio-less" design, having only one transistor active after an input transition. This lets V_{OL} go all the way to ground potential, resulting in better noise tolerance than NMOS inverters. It is also a simple matter to design CMOS circuits with outputs that have equal rise and fall times. While this is important in some circuits, it is generally not taken advantage of in VLSI designs because it requires greater chip area.

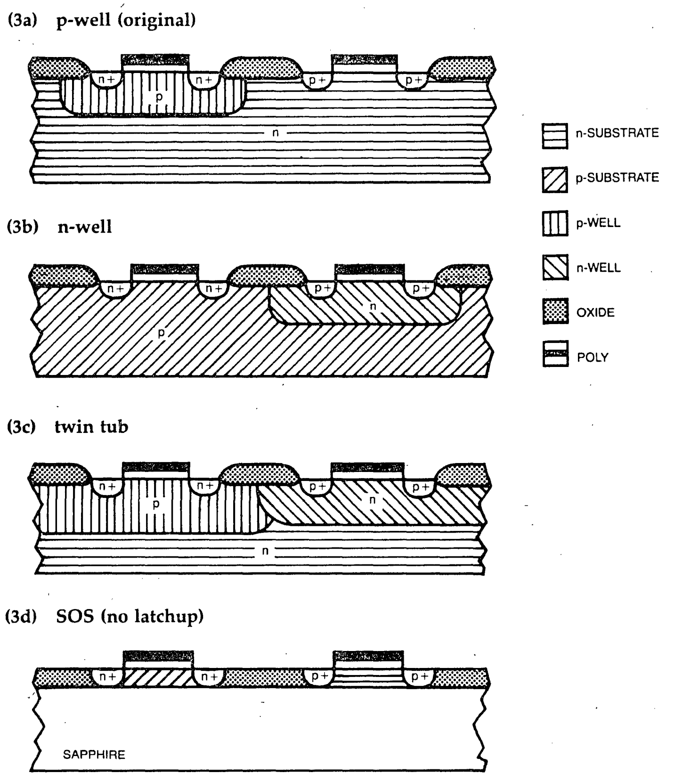
For NMOS and CMOS technologies with similar transistor dimensions and gate oxides, gate delays are essentially identical. The speed-power products for such a set of NMOS and CMOS technologies are shown in figure 2. This graphically illustrates the tremendous power advantage CMOS offers when used in high-performance VLSI designs.

While CMOS enjoys significant electrical advantages over NMOS, it does have a cost disadvantage. One small factor is the larger number of process steps needed to fabricate a CMOS device. More significant is the larger die required because CMOS has lower gate density.

CMOS Technologies

Figure 3 shows the four major CMOS technologies in use today: p-well bulk, n-well bulk, twin-tub bulk, and silicon-on-sapphire (SOS). P-well CMOS uses a p-type diffusion into an n-type bulk silicon substrate to form an n-channel transistor. The p-channel transistor is built directly in the bulk. This is the original CMOS technology, which has many years of good performance and reliability behind it.

The n-well CMOS process starts with a p-type substrate. N-type material is diffused into it to form the n-well in which p-channel devices are built. N-channel devices are built directly in the bulk substrate. An n-well CMOS process is usually derived from an advanced NMOS process. It also permits a highly optimized n-channel transistor, which yields a slight performance advantage over a p-well CMOS process.


(3a) p-well (original)

(3b) n-well

(3c) twin tub

(3d) SOS (no latchup)

Figure 3: *Cross sections of transistors formed by each of the four major CMOS processes. Figure 3a is a p-well bulk CMOS transistor; figure 3b shows an n-well bulk device; figure 3c is an example of a twin-tub bulk CMOS transistor; the transistor in figure 3d is formed using silicon-on-sapphire technology.*

Twin-tub CMOS combines n-well and p-well technologies by diffusing both an n-well for the p-channel transistor and a p-well for the n-channel transistor. The twin wells are usually formed in a lightly doped n-type substrate. While it is a slightly more complex and costly process than either n-well CMOS or p-well CMOS, twin-tub CMOS has the advantage of being able to optimize the performance of both the n-channel and p-channel devices. Thus, this process gives the highest overall performance of the bulk CMOS technologies.

The highest performing CMOS technology is SOS. Silicon islands are grown on an insulating sapphire substrate. N-channel or p-channel transistors are then built on the islands. High performance is achieved due to the significant reduction of parasitic capacitance. SOS also offers good gate density because no parasitic bipolar transistors are around to cause a phenomenon called *latch up.* Unfortunately, SOS devices are difficult and expensive to manufacture. For example, unused sapphire wafers cost approximately 10 times more than bulk silicon wafers.

While CMOS suffers a cost penalty of about 20 percent due to process differences, it generally suffers more significantly because of die size. (While processing steps have a linear relationship with cost, die size has an exponential relationship.) CMOS dies are larger than equivalent NMOS designs even when aggressive transistor scaling is employed. Three major factors contribute to this: the area used in trying to prevent latch up, CMOS logic-gate structure, and static design techniques.

Latch Up Prevention

Bulk CMOS technologies have parasitic bipolar transistors that, if improperly biased, can cause a phenomenon called latch up. This potentially destructive action results from triggering an SCR (silicon-controlled rectifier) formed by the transistors and can cause extremely large currents to flow. Figure 4 shows the construction of the parasitic SCR in an n-well bulk CMOS device.

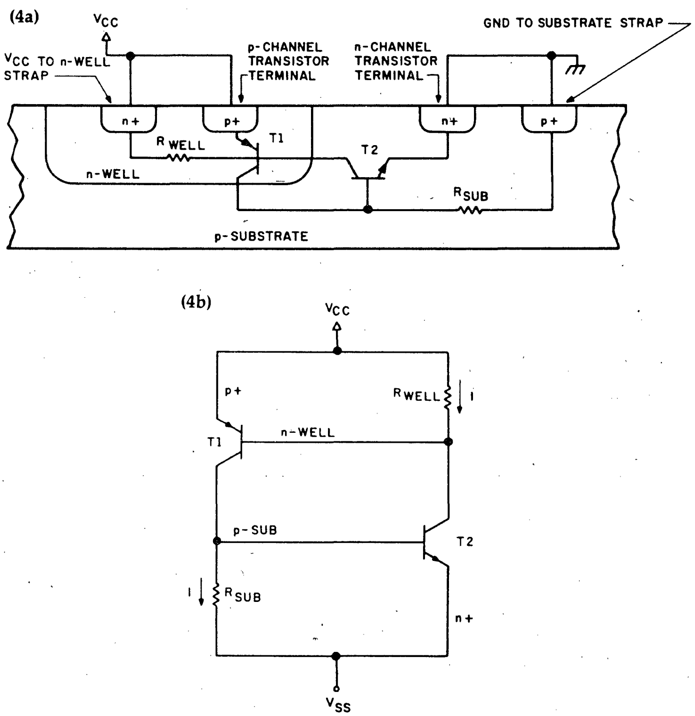


Figure 4: *Parasitic SCR in bulk CMOS can cause latch up. Figure 4a shows how the parasitic transistors are formed in the silicon; figure 4b is a diagram of the equivalent circuit.*

Two well-defined conditions must exist before latch up can occur. First, for the SCR to be triggered, IR_{well} or IR_{sub} must be greater than or equal to 0.7 V. This forward-biases the base-emitter junctions of the parasitic bipolar transistors. Second, to sustain the latch up condition, the product of the βs (gains) of the two bipolar transistors must equal at least 1.

In order to minimize the chance of one of the SCR's transistors being forward-biased, every attempt is made to reduce the resistance values as much as is feasible. This has the effect of requiring significantly larger injected currents before the SCR can be triggered. To reduce the resistance values, *guard rings* are used in the circuitry. (Guard rings are low-resistivity connections to the supply voltages placed around the CMOS p-channel and n-channel transistors.) While guard rings reduce the SCR bias resistor values, they also increase the space between n-channel transistors and p-channel transistors (thus reducing the gate density). To somewhat minimize this effect, particularly sensitive areas (like VLSI component's I/O pins) are heavily guard ringed, while the more protected internal circuitry is less so.

A less controllable method of preventing latch up is to try and decrease the βs of the parasitic transistors. While the vertical pnp transistor's β is set by the process design, the lateral npn transistor is more directly controllable. Its β can be drastically reduced by increasing the n-well-to-n+ diffusion spacing (or p-well-to-p+ diffusion spacing in p-well technology). This method reduces the β by increasing the width of the transistor's base. While this is an effective way of decreasing the gain of the parasitic structure, it also reduces the gate density.

In bulk CMOS technologies, to

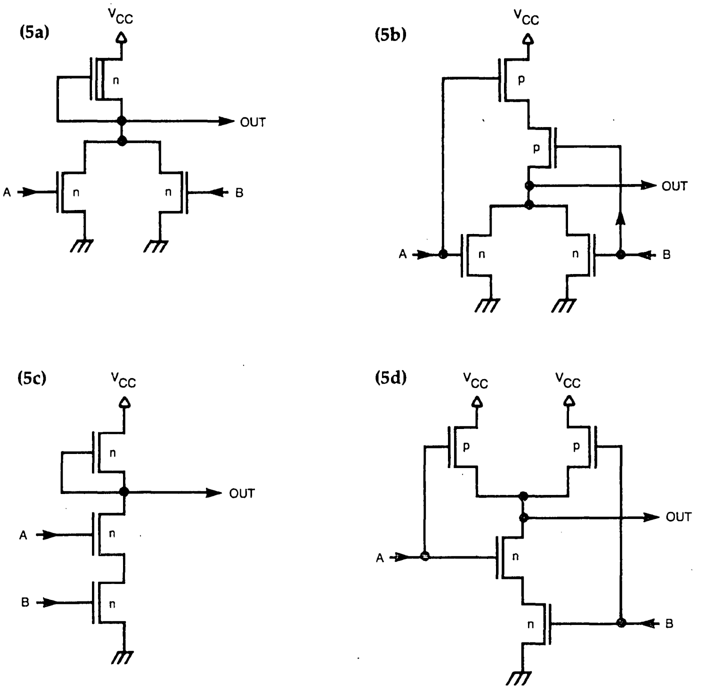


Figure 5: *A comparison of typical logic gates in NMOS and CMOS form. Figure 5a is an NMOS NOR gate, while figure 5b is a CMOS version; figure 5c is an NMOS NAND gate, and figure 5d is a CMOS version.*

give absolute protection against latch up is not only tremendously expensive in silicon area, but it is also virtually impossible. CMOS designers sacrifice area to ensure there is enough margin in their design to protect it from latch up in normal operating-system environments.

Logic-Gate Structures

Gate densities are also reduced in CMOS because standard CMOS logic gates are built from more transistors than their NMOS equivalents. Standard CMOS logic-gate design has a 1:1 ratio of n-channel transistors to p-channel transistors. For example, the two input gates shown in figure 5 take four transistors in CMOS and only three in NMOS. The relative density decreases as the number of inputs increases. For example, three-input gates require six transistors in CMOS and only four in NMOS; four-input gates require eight transistors in CMOS and only five in NMOS, etc. As a matter of fact, it is rare to have a standard CMOS gate with more than three inputs because the self-loading and the transistor stack make the structure inefficient in both speed and area. On the other hand, it is not unusual in NMOS to have gates with as many as eight inputs.

Static Design Techniques

A final reason for the lower CMOS gate densities is the use of static logic (modern VLSI NMOS microcomputer designs rely heavily on dynamic circuitry). Dynamic circuitry essentially uses a small capacitor as a latch to store logic values. This technique saves both area (by reducing the number of transistors in a gate) and power (by reducing the number of gates in structures like latches, flip-flops, shift registers, etc.). Employing dynamic design can reduce an NMOS latch's area by 30 percent and its power consumption by 50 percent. However, the problem with dynamic circuitry is that the capacitor used to store the logic value is leaky and will, over time, discharge and lose its data. This is the same problem faced by dynamic memory designers. The solution is to periodically refresh the capacitor, which forces a minimum operating frequency to be adhered to.

CMOS can also use dynamic circuitry, especially to increase the ratio of n-channel transistors to p-channel transistors. Because static CMOS designs have a 1:1 ratio of n-channel to p-channel transistors, being able to increase this ratio will have the effect of giving CMOS a higher gate density (but the minimum operating-frequency characteristic of dynamic circuitry often conflicts with the CMOS potential of absolutely minimizing power). Therefore, while true static CMOS design does give the lowest possible power consumption (by allowing the device to operate at frequencies all the way to DC), dynamic CMOS designs, being more dense and resulting in smaller die sizes, tend to be more cost-effective. Thus, two trends are developing in the use of CMOS for VLSI microcomputer design.

Designers of the next generation of 16- and 32-bit microprocessors are choosing CMOS. Here, the goal is not to operate at the lowest possible power level but rather to keep the operating power under a maximum level for cooler junction temperatures, higher performance levels, and the ability to use standard low-cost packages. In these designs, extensive use is made of dynamic logic. The ratio of n-channel transistors to p-channel transistors is often as high as 3:1.

Designers of 4- and 8-bit single-chip microcomputers are choosing CMOS to accommodate a host of new portable, hand-held, and ultra-low-power applications. Here, the goal is to minimize the operating power levels consistent with the performance required by the application. In the simpler microcomputers, true CMOS static logic is used—their simpler structure still allows a relatively small die size, while the low-performance applications they are appropriate for allow low operating frequencies. On the other hand, the more complex, higher-performance, single-chip VLSI components still make maximum use of static logic but are forced into dynamic logic for large arrays to keep the die cost down.

Future CMOS

CMOS will be the technology of choice for VLSI microcomputer designs. For one thing, with the advent of hundreds of thousands of transistors on a die, CMOS is the only technology that offers a cost-effective solution to the power-density problem.

A second and more subtle future issue is reduced supply voltage. As MOS transistors continue to be scaled to smaller dimensions to eke out further performance and density advances, the standard 5-V supply voltage must be reduced, if only for internal circuitry, to limit substrate current and hot-electron effects. CMOS is better suited for lower supply-voltage operation because its switch point is a fixed percentage of the supply voltage. Also, due to its "ratio-less" structures, CMOS enjoys better noise tolerance than NMOS, another important factor at lower supply voltages.

Finally, CMOS has made and will continue to make major strides in its relative cost disadvantage to NMOS. Where CMOS formerly sold at as much as a fourfold premium, today it is selling at somewhat less than twice the price of comparable NMOS devices. With its continued use of standard, low-cost packaging technology as well as the more creative use of dynamic circuitry and hybrid static/dynamic designs, CMOS will rapidly approach the cost of NMOS. As a matter of fact, several major semiconductor manufacturers have stated that CMOS/NMOS price parity will occur this decade, and some manufacturers say it will happen as early as 1985. When CMOS and NMOS cost the same, why would anyone buy NMOS?■

A CMOS Single-Chip Computer: Intel's 80C51

by Martin B. Pawloski

Intel's 80C51 is an interesting example of how the static logic versus dynamic logic trade-off was made in an actual product design. The 8051 is an 8-bit, single-chip microcomputer with 4K bytes of ROM, 128 bytes of RAM, two 16-bit counter/timers, multilevel interrupt control, 32 I/O pins, full-duplex UART (universal asynchronous receiver/transmitter), and on-chip oscillator and clock circuits. A die with the sections identified by functions is shown in photo 1 on page 94.

Product	Technology	V_{CC} Supply Voltage Range	Operating-Frequency Range	Normal Operating Mode (I_{CC} Max)	Idle Mode (I_{CC} Max)	Power Down Mode (I_{CC} Max)
80C51	CMOS	4–6 V	12 MHz Max	24 mA	3 mA	50 μA
			1.2 MHz Min	2.4 mA	0.3 mA	50 μA
8051	HMOS	4.5–5.5 V	12 MHz Max	150 mA	—	20 mA
			1.2 MHz Min	130 mA	—	20 mA

Table 1: *A comparison of the CMOS and NMOS versions of the 8051.*

The CMOS version of the 8051, called the 80C51, is targeted at a number of applications that require both high performance and low power consumption. In areas like telephony, automotive control, industrial control, and portable instrumentation, the 80C51 operates at or near its maximum speed, even if only for short intervals. (For example, most real-time applications need an external-interrupt response time of less than 100 microseconds (μs); more demanding applications require better than 10-μs response. While the response must be quick, and the interrupt routine executed quickly; the processor spends a significant portion of its time idle.)

Once the performance requirements of the application are known, it is possible to specify a minimum operating frequency. For the 80C51, a hybrid static/dynamic design was proposed that allows a minimum die size and includes various modes of operation to minimize power consumption.

First, the only areas of the design that were made dynamic were the (very large) ROM and Control arrays. These arrays contain almost 50,000 transistors and constitute a major portion of the die. By making them dynamic, an area savings on the order of 40 to 50 percent was accomplished.

Second, the processor, all the peripheral functions, the RAM, and the I/O ports were made static. This allowed two modes of operation other than normal operating mode: *Idle* mode and *Power Down* mode.

Because in many applications the processor does nothing more than wait for an event to happen, in the Idle mode the major clocks of the device are stopped and only smaller ancillary clocks operate to drive the peripheral counter/timers, external-interrupt control, and the serial channel. When one of the peripherals generates an interrupt, the processor clocks are restarted and instruction execution resumes in the interrupt-service routine. The Idle mode reduces power consumption by almost an order of magnitude.

In Power Down mode, *all* the clocks inside the device are shut off and only the internal 128 bytes of RAM are "kept alive." The only current consumed is a minute amount due to pn-junction leakage. Static logic was designed in the peripheral sections in order to support this mode because no clocks are available to refresh dynamic logic. In both the Idle and Power Down modes, special provisions are made for the dynamic circuits in the ROM and Control areas to enter a pseudostatic condition that prevents any extraneous power consumption due to voltage drift on capacitive storage nodes.

Table 1 compares the NMOS 8051 to the CMOS 80C51. The 80C51, designed in Intel's HMOS-derived n-well process called CHMOS, is less than 10 percent larger than the NMOS design and consumes only 15 percent of the normal operating power. More significant power savings are possible by operating the 80C51 at lower frequencies or by using the Idle mode.

Martin B. Pawloski (5000 West Williams Field Rd., Chandler, AZ 85224) is involved in the product planning, definition, and implementation of both NMOS and CMOS single-chip microcomputers at Intel Corp.

A Look at CMOS Dynamic Memory

by Joe Altnether

The fast-growing portable-computer market is placing severe demands on semiconductor memory. For optimum system performance, these components must limit their power dissipation to suit battery operation and backup, and they must achieve the high data bandwidths and increased speeds needed for fast processing and high-resolution graphics. As the market reaches a projected $4.8 billion level by 1987 (a tenfold increase over 1982 levels), these requirements will combine to fuel the use of high-performance CMOS dynamic RAMs.

One architecture that can increase the speed of a CMOS dynamic RAM incorporates static-column address decoders: static circuits perform the selection of the column address of the RAM. Previously, this architecture has not been used with dynamic RAMs because of the increased power consumption of the static circuits over that of the dynamic circuits, and the advantage of low power consumption would have been lost. But with CMOS, the increased power consumption is negligible.

Memory-device Architecture

RAMs are organized internally as rows and columns of storage cells. Data access occurs at the intersection of a row address and a column address. In dynamic RAMs, the row and column addresses are multiplexed to reduce package size and pin count: the row addresses are clocked into the device with the RAS (row address strobe) signal, causing one row of data (1 bit from each of the 256 columns in a 64K-bit dynamic RAM) to be fed into the 256 internal sense amplifiers. (Because of the low internal signal levels, each column must have an associated sense amplifier to sense and restore memory-cell data.) Next, column addresses are presented to the device and clocked into it with the CAS (column address strobe) signal. These column addresses are then decoded to select one of the 256 bits. Faster access and cycle times are obtained within a row (or "page") after the first access to it because the 256 bits within the row continue to reside in the sense amplifiers and need not be refetched. Reapplying only column addresses, then, in what is known as *Page Mode* operation, provides fast serial accesses and can increase cycle times by a factor of 2.

The CMOS dynamic RAM can incorporate static-column circuits to provide performance equivalent to that of high-speed static RAMs. With CMOS, the static-decoding circuits reduce the internal number of clocks by a factor of 3, eliminating the need to allow for setup and hold times of signals with respect to clocks and the need to compensate for timing skews due to process variances. With static-column circuits, precharge times are drastically reduced (in Page Mode operation of the Static-Column-mode device, precharge time is reduced from 30 nanoseconds [ns] to 5 ns). This precharge time reduction and the faster access times typically increase the memory's bandwidth to 20 MHz. (Performance of memory discussed here is based on the experimental 64K-bit CMOS dynamic RAM that Intel presented at the ISSCC conference in February 1983.)

With static-column architecture, two different types of Page Mode operation are possible: Static Column mode and Ripplemode. Static Column mode uses the RAS line and row addresses in the conventional manner, but once the row has been selected, data can be accessed merely by changing column addresses. As with a static RAM, column addresses must remain stable and valid for the entire address access cycle. Access time is measured from column addresses rather than the occurrence of CAS. (Typically, access from column addresses is 30 ns; from CAS, it is 10 ns.)

In operation, CAS is used to place the output in a high-impedance state or to activate an output buffer. CAS can be held active during the entire page cycle. In fact, it is possible to keep CAS permanently active (i.e., grounded). During a write cycle, however, addresses as well as data are latched by CAS or WE, whichever occurs last. Operation is identical to that of an NMOS dynamic RAM in this case. This action ensures that the data is written into the proper memory location.

Although Static Column mode provides fast, easy accesses, speed at the system level is limited by how fast addresses for the next cycle become valid; the time to generate and stabilize the addresses must be added to the cycle time. Increased system speed can be obtained by using Ripplemode. With this mode, static-column circuits are again used to obtain access from valid column addresses, but the addresses are latched on the falling edge of CAS, removing the requirement for addresses to remain valid throughout the entire cycle. As a result, during the current cycle addresses for the next cycle can be set up or *pipelined.*

Column addresses enter the RAM through the internal address latch. This latch, controlled by CAS, provides flow-through operation. When CAS is inactive, the latch is open, and addresses pass through continuously to the static-column decoders. Any change in address is transmitted immediately to the decoder. Consequently, access to the RAM is again measured from valid column addresses. The latch captures the current address on the fall of CAS, permitting the system address to change while the access occurs. CAS also serves as an output enable on the data output. Static Column mode and Ripplemode both permit continuous data streams up to 20 MHz.

CMOS technology and static-column architecture provide more than low power consumption and high bandwidth. In addition, static-column decoding simplifies system design by eliminating critical timing relationships while providing higher system speed. Access from column addresses gives usable speed for single random accesses within the RAM. Also, the CMOS technology enhances reliability by incorporating a mechanism to significantly reduce soft errors. Finally, increased stored charge creates larger internal signal levels, which can more easily be differentiated from noise. As a result, the CMOS dynamic RAM has wider operating margins and system reliability is improved.

Power Consumption

At the system level, dynamic memory has three components of power: active, standby, and refresh. The system's power consumption is defined as

$$P=V(MI_A+KI_S+NI_R)$$

where P = system power, V = voltage (5.5 V worst case), I_A = active current, I_A = standby current, I_R = refresh current, M = number of active devices, K = number of devices in standby, and N = total number of devices.

CMOS reduces the first term, the active current, relative to NMOS by a factor of 2. In addition, the lower active current reduces supply voltage transients, thus simplifying printed-circuit-board design and reducing decoupling-capacitor requirements.

The second term, standby current, is also reduced by a factor of 2 at TTL input levels. Driving the RAS signal to a CMOS level ($V_{DD}-0.5$ V) places the device in a low-power-standby mode and typically draws 10 microamperes (μA)—a factor of 50 reduction over NMOS!

Refresh current, the third term in the equation, is cycle-time dependent. Current increases with the frequency of refresh. In dynamic RAMs, data is stored on a capacitor that must be replenished or recharged every 2 or 4 milliseconds (ms). This refresh time is a function of the stored charge and the leakage current. With the CMOS dynamic RAM, the cell storage capacitance is 0.125 picofarad (pF) compared to 0.040 pF to 0.085 pF in an NMOS dynamic RAM. This low capacitance, coupled with lower leakage currents, permits the CMOS refresh period to be extended to 64 ms in standby.

At the standard 128 refresh cycles/2 ms (equivalent to a 15.625-μs refresh period), the NMOS device draws about 4.8 milliamperes (mA) and asymptotically approaches the standby current of 4 mA as the refresh period approaches infinity. Even eliminating refresh entirely only reduces the current to 4 mA, which is only a 16 percent improvement. As a result, extending NMOS refresh does not significantly reduce the system's power consumption.

Contrast this characteristic to the improvement CMOS offers. At 15.625 μs, the CMOS dynamic RAM draws approximately 10 percent of the NMOS current, or 0.42 mA at TTL levels. Extending the refresh period reduces the current asymptotically to the standby current of 0.05 mA. At a 64-ms refresh period, the current is reduced to 0.15 mA, a 300 percent reduction. When battery powered, the CMOS system has a 10 times longer life than does the NMOS system, and an extended refresh mode offers another fivefold improvement. A 256K-byte CMOS memory can retain data for nearly one week on only AA nickel-cadmium (nicad) cells—more than sufficient for most portable systems.

High-Speed Applications

Ripplemode and Static Column mode are ideal for applications involving high-speed buffers, telecommunications, and graphics. Bit-mapped graphics systems would seem to be a natural fit with Page Mode operation. However, this was not always the case. Prior to the Intel 2164A 64K by 1-bit NMOS dynamic RAM, it was difficult to retrieve all 256 bits within a single row of memory because of the RAS-low time limitation of 10 μs. Even with a Page Mode cycle time of 125 ns, to retrieve all 256 bits would require 32 μs—three times longer than allowed. The 2164A extended the RAS-low time to 75 μs, permitting the extraction of all 256 bits during a single Page Mode cycle.

At the end of the cycle, the device cannot be reaccessed again until after a certain off-time allows internal nodes to be precharged to be ready for the next cycle. As a result, the 2164A can stream data at greater than a 7-MHz rate continuously. This function matches the timing and operation of low-performance, bit-mapped graphics memories. One 2164A, for example, can map all the data for the 256 by 256 matrix of a graphics display. During the horizontal scan time, the RAM performs a Page Mode cycle and one full line is displayed. During retrace time, the memory must be refreshed and can be updated with new data if required. This type of update is relatively slow; consequently, it limits the speed of animation on the screen because the processor has access to the memory only 25 percent of the time.

To increase resolution, more lines, each with more pixels, must be used. By performing two sequential Page Mode cycles from two different RAMs, pixel densities to 512 bits per line can be achieved. As pixel density increases, the memory cycle time must decrease to paint more pixels on a line in the same amount of time. This cycle-time limitation plus the fact that memory can be updated only during blanking has precluded dynamic RAMs from use in higher-resolution graphics displays. These systems are usually built with high-speed, expensive static RAMs.

With Ripplemode, memory update during screen display time, also known as cycle stealing, is possible. As an example, a 512 by 384 display requires 512 bits/line and 1 bit every 67 ns. Data is read from four memory devices in a series of eight Ripplemode reads each. Data is temporarily stored in a video-output register file and then shifted to the video screen at a rate slower than the Ripplemode reads. Following this, enough time is available to perform an update cycle before the next eight Ripplemode reads are performed to continue screen refresh. Eight was the number chosen to minimize the time the processor must wait to update the memory. In addition to this cycle stealing, which updates during display time, memory updates are also performed during blanking. Along with this system, a similar system was built using 2164As with Extended Page Mode operation. Each system used an iAPX 86 processor and similar software. A comparison of both systems showed the CHMOS (complementary high-speed metal-oxide semiconductor) system to have a 42 percent higher drawing speed. Animation on the CHMOS system was vastly improved.

Usable Speed

Memory design using dynamic RAMs has always been a challenge. Although multiplexing addresses does reduce the package pin count and increase system density, it limits the access and cycle times in the system. To access a dynamic RAM, low-order row addresses are presented and latched into the dynamic RAM with RAS. Row addresses must be held for a period t_{RAH} after the fall of RAS to guarantee proper operation. Next, the addresses must be changed to high-order column addresses and latched into the dynamic RAM with CAS, creating a timing window t_{RCD}, which is the RAS-to-CAS delay.

Within this window, the designer must guarantee row address hold time, change the addresses, and account for any timing skew on the CAS signal. If column addresses are valid at the maximum specified t_{RCD}, access time t_{RAC} is measured from the high-to-low transition of RAS.

The cycle time is the sum of the access time and the cycle precharge time t_{RP}. The access time is a function of t_{RCD}, which has contradictory requirements. It must be as long as possible to simplify system design and at the same time as short as possible to enhance system speed. Cycle time is affected directly by the length of t_{RP}.

Static-column operation eliminates the t_{RCD} problem. After row addresses have been latched into the RAM, the second portion of the access begins from valid column addresses. In other words, column access does not wait for CAS to become valid, but operates in a fashion similar to that of a static RAM. This is due to the flow-through operation of the CAS latch. CAS serves only to latch the addresses and to provide an output enable. Access from valid column addresses simplifies design by removing the CAS signal from the critical timing path.

Systems using dynamic RAMs are typically CAS access-limited because controllers generate timing signals in discrete clock increments. A CMOS dynamic RAM system might operate at 8 MHz without Wait states. Using any other 64K-bit dynamic RAM would require the injection of one or two Wait states, resulting in a corresponding performance penalty. Consequently, the advantage of higher processor speed is negated without the high-speed dynamic RAM. For systems incorporating either discrete or LSI controllers, the CMOS dynamic RAM simplifies the system design and offers higher system performance.

High Reliability

Soft errors are random, nonrecurring failures caused by ionizing radiation present within the environment. All matter contains small amounts of radioactive material. Alpha particles emitted by an IC's packaging material can penetrate the enclosed circuit. As they do so, they generate hole-electron pairs. Any high-impedance node in the vicinity sensitive to 1 million electrons may be affected, because the difference between a 1 and a 0 (known as the critical charge) is about 1 million electrons. Consequently, data in one cell could change from a 1 to a 0 or vice versa. Correct data can be rewritten into the affected cell and the memory will again function correctly, thus the term "soft error."

When first discovered during tests of 16K-bit dynamic RAMs, soft errors occurred at a rate five times greater than catastrophic or hard-error failures. While device designers worked to eliminate the alpha-particle sensitivity, systems designers added error-correcting circuits (ECC), which increased system reliability, but the systems were larger and more expensive due to the additional components required. Also, the system had to test and correct the data, slowing the system's performance. All this was due to soft errors. Obviously, what is really required is the elimination of soft errors.

CMOS technology offers such a solution. The CMOS dynamic RAM cell is built on an n-well in a p-substrate, creating a p-n junction or diode at the boundary. When alpha particles create hole-electron pairs in a CMOS device, something else occurs. First, the n-well is very shallow, and the majority of hole-electron pairs are created in the p-substrate. Holes cannot transfer across the reverse-biased p-n junction, which acts as a barrier to soft-error effects. Any electrons that do cross the junction are gathered at the +5-V node away from the storage cell. The probability that sufficient hole-electron pairs are created within the n-well that cell upset could occur is so low that the soft-error rate of CMOS dynamic RAMs is typically orders of magnitude below that of their NMOS counterparts.

High storage capacitance also plays a role in the reduction of soft errors. The number of stored charged electrons representing a 1 or a 0 is directly proportional to the storage capacitance. Higher capacitance equates to more stored charge, which in turn increases the critical charge. The critical charge is the number of particles that differentiate a 1 from a 0. Increasing the critical charge beyond 1 million electrons significantly reduces the susceptibility to soft errors. This, in addition to the n-well mechanism, reduces the soft-error rate to much less than 0.001 percent per 1000 hours.

Studies were performed to compare reliability of systems with and without error correction for both NMOS and CMOS dynamic RAMs. The results show one surprise: at 256K bytes and below, the CMOS system *without* ECC is more reliable than the NMOS system *with* ECC, because of the cycle-time dependence of soft errors. In small systems, the memory is accessed more frequently, and the probability of a soft error is increased. With a soft-error rate at the very minimum 100 times less than NMOS, the CMOS dynamic RAM does not experience this effect.

Systems below 256K-byte capacities benefit by the elimination of ECC circuits from a cost, performance, and simplicity-of-design standpoint. First, ECC increases the access time of the system by 50 ns to check and correct data. Assuming a 120-ns RAM access, ECC increases the access by 42 percent. Moreover, the penalty on cycle time is even greater, especially when you are writing a single byte into a 2-byte word. In this instance, data must be accessed and corrected, the new byte merged into the word, and check bits generated. Finally, the system must write the new data into memory. Added to this are any system-timing skews. As a result, a 200-ns cycle time stretches to a 335-ns system cycle time or an increase of 68 percent. Therefore, using a CMOS dynamic RAM not only improves system reliability but enhances system speed and simplicity of design.■

Joe Altnether is technical marketing manager at Intel Corp. (2111 N.E. 25th Ave., Hillsboro, OR 97123).

From *Inside CMOS Technology* by Martin B. Pawloski, Tomy Moroyan, and Joe Altnether appearing in the September 1983 issue of BYTE magazine. Copyright © 1983 Byte Publications, Inc. Used with the permission of Byte Publications, Inc.

The Single Component MCS®-48 System

13

CHAPTER 13
THE SINGLE COMPONENT MCS®-48 SYSTEM

13.0 INTRODUCTION

Sections 13.1 through 13.4 describe in detail the functional characteristics of the 8748H and 8749H EPROM, 8048AH/8049AH/8050AH ROM, and 8035AHL/8039AHL/8040-AHL CPU only single component microcomputers. Unless otherwise noted, details within these sections apply to all versions. This chapter is limited to those functions useful in single-chip implementations of the MCS®-48.Chapter 14 discusses functions which allow expansion of program memory, data memory, and input output capability.

13.1 ARCHITECTURE

The following sections break the MCS-48 Family into functional blocks and describe each in detail. The following description will use the 8048AH as the representative product for the family. See Figure 15.1.

13.1.1 Arithmetic Section

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

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

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

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

INSTRUCTION DECODER

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

ARITHMETIC LOGIC UNIT

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

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

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

ACCUMULATOR

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

13.1.2 Program Memory

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

There are three locations in Program Memory of special importance as shown in Figure 13.2.

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

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

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

Therefore, the first instruction to be executed after initialization is stored in location 0, the first word of an external interrupt service subroutine is stored in location 3, and the first word of a timer/counter service routines

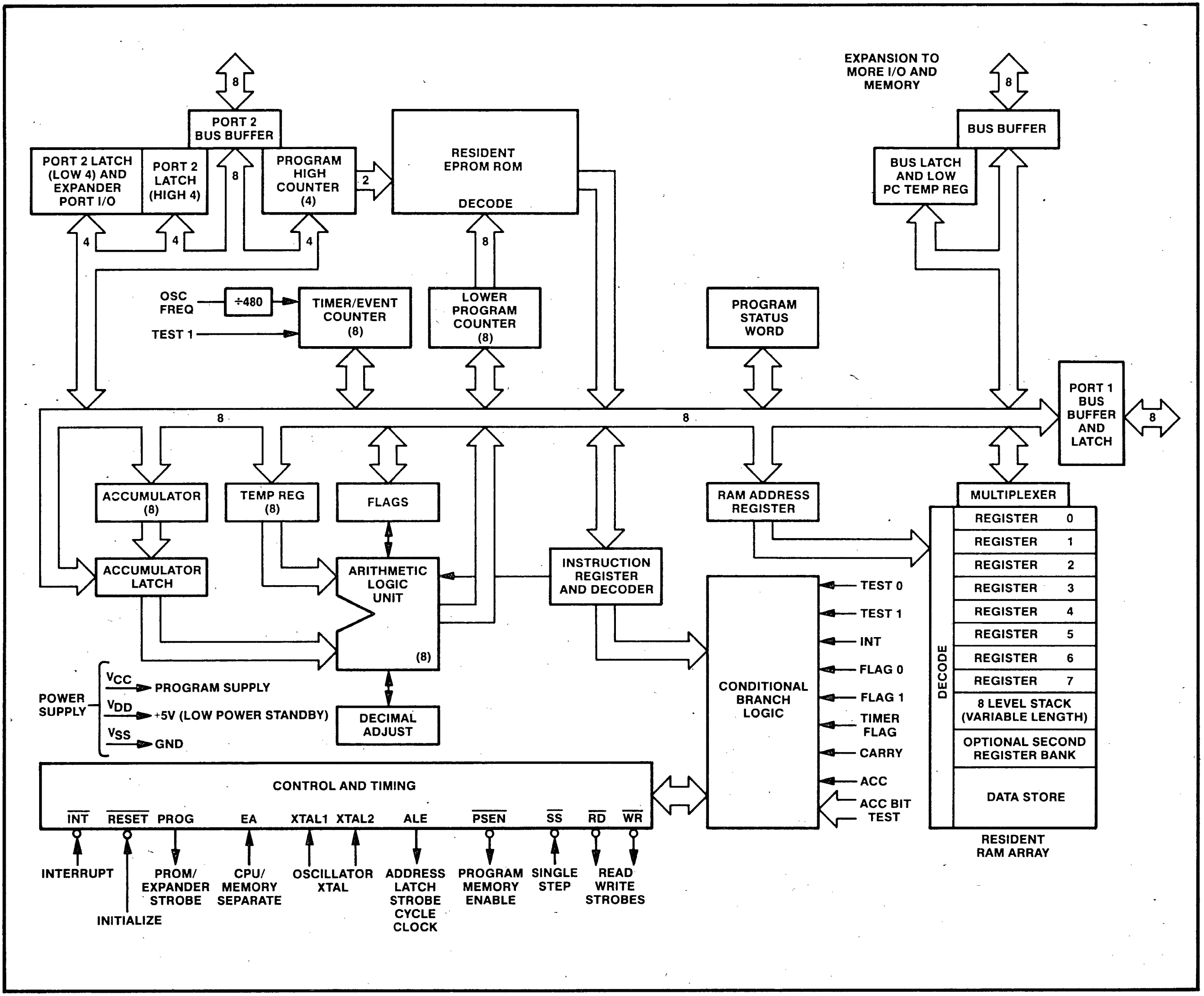


Figure 13-1. 8748H/8048AH/8749H/8049AH/8050AH Block Diagram

is stored in location 7. Program memory can be used to store constants as well as program instructions. Instructions such as MOVP and MOVP3 allow easy access to data ''lookup'' tables.

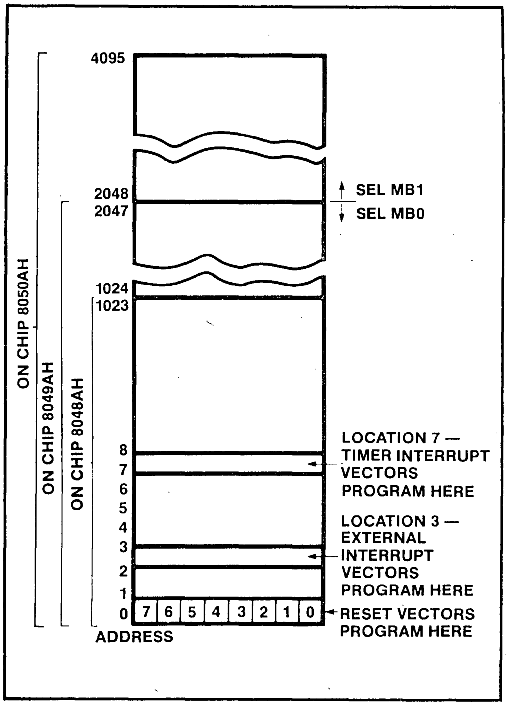

4095
2048
2047
1024
1023
8
7
6
5
4
3
2
1
0
7 6 5 4 3 2 1 0
ADDRESS
SEL MB1
SEL MB0
LOCATION 7 — TIMER INTERRUPT VECTORS PROGRAM HERE
LOCATION 3 — EXTERNAL INTERRUPT VECTORS PROGRAM HERE
RESET VECTORS PROGRAM HERE
ON CHIP 8050AH
ON CHIP 8049AH
ON CHIP 8048AH

Figure 13-2. Program Memory Map

13.1.3 Data Memory

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

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

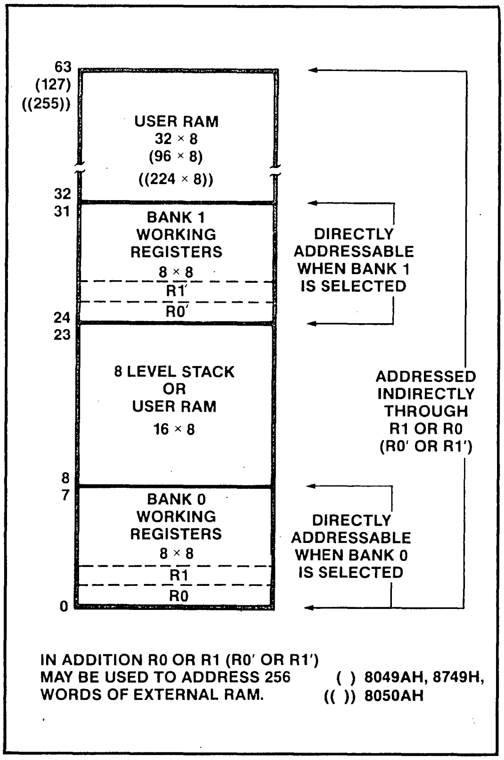
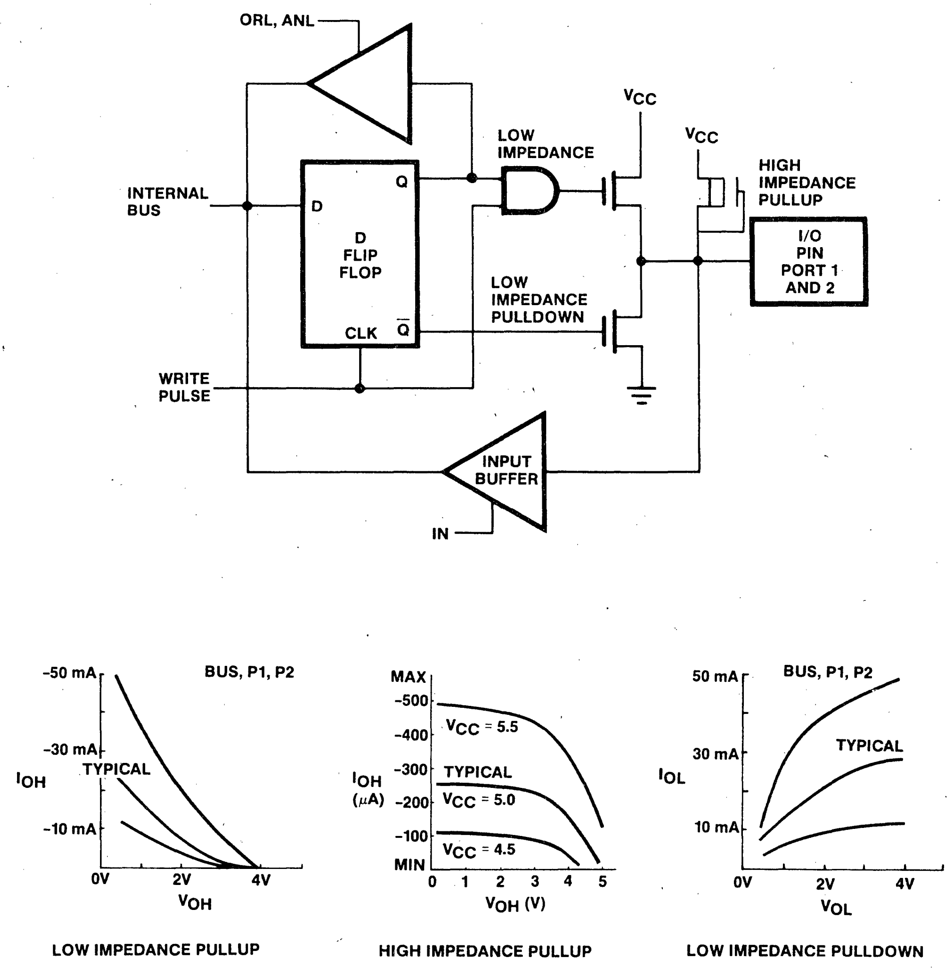

63
(127)
((255))
USER RAM
32 × 8
(96 × 8)
((224 × 8))
32
31
BANK 1
WORKING
REGISTERS
8 × 8
R1′
R0′
24
23
8 LEVEL STACK
OR
USER RAM
16 × 8
8
7
BANK 0
WORKING
REGISTERS
8 × 8
R1
R0
0
DIRECTLY ADDRESSABLE WHEN BANK 1 IS SELECTED
ADDRESSED INDIRECTLY THROUGH R1 OR R0 (R0′ OR R1′)
DIRECTLY ADDRESSABLE WHEN BANK 0 IS SELECTED
IN ADDITION R0 OR R1 (R0′ OR R1′) MAY BE USED TO ADDRESS 256 WORDS OF EXTERNAL RAM.
( ) 8049AH, 8749H,
(( )) 8050AH

Figure 13-3. Data Memory Map



These graphs are for informational purposes only and are not guaranteed minimums or maximums.

Figure 13-4. "Quasi-bidirectional" Port Structure

13.1.4 Input/Output

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

PORTS 1 AND 2

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

The lines of ports 1 and 2 are called quasi-bidirectional because of a special output circuit structure which allows each line to serve as an input, and output, or both even though outputs are statically latched. Figure 13-4 shows the circuit configuration in detail. Each line is continuously pulled up to V_{CC} through a resistive device of relatively high impedance.

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

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

BUS

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

As a static port, data is written and latched using the OUTL instruction and inputted using the INS instruction. The INS and OUTL instructions generate pulses on the corresponding $\overline{\text{RD}}$ and $\overline{\text{WR}}$ output strobe lines; however, in the static port mode they are generally not used. As a bidirectional port the MOVX instructions are used to read and write the port. A write to the port generates a pulse on the $\overline{\text{WR}}$ output line and output data is valid at the trailing edge of $\overline{\text{WR}}$. A read of the port generates a pulse on the $\overline{\text{RD}}$ output line and input data must be valid at the trailing edge of $\overline{\text{RD}}$. When not being written or read, the BUS lines are in a high impedance state. See also sections 14.6 and 14.7.

13.1.5 Test and INT Inputs

Three pins serve as inputs and are testable with the conditional jump instruction. These are T0, T1, and $\overline{\text{INT}}$. These pins allow inputs to cause program branches without the necessity to load an input port into the accumulator. The T0, T1, and $\overline{\text{INT}}$ pins have other possible functions as well. See the pin description in Section 13.2.

13.1.6 Program Counter and Stack

The Program Counter is an independent counter while the Program Counter Stack is implemented using pairs of registers in the Data Memory Array. Only 10, 11, or 12 bits of the Program Counter are used to address the 1024, 2048, or 4096 words of on-board program memory of the 8048AH, 8049AH, or 8050AH, while the most significant bits can be used for external Program Memory fetches. See Figure 13.5. The Program Counter is initialized to zero by activating the Reset line.

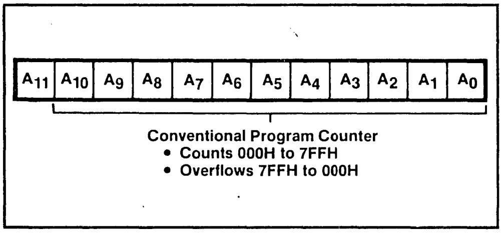

A11 A10 A9 A8 A7 A6 A5 A4 A3 A2 A1 A0
Conventional Program Counter
• Counts 000H to 7FFH
• Overflows 7FFH to 000H

Figure 13-5. Program Counter

An interrupt or CALL to a subroutine causes the contents of the program counter to be stored in one of the 8 register pairs of the Program Counter Stack as shown in Figure 13-6. The pair to be used is determined by a 3-bit Stack Pointer which is part of the Program Status Word (PSW).

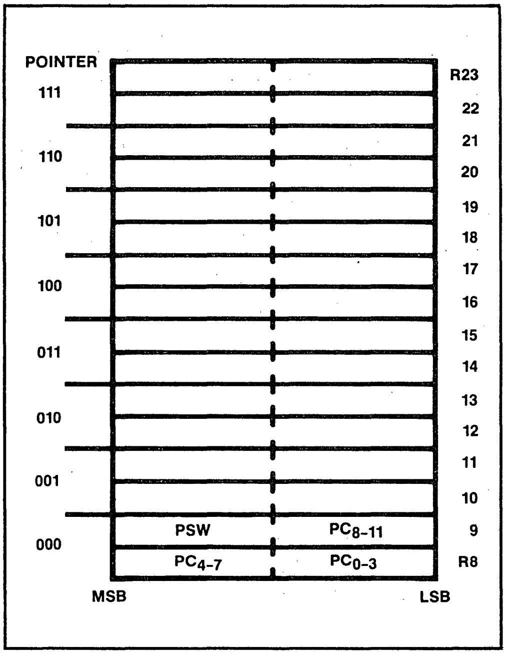


Figure 13-6. Program Counter Stack

Data RAM locations 8-23 are available as stack registers and are used to store the Program Counter and 4 bits of PSW as shown in Figure 13-6. The Stack Pointer when initialized to 000 points to RAM locations 8 and 9. The first subroutine jump or interrupt results in the program counter contents being transfeṙred to locations 8 and 9 of the RAM array. The stack pointer is then incremented by one to point to locations 10 and 11 in anticipation of another CALL. Nesting of subroutines within subroutines can continue up to 8 times without overflowing the stack. If overflow does occur the deepest address stored (locations 8 and 9) will be overwritten and lost since the stack pointer overflows from 111 to 000. It also underflows from 000 to 111.

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

13.1.7 Program Status Word

An 8-bit status word which can be loaded to and from the accumulator exists called the Program Status Word (PSW). Figure 13-7 shows the information available in the word. The Program Status Word is actually a collection of flip-flops throughout the machine which can be read or written as a whole. The ability to write to PSW allows for easy restoration of machine status after a power down sequence.

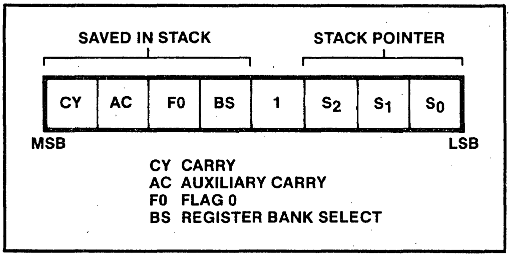


Figure 13-7. Program Status Word (PSW)

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

The PSW bit definitions are as follows:

Bits 0–2:	Stack Pointer bits (S_0, S_1, S_2)
Bit 3:	Not used ("1" level when read)
Bit 4:	Working Register Bank Switch Bit (BS) 0 = Bank 0 1 = Bank 1
Bit 5:	Flag 0 bit (F0) user controlled flag which can be complemented or cleared, and tested with the conditional jump instruction JF0.
Bit 6:	Auxiliary Carry (AC) carry bit generated by an ADD instruction and used by the decimal adjust instruction DA A.
Bit 7:	Carry (CY) carry flag which indicates that the previous operation has resulted in overflow of the accumulator.

13.1.8 Conditional Branch Logic

The conditional branch logic within the processor enables several conditions internal and external to the processor to be tested by the users program. By using the conditional jump instruction the conditions that are listed in Table 13-1 can effect a change in the sequence of the program execution.

Table 13-1

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

13.1.9 Interrupt

An interrupt sequence is initiated by applying a low "0" level input to the INT pin. Interrupt is level triggered and active low to allow "WIRE ORing" of several interrupt sources at the input pin. Figure 13-8 shows the interrupt logic of the 8048AH. The Interrupt line is sampled every instruction cycle and when detected causes a "call to subroutine" at location 3 in program memory as soon as all cycles of the current instruction are complete. On 2-cycle instructions the interrupt line is sampled on the 2nd cycle only. $\overline{\text{INT}}$ must be held low for at least 3 machine cycles to ensure proper interrupt operations. as in any CALL to subroutine, the Program Counter and Program Status word are saved in the stack. For a description of this operation see the previous section, Program Counter and Stack. Program Memory location 3 usually contains an unconditional jump to an interrupt service subroutine elsewhere in program memory. The end of an interrupt service subroutine is signalled by the execution of a Return and Restore Status instruction RETR. The interrupt system is single level in that once an interrupt is detected all further interrupt requests are ignored until execution of an RETR reenables the interrupt input logic. This occurs at the beginning of the second cycle of the RETR instruction. This sequence holds true also for an internal interrupt generated by timer overflow. If an internal timer/counter generated interrupt and an external interrupt are detected at the same time, the external source will be recognized. See the following Timer/Counter section for a description of timer interrupt. If needed, a second external interrupt can be created by enabling the timer/counter interrupt, loading FFH in the Counter (ones less than terminal count), and enabling the event counter mode. A "1" to "0" transition on the T1 input will then cause an interrupt vector to location 7.

INTERRUPT TIMING

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

13.1.10 Timer/Counter

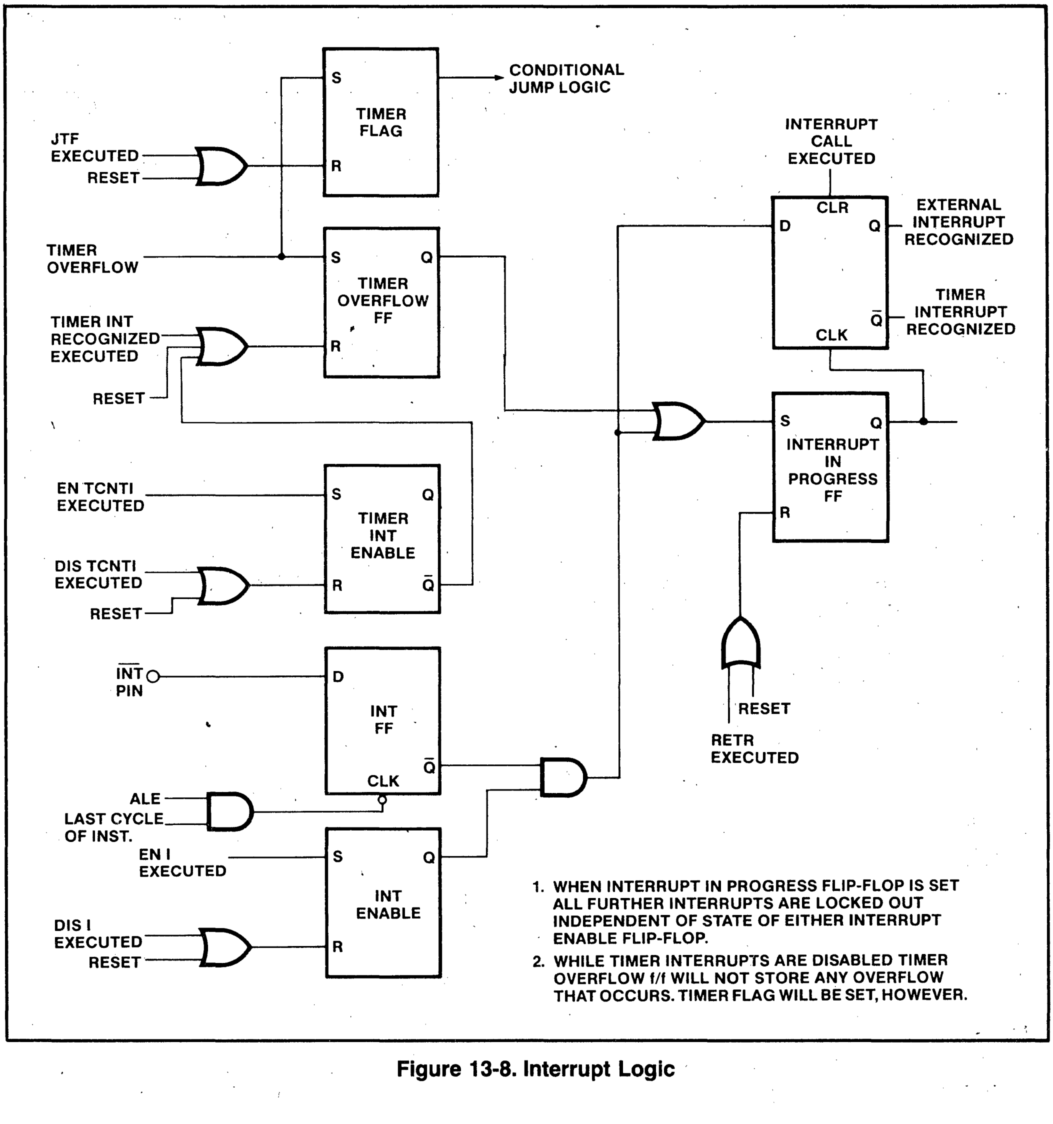
The 8048AH contains a counter to aid the user in counting external events and generating accurate time delays without placing a burden on the processor for these functions. In both modes the counter operation is the same, the only difference being the source of the input to the counter. The timer/event counter is shown in Figure 13-9.

COUNTER

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

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

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



1. WHEN INTERRUPT IN PROGRESS FLIP-FLOP IS SET ALL FURTHER INTERRUPTS ARE LOCKED OUT INDEPENDENT OF STATE OF EITHER INTERRUPT ENABLE FLIP-FLOP.
2. WHILE TIMER INTERRUPTS ARE DISABLED TIMER OVERFLOW f/f WILL NOT STORE ANY OVERFLOW THAT OCCURS. TIMER FLAG WILL BE SET, HOWEVER.

Figure 13-8. Interrupt Logic

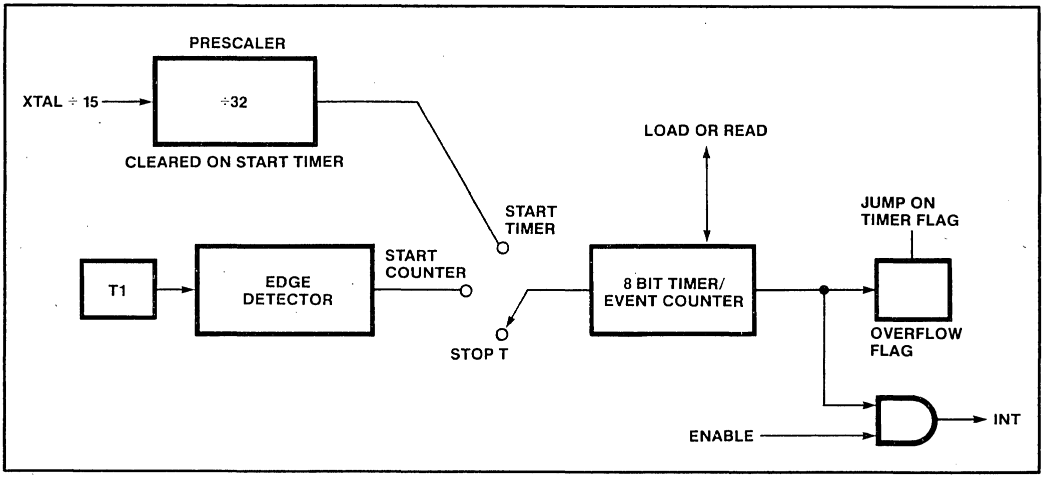


Figure 13-9. Timer/Event Counter

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

AS AN EVENT COUNTER

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

AS A TIMER

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

Often a serial link is desirable in an MCS-48 family member. Table 13-2 lists the timer counts and cycles needed for a specific baud rate given a crystal frequency.

13.1.11 Clock and Timing Circuits

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

OSCILLATOR

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

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

Table 13-2. Baud Rate Generation

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

STATE COUNTER

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

CYCLE COUNTER

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

13.1.12 Reset

The reset input provides a means for initialization for the processor. This Schmitt-trigger input has an internal pull-up device which in combination with an external 1 µ fd capacitor provides an internal reset pulse of sufficient length to guarantee all circuitry is reset, as shown in Figure 13-12. If the reset pulse is generated externally the RESET pin must be held low for at least 10 milliseconds after the power supply is within tolerance. Only 5 machine cycles (6.8 µs @ 11 MHz) are required if power is already on and the oscillator has stabilized. ALE and PSEN (if EA = 1) are active while in Reset.

Reset performs the following functions:

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

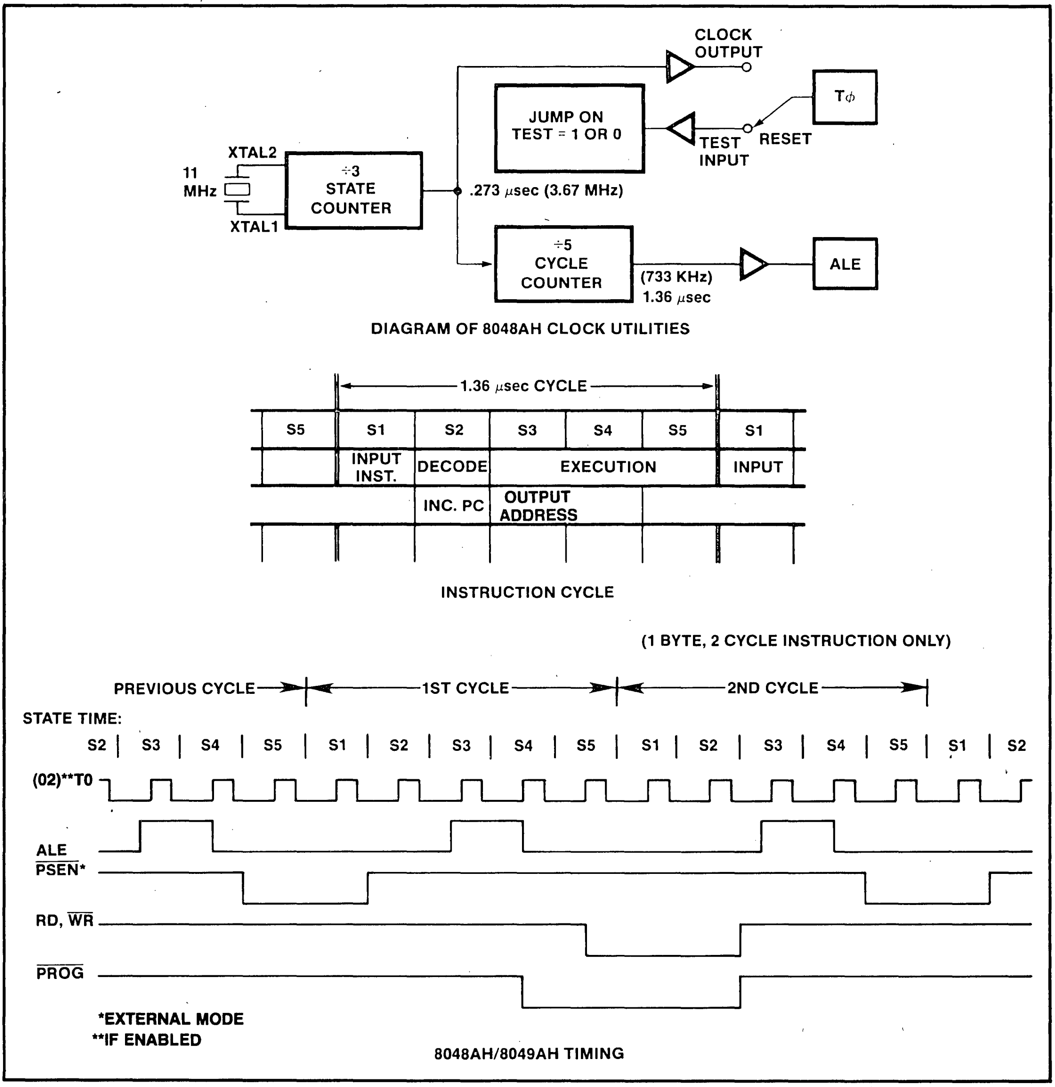


Figure 13-10. MCS®-48 Timing Generation and Cycle Timing

13.1.13 Single-Step

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

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

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

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

Figure 13-11. 8048AH/8049AH Instruction Timing Diagram

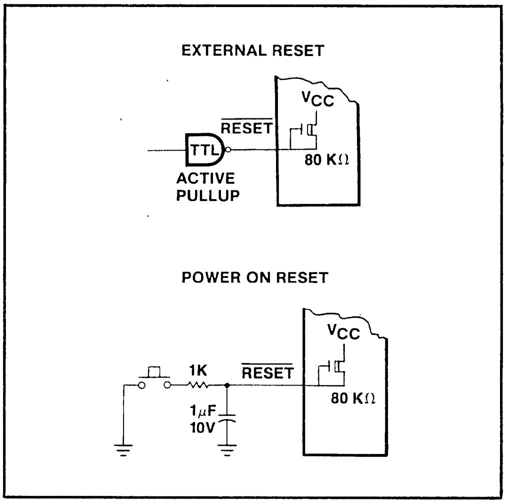


Figure 13-12

TIMING

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

1) The processor is requested to stop by applying a low level on $\overline{SS}$.
2) The processor responds by stopping during the address fetch portion of the next instruction. If a double cycle instruction is in progress when the single step command is received, both cycles will be completed before stopping.
3) The processor acknowledges it has entered the stopped state by raising ALE high. In this state (which can be maintained indefinitely) the address of the next instruction to be fetched is present on BUS and the lower half of port 2.
4) $\overline{SS}$ is then raised high to bring the processor out of the stopped mode allowing it to fetch the next instruction. The exit from stop is indicated by the processor bringing ALE low.
5) To stop the processor at the next instruction $\overline{SS}$ must be brought low again soon after ALE goes low. If $\overline{SS}$ is left high the processor remains in a "Run" mode.

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

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

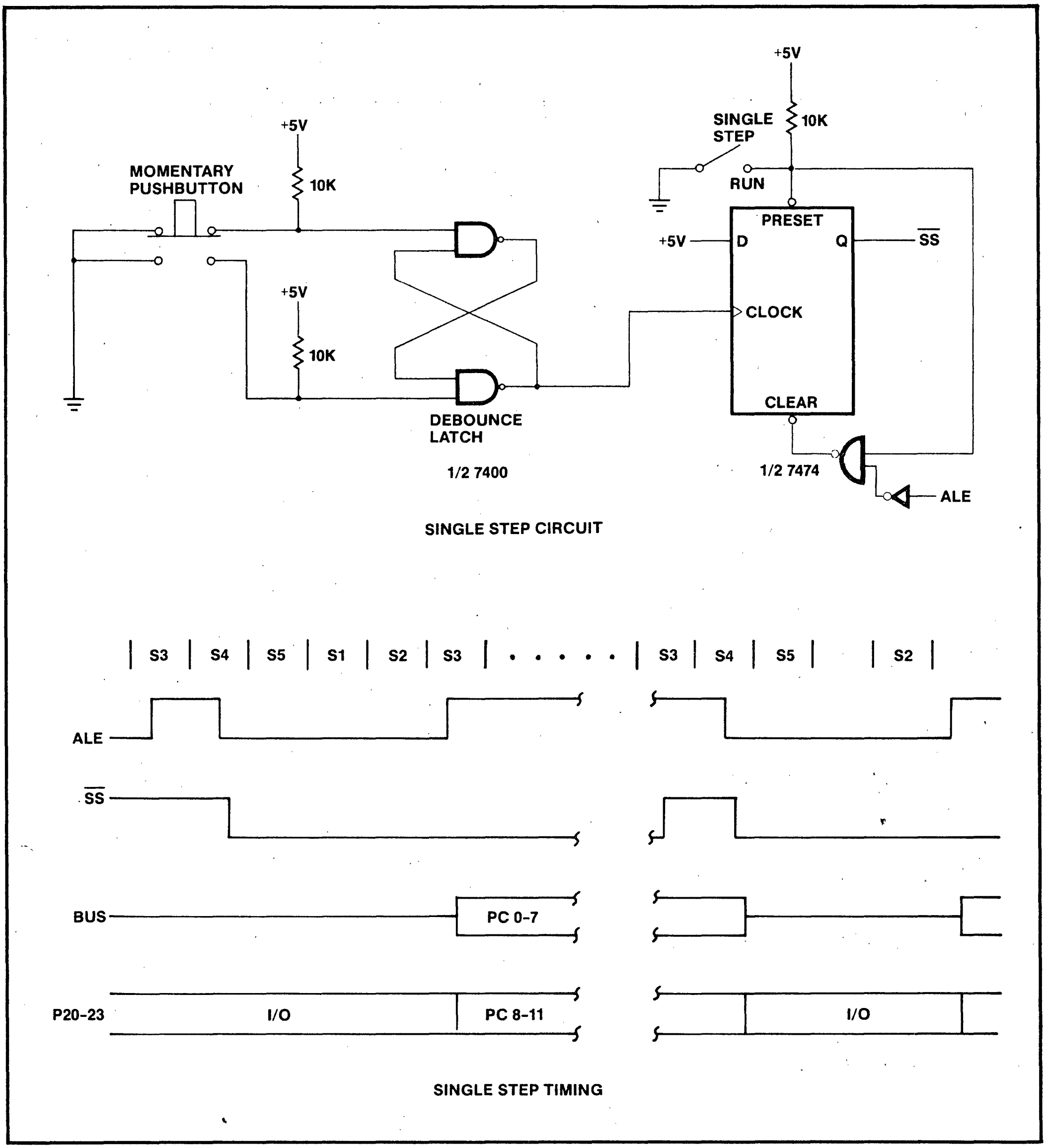
Extra circuitry has been added to the 8048AH/8049AH/8050AH, ROM version to allow power to be removed from all but the data RAM array for low power standby operation. In the power down mode the contents of data RAM can be maintained while drawing typically 10% to 15% of normal operating power requirements.

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

A typical power down sequence (Figure 13-14) occurs as follows:

1) Imminent power supply failure is detected by user defined circuitry. Signal must be early enough to allow 8048AH to save all necessary data before V_{CC} falls below normal operating limits.
2) Power fail signal is used to interrupt processor and vector it to a power fail service routine.
3) Power fail routine saves all important data and machine status in the internal data RAM array. Routine may also initiate transfer of backup supply to the V_{DD} pin and indicate to external circuitry that power fail routine is complete.
4) Reset is applied to guarantee data will not be altered as the power supply falls out of limits. Reset must be held low until V_{CC} is at ground level.

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



SINGLE STEP CIRCUIT

SINGLE STEP TIMING

Figure 13-13. Single Step Operation

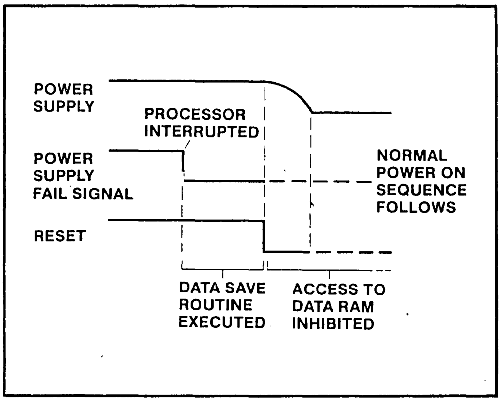

POWER SUPPLY
PROCESSOR INTERRUPTED
POWER SUPPLY FAIL SIGNAL
NORMAL POWER ON SEQUENCE FOLLOWS
RESET
DATA SAVE ROUTINE EXECUTED
ACCESS TO DATA RAM INHIBITED

Figure 13-14. Power Down Sequence

13.1.15 External Access Mode

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

The External Access mode is very useful in system test and debug because it allows the user to disable his internal applications program and substitute an external program of his choice—a diagnostic routine for instance. In addition, section 13.4 explains how internal program memory can be read externally, independent of the processor. A "1" level on EA initiates the external access mode. For proper operation, Reset should be applied while the EA input is changed.

13.1.16 Sync Mode

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

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

13.1.17 Idle Mode

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

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

Externally, the following occurs during idle:

1) The ports remain in the logical state they were in when idle was executed.

2) The bus remains in the logical state it was in when idle was executed if the bus was latched.

   If the bus was in a high Z condition or if external program memory is used the bus will remain in the float state.

3) ALE remains in the inactive state (low).

4) RD′, WR′, PROG′, and PSEN′ remains in the inactive state (high).

5) T0 outputs clock if enabled.

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

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

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

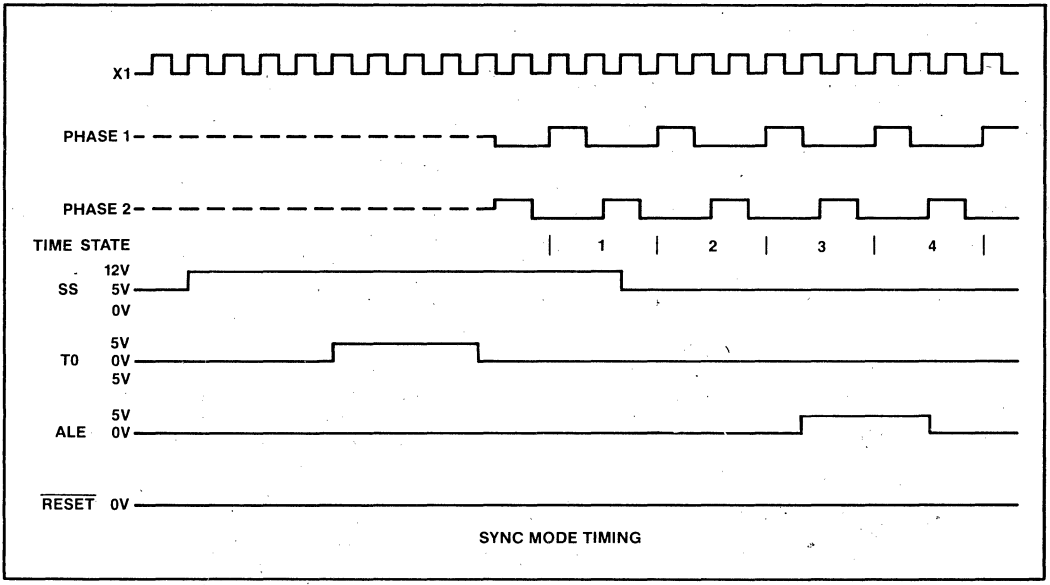


Figure 13-15. Sync Mode Timing

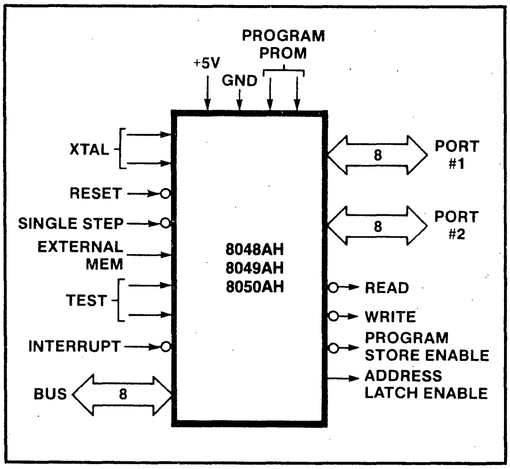


Figure 13-16. 8048AH and 8049AH Logic Symbol

2.2 PIN DESCRIPTION

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

Table 13-3. Pin Description

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

Table 13-3. Pin Description (Continued)

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

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

13.3 PROGRAMMING, VERIFYING AND ERASING EPROM

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

13.3.1 Programming/Verification

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

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

8748H AND 8749H ERASURE CHARACTERISTICS

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

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

The recommended erasure procedure for the 8748H and 8749H is exposure to shortwave ultraviolet light which has a wavelength of 2537 Angstroms (A). The integrated dose (i.e., UV intensity X exposure time) for erasure should be a minimum of 15W-sec/cm^2. The erasure time with this dosage is approximately 15 to 20 minutes using an ultraviolet lamp with a 12000μW/cm^2 power rating. The 8748H and 8749H should be placed within one inch from the lamp tubes during erasure. Some lamps have a filter in their tubes and this filter should be removed before erasure.

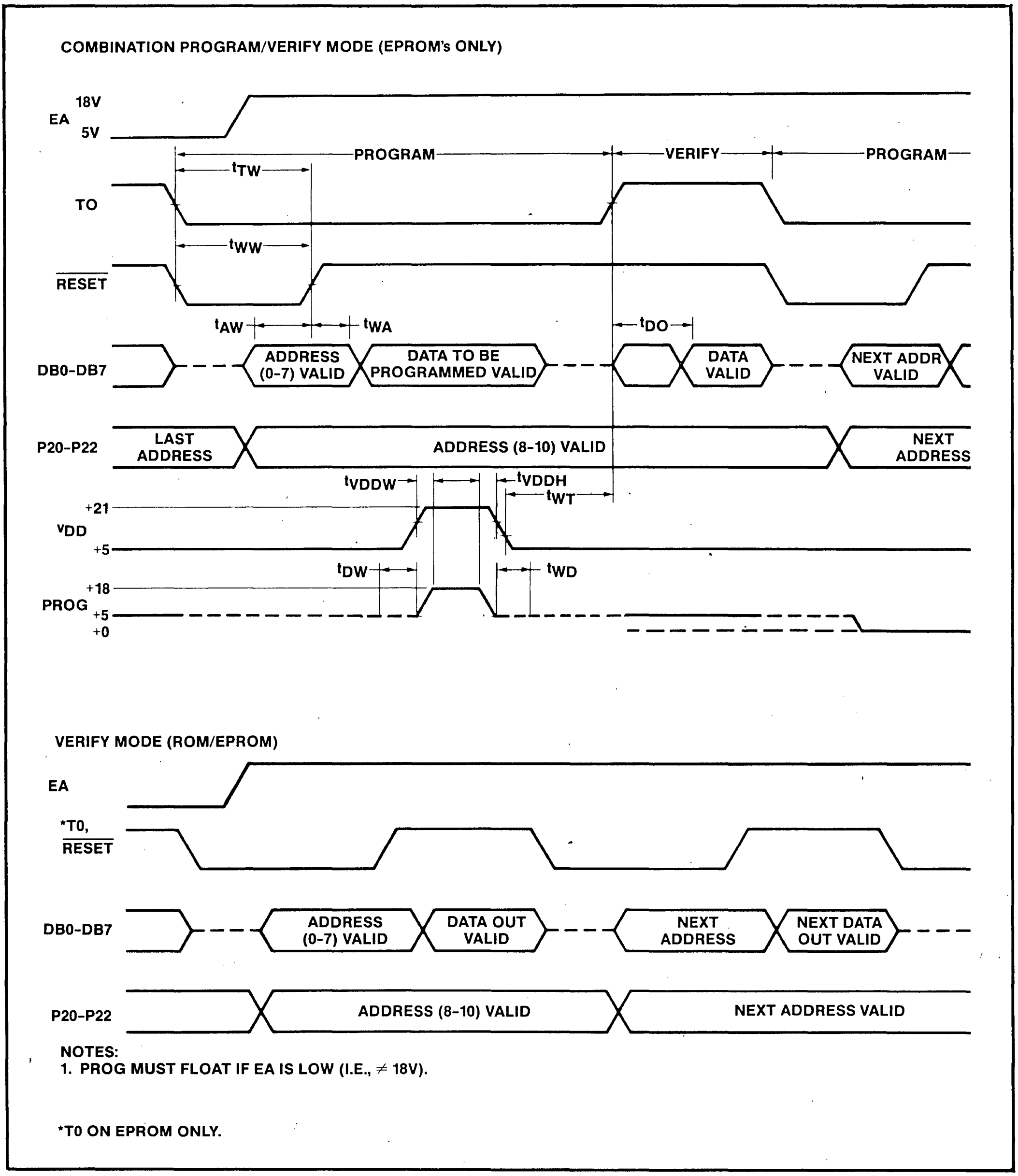


Figure 13-17. Program/Verify Sequence for 8749H/8748H

Expanded MCS®-48 System

14

CHAPTER 14
EXPANDED MCS®-48 SYSTEM

14.0 INTRODUCTION

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

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

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

1) Expander I/O —A special I/O Expander circuit, the 8243, provides for the addition of four 4-bit Input/Output ports with the sacrifice of only the lower half (4-bits) of port 2 for inter-device communication. Multiple 8243's may be added to this 4-bit bus by generating the required "chip select" lines.
2) Standard 8085 Bus —One port of the 8048AH/8049AH is like the 8-bit bidirectional data bus of the 8085 microcomputer system allowing interface to the numerous standard memories and peripherals of the MCS®-80/85 microcomputer family.

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

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

14.1 EXPANSION OF PROGRAM MEMORY

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

14.1.1 Instruction Fetch Cycle (External)

As shown in Figure 14-1, for all instruction fetches from addresses of 1024 (2048) or greater, the following will occur:

1) The contents of the 12-bit program counter will be output on BUS and the lower half of port 2.
2) Address Latch Enable (ALE) will indicate the time at which address is valid. The trailing edge of ALE is used to latch the address externally.
3) Program Store Enable ($\overline{\text{PSEN}}$) indicates that an external instruction fetch is in progress and serves to enable the external memory device.
4) BUS reverts to input (floating) mode and the processor accepts its 8-bit contents as an instruction word.

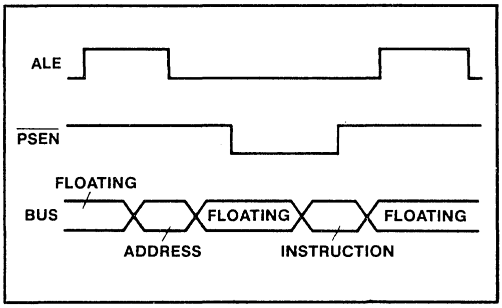


Figure 14-1. Instruction Fetch from External Program Memory

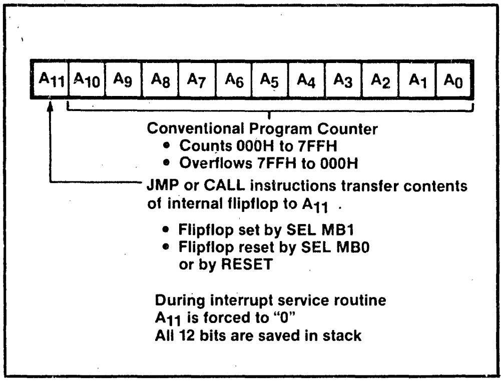
All instruction fetches, including internal addresses, can be forced to be external by activating the EA pin of the 8048AH/8049AH/8050AH. The 8035AHL/8039AHL/8040AHL processors without program memory always operate in the external program memory mode (EA = 5V).

14.1.2 Extended Program Memory Addressing (Beyond 2K)

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

PROGRAM MEMORY BANK SWITCH

The switching of 2K program memory banks is accomplished by directly setting or resetting the most significant bit of the program counter (bit 11); see Figure 14-2. Bit 11 is not altered by normal incrementing of the program counter but is loaded with the contents of a special flip-flop each time a JMP or CALL instruction is executed. This special flip-flop is set by executing an SEL MB1 instruction and reset by SEL MB0. Therefore, the SEL MB instruction may be executed at any time prior to the actual bank switch which occurs during the next branch instruction encountered. Since all twelve bits of the program counter, including bit 11, are stored in the stack, when a Call is executed, the user may jump to subroutines across the 2K boundary and the proper bank will be restored upon return. However, the bank switch flip-flop will not be altered on return.



Conventional Program Counter
- Counts 000H to 7FFH
- Overflows 7FFH to 000H

JMP or CALL instructions transfer contents of internal flipflop to A_{11}.
- Flipflop set by SEL MB1
- Flipflop reset by SEL MB0 or by RESET

During interrupt service routine A_{11} is forced to "0"
All 12 bits are saved in stack

Figure 14-2. Program Counter

INTERRUPT ROUTINES

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

14.1.3 Restoring I/O Port Information

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

14.1.4 Expansion Examples

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

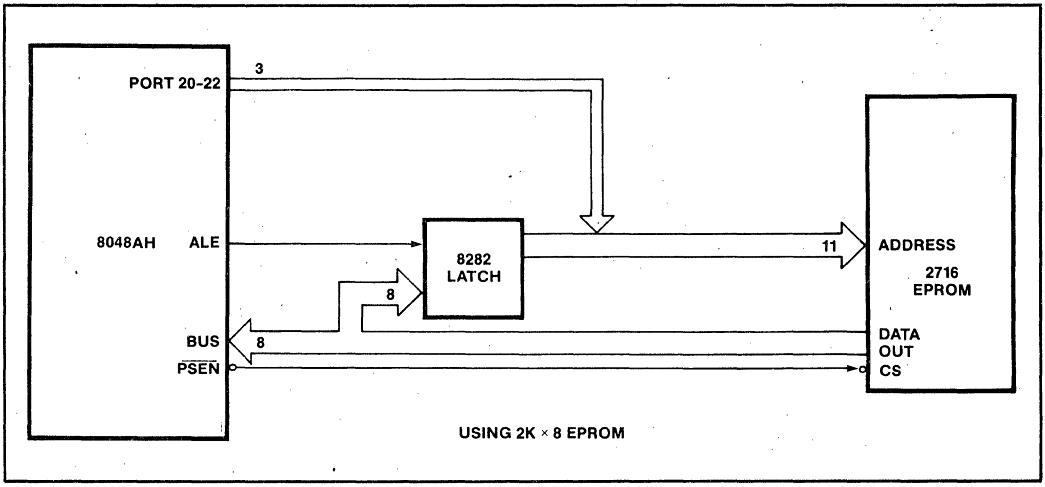


Figure 14-3. Expanding MCS®-48 Program Memory Using Standard Memory Products

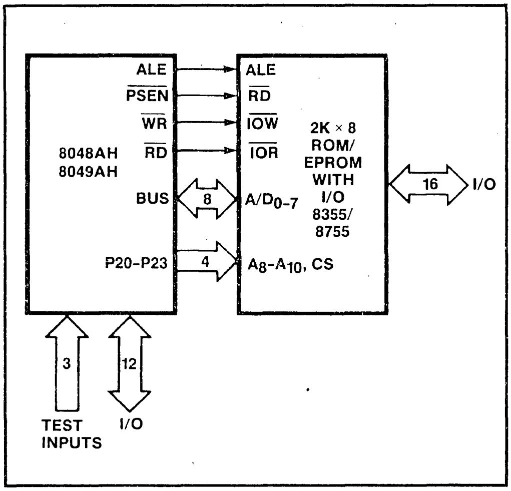


Figure 14-4. External Program Memory Interface

Figure 14-4 shows how the 8755/8355 EPROM/ROM with I/O interfaces directly to the 8048AH without the need for an address latch. The 8755/8355 contains an internal 8-bit address latch eliminating the need for an 8212 latch. In addition to a 2K x 8 program memory, the 8755/8355 also contains 16 I/O lines addressable as two 8-bit ports. These ports are addressed as external RAM; therefore the $\overline{RD}$ and $\overline{WR}$ outputs of the 8048AH are required. See the following ection on data memory expansion for more detail. The subsequent section on I/O expansion explains the operation of the 16 I/O lines.

14.2 EXPANSION OF DATA MEMORY

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

14.2.1 Read/Write Cycle

All address and data is transferred over the 8 lines of BUS. As shown in Figure 14-5, a read or write cycle occurs as follows:

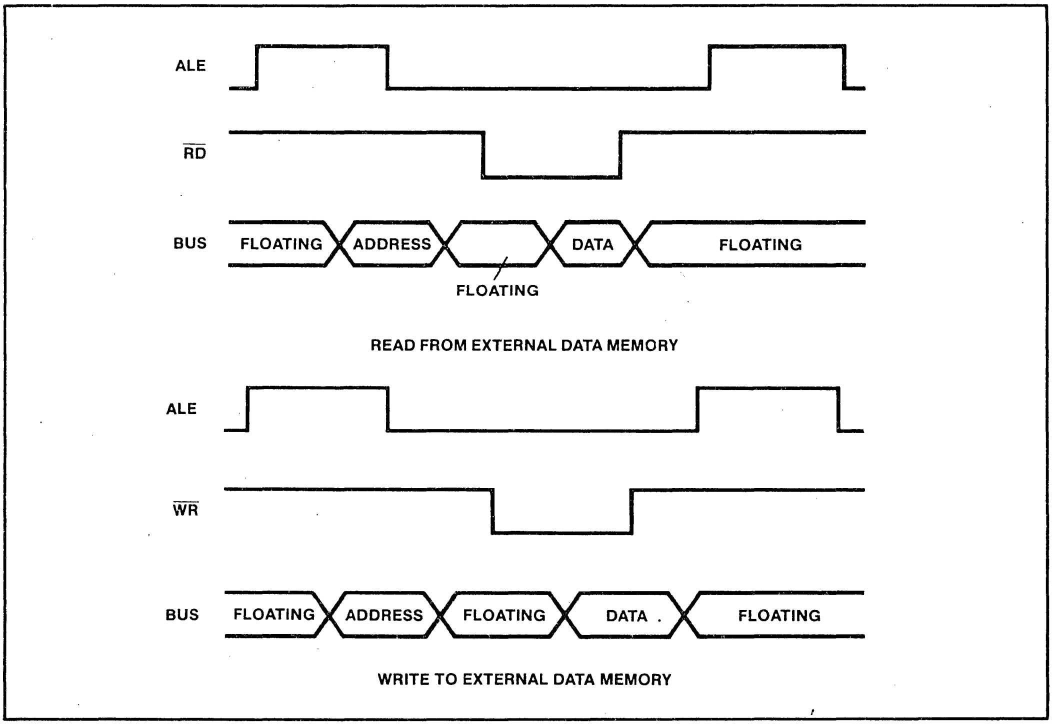


Figure 14-5. External Data Memory Timings

1) The contents of register R0 or R1 is outputed on BUS.

2) Address Latch Enable (ALE) indicates address is valid. The trailing edge of ALE is used to latch the address externally.

3) A read ($\overline{RD}$) or write ($\overline{WR}$) pulse on the corresponding output pins of the 8048AH indicates the type of data memory access in progress. Output data is valid at the trailing edge of $\overline{WR}$ and input data must be valid at the trailing edge of $\overline{RD}$.

4) Data (8 bits) is transferred in or out over BUS.

14.2.2 Addressing External Data Memory

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

14.2.3 Examples of Data Memory Expansion

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

14.3 EXPANSION OF INPUT/OUTPUT

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

14.3.1 I/O Expander Device

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

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

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

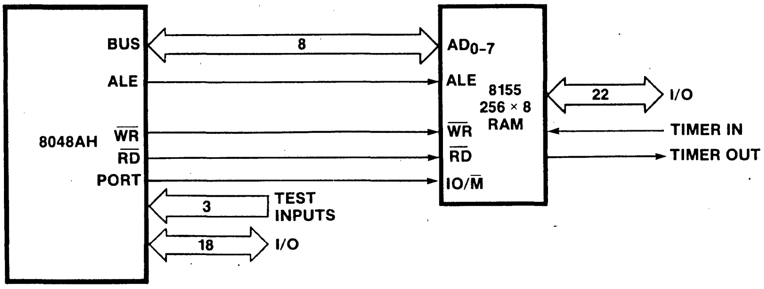


Figure 14-6. 8048AH Interface to 256 x 8 Standard Memories

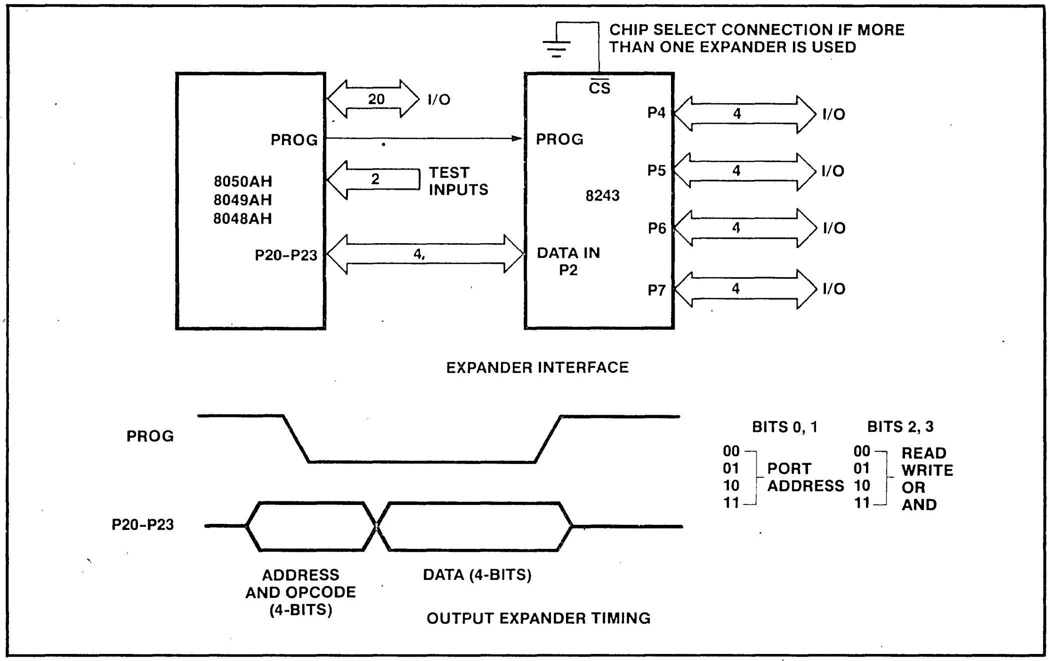


Figure 14-7. 8243 Expander I/O Interface

Nibble 1		Nibble 2
3 2 1 0		3 2 1 0
I I A A		d d d d
Instruction Code	Port Address	data

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

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

I/O PORT CHARACTERISTICS

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

14.3.2 I/O Expansion with Standard Peripherals

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

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

14.3.3 Combination Memory and I/O Expanders

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

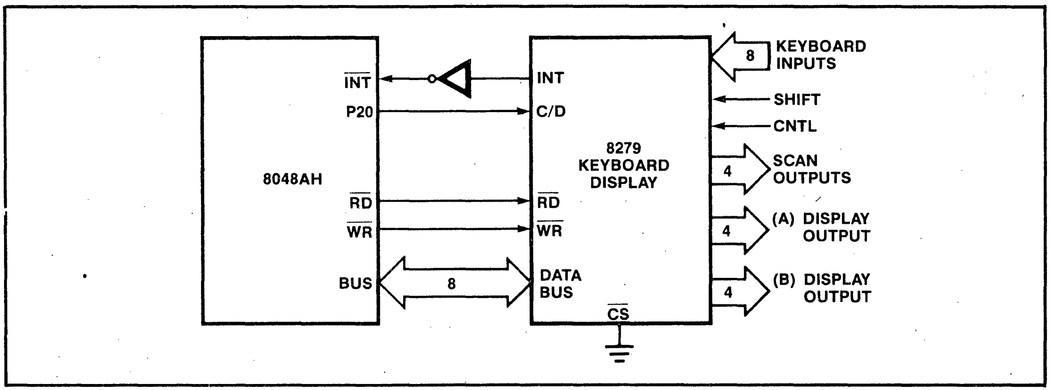


Figure 14-8. Keyboard/Display Interface

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

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

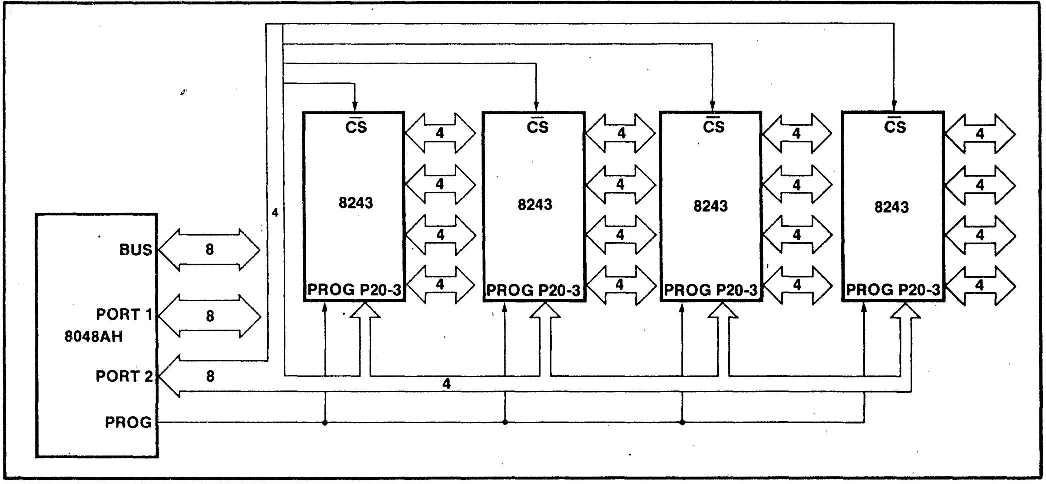


Figure 14-9. Low Cost I/O Expansion

I/O EXPANSION EXAMPLES (SEE ALSO CHAPTER 5)

Figure 14-9 shows the expansion of I/O using multiple 8243's. The only difference from a single 8243 system is the addition of chip selects provided by additional 8048AH output lines. Two output lines and a decoder could also be used to address the four chips. Large numbers of 8243's would require a chip select decoder chip such as the 8205 to save I/O pins.

Figure 14-10 shows the 8048AH interface to a standard MCS®-80 peripheral; in this case, the 8255 Programmable Peripheral Interface, a 40-pin part which provides three 8-bit programmable I/O ports. The 8255 bus interface is typical of programmable MCS®-80 peripherals with an 8-bit bidirectional data bus, a RD and WR input for Read/Write control, a CS (chip select) input used to enable the Read/Write control logic and the address inputs used to select various internal registers.

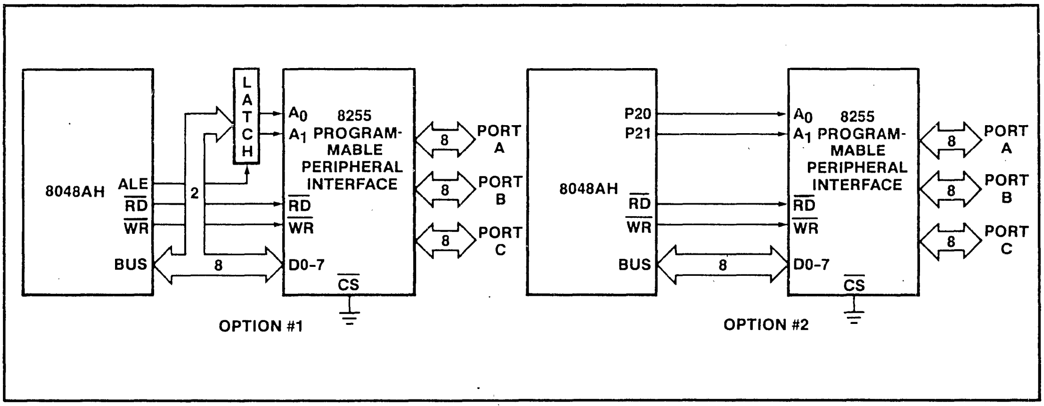


Figure 14-10. Interface to MCS®-80 Peripherals

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

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

14.4 MULTI-CHIP MCS®-48 SYSTEMS

Figure 14-11 shows the addition of two memory expanders to the 8048AH, one 8355/8755 ROM and one 8156 RAM. The main consideration in designing such a system is the addressing of the various memories and I/O ports. Note that in this configuration address,lines A_{10} and A_{11} have been O Red to chip select the 8355. This ensures that the chip is active for all external program memory fetches in the 1K to 3K range and is disabled for all other addresses. This gating has been added to allow the I/O port of the 8355 to be used. If the chip was left selected all the time, there would be conflict between these ports and the RAM and I/O of the 8156. The NOR gate could be eliminated and A_{11} connected directly to the CE (instead of $\overline{CE}$) input of the 8355; however, this would create a 1K word "hole" in the program memory by causing the 8355 to be active in the 2K and 4K range instead of the normal 1K to 3K range.

In this system the various locations are addressed as follows:

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

See the memory map in Figure 14-12.

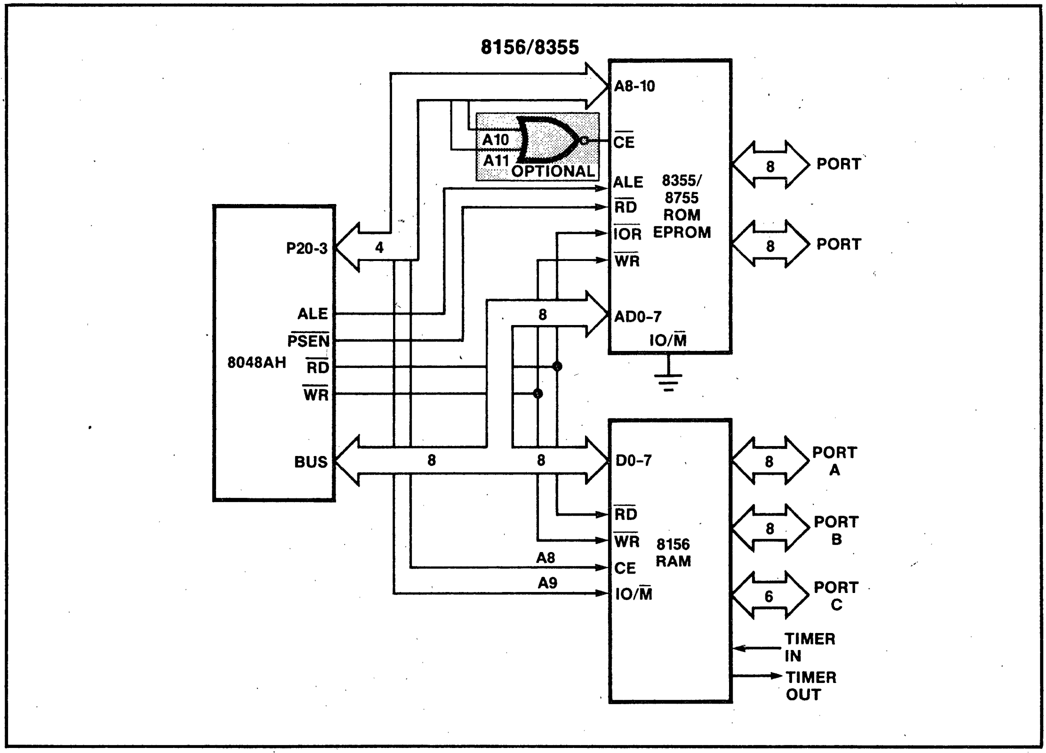


Figure 14-11. The Three-Component MCS®-48 System

14.5 MEMORY BANK SWITCHING

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

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

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

14.6 CONTROL SIGNAL SUMMARY

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

Table 14-1. MCS®-48 Control Signals

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

14.7 PORT CHARACTERISTICS

14.7.1 BUS Port Operations

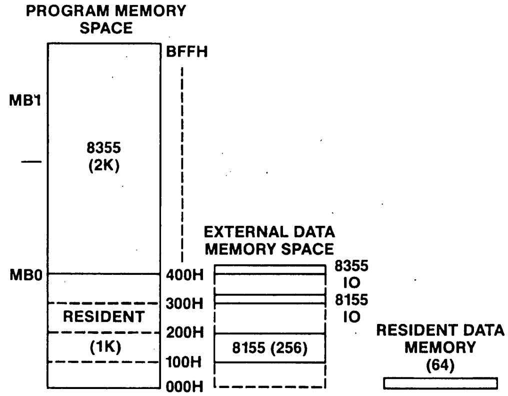
The BUS port can operate in three different modes: as a latched I/O port, as a bidirectional bus port, or as a program memory address output when external memory is used. The BUS port lines are either active high, active low, or high impedance (floating).

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

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

14.7.2 Port 2 Operations

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



SECTION	ADDRESS	DESIGNATION
PROG. MEM	000—BFF	
DATA MEM	100—IFF	
8155 PORTS	300	CMD/STATUS
	301	PORT A
	302	PORT B
	303	PORT C
	304	TIMER LOW
8355 PORTS	305	TIMER HI
	400	PORT A
	401	PORT B
	402	DDR A
	403	DDR B

Figure 14-12. Memory Map for Three-Component MCS®-48 Family

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

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

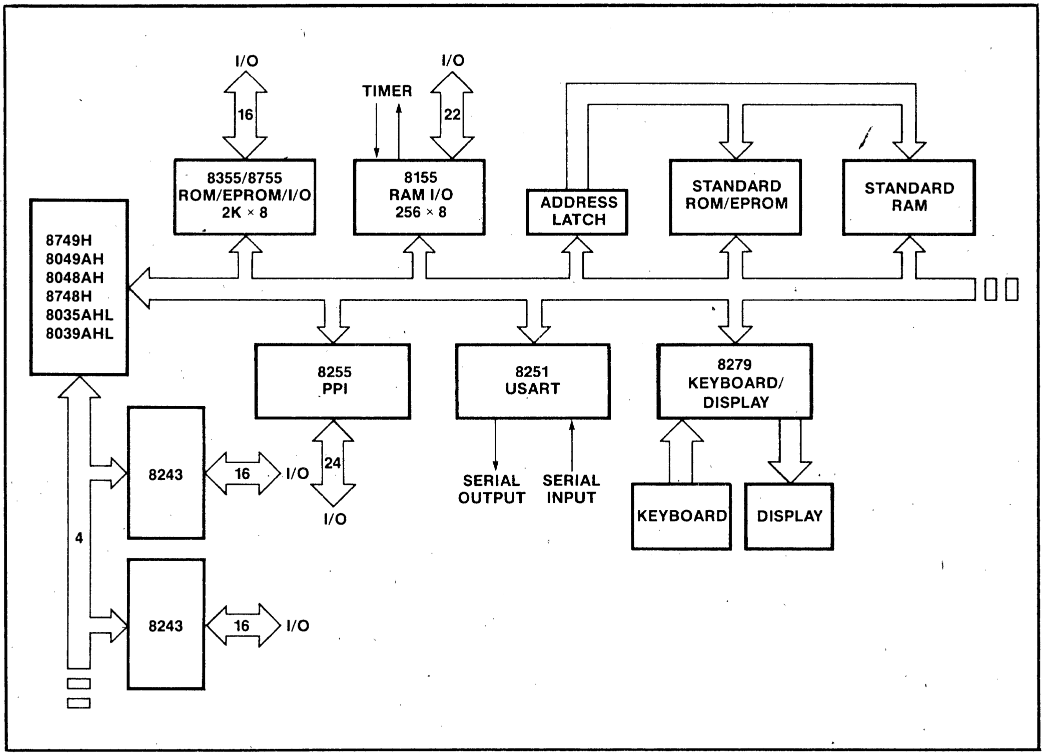


Figure 14-13. MCS®-48 Expansion Capability

MCS®-48 Instruction Set

15

CHAPTER 15
MCS®-48 INSTRUCTION SET

15.0 INTRODUCTION

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

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

15.0.1 Data Transfers

As can be seen in Figure 15-1, the 8-bit accumulator is the central point for all data transfers within the 8048. Data can be transferred between the 8 registers of each working register bank and the accumulator directly, i.e., the source or destination register is specified by the instruction. The remaining locations of the internal RAM array are referred to as Data Memory and are addressed indirectly via an address stored in either R0 or R1 of the active register bank. R0 and R1 are also used to indirectly address external data memory when it is present. Transfers to and from internal RAM require one cycle, while transfers to external RAM require two. Constants stored in Program Memory can be loaded directly to the accumulator and to the 8 working registers. Data can also be transferred directly between the accumulator and the on-

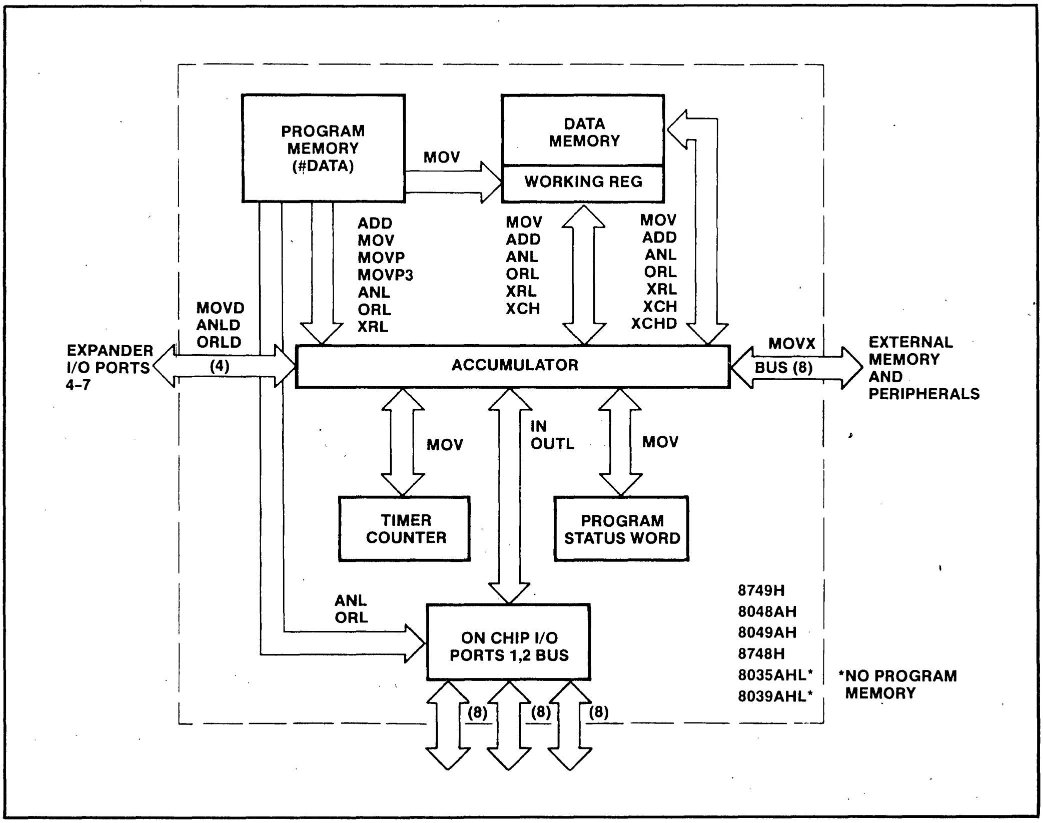


Figure 15-1. Data Transfer Instructions

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

15.0.2 Accumulator Operations

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

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

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

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

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

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

15.0.3 Register Operations

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

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

15.0.4 Flags

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

15.0.5 Branch Instructions

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

Conditional jumps can test the following inputs and machine status:

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

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

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

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

15.0.6 Subroutines

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

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

15.0.7 Timer Instructions

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

15.0.8 Control Instructions

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

There are four memory bank select instructions, two to designate the active working register bank and two to control program memory banks. The operation of the program memory bank switch is explained in section 14.1.2.

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

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

15.0.9 Input/Output Instructions

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

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

The basic three on-board I/O ports can be expanded via a 4-bit expander bus using half of port 2. I/O expander devices on this bus consisit of four 4-bit ports which are addressed as ports 4 through 7. These ports have their own AND and OR instructions like the on-board ports as well as move instructions to transfer data in or out. The expander AND and OR instructions, however, combine the contents of accumulator with the selected port rather than immediate data as is done with the on-board ports. I/O devices can also be added externally using the BUS port as the expansion bus. In this case the I/O ports become "memory mapped", i.e., they are addressed in the same way as external data memory and exist in the external data memory address space addressed by pointer register R0 or R1.

15.1 INSTRUCTION SET DESCRIPTION

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

The alphabetical listing includes the following information.

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

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

Arbitrary
Label: Mnemonic, Operand;
Descriptive Comment

MCS®-48 INSTRUCTION SET

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

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

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

*For use with internal memory only.

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

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

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

MCS®-48 INSTRUCTION SET
Symbols and Abbreviations Used

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

ADD A,R_r Add Register Contents to Accumulator

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

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

Operation: (A) ← (A) + (Rr) r = 0-7

Example:

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

ADD A,@R_i Add Data Memory Contents to Accumulator

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

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

Operation: (A) ← (A) + ((Ri)) i = 0–1

Example:

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

ADD A,#data Add Immediate Data to Accumulator

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

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

Operation: (A) ← (A) + data

Example:

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

ADDC A,R_r Add Carry and Register Contents to Accumulator

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

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

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

Example:

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

** 0–5 in 8048AH/8748H
0–6 in 8049AH/8749H
0–7 in 8050AH

ADDC A,@R$_i$ Add Carry and Data Memory Contents to Accumulator

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

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

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

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

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

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

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

Operation: (A) ← (A) + data + (C)

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

ANL A,R$_r$ Logical AND Accumulator with Register Mask

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

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

Operation: (A) ← (A) AND (Rr) r = 0–7

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

ANL A,@R$_i$ Logical AND Accumulator with memory Mask

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

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

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

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

** 0–5 in 8048AH/8748H
0–6 in 8049AH/8749H
0–7 in 8050AH

ANL A,#data Logical AND Accumulator with Immediate Mask

Encoding:

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

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

Operation: (A) ← (A) AND data

Examples:

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

ANL BUS,#data* Logical AND BUS with Immediate Mask

Encoding:

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

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

Operation: (BUS) ← (BUS) AND data

Example:

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

ANL Pp,#data Logical AND Port 1-2 with Immediate Mask

Encoding:

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

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

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

Example:

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

* For use with internal program memory ONLY.

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

Encoding:

1 0 0 1	1 1 p p	9CH-9FH

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

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

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

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

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

CALL address Subroutine Call

Encoding:

$a_{10}\ a_9\ a_8\ 1$	0 1 0 0	$a_7\ a_6\ a_5\ a_4$	$a_3\ a_2\ a_1\ a_0$

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

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

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

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

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

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

CLR A Clear Accumulator

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

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

Operation: A ← 0

CLR C Clear Carry Bit

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

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

Operation: C ← 0

CLR F1 Clear Flag 1

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

Description: Flag 1 is cleared to zero.

Operation: (F1) ← 0

CLR F0 Clear Flag 0

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

Description: Flag 0 is cleared to zero.

Operation: (F0) ← 0

CPL A Complement Accumulator

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

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

Operation: (A) ← NOT (A)

Example: Assume accumulator contains 01101010.

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

CPL C Complement Carry Bit

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

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

Operation: (C) ← NOT (C)

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

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

CPL F0 Complement Flag 0

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

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

Operation: F0 ← NOT (F0)

CPL F1 Complement Flag 1

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

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

Operation: (F1) ← NOT (F1)

DA A Decimal Adjust Accumulator

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

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

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

Example: Assume accumulator contains 10011011.

```
DA A                            ;ACC Adjusted to 00000001
                                ;WITH C SET
C AC 7       4 3       0
0  0 1 0 0 1 1 0 1 1
     0 0 0 0 0 1 1 0            ADD SIX TO BITS 0-7
0  1 1 0 1 0 0 0 0 1
     0 1 1 0                    ADD SIX TO BITS 4-7
1  0 0 0 0 0 0 0 0 1            OVERFLOW TO C
```

DEC A Decrement Accumulator

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

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

Operation: (A) ← (A) -1

Example: Decrement contents of external data memory location 63.

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

DEC Rr Decrement Register

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

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

Operation: (Rr) ← (Rr) -1   r = 0–7

Example:

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

DIS I External Interrupt

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

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

DIS TCNTI Disable Timer/Counter Interrupt

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

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

DJNZ R_r, address Decrement Register and Test

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

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

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

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

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

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

EN I Enable External Interrupt

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

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

EN TCNTI Enable Timer/Counter Interrupt

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

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

ENT0 CLK Enable Clock Output

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

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

Example:

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

IN A,Pp Input Port or Data to Accumulator

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

Description: This is a 2-cycle instruction. Data present on port 'p' is transferred (read) to the accumulator.

Operation: (A) ← (Pp) p = 1–2

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

INC A Increment Accumulator

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

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

Operation: (A) ← (A) +1

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

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

INC R_r Increment Register

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

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

Operation: (Rr) ← (Rr) + 1 r = 0–7

Example:

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

INC @R_i Increment Data Memory Location

Encoding: | 0 0 0 1 | 0 0 0 i | 10H–11H

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

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

Example:

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

INS A,BUS* Strobed Input of BUS Data to Accumulator

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

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

Operation: (A) ← (BUS)

Example:

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

* For use with internal program memory ONLY.

** 0–5 in 8048AH/8748H
0–6 in 8049AH/8749H
0–7 in 8050AH

JBb address Jump If Accumulator Bit Is Set

Encoding:

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

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

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

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

Example: JB4IS1: JB4 NEXT ;JUMP TO ‘NEXT’ ROUTINE
;IF ACC BIT 4 = 1

JC address Jump If Carry Is Set

Encoding:

1 1 1 1	0 1 1 0	a_7 a_6 a_5 a_4	a_3 a_2 a_1 a_0	F6H

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

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

Example: JC1: JC OVFLOW ;JUMP TO ‘OVFLOW’ ROUTINE
;IF C = 1

JF0 address Jump If Flag 0 Is Set

Encoding:

1 0 1 1	0 1 1 0	a_7 a_6 a_5 a_4	a_3 a_2 a_1 a_0	B6H

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

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

Example: JF0IS1: JF0 TOTAL ;JUMP TO ‘TOTAL’ ROUTINE IF F0 = 1

JF1 address Jump If Flag 1 Is Set

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

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

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

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

JMP address Direct Jump within 2K Block

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

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

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

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

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

JMPP @A Indirect Jump within Page

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

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

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

Example: Assume accumulator contains 0FH.

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

JNC address   Jump If Carry Is Not Set

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

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

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

Example:

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

JNI address   Jump If Interrupt Input Is Low

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

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

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

Example:

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

JNT0 address   Jump If Test 0 is Low

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

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

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

Example:

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

JNT1 address Jump If Test 1 Is Low

Encoding:

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

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

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

JNZ Address Jump If Accumulator Is Not Zero

Encoding:

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

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

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

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

JTF address Jump If Timer Flag Is Set

Encoding:

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

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

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

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

JT0 address Jump If Test 0 Is High

Encoding:

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

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

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

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

JT1 address Jump If Test 1 Is High

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

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

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

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

JZ address Jump If Accumulator Is Zero

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

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

Operation: (PC_{0-7}) ← addr If A = 0
(PC) = (PC) + 2 If A ≠ 1

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

MOV A,#data Move Immediate Data to Accumulator

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

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

Operation: (A) ← data

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

MOV A,PSW Move PSW Contents to Accumulator

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

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

Operation: (A) ← (PSW)

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

MOV A,R_r Move Register Contents to Accumulator

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

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

Operation: (A) ← (Rr) r = 0-7

Example:

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

MOV A,@R_i Move Data Memory Contents to Accumulator

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

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

Operation: (A) ← ((Ri)) i = 0-1

Example: Assume R1 contains 00110110.

```
MADM: MOV A,@R1                ;MOVE CONTENTS OF DATA MEM
                               ;LOCATION 54 TO ACC
```

MOV A,T Move Timer/Counter Contents to Accumulator

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

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

Operation: (A) ← (T)

Example: Jump to "EXIT" routine when timer reaches '64', that is, when bit 6 set—assuming initialization 64,

```
TIMCHK: MOV A,T                ;MOVE TIMER CONTENTS TO ACC
        JB6 EXIT               ;JUMP TO 'EXIT' IF ACC BIT 6 = 1
```

MOV PSW,A Move Accumulator Contents to PSW

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

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

Operation: (PSW) ← (A)

Example: Move up stack pointer by two memory locations, that is, increment the pointer by one.

```
INCPTR: MOV A,PSW              ;MOVE PSW CONTENTS TO ACC
        INC A                  ;INCREMENT ACC BY ONE
        MOV PSW,A              ;MOVE ACC CONTENTS TO PSW
```

** 0–5 in 8048AH/8748H
0–6 in 8049AH/8749H
0–7 in 8050AH

MOV R_r,A Move Accumulator Contents to Register

Encoding:

1 0 1 0	1 r r r	A8H–AFH

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

Operation: (Rr) ← (A) r = 0–7

Example:

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

MOV R_r,#data Move Immediate Data to Register

Encoding:

1 0 1 1	1 r_2 r_1 r_0	d_7 d_6 d_5 d_4	d_3 d_2 d_1 d_0	B8H–BFH

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

Operation: (Rr) ← data r = 0–7

Examples:

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

MOV @ R_i,A Move Accumulator Contents to Data Memory

Encoding:

1 0 1 0	0 0 0 i	A0H–A1H

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

Operation: ((Ri)) ← (A) i = 0–1

Example: Assume R0 contains 00000111.

```
MDMA:  MOV @R0,A               ;MOVE CONTENTS OF ACC TO
                               ;LOCATION 7 (REG 7)
```

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

Encoding:

1 0 1 1	0 0 0 i	d_7 d_6 d_5 d_4	d_3 d_2 d_1 d_0	B0H–B1H

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

Operation: ((Ri)) ← data i = 0–1

Examples: Move the hexadecimal value AC3F to locations 62–63.

```
MIDM:  MOV R0,#62              ;MOVE '62' DEC TO ADDR REG 0
       MOV @R0,#0ACH           ;MOVE 'AC' HEX TO LOCATION 62
       INC R0                  ;INCREMENT REG 0 to '63'
       MOV @R0,#3FH            ;MOVE '3F' HEX TO LOCATION 63
```

** 0–5 in 8048AH/8748H
0–6 in 8049AH/8749H
0–7 in 8050AH

MOV T,A Move Accumulator Contents to Timer/Counter

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

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

Operation: (T) ← (A)

Example: Initialize and start event counter.

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

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

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

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

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

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

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

Example:

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

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

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

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

Operation: (Pp) ← (A_{0-3}) P = 4-7

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

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

MOVP A,@A Move Current Page Data to Accumulator

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

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

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

Example:

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

MOVP3 A,@A Move Page 3 Data to Accumulator

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

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

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

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

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

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

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

MOVX A,@R_i Move External-Data-Memory Contents to Accumulator

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

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

Operation: (A) ← ((Ri)) i = 0–1

Example: Assume R1 contains 01110110.

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

MOVX @R_i,A Move Accumulator Contents to External Data Memory

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

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

Operation: ((Ri)) ← A i = 0–1

Example: Assume R0 contains 11000111.

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

NOP The NOP Instruction

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

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

ORL A,R_r Logical OR Accumulator With Register Mask

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

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

Operation: (A) ← (A) OR (Rr) r = 0–7

Example:

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

ORL A,@R_i Logical OR Accumulator With Memory Mask

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

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

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

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

ORL A,#data Logical OR Accumulator With Immediate Mask

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

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

Operation: (A) ← (A) OR data

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

ORL BUS,#data* Logical OR BUS With Immediate Mask

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

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

Operation: (BUS) ← (BUS) OR data

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

ORL Pp, #data Logical OR Port 1 or 2 With Immediate Mask

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

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

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

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

* For use with internal program memory ONLY.

** 0–5 in 8048AH/8748H
0–6 in 8049AH/8749H
0–7 in 8050AH

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

Encoding:

1 0 0 0	1 1 p p	8CH-8FH

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

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

Example:

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

OUTL BUS,A* Output Accumulator Data to BUS

Encoding:

0 0 0 0	0 0 1 0	02H

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

Operation: (BUS) ← (A)

Example:

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

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

Encoding:

0 0 1 1	1 0 p p	39H-3AH

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

Operation: (Pp) ← (A) p = 1-2

Example:

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

* For use with internal program memory ONLY.

RET Return Without PSW Restore

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

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

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

RETR Return with PSW Restore

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

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

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

RL A Rotate Left without Carry

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

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

Operation:

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

Example: Assume accumulator contains 10110001.

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

RLC A Rotate Left through Carry

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

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

Operation:

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

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

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

RR A Rotate Right without Carry

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

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

Operation:

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

Example: Assume accumulator contains 10110001.

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

RRC A Rotate Right through Carry

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

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

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

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

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

SEL MB0 Select Memory Bank 0

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

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

Operation: (DBF) ← 0

Example: Assume program counter contains 834 Hex.

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

SEL MB1 Select Memory Bank 1

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

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

Operation: (DBF) ← 1

SEL RB0 Select Register Bank 0

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

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

Operation: (BS) ← 0

SEL RB1 Select Register Bank 1

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

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

Operation: (BS) ← 1

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

Operation:

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

STOP TCNT Stop Timer/Event-Counter

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

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

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

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

STRT CNT Start Event Conter

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

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

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

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

STRT T Start Timer

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

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

Example: Initialize and start timer.

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

SWAP A Swap Nibbles within Accumulator

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

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

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

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

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

XCH A,R_r Exchange Accumulator-Register Contents

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

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

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

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

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

XCH A,@R_i Exchange Accumulator and Data Memory Contents

Encoding: | 0 0 1 0 | 0 0 0 i | 20H–21H

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

Operation: (A) ⇆ ((Ri)) i = 0–1

Example: Decrement contents of location 52.

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

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

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

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

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

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

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

XRL A,R_r Logical XOR Accumulator With Register Mask

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

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

Operation: (A) ← (A) XOR (Rr) r = 0–7

Example:

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

** 0–5 in 8048AH/8748H
0–6 in 8049AH/8749H
0–7 in 8050AH

XRL A,@R$_i$ Logical XOR Accumulator With Memory Mask

Encoding:

1 1 0 1	0 0 0 i	D0H-D1H

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

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

Example:

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

XRL A,#data Logical XOR Accumulator With Immediate Mask

Encoding:

1 1 0 1	0 0 1 1	d_7 d_6 d_5 d_4	d_3 d_2 d_1 d_0	D3H

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

Operation: (A) ← (A) XOR data

Example:

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

** 0–5 in 8048AH/8748H
0–6 in 8049AH/8749H
0–7 in 8050AH

MCS®-48 Data Sheets

16

PRELIMINARY

8243
MCS®-48 INPUT/OUTPUT EXPANDER

- ■ 0° C to 70° C Operation

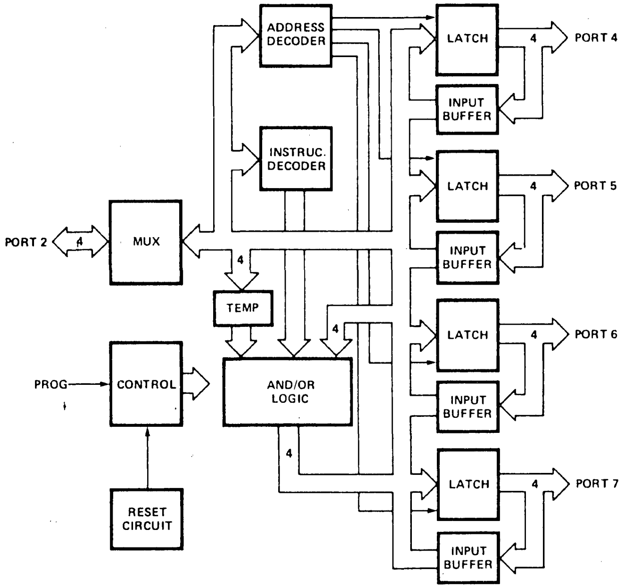


Figure 1. 8243 Block Diagram

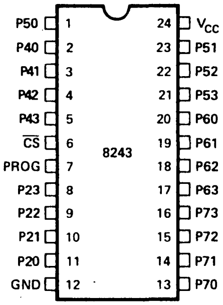


Figure 2. 8243 Pin Configuration

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.
$\overline{\text{CS}}$	6	Chip Select Input. A high on CS inhibits any change of output or internal status.
P20-P23	11-8	Four (4) bit bi-directional port contains the address and control bits on a high to low transition of PROG. During a low to high transition contains the data for a selected output port if a write operation, or the data from a selected port before the low to high transition if a read operation.
GND	12	0 volt supply.
P40-P43	2-5	Four (4) bit bi-directional I/O ports.
P50-P53 P60-P63 P70-P73	1, 23-21 20-17 13-16	May be programmed to be input (during read), low impedance latched output (after write), or a tri-state (after read). Data on pins P20-P23 may be directly written, ANDed or ORed with previous data.
V_{CC}	24	+5 volt supply.

FUNCTIONAL DESCRIPTION

General Operation

The 8243 contains four 4-bit I/O ports which serve as an extension of the on-chip I/O and are addressed as 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.

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%)

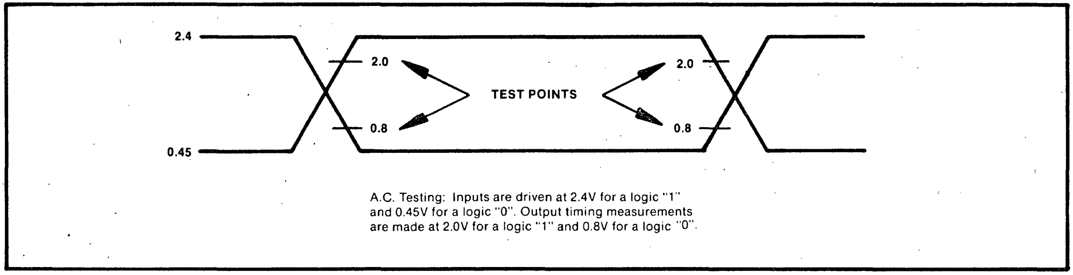
Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
V_{IL}	Input Low Voltage	-0.5		0.8	V	
V_{IH}	Input High Voltage	2.0		V_{CC}+0.5	V	
V_{OL1}	Output Low Voltage Ports 4-7			0.45	V	I_{OL} = 4.5 mA*
V_{OL2}	Output Low Voltage Port 7			1	V	I_{OL} = 20 mA
V_{OH1}	Output High Voltage Ports 4-7	2.4			V	I_{OH} = 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			0.45	V	I_{OL} = 0.6 mA
I_{CC}	V_{CC} Supply Current		10	20	mA	Note 1
V_{OH2}	Output Voltage Port 2	2.4				I_{OH} = 100μA
I_{OL}	Sum of all I_{OL} from 16 Outputs			72	mA	4.5 mA Each Pin

*See following graph for additional sink current capability

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

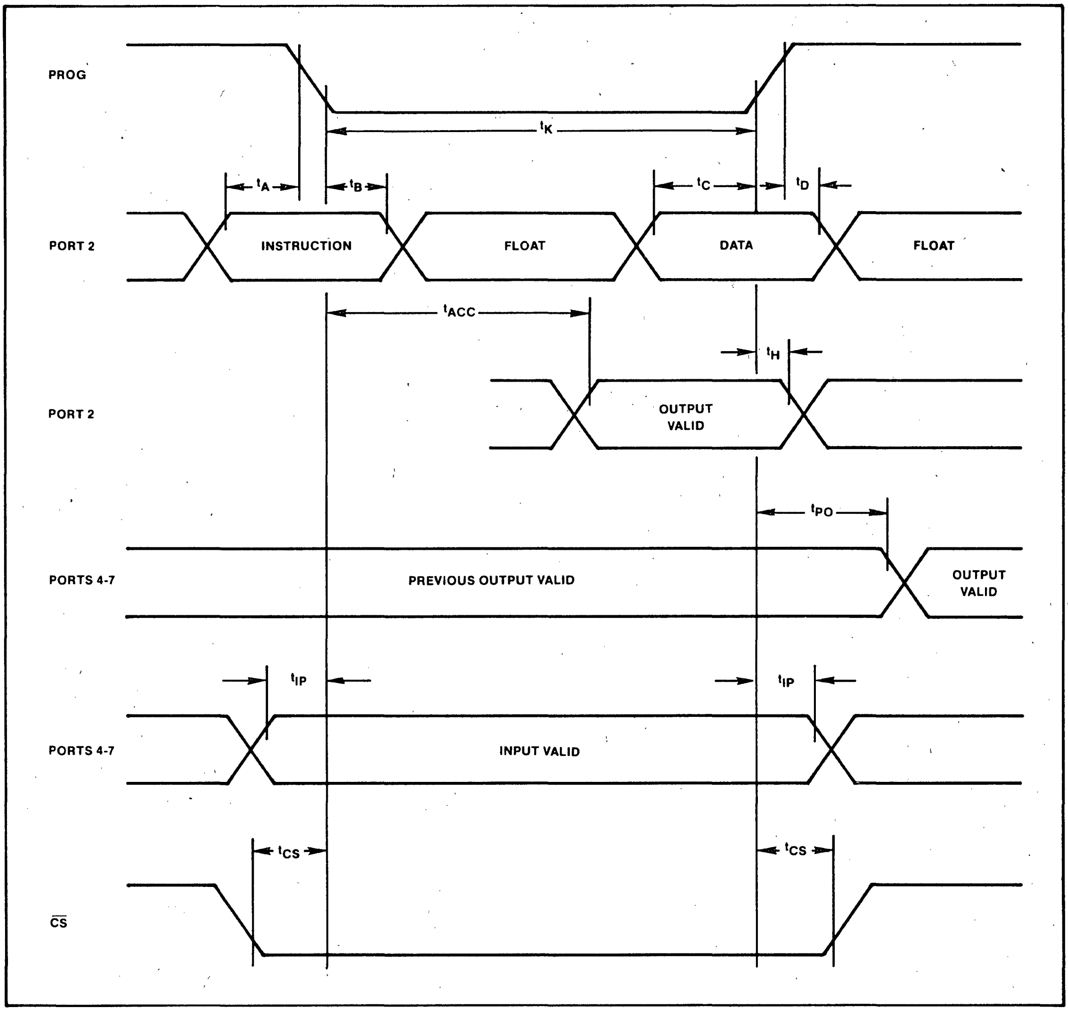
Symbol	Parameter	Min	Max	Units	Test Conditions
t_A	Code Valid Before PROG	100		ns	80 pF Load
t_B	Code Valid After PROG	60		ns	20 pF Load
t_C	Data Valid Before PROG	200		ns	80 pF Load
t_D	Data Valid After PROG	20		ns	20 pF Load
t_H	Floating After PROG	0	150	ns	20 pF Load
t_K	PROG Negative Pulse Width	700		ns	
t_{CS}	CS Valid Before/After PROG	50		ns	
t_{PO}	Ports 4-7 Valid After PROG		700	ns	100 pF Load
t_{LP1}	Ports 4-7 Valid Before/After PROG	100		ns	
t_{ACC}	Port 2 Valid After PROG		650	ns	80 pF Load

Note 1: I_{CC} (-40° C to 85° C EXPRESS options) 15 mA typical/25 mA maximum.



A.C. Testing: Inputs are driven at 2.4V for a logic "1" and 0.45V for a logic "0". Output timing measurements are made at 2.0V for a logic "1" and 0.8V for a logic "0".

WAVEFORMS



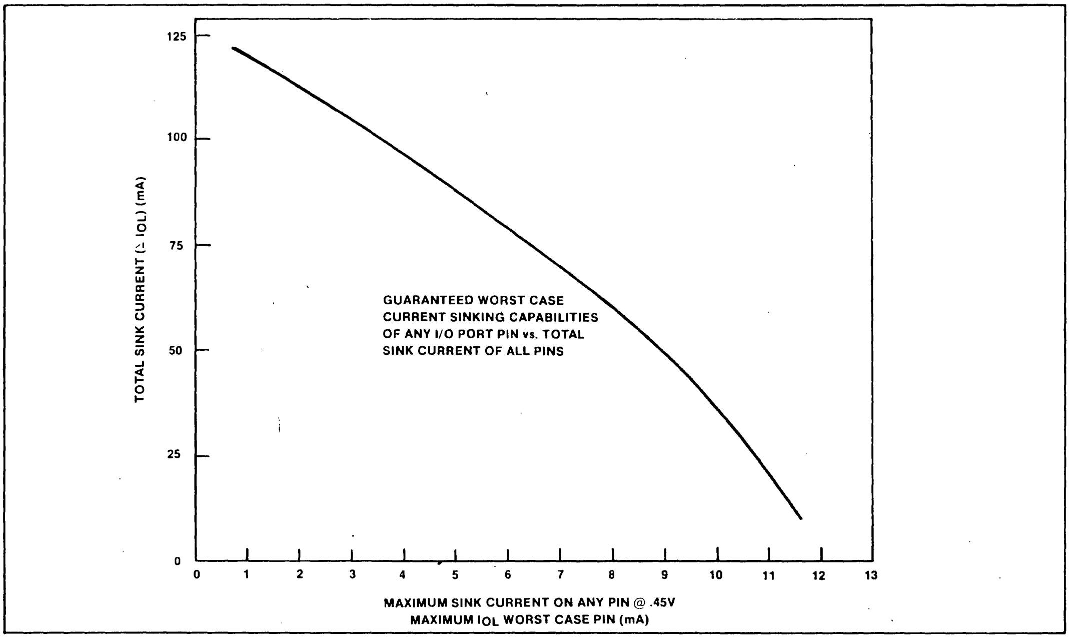


Figure 3

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.

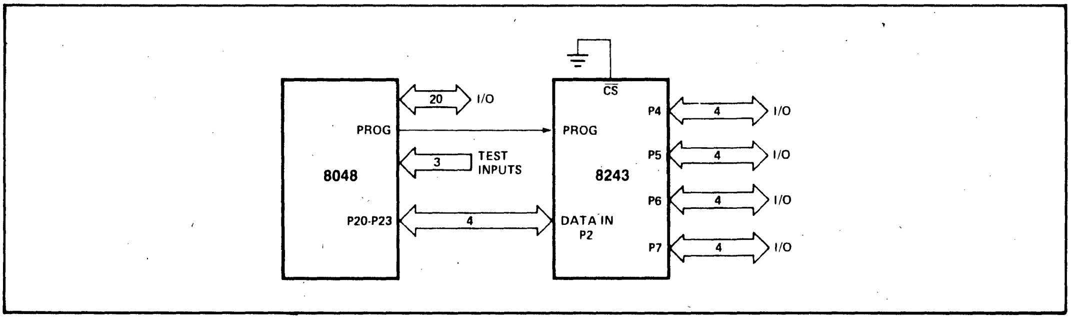


Figure 4. Expander Interface

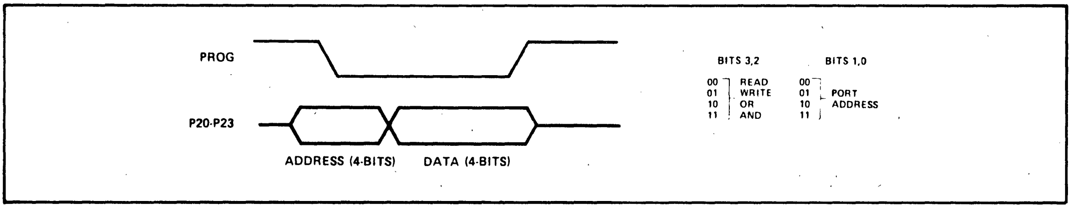


Figure 5. Output Expander Timing

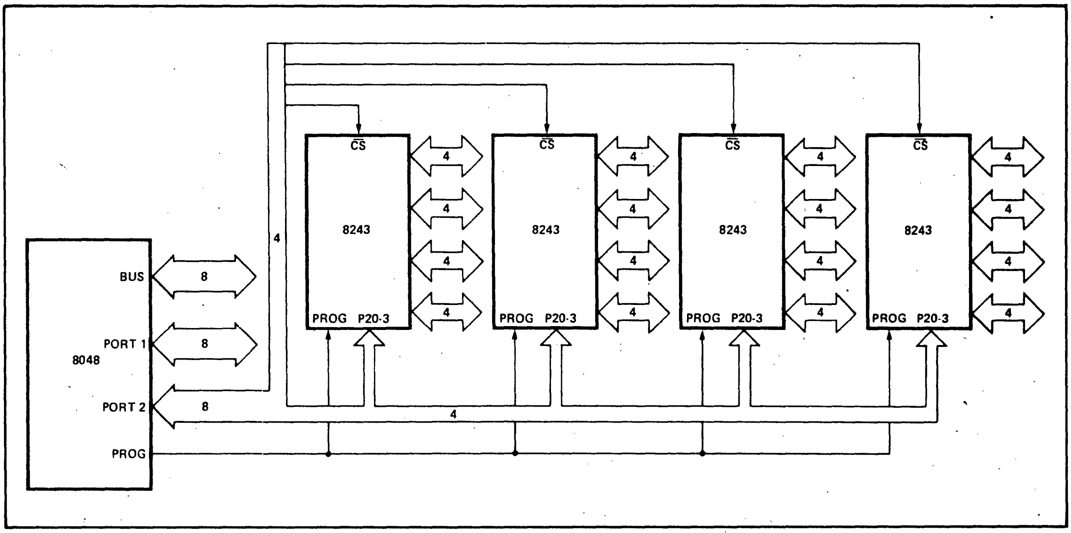


Figure 6. Using Multiple 8243's

8048AH/8748H/8035AHL/8049AH 8749H/8039AHL/8050AH/8040AHL HMOS SINGLE-COMPONENT 8-BIT MICROCOMPUTER

- **High Performance HMOS II**
- **Interval Timer/Event Counter**
- **Two Single Level Interrupts**
- **Single 5-Volt Supply**
- **Over 96 Instructions; 90% Single Byte**
- **Reduced Power Consumption**
- **Compatible with 8080/8085 Peripherals**
- **Easily Expandable Memory and I/O**
- **Up to 1.36 μSec Instruction Cycle All Instructions 1 or 2 cycles**

The Intel MCS®-48 family are totally self-sufficient, 8-bit parallel computers fabricated on single silicon chips using Intel's advanced N-channel silicon gate HMOS process.

The family contains 27 I/O lines, an 8-bit timer/counter, and on-board oscillator/clock circuits. For systems that require extra capability, the family can be expanded using MCS®-80/MCS®-85 peripherals.

To minimize development problems and provide maximum flexibility, a logically and functionally pin-compatible version of the ROM devices with UV-erasable user-programmable EPROM program memory is available with minor differences.

These microcomputers are designed to be efficient controllers as well as arithmetic processors. They have extensive bit handling capability as well as facilities for both binary and BCD arithmetic. Efficient use of program memory results from an instruction set consisting mostly of single byte instructions and no instructions over 2 bytes in length.

Device	Internal Memory		RAM Standby
8050AH	4K × 8 ROM	256 × 8 RAM	yes
8049AH	2K × 8 ROM	128 × 8 RAM	yes
8048AH	1K × 8 ROM	64 × 8 RAM	yes
8040AHL	none	256 × 8 RAM	yes
8039AHL	none	128 × 8 RAM	yes
8035AHL	none	64 × 8 RAM	yes
8749H	2K × 8 EPROM	128 × 8 RAM	no
8748H	1K × 8 EPROM	64 × 8 RAM	no

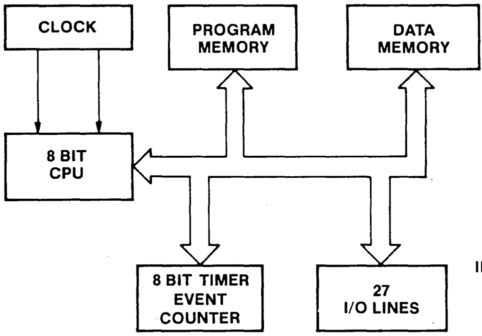


Figure 1. Block Diagram

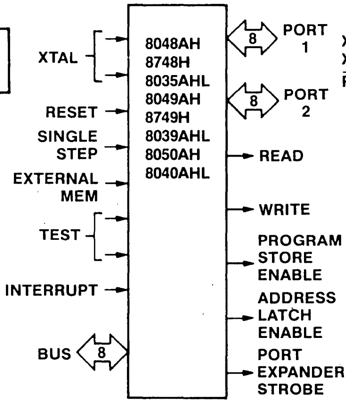


Figure 2. Logic Symbol

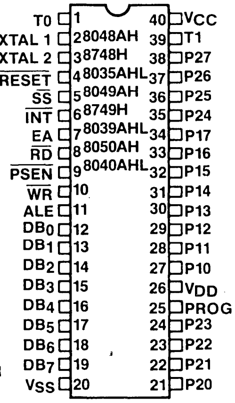


Figure 3. Pin Configuration

Intel Corporation Assumes No Responsibility for the Use of Any Circuitry Other Than Circuitry Embodied in an Intel Product. No Other Circuit Patent Licenses are Implied.

January 1983

Order Number 210983-001

Table 1. Pin Description

Symbol	Pin No.	Function	Device
V_{SS}	20	Circuit GND potential	All
V_{DD}	26	+5V during normal operation.	All
		Low power standby pin.	8048AH 8035AHL 8049AH 8039AHL 8050AH 8040AHL
		Programming power supply (+21V).	8748H 8749H
V_{CC}	40	Main power supply; +5V during operation and programming.	All
PROG	25	Output strobe for 8243 I/O expander.	All
		Program pulse (+18V) input pin during programming.	8748H 8749H (See Note)
P10-P17 Port 1	27-34	8-bit quasi-bidirectional port.	All
P20-P23 P24-P27 Port 2	21-24 35-38	8-bit quasi-bidirectional port. P20-P23 contain the four high order program counter bits during an external program memory fetch and serve as a 4-bit I/O expander bus for 8243.	All
DB0-DB7 BUS	12-19	True bidirectional port which can be written or read synchronously using the $\overline{\text{RD}}$, $\overline{\text{WR}}$ strobes. The port can also be statically latched.	All
(Con't)		Contains the 8 low order program counter bits during an external program memory fetch, and receives the addressed instruction under the control of $\overline{\text{PSEN}}$. Also contains the address and data during an external RAM data store instruction, under control of ALE, $\overline{\text{RD}}$, and $\overline{\text{WR}}$.	
T0	1	Input pin testable using the conditional transfer instructions JT0 and JNT0. T0 can be designated as a clock output using ENT0 CLK instruction	All
		Used during programming.	8748H 8749H
T1	39	Input pin testable using the JT1, and JNT1 instructions. Can be designated the timer/counter input using the STRT CNT instruction.	All
$\overline{\text{INT}}$	6	Interrupt input. Initiates an interrupt if interrupt is enabled. Interrupt is disabled after a reset. Also testable with conditional jump instruction. (Active low) interrupt must remain low for at least 3 machine cycles for proper operation.	All

Table 1. Pin Description (Continued)

Symbol	Pin No.	Function	Device
$\overline{RD}$	8	Output strobe activated during a BUS read. Can be used to enable data onto the bus from an external device. Used as a read strobe to external data memory. (Active low)	All
$\overline{RESET}$	4	Input which is used to initialize the processor. (Active low) (Non TTL V_{IH})	All
		Used during power down.	8048AH 8035AHL 8049AH 8039AHL 8050AH 8040AHL
		Used during programming.	8748H 8749H
		Used during ROM verification.	8048AH 8748H 8049AH 8749H 8050AH
$\overline{WR}$	10	Output strobe during a bus write. (Active low) Used as write strobe to external data memory.	All
ALE	11	Address latch enable. This signal occurs once during each cycle and is useful as a clock output. The negative edge of ALE strobes address into external data and program memory.	All

Symbol	Pin No.	Function	Device
$\overline{PSEN}$	9	Program store enable. This output occurs only during a fetch to external program memory. (Active low)	ALL
$\overline{SS}$	5	Single step input can be used in conjunction with ALE to "single step" the processor through each instruction.	All
		(Active low) Used in sync mode	8048AH 8035AHL 8049AH 8039AHL 8050AH 8040AHL
EA	7	External access input which forces all program memory fetches to reference external memory. Useful for emulation and debug. (Active high)	All
		Used during (18V) programming	8748H 8749H
		Used during ROM verification (12V)	8048AH 8049AH 8050AH
XTAL1	2	One side of crystal input for internal oscillator. Also input for external source. (Non TTL V_{IH})	All
XTAL2	3	Other side of crystal input.	All

NOTE: On the 8749H, PROG must be clamped to V_{CC} when not programming. A diode should be used when using an 8243; otherwise, a direct connection is permissible.

Table 2. Instruction Set

Accumulator

Mnemonic	Description	Bytes	Cycles
ADD A, R	Add register to A	1	1
ADD A, @R	Add data memory to A	1	1
ADD A, # data	Add immediate to A	2	2
ADDC A, R	Add register with carry	1	1
ADDC A, @R	Add data memory with carry	1	1
ADDC A, # data	Add immediate with carry	2	2
ANL A, R	And register to A	1	1
ANL A, @R	And data memory to A	1	1
ANL A, # data	And immediate to A	2	2
ORL A, R	Or register to A	1	1
ORL A @R	Or data memory to A	1	1
ORL A, # data	Or immediate to A	2	2
XRL A, R	Exclusive or register to A	1	1
XRL A, @R	Exclusive or data memory to A	1	1
XRL, A, # data	Exclusive or immediate to A	2	2
INC A	Increment A	1	1
DEC A	Decrement A	1	1
CLR A	Clear A	1	1
CPL A	Complement A	1	1
DA A	Decimal adjust A	1	1
SWAP A	Swap nibbles of A	1	1
RL A	Rotate A left	1	1
RLC A	Rotate A left through carry	1	1
RR A	Rotate A right	1	1
RRC A	Rotate A right through carry	1	1

Input/Output

Mnemonic	Description	Bytes	Cycles
IN A, P	Input port to A	1	2
OUTL P, A	Output A to port	1	2
ANL P, # data	And immediate to port	2	2
ORL P, # data	Or immediate to port	2	2
INS A, BUS	Input BUS to A	1	2
OUTL BUS, A	Output A to BUS	1	2
ANL BUS, # data	And immediate to BUS	2	2
ORL BUS, # data	Or immediate to BUS	2	2
MOVD A, P	Input expander port to A	1	2
MOVD P, A	Output A to expander port	1	2
ANLD P, A	And A to expander port	1	2
ORLD P, A	Or A to expander port	1	2

Registers

Mnemonic	Description	Bytes	Cycles
INC R	Increment register	1	1
INC @R	Increment data memory	1	1
DEC R	Decrement register	1	1

Branch

Mnemonic	Description	Bytes	Cycles
JMP addr	Jump unconditional	2	2
JMPP @A	Jump indirect	1	2
DJNZ R, addr	Decrement register and skip	2	2
JC addr	Jump on carry = 1	2	2
JNC addr	Jump on carry = 0	2	2
JZ addr	Jump on A zero	2	2
JNZ addr	Jump on A not zero	2	2
JT0 addr	Jump on T0 = 1	2	2
JNT0 addr	Jump on T0 = 0	2	2
JT1 addr	Jump on T1 = 1	2	2
JNT1 addr	Jump on T1 = 0	2	2
JF0 addr	Jump on F0 = 1	2	2
JF1 addr	Jump on F1 = 1	2	2
JTF addr	Jump on timer flag	2	2
JNI addr	Jump on INT = 0	2	2
JBb addr	Jump on accumulator bit	2	2

Subroutine

Mnemonic	Description	Bytes	Cycles
CALL addr	Jump to subroutine	2	2
RET	Return	1	2
RETR	Return and restore status	1	2

Flags

Mnemonic	Description	Bytes	Cycles
CLR C	Clear carry	1	1
CPL C	Complement carry	1	1
CLR F0	Clear flag 0	1	1
CPL F0	Complement flag 0	1	1
CLR F1	Clear flag 1	1	1
CPL F1	Complement flag 1	1	1

Table 2. Instruction Set (Continued)

Data Moves

Mnemonic	Description	Bytes	Cycles
MOV A, R	Move register to A	1	1
MOV A, @R	Move data memory to A	1	1
MOV A, # data	Move immediate to A	2	2
MOV R, A	Move A to register	1	1
MOV @R, A	Move A to data memory	1	1
MOV R, # data	Move immediate to register	2	2
MOV @R, # data	Move immediate to data memory	2	2
MOV A, PSW	Move PSW to A	1	1
MOV PSW, A	Move A to PSW	1	1
XCH A, R	Exchange A and register	1	1
XCH A, @R	Exchange A and data memory	1	1
XCHD A, @R	Exchange nibble of A and register	1	1
MOVX A, @R	Move external data memory to A	1	2
MOVX @R, A	Move A to external data memory	1	2
MOVP A, @A	Move to A from current page	1	2
MOVP3 A, @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 I	Enable external interrupt	1	1
DIS I	Disable external interrupt	1	1
SEL RB0	Select register bank 0	1	1
SEL RB1	Select register bank 1	1	1
SEL MB0	Select memory bank 0	1	1
SEL MB1	Select memory bank 1	1	1
ENT0 CLK	Enable clock output on T0	1	1

Mnemonic	Description	Bytes	Cycles
NOP	No operation	1	1

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	1.0 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of device at these or any other conditions above those indicated in the operational sections of this specification is not implied.*

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

Symbol	Parameter	Limits			Unit	Test Conditions	Device
		Min	Typ	Max			
V_{IL}	Input Low Voltage (All Except RESET, X1, X2)	-.5		.8	V		All
V_{IL1}	Input Low Voltage ($\overline{\text{RESET}}$, X1, X2)	-.5		.6	V		All
V_{IH}	Input High Voltage (All Except XTAL1, XTAL2, $\overline{\text{RESET}}$)	2.0		V_{CC}	V		All
V_{IH1}	Input High Voltage (X1, X2, $\overline{\text{RESET}}$)	3.8		V_{CC}	V		All
V_{OL}	Output Low Voltage (BUS)			.45	V	I_{OL} = 2.0 mA	All
V_{OL1}	Output Low Voltage ($\overline{\text{RD}}$, $\overline{\text{WR}}$, $\overline{\text{PSEN}}$, ALE)			.45	V	I_{OL} = 1.8 mA	All
V_{OL2}	Output Low Voltage (PROG)			.45	V	I_{OL} = 1.0 mA	All
V_{OL3}	Output Low Voltage (All Other Outputs)			.45	V	I_{OL} = 1.6 mA	All
V_{OH}	Output High Voltage (BUS)	2.4			V	I_{OH} = -400 μA	All
V_{OH1}	Output High Voltage ($\overline{\text{RD}}$, $\overline{\text{WR}}$, $\overline{\text{PSEN}}$, ALE)	2.4			V	I_{OH} = -100 μA	All
V_{OH2}	Output High Voltage (All Other Outputs)	2.4			V	I_{OH} = -40 μA	All

D.C. CHARACTERISTICS: (T_A = 0°C to 70°C; $V_{CC} = V_{DD}$ = 5V ± 10%; V_{SS} = 0V) (Continued)

Symbol	Parameter	Limits			Unit	Test Conditions	Device
		Min	Typ	Max			
I_{L1}	Leakage Current (T1, $\overline{\text{INT}}$)			±10	μA	$V_{SS} \leqslant V_{IN} \leqslant V_{CC}$	All
I_{LI1}	Input Leakage Current (P10-P17, P20-P27, EA, $\overline{\text{SS}}$)			-500	μA	$V_{SS}+.45 \leqslant V_{IN} \leqslant V_{CC}$	All
I_{LI2}	Input Leakage Current $\overline{\text{RESET}}$	−10		-300	μA	$V_{SS} \leqslant V_{IN} \leqslant V_{IH1}$	All
I_{LO}	Leakage Current (BUS, T0) (High Impedance State)			±10	μA	$V_{SS} \leqslant V_{IN} \leqslant V_{CC}$	All
I_{DD}	V_{DD} Supply Current (RAM Standby)		3	5	mA		8048AH 8035AHL
			4	7	mA		8049AH 8039AHL
			5	10	mA		8050AH 8040AHL
$I_{DD} + I_{CC}$	Total Supply Current		30	65	mA		8048AH 8035AHL
			35	70	mA		8049AH 8039AHL
			40	80	mA		8050AH 8040AHL
			30	90	mA		**8748H**
			50	110	mA		**8749H**
V_{DD}	RAM Standby Voltage	2.2		5.5	V	Standby Mode Reset $\leqslant V_{IL1}$	8048AH 8035AH
		2.2		5.5	V		8049AH 8039AH
		2.2		5.5	V		8050AH 8040AH

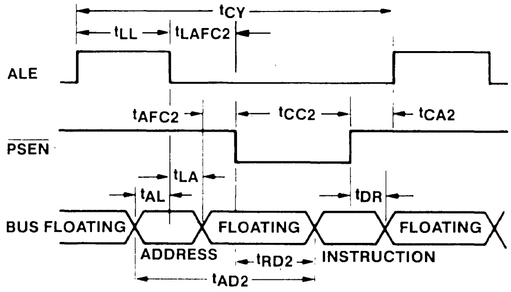
A.C. CHARACTERISTICS: (T_A = 0°C to 70°C; V_{CC} = V_{DD} = 5V ± 10%; V_{SS} = 0V)

Symbol	Parameter	f (t) (Note 3)	11 MHz Min	11 MHz Max	Unit	Conditions (Note 1)
t	Clock Period	1/xtal freq	90.9	1000	ns	(Note 3)
t_{LL}	ALE Pulse Width	3.5t-170	150		ns	
t_{AL}	Addr Setup to ALE	2t-110	70		ns	(Note 2)
t_{LA}	Addr Hold from ALE	t-40	50		ns	
t_{CC1}	Control Pulse Width ($\overline{RD}$, $\overline{WR}$)	7.5t-200	480		ns	
t_{CC2}	Control Pulse Width ($\overline{PSEN}$)	6t-200	350		ns	
t_{DW}	Data Setup before $\overline{WR}$	6.5t-200	390		ns	
t_{WD}	Data Hold after $\overline{WR}$	t-50	40		ns	
t_{DR}	Data Hold ($\overline{RD}$, $\overline{PSEN}$)	1.5t-30	0	110	ns	
t_{RD1}	$\overline{RD}$ to Data in	6t-170		350	ns	
t_{RD2}	$\overline{PSEN}$ to Data in	4.5t-170		190	ns	
t_{AW}	Addr Setup to $\overline{WR}$	5t-150	300		ns	
t_{AD1}	Addr Setup to Data ($\overline{RD}$)	10.5t-220		730	ns	
t_{AD2}	Addr Setup to Data ($\overline{PSEN}$)	7.5t-200		460	ns	
t_{AFC1}	Addr Float to $\overline{RD}$, $\overline{WR}$	2t-40	140		ns	(Note 2)
t_{AFC2}	Addr Float to $\overline{PSEN}$	.5t-40	10		ns	(Note 2)
t_{LAFC1}	ALE to Control ($\overline{RD}$, $\overline{WR}$)	3t-75	200		ns	
t_{LAFC2}	ALE to Control ($\overline{PSEN}$)	1.5t-75	60		ns	
t_{CA1}	Control to ALE ($\overline{RD}$, $\overline{WR}$, $\overline{PROG}$)	t-40	50		ns	
t_{CA2}	Control to ALE ($\overline{PSEN}$)	4t-40	320		ns	
t_{CP}	Port Control Setup to $\overline{PROG}$	1.5t-80	50		ns	
t_{PC}	Port Control Hold to $\overline{PROG}$	4t-260	100		ns	
t_{PR}	$\overline{PROG}$ to P2 Input Valid	8.5t-120		650	ns	
t_{PF}	Input Data Hold from $\overline{PROG}$	1.5t	0	140	ns	
t_{DP}	Output Data Setup	6t-290	250		ns	
t_{PD}	Output Data Hold	1.5t-90	40		ns	
t_{PP}	$\overline{PROG}$ Pulse Width	10.5t-250	700		ns	
t_{PL}	Port 2 I/O Setup to ALE	4t-200	160		ns	
t_{LP}	Port 2 I/O Hold to ALE	1.5t-120	15		ns	
t_{PV}	Port Output from ALE	4.5t+100		510	ns	
t_{0PRR}	T0 Rep Rate	3t	270		ns	
t_{CY}	Cycle Time	15t	1.36	15.0	μs	

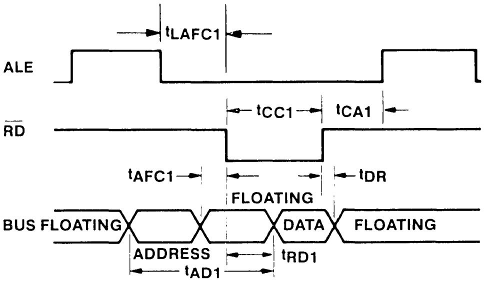
Notes:

1. Control Outputs CL = 80pF
   BUS Outputs CL = 150pF
2. BUS High Impedance
   Load 20pF
3. f(t) assumes 50% duty cycle on X1, X2. Max clock period is for a 1 MHz crystal input.

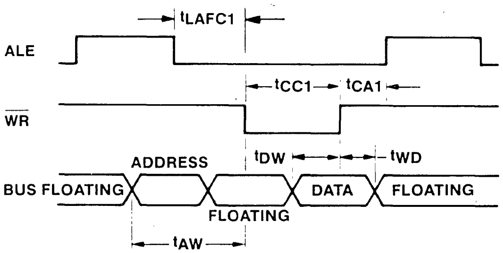
WAVEFORMS



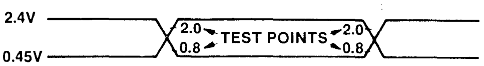
Instruction Fetch From Program Memory



Read From External Data Memory



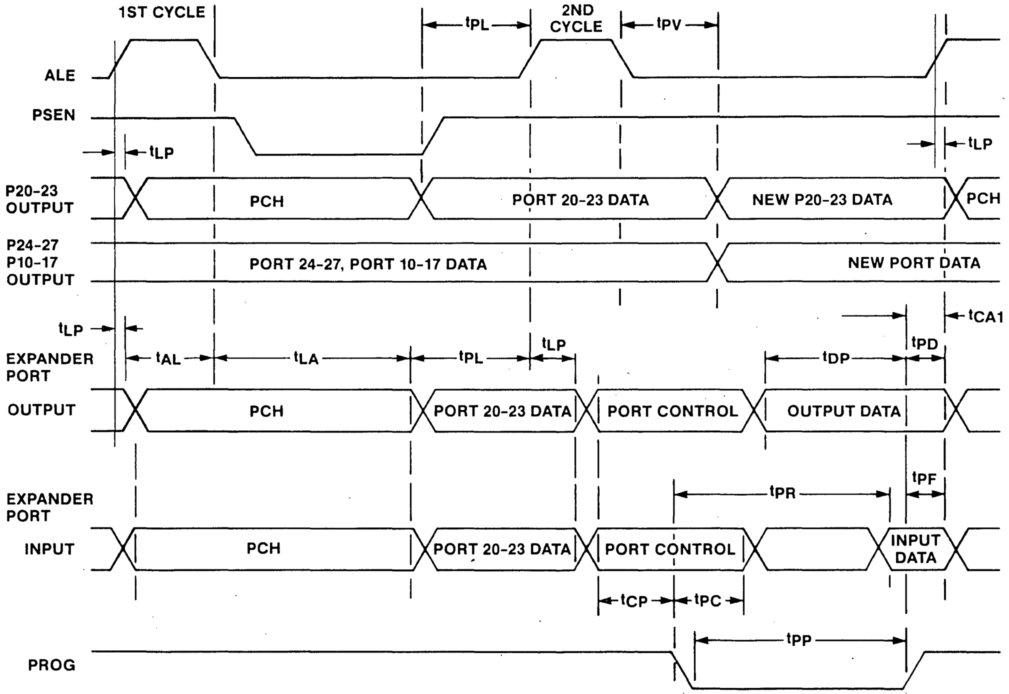
Write To External Data Memory



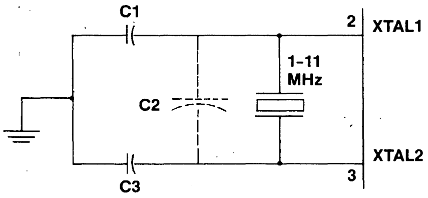
A.C. testing inputs are driven at 2.4V for a logic "1" and 0.45V for a logic "0." Output timing measurements are made at 2.0V for a logic "1" and 0.8V for a logic "0."

Input And Output For A.C. Tests

PORT 1/PORT 2 TIMING



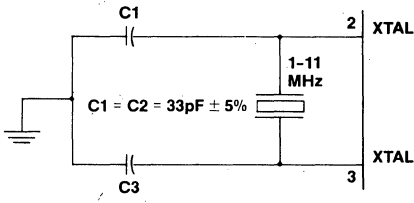
CRYSTAL OSCILLATOR MODE



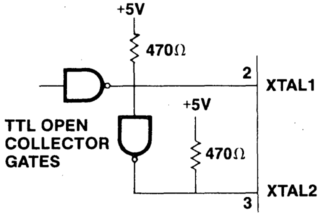
C1 = 5pF ± 1/2pF + (STRAY < 5pF)
C2 = (CRYSTAL + STRAY) < 8pF
C3 = 20pF ± 1pF + (STRAY < 5pF)

Crystal series resistance should be less than 30Ω at 11 MHz; less than 75Ω at 6 MHz; less than 180Ω at 3.6 MHz.

CERAMIC RESONATOR MODE



DRIVING FROM EXTERNAL SOURCE



For XTAL1 and XTAL2 define "high" as voltages above 1.6V and "low" as voltages below 1.6V. The duty cycle requirements for externally driving XTAL1 and XTAL2 using the circuit shown above are as follows: XTAL1 must be high 35-65% of the period and XTAL2 must be high 36-65% of the period. Rise and fall times must be faster than 20 nS.

PROGRAMMING, VERIFYING, AND ERASING THE 8749H (8748H) EPROM

Programming Verification

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

Pin	Function
XTAL1	Clock Input (3 to 4.0 MHz)
Reset	Initialization and Address Latching
Test 0	Selection of Program or Verify Mode
EA	Activation of Program/Verify Modes
BUS	Address and Data Input Data Output During Verify
P20-P22	Address Input
V_{DD}	Programming Power Supply
PROG	Program Pulse Input

WARNING:
An attempt to program a missocketed 8749H (8748H) will result in severe damage to the part. An indication of a properly socketed part is the appearance of the ALE clock output. The lack of this clock may be used to disable the programmer.

The Program/Verify sequence is:

1. V_{DD} = 5V, Clock applied or internal oscillator operating. RESET = 0V, TEST 0 = 5V, EA = 5V, BUS and PROG floating. P10 and P11 must be tied to ground.
2. Insert 8749H (8748H) in programming socket.
3. TEST 0 = 0V (select program mode)
4. EA = 18V (activate program mode)
5. Address applied to BUS and P20-22
6. RESET = 5V (latch address)
7. Data applied to BUS
8. V_{DD} = 21V (programming power)
9. PROG = V_{CC} or float followed by one 50ms pulse to 18V
10. V_{DD} = 5V
11. TEST 0 = 5V (verify mode)
12. Read and verify data on BUS
13. TEST 0 = 0V
14. RESET = 0V and repeat from step 5
15. Programmer should be at conditions of step 1 when 8749H (8748H) is removed from socket.

A.C. TIMING SPECIFICATION FOR PROGRAMMING 8748H/8749H ONLY:

(T_A = 25° C ± 5° C; V_{CC} = 5V ± 5%; V_{DD} = 21 ± .5V)

Symbol	Parameter	Min	Max	Unit	Test Conditions
t_{AW}	Address Setup Time to $\overline{\text{RESET}}$↑*	$4t_{CY}$			
t_{WA}	Address Hold Time After $\overline{\text{RESET}}$↑*	$4t_{CY}$			
t_{DW}	Data in Setup Time to PROG↑	$4t_{CY}$			
t_{WD}	Data in Hold Time After PROG↓	$4t_{CY}$			
t_{PH}	$\overline{\text{RESET}}$ Hold Time to Verify	$4t_{CY}$			
t_{VDDW}	V_{DD} Hold Time Before PROG↑	0	1.0	ms	
t_{VDDH}	V_{DD} Hold Time After PROG↓	0	1.0	ms	
t_{PW}	Program Pulse Width	50	60	ms	
t_{TW}	Test 0 Setup Time for Program Mode	$4t_{CY}$			
t_{WT}	Test 0 Hold Time After Program Mode	$4t_{CY}$			
t_{DO}	Test 0 to Data Out Delay		$4t_{CY}$		
t_{WW}	$\overline{\text{RESET}}$ Pulse Width to Latch Address*	$4t_{CY}$			
t_r, t_f	V_{DD} and PROG Rise and Fall Times	0.5	100	μs	
t_{CY}	CPU Operation Cycle Time	3.75	5	μs	
t_{RE}	$\overline{\text{RESET}}$ Setup Time before EA↑*	$4t_{CY}$			

NOTE: If Test 0 is high, t_{DO} can be triggered by $\overline{\text{RESET}}$↑. *Valid for 8048AH/8049AH/8050AH also.

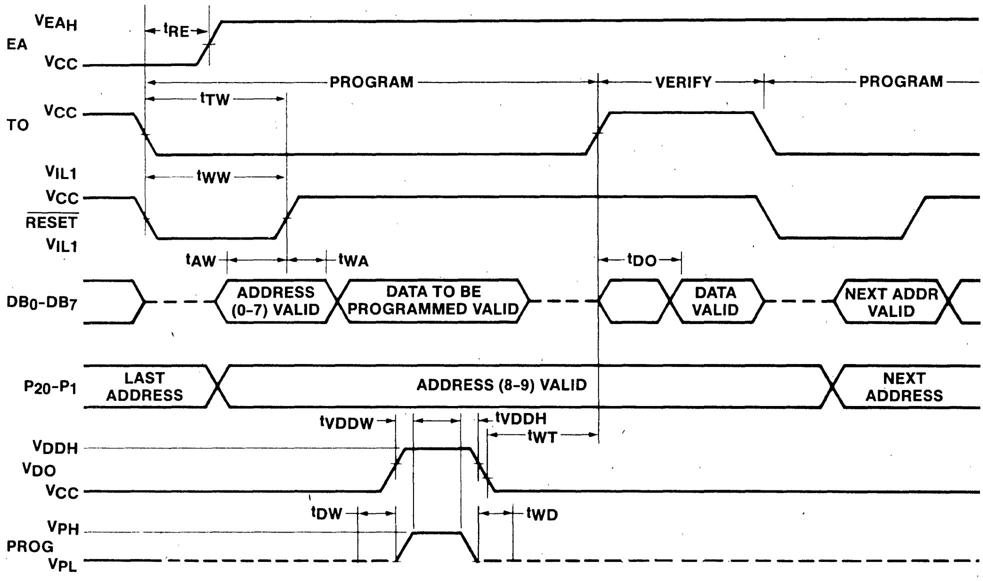
D.C. TIMING SPECIFICATION FOR PROGRAMMING 8748H/8749H ONLY:

(T_A = 25° C ± 5° C; V_{CC} = 5V ± 5%; V_{DD} = 21 ± .5V)

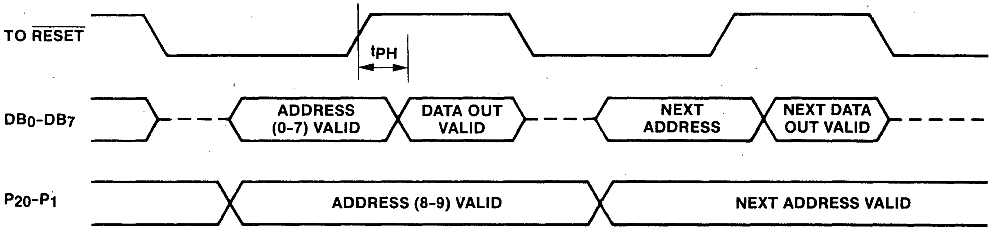
Symbol	Parameter	Min	Max	Unit	Test Conditions
V_{DDH}	V_{DD} Program Voltage High Level	20.5	21.5	V	
V_{DDL}	V_{DD} Voltage Low Level	4.75	5.25	V	
V_{PH}	PROG Program Voltage High Level	17.5	18.5	V	
V_{PL}	PROG Voltage Low Level	4.0	V_{CC}	V	
V_{EAH}	EA Program or Verify Voltage High Level	17.5	18.5	V	
I_{DD}	V_{DD} High Voltage Supply Current		20.0	mA	
I_{PROG}	PROG High Voltage Supply Current		1.0	mA	
I_{EA}	EA High Voltage Supply Current		1.0	mA	

WAVEFORMS

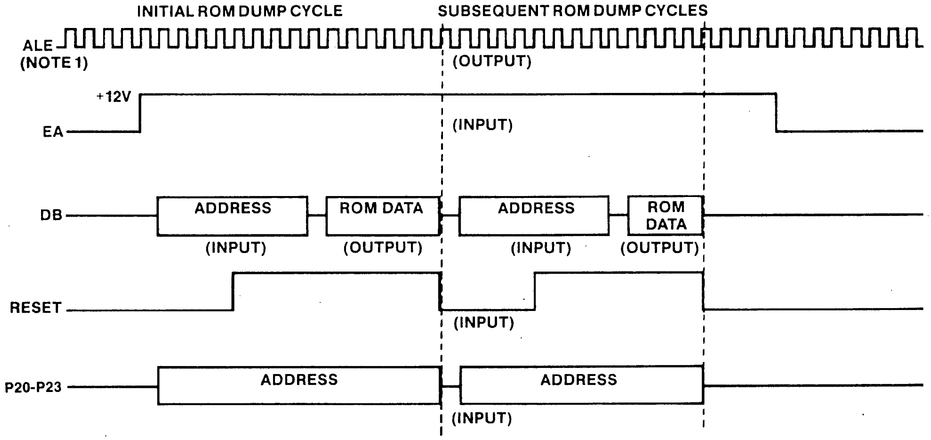
COMBINATION PROGRAM/VERIFY MODE (EPROM'S ONLY)



VERIFY MODE (ROM/EPROM)



SUGGESTED ROM VERIFICATION ALGORITHM FOR H-MOS DEVICE ONLY



	48H	49H	50H
A10	0	ADDR	ADDR
A11	0	0	ADDR

$V_{CC} = V_{DD} = +5V$

$V_{SS} = 0V$

NOTE: ALE is function of X1, X2 inputs.

PRELIMINARY

SINGLE-COMPONENT 8-BIT MICROCOMPUTERS

EXPRESS

- 0°C to 70°C Operation
- -40°C to 85°C Operation
- 168 Hr. Burn-In

- 8048AH/8035AHL
- 8049AH/8039AHL
- 8050AH/8040AHL
- 8748H
- 8243
- 8749H

The new Intel EXPRESS family of single-component 8-bit microcomputers offers enhanced processing options to the familiar 8048AH/8035AHL, 8748H, 8049AH/8039AHL/8749H, 8050AH/8040AHL Intel components. These EXPRESS products are designed to meet the needs of those applications whose operating requirements exceed commercial standards, but fall short of military conditions.

The EXPRESS options include the commercial standard and −40°C to 85°C operation with or without 168 ±8 hours of dynamic burn-in at 125°C per MIL-STD-883B, method 1015. Figure 1 summarizes the option marking designators and package selections.

For a complete description of 8048AH/8035AHL, 8748H, 8049AH/8309AHL features and operating characteristics, refer to the respective standard commercial grade data sheet. This document highlights only the electrical specifications which differ from the respective commercial part.

Temp Range C°	0-70	-40-85	0-70	-40-85
	0	0	168	168
	P8048AH	TP8048AH	QP8048AH	—
	D8048AH	TD8048AH	QD8048AH	LD8048AH
	D8748H	TD8748H	QD8748H	LD8748H
	P8035AHL	TP8035AHL	QP8035AHL	—
	D8035AHL	TD8035AHL	QD8035AHL	LD8035AHL
	P8049AH	TP8049AH	QP8049AH	—
	D8049AH	TD8049AH	QD8049AH	LD8049AH
	D8749H	TD8749H	QD8749H	LD8749H
	P8039AHL	TP8039AHL	QP8039AHL	—
	D8039AHL	TD8039AHL	QD8039AHL	LD8039AHL
	P8050AH	TP8050AH	QP8050AH	—
	D8050AH	TD8050AH	QD8050AH	LD8050AH
	P8040AHL	TP8040AHL	QP8040AHL	—
	D8040AHL	TD8040AHL	QD8040AHL	LD8040AHL
	P8243	TP8243	QP8243	—
	D8243	TD8243	QD8243	LD8243

* Commercial Grade
P Plastic Package
D Cerdip Package

*Extended Temperature Electrical Specification Deviations**

TD8048AH/TD8035AHL/LD8048AH/LD8035AHL

D.C. CHARACTERISTICS: (T_A = -40° C to 85° C; V_{CC} = V_{DD} = 5V ± 10%; V_{SS} = 0V)

Symbol	Parameter	Limits			Unit	Test Conditions
		Min	Typ	Max		
V_{IH}	Input High Voltage (All Except XTAL1, XTAL2, $\overline{\text{RESET}}$)	2.2		V_{CC}	V	
I_{DD}	V_{DD} Supply Current		4	8	mA	
I_{DD} + I_{CC}	Total Supply Current		40	80	mA	

TD8049AH/TD8039AHL/LD8049AH/LD8039AHL

D.C. CHARACTERISTICS: (T_A = -40° C to 85° C; V_{CC} = V_{DD} = 5V ± 10%; V_{SS} = 0V)

Symbol	Parameter	Limits			Unit	Test Conditions
		Min	Typ	Max		
V_{IH}	Input High Voltage (All Except XTAL1, XTAL2, $\overline{\text{RESET}}$)	2.2		V_{CC}	V	
I_{DD}	V_{DD} Supply Current		5	10	mA	
I_{DD} + I_{CC}	Total Supply Current		50	100	mA	

TD8050AH/TD8040AHL/LD8050AH/LD8040AHL

D.C. CHARACTERISTICS: (T_A = -40° C to 85° C; V_{CC} = V_{DD} = 5V ± 10%; V_{SS} = 0V)

Symbol	Parameter	Limits			Unit	Test Conditions
		Min	Typ	Max		
V_{IH}	Input High Voltage (All Except XTAL1, XTAL2, $\overline{\text{RESET}}$)	2.2		V_{CC}	V	
I_{DD}	V_{DD} Supply Current		10	20	mA	
I_{DD} + I_{CC}	Total Supply Current		75	120	mA	

*Extended Temperature Electrical Specification Deviations**

TD8748H/LD8748H

D.C. CHARACTERISTICS: (T_A = -40° C to 85° C; V_{CC} = V_{DD} = 5V ± 10%; V_{SS} = 0V)

Symbol	Parameter	Limits			Unit	Test Conditions
		Min	Typ	Max		
V_{IH}	Input High Voltage (All Except XTAL1, XTAL2, $\overline{\text{RESET}}$)	2.2		V_{CC}	V	
I_{DD} + I_{CC}	Total Supply Current		50	130	mA	

TD8749H/LD8749H

D.C. CHARACTERISTICS: (T_A = -40° C to 85° C; V_{CC} = V_{DD} = 5V ± 10%; V_{SS} = 0V)

Symbol	Parameter	Limits			Unit	Test Conditions
		Min	Typ	Max		
V_{IH}	Input High Voltage (All Except XTAL1, XTAL2, $\overline{\text{RESET}}$)	2.2		V_{CC}	V	
I_{DD} + I_{CC}	Total Supply Current		75	150	mA	

TP8243/TO8243/LD8243

D.C. CHARACTERISTICS: (T_A = −40°C to 85°C; V_{CC} = 5V ± 10%; V_{SS} = 0V)

Symbol	Parameter	Limits			Unit	Test Conditions
		Min	Typ	Max		
I_{CC}	V_{CC} Supply Current		15	25	mA	

*Extended Temperature Electrical Specification Deviations**

TD8022/LD8022

D.C. CHARACTERISTICS: (T_A = -40°C to 85°C; V_{CC} = 5.5V ± 1V; V_{SS} = 0V)

Symbol	Parameter	Limits			Unit	Test Conditions
		Min	Typ	Max		
V_{IL1}	Input Low Voltage (Port 0)	-0.5		V_{TH}-0.2	V	
V_{IH}	Input High Voltage (All Except XTAL1, RESET)	2.3		V_{CC}	V	V_{CC} = 5.0V ± 10% V_{TH} Floating
V_{IH1}	Input High Voltage (All Except XTAL1, RESET)	3.8		V_{CC}	V	V_{CC} = 5.5V ± 1V V_{TH} Floating
V_{IH2}	Input High Voltage (Port 0)	V_{TH}+0.2		V_{CC}	V	
V_{IH3}	Input High Voltage (RESET, XTAL1)	3.8		V_{CC}	V	
V_{IL}	Input Low Voltage	-0.5		0.5	V	
V_{OL}	Output Low Voltage			0.45	V	I_{OL} = 0.8 mA
V_{OL1}	Output Low Voltage (P10, P11)			2.5	V	I_{OL} = 3 mA
V_{OH}	Output High Voltage (All unless open drain option Port 0)	2.4			V	I_{OH} = 30 μA
I_{LI}	Input Current (T1)			±700	μA	$V_{CC} \geqslant V_{IN} \geqslant V_{SS}$ + 0.45V
I_{LI1}	Input Current to Ports			500	μA	V_{IN} = 0.45V
I_{CC}	V_{CC} Supply Current			120	mA	

A.C. CHARACTERISTICS: (T_A = -40°C to 85°C; V_{CC} = 5.5V ± 1V; V_{SS} = 0V)

Symbol	Parameter	Min	Max	Unit	Test Conditions
t_{CY}	Cycle Time	8.38	50.0	μs	3.58 MHz XTAL = 8.38 μs t_{CY}
V_{T1}	Zero-Cross Detection Input (T1)	1	3	VACpp	AC Coupled
A_{ZC}	Zero-Cross Accuracy		±200	mV	60 Hz Sine Wave
F_{T1}	Zero-Cross Detection Input Frequency (T1)	0.05	1	kHz	
t_{LL}	ALE Pulse Width	3.9	23.0	μs	t_{CY} = 8.38 μs for min

NOTE: Control Outputs: C_L = 80 pf; T_{CY} = 8.38 μsec.

A/D CONVERTER CHARACTERISTICS: (AVCC = 5.5V ± 1V; AVSS = 0V; AVCC/2 ⩽ VAREF ⩽ AVCC)

Parameter	Limits			Unit	Test Conditions
	Min	Typ	Max		
Absolute Accuracy			1.6% FSR ± ½ LSB	LSB	

NOTE: The analog input must be maintained at a constant voltage during the sample time (tss + tsh).
*Refer to individual commercial grade data sheets for complete operating characteristics.

PRELIMINARY

80C49-7/80C39-7
CHMOS SINGLE-COMPONENT 8-BIT MICROCOMPUTER

- **80C49-7 Low Power Mask Programmable ROM**
- **80C39-7 Low Power, CPU only**

- **Pin-to-pin Compatible with Intel's 8049AH/8039AHL**
- **1.36 μsec Instruction Cycle. All Instructions 1 or 2 Cycles**
- **Ability to Maintain Operation during AC Power Line Interruptions**
- **Exit Idle Mode with an External or Internal Interrupt Signal**
- **Battery Operation**
- **3 Power Consumption Selections**
  —Normal Operation: 12 mA @ 11 MHz @ 5V
  —Idle Mode: 5 mA @ 11 MHz @ 5V
  —Power Down: 2 μA @ 2.0V
- **11 MHz, TTL Compatible Operation: V_{CC} = 5V ± 10%**
  CMOS Compatible Operation; V_{CC} = 5V ± 20%

Intel's 80C49-7/80C39-7 are low power, CHMOS versions of the popular MCS®-48 HMOS family members. CHMOS is a technology built on HMOS II and features high resistivity P substrate, diffused N well, and scaled N and P channel devices. The 80C49-7/80C39-7 have been designed to provide low power consumption and high performance.

The 80C49-7 contains a 2K x 8 program memory, a 128 x 8 x 8 RAM data memory, 27 I/O lines, and an 8-bit timer/counter in addition to an on-board oscillator and clock circuits. For systems that require extra capability, the 80C49-7 can be expanded using CMOS external memories and MCS®-80 and MCS®-85 peripherals. The 80C39-7 is the equivalent of the 80C49-7 without program memory on-board.

The CHMOS design of the 80C49-7 opens new application areas that require battery operation, low power standby, wide voltage range, and the ability to maintain operation during AC power line interruptions. These applications include portable and hand-held instruments, telecommunications, consumer, and automotive.

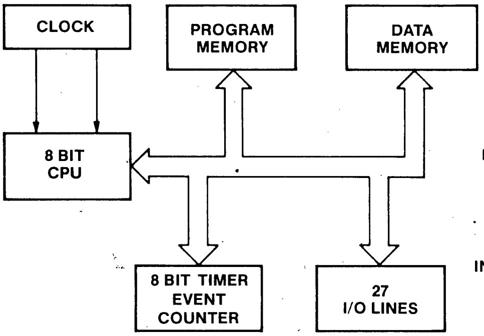


Figure 1. Block Diagram

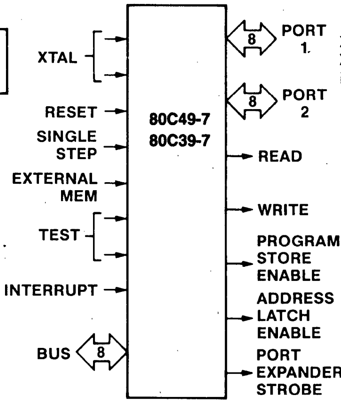


Figure 2. Logic Symbol

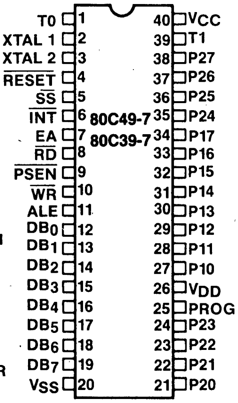


Figure 3. Pin Configuration

Intel Corporation Assumes No Responsibility for the Use of Any Circuitrv Other Than Circuitry Embodied in an Intel Product. No Other Circuit Patent Licenses are Implied.

January 1983

Order Number 210936-001

Table 1. Pin Description

Symbol	Pin No.	Function
V_{SS}	20	Circuit GND potential
V_{DD}	26	Low Power standby pin
V_{CC}	40	Main power supply; +5V during operation.
PROG	25	Output strobe for 82C43 I/O expander.
P10–P17 Port 1	27-34	8-bit quasi-bidirectional port.
P20–P23	21-24	8-bit quasi-bidirectional port.
P24–P27 Port 2	35-38	P20-P23 contain the four high order program counter bits during an external program memory fetch and serve as a 4-bit I/O expander bus for 8243.
DB0-DB7 BUS	12-19	True bidirectional port which can be written or read synchronously using the RD, WR strobes. The port can also be statically latched. Contains the 8 low order program counter bits during an external program memory fetch, and receives the addressed instruction under the control of PSEN. Also contains the address and data during an external RAM data store instruction, under control of ALE, RD, and WR.
T0	1	Input pin testable using the conditional transfer instructions JT0 and JNTo. T0 can be designated as a clock output using ENT0 CLK instruction.
T1	39	Input pin testable using the JT1, and JNT1 instructions.
		Can be designated the timer/counter input using the STRT CNT instruction.
INT	6	Interrupt input. Initiates an interrupt if interrupt is enabled. Interrupt is disabled after a reset. Also testable with conditional jump instruction. (Active low) Interrupt must remain low for at least 3 machine cycles for proper operation.
RD	8	Output strobe activated during a BUS read. Can be used to enable data onto toe bus from an external device. Used as a read strobe to external data memory. (Active low)
RESET	4	Input which is used to initialize the processor. (Active low) (Non TTL V_{IH})
WR	10	Output strobe during a bus write. (Active low) Used as write strobe to external data memory.
ALE	11	Address latch enable. This signal occurs once during each cycle and is useful as a clock output. The negative edge of ALE strobes address into external data and program memory.
PSEN	9	Program store enable. This output occurs only during a fetch to external program memory. (Active low)
SS	5	Single step input can be used in conjunction with

Table 1. Pin Description (Continued)

Symbol	Pin No.	Function
SS (Con't)		ALE to "single step" the processor through each instruction (Active low)
EA	7	External access input which forces all program memory fetches to reference external memory. Useful for emulation and debug, and essential for testing
		and program verification. (Active high)
XTAL1	2	One side of crystal input for internal oscillator. Also input for external source. (Non TTL V_{IH})
XTAL2	3	Other side of crystal input.

IDLE MODE DESCRIPTION

The 80C49-7, when placed into Idle mode, keeps the oscillator, the internal timer and the external interrupt and counter pins functioning and maintains the internal register and RAM status.

To place the 80C49-7 in Idle mode, a command instruction (op code 01H) is executed. To terminate Idle mode, a reset must be performed or interrupts must be enabled and an interrupt signal generated. There are two interrupt sources that can restore normal operation. One is an external signal applied to the interrupt pin. The other is from the overflow of the timer/counter. When either interrupt is invoked, the CPU is taken out of Idle mode and vectors to the interrupt's service routine address. Along with the Idle mode, the standard MCS®-48 power-down mode is still maintained.

Table 2. Instruction Set

Accumulator

Mnemonic	Description	Bytes	Cycles
ADD A, R	Add register to A	1	1
ADD A, @R	Add data memory to A	1	1
ADD A, # data	Add immediate to A	2	2
ADDC A, R	Add register with carry	1	1
ADDC A, @R	Add data memory with carry	1	1
ADDC A, # data	Add immediate with carry	2	2
ANL A, R	And register to A	1	1
ANL A, @R	And data memory to A	1	1
ANL A, # data	And immediate to A	2	2
ORL A, R	Or register to A	1	1
ORL A @R	Or data memory to A	1	1
ORL A, # data	Or immediate to A	2	2
XRL A, R	Exclusive or register to A	1	1
XRL A, @R	Exclusive or data memory to A	1	1
XRL, A, # data	Exclusive or immediate to A	2	2
INC A	Increment A	1	1
DEC A	Decrement A	1	1
CLR A	Clear A	1	1
CPL A	Complement A	1	1
DA A	Decimal adjust A	1	1
SWAP A	Swap nibbles of A	1	1
RL A	Rotate A left	1	1
RLC A	Rotate A left through carry	1	1
RR A	Rotate A right	1	1
RRC A	Rotate A right through carry	1	1

Input/Output

Mnemonic	Description	Bytes	Cycles
IN A, P	Input port to A	1	2
OUTL P, A	Output A to port	1	2
ANL P, # data	And immediate to port	2	2
ORL P, # data	Or immediate to port	2	2
INS A, BUS	Input BUS to A	1	2
OUTL BUS, A	Output A to BUS	1	2
ANL BUS, # data	And immediate to BUS	2	2
ORL BUS, # data	Or immediate to BUS	2	2
MOVD A, P	Input expander port to A	1	2
MOVD P, A	Output A to expander port	1	2
ANLD P, A	And A to expander port	1	2
ORLD P, A	Or A to expander port	1	2

Registers

Mnemonic	Description	Bytes	Cycles
INC R	Increment register	1	1
INC @R	Increment data memory	1	1
DEC R	Decrement register	1	1

Branch

Mnemonic	Description	Bytes	Cycles
JMP addr	Jump unconditional	2	2
JMPP @A	Jump indirect	1	2
DJNZ R, addr	Decrement register and skip	2	2
JC addr	Jump on carry = 1	2	2
JNC addr	Jump on carry = 0	2	2
JZ addr	Jump on A zero	2	2
JNZ addr	Jump on A not zero	2	2
JT0 addr	Jump on T0 = 1	2	2
JNT0 addr	Jump on T0 = 0	2	2
JT1 addr	Jump on T1 = 1	2	2
JNT1 addr	Jump on T1 = 0	2	2
JF0 addr	Jump on F0 = 1	2	2
JF1 addr	Jump on F1 = 1	2	2
JTF addr	Jump on timer flag	2	2
JNI addr	Jump on INT = 0	2	2
JBb addr	Jump on accumulator bit	2	2

Subroutine

Mnemonic	Description	Bytes	Cycles
CALL addr	Jump to subroutine	2	2
RET	Return	1	2
RETR	Return and restore status	1	2

Flags

Mnemonic	Description	Bytes	Cycles
CLR C	Clear carry	1	1
CPL C	Complement carry	1	1
CLR F0	Clear flag 0	1	1
CPL F0	Complement flag 0	1	1
CLR F1	Clear flag 1	1	1
CPL F1	Complement flag 1	1	1

Table 2. Instruction Set (Continued)

Data Moves

Mnemonic	Description	Bytes	Cycles
MOV A, R	Move register to A	1	1
MOV A, @R	Move data memory to A	1	1
MOV A, # data	Move immediate to A	2	2
MOV R, A	Move A to register	1	1
MOV @R, A	Move A to data memory	1	1
MOV R, # data	Move immediate to register	2	2
MOV @R, # data	Move immediate to data memory	2	2
MOV A, PSW	Move PSW to A	1	1
MOV PSW, A	Move A to PSW	1	1
XCH A, R	Exchange A and register	1	1
XCH A, @R	Exchange A and data memory	1	1
XCHD A, @R	Exchange nibble of A and register	1	1
MOVX A, @R	Move external data memory to A	1	2
MOVX @R, A	Move A to external data memory	1	2
MOVP A, @A	Move to A from current page	1	2
MOVP3 A, @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 I	Enable external interrupt	1	1
DIS I	Disable external interrupt	1	1
SEL RB0	Select register bank 0	1	1
SEL RB1	Select register bank 1	1	1
SEL MB0	Select memory bank 0	1	1
SEL MB1	Select memory bank 1	1	1
ENT0 CLK	Enable clock output on T0	1	1

Mnemonic	Description	Bytes	Cycles
NOP	No operation	1	1
IDL	Select Idle Operation	1	1

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias . . . 0° C to 70° C
Storage Temperature . . . . -65° C to +150° C
Voltage On Any Pin With Respect
to Ground . . . . -0.5V to V_{CC}+1V
Maximum Voltage On Any Pin
With Respect to Ground 7V
Power Dissipation . . . . 1.0 Watt

NOTICE: *Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of device at these or any other conditions above those indicated in the operational sections of this specification is not implied.*

D.C. CHARACTERISTICS: (T_A = 0° C to 70° C; $V_{CC} = V_{DD}$ = 5V ± 20%; $|V_{CC} - V_{DD}| \leqslant 1.5V$; V_{SS} = 0V)

Symbol	Parameter	Limits			Unit	Test Conditions
		Min	Typ	Max		
V_{IL}	Input Low Voltage (All Except X1, $\overline{RESET}$)	-.5		.18 V_{CC}	V	
V_{IL1}	Input Low Voltage X1, $\overline{RESET}$	-5		.13 V_{CC}	V	
V_{IH}	Input High Voltage (All Except XTAL1, $\overline{RESET}$)	.4 V_{CC}		V_{CC}	V	
V_{IH1}	Input High Voltage (X1, $\overline{RESET}$)	.7 V_{CC}		V_{CC}	V	
V_{OL}	Output Low Voltage (BUS)			.6	V	I_{OL} = 2.0 mA
V_{OL1}	Output Low Voltage ($\overline{RD}$, $\overline{WR}$, $\overline{PSEN}$, ALE)			.6	V	I_{OL} = 1.8 mA
V_{OL2}	Output Low Voltage (PROG)			.6	V	I_{OL} = 1.0 mA
V_{OL3}	Output Low Voltage (All Other Outputs)			.6	V	I_{OL} = 1.6 mA
V_{OH}	Output High Voltage (BUS)	.75 V_{CC}			V	I_{OH} = -400 μA
V_{OH1}	Output High Voltage ($\overline{RD}$, $\overline{WR}$, $\overline{PSEN}$, ALE)	.75 V_{CC}			V	I_{OH} = -100 μA
V_{OH2}	Output High Voltage (All Other Outputs)	.75 V_{CC}			V	I_{OH} = -40 μA
I_{L1}	Input Leakage Current (T1, $\overline{INT}$, EA)			±5	μA	$V_{SS} \leqslant V_{IN} \leqslant V_{CC}$
I_{LI1}	Input Leakage Current (P10-P17, P20-P27, $\overline{SS}$)			-500	μA	$V_{SS} \leqslant V_{IN} \leqslant V_{CC}$
I_{LO}	Output Leakage Current (BUS, TO) (High Impedance State)			±5	μA	$V_{SS} \leqslant V_{IN} \leqslant V_{CC}$
I_{LR}	Input Leakage Current ($\overline{RESET}$)	−10		-300	μA	$V_{SS} \leqslant V_{IN} \leqslant V_{IH1}$
I_{PD}	Power Down Standby Current			2	μA	V_{DD} = 2.0V $\overline{RESET} \leqslant V_{IL}$

I_{CC} Active Current (mA)

V_{CC}	4V	5V	6V
1 MHz	1.5	2.3	3
6 MHz	5	6.8	8.5
11 MHz	9	12	15

I_{CC} Idle Current (mA)

V_{CC}	4V	5V	6V
1 MHz	.7	1	1.2
6 MHz	2	3	4
11 MHz	3.5	4.8	6

Absolute Maximum Unloaded Current

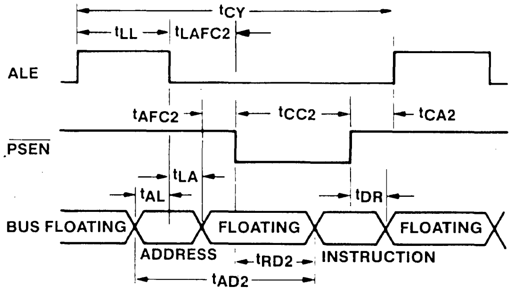
A.C. CHARACTERISTICS: (T_A = 0° C to 70° C; $V_{CC} = V_{DD} = 5V \pm 20\%$; $|V_{CC} - V_{DD}| \leq 1.5V$; $V_{SS} = 0V$)

Symbol	Parameter	f (t) (Note 3)	11 MHz		Unit	Conditions (Note 1)
			Min	Max		
t	Clock Period	1/xtal freq	90.9	1000	ns	(Note 3)
t_{LL}	ALE Pulse Width	3.5t–170	150		ns	
t_{AL}	Addr Setup to ALE	2t–110	70		ns	(Note 2)
t_{LA}	Addr Hold from ALE	t–40	50		ns	
t_{CC1}	Control Pulse Width ($\overline{\text{RD}}$, $\overline{\text{WR}}$)	7.5t–200	480		ns	
t_{CC2}	Control Pulse Width ($\overline{\text{PSEN}}$)	6t–200	350		ns	
t_{DW}	Data Setup before $\overline{\text{WR}}$	6.5t–200	390		ns	
t_{WD}	Data Hold after $\overline{\text{WR}}$	t–50	40		ns	
t_{DR}	Data Hold ($\overline{\text{RD}}$, $\overline{\text{PSEN}}$)	1.5t–30	0	110	ns	
t_{RD1}	$\overline{\text{RD}}$ to Data in	6t–170		350	ns	
t_{RD2}	$\overline{\text{PSEN}}$ to Data in	4.5t–170		190	ns	
t_{AW}	Addr Setup to $\overline{\text{WR}}$	5t–150	300		ns	
t_{AD1}	Addr Setup to Data ($\overline{\text{RD}}$)	10.5t–220		730	ns	
t_{AD2}	Addr Setup to Data ($\overline{\text{PSEN}}$)	7.5t–220		460	ns	
t_{AFC1}	Addr Float to $\overline{\text{RD}}$, $\overline{\text{WR}}$	2t–40	140		ns	(Note 2)
t_{AFC2}	Addr Float to $\overline{\text{PSEN}}$	.5t–40	10		ns	(Note 2)
t_{LAFC1}	ALE to Control ($\overline{\text{RD}}$, $\overline{\text{WR}}$)	3t–75	200		ns	
t_{LAFC2}	ALE to Control ($\overline{\text{PSEN}}$)	1.5t–75	60		ns	
t_{CA1}	Control to ALE ($\overline{\text{RD}}$, $\overline{\text{WR}}$, $\overline{\text{PROG}}$)	t–40	50		ns	
t_{CA2}	Control to ALE ($\overline{\text{PSEN}}$)	4t–40	320		ns	
t_{CP}	Port Control Setup to $\overline{\text{PROG}}$	1.5t–80	50		ns	
t_{PC}	Port Control Hold to $\overline{\text{PROG}}$	4t–260	100		ns	
t_{PR}	$\overline{\text{PROG}}$ to P2 Input Valid	8.5t–120		650	ns	
t_{PF}	Input Data Hold from $\overline{\text{PROG}}$	1.5t	0	140	ns	
t_{DP}	Output Data Setup	6t–290	250		ns	
t_{PD}	Output Data Hold	1.5t–90	40		ns	
t_{PP}	$\overline{\text{PROG}}$ Pulse Width	10.5t–250	700		ns	
t_{PL}	Port 2 I/O Setup to ALE	4t–200	160		ns	
t_{LP}	Port 2 I/O Hold to ALE	1.5t–120	15		ns	
t_{PV}	Port Output from ALE	4.5t+100		510	ns	
t_{0PRR}	T0 Rep Rate	3t	270		ns	
t_{CY}	Cycle Time	15t	1.36	15.0	μs	

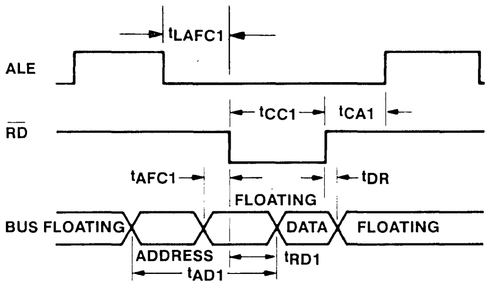
Notes:

1. Control Outputs CL = 80pF
   BUS Outputs CL = 150pF
2. BUS High Impedance
   Load 20pF
3. f(t) assumes 50% duty cycle on X1, X2. Max clock period is for a 1 MHz crystal input.

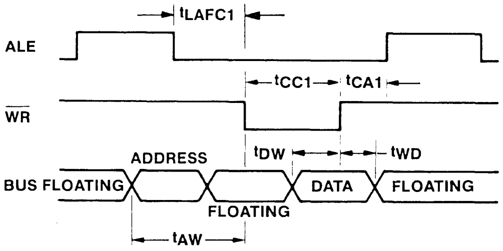
WAVEFORMS



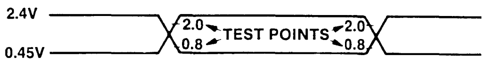
Instruction Fetch From Program Memory



Read From External Data Memory



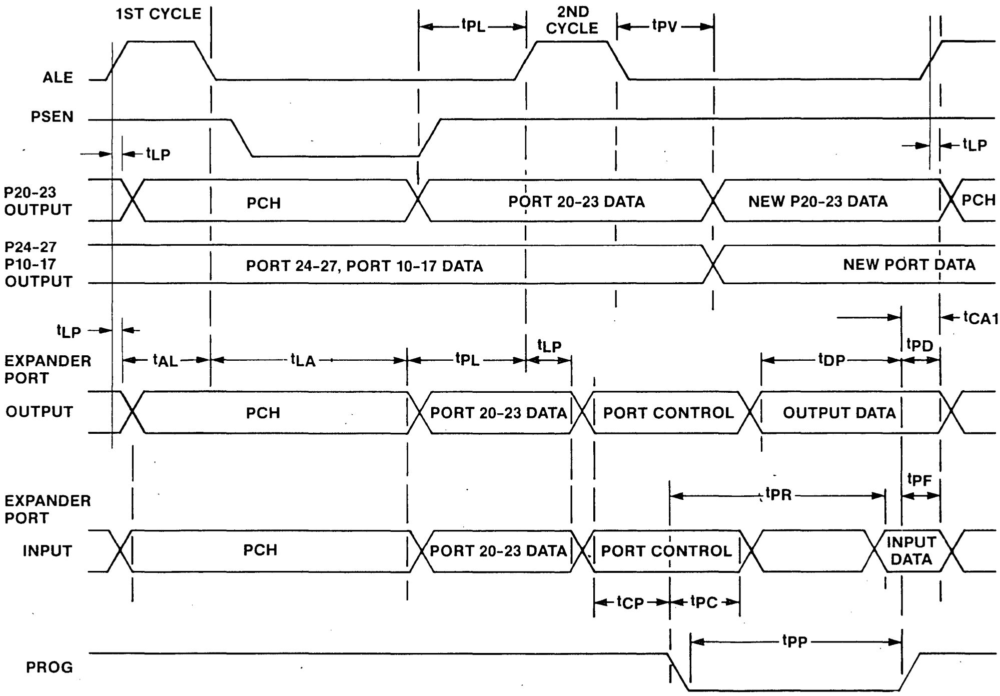
Write To External Data Memory



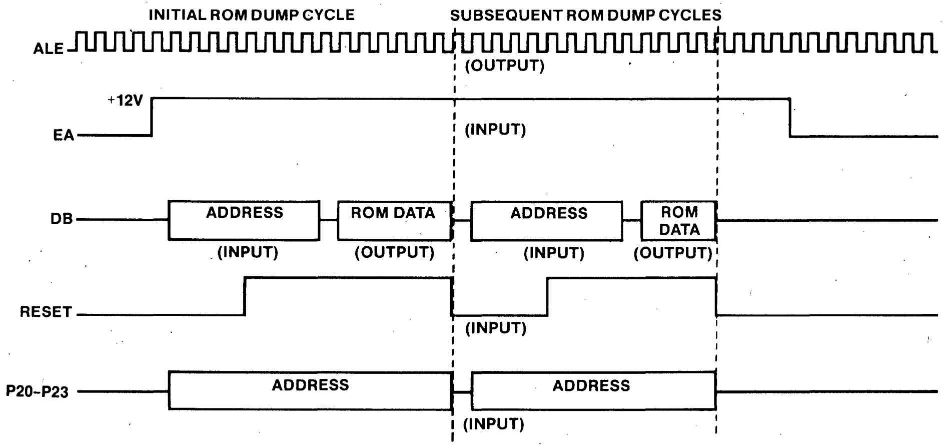
A.C. testing inputs are driven at 2.4V for a logic "1" and 0.45V for a logic "0." Output timing measurements are made at 2.0V for a logic "1" and 0.8V for a logic "0."

Input And Output For A.C. Tests

PORT 1/PORT 2 TIMING



SUGGESTED ROM VERIFICATION ALGORITHM FOR CHMOS DEVICES ONLY



	C48	C49	C50
A10	0	ADDR	ADDR
A11	0	0	ADDR

$V_{CC} = V_{DD} = +5V$

$V_{SS} = 0V$

The RUPI™-44 Family 17

CHAPTER 17
THE RUPI™-44 FAMILY: MICROCONTROLLER WITH ON-CHIP COMMUNICATION CONTROLLER

17.0 INTRODUCTION

The RUPI-44 family is designed for applications requiring local intelligence at remote nodes, and communication capability among these distributed nodes. The RUPI-44 integrates onto a single chip Intel's highest performance microcontroller, the 8051-core, with an intelligent and high performance Serial communication controller, called the Serial Interface Unit, or SIU. See Figure 17-1. This dual controller architecture allows complex control and high speed data communication functions to be realized cost effectively.

The RUPI-44 family consists of three pin compatible parts:

- 8344—8051 Microcontroller with SIU
- 8044—An 8344 with 4K bytes of on-chip ROM program memory.
- 8744—An 8344 with 4K bytes of on-chip EPROM program memory.

17.1 ARCHITECTURE OVERVIEW

The 8044's dual controller architecture enables the RUPI to perform complex control tasks and high speed communication in a distributed network environment.

The 8044 microcontroller is the 8051-core, and maintains complete software compatibility with it. The microcontroller contains a powerful CPU with on-chip peripherals, making it capable of serving sophisticated real-time control applications such as instrumentation, industrial control, and intelligent computer peripherals. The microcontroller features on-chip peripherals such as two 16-bit timer/counters and 5 source interrupt capability with programmable priority levels. The microcontroller's high performance CPU executes most instructions in 1 microsecond, and can perform an 8 × 8 multiply in 4 microseconds. The CPU features a Boolean processor that can perform operations on 256 directly addressable bits. 192 bytes of on-chip data RAM can be extended to 64K bytes externally. 4K bytes of on-chip program ROM can be extended to 64K bytes externally. The CPU and SIU run concurrently. See Figure 17-2.

The SIU is designed to perform serial communications with little or no CPU involvement. The SIU supports data rates up to 2.4Mbps, externally clocked, and 375K bps self clocked (i.e., the data clock is recovered by an on-chip digital phase locked loop). SIU hardware supports the HDLC/SDLC protocol: zero bit insertion/deletion, address recognition, cyclic redundancy check, and frame number sequence check are automatically performed.

The SIU's Auto mode greatly reduces communication software overhead. The AUTO mode supports the SDLC Normal Response Mode, by performing secondary station responses in hardware without any CPU involvement. The Auto mode's interrupt control and frame sequence numbering capability eliminates software overhead normally required in conventional systems. By using the Auto mode, the CPU is free to concentrate on real time control of the application.

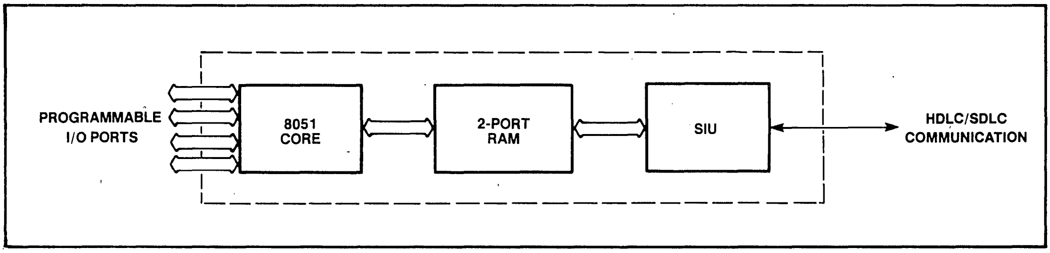


Figure 17-1. RUPI™-44 Dual Controller Architecture

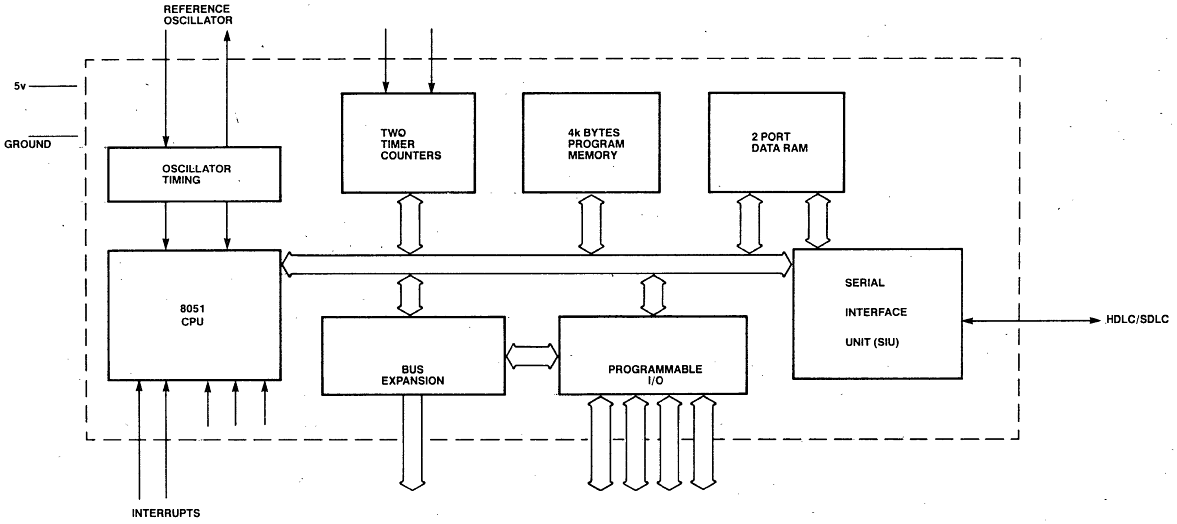


Figure 17-2. Simplified 8044 Block Diagram

17.2 THE HDLC/SDLC PROTOCOLS

17.2.1 HDLC/SDLC Advantages Over Async

The High Level Data Link Control, HDLC, is a standard communication link control established by the International Standards Organization (ISO). SDLC is a subset of HDLC.

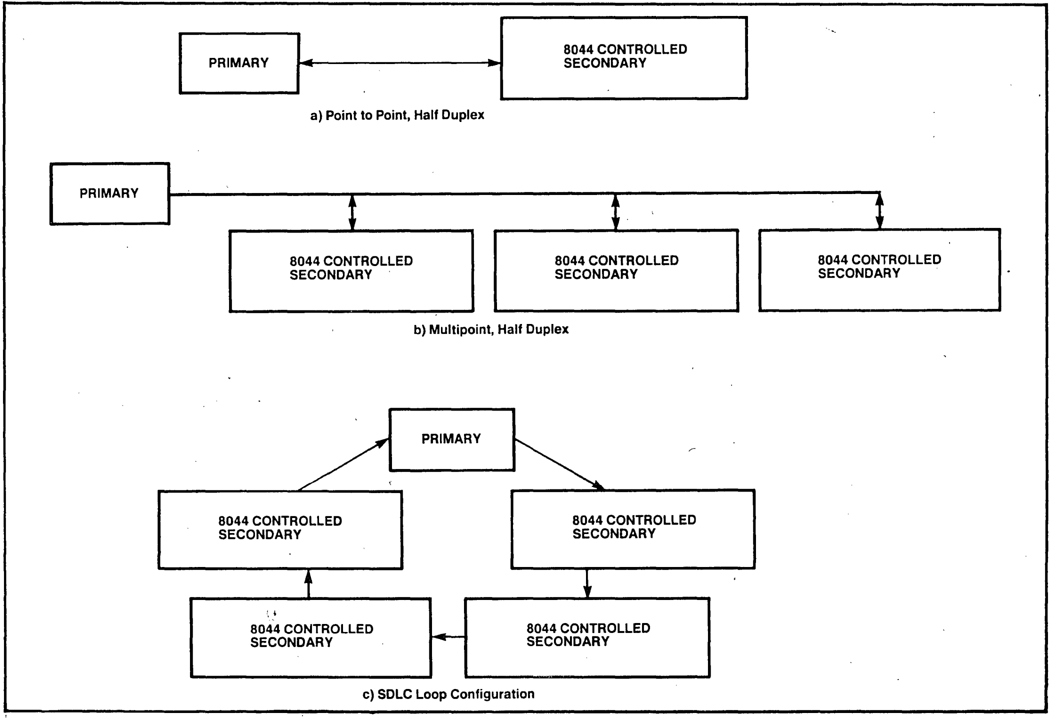
HDLC and SDLC are both well recognized standard serial protocols. The Synchronous Data Link Control, SDLC, is an IBM standard communication protocol. IBM originally developed SDLC to provide efficient, reliable and simple communication between terminals and computers.

The major advantages of SDLC/HDLC over Asynchronous communications protocol (Async):

• SIMPLE:	Data Transparency
• EFFICIENT:	Well Defined Message-Level Operation
• RELIABLE:	Frame Check Sequence and Frame Numbering

The SDLC reduces system complexity. HDLC/SDLC are "data transparent" protocols. Data transparency means that an arbitrary data stream can be sent without concern that some of data could be mistaken for a protocol controller. Data transparency relieves the communication controller having to detect special characters.

SDLC/HDLC provides more data throughout than Async. SDLC/HDLC runs at Message-level Operation which transmits multiple bytes within the frame. Whereas Async is based on character-level operation. Async transmits or receives a character at a time. Since Async requires start and stop bits in every transmission, there is a considerable waste of overhead compared to SDLC/HDLC.


PRIMARY
8044 CONTROLLED SECONDARY
PRIMARY
8044 CONTROLLED SECONDARY
8044 CONTROLLED SECONDARY
8044 CONTROLLED SECONDARY
PRIMARY
8044 CONTROLLED SECONDARY
8044 CONTROLLED SECONDARY
8044 CONTROLLED SECONDARY
8044 CONTROLLED SECONDARY

a) Point to Point, Half Duplex

b) Multipoint, Half Duplex

c) SDLC Loop Configuration

Figure 17-3. RUPI™-44 Supported Network Configurations

Due to SDLC/HDLC's well delineated field (see Figure 17-4) the CPU does not have to interpret character by character to determine control field and information field. In the case of Async, CPU must look at each character to interpret what it means. The practical advantage of such feature is straight forward use of DMA for information transfer.

In addition, SDLC/HDLC further improves Data throughput using implied Acknowledgement of transferred information. A station using SDLC/HDLC may acknowledge previously received information while transmitting different information in the same frame. In addition, up to 7 messages may be outstanding before an acknowledgement is required.

The HDLC/SDLC protocol can be used to realize reliable data links. Reliable Data transmission is ensured at the bit level by sending a frame check sequence, cyclic redundancy checking, within the frame. Reliable frame transmission is ensured by sending a frame number identification with each frame. This means that a receiver can sequentially count received frames and at any time infer what the number of the next frame to be received should be. More important, it provides a means for the receiver to identify to the sender some particular frame that it wishes to have resent because of errors.

17.2.2 HDLC/SDLC Networks

In both the HDLC and SDLC line protocols a (Master) primary station controls the overall network (data link) and issues commands to the secondary (Slave) stations. The latter complies with instructions and responds by sending appropriate responses. Whenever a transmitting station must end transmission prematurely, it sends an abort character. Upon detecting an abort character, a receiving station ignores the transmission block called a frame.

RUPI-44 supported HDLC/SDLC network configurations are point to point (half duplex) multipoint (half duplex), and loop. In the loop configuration the stations themselves act as repeaters, so that long links can be easily realized, see Figure 17-3.

17.2.3 Frames

An HDLC/SDLC frame consists of five basic fields: Flag, Address, Control, Data and Error Detection. A frame is bounded by flags—opening and closing flags. An address field is 8 bits wide in SDLC, extendable to 2 or more bytes in HDLC. The control field is also 8 bits wide, extendable to two bytes in HDLC. The SDLC data field or information field may be any number of bytes. The HDLC data field may or may not be on an 8 bit boundary. A powerful error detection code called Frame Check Sequence contains the calculated CRC (Cycle Redundancy Code) for all the bits between the flags. See Figure 17-4.

In HDLC and SDLC are three types of frames; an Information Frame is used to transfer data, a Supervisory Frame is used for control purposes, and a Nonsequenced Frame is used for initialization and control of the secondary stations.

For a more detailed discussion of higher level protocol functions interested readers may refer to the references listed in Section 17.2.6.

17.2.4 Zero Bit Insertion

In data communications, it is desirable to transmit data which can be of arbitrary content. Arbitrary data transmission requires that the data field cannot contain characters which are defined to assist the transmission protocol (like opening flag in HDLC/SDLC communications). This property is referred to as "data transparency". In HDLC/SDLC, this code transparency is made possible by Zero Bit Insertion (ZBI).

The flag has a unique binary bit pattern: 01111110 (7E HEX). To eliminate the possibility of the data field containing a 7E HEX pattern, a bit stuffing technique

OPENING FLAG	ADDRESS FIELD	CONTROL FIELD	INFORMATION FIELD	FRAME CHECK SEQUENCE (FCS)	CLOSING FLAG
01111110	8 BITS	8 BITS	VARIABLE LENGTH (ONLY IN I FRAMES)	16 BITS	01111110

Figure 17-4. Frame Format

called Zero Bit Insertion is used. This technique specifies that during transmission, a binary 0 be inserted by the transmitter after any succession of five contiguous binary 1's. This will ensure that no pattern of 0 1 1 1 1 1 1 0 is ever transmitted between flags. On the receiving side, after receiving the flag, the receiver hardware automatically deletes any 0 following five consecutive 1's. The 8044 performs zero bit insertion and deletion automatically.

17.2.5 Non-return to Zero Inverted (NRZI)

NRZI is a method of clock and data encoding that is well suited to the HDLC/SDLC protocol. It allows HDLC/SDLC protocols to be used with low cost asynchronous modems. NRZI coding is done at the transmitter to enable clock recovery from the data at the receiver terminal by using standard digital phase locked loop (DPLL) techniques. NRZI coding specifies that the signal condition does not change for transmitting a 1, while an 0 causes a change of state. NRZI coding ensures that an active data line will have a transition at least every 5-bit times (recall Zero Bit Insertion), while contiguous 0's will cause a change of state. Thus, ZBI and NRZI encoding makes it possible for the 8044's on-chip DPLL to recover a receive clock (from received data) synchronized to the received data and at the same time ensure data transparency.

17.2.6 References

1. *IBM Synchronous Data Link Control General Information GA27-3093-2, File No. GENL-09.*
2. *Microcontroller with Integrated High Performance Communications Interface, By Charles Yager Professional Program Session Record 8, WESCON/83.*
3. *Standard Network Access Protocol Specification, DATAPAC Trans-Canada Telephone System CCG111.*
4. *IBM 3650 Retail Store System Loop Interface OEM Information, IBM, GA 27-3098-0.*
5. *Guidebook to Data Communications, Training Manual, Hewlett-Packard 5955-1715.*
6. *"Using the 8273 SDLC/HDLC Protocol Controller", Application Note #36. March, 1978. Intel Corporation.*
7. *"LSI Devices Control Loop-Mode SDLC Data Links", Article Reprint #94. August, 1979. Intel Corporation.*

17.3 RUPI™-44 DESIGN SUPPORT

17.3.1 Design Tool Support

A critical design consideration is time to market. Intel provides a sophisticated set of design tools to speed hardware and software development time of 8044 based products. These include ICE-44, ASM-51, and PL/M-51.

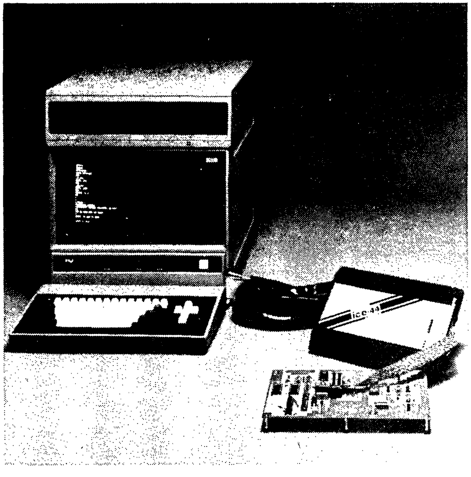


Figure 17-5. RUPI™-44 Development Support Configuration Intellec® System, ICE™-44 Buffer Box, and ICE-44 Module Plugged into a User Prototype Board.

A primary tool is the 8044 In Circuit Emulator, called ICE-44. See Figure 17-5. In conjunction with Intel's Intellec® Microprocessor Development System, the ICE-44 emulator allows hardware and software development to proceed interactively. This approach is more effective than the traditional method of independent hardware and software development followed by system integration. With the ICE-44 module, prototype hardware can be added to the system as it is designed. Software and hardware integration occurs while the product is being developed.

The ICE-44 emulator assists four stages of development:

1) Software Debugging.

It can be operated without being connected to the user's system before any of the user's hardware is available. In this stage ICE-44 debugging capabilities can be used in conjunction with the Intellec text editor and 8044 macroassembler to facilitate program development.

2) Hardware Development

The ICE-44 module's precise emulation characteristics and full-speed program RAM make it a valuable tool for debugging hardware, including the time-critical SDLC serial port, parallel port, and timer interfaces.

3) System Integration

Integration of software and hardware can begin when any functional element of the user system hardware is connected to the 8044 socket. As each section of the user's hardware is completed, it is added to the prototype. Thus, each section of the hardware and software is system tested in real-time operation as it becomes available.

4) System Test

When the user's prototype is complete, it is tested with the final version of the user system software. The ICE-44 module is then used for real-time emulation of the 8044 to debug the system as a completed unit.

The final product verification test may be performed using the 8744 EPROM version of the 8044 microcomputer. Thus, the ICE-44 module provides the user with the ability to debug a prototype or production system at any stage in its development.

A conversion kit, ICE-44 CON, is available to upgrade an ICE-51 module to ICE-44.

Intel's ASM-51 Assembler supports the 8044 special function registers and assembly program development. PL/M-51 provides designers with a high level language for the 8044. Programming in PL/M can greatly reduce development time, and ensure quick time to market.

17.3.2 8051 Workshop

Intel provides 8051 training to its customers through the 5-day 8051 workshop. Familiarity with the 8051 and 8044 is achieved through a combination of lecture and laboratory exercises.

For designers not familiar with the 8051, the workshop is an effective way to become proficient with the 8051 architecture and capabilities.

8044 Architecture

18

CHAPTER 18
8044 ARCHITECTURE

18.0 GENERAL

The 8044 is based on the 8051 core. The 8044 replaces the 8051's serial port with an intelligent HDLC/SDLC controller called the Serial Interface or SIU. Thus the differences between the two result from the 8044's increased on-chip RAM (192 bytes) and additional special function registers necessary to control the SIU. Aside from the increased memory, the SIU itself, and differences in 5 pins (for the serial port), the 8044 and 8051 are compatible.

This chapter describes the differences between the 8044 and 8051. Information pertaining to the 8051 core, eg. instruction set, port operation, EPROM programming, etc. is located in the 8051 sections of this manual.

A block diagram of the 8044 is shown in Figure 18-1. The pinpoint is shown on the inside front cover.

18.1 MEMORY ORGANIZATION OVERVIEW

The 8044 maintains separate address spaces for Program Memory and Data Memory. The Program Memory can be up to 64K bytes long, of which the lowest 4K bytes are in the on-chip ROM.

If the $\overline{\text{EA}}$ pin is held high, the 8044 executes out of internal ROM unless the Program Counter exceeds 0FFFH. Fetches from locations 1000H through FFFFH are directed to external Program Memory.

If the $\overline{\text{EA}}$ pin is held low, the 8044 fetches all instructions from external Program Memory.

The Data Memory consists of 192 bytes of on-chip RAM, plus 35 Special Function Registers, in addition to which the device is capable of accessing up to 64K bytes of external data memory.

The Program Memory uses 16-bit addresses. The external Data Memory can use either 8-bit or 16-bit addresses. The internal Data Memory uses 8-bit addresses, which provide a 256-location address space. The lower 192 addresses access the on-chip RAM. The Special Function Registers occupy various locations in the upper 128 bytes of the same address space.

The lowest 32 bytes in the internal RAM (locations 00 through 1FH) are divided into 4 banks of registers, each bank consisting of 8 bytes. Any one of these banks can be selected to be the "working registers" of the CPU, and can be accessed by a 3-bit address in the same byte as the opcode of an instruction. Thus, a large number of instructions are one-byte instructions.

The next higher 16 bytes of the internal RAM (locations 20H through 2FH) have individually addressable bits. These are provided for use as software flags or for one-bit (Boolean) processing. This bit-addressing capability is an important feature of the 8044. In addition to the 128 individually addressable bits in RAM, twelve of the Special Function Registers also have individually addressable bits.

A memory map is shown in Figure 18-2.

18.1.1 Special Function Registers

The Special Function Registers are as follows:

*	ACC	Accumulator (A Register)
*	B	B Register
*	PSW	Program Status Word
	SP	Stack Pointer
	DPTR	Data Pointer (consisting of DPH AND DPL)
*	P0	Port 0
*	P1	Port 1
*	P2	Port 2
*	P3	Port 3
*	IP	Interrupt Priority
*	IE	Interrupt Enable
	TMOD	Timer/Counter Mode
*	TCON	Timer/Counter Control
	TH0	Timer/Counter 0 (high byte)
	TL0	Timer/Counter 0 (low byte)
	THI	Timer/Counter 1 (high byte)
	TL1	Timer/Counter 1 (low byte)
	SMD	Serial Mode
*	STS	Status/Command
*	NSNR	Send/Receive Count
	STAD	Station Address
	TBS	Transmit Buffer Start Address
	TBL	Transmit Buffer Length
	TCB	Transmit Control Byte
	RBS	Receive Buffer Start Address
	RBL	Receive Buffer Length
	RFL	Received Field Length
	RCB	Received Control Byte
	DMA CNT	DMA Count
	FIFO	FIFO (three bytes)
	SIUST	SIU State Counter
	PCON	Power Control

The registers marked with * are both byte- and bit-addressable.

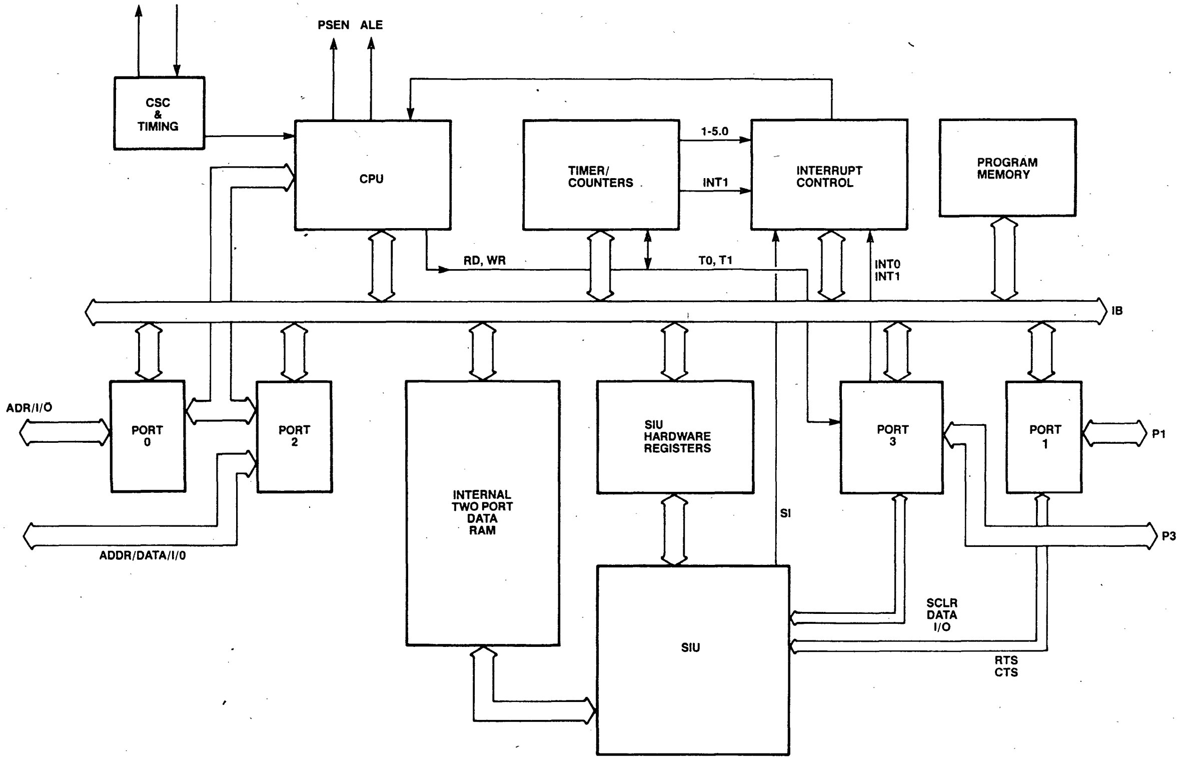


Figure 18-1 RUPI™ Block Diagram

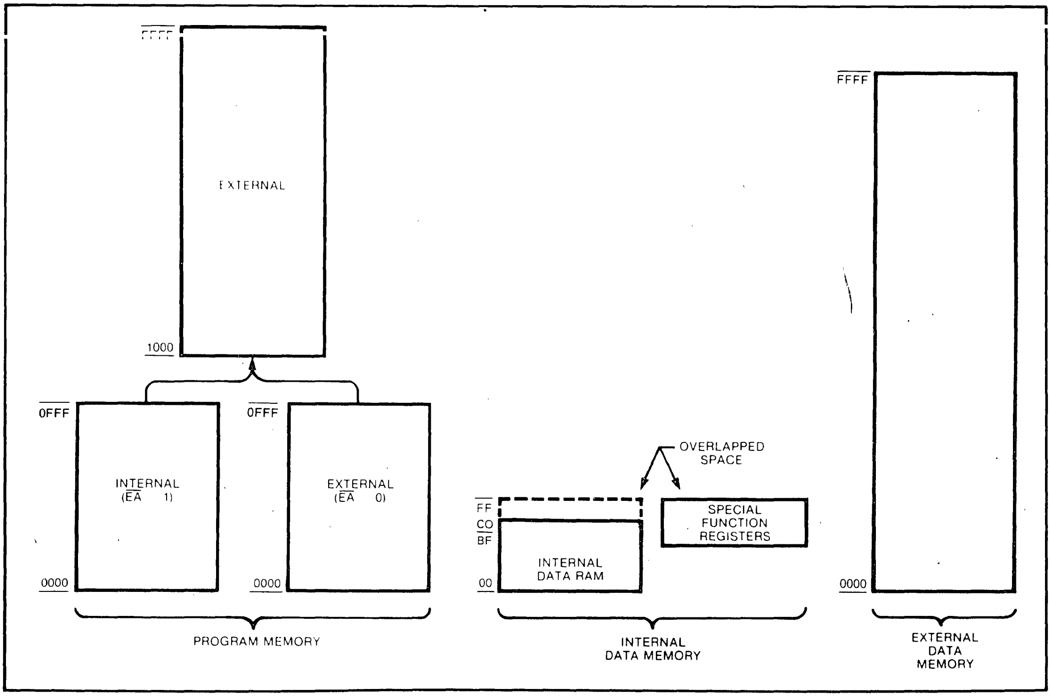


Figure 18-2. RUPI™-44 Memory Map

Stack Pointer

The Stack Pointer is 8 bits wide. The stack can reside anywhere in the 192 bytes of on-chip RAM. When the 8044 is reset, the stack pointer is initialized to 07H. When executing a PUSH or a CALL, the stack pointer is incremented before data is stored, so the stack would begin at location 08H.

18.1.2 Interrupt Control Registers

The Interrupt Request Flags are as listed below:

Source	*Request Flag*	*Location*
External Interrupt 0	$\overline{\text{INT0}}$, if IT0 = 0	P3.2
	IE0, if IT0 = 1	TCON.1
Timer 0 Overflow	TF0	TCON.5
External Interrupt 1	$\overline{\text{INT1}}$, if IT1 = 0	P3.3
	IE1, if IT1 = 1	TCON.3
Timer 1 Overflow	TF1	TCON.7
Serial Interface Unit	SI	STS.4

External Interrupt control bits IT0 and IT1 are in TCON.0 and TCON.2, respectively. Reset leaves all flags inactive, with IT0 and IT1 cleared.

All the interrupt flags can be set or cleared by software, with the same effect as by hardware.

The Enable and Priority Control Registers are shown below. All of these control bits are set or cleared by software. All are cleared by reset.

IE: Interrupt Enable Register (bit-addressable)

Bit:	7	6	5	4	3	2	1	0
	EA	X	X	ES	ET1	EX1	ET0	EX0

where:

- EA disables all interrupts. If EA = 0, no interrupt will be acknowledged. If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.

- ES — enables or disables the Serial Interface Unit interrupt. If ES = 0, the Serial Interface Unit interrupt is disabled.
- ET1 — enables or disables the Timer 1 Overflow interrupt. If ET1 = 0, the Timer 1 interrupt is disabled.
- EX1 — enables or disables External Interrupt 1. If EX1 = 0, External Interrupt 1 is disabled.
- ET0 — enables or disables the Timer 0 Overflow interrupt. If ET0 = 0, the Timer 0 interrupt is disabled.

IP: Interrupt Priority Register (bit-addressable)

Bit:	7	6	5	4	3	2	1	0
	X	X	X	PS	PT1	PX1	PT0	PX0

where:

- PS — defines the Serial Interface Unit interrupt priority level. PS = 1 programs it to the higher priority level.
- PT1 — defines the Timer 1 interrupt priority level. PT1 = 1 programs it to the higher priority level.
- PX1 — defines the External Interrupt 1 priority level. PX1 = 1 programs it to the higher priority level.
- PT0 — defines the Timer 0 interrupt priority level. PT0 = 1 programs it to the higher priority level.
- PX0 — defines the External Interrupt 0 priority level. PX0 = 1 programs it to the higher priority level.

18.2 Memory Organization Details

In the 8044 family the memory is organized over three address spaces and the program counter. The memory spaces shown in Figure 18-2 are the:

- 64K-byte Program Memory address space
- 64K-byte External Data Memory address space
- 320-byte Internal Data Memory address space

The 16-bit Program Counter register provides the 8044 with its 64K addressing capabilities. The Program Counter allows the user to execute calls and branches to any location within the Program Memory space. There are no instructions that permit program execution to move from the Program Memory space to any of the data memory spaces.

In the 8044 and 8744 the lower 4K of the 64K Program Memory address space is filled by internal ROM and EPROM, respectively. By tying the $\overline{EA}$ pin high, the processor can be forced to fetch from the internal ROM/EPROM for Program Memory addresses 0 through 4K. Bus expansion for accessing Program Memory beyond 4K is automatic since external instruction fetches occur automatically when the Program Counter increases above 4095. If the $\overline{EA}$ pin is tied low all Program Memory fetches are from external memory. The execution speed of the 8044 is the same regardless of whether fetches are from internal or external Program Memory. If all program storage is on-chip, byte location 4095 should be left vacant to prevent an undesired prefetch from external Program Memory address 4096.

Certain locations in Program Memory are reserved for specific programs. Locations 0000 through 0002 are reserved for the initialization program. Following reset, the CPU always begins execution at location 0000. Locations 0003 through 0042 are reserved for the five interrupt-request service programs. Each resource that can request an interrupt requires that its service program be stored at its reserved location.

The 64K-byte External Data Memory address space is automatically accessed when the MOVX instruction is executed.

Functionally the Internal Data Memory is the most flexible of the address spaces. The Internal Data Memory space is subdivided into a 256-byte Internal Data RAM address space and a 128-byte Special Function Register address space as shown in Figure 18-3.

The Internal Data RAM address space is 0 to 255. Four 8-Register Banks occupy locations 0 through 31. The stack can be located anywhere in the Internal Data RAM address space. In addition, 128 bit locations of the on-chip RAM are accessible through Direct Addressing. These bits reside in Internal Data RAM at byte locations 32 through 47. Currently locations 0 through 191 of the Internal Data RAM address space are filled with on-chip RAM.

The stack depth is limited only by the available Internal Data RAM, thanks to an 8-bit reloadable Stack Pointer. The stack is used for storing the Program Counter during subroutine calls and may be used for passing parameters. Any byte of Internal Data RAM or Special Function Register accessible through Direct Addressing can be pushed/popped.

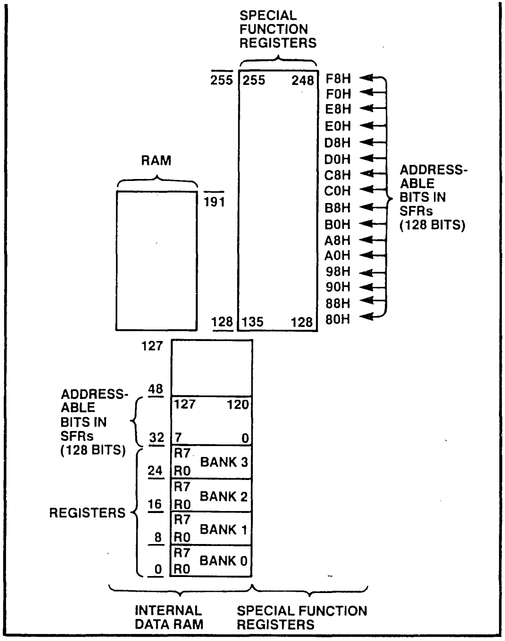


Figure 18-3. Internal Data Memory Address Space

ARITHMETIC REGISTERS:
Accumulator*, B register*,
Progam Status Word*
POINTERS:
Stack Pointer, Data Pointer (high & low)
PARALLEL I/O PORTS:
Port 3*, Port 2*, Port 1*, Port 0*
INTERRUPT SYSTEM:
Interrupt Priority Control*,
Interrupt Enable Control*
TIMERS:
Timer MODe, Timer CONtrol*, Timer 1 (high & low), Timer 0 (high & low)
SERIAL INTERFACE UNIT:
Transmit Buffer Start,
Transmit Buffer Length,
Transmit Control Byte,
Send Count Receive Count*,
DMA Count,
Station Address
Receive Field Length
Receive Buffer Start
Receive Buffer Length
Receive Control Byte,
Serial Mode,
Status Register.*

* Bits in these registers are bit addressable.

Figure 18-4. Special Function Registers

The Special Function Register address space is 128 to 255. All registers except the Program Counter and the four 8-Register Banks reside here. Memory mapping the Special Function Registers allows them to be accessed as easily as internal RAM. As such, they can be operated on by most instructions. In the overlapping memory space (address 128–191), indirect addressing is used to access RAM, and direct addressing is used to access the SFR's. The SFR's at addresses 192–255 are also accessed using direct addressing. The Special Function Registers are listed in Figure 18-4. Their mapping in the Special Function Register address space is shown in Figures 18-5 and 18-6.

Performing a read from a location of the Internal Data memory where neither a byte of Internal Data RAM (i.e., RAM addresses 192–255) nor a Special Function Register exists will access data of indeterminable value.

Architecturally, each memory space is a linear sequence of 8-bit wide bytes. By Intel convention the storage of multi-byte address and data operands in program and data memories is the least significant byte at the low-order address and the most significant byte at the high-order address. Within byte X, the most significant bit is represented by X.7 while the least significant bit is X.0. Any deviation from these conventions will be explicitly stated in the text.

18.2.1 Operand Addressing

There are five methods of addressing source operands. They are Register Addressing, Direct Addressing, Register-Indirect Addressing, Immediate Addressing and Base-Register-plus Index-Register-Indirect Addressing. The first three of these methods can also be used to address a destination operand. Since operations in the 8044 require 0 (NOP only), 1, 2, 3 or 4 operands, these five addressing methods are used in combinations to provide the 8044 with its 21 addressing modes.

Most instructions have a "destination, source" field that specifies the data type, addressing methods and operands involved. For operations other than moves, the destination operand is also a source operand. For example, in "subtract-with-borrow A, #5" the A register receives the result of the value in register A minus 5, minus C.

Most operations involve operands that are located in Internal Data Memory. The selection of the Program Memory space or External Data Memory space for a

ARITHMETIC REGISTERS:
  Accumulator*, B register*,
  Progam Status Word*
POINTERS:
  Stack Pointer, Data Pointer (high & low)
PARALLEL I/O PORTS:
  Port 3*, Port 2*, Port 1*, Port 0*
INTERRUPT SYSTEM:
  Interrupt Priority Control*,
  Interrupt Enable Control*
TIMERS:
  Timer Mode, Timer Control*, Timer 1 (high & low), Timer 0 (high & low)
SERIAL INTERFACE UNIT:
  Serial Mode, Status/Command*,
  Send/Receive Count*, Station Address,
  Transmit Buffer Start Address,
  Transmit Buffer Length,
  Transmit Control Byte,
  Receive Buffer Start Address,
  Receive Buffer Length,
  Receive Field Length,
  Receive Control Byte,
  DMA Count,
  FIFO (three bytes),
  SIU Controller State Counter

* Bits in these registers are bit-addressable

Figure 18-5. Mapping of Special Function Registers

second operand is determined by the operation mnemonic unless it is an immediate operand. The subset of the Internal Data Memory being addressed is determined by the addressing method and address value. For example, the Special Function Registers can be accessed only through Direct Addressing with an address of 128–255. A summary of the operand addressing methods is shown in Figure 18-6. The following paragraphs describe the five addressing methods.

18.2.2 Register Addressing

Register Addressing permits access to the eight registers (R7-R0) of the selected Register Bank (RB). One of the four 8-Register Banks is selected by a two-bit field in the PSW. The registers may also be accessed through Direct Addressing and Register-Indirect Addressing, since the four Register Banks are mapped into the lowest 32 bytes of Internal Data RAM as shown in Figures 18-9 and 18-10. Other Internal Data Memory locations that are addressed as registers are A, B, C, AB and DPTR.

REGISTER NAMES	SYMBOLIC ADDRESS	BIT ADDRESS	BYTE ADDRESS	
B REGISTER	B	247 through 240	240	(F0H)
ACCUMULATOR	ACC	231 through 224	224	(E0H)
*THREE BYTE FIFO	FIFO		223	(DFH)
	FIFO		222	(DEH)
	FIFO		221	(DDH)
TRANSMIT BUFFER START	TBS		220	(DCH)
TRANSMIT BUFFER LENGTH	TBL		219	(DBH)
TRANSMIT CONTROL BYTE	TCB		218	(DAH)
*SIU STATE COUNTER	SIUST		217	(D9H)
SEND COUNT RECEIVE COUNT	NSNR	223 through 216	216	(D8H)
PROGRAM STATUS WORD	PSW	215 through 208	208	(D0H)
*DMA COUNT	DMA CNT		207	(CFH)
STATION ADDRESS	STAD		206	(CEH)
RECEIVE FIELD LENGTH	RFL		205	(CDH)
RECEIVE BUFFER START	RBS		204	(CCH)
RECEIVE BUFFER LENGTH	RBL		203	(CBH)
RECEIVE CONTROL BYTE	RCB		202	(CAH)
SERIAL MODE	SMD		201	(C9H)
STATUS REGISTER	STS	207 through 200	200	(C8H)
INTERRUPT PRIORITY CONTROL	IP	191 through 184	184	(B8H)
PORT 3	P3	183 through 176	176	(B0H)
INTERRUPT ENABLE CONTROL	IE	175 through 168	168	(A8H)
PORT 2	P2	167 through 160	160	(A0H)
PORT 1	P1	151 through 144	144	(90H)
TIMER HIGH 1	TH1		141	(8DH)
TIMER HIGH 0	TH0		140	(8CH)
TIMER LOW 1	TL1		139	(8BH)
TIMER LOW 0	TL0		138	(8AH)
TIMER MODE	TMOD		137	(89H)
TIMER CONTROL	TCON	143 through 136	136	(88H)
DATA POINTER HIGH	DPH		131	(83H)
DATA POINTER LOW	DPL		130	(82H)
STACK POINTER	SP		129	(81H)
PORT 0	P0	135 through 128	128	(80H)

SFR's CONTAINING DIRECT ADDRESSABLE BITS

*ICE Support Hardware registers. Under normal operating conditions there is no need for the CPU to access these registers.

18-6. Mapping of Special Function Registers

Direct Byte Address	Bit Address (MSB)							(LSB)	Hardware Register Symbol
240	F7	F6	F5	F4	F3	F2	F1	F0	8
224	E7	E6	E5	E4	E3	E2	E1	E0	ACC
	NS2	NS1	NS0	SES	NR2	NR1	NR0	SER	
216	DF	DE	DD	DC	DB	DA	D9	D8	NSNR
	CY	AC	F0	RS1	RS0	OV		P	
204	D7	D6	D5	D4	D3	D2	D1	D0	PSW
	TBF	RE	RTS	SI	BV	CPB	AM	RBP	
200	CF	CE	CD	CC	CB	CA	C9	C8	STS
				PS	PT1	PX1	PT0	PX0	
184	—	—	—	BC	BB	BA	B9	B8	1P
176	B7	B6	B5	B4	B3	B2	B1	B0	P3
	EA			E5	ET1	EX1	ET0	EX0	
168	AF	—	—	AC	AB	AA	A9	A8	1E
160	A7	A6	A5	A4	A3	A2	A1	A0	P2
144	97	96	95	94	93	92	91	90	P1
	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	
136	8F	8E	8D	8C	8B	8A	89	88	TCON
128	87	86	85	84	83	82	81	80	P0

Figure 18-7. Special Function Register Bit Address

- Register Addressing
  - R7-R0
  - A,B,C (bit), AB (two bytes), DPTR (double byte)
- Direct Addressing
  - Lower 128 bytes of Internal Data RAM
  - Special Function Registers
  - 128 bits in subset of Special Function Register address space
- Register-Indirect Addressing
  - Internal Data RAM [@R1, @R0, @SP (PUSH and POP only)]
  - Least Significant Nibbles in Internal Data RAM (@R1, @R0)
  - External Data Memory (@R1, @R0, @DPTR)
- Immediate Addressing
  - Program Memory (in-code constant)
- Base-Register-plus Index-Register-Indirect Addressing
  - Program Memory (@ DPTR + A, @ PC+A)

Figure 18-8. Operand Addressing Methods

RAM BYTE	(MSB)							(LSB)	
BFH									191
2FH	7F	7E	7D	7C	7B	7A	79	78	47
2EH	77	76	75	74	73	72	71	70	46
2DH	6F	6E	6D	6C	6B	6A	69	68	45
2CH	67	66	65	64	63	62	61	60	44
2BH	5F	5E	5D	5C	5B	5A	59	58	43
2AH	57	56	55	54	53	52	51	50	42
29H	4F	4E	4D	4C	4B	4A	49	48	41
28H	47	46	45	44	43	42	41	40	40
27H	3F	3E	3D	3C	3B	3A	39	38	39
26H	37	36	35	34	33	32	31	30	38
25H	2F	2E	2D	2C	2B	2A	29	28	37
24H	27	26	25	24	23	22	21	20	36
23H	1F	1E	1D	1C	1B	1A	19	18	35
22H	17	16	15	14	13	12	11	10	34
21H	0F	0E	0D	0C	0B	0A	09	08	33
20H	07	06	05	04	03	02	01	00	32
1FH	Bank 3								31
18H									24
17H	Bank 2								23
10H									16
0FH	Bank 1								15
08H									8
07H	Bank 0								7
00H									0

Figure 18-9. RAM Bit Addresses

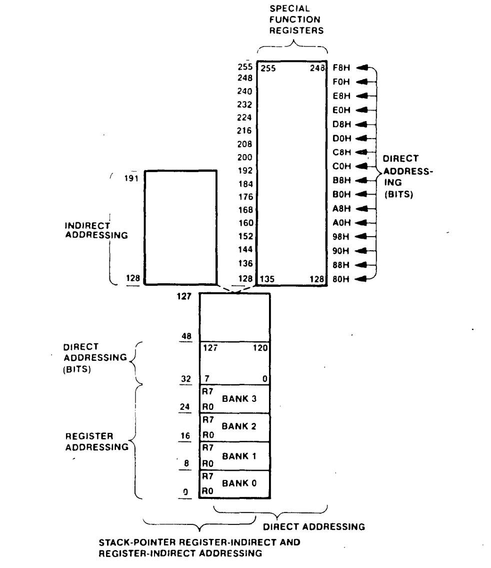


Figure 18-10. Addressing Operands in Internal Data Memory

18.2.3 Direct Addressing

Direct Addressing provides the only means of accessing the memory-mapped byte-wide Special Function Registers and memory mapped bits within the Special Function Registers and Internal Data RAM. Direct Addressing of bytes may also be used to access the lower 128 bytes of Internal Data RAM. Direct Addressing of bits gains access to a 128 bit subset of the Internal Data RAM and 128 bit subset of the Special Function Registers as shown in Figures 18-5, 18-6, 18-9, and 18-10.

Register-Indirect Addressing using the content of R1 or R0 in the selected Register Bank, or using the content of the Stack Pointer (PUSH and POP only), addresses the Internal Data RAM. Register-Indirect Addressing is also used for accessing the External Data Memory. In this case, either R1 or R0 in the selected Register Bank may be used for accessing locations within a 256-byte block. The block number can be preselected by the contents of a port. The 16-bit Data Pointer may be used for accessing any location within the full 64K external address space.

18.3 RESET

Reset is accomplished by holding the RST pin high for at least two machine cycles (24 oscillator periods) *while the oscillator is running*. The CPU responds by executing an internal reset. It also configures the ALE and PSEN pins as inputs. (They are quasi-bidirectional.) The internal reset is executed during the second cycle in which RST is high and is repeated every cycle until RST goes low. It leaves the internal registers as follows:

Register	*Content*
PC	0000H
A	00H
B	00H
PSW	00H
SP	07H
DPTR	0000H
P0 - P3	0FFH
IP	(XXX00000)
IE	(0XX00000)
TMOD	00H
TCON	00H
TH0	00H
TL0	00H
TH1	00H
TL1	00H
SMD	00H
STS	00H
NSNR	00H
STAD	00H
TBS	00H
TBL	00H
TCB	00H
RBS	00H
RBL	00H
RFL	00H
RCB	00H
DMA CNT	00H
FIFO1	00H
FIFO2	00H
FIFO3	00H
SIUST	01H
PCON	(0XXXXXXX)

The internal RAM is not affected by reset. When VCC is turned on, the RAM content is indeterminate unless VPD was applied prior to VCC being turned off (see Power Down Operation.)

18.4 RUPI™-44 FAMILY PIN DESCRIPTION

VSS: Circuit ground potential.

VCC: Supply voltage during programming (of the 8744), verification (of the 8044 or 8744), and normal operation.

Port 0: Port 0 is an 8-bit open drain bidirectional I/O port. It is also the multiplexed low-order address and data bus during accesses to external memory (during which accesses it activates internal pullups). It also outputs instruction bytes during program verification. (External pullups are required during program verification.) Port 0 can sink eight LS TTL inputs.

Port 1: Port 1 is an 8-bit bidirectional I/O port with internal pullups. It receives the low-order address byte during program verification in the 8044 or 8744. Port 1 can sink/source four LS TTL inputs. It can drive MOS inputs without external pullups.

Two of the Port 1 pins serve alternate functions, as listed below:

Port Pin	*Alternate Function*
P1.6	$\overline{\text{RTS}}$ (Request to Send). In a non-loop configuration, $\overline{\text{RTS}}$ signals that the 8044 is ready to transmit data.

P1.7 CTS (Clear to Send). In a non-loop configuration, CTS signals to the 8044 that the receiving station is ready to accept data.

Port 2: Port 2 is an 8-bit bidirectional I/O port with internal pullups. It emits the high-order address byte during accesses to external memory. It also receives the high-order address bits and control signals during program verification in the 8044 or 8744. Port 2 can sink/source four LS TTL inputs. It can drive MOS inputs without external pullups.

Port 3: Port 3 is an 8-bit bidirectional I/O port with internal pullups. Port 3 can sink/source four LS TTL inputs. It can drive MOS inputs without external pullups.

Port 3 pins also serve alternate functions, as listed below:

Port Pin	*Alternate Function*
P3.0	RXD (serial input port in loop configuration). I/O (data direction control in non-loop configuration).
P3.1	TXD (serial output port in loop configuration). DATA input/output pin in non-loop configuration.
P3.2	INT0 (external interrupt)
P3.3	INT1 (external interrupt)
P3.4	T0 (Timer 0 external input)
P3.5	T1 (Timer 1 external input) SCLK (Serial Data Clock Input)
P3.6	WR (external Data Memory write strobe)
P3.7	RD (external Data Memory read strobe)

RST/VPD: A high level on this pin for two machine cycles while the oscillator is running resets the device. An internal pulldown permits Power-On reset using only a capacitor connected to VCC.

ALE/PROG: Address Latch Enable output for latching the low byte of the address during accesses to external memory. ALE is activated though for this purpose at a constant rate of 1/6 the oscillator frequency even when external memory is not being accessed. Consequently it can be used for external clocking or timing purposes. (However, one ALE pulse is skipped during each access to external *Data* Memory.) This pin is also the program pulse input (PROG) during EPROM programming.

PSEN: Program Store Enable output is the read strobe to external Program Memory. PSEN is activated twice each machine cycle during fetches from external Program Memory. (However, when executing out of external Program Memory two activations of PSEN are skipped during each access to external *Data* Memory.) PSEN is not activated during fetches from internal Program Memory.

EA/VPP: When EA is held high the CPU executes out of internal Program Memory (unless the Program Counter exceeds (0FFFH). When EA is held low the CPU executes only out of external Program Memory. In the 8344, EA must be externally wired low. In the 8744, this pin also receives the 21V programming supply voltage (VPP) during EPROM programming.

XTAL1: Input to the inverting amplifier that forms the oscillator. Should be grounded when an external oscillator is used.

XTAL2: Output of the inverting amplifier that forms the oscillator, and input to the internal clock generator. Receives the external oscillator signal when an external oscillator is used.

8044 Serial Interface 19

CHAPTER 19
THE 8044 SERIAL INTERFACE UNIT

19.0 SERIAL INTERFACE

The serial interface provides a high-performance communication link. The protocol used for this communication is based on the IBM Synchronous Data Link Control (SDLC). The serial interface also supports a subset of the ISO HDLC (International Standards Organization High-Level Data Link Control) protocol.

The SDLC/HDLC protocols have been accepted as standard protocols for many high-level teleprocessing systems. The serial interface performs many of the functions required to service the data link without intervention from the 8044's own CPU. The programmer is free to concentrate on the 8044's function as a peripheral controller, rather than having to deal with the details of the communication process.

Five pins on the 8044 are involved with the serial interface (refer to Section 12.4, Family Pin Description, for details):

Pin 7	$\overline{\text{RTS}}$/P16
Pin 8	$\overline{\text{CTS}}$/P17
Pin 10	I/$\overline{\text{O}}$/RXD/P30
Pin 11	DATA/TXD/P31
Pin 15	SCLK/T1/P35

Figure 19-1 is a functional block diagram of the serial interface unit (SIU). More details on the SIU hardware are given in Section 19.9.

19.1 DATA LINK CONFIGURATIONS

The serial interface is capable of operating in three serial data link configurations:

1) Half-Duplex, point-to-point

2) Half-Duplex, multipoint (with a half-duplex or full-duplex primary)

3) Loop

Figure 19-2 shows these three configurations. The RTS (Request to Send) and CTS (Clear to Send) handshaking signals are available in the point-to-point and multipoint configurations.

19.2 DATA CLOCKING OPTIONS

The serial interface can operate in an externally clocked mode or in a self clocked mode.

Externally Clocked Mode

In the externally clocked mode, a common Serial Data Clock (SCLK on pin 15) synchronizes the serial bit stream. This clock signal may come from the master CPU or primary station, or from an external phase-locked loop local to the 8044. Figure 19-3 illustrates the timing relationships for the serial interface signals when the externally clocked mode is used in point-to-point and multipoint data link configurations.

Incoming data is sampled at the rising edge of SCLK, and outgoing data is shifted out at the falling edge of SCLK. More detailed timing information is given in the 8044 data sheet.

Self Clocked (Asynchronous) Mode

The self clocked mode allows data transfer without a common system data clock. Using an on-chip DPLL (digital phase locked loop) the serial interface recovers the data clock from the data stream itself. The DPLL requires a reference clock equal to either 16 times or 32 times the data rate. This reference clock may be externally supplied or internally generated. When the serial interface generates this clock internally, it uses either the 8044's internal logic clock (half the crystal frequency's PH2) or the "timer 1" overflow. Figure 19-4 shows the serial interface signal timing relationships for the loop configuration, when the unclocked mode is used.

The DPLL monitors the received data in order to derive a data clock that is centered on the received bits. Centering is achieved by detecting all transitions of the received data, and then adjusting the clock transition (in increments of 1/16 bit period) toward the center of the received bit. The DPLL converges to the nominal bit center within eight bit transitions, worst case.

To aid in the phase locked loop capture process, the 8044 has a NRZI (non-return-to-zero inverted) data encoding and decoding option. NRZI coding specifies that a signal does not change state for a transmitted binary 1, but does change state for a binary 0. Using the NRZI coding with zero-bit insertion, it can be guaranteed that an active signal line undergoes a transition at least every six bit times.

19.3 DATA RATES

The maximum data rate in the externally clocked mode is 2.4M bits per second (bps) a half-duplex configuration, and 1.0M in a loop configuration.

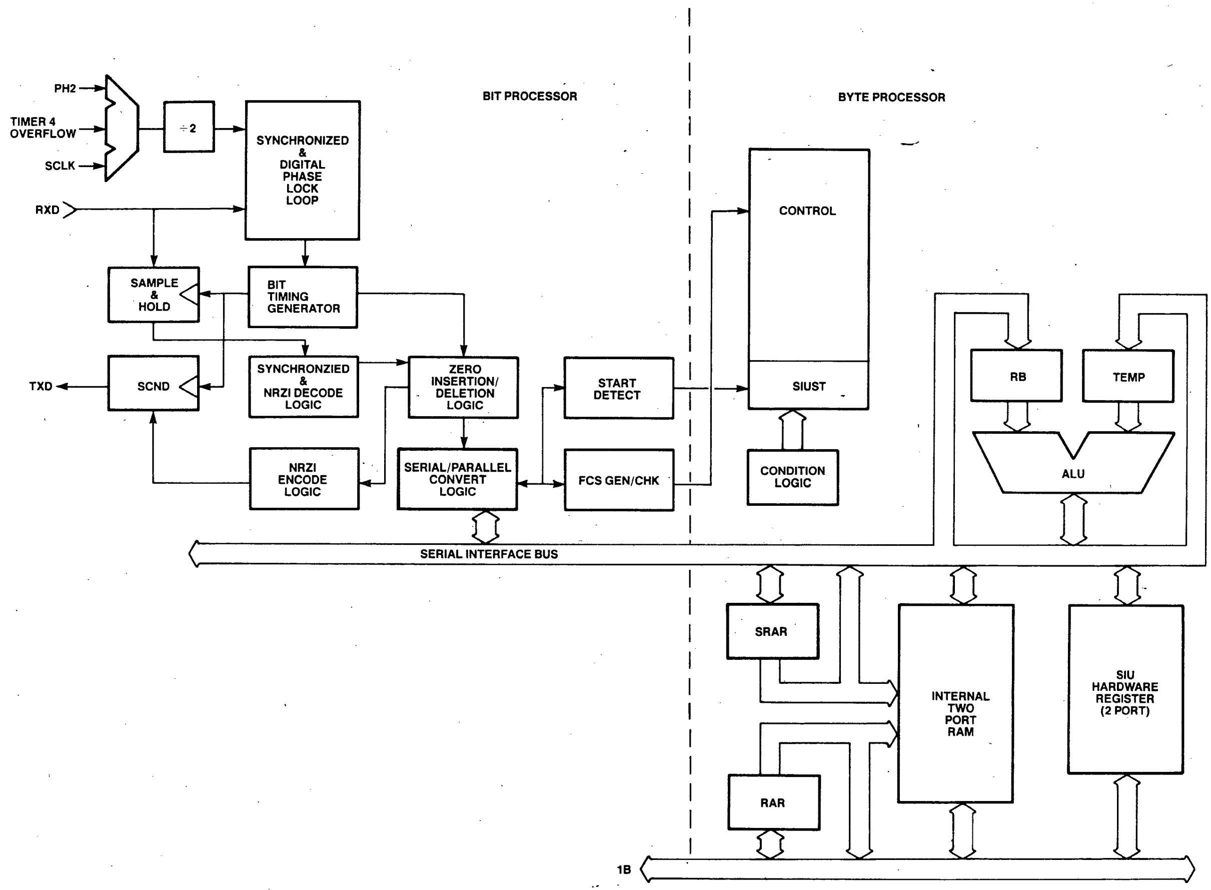


Figure 19-1. SIU Block Diagram

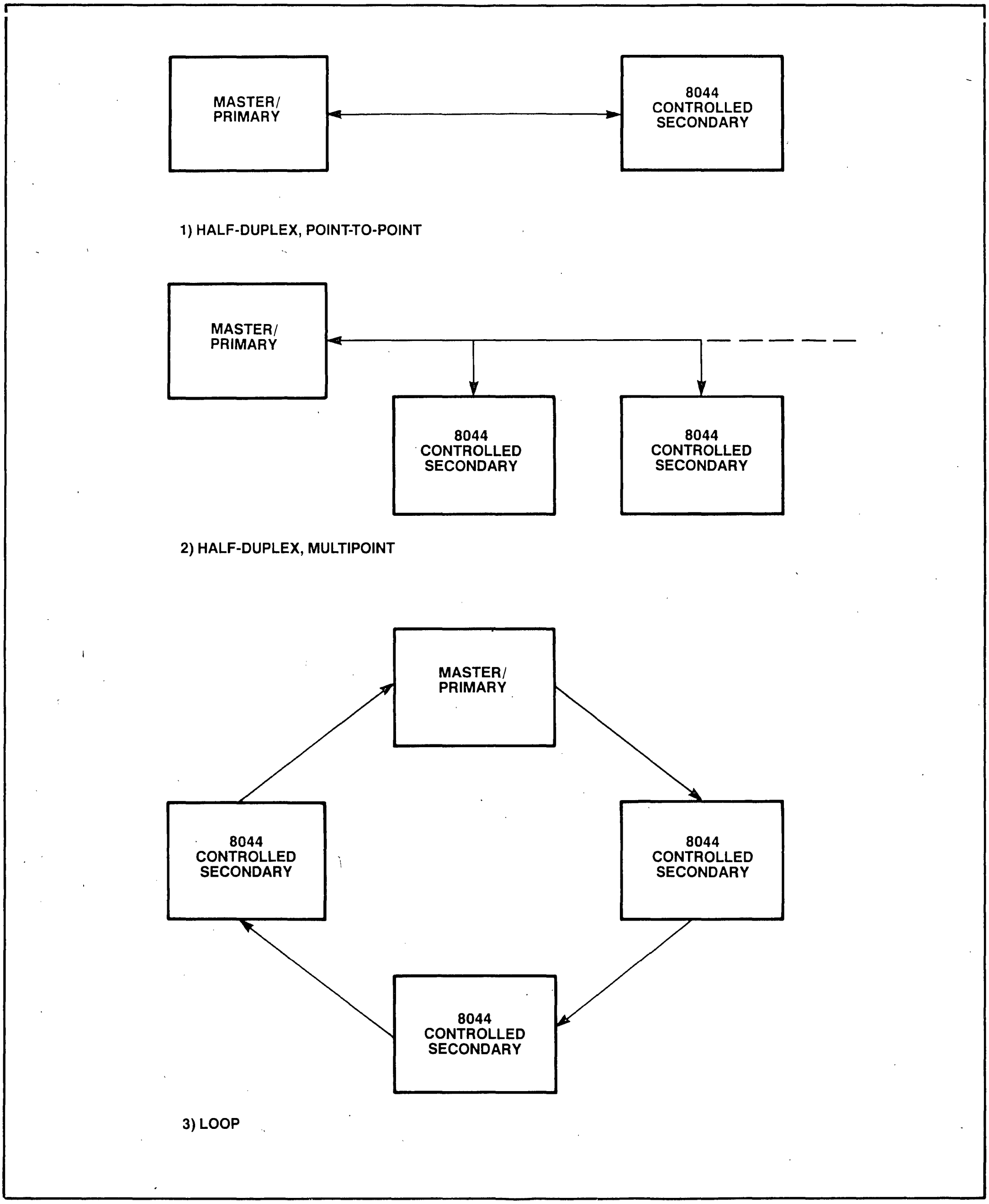


Figure 19-2. RUPI-44 Data Link Configurations

In the self clocked mode with an external reference clock, the maximum data rate is 375K bps.

In the self clocked mode with an internally generated reference clock, and the 8044 operating with a 12 MHz crystal, the available data rates are 244 bps to 62.5k bps, 187.5K bps and 375K bps.

For more details see the table in the SMD register description, below.

19.4 OPERATIONAL MODES

The Serial Interface Unit (SIU) can operate in either of two response modes:

1) AUTO mode

2) FLEXIBLE (NON-AUTO) mode

In the AUTO mode, the SIU performs in hardware a subset of the SDLC protocol called the normal response mode. The AUTO mode enables the SIU to recognize and respond to certain kinds of SDLC frames without intervention from the 8044's CPU. AUTO mode provides a faster turnaround time and a simplified software interface, whereas NON-AUTO mode provides a greater flexibility with regard to the kinds of operation permitted.

In AUTO mode, the 8044 can act only as a normal response mode secondary station—that is, it can transmit only when instructed to do so by the primary station. All such AUTO mode responses adhere strictly to IBM's SDLC definitions.

In the FLEXIBLE mode, reception or transmission of each frame by the SIU is performed under the control of the CPU. In this mode the 8044 can be either a primary station or a secondary station.

If both AUTO and FLEXIBLE modes, short frames, aborted frames, or frames which have had CRC's are ignored by the SIU.

The basic format of an SDLC frame is as follows:

Flag	Address	Control	Information	FCS	Flag

Format variations consist of omitting one or more of the fields in the SDLC frame. For example, a supervisory frame is formed by omitting the information field. Supervisory frames are used to confirm received frames, indicate ready or busy conditions, and to report errors. More details on frame formats are given in the SDLC Frame Format Options section, below.

19.4.1 AUTO Mode

To enable the SIU to receive a frame in AUTO mode, the 8044 CPU sets up a receive buffer. This is done by writing two registers—Receive Buffer Start (RBS) Address and Receive Buffer Length (RBL).

The SIU receives the frame, examines the control byte, and takes the appropriate action. If the frame is an information frame, the SIU will load the receive buffer, interrupt the CPU (to have the receive buffer read), and make the required acknowledgement to the primary station. Details on these processes are given in the Operation section, below.

In addition to receiving the information frames, the SIU in AUTO mode is capable of responding to the following commands (found in the control field of supervisory frames) from the primary station:

RR (Receive Ready): Acknowledges that the Primary station has correctly received numbered frames up through $N_R - 1$, and that it is ready to receive frame N_R.

RNR (Receive Not Ready): Indicates a temporary busy condition (at the primary station) due to buffering or other internal constraints. The quantity N_R in the control field indicates the number of the frame expected after the busy condition ends, and may be used to acknowledge the correct reception of the frames up through $N_R - 1$.

REJ (Reject): Acknowledges the correct reception of frames up through $N_R - 1$, and requests transmission or retransmission starting at frame N_R. The 8044 is capable of retransmitting at most the previous frame, and then only if it is still available in the transmit buffer.

UP (Unnumbered Poll): Also called NSP (Non-Sequenced Poll) or ORP (Optional Response Poll). This command is used in the loop configuration.

To enable the SIU to transmit an information frame in AUTO mode, the CPU sets up a transmit buffer. This is done by writing two registers—Transmit Buffer Start (TBS) Address and Transmit Buffer Length (TBL), and filling the transmit buffer with the information to be transmitted.

When the transmit buffer is full, the SIU can automatically (without CPU intervention) send an information frame (I-frame) with the appropriate sequence numbers, when the data link becomes available (when the 8044 is polled for information). After the SIU has transmitted the I-frame, it waits for acknowledgement from the receiving station. If the acknowledgement is negative, the SIU retransmits the frame. If the acknowledgement is positive, the SIU interrupts the CPU, to indicate that the transmit buffer may be reloaded with new information.

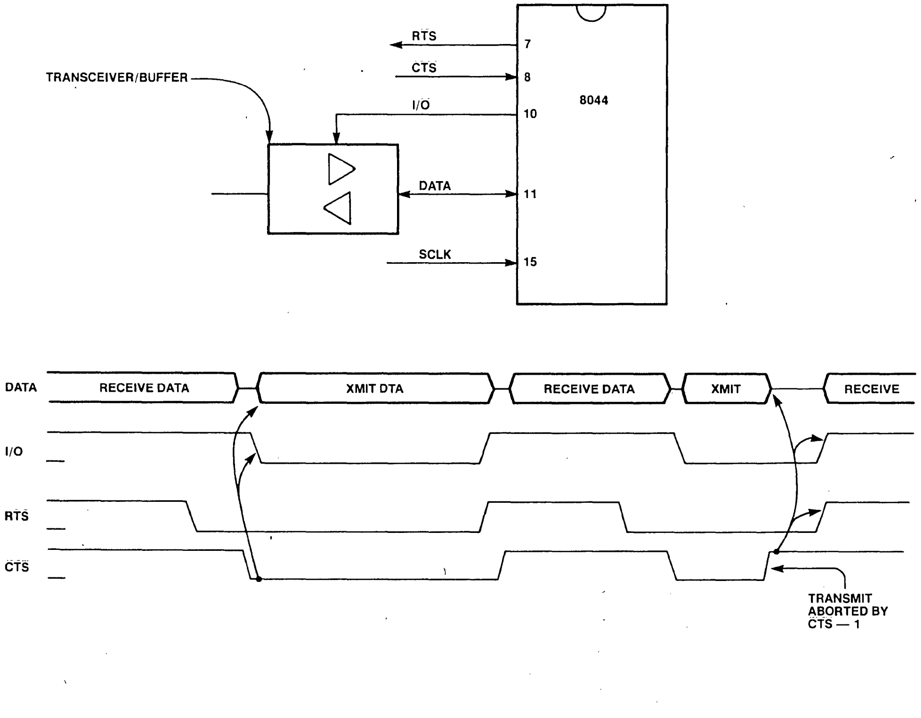


Figure 19-3. Serial Interface Timing—Clocked Mode

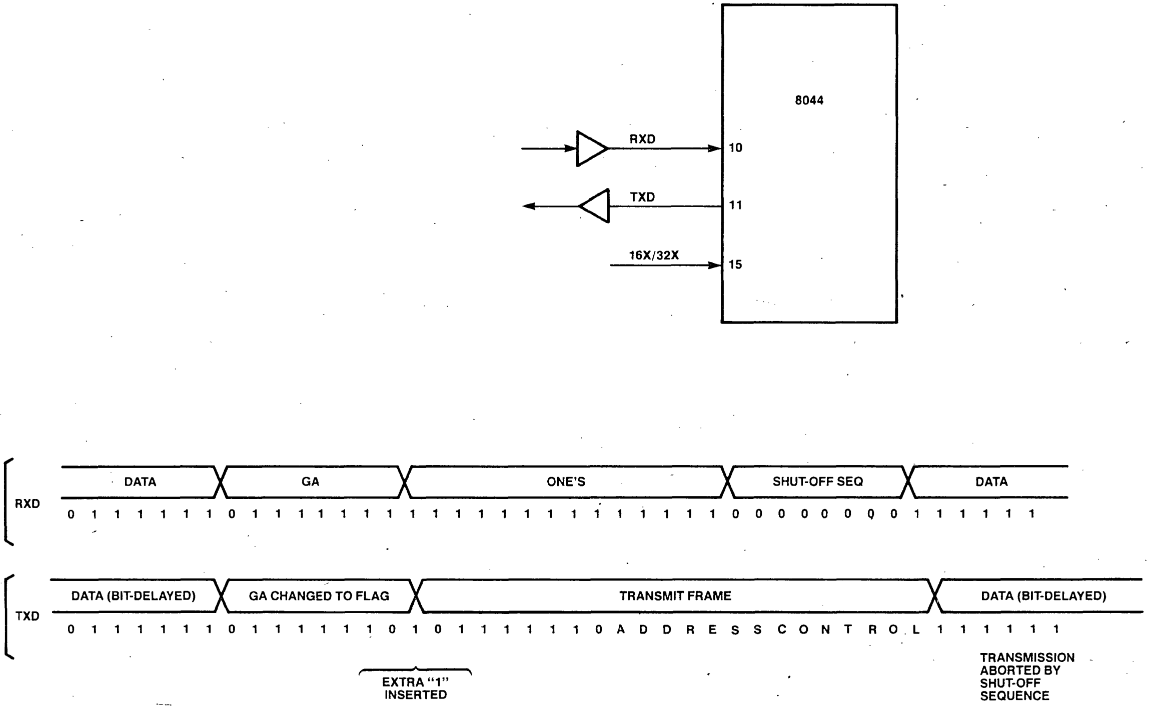


Figure 19-4. Serial Interface Timing—Self Clocked Mode

In addition to transmitting the information frames, the SIU in AUTO mode is capable of sending the following responses to the primary station:

RR (Receive Ready): Acknowledges that the 8044 has correctly received numbered frames up through $N_R - 1$, and that it is ready to receive frame N_R.

RNR (Receive Not Ready): Indicates a temporary busy condition (at the 8044) due to buffering or other internal constraints. The quantity N_R in the control field indicates the number of the frame expected after the busy condition ends, and acknowledges the correct reception of the frames up through $N_R - 1$.

19.4.2 FLEXIBLE Mode

In the FLEXIBLE (or non-auto) mode, all reception and transmission is under the control of the CPU. The full SDLC and HDLC protocols can be implemented, as well as any bit-synchronous variants of these protocols.

FLEXIBLE mode provides more flexibility than AUTO mode, but it requires more CPU overhead, and much longer recognition and response times. This is especially true when the CPU is servicing an interrupt that has higher priority than the interrupts from the SIU.

In FLEXIBLE mode, when the SIU receives a frame, it interrupts the CPU. The CPU then reads the control byte from the Receive Control Byte (RCB) register. If the received frame is an information frame, the CPU also reads the information from the receive buffer, according to the values in the Receive Buffer Start (RBS) address register and the Received Field Length (RFL) register.

In FLEXIBLE mode, the 8044 can initiate transmissions without being polled, and thus it can act as the primary station. To initiate transmission or to generate a response, the CPU sets up and enables the SIU. The SIU then formats and transmits the desired frame. Upon completion of the transmission, without waiting for a positive acknowledgement from the receiving station, the SIU interrupts the CPU.

19.5 8044 FRAME FORMAT OPTIONS

As mentioned above, variations on the basic SDLC frame consist of omitting one or more of the fields. The choice of which fields to omit, as well as the selection of AUTO mode versus FLEXIBLE mode, is specified by the settings of the following three bits in the Serial Mode Register (SMD) and the Status/Control Register (STS):

SMD Bit 0: NFCS (No Frame Check Sequence)

SMD Bit 1: NB (Non-Buffered Mode—No Control Field)

STS Bit 1: AM (AUTO Mode or Addressed Mode)

Figure 19-5 shows how these three bits control the frame format.

The following paragraphs discuss some properties of the standard SDLC format, and the significance of omitting some of the fields.

19.5.1 Standard SDLC Format

The standard SDLC format consists of an opening flag, an 8-bit address field, and 8-bit control field, an n-byte information field, a 16-bit Frame Check Sequence (FCS), and a closing flag. The FCS is based on the CCITT-CRC polynominal ($X^{16} + X^{12} + X^5 + 1$). The address and control fields may not be extended. Within the 8044, the address field is held in the Station Address (STAD) register, and the control field is held in the Receive Control Byte (RCB) or Transmit Control Byte (TCB) register. The standard SDLC format may be used in either AUTO mode or FLEXIBLE mode.

19.5.2 No Control Field (Non-Buffered Mode)

When the control field is not present, the RCB and TCB registers are not used. The information field begins immediately after the address field, or, if the address field is also absent, immediately after the opening flag. The entire information field is stored in the 8044's on-chip RAM. If there is no control field, FLEXIBLE mode must be used. Control information may, of course, be present in the information field, and in this manner the No Control Field option may be used for implementing extended control fields.

19.5.3 No Control Field and No Address Field

The No Address Field option is available only in conjunction with the No Control Field option. The STAD, RCB, and TCB registers are not used. When both these fields are absent, the information field begins immediately after the opening flag. The entire information field is stored in on-chip RAM. FLEXIBLE mode must be used. Formats without an address field have the following applications:

Point-to-point data links (where no addressing is necessary)

Monitoring line activity (receiving all messages regardless of the address field)

Extended addressing

FRAME OPTION	NFCS	NB	AM	FRAME FORMAT
Standard SDLC FLEXIBLE Mode	0	0	0	F A C I FCS F
Standard SDLC AUTO Mode	0	0	1	F A C I FCS F
No Control Field FLEXIBLE Mode	0	1	1	F A I FCS F
No Control Field No Address Field FLEXIBLE Mode	0	1	0	F I FCS F
No FCS Field FLEXIBLE Mode	1	0	0	F A C I F
No FCS Field AUTO Mode	1	0	1	F A C I F
No FCS Field No Control Field FLEXIBLE Mode	1	1	1	F A I F
No FCS Field No Control Field No Address Field FLEXIBLE Mode	1	1	0	F I F

Key to Abbreviations:

F = Flag (01111110)	I = Information Field
A = Address Field	FCS = Frame Check Sequence
C = Control Field	

Note: The AM bit is AUTO mode control bit when NB = 0, and Address Mode control bit when NB = 1.

Figure 19-5. Frame Format Options

19.5.4 No FCS Field

In the normal case (NFCS=0), the last 16 bits before the closing flag are the Frame Check Sequence (FCS) field. These bits are not stored in the 8044's RAM. Rather, they are used to compute a cyclic redundancy check (CRC) on the data in the rest of the frame. A received frame with a CRC error (incorrect FCS) is ignored. In transmission, the FCS field is automatically computed by the SIU, and placed in the transmitted frame just prior to the closing flag.

The NFCS bit (SMD Bit 0) gives the user the capability of overriding this automatic feature. When this bit is set (NFCS=1), all bits from the beginning of the information field to the beginning of the closing flag are treated as part of the information field, and are stored in the on-chip RAM. No FCS checking is done on the received frames, and no FCS is generated for the transmitted frames. The No FCS Field option may be used in conjunction with any of the other options. It is typically used in FLEXIBLE mode, althought it does not strictly include AUTO mode. Use of the No FCS Field option

AUTO Mode may, however, result in SDLC protocol violations, since the data integrity is not checked by the SIU.

Formats without an FCS field have the following applications:

Receiving and transmitting frames without verifying data integrity

Using an alternate data verification algorithm

Using an alternate CRC-16 polynomial (such as $X^{16} + X^{15} + X^2 + 1$), or a 32-bit CRC

Performing data link diagnosis by forcing false CRCs to test error detection mechanisms

In addition to the applications mentioned above, all of the format variations are useful in the support of non-standard bit-synchronous protocols.

19.6 HDLC

In addition to its support of SDLC communications, the 8044 also supports some of the capabilities of HDLC. The following remarks indicate the principal differences between SDLC and HDLC.

HDLC permits any number of bits in the information field, whereas SDLC requires a byte structure (multiple of 8 bits). The 8044 itself operates on byte boundaries, and thus it restricts fields to multiples of 8 bits.

HDLC provides functional extensions to SDLC: an unlimited address field is allowed, and extended frame number sequencing.

HDLC does not support operation in loop configurations.

19.7 SIU SPECIAL FUNCTION REGISTERS

The 8044 CPU communicates with and controls the SIU through hardware registers. These registers are accessed using direct addressing. The SIU special function registers (SIU SFRs) are of three types:

Control and Status Registers

Parameter Registers

ICE Support Registers

19.7.1 Control and Status Registers

There are three SIU Control and Status Registers:

Serial Mode Register (SMD)

Status/Command Register (STS)

Send/Receive Count Register (NSNR)

The SMD, STS, and NSNR registers are all cleared by system reset. This assures that the SIU will power up in an idle state (neither receiving nor transmitting).

These registers and their bit assignments are described below (see also the More Details on Registers section).

SMD: Serial Mode Register (byte-addressable)

Bit:	7	6	5	4	3	2	1	0
	SCM2	SCM1	SCM0	NRZI	LOOP	PFS	NB	NFCS

The Serial Mode Register (Address C9H) selects the operational modes of the SIU. The 8044 CPU can both read and write SMD. The SIU can read SMD but cannot write to it. To prevent conflict between CPU and SIU access to SMD, the CPU should write SMD only when the Request To Send (RTS) and Receive Buffer Empty (RBE) bits (in the STS register) are both false (0). Normally, SMD is accessed only during initialization.

The individual bits of the Serial Mode Register are as follows:

Bit #	Name	Description
SMD.0	NFCS	No FCS field in the SDLC frame.
SMD.1	NB	Non-Buffered mode. No control field in the SDLC frame.
SMD.2	PFS	Pre-Frame Sync mode. In this mode, the 8044 transmits two bytes before the first flag of a frame, for DPLL synchronization. If NRZI is enabled, 00H is sent; otherwise, 55H is sent. In either case, 16 pre-frame transitions are guaranteed.
SMD.3	LOOP	Loop configuration.
SMD.4	NRZI	NRZI coding option.
SMD.5	SCM0	Select Clock Mode — Bit 0
SMD.6	SCM1	Select Clock Mode — Bit 1
SMD.7	SCM2	Select Clock Mode — Bit 2

The SCM bits decode as follows:

SCM				Data Rate
2	1	0	Clock Mode	(Bits/sec)*
0	0	0	Externally clocked	0–2.4M**
0	0	1	Undefined	
0	1	0	Self clocked, timer overflow	244–62.5K
0	1	1	Undefined	
1	0	0	Self clocked, external 16x	0–375K

SCM				Data Rate
2	1	0	Clock Mode	(Bits/sec)*
1	0	1	Self clocked, external 32x	0-187.5K
1	1	0	Self clocked, internal fixed	375K
1	1	1	Self clocked, internal fixed	187.5k

*Based on a 12 Mhz crystal frequency
**0-1M bps in loop configuration

STS: Status/Command Register (bit-addressable)

Bit:	7	6	5	4	3	2	1	0
	TBF	RBE	RTS	SI	BOV	OPB	AM	RBP

The Status/Command Register (Address C8H) provides operational control of the SIU by the 8044 CPU, and enables the SIU to post status information for the CPU's access. The SIU can read STS, and can alter certain bits, as indicated below. The CPU can both read and write STS asynchronously. However, 2-cycle instructions that access STS during both cycles ('JBC/B, REL' and 'MOV /B,C.') should not be used, since the SIU may write to STS between the two CPU accesses.

The individual bits of the Status/Command Register are as follows:

Bit #	Name	Description
STS.0	RBP	Receive Buffer Protect. Inhibits writing of data into the receive buffer. In AUTO mode, RBP forces an RNR response instead of an RR.
STS.1	AM	AUTO Mode/Addressed Mode. Selects AUTO mode where AUTO mode is allowed. If NB is true, (=1), the AM bit selects the addressed mode. AM may be cleared by the SIU.
STS.2	OPB	Optional Poll Bit. Determines whether the SIU will generate an AUTO response to an optional poll (UP with P=0). OPB may be set or cleared by the SIU.
STS.3	BOV	Receive Buffer Overrun. BOV may be set or cleared by the SIU.
STS.4	SI	SIU Interrupt. This is one of the five interrupt sources to the CPU. The vector location = 23H. SI may be set by the SIU. It should be cleared by the CPU before returning from an interrupt routine.
STS.5	RTS	Request To Send. Indicates that the 8044 is ready to transmit or is transmitting. RTS may be read or written by the CPU. RTS may be read by the SIU, and in AUTO mode may be written by the SIU.
STS.6	RBE	Receive Buffer Empty. RBE can be thought of as Receive Enable. RBE is set to one by the CPU when it is ready to receive a frame, or has just read the buffer, and to zero by the SIU when a frame has been received.
STS.7	TBF	Transmit Buffer Full. Written by the CPU to indicate that it has filled the transmit buffer. TBF may be cleared by the SIU.

NSNR: Send/Receive Count Register (bit-addressable)

Bit:	7	6	5	4	3	2	1	0
	NS2	NS1	NS0	SES	NR2	NR1	NR0	SER

The Send/Receive Count Register (Address D8H) contains the transmit and receive sequence numbers, plus tally error indications. The SIU can both read and write NSNR. The 8044 CPU can both read and write NSNR asynchronously. However, 2-cycle instructions that access NSNR during both cycles ('JBC /B, REL', and 'MOV /B,C') should not be used, since the SIU may write to NSNR between the two 8044 CPU accesses.

The individual bits of the Send/Receive Count Register are as follows:

Bit #	Name	Description
NSNR.0	SER	Receive Sequence Error: NS (P) ≠ NR (S)
NSNR.1	NR0	Receive Sequence Counter—Bit 0
NSNR.2	NR1	Receive Sequence Counter—Bit 1
NSNR.3	NR2	Receive Sequence Counter—Bit 2
NSNR.4	SES	Send Sequence Error: NP (P) ≠ NS (S) and NP (P) ≠ NS (S) + 1
NSNR.5	NS0	Send Sequence Counter — Bit 0
NSNR.6	NS1	Send Sequence Counter — Bit 1
NSNR.7	NS2	Send Sequence Counter — Bit 2

19.7.2 Parameter Registers

There are eight parameter registers that are used in connection with SIU operation. All eight registers may be read or written by the 8044 CPU. RFL and RCB are normally loaded by the SIU.

The eight parameter registers are as follows:

STAD: Station Address Register (byte-addressable)

The Station Address register (Address CEH) contains the station address. To prevent access conflict, the CPU should access STAD only when the SIU is idle (RTS=0 and RBE=0). Normally, STAD is accessed only during. initialization.

TBS: Transmit Buffer Start Address Register (byte-addressable)

The Transmit Buffer Start address register (Address DCH) points to the location in on-chip RAM for the beginning of the I-field of the frame to be transmitted. The CPU should access TBS only when the SIU is not transmitting a frame (when TBF=0).

TBL: Transmit Buffer Length Register (byte-addressable)

The Transmit Buffer Length register (Address DBH) contains the length (in bytes) of the I-field to be transmitted. A blank I-field (TBL=0) is valid. The CPU should access TBL only when the SIU is not transmitting a frame (when TBF=0).

NOTE: The transmit and recieve buffers are not allowed to "wrap around" in the on-chip RAM. A "buffer end" is automatically generated if address 191 (BFH) is reached.

TCB: Transmit Control Byte Register (byte-addressable)

The Transmit Control Byte register (Address DAH) contains the byte which is to be placed in the control field of the transmitted frame, during NON-AUTO mode transmission. The CPU should access TCB only when the SIU is not transmitting a frame (when TBF=0). The N_S and N_R counters are not used in the NON-AUTO mode.

RBS: Receive Buffer Start Address Register (byte-addressable)

The Receive Buffer Start address register (Address CCH) points to the location in on-chip RAM where the beginning of the I-field of the frame being received is to be stored. The CPU should write RBS only when the SIU is not receiving a frame (when RBE=0).

RBL: Receive Buffer Length Register (byte-addressable)

The Receive Buffer Length register (Address CBH) contains the length (in bytes) of the area in on-chip RAM allocated for the received I-field. RBL=0 is valid. The CPU should write RBL only when RBE=0.

RFL: Receive Field Length Register (byte-addressable)

The Received Field Length register (Address CDH) contains the length (in bytes) of the received I -field that has just been loaded into on-chip RAM. RFL is loaded by the SIU. RFL=0 is valid. RFL should be accesssed by the CPU only when RBE=0.

RCB: Receive Control Byte Register (byte-addressable)

The Received Control Byte register (Address CAH) contains the control field of the frame that has just been received. RCB is loaded by the SIU. The CPU can only read RCB, and should only access RCB when RBE=0.

19.7.3 ICE Support Registers

The 8044 In-Circuit Emulator (ICE-44) allows the user to exercise the 8044 application system and monitor the execution of instructions in real time.

The emulator operates with Intel's Intellec® development system. The development system interfaces with the user's 8044 system through an in-cable buffer box. The cable terminates in a 8044 pin-compatible plug, which fits into the 8044 socket in the user's system. With the emulator plug in place, the user can excercise his system in real time while collecting up to 255 instruction cycles of real-time data. In addition, he can single-step the program.

Static RAM is available (in the in-cable buffer box) to emulate the 8044 internal and external program memory and external data memory. The designer can display and alter the contents of the replacement memory in the buffer box, the internal data memory, and the internal 8044 registers, including the SFRs.

Among the SIU SFRs are the following registers that support the operation of the ICE:

DMA CNT: DMA Count Register (byte-addressable)

The DMA Count register (Address CFH) indicates the number of bytes remaining in the information block that is currently being used.

FIFO: Three-Byte (byte-addressable)

The Three-Byte FIFO (Address DDH, DEH, and DFH) is used between the eight-bit shift register and the information buffer when an information block is received.

SIUST: SIU State Counter (byte-addressable)

The SIU State Counter (Address D9H) reflects the state of the internal logic which is under SIU control. Therefore, care must be taken not to write into this register.

The SIUST register can serve as a helpful aid to determine which field of a receive frame that the SIU expects next. The table below will help in debugging 8044 reception problems.

SIUST VALUE	FUNCTION
01H	Waiting for opening flag.
08H	Waiting for address field.
10H	Waiting for control field.
18H	Waiting for first byte of I field. This state is only entered if a FCS is expected. It pushes the received byte onto the top of the FIFO.
20H	Wating for second byte of I field. This state always follows state 18H.
28H	Waiting for I field byte. This sate can be entered from state 20H or from states 01H, 08H, or 10H depending upon the SIU's mode configuration. (Each time a byte is received, it is pushed onto the top of the FIFO and the byte at the bottom is put into memory. For no FCS formatted frames, the FIFO is collapsed into a single register).
30H	Waiting for the closing flag after having overflowed the receive buffer. Note that even if the receive frame overflows the assigned receive buffer length, the FCS is still checked.

Examples of SIUST status sequences for different frame formats are shown below. Note that status changes after acceptance of the received field byte.

19.8 OPERATION

The SIU is initialized by a reset signal (on pin 9), followed by write operations to the SIU SFRs. Once initialized, the SIU can function in AUTO mode or NON-AUTO mode. Details are given below.

Table 19-1. SIUST Status Sequences

										Frame Option		
										NFCS	NB	AM
Example 1:												
Frame Format	(Idle)	F	A	C	I			FCS	F	0	0	1
SIUST Value	01	01	08	10	18	20	28	28	01			
Example 2:												
Frame Format	(Idle)	F	A	I			FCS	F		0	1	1
SIUST Value	01	01	08	18	20	28	28	01				
Example 3:												
Frame Format	(Idle)	F	I			FCS	F			0	1	0
SIUST Value	01	01	18	20	28	28	01					
Example 4:												
Frame Format	(Idle)	F	A	I	F					1	1	1
SIUST Value	01	01	08	28	01							
Example 5:												
Frame Format	(Idle)	F	I	F						1	1	0
SIUST Value	01	01	28	01								
Example 6:												
Frame Format	(Idle)	F	I			I OVERFLOW	FCS	F		0	1	0
SIUST Value	01	01	18	20	28	30	30	01				

19.8.1 Initialization

Figure 19-6 is the SIU. Registers SMD, STS, and NSNR are cleared by reset. This puts the 8044 into an idle state—neither receiving nor transmitting. The following registers must be initialized before the 8044 leaves the idle state:

STAD—to establish the 8044's SDLC station address.

SMD—to configure the 8044 for the proper operating mode.

RBS, RBL—to define the area in RAM allocated for the Receive Buffer.

TBS, TBL—to define the area in RAM allocated for the Transmit Buffer.

Once these registers have been initialized, the user may write to the STS register to enable the SIU to leave the idle state, and to begin transmits and/or receives.

Setting RBE to 1 enables the SIU for receive. When RBE = 1, the SIU monitors the received data stream for a flag pattern. When a flag pattern is found, the SIU enters Receive mode and receives the frame.

Setting RTS to 1 enables the SIU for transmit. When RTS = 1, the SIU monitors the received data stream for a GA pattern (loop configuration) or waits for a CTS

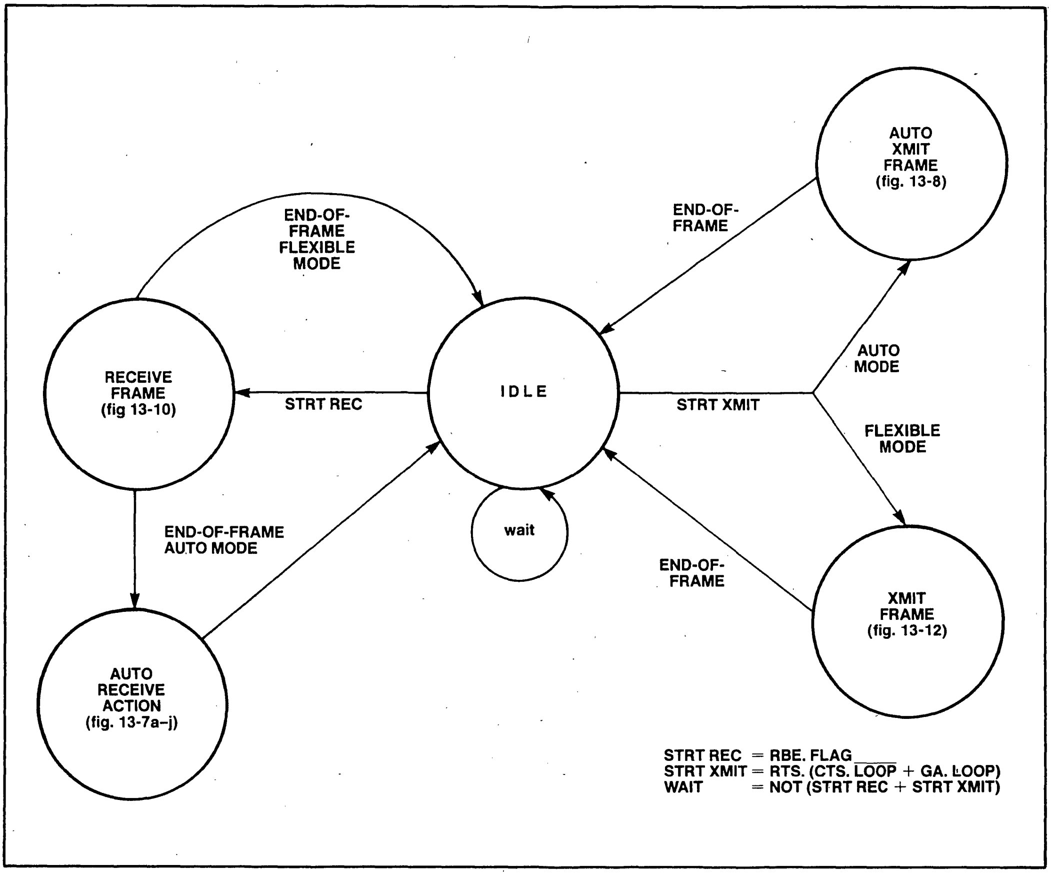

AUTO XMIT FRAME (fig. 13-8)
END-OF-FRAME
END-OF-FRAME FLEXIBLE MODE
RECEIVE FRAME (fig 13-10)
STRT REC
IDLE
STRT XMIT
AUTO MODE
FLEXIBLE MODE
wait
END-OF-FRAME AUTO MODE
END-OF-FRAME
XMIT FRAME (fig. 13-12)
AUTO RECEIVE ACTION (fig. 13-7a–j)
STRT REC = RBE. FLAG
STRT XMIT = RTS. (CTS. LOOP + GA. LOOP)
WAIT = NOT (STRT REC + STRT XMIT)

Figure 19-6. SIU State Diagram

(non-loop configuration). When the GA or CTS arrives, the SIU enters Transmit mode and transmits a frame.

In AUTO mode, the SIU sets RTS to enable automatic transmissions of appropriate responses.

19.8.2 AUTO Mode

Figure 19-7 illustrates the receive operations in AUTO mode. The overall operation is shown in Figure 19-7a. Particular cases are illustrated in Figures 19-7b through 19-7j. If any Unnumbered Command other than UP is received, the AM bit is cleared and the SIU responds as if in the FLEXIBLE mode, by interrupting the CPU for supervision. This will also happen if a BOV or SES condition occurs. If the received frame contains a poll, the SIU sets the RTS bit to generate a response.

Figure 19-8 illustrates the transmit operations in AUTO mode. When the SIU gets the opportunity to transmit, and if the transmit buffer is full, it sends an I-frame. Otherwise, it sends an RR if the buffer is free, or an RNR if the buffer is protected. The sequence counters NS and NR are used to construct the appropriate control fields.

Figure 19-9 shows how the CPU responds to an SI (serial interrupt) in AUTO mode. The CPU tests the AM bit (in the STS register). If AM = 1, it indicates that the SIU has received either an I-frame, or a positive response to a previously transmitted I-frame.

19.8.3 FLEXIBLE Mode

Figure 19-10 illustrates the receive operations in NON-AUTO mode. When the SIU successfully completes a task, it clears RBF and interrupts the CPU by setting SI to 1. The exact CPU response to SI is determined by software. A typical response is shown in Figure 19-11.

Figure 19-12 illustrates the transmit operations in FLEXIBLE mode. The SIU does not wait for a positive acknowledge response to the transmitted frame. Rather, it interrupts the CPU (by setting SI to 1) as soon as it finishes transmitting the frame. The exact CPU response to SI is determined by software. A typical response is shown in Figure 19-13. This response results in another transmit frame being set up. The sequence of operations shown in Figure 19-13 can also be initiated by the CPU, without an SI. Thus the CPU can initiate a transmission in FLEXIBLE mode without a poll, simply by setting the RTS bit in the STS register. The RTS bit is always used to initiate a transmission, but it is applied to the RTS pin only when a non-loop configuration is used.

19.8.4 8044 Data Link Particulars

The following facts should be noted:

1) In a non-loop configuration, one or two bits are transmitted before the opening flag. This is necessary for NRZI synchronization.

2) In a non-loop configuration, one to eight extra dribble bits are transmitted after the closing flag. These bits are a zero followed by ones.

3) In a loop configuration, when a GA is received and the 8044 begins transmitting, the sequence is 0111111010111110... (FLAG, 1, FLAG, ADDRESS, etc.). The first flag is created from the GA. The second flag begins the message.

4) CTS is sampled after the rising edge of the serial data, at about the center of the bit cell, except during a non-loop, externally clocked mode transmit, in which case it is sampled just after the falling edge.

5) The SIU does not check for illegal I-fields. In particular, if a supervisory command is received in AUTO mode, and if there is also an I-field, it will be loaded into the receive buffer (if RBP=0), but it cannot cause a BOV.

6) In relation to the Receive Buffer Protect facility, the user should set RFL to 0 when clearing RBP, such that, if the SIU is in the process of receiving a frame, RFL will indicate the proper value when reception of the frame has been completed.

19.8.5 Turn Around Timing

In AUTO mode, the SIU generates an RTS immediately upon being polled. Assuming that the 8044 sends an information frame in response to the poll, the primary station sends back an acknowledgement. If, in this acknowledgement, the 8044 is polled again, a response may be generated even before the CPU gets around to processing the interrupt caused by the acknowledge. In such a case, the response would be an RR (or RNR), since TBF would have been set to 0 by the SIU, due to the acknowledge.

If the system designer does not wish to take up channel time with RR responses, but prefers to generate a new I-frame as a response, there are several ways to accomplish this:

1) Operate the 8044 in FLEXIBLE mode.

2) Specify that the master should never acknowledge and poll in one message. This is typically how a loop system operates, with the poll operation confined to the UP command. This leaves plenty of time for the

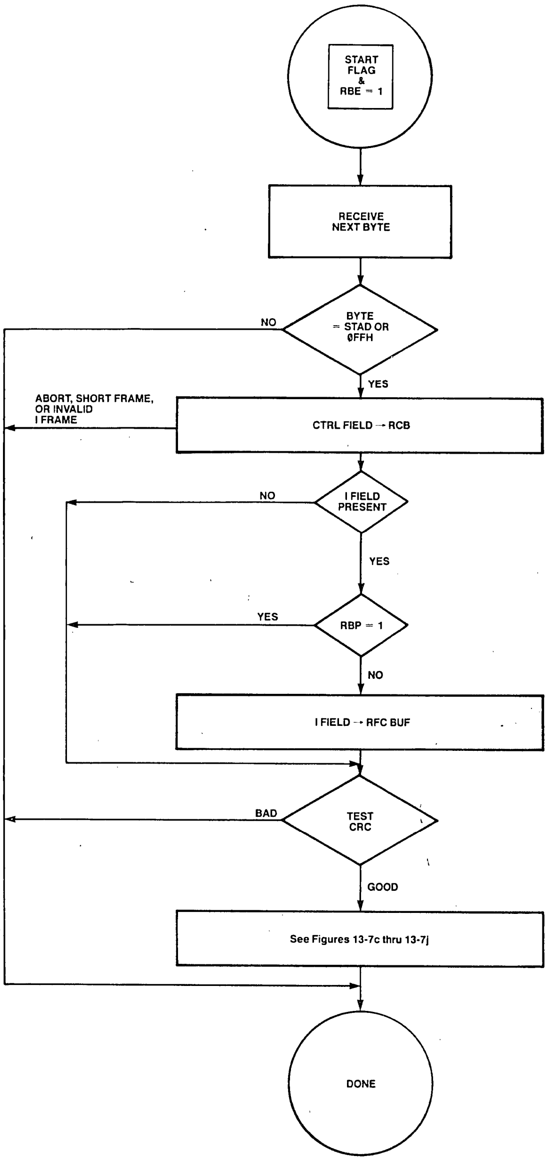


Figure 19-7a. SIU AUTO Mode Receive Flowchart—General

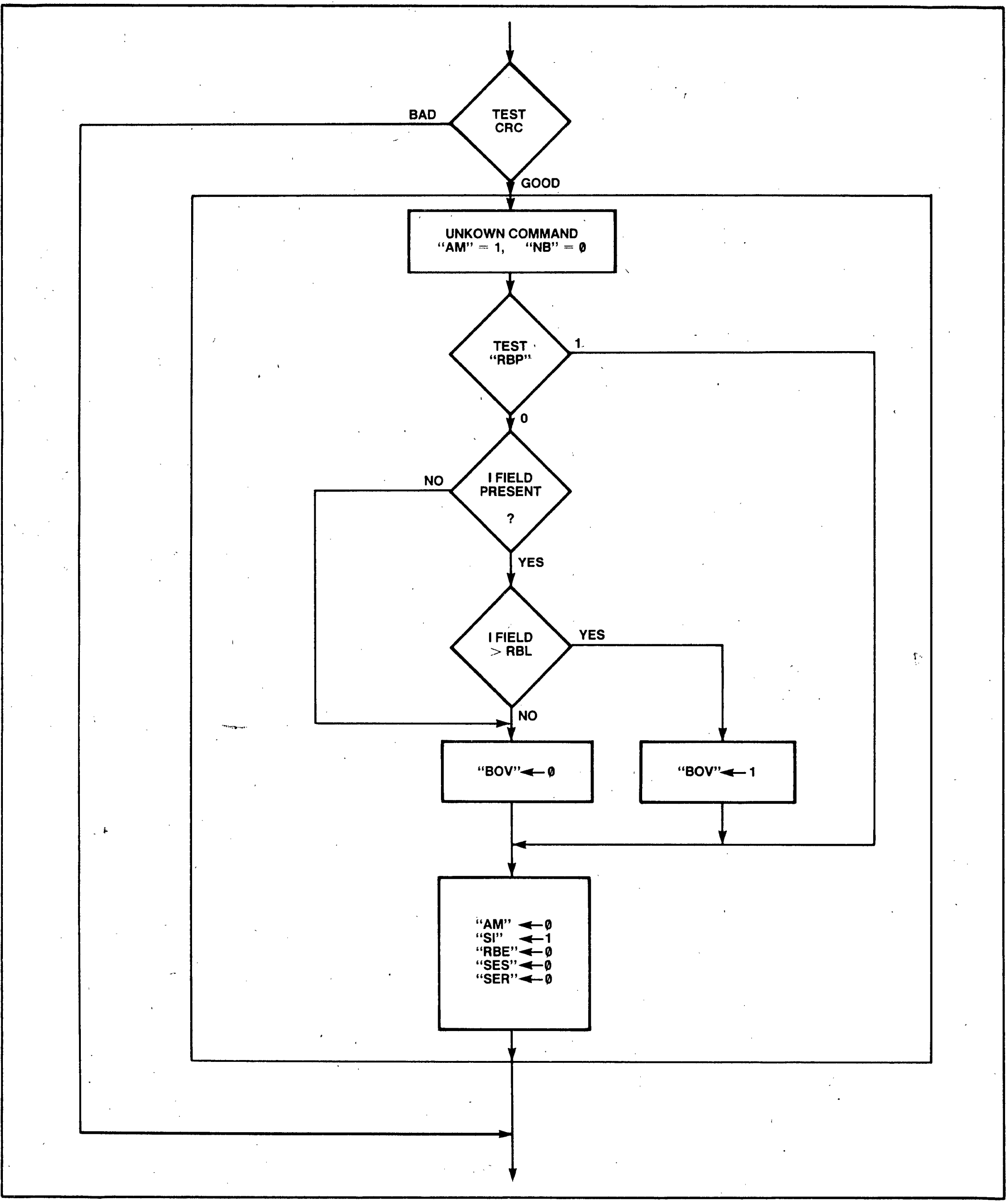


Figure 19-7b. SIU AUTO Mode Receive Flowchart—Unknown Command

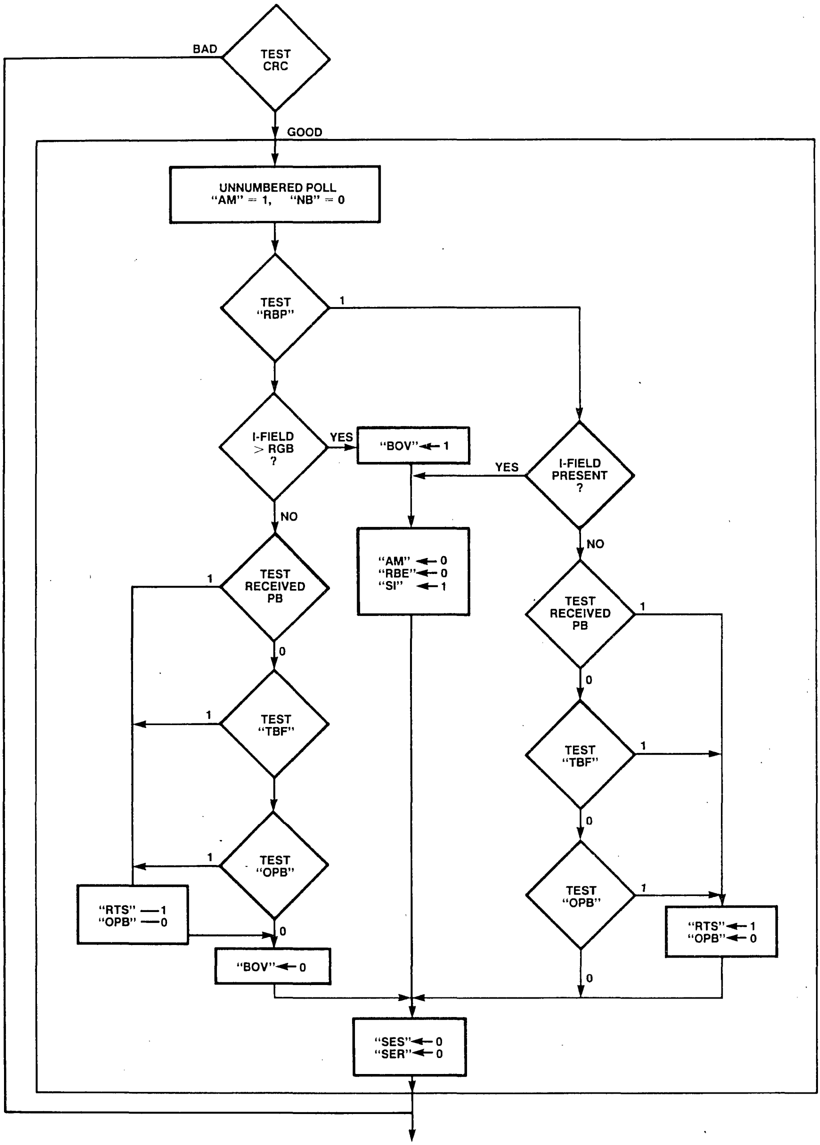


Figure 19-7c. SIU AUTO Mode Receive Flowchart—Unnumbered Poll

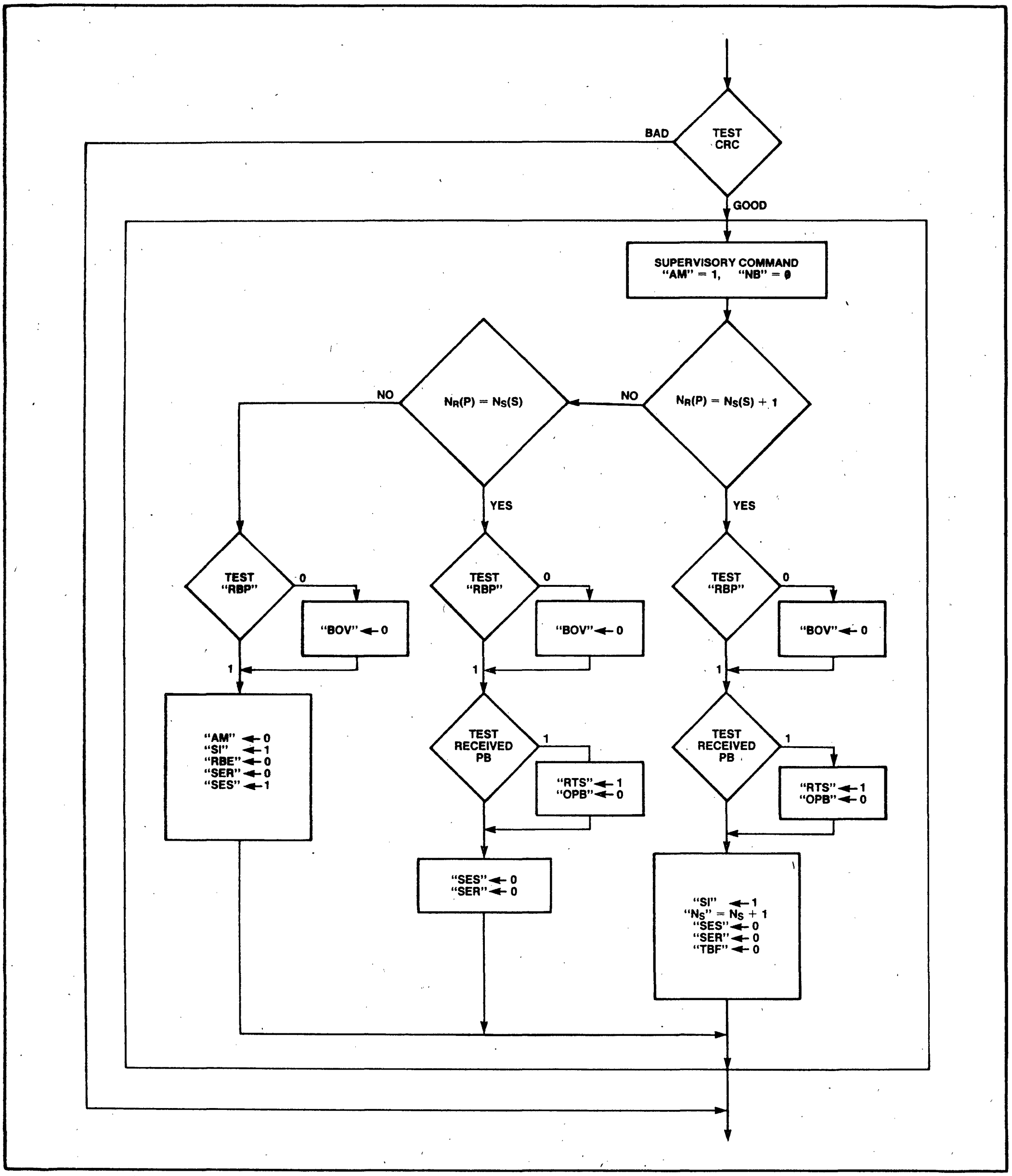


Figure 19-7d. SIU AUTO Mode Receive Flowchart—Supervisory Command

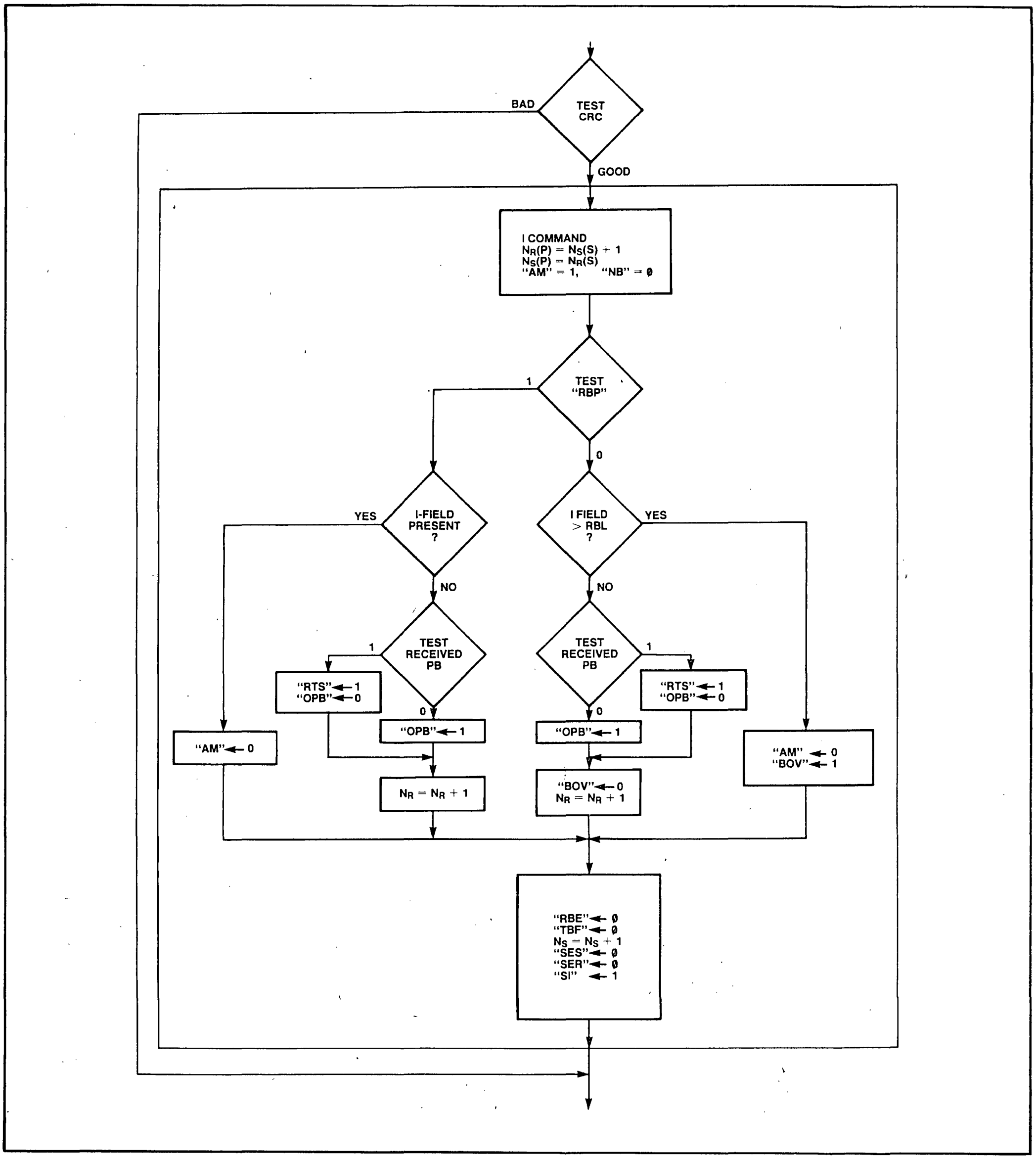


Figure 19-7e. SIU AUTO Mode Receive Flowchart—I Command: Prior Transmitted I-Field Confirmed, Current Received I-Field in Sequence

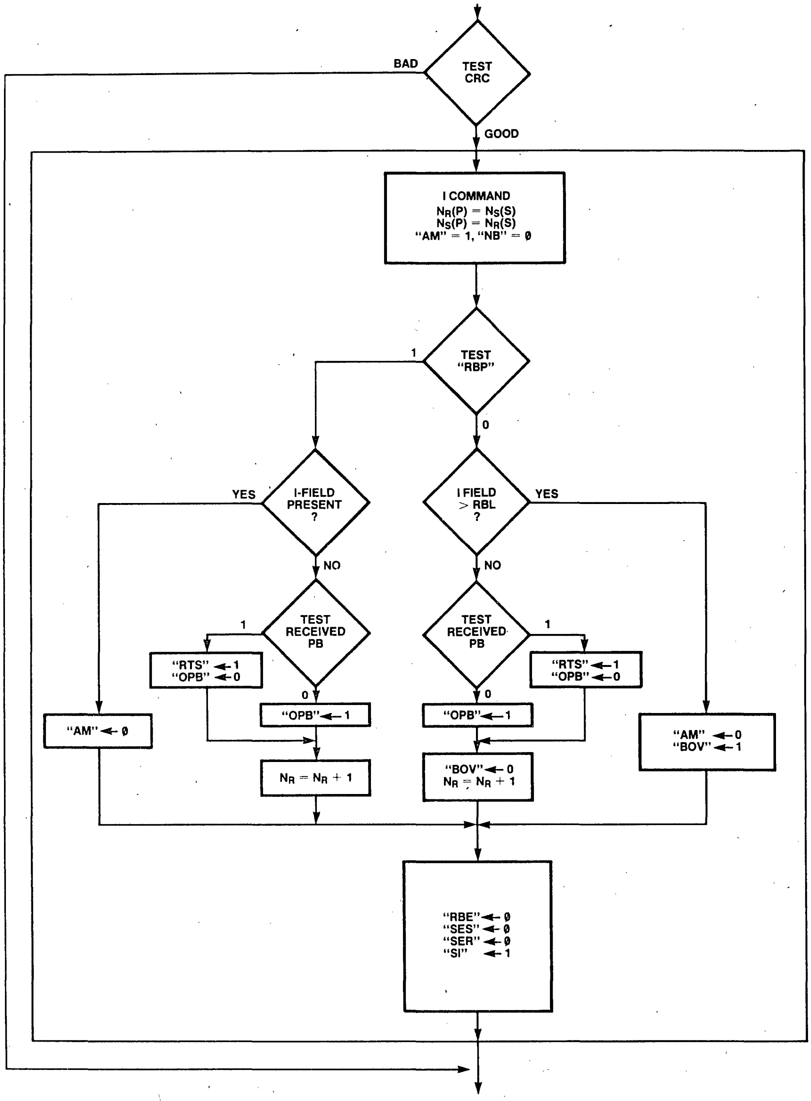


Figure 19-7f. SIU AUTO Mode Receive Flowchart—I Command: Prior Transmitted I-Field Not Confirmed, Current Received I-Field in Sequence

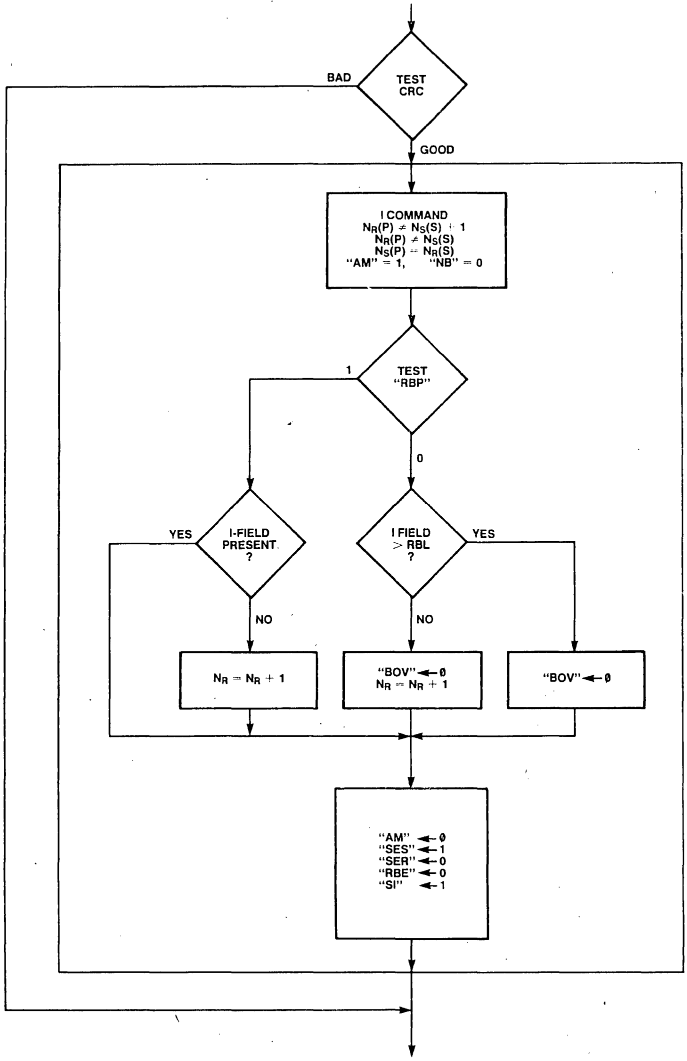


Figure 19-7g. SIU AUTO Mode Receive Flowchart—I Command: Sequence Error Send, Current Received I-Field in Sequence

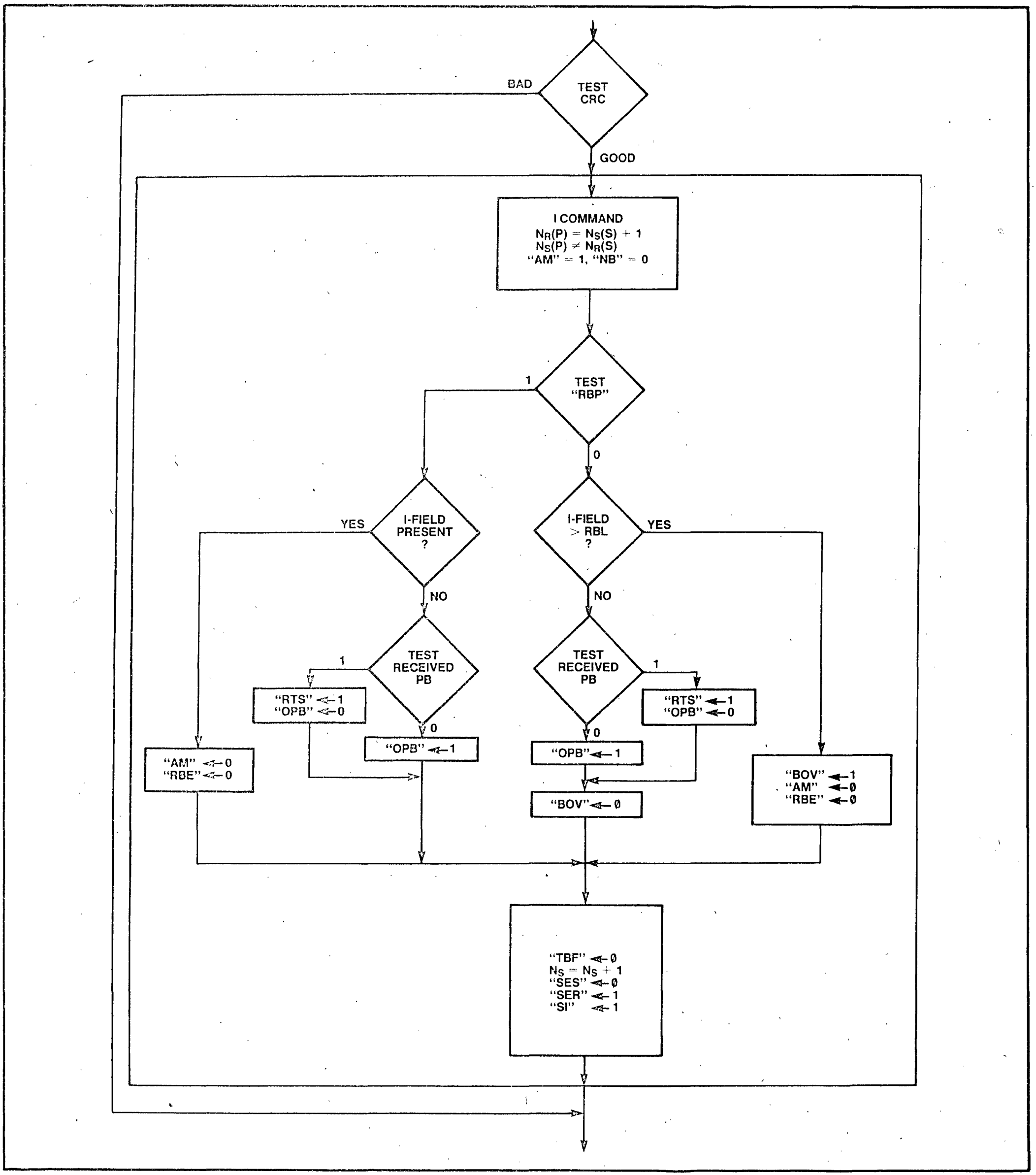


Figure 19-7h. SIU AUTO Mode Receive Flowchart—I Command: Prior Transmitted I-Field Confirmed Sequence Error Receive

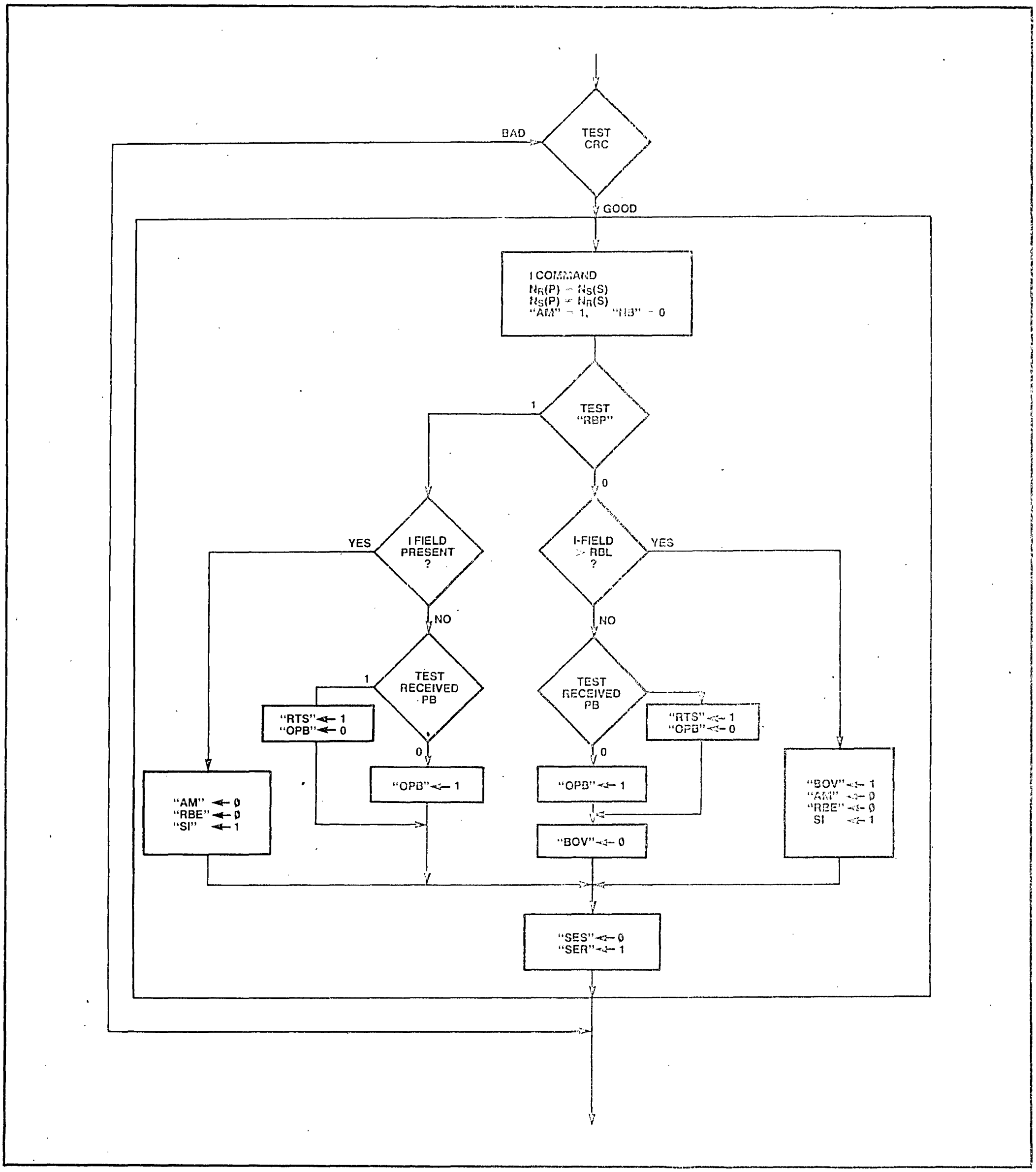


Figure 19-7i. SIU AUTO Mode Receive Flowchart—I Command: Prior Transmitted I-Field Not Confirmed, Sequence Error Receive

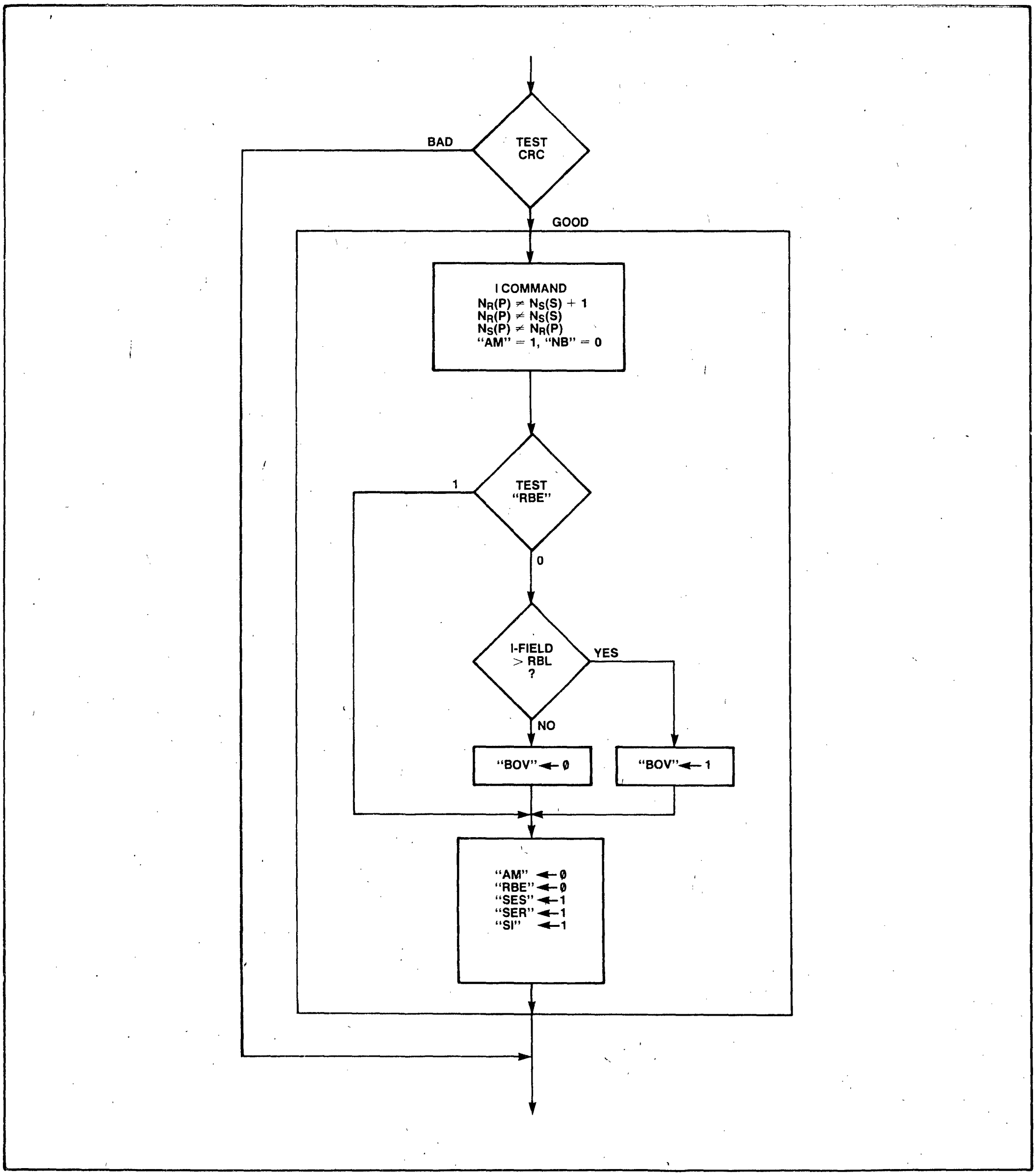


Figure 19-7j. SIU SUTO Mode Receive Flowchart—I Command: Sequence Error Send and Sequence Error Receive

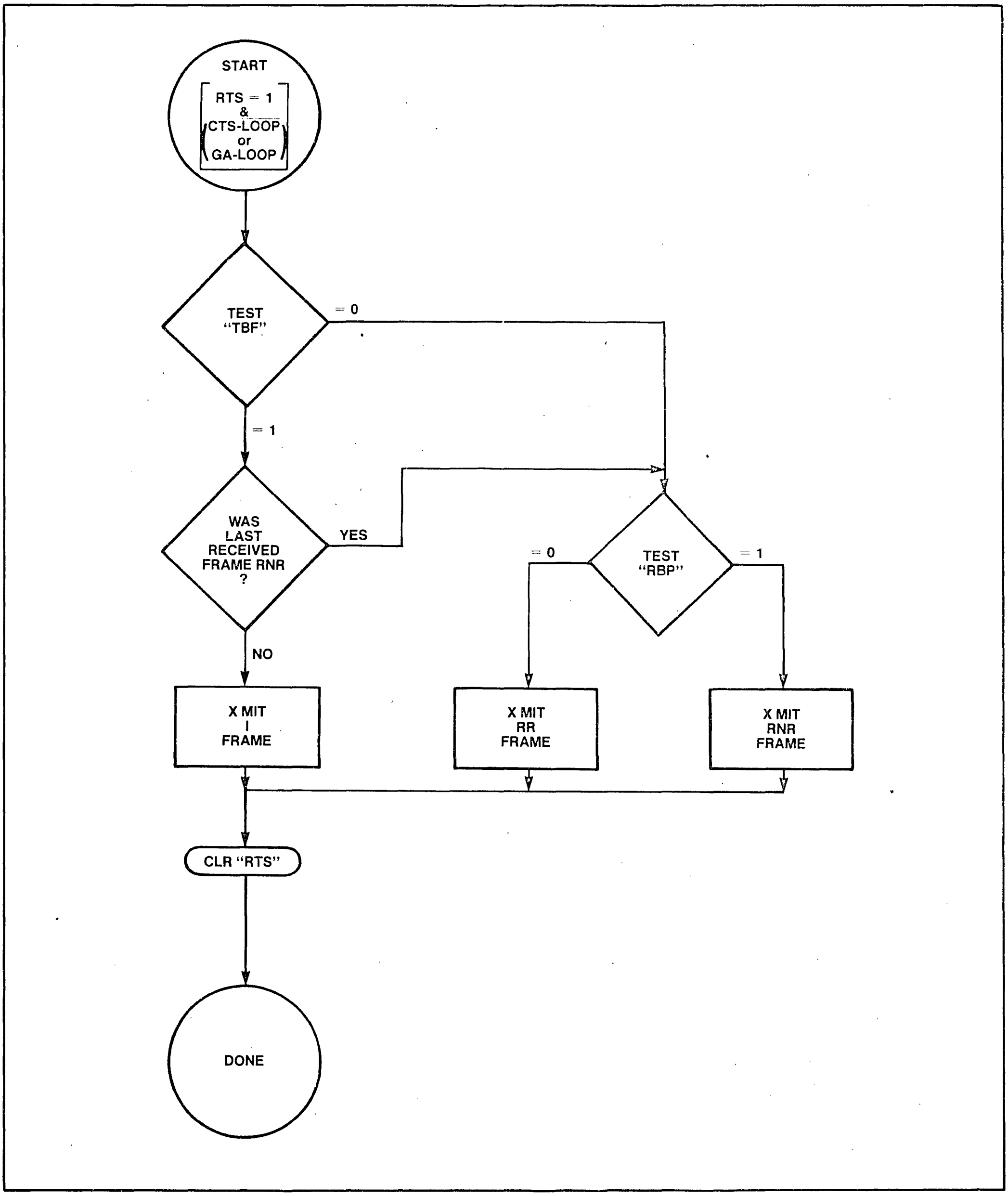


Figure 19-8. SIU AUTO Mode Transmit Flowchart

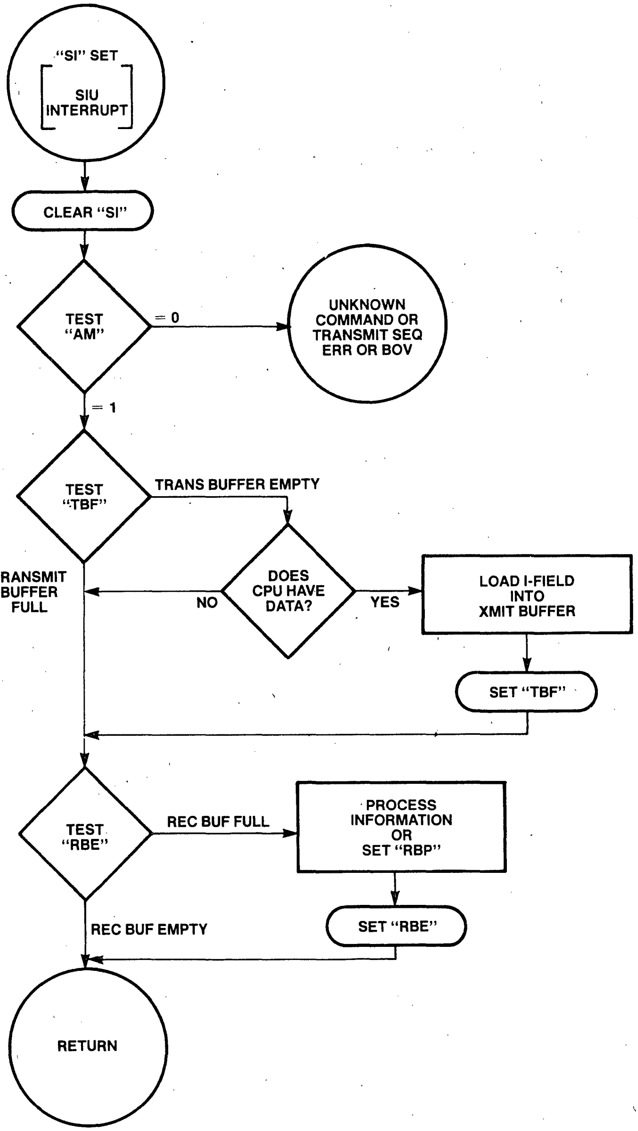


Figure 19-9. AUTO Mode Response to "SI"

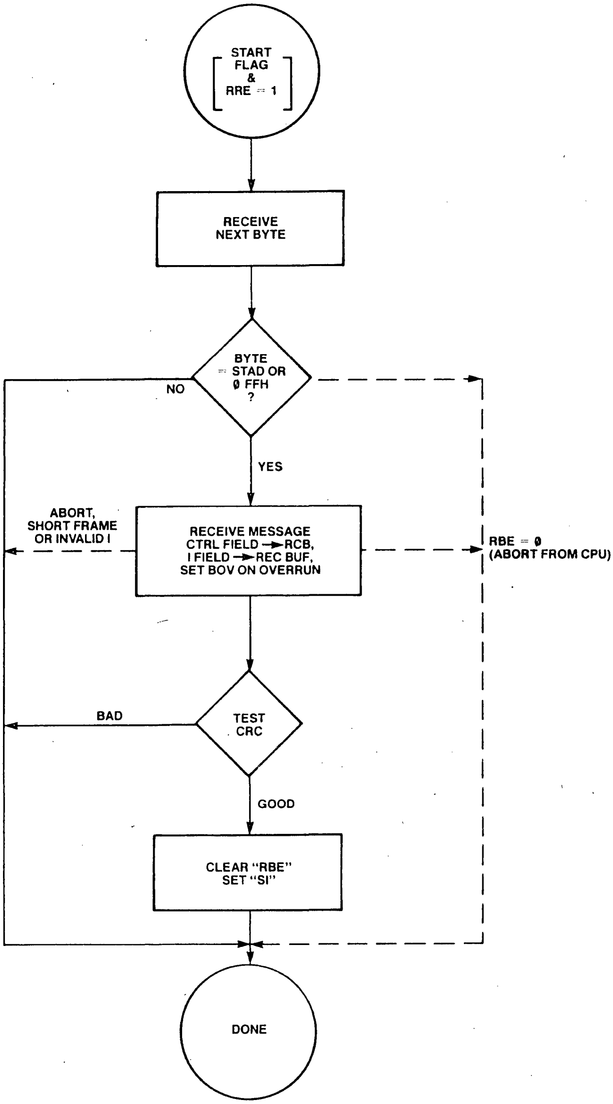


Figure 19-10. SIU FLEXIBLE Mode Receive Flowchart

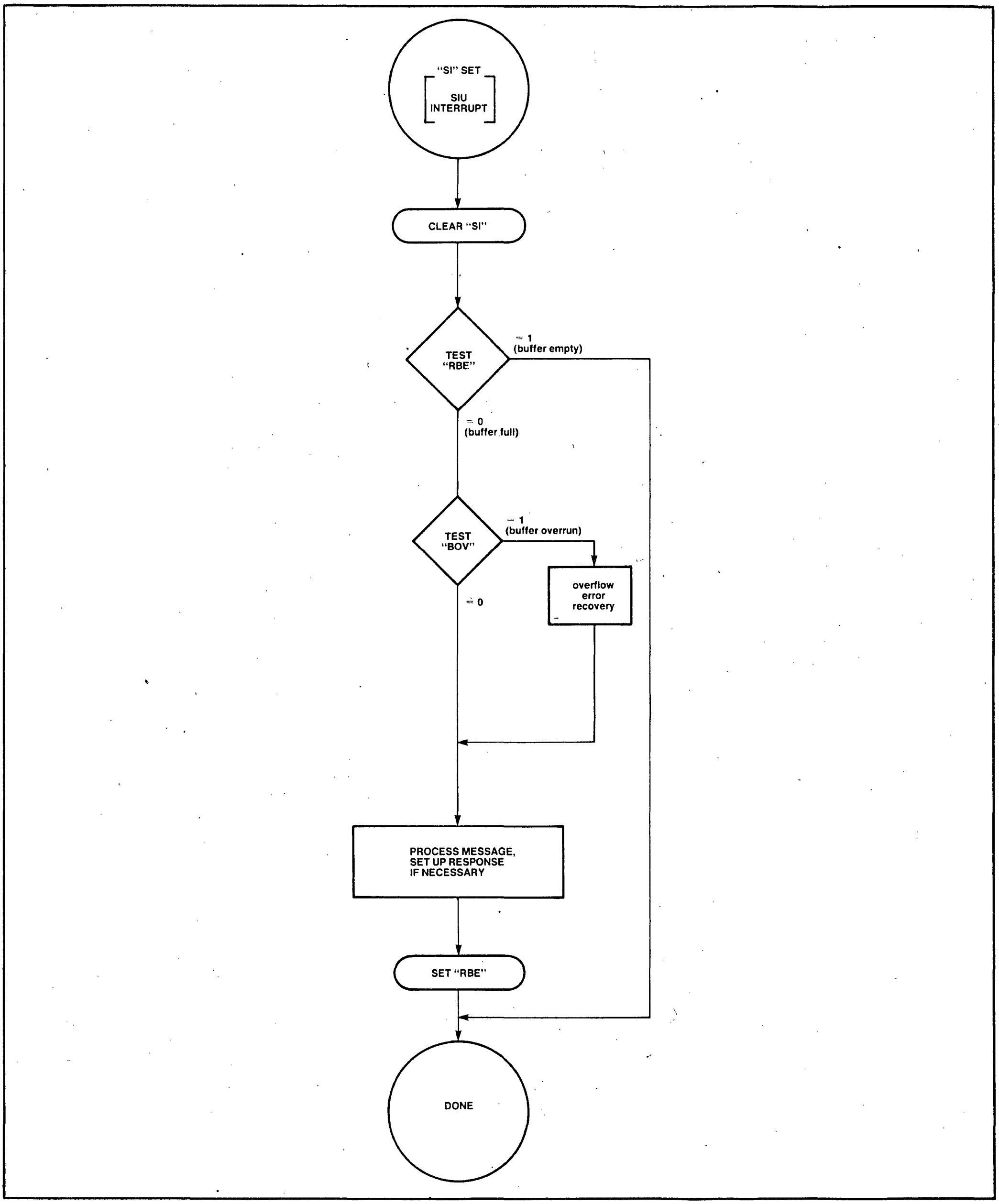


Figure 19-11. FLEXIBLE Mode Response to Receive "SI"

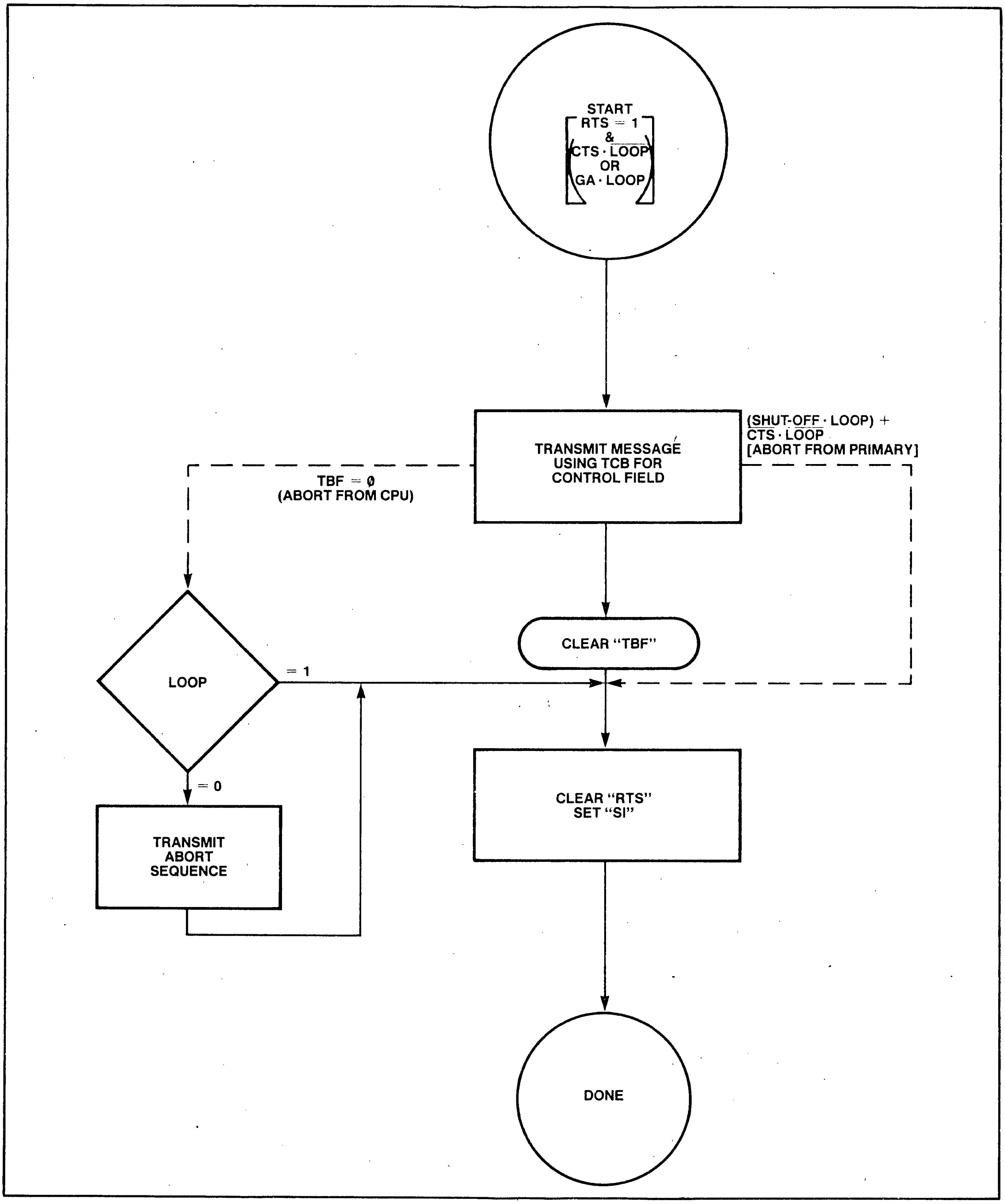


Figure 19-12. SIU FLEXIBLE Mode Transmit Flowchart

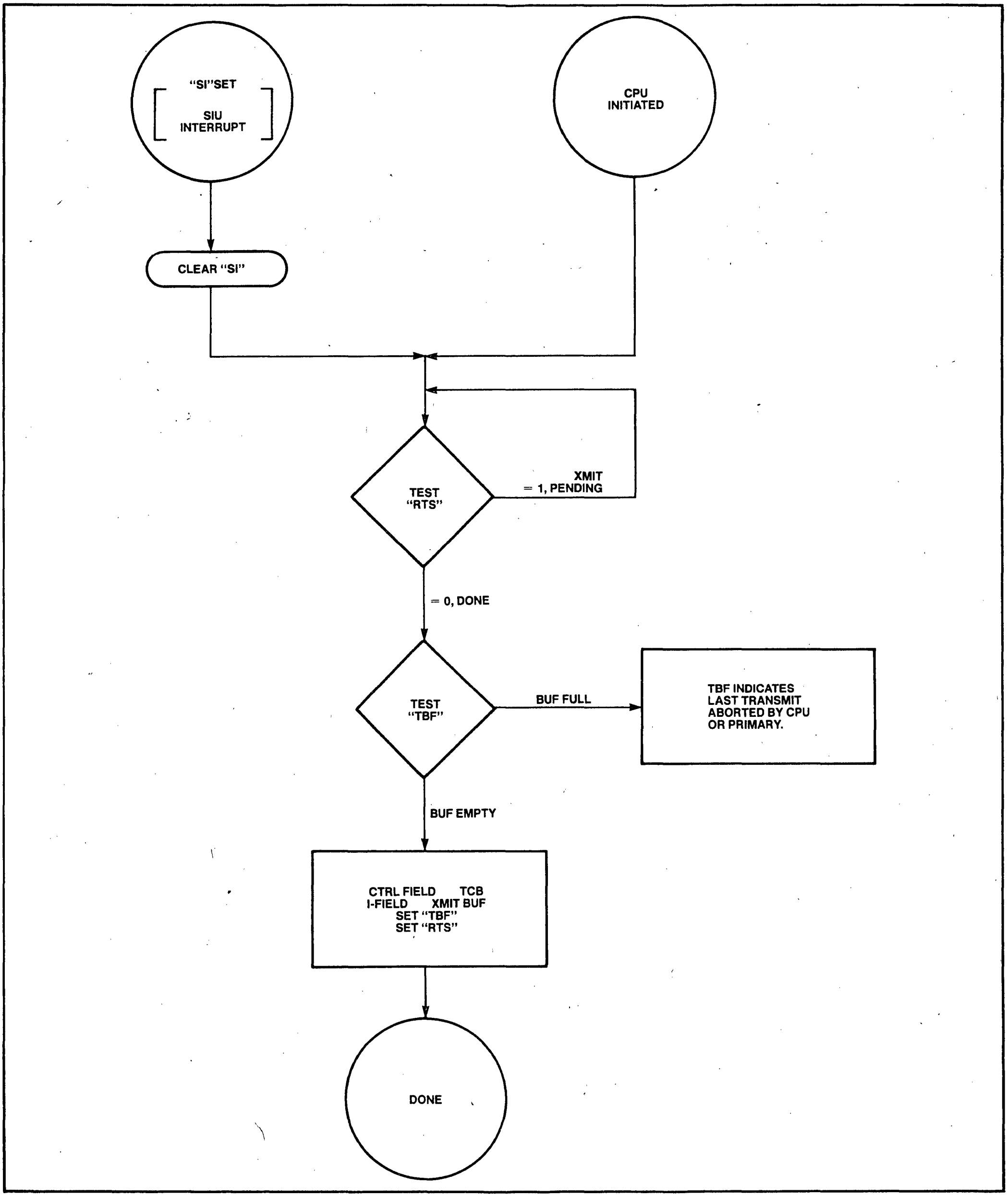


Figure 19-13. FLEXIBLE Mode Response to Transmit "SI"

8044 to get its transmit buffer loaded with new information after an acknowledge.

3) The 8044 CPU can clear RTS. This will prevent a response from being sent, or abort it if it is already in progress. A system using external RTS/CTS handshaking could use a one-shot to delay RTS or CTS, thereby giving the CPU more time to disable the response.

19.9 MORE DETAILS ON SIU HARDWARE

The SIU divides functionally into two sections—a bit processor (BIP) and a byte processor (BYP)—sharing some common timing and control logic. As shown in Figure 19-14, the BIP operates between the serial port pins and the SIU bus, and performs all functions necessary to transmit/receive a byte of data to/from the serial data stream. These operations include shifting, NRZI encoding/decoding, zero insertion/deletion, and FCS generation/checking. The BYP manipulates bytes of data to perform message formatting, and other transmitting and receiving functions. It operates between the SIU bus (SIB) and the 8044's internal bus (IB). The interface between the SIU and the CPU involves an interrupt and some locations in on-chip RAM space which are managed by the BYP.

The maximum possible data rate for the serial port is limited to 1/2 the internal clock rate. This limit is imposed by both the maximum rate of DMA to the on-chip RAM, and by the requirements of synchronizing to an external clock. The internal clock rate for an 8044 running on a 12 MHz crystal is 6 MHz. Thus the maximum 8044 serial data rate is 3 MHz. This data rate drops down to 2.4 MHz when time is allowed for external clock synchronization.

19.9.1 The Bit Processor

Figure 13-15 is a block diagram of the bit processor (BIP). In the asynchronous (self clocked) modes the clock is extracted from the data stream using the on-chip digital phase-locked-loop (DPLL). The DPLL requires a clock input at 16 times the data rate. This 16× clock may originate from SCLK, Timer 1 Overflow, or PH2 (one half the oscillator frequency). The extra divide-by-two described above allows these sources to be treated alternatively as 32× clocks.

The DPLL is a free-running four-bit counter running off the 16× clock. When a transition is detected in the receive data stream, a count is dropped (by suppressing the carry-in) if the current count value is greater than 8. A count is added (by injecting a carry into the second stage rather than the first) if the count is less than 8. No adjustment is made if the transition occurs at the count of 8. In this manner the counter locks in on the point at which transitions in the data stream occur at the count of 8, and a clock pulse is generated when the count overflows to 0.

In order to perform NRZI decoding, the NRZI decoder compares each bit of input data to the previous bit. There are no clock delays in going through the NRZI decoder.

The zero insert/delete circuitry (ZID) performs zero insertion/deletion, and also detects flags, GA's (Go-Ahead's), and aborts (same as GA's) in the data stream. The pattern 1111110 is detected as an early GA, so that the GA may be turned into a flag for loop mode transmission.

The shut-off detector monitors the receive data stream for a sequence of eight zeros, which is a shut-off command for loop mode transmissions. The shut-off detector is a three-bit counter which is cleared whenever a one is found in the receive data stream. Note that the ZID logic could not be used for this purpose, because the receive data must be monitored even when the ZID is being used for transmission.

As an example of the operation of the bit processor, the following sequence occurs in relation to the receive data:

1) RXD is sampled by SCLK, and then synchronized to the internal processor clock (IPC).

2) If the NRZI mode is selected, the incoming data is NRZI decoded.

3) When receiving other than the flag pattern, the ZID deletes the '0' after 5 consecutive '1's (during transmission this zero is inserted). The ZID locates the byte boundary for the rest of the circuitry. The ZID deletes the '0's by preventing the SR (shift register) from receiving a clocking pulse.

4) The FCS (which is a function of the data between the flags—not including the flags) is initialized and started at the detection of the byte boundary at the end of the opening flag. The FCS is computed each bit boundary until the closing flag is detected. Note that the received FCS has gone through the ZID during transmission.

19.9.2 The Byte Processor

Figure 19-16 is a block diagram of the byte processor (BYP). The BYP contains the registers and controllers necessary to perform the data manipulations associated with SDLC communications. The BYP registers may be read or written by the CPU over the 8044's internal bus

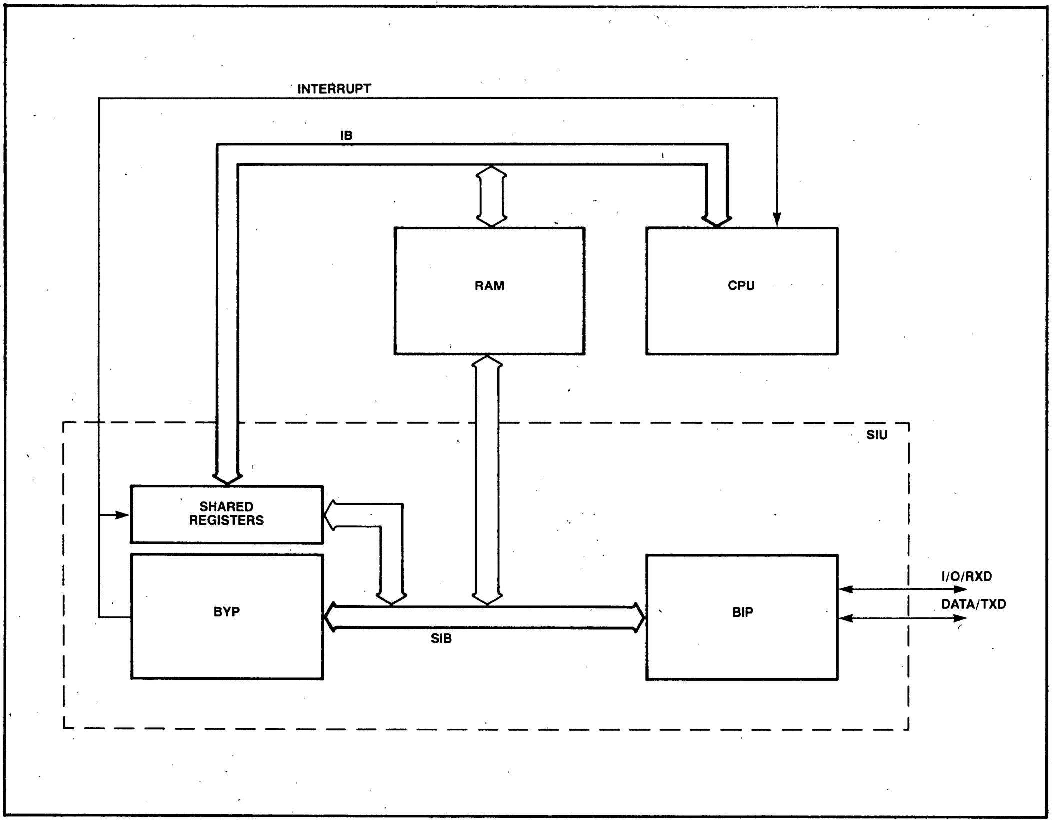


Figure 19-14. The Bit and Byte Processors

(IB), using standard 8044 hardware register operations. The 8044 register select PLA controls these operations. Three of the BYP registers connect to the IB through the IBS, a sub-bus which also connects to the CPU interrupt control registers.

Simultaneous access of a register by both the IB and the SIB is prevented by timing. In particular, RAM access is restricted to alternate internal processor cycles for the CPU and the SIU, in such a way that collisions do not occur.

As an example of the operation of the byte processor, the following sequence occurs in relation to the receive data:

1) Assuming that there is an address field in the frame, the BYP takes the station address from the register file into temporary storage. After the opening flag, the next field (the address field) is compared to the station address in the temporary storage. If a match occurs, the operation continues.

2) Assuming that there is a control field in the frame, the BYP takes the next byte and loads it into the RCB register. The RCB register has the logic to update the NSNR register (increment receive count, set SES and SER flags, etc.).

3) Assuming that there is an information field, the next byte is dumped into RAM at the RBS location. The DMA CNT (RBL at the opening flag) is loaded from the DMA CNT register into the RB register and decremented. The RFL is then loaded into the RB register, incremented, and stored back into the register file.

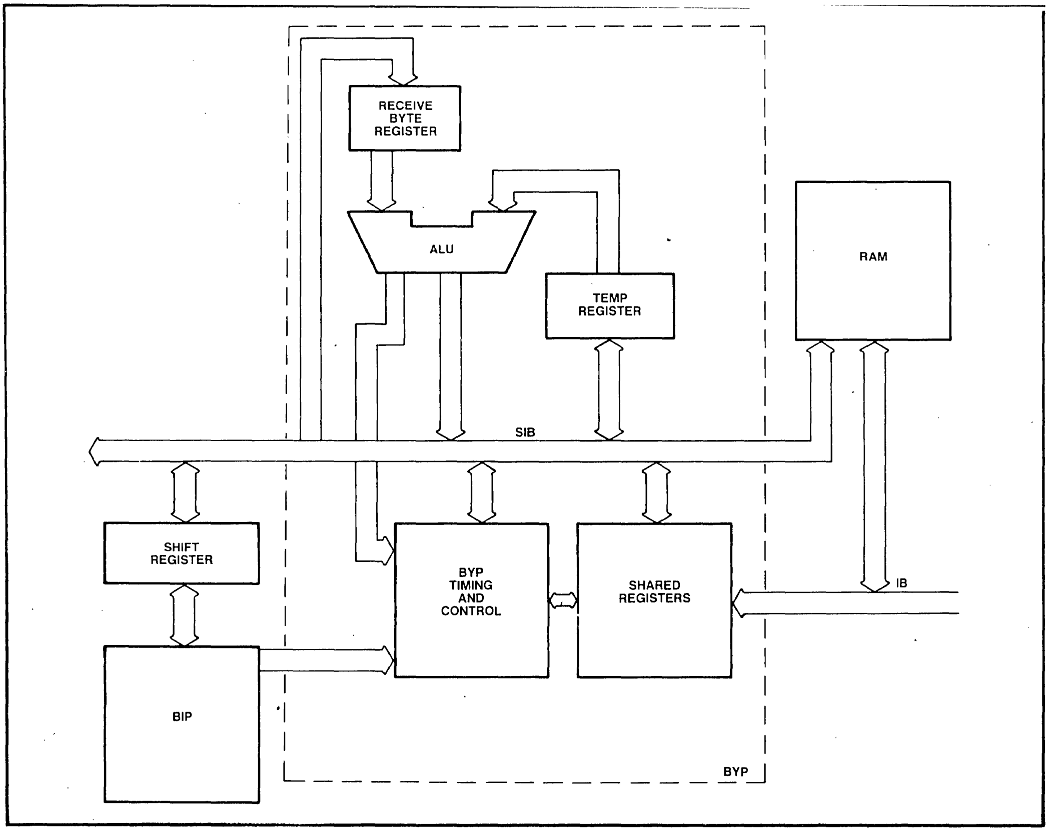


Figure 19-15. The Byte Processor

4) This process continues until the DMA CNT reaches zero, or until a closing flag is received. Upon either event, the BYP updates the status, and, if the CRC is good, the NSNR register.

19.10 DIAGNOSTICS

An SIU test mode has been provided, so that the on-chip CPU can perform limited diagnostics on the SIU. The test mode utilizes the output latches for P3.0 and P3.1 (pins 10 and 11). These port 3 pins are not useful as out-put ports, since the pins are taken up by the serial port functions. Figure 19-16 shows the signal routing associated with the SIU test mode.

Writing a 0 to P3.1 enables the serial test mode (P3.1 is set to 1 by reset). In test mode the P3.0 bit is mapped into the received data stream, and the 'write port 3' control signal is mapped into the SCLK path in place of T1. Thus, in test mode, the CPU can send a serial data stream to the SIU by writing to P3.0. The transmit data stream can be monitored by reading P3.1. Each sucessive bit is transmitted from the SIU by writing to any bit in Port 3, which generates SCLK.

In test mode, the P3.0 and P3.1 pins are placed in a high voltage, high impedance state. When the CPU reads P3.0 and P3.1 the logic level applied to the pin will be returned. In the test mode, when the CPU reads 3.1, the transmit data value will be returned, not the voltage on the pin. The transmit data remains constant for a bit time. Writing to P3.0 will result in the signal being outputted for a short period of time. However, since the signal is not latched, P3.0 will quickly return to a high voltage, high impedance state.

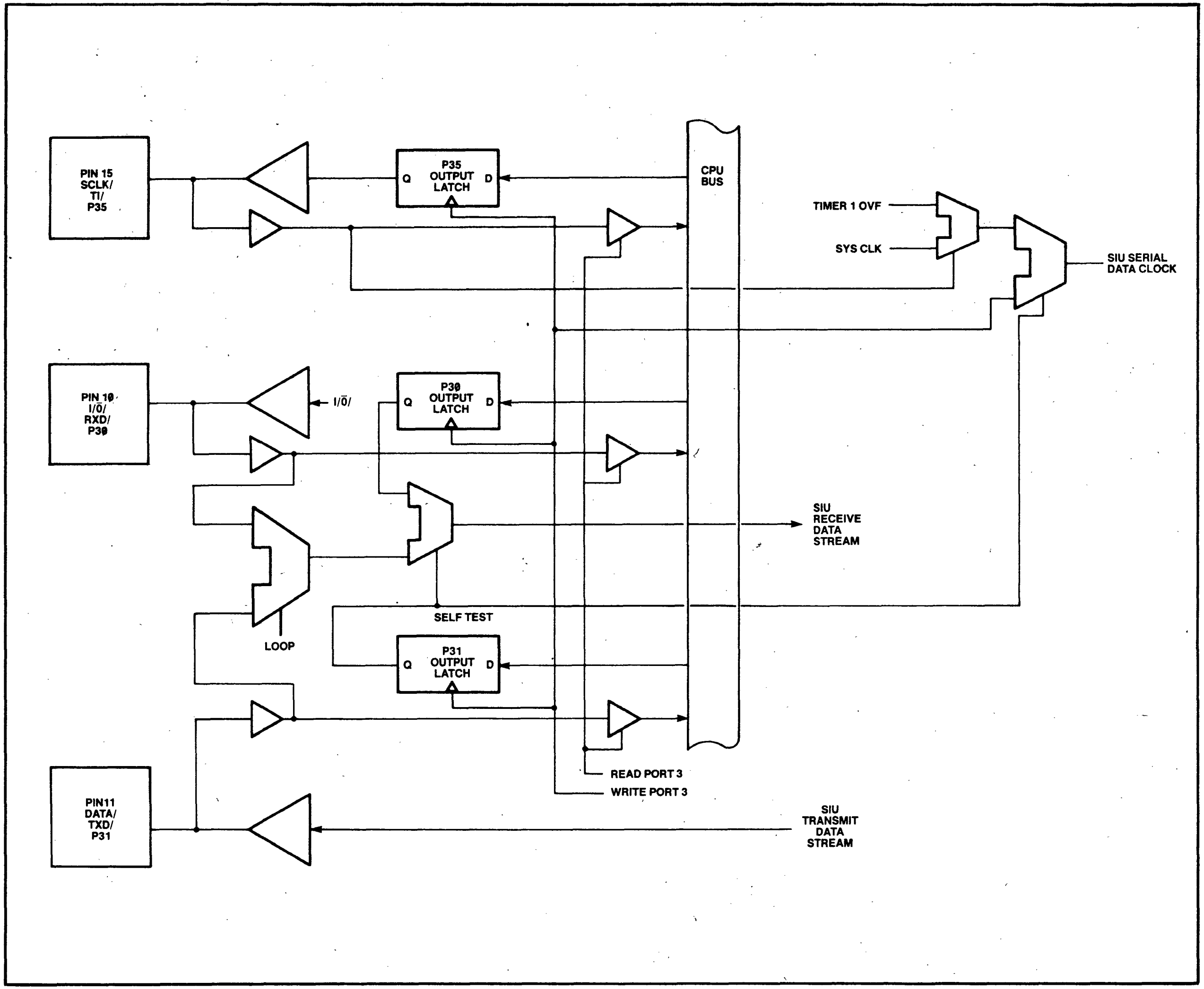


Figure 19-16. SIU Test Mode

The serial test mode is disabled by writing a 1 to P3.1. Care must be taken that a 0 is never written to P3.1 in the course of normal operation, since this causes the test mode to be entered.

Figure 19-17 is an example of a simple program segment that can be imbedded into the user's diagnostic program. That example shows how to put the 8044 into "Loop-back mode" to test the basic transmitting and receiving functions of the SIU.

Loop-back mode is functionally equivalent to a hardwire connection between pins 10 and 11 on the 8044.

In this example, the 8044 CPU plays the role of the primary station. The SIU is in the AUTO mode. The CPU sends the SIU a supervisory frame with the poll bit set and an RNR command. The SIU responds with a supervisory frame with the poll bit set and an RR command.

The operation proceeds as follows:

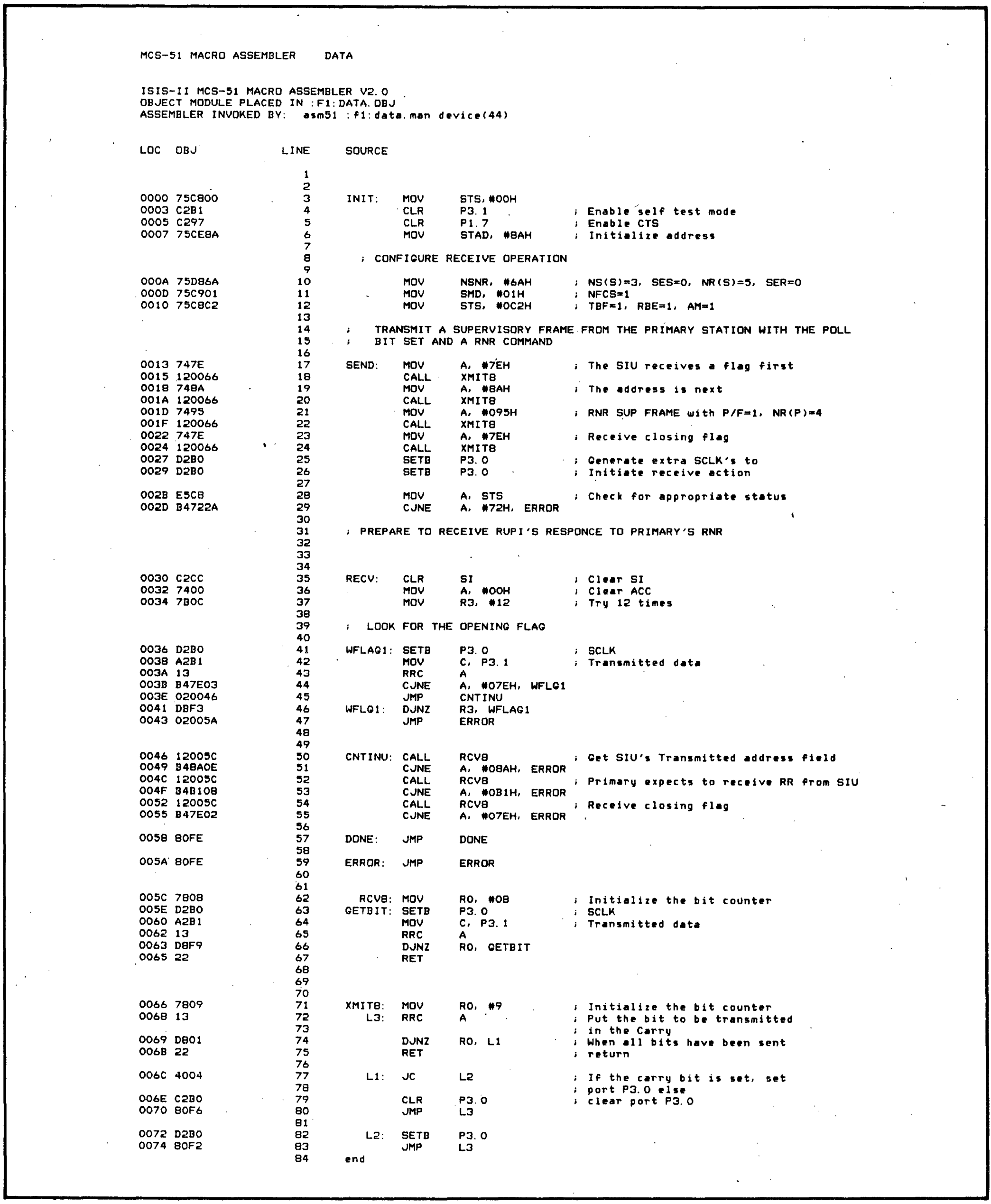
Interrupts are disabled, and the self test mode is enabled by writing a zero to P3.1. This establishes P3.0 as the data path from the CPU to the SIU. CTS (clear-to-send) is enabled by writing a zero to P1.7. The station address is initialized by writing 08AH into the STAD (station address register).

The SIU is configured for receive operation in the clocked mode and in AUTO mode. The CPU then transmits a supervisory frame. This frame consists of an opening flag, followed by the station address, a control field indicating that this is a supervisory frame with an RNR command, and then a closing flag.

Each byte of the frame is transmitted by writing that byte into the A register and then calling the subroutine XMIT8. Two additional SCLKs are generated to guarantee that the last bits in the frame have been clocked into the SIU. Finally the CPU reads the status register (STS). If the operation has proceeded correctly, the status will be 072H. If it is not, the program jumps to the ERROR loop and terminates.

The SIU generates an SI (SIU interrupt) to indicate that it has received a frame. The CPU clears this interrupt, and then begins to monitor the data stream that is being generated by the SIU in response to what it has received. As each bit arrives (via P3.1), it is moved into the accumulator, and the CPU compares the byte in the accumulator with 07EH, which is the opening flag. When a match occurs, the CPU identifies this as byte boundary, and thereafter processes the information byte-by-byte.

The CPU calls the RCV8 subroutine to get each byte into the accumulator. The CPU performs compare operations on (successively) the station address, the control field (which contains the RR response), and the closing flag. If any of these do not compare, the program jumps to the ERROR loop. If no error is found, the program jumps to the DONE loop.



```
MCS-51 MACRO ASSEMBLER    DATA


ISIS-II MCS-51 MACRO ASSEMBLER V2.0
OBJECT MODULE PLACED IN :F1:DATA.OBJ
ASSEMBLER INVOKED BY:  asm51 :f1:data.man device(44)


LOC  OBJ             LINE     SOURCE

                        1
                        2
0000 75C800             3     INIT:    MOV     STS,#00H
0003 C2B1               4              CLR     P3.1                ; Enable self test mode
0005 C297               5              CLR     P1.7                ; Enable CTS
0007 75CE8A             6              MOV     STAD, #8AH          ; Initialize address
                        7
                        8       ; CONFIGURE RECEIVE OPERATION
                        9
000A 75D86A            10              MOV     NSNR, #6AH          ; NS(S)=3, SES=0, NR(S)=5, SER=0
000D 75C901            11              MOV     SMD, #01H           ; NFCS=1
0010 75C8C2            12              MOV     STS, #0C2H          ; TBF=1, RBE=1, AM=1
                       13
                       14     ;    TRANSMIT A SUPERVISORY FRAME FROM THE PRIMARY STATION WITH THE POLL
                       15     ;    BIT SET AND A RNR COMMAND
                       16
0013 747E              17     SEND:    MOV     A, #7EH             ; The SIU receives a flag first
0015 120066            18              CALL    XMIT8
0018 748A              19              MOV     A, #8AH             ; The address is next
001A 120066            20              CALL    XMIT8
001D 7495              21              MOV     A, #095H            ; RNR SUP FRAME with P/F=1, NR(P)=4
001F 120066            22              CALL    XMIT8
0022 747E              23              MOV     A, #7EH             ; Receive closing flag
0024 120066            24              CALL    XMIT8
0027 D2B0              25              SETB    P3.0                ; Generate extra SCLK's to
0029 D2B0              26              SETB    P3.0                ; Initiate receive action
                       27
002B E5C8              28              MOV     A, STS              ; Check for appropriate status
002D B4722A            29              CJNE    A, #72H, ERROR
                       30
                       31     ; PREPARE TO RECEIVE RUPI'S RESPONCE TO PRIMARY'S RNR
                       32
                       33
                       34
0030 C2CC              35     RECV:    CLR     SI                  ; Clear SI
0032 7400              36              MOV     A, #00H             ; Clear ACC
0034 7B0C              37              MOV     R3, #12             ; Try 12 times
                       38
                       39     ;  LOOK FOR THE OPENING FLAG
                       40
0036 D2B0              41     WFLAG1:  SETB    P3.0                ; SCLK
0038 A2B1              42              MOV     C, P3.1             ; Transmitted data
003A 13                43              RRC     A
003B B47E03            44              CJNE    A, #07EH, WFLG1
003E 020046            45              JMP     CNTINU
0041 DBF3              46     WFLG1:   DJNZ    R3, WFLAG1
0043 02005A            47              JMP     ERROR
                       48
                       49
0046 12005C            50     CNTINU:  CALL    RCV8                ; Get SIU's Transmitted address field
0049 B48A0E            51              CJNE    A, #08AH, ERROR
004C 12005C            52              CALL    RCV8                ; Primary expects to receive RR from SIU
004F B4B108            53              CJNE    A, #0B1H, ERROR
0052 12005C            54              CALL    RCV8                ; Receive closing flag
0055 B47E02            55              CJNE    A, #07EH, ERROR
                       56
0058 80FE              57     DONE:    JMP     DONE
                       58
005A 80FE              59     ERROR:   JMP     ERROR
                       60
                       61
005C 7808              62       RCV8:  MOV     R0, #08             ; Initialize the bit counter
005E D2B0              63     GETBIT:  SETB    P3.0                ; SCLK
0060 A2B1              64              MOV     C, P3.1             ; Transmitted data
0062 13                65              RRC     A
0063 D8F9              66              DJNZ    R0, GETBIT
0065 22                67              RET
                       68
                       69
                       70
0066 7809              71     XMIT8:   MOV     R0, #9              ; Initialize the bit counter
0068 13                72         L3:  RRC     A                   ; Put the bit to be transmitted
                       73                                          ; in the Carry
0069 D801              74              DJNZ    R0, L1              ; When all bits have been sent
006B 22                75              RET                         ; return
                       76
006C 4004              77         L1:  JC      L2                  ; If the carry bit is set, set
                       78                                          ; port P3.0 else
006E C2B0              79              CLR     P3.0                ; clear port P3.0
0070 80F6              80              JMP     L3
                       81
0072 D2B0              82         L2:  SETB    P3.0
0074 80F2              83              JMP     L3
                       84     end
```

Figure 19-17. Loop-Back Mode Software

8044 Application Examples 20

CHAPTER 20
8044 APPLICATION EXAMPLES

20.0 8044 APPLICATION EXAMPLES

20.1 INTERFACING THE 8044 TO A MICROPROCESSOR

The 8044 is designed to serve as an intelligent controller for remote peripherals. However, it can also be used as an intelligent HDLC/SDLC front end for a microprocessor, capapble of extensively off-loading link control functions for the CPU. In some applications, the 8044 can even be used for communications preprocessing, in addition to data link control.

This section describes a sample hardware interface for attaching the 8044 to an 8088. It is general enough to be extended to other microprocessors such as the 8086 or the 80186.

OVERVIEW

A sample interface is shown in Figure 20-1. Transmission occurs when the 8088 loads a 64 byte block of memory with some known data. The 8088 then enables the 8237A to DMA this data to the 8044. When the 8044 has received all of the data from the 8237A, it sends the data in a SDLC frame. The frame is captured by the Spectron Datascope®* which displays it on a CRT in hex format.

In reception, the Datascope sends an SDLC information frame to the 8044. The 8044 receives the SDLC frame, buffers it, and sends it to the 8088's memory. In this example the 8044 is being operated in the NON-AUTO mode; therefore, it does not need to be polled by a primary station in order to transmit.

THE INTERFACE

The 8044 does not have a parallel slave port. The 8044's 32 I/O lines can be configured as a local microprocessor bus master. In this configuration, the 8044 can expand the ROM and RAM memory, control peripherals, and communicate with a microprocessor.

The 8044, like the 8051, does not have a Ready line, so there is no way to put the 8044 in wait state. The clock on the 8044 cannot be stopped. Dual port RAM could still be used, however, software arbitration would be the only way to prevent collisions. Another way to interface the 8044 with another CPU is to put a FIFO or queue between the two processors, and this was the method chosen for this design.

Figure 20-2 shows the schematic of the 8044/8088 interface. It involves two 8 bit tri-state latches, two SR flip-flops, and some logic gates (6 TTL packs). The circuitry implements a one byte FIFO. RS422 transceivers are used, which can be connected to a multidrop link. Figure 20-3 shows the 8088 and support circuitry; the memory and decoders are not shown. It is a basic 8088 Min Mode system with an 8237A DMA controller and an 8259A interrupt controller.

DMA Channel One transfers a block of memory to the tri-state latch, while Channel Zero transfers a block of data from the latch to 8088's memory. The 8044's Interrupt 0 signal vectors the CPU into a routine which reads from the internal RAM and writes to the latch. The 8044's Interrupt 1 signal causes the chip to read from the latch and write to its on-chip data RAM. Both DMA requests and acknowledges are active low.

Initially, when the power is applied, a reset pulse coming from the 8284A initializes the SR flip-flops. In this initialization state, the 8044's transmit interrupt and the 8088's transmit DMA request are active; however, the software keeps these signals disabled until either of the two processors are ready to transmit. The software leaves the receive signals enabled, unless the receive buffers are full. In this way either the 8088 or the 8044 are always ready to receive, but they must enable the transmit signal when they have prepared a block to transmit. After a block has been transmitted or received, the DMA and interrupt signals return to the initial state.

The receive and transmit buffer sizes for the blocks of data sent between the 8044 and the 8088 have a maximum fixed length. In this case the buffer size was 64 bytes. The buffer size must be less than 192 bytes to enable 8044 to buffer the data in its on-chip RAM. This design allows blocks of data that are less than 64 bytes, and accommodates networks that allow frames of varying size. The first byte transferred between the 8088 and the 8044 is the byte count to follow; thus the 8044 knows how many bytes to receive before it transmits the SDLC frame. However, when the 8044 sends data to the 8088's memory, the 8237A will not know if the 8044 will send less than the count the 8237A was programmed for. To solve this problem, the 8237A is operated in the single mode. The 8044 uses an I/O bit to generate an interrupt request to the 8259A. In the 8088's interrupt routine, the 8237A's receive DMA channel is disabled, thus allowing blocks of data less than 64 bytes to be received.

THE SOFTWARE

The software for the 8044 and the 8088 is shown in Table 20-1. The 8088 software was written in PL/M86, and the 8044 software was written in assembly language.

The 8044 software begins by initializing the stack, interrupt priorities, and triggering types for the interrupts. At this point, the SIU parameter registers are initialized. The receive and transmit buffer starting addresses and lengths are loaded for the on-chip DMA. This DMA is for the serial port. The serial station address and the transmit control bytes are loaded too.

*Datascope is a trademark of Spectron Inc.

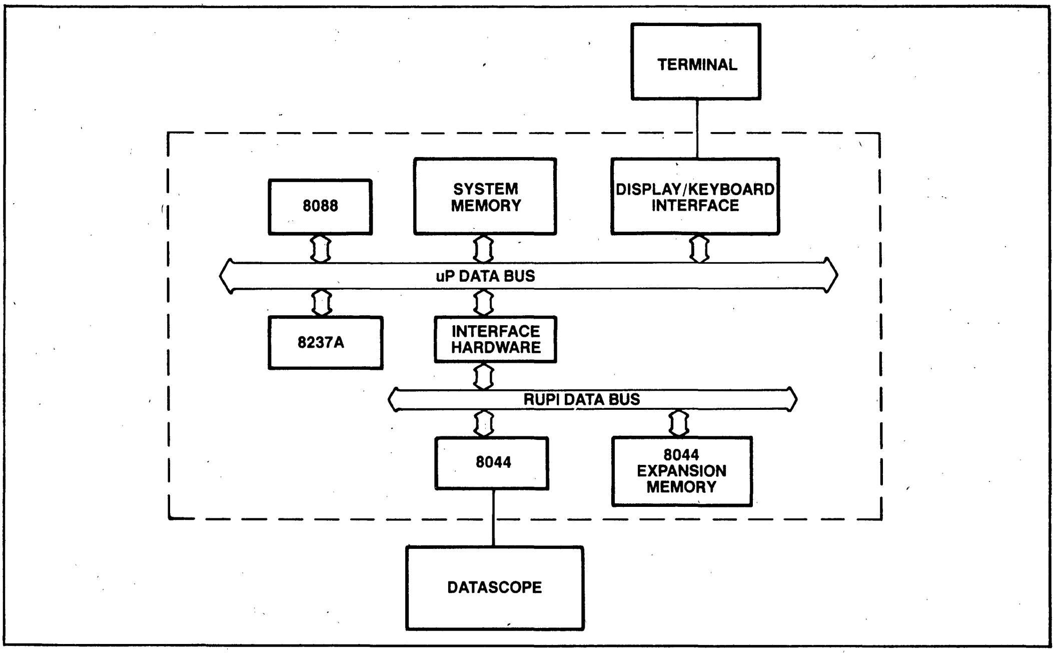


Figure 20-1. Block Diagram of 8088/8044 Interface Test

Once the initialization has taken place, the SIU interrupt is enabled, and the external interrupt which receives bytes from the 8088 is enabled. Setting the 8044's Receive Buffer Empty (RBE) bit enables the receiver. If this bit is reset, no serial data can be received. The 8044 then waits in a loop for either RECEIVE DMA interrupt or the SERIAL INT interrupt.

The RECEIVE DMA interrupt occurs when the 8237A is transferring a block of data to the 8044. The first time this interrupt occurs, the 8044 reads the latch and loads the count value into the R2 register. On subsequent interrupts, the 8044 reads the latch, loads the data into the transmit buffer, and decrements R2. When R2 reaches zero, the interrupt routine sends the data in an SDLC frame, and disables the RECEIVE DMA interrupt. After the frame has been transmitted, a serial interrupt is generated. The SERIAL INT routine detects that a frame has been transmitted and re-enables the RECEIVE DMA interrupt. Thus, while the frame is being transmitted through the SIU, the 8237A is inhibited from sending data to the 8044's transmit buffer.

The TRANSMIT DMA routine sends a block of data from the 8044's receive buffer to the 8088's memory. Normally this interrupt remains disabled. However, if a serial interrupt occurs, and the SERIAL INT routine detects that a frame has been received, it calls the SEND subroutine. The SEND subroutine loads the number of bytes which were received in the frame into the receive buffer. Register R1 points to the receive buffer and R2 is loaded with the count. The TRANSMIT DMA interrupt is enabled, and immediately upon returning from the SERIAL INT routine, the interrupt is acknowledged. Each time the TRANSMIT DMA interrupt occurs, a byte is read from the receive buffer, written to the latch, and R2 is decremented. When R2 reaches 0, the TRANSMIT DMA interrupt is disabled, the SIU receiver is re-enabled, and the 8044 interrupts the 8088.

The 8088 software simply transmits a block of data and receives a block of data, then stops. The software begins by initializing the 8237A, and the 8259A. It then loads a block of memory with some data and enables the 8237A to transmit the data. In the meantime the 8088 waits in a loop. After a block of data is received from the 8044, the 8088 is interrupted, and it shuts off the 8237A receive DMA.

CONCLUSION

For the software shown in Table 20-1, the transfer rate from the 8088's memory to the 8044 was measured at 75K bytes/sec. This transfer rate largely depends upon the number of instructions in the 8044's interrupt service routine. Fewer instructions result in a higher transfer rate.

There are many ways of interfacing the 8044 locally to another microprocessor: FIFO's, dual port RAM with software arbitration, and 8255's are just a few. Alternative approaches, which may be more optimal for certain applications, are certainly possible.

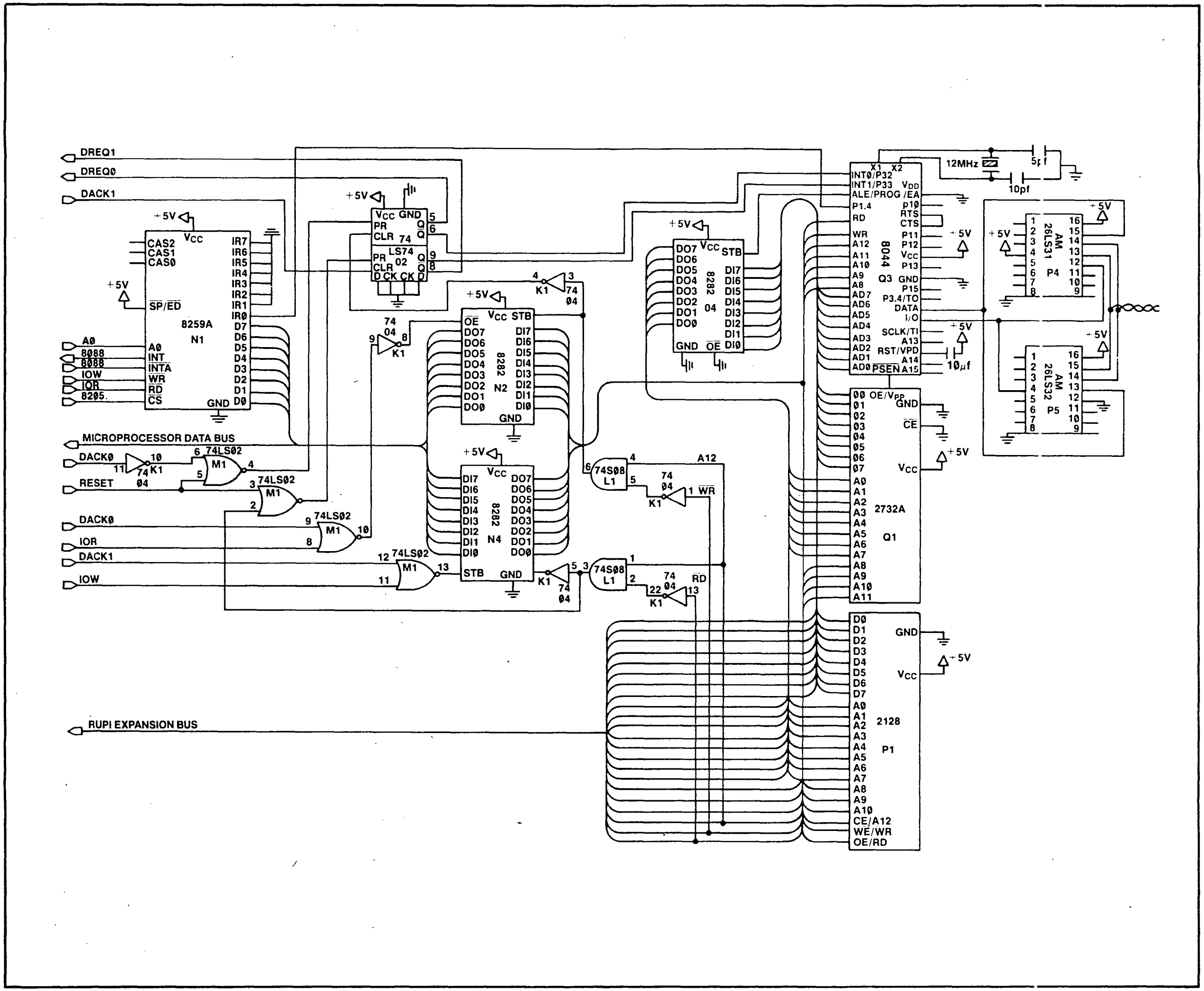


Figure 20-2. 8044 interface to the 8088

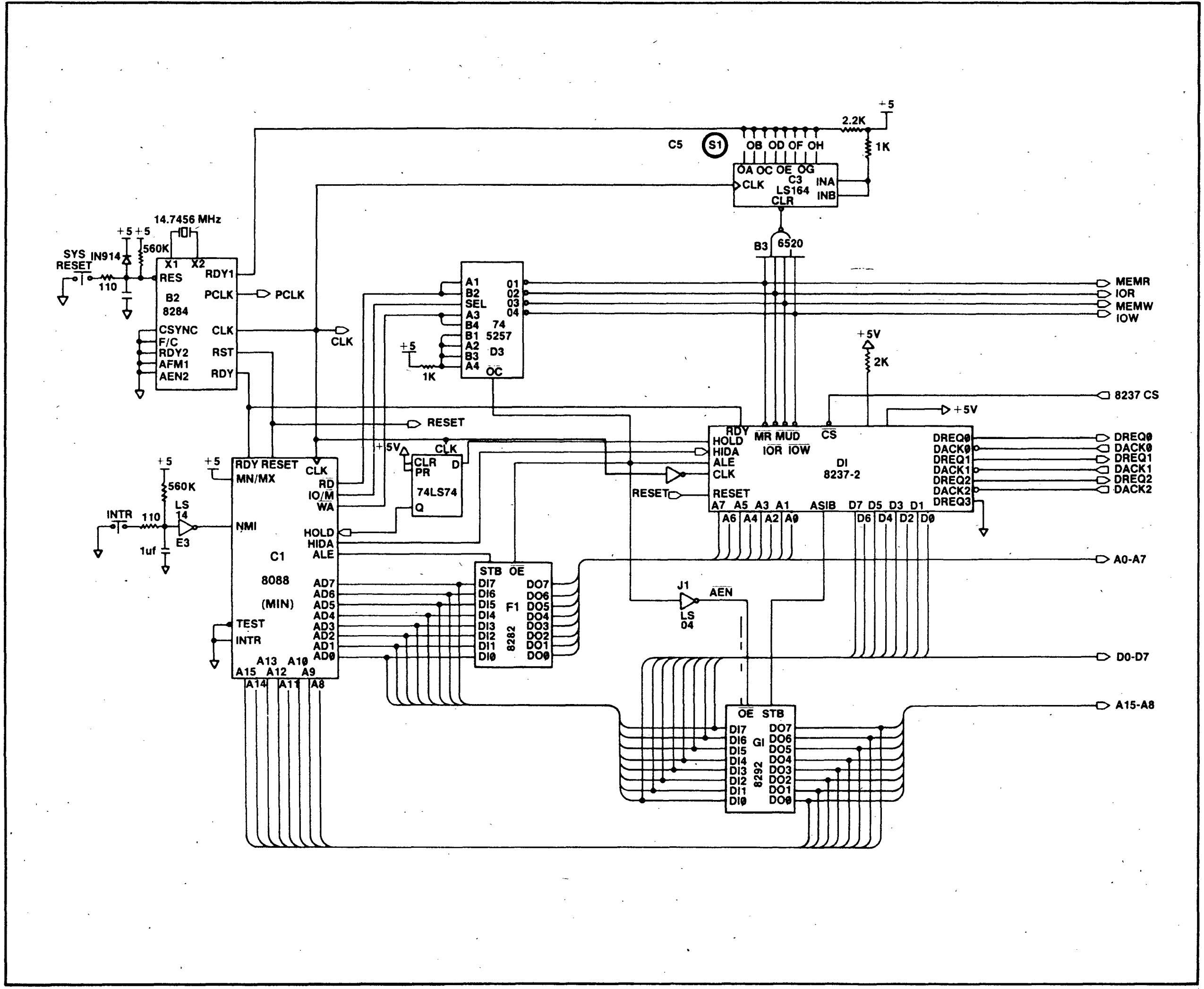


Figure 20-3. 8088 Min Mode System

Table 20-1. Transmit and Receive Software for an 8044/8088 System

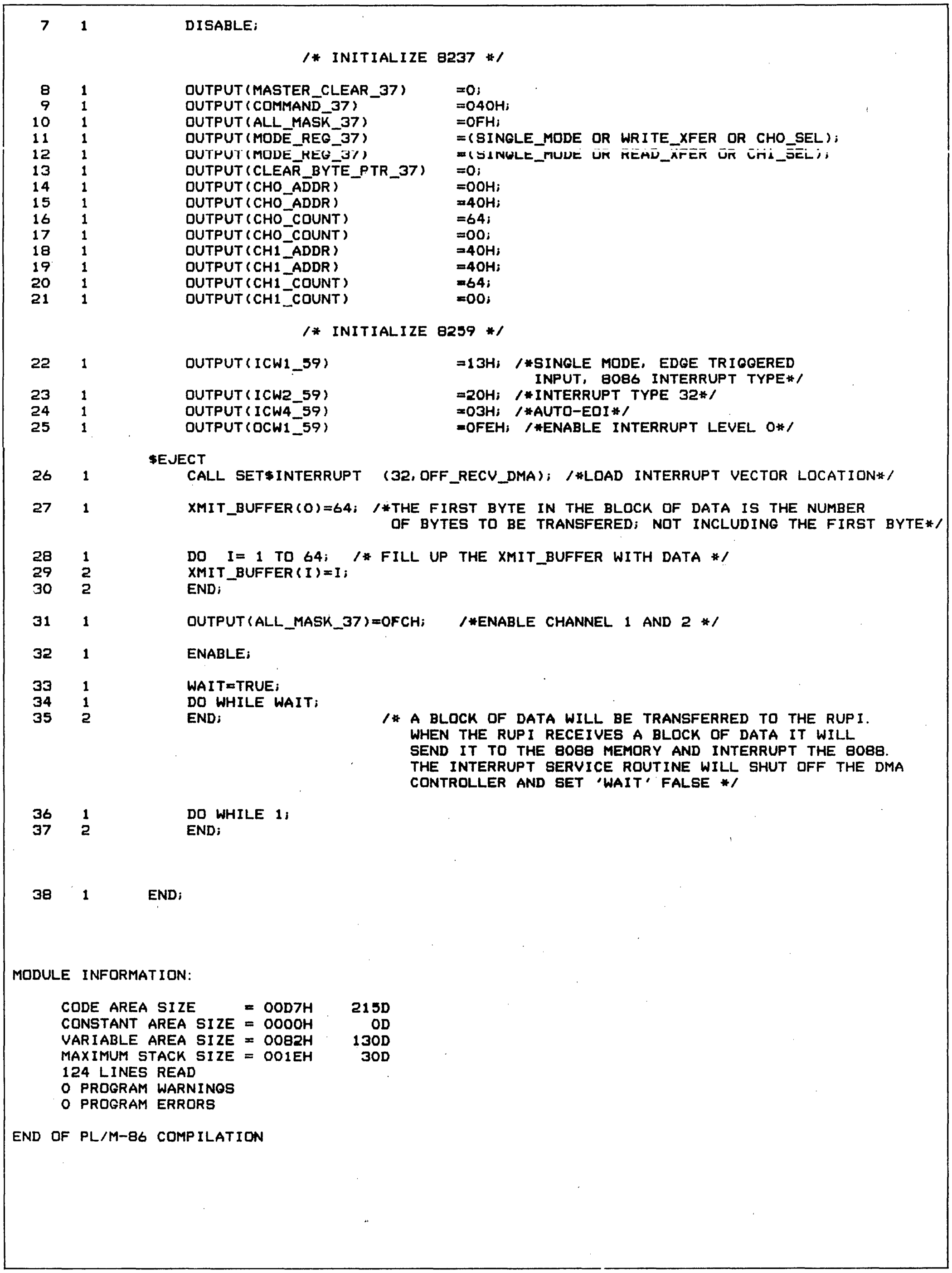
```
LOC   OBJ       LINE   SOURCE

                       $debug      title   (8044/8088 INTERFACE)


 0000                  FIRST_BYTE  BIT     0               ; FLAG

0000                               ORG     0
0000 8024                          SJMP    INIT

0026                               ORG     26H

0026  7581AA           INIT:       MOV     SP, #170        ; INITIALIZE STACK
0029  75B800                       MOV     IP, #00         ; ALL INTERRUPTS ARE EQUAL PRIORITY
002C  75C954                       MOV     SMD, #54H       ; TIMER 1 OVERFLOW, NRZI, PRE-FRAME SYNC
002F  758844                       MOV     TCON, #44H      ; EDGE TRIGGERED EXTERNAL INTERRUPT 1
                                                           ; LEVEL TRIGGERED EXTERNAL INTERRUPT 0
                                                           ; TIMER 1 ON
0032  758DEC                       MOV     TH1, #0ECH      ; INITIALIZE TIMER, 3125 BPS
0035  758920                       MOV     TMOD, #20H      ; TIMER 1 AUTO RELOAD

0038  75DC6A                       MOV     TBS, #106       ; SET UP SIU PARAMETER REGISTERS
003B  75DB40                       MOV     TBL, #64
003E  75CC2A                       MOV     RBS, #42
0041  75CB40                       MOV     RBL, #64
0044  75CE55                       MOV     STAD, #55H
0047  75DA11                       MOV     TCB, #00010001B ; RR, P/F=1

004A  901000                       MOV     DPTR, #1000H    ; DPTR POINTS TO TRI-STATE LATCH
004D  D200                         SETB    FIRST_BYTE      ; FLAG TO INDICATE FIRST BYTE
                                                           ; FOR RECEIVE INTERRUPT ROUTINE
004F  D2CE                         SETB    RBE             ; READY TO RECEIVE
0051  75A894                       MOV     IE, #10010100B  ; ENABLE RECEIVE DMA AND SIU INTERRUPT

0054  80FE                         SJMP    $               ; WAIT HERE FOR INTERRUPTS

0056  80FE             ERROR:      SJMP    ERROR
                   +1  $EJ
                       ;*************************   SUBROUTINES   ********************************

0058  85CD29           SEND:       MOV     41, RFL         ; FIRST BYTE IN BLOCK IS COUNT
005B  7929                         MOV     R1, #41         ; POINT TO BLOCK OF DATA
005D  AACD                         MOV     R2, RFL         ; LOAD COUNT
005F  0A                           INC     R2
0060  D2A8                         SETB    EX0             ; ENABLE DMA TRANSMIT INTERRUPT
0062  22                           RET



                       ;******************   INTERRUPT SERVICE ROUTINES   ********************

0063                   LOC_TMP SET         $              ; SET UP INTERRUPT TABLE JUMP
0013                               ORG     0013H
0013  020063                       LJMP    RECEIVE_DMA
0063                               ORG     LOC_TMP

                       RECEIVE_DMA:
```

```

0063 10000E             JBC     FIRST_BYTE, L1 ; THE FIRST BYTE TRANSFERRED IS THE COUNT

0066 E0                 MOVX    A, @DPTR       ; READ THE LATCH
0067 F6                 MOV     @R0, A         ; PUT IT IN TRANSMIT BUFFER
0068 08                 INC     R0
0069 DA08               DJNZ    R2, L2         ; AFTER READING BYTES,

006B D2CF               SETB    TBF            ; SEND DATA
006D D2CD               SETB    RTS
006F D200               SETB    FIRST_BYTE
0071 C2AA               CLR     EX1

0073 32         L2:     RETI

0074 786A       L1:     MOV     R0, #106       ; R0 IS A POINTER TO THE TRANSMIT
                                               ; BUFFER STARTING ADDRESS
0076 E0                 MOVX    A, @DPTR       ; PUT THE FIRST BYTE INTO
0077 FA                 MOV     R2, A          ; R2 FOR THE COUNT
0078 32                 RETI

  0079          LOC_TMP SET     $
0003                    ORG     0003H
0003 020079             LJMP    TRANSMIT_DMA
0079                    ORG     LOC_TMP


                TRANSMIT_DMA

0079 E7                 MOV     A, @R1         ; READ BYTE OUT OF THE RECEIVE BUFFER
007A F0                 MOVX    @DPTR, A       ; WRITE IT TO THE LATCH
007B 09                 INC     R1
007C DA08               DJNZ    R2, L3         ; WHEN ALL BYTES HAVE BEEN SENT

007E C2A8               CLR     IE. 0          ; DISABLE INTERRUPT
0080 C294               CLR     P1. 4          ; CAUSE 8088 INTERRUPT TO TERMINATE DMA
0082 D294               SETB    P1. 4
0084 D2CE               SETB    RBE            ; ENABLE RECEIVER AGAIN

0086 32         L3:     RETI



  0087          LOC_TMP SET     $
0023                    ORG     0023H
0023 020087             LJMP    SERIAL_INT
0087                    ORG     LOC_TMP

                SERIAL_INT:

0087 30CE06             JNB     RBE, RCV       ; WAS A FRAME RECEIVED
008A 30CF0B             JNB     TBF, XMIT      ; WAS A FRAME TRANSMITTED
008D 020056             LJMP    ERROR          ; IF NEITHER ERROR

0090 20CBC3     RCV:    JB      BOV, ERROR     ; IF BUFFER OVERRUN THEN ERROR
0093 1158               CALL    SEND           ; SEND THE FRAME TO THE 8088
0095 C2CC               CLR     SI
0097 32                 RETI

0098 C2CC       XMIT:   CLR     SI
```

```
009A D2AA      115            SETB    EX1
009C 32        116            RETI
               117
               118            END

SYMBOL TABLE LISTING

NAME                TYPE        VALUE            ATTRIBUTES

BOV . . . . . .     B ADDR      00C8H.3  A
ERROR . . . . .     C ADDR      0056H    A
EX0 . . . . . .     B ADDR      00A8H.0  A
EX1 . . . . . .     B ADDR      00A8H.2  A
FIRST_BYTE  . .     B ADDR      0020H.0  A
IE . . . . . . .    D ADDR      00A8H    A
INIT . . . . . .    C ADDR      0026H    A
IP . . . . . . .    D ADDR      00B8H    A
L1 . . . . . . .    C ADDR      0074H    A
L2 . . . . . . .    C ADDR      0073H    A
L3 . . . . . . .    C ADDR      0086H    A
LOC_TMP . . .       C ADDR      0087H    A
P1 . . . . . . .    D ADDR      0090H    A
RBE . . . . . .     B ADDR      00C8H.6  A
RBL . . . . . .     D ADDR      00CBH    A
RBS . . . . . .     D ADDR      00CCH    A
RCV . . . . . .     C ADDR      0090H    A
RECEIVE_DMA  .      C ADDR      0063H    A
RFL . . . . . .     D ADDR      00CDH    A
RTS . . . . . .     B ADDR      00C8H.5  A
SEND . . . . .      C ADDR      0058H    A
SERIAL_INT  . .     C ADDR      0087H    A
SI . . . . . . .    B ADDR      00C8H.4  A
SMD . . . . . .     D ADDR      00C9H    A
SP . . . . . . .    D ADDR      0081H    A
STAD . . . . .      D ADDR      00CEH    A
TBF . . . . . .     B ADDR      00C8H.7  A
TBL . . . . . .     D ADDR      00DBH    A
TBS . . . . . .     D ADDR      00DCH    A
TCB . . . . . .     D ADDR      00DAH    A
TCON . . . . .      D ADDR      0088H    A
TH1 . . . . . .     D ADDR      008DH    A
TMOD . . . . .      D ADDR      0089H    A
TRANSMIT_DMA  .     C ADDR      0079H    A
XMIT . . . . .      C ADDR      0098H    A

  REGISTER BANK(S) USED: 0, TARGET MACHINE(S): 8044

  ASSEMBLY COMPLETE, NO ERRORS FOUND
```

Table 20-2. PL/M-86 Compiler Rupi/8088 Interface Example

```
SERIES-III PL/M-86 V1.0 COMPILATION OF MODULE RUPI_88
OBJECT MODULE PLACED IN :F1:R88.OBJ
COMPILER INVOKED BY:  PLM86.86 :F1:R88.SRC


              $DEBUG
              $TITLE  ('RUPI/8088 INTERFACE EXAMPLE')

   1          RUPI_88:DO;

   2    1         DECLARE

                  LIT                  LITERALLY   'LITERALLY',
                  TRUE                 LIT         '01H',
                  FALSE                LIT         '00H',

                  RECV_BUFFER(64)      BYTE,
                  XMIT_BUFFER(64)      BYTE,
                  I                    BYTE,
                  WAIT                 BYTE,

                                 /* 8237 PORTS*/

                  MASTER_CLEAR_37      LIT            'OFFDDH',
                  COMMAND_37           LIT            'OFFD8H',
                  ALL_MASK_37          LIT            'OFFDFH',
                  SINGLE_MASK_37       LIT            'OFFDAH',
                  STATUS_37            LIT            'OFFD8H',
                  REQUEST_REG_37       LIT            'OFFD9H',
                  MODE_REG_37          LIT            'OFFDBH',
                  CLEAR_BYTE_PTR_37    LIT            'OFFDCH',

                  CH0_ADDR             LIT            'OFFD0H',
                  CH0_COUNT            LIT            'OFFD1H',
                  CH1_ADDR             LIT            'OFFD2H',
                  CH1_COUNT            LIT            'OFFD3H',
                  CH2_ADDR             LIT            'OFFD4H',
                  CH2_COUNT            LIT            'OFFD5H',
                  CH3_ADDR             LIT            'OFFD6H',
                  CH3_COUNT            LIT            'OFFD7H',

                             /* 8237 BIT ASSIGNMENTS */

                  CH0_SEL              LIT            '00H',
                  CH1_SEL              LIT            '01H',
                  CH2_SEL              LIT            '02H',
                  CH3_SEL              LIT            '03H',
                  WRITE_XFER           LIT            '04H',
                  READ_XFER            LIT            '08H',
                  DEMAND_MODE          LIT            '00H',
                  SINGLE_MODE          LIT            '40H',
                  BLOCK_MODE           LIT            '80H',
                  SET_MASK             LIT            '04H',

             $EJECT
                                /* 8259 PORTS */

                STATUS_POLL_59       LIT            'OFFE0H',
                ICW1_59              LIT            'OFFE0H',
                OCW1_59              LIT            'OFFE1H',
                OCW2_59              LIT            'OFFE0H',
                OCW3_59              LIT            'OFFE0H',
                ICW2_59              LIT            'OFFE1H',
                ICW3_59              LIT            'OFFE1H',
                ICW4_59              LIT            'OFFE1H';

                         /* INTERRUPT SERVICE ROUTINE */

  3    1     OFF_RECV_DMA:    PROCEDURE    INTERRUPT 32;

  4    2        OUTPUT(SINGLE_MASK_37)=40H;
  5    2        WAIT=FALSE;
  6    2        END;
```



```
  7   1           DISABLE;

                              /* INITIALIZE 8237 */

  8   1           OUTPUT(MASTER_CLEAR_37)      =0;
  9   1           OUTPUT(COMMAND_37)           =040H;
 10   1           OUTPUT(ALL_MASK_37)          =0FH;
 11   1           OUTPUT(MODE_REG_37)          =(SINGLE_MODE OR WRITE_XFER OR CH0_SEL);
 12   1           OUTPUT(MODE_REG_37)          =(SINGLE_MODE OR READ_XFER OR CH1_SEL);
 13   1           OUTPUT(CLEAR_BYTE_PTR_37)    =0;
 14   1           OUTPUT(CH0_ADDR)             =00H;
 15   1           OUTPUT(CH0_ADDR)             =40H;
 16   1           OUTPUT(CH0_COUNT)            =64;
 17   1           OUTPUT(CH0_COUNT)            =00;
 18   1           OUTPUT(CH1_ADDR)             =40H;
 19   1           OUTPUT(CH1_ADDR)             =40H;
 20   1           OUTPUT(CH1_COUNT)            =64;
 21   1           OUTPUT(CH1_COUNT)            =00;

                              /* INITIALIZE 8259 */

 22   1           OUTPUT(ICW1_59)              =13H; /*SINGLE MODE, EDGE TRIGGERED
                                                      INPUT, 8086 INTERRUPT TYPE*/
 23   1           OUTPUT(ICW2_59)              =20H; /*INTERRUPT TYPE 32*/
 24   1           OUTPUT(ICW4_59)              =03H; /*AUTO-EOI*/
 25   1           OUTPUT(OCW1_59)              =0FEH; /*ENABLE INTERRUPT LEVEL 0*/

        $EJECT
 26   1           CALL SET$INTERRUPT  (32,OFF_RECV_DMA); /*LOAD INTERRUPT VECTOR LOCATION*/

 27   1           XMIT_BUFFER(0)=64; /*THE FIRST BYTE IN THE BLOCK OF DATA IS THE NUMBER
                                       OF BYTES TO BE TRANSFERED; NOT INCLUDING THE FIRST BYTE*/

 28   1           DO  I= 1 TO 64;  /* FILL UP THE XMIT_BUFFER WITH DATA */
 29   2           XMIT_BUFFER(I)=I;
 30   2           END;

 31   1           OUTPUT(ALL_MASK_37)=0FCH;   /*ENABLE CHANNEL 1 AND 2 */

 32   1           ENABLE;

 33   1           WAIT=TRUE;
 34   1           DO WHILE WAIT;
 35   2           END;                 /* A BLOCK OF DATA WILL BE TRANSFERRED TO THE RUPI.
                                         WHEN THE RUPI RECEIVES A BLOCK OF DATA IT WILL
                                         SEND IT TO THE 8088 MEMORY AND INTERRUPT THE 8088.
                                         THE INTERRUPT SERVICE ROUTINE WILL SHUT OFF THE DMA
                                         CONTROLLER AND SET 'WAIT' FALSE */

 36   1           DO WHILE 1;
 37   2           END;


 38   1       END;



MODULE INFORMATION:

     CODE AREA SIZE     = 00D7H    215D
     CONSTANT AREA SIZE = 0000H      0D
     VARIABLE AREA SIZE = 0082H    130D
     MAXIMUM STACK SIZE = 001EH     30D
     124 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

END OF PL/M-86 COMPILATION
```

APPENDIX B
LISTINGS OF SOFTWARE MODULES

```
PL/M-51 COMPILER    RUPI-44 Secondary Station Driver                         20:24:47  09/20/83  PAGE   1

ISIS-II PL/M-51 V1.0
COMPILER INVOKED BY:  :F2:PLM51 :F2:APNOTE.SRC


            $TITLE      ('RUPI-44 Secondary Station Driver')
            $DEBUG
            $REGISTERBANK(1)
   1    1   MAIN$MOD:DO;
            $NOLIST

            /* To save paper the RUPI registers are not listed, but this is the statement
               used to include them: $INCLUDE (:F2:REG44.DCL) */

   5    1   DECLARE LIT             LITERALLY   'LITERALLY',
                    TRUE            LIT         'OFFH',
                    FALSE           LIT         'OOH',
                    FOREVER         LIT         'WHILE 1';

            /* SDLC commands and responses */

   6    1   DECLARE SNRM            LIT         '83H',
                    UA              LIT         '73H',
                    DISC            LIT         '43H',
                    DM              LIT         '1FH',
                    FRMR            LIT         '97H',
                    REQ_DISC        LIT         '53H',
                    UP              LIT         '33H',
                    TEST            LIT         'OE3H',


                            /* User states */

                    OPEN_S            LIT         'OOH',
                    CLOSED_S          LIT         'O1H',

                            /* Station states */

                    DISCONNECT_S    LIT         'OOH', /* LOGICALLY DISCONNECTED STATE*/
                    FRMR_S          LIT         'O1H', /* FRAME REJECT STATE */
                    I_T_S           LIT         'O2H', /* INFORMATION TRANSFER STATE */

                    /* Status values returned from TRANSMIT procedure */

                    USER_STATE_CLOSED   LIT     'OOH',
                    LINK_DISCONNECTED   LIT     'O1H',
                    OVERFLOW            LIT     'O2H',
                    DATA_TRANSMITTED    LIT     'O3H',

                    /* Parameters passed to XMIT_FRMR */

                    UNASSIGNED_C        LIT     'OOH',
                    NO_I_FIELD_ALLOWED  LIT     'O1H',
                    BUFF_OVERRUN        LIT     'O2H',
                    SES_ERR             LIT     'O3H',
                               /* Variables */

                      USER_STATE             BYTE    AUXILIARY,
                      STATION_STATE          BYTE    AUXILIARY,
                      I_FRAME_LENGTH         BYTE    AUXILIARY,

                               /* Buffers */

                      BUFFER_LENGTH                      LIT      '60',
                      SIU_XMIT_BUFFER(BUFFER_LENGTH)     BYTE     PUBLIC     IDATA,
                      SIU_RECV_BUFFER(BUFFER_LENGTH)     BYTE     PUBLIC,
                      FRMR_BUFFER(3)          BYTE,

                               /* Flags */

                      XMIT_BUFFER_EMPTY          BIT PUBLIC;

   7    2     SIU_RECV: PROCEDURE (LENGTH) EXTERNAL;
   8    2          DECLARE LENGTH  BYTE;
   9    1     END SIU_RECV;

  10    2     OPEN: PROCEDURE PUBLIC USING 2;
  11    2             USER_STATE=OPEN_S;
  12    1     END OPEN;

  13    2     CLOSE: PROCEDURE PUBLIC USING 2;
  14    2             AM=0;
  15    2             USER_STATE=CLOSED_S;
  16    1     END CLOSE;

  17    2     POWER_ON_D: PROCEDURE   PUBLIC USING 0;

  18    2             USER_STATE=CLOSED_S;
  19    2             STATION_STATE=DISCONNECT_S;
  20    2             TBS=.SIU_XMIT_BUFFER(0);
  21    2             RBS=.SIU_RECV_BUFFER(0);
  22    2             RBL=BUFFER_LENGTH;
  23    2             RBE=1;       /* Enable the SIU's receiver */
  24    2             XMIT_BUFFER_EMPTY=1;

  25    1     END POWER_ON_D;

  26    2     TRANSMIT: PROCEDURE (XMIT_BUFFER_LENGTH) BYTE   PUBLIC USING 0;

              /* User must check XMIT_BUFFER_EMPTY flag before calling this procedure */

  27    2             DECLARE XMIT_BUFFER_LENGTH  BYTE,
                              I                   BYTE    AUXILIARY,
                              STATUS              BYTE    AUXILIARY;

  28    2             IF USER_STATE=CLOSED_S
                          THEN STATUS=USER_STATE_CLOSED;
  30    2             ELSE IF STATION_STATE=DISCONNECT_S
                              THEN STATUS=LINK_DISCONNECTED;
  32    2             ELSE IF XMIT_BUFFER_LENGTH>BUFFER_LENGTH
                              THEN STATUS=OVERFLOW;
  34    3             ELSE DO;
```

```
 35    3                XMIT_BUFFER_EMPTY=0;
 36    3                TBL=XMIT_BUFFER_LENGTH;
 37    3                I_FRAME_LENGTH=XMIT_BUFFER_LENGTH; /* Store length in case station
                                                      is reset by FRMR, SNRM etc. */
 38    3                TBF=1;
 39    3                STATUS=DATA_TRANSMITTED;
 40    3              END;
 41    2        RETURN STATUS;
 42    1    END TRANSMIT;

 43    2    XMIT_UNNUMBERED: PROCEDURE (CONTROL_BYTE) ;

 44    2        DECLARE CONTROL_BYTE    BYTE;

 45    2            TCB=CONTROL_BYTE;
 46    2            TBF=1;
 47    2            RTS=1;
 48    3            DO WHILE NOT SI;
 49    3            END;
 50    2            SI=0;

 51    1    END XMIT_UNNUMBERED;

 52    2    SNRM_RESPONSE: PROCEDURE ;

 53    2        STATION_STATE=I_T_S;
 54    2        NSNR=0;
 55    2        IF (RCB AND 10H) <> 0 /* Respond if polled */
               THEN DO;
 57    3                TBL=0;
 58    3                CALL XMIT_UNNUMBERED(UA);
 59    3            END;
 60    2        IF XMIT_BUFFER_EMPTY=0  /* If an I frame was left pending transmission
                                        then restore it */
                THEN DO;
 62    3                TBL=I_FRAME_LENGTH;
 63    3                TBF=1;
 64    3             END;
 65    2        AM=1;

 66    1    END SNRM_RESPONSE;


 67    2    XMIT_FRMR: PROCEDURE (REASON)  ;

 68    2            DECLARE REASON  BYTE;


 69    2            TCB=FRMR;

 70    2            TBS=.FRMR_BUFFER(0);
 71    2            TBL=3;
 72    2            FRMR_BUFFER(0)=RCB;
                    /* Swap nibbles in NSNR */
 73    2            FRMR_BUFFER(1)=(SHL((NSNR AND 0EH),4) OR  SHR((NSNR AND 0E0H),4));
 74    3            DO CASE REASON;
 75    3                FRMR_BUFFER(2)=01H;  /* UNASSIGNED_C */
```

```
  76    3                 FRMR_BUFFER(2)=02H;   /* NO_I_FIELD_ALLOWED */
  77    3                 FRMR_BUFFER(2)=04H;   /* BUFF_OVERRUN */
  78    3                 FRMR_BUFFER(2)=08H;   /* SES_ERR */
  79    3             END;

  80    2            STATION_STATE=FRMR_S;

  81    2           IF (RCB AND 10H) <>0
                           THEN DO;
  83    3                          TBF=1;
  84    3                          RTS=1;
  85    4                          DO WHILE NOT SI;
  86    4                          END;
  87    3                          SI=0;
  88    3                       END;
  89    1     END XMIT_FRMR;


  90    2     IN_DISCONNECT_STATE: PROCEDURE ;/* Called from SIU_INT procedure */

  91    2             IF ((USER_STATE=OPEN_S) AND ((RCB AND 0EFH)=SNRM))
                           THEN CALL SNRM_RESPONSE;

  93    2             ELSE IF (RCB AND 10H) <> 0
                               THEN DO;
  95    3                              TBL=0;
  96    3                              CALL XMIT_UNNUMBERED(DM);
  97    3                           END;
  98    1     END IN_DISCONNECT_STATE;


  99    2     IN_FRMR_STATE: PROCEDURE ; /* Called by SIU_INT when a frame has been received
                                     when in the FRMR state */

 100    2          IF (RCB AND 0EFH)=SNRM
                      THEN DO;
 102    3                      CALL SNRM_RESPONSE;
 103    3                      TBS=.SIU_XMIT_BUFFER(0);  /* Restore transmit buffer start address */
 104    3                   END;


 105    2         ELSE IF (RCB AND 0EFH)=DISC
                      THEN DO;
 107    3                      STATION_STATE=DISCONNECT_S;
 108    3                      TBS=.SIU_XMIT_BUFFER(0);   /* Restore transmit buffer start address */
 109    3                      IF (RCB AND 10H)<> 0
                                   THEN DO;
 111    4                                  TBL=0;
 112    4                                  CALL XMIT_UNNUMBERED(UA);
 113    4                               END;
 114    3                   END;

 115    3         ELSE DO;       /* Receive control byte is something other than DISC or SNRM */
 116    3                  IF (RCB AND 10H) <> 0
                               THEN DO;
 118    4                              TBF=1;
 119    4                              RTS=1;
```

```
PL/M-51 COMPILER    RUPI-44 Secondary Station Driver                                    20:24:47  09/20/83  PAGE   5

 120   5                            DO WHILE NOT SI;
 121   5                            END;
 122   4                         END;
 123   3             END;

 124   1      END IN_FRMR_STATE;

 125   2      COMMAND_DECODE: PROCEDURE ;


 126   2          IF (RCB AND OEFH)=SNRM
                       THEN  CALL SNRM_RESPONSE;

 128   2          ELSE IF (RCB AND OEFH)=DISC
                           THEN  DO;
 130   3                            STATION_STATE=DISCONNECT_S;
 131   3                            IF (RCB AND 10H)<>0
                                        THEN DO;
 133   4                                        TBL=0;
 134   4                                        CALL XMIT_UNNUMBERED(UA);
 135   4                                     END;
 136   3                      END;

 137   2          ELSE IF (RCB AND OEFH)=TEST
                           THEN DO;
 139   3                       IF (RCB AND 10H)>0  /* Respond if polled */
                                  THEN DO;     /* FOR BOV=1, SEND THE TEST RESPONSE WITHOUT AN I FIELD */
 141   4                                IF (BOV=1)
                                            THEN DO;
 143   5                                            TBL=0;
 144   5                                            CALL XMIT_UNNUMBERED(TEST OR 10H);
 145   5                                         END;
 146   5                                ELSE DO; /* If no BOV, send received I field back to primary */
 147   5                                        TBL=RFL;
 148   5                                        TBS=RBS;
 149   5                                        CALL XMIT_UNNUMBERED(TEST OR 10H);
 150   5                                        TBS=.SIU_XMIT_BUFFER(0);  /* Restore TBS */
 151   5                                     END;

                                        /* If an I frame was pending, set it up again */

 152   4                                IF XMIT_BUFFER_EMPTY=0
                                            THEN DO;
 154   5                                            TBL=I_FRAME_LENGTH;
 155   5                                            TBF=1;
 156   5                                         END;
 157   4                             END;
 158   3                       AM=1;
 159   3                     END;

 160   2          ELSE IF (RCB AND 01H) = 0 /* Kicked out of the AUTO mode because
                                             an I frame was received while RPB = 1 */
                           THEN DO;
 162   3                       AM = 1;
 163   3                       IF XMIT_BUFFER_EMPTY = 1
                                  THEN TBL = 0;
 165   3                       TBF = 1;     /* Send an AUTO mode response */
```

```
PL/M-51 COMPILER    RUPI-44 Secondary Station Driver                                       20:24:47  09/20/83  PAGE   6

 166   3                           RTS = 1;
 167   3                        END;

 168   2           ELSE CALL XMIT_FRMR(UNASSIGNED_C);  /* Received an undefined or not implemented command */

 169   1       END COMMAND_DECODE;


 170   2       SIU_INT: PROCEDURE INTERRUPT 4;

 171   2               DECLARE     I   BYTE    AUXILIARY;

 172   2               SI=0;
 173   2               IF STATION_STATE<> I_T_S /* Must be in NON-AUTO mode */
                           THEN DO;
 175   3                           IF RBE=0 /* Received a frame?  Give response */
                                       THEN DO;
 177   5                                       DO CASE STATION_STATE;
 178   5                                           CALL IN_DISCONNECT_STATE;
 179   5                                           CALL IN_FRMR_STATE;
 180   5                                       END;
 181   4                                       RBE=1;
 182   4                                    END;
 183   3                       RETURN;
 184   3                       END;

               /* If the program reaches this point, STATION_STATE=I_T_S
                   which means the SIU either was, or still is in the AUTO MODE */


 185   2               IF AM=0
                           THEN DO;
 187   3                           IF (RCB AND 0EFH)=DISC
                                       THEN CALL COMMAND_DECODE;
 189   3                           ELSE IF USER_STATE=CLOSED_S
                                       THEN DO;
 191   4                                       TBL=0;
 192   4                                       CALL XMIT_UNNUMBERED(REQ_DISC);
 193   4                                   END;
 194   3                           ELSE IF SES=1
                                       THEN CALL XMIT_FRMR(SES_ERR);
 196   3                           ELSE IF BOV=1
                                       THEN DO;/* DON'T SEND FRMR IF A TEST WAS RECEIVED*/
 198   4                                       IF (RCB AND 0EFH)=TEST
                                                   THEN CALL COMMAND_DECODE;
 200   4                                       ELSE     CALL XMIT_FRMR(BUFF_OVERRUN);
 201   4                                   END;
 202   3                           ELSE CALL COMMAND_DECODE;
 203   3                           RBE=1;
 204   3                        END;

 205   3               ELSE DO;  /* MUST STILL BE IN AUTO MODE */
 206   3                       IF TBF=0
                                   THEN XMIT_BUFFER_EMPTY=1;   /* TRANSMITTED A FRAME */
 208   3                       IF RBE=0
                                   THEN DO;


PL/M-51 COMPILER    RUPI-44 Secondary Station Driver                                       20:24:47  09/20/83  PAGE   7

 210   4                                   RBP=1; /* RNR STATE */
 211   4                                   RBE=1; /* RE-ENABLE RECEIVER */
 212   4                                   CALL SIU_RECV(RFL);
 213   4                                   RBP=0; /* RR STATE */
 214   4                                END;
 215   3                          END;
 216   1       END SIU_INT;

 217   1       END MAIN$MOD;

Software and application note written by Charles Yager


WARNINGS:
    4 IS THE HIGHEST USED INTERRUPT


MODULE INFORMATION:                (STATIC+OVERLAYABLE)
    CODE SIZE                      = 028FH        655D
    CONSTANT SIZE                  = 0000H         0D
    DIRECT VARIABLE SIZE           =   3FH+02H    63D+  2D
    INDIRECT VARIABLE SIZE         =   3CH+00H    60D+  0D
    BIT SIZE                       =   01H+00H     1D+  0D
    BIT-ADDRESSABLE SIZE           =   00H+00H     0D+  0D
    AUXILIARY VARIABLE SIZE        = 0006H         6D
    MAXIMUM STACK SIZE             = 0017H        23D
    REGISTER-BANK(S) USED:           0 1 2
    460 LINES READ
    0 PROGRAM ERROR(S)
END OF PL/M-51 COMPILATION
```

```
PL/M-51 COMPILER    Application Module: Async/SDLC Protocol converter                    16:50:53  09/19/83  PAGE   1

ISIS-II PL/M-51 V1.0
COMPILER INVOKED BY:   :f2:plm51 :f2:unote.src


              $TITLE      ('Application Module: Async/SDLC Protocol converter')
              $debug
              $registerbank(0)
   1    1     user$mod:do;
              $NOLIST
   5    1     DECLARE     LIT                   LITERALLY         'LITERALLY',
                          TRUE                  LIT               'OFFH',
                          FALSE                 LIT               '00H',
                          FOREVER               LIT               'WHILE 1',
                          ESC                   LIT               '1BH',
                          LF                    LIT               '0AH',
                          CR                    LIT               '0DH',
                          BS                    LIT               '08H',
                          BEL                   LIT               '07H',
                          EMPTY                 LIT               '00H',
                          INUSE                 LIT               '01H',
                          FULL                  LIT               '02H',
                          USER_STATE_CLOSED     LIT               '00H',
                          LINK_DISCONNECTED     LIT               '01H',
                          OVERFLOW              LIT               '02H',
                          DATA_TRANSMITTED      LIT               '03H',

                              /* BUFFERS */

                          BUFFER_LENGTH         LIT               '60',
                          SIU_XMIT_BUFFER(BUFFER_LENGTH)      BYTE     EXTERNAL     IDATA,
                          SIU_RECV_BUFFER(BUFFER_LENGTH)      BYTE     EXTERNAL,
                          FIFO_T(256)               BYTE    AUXILIARY,
                          IN_PTR_T                  BYTE    AUXILIARY,
                          OUT_PTR_T                 BYTE    AUXILIARY,
                          BUFFER_STATUS_T           BYTE    AUXILIARY,
                          FIFO_R(256)               BYTE    AUXILIARY,
                          IN_PTR_R                  BYTE    AUXILIARY,
                          OUT_PTR_R                 BYTE    AUXILIARY,
                          BUFFER_STATUS_R           BYTE    AUXILIARY,

                              /* Variables and Parameters */

                          LENGTH                    BYTE    AUXILIARY,
                          CHAR                      BYTE    AUXILIARY,
                          I                         BYTE    AUXILIARY,
                          USART_CMD                 BYTE    AUXILIARY,
                          DESTINATION_ADDRESS       BYTE    AUXILIARY,
                          SEND_DATA                 BYTE    AUXILIARY,
                          RESULT                    BYTE    AUXILIARY,
                          ERR_MESSAGE_INDEX         BYTE    AUXILIARY,
                          ERR_MESSAGE_PTR           WORD    AUXILIARY,

                            /*   Messages Sent to the Terminal   */

                          PARITY(*) BYTE CONSTANT(LF,CR,'Parity Error Detected',LF,CR,00H),
                          FRAME(*) BYTE CONSTANT(LF,CR,'Framing Error Detected',LF,CR,00H),
OVER_RUN(*) BYTE CONSTANT(LF,CR,'Overrun Error Detected',LF,CR,0),
LINK(*) BYTE CONSTANT(LF,CR,'Unable to Get Online',LF,CR,00H),
DEST_ADDR(*) BYTE CONSTANT(CR,LF,LF,
            'Enter the destination address: __', BS, BS, 0),

D_ADDR_ACK(*) BYTE CONSTANT(CR,LF,LF,
            'The new destination address is ',0),

STAT_ADDR(*) BYTE CONSTANT(CR,LF,LF,
            'Enter the station address: __',BS,BS,0),

S_ADDR_ACK(*) BYTE CONSTANT(CR,LF,LF,
            'The new station address is ',0),

ADDR_ACK_FIN(*) BYTE CONSTANT('H',CR,LF,LF,0),



SIGN_ON(*)  BYTE CONSTANT(CR, LF,LF,
'(\/)  RUPI-44 Secondary Station', CR, LF,
' \/', CR, LF, LF,
'1 - Set the Station Address',LF,CR,
'2 - Set the Destination Address',CR,LF,
'3 - Go Online',CR,LF,
'4 - Go Offline',CR,LF,
'5 - Return to terminal mode',CR,LF,LF,
'    Enter option: _', BS, 0),

FIN(*)  BYTE CONSTANT(CR,LF,LF,0),



/* Characters Received From the Terminal */

HEX_TABLE(17) BYTE CONSTANT('0123456789ABCDEF',BEL),
MENU_CHAR(6) BYTE CONSTANT('12345',BEL),

  /* Flags and Bits */

XMIT_BUFFER_EMPTY        BIT     EXTERNAL, /* Semaphore  for RUPI SIOU Transmit Buffer */
STOP_BIT                 BIT     AT(147) REG, /* Terminal parameters */
ECHO                     BIT     AT(0B4H) REG,
WAIT                     BIT,              /* Timeout flag */
ERROR_FLAG               BIT,              /* Error message Flag */

  /* Periheral Addresses */

USART_STATUS             BYTE    AT(0801H) AUXILIARY,
USART_DATA               BYTE    AT(0800H) AUXILIARY,
TIMER_CONTROL            BYTE    AT(1003H) AUXILIARY,
TIMER_0                  BYTE    AT(1000H) AUXILIARY,
TIMER_1                  BYTE    AT(1001H) AUXILIARY,
TIMER_2                  BYTE    AT(1002H) AUXILIARY;


    /* External Procedures */
  6   2       POWER_ON_D: PROCEDURE   EXTERNAL;
  7   1       END POWER_ON_D;

  8   2       CLOSE: PROCEDURE EXTERNAL USING 2;
  9   1       END CLOSE;

 10   2       OPEN: PROCEDURE EXTERNAL USING 2;
 11   1       END OPEN;

 12   2       TRANSMIT:   PROCEDURE   (XMIT_BUFFER_LENGTH)    BYTE   EXTERNAL;
 13   2                   DECLARE XMIT_BUFFER_LENGTH  BYTE;
 14   1       END TRANSMIT;

                              /* Local Procedures */

 15   2       TIMER_O_INT: PROCEDURE INTERRUPT 1 USING 1;
 16   2           WAIT=0;
 17   1       END TIMER_O_INT;

 18   2       POWER_ON:   PROCEDURE USING 0;

 19   2               DECLARE TEMP    BYTE    AUXILIARY;

 20   2               SMD=54H;              /* Using DPLL, NRZI, PFS, TIMER 1, @ 62.5 Kbps */
 21   2               TMOD=21H;             /* Timer 0 16 bit, Timer 1 auto reload */
 22   2               TH1=0FFH;
 23   2               TCON=40H;

 24   2               TIMER_CONTROL=37H;   /* Initialize USART's system clock; 8254 */
 25   2               TIMER_0=04H;
 26   2               TIMER_0=00H;
 27   2               TIMER_CONTROL=77H;  /* Initialize TxC, RxC  */

      /*        Definition for dip switch tied to P1.0 to P1.6

                   Bit Rate   3       2       1
                 ________________________________
                     300     on      on      on
                    1200     on      on      off
                    2400     on      off     on
                    4800     on      off     off
                    9600     off     on      on
                   19200     off     on      off

                   Stop bit    4
                   _________________
                      1        on
                      2        off

                   Parity     6       5
                   ________________________
                   off        on      on
                   odd        on      off
                   off        off     on
                   even       off     off
```

```
                        Echo        7
                        ______________
                        on          on
                        off         off                  */

 28   2             TEMP=P1 AND 07H;  /* Read the dip switch to determine the bit rate */
 29   2             IF TEMP>5
                        THEN TEMP=0;
 31   3             DO CASE TEMP;
            /* 300 */
 32   4                 DO;
 33   4                     TIMER_1=83H;
 34   4                     TIMER_1=20H;
 35   4                 END;

 36   4     /* 1200 */  DO;
 37   4                     TIMER_1=20H;
 38   4                     TIMER_1=05H;
 39   4                 END;

 40   4     /* 2400 */  DO;
 41   4                     TIMER_1=60H;
 42   4                     TIMER_1=02H;
 43   4                 END;

 44   4     /* 4800 */  DO;
 45   4                     TIMER_1=30H;
 46   4                     TIMER_1=01H;
 47   4                 END;

 48   4     /* 9600 */  DO;
 49   4                     TIMER_1=65H;
 50   4                     TIMER_1=0;
 51   4                 END;

 52   4     /* 19200 */ DO;
 53   4                     TIMER_1=33H;
 54   4                     TIMER_1=0;
 55   4                 END;
 56   3             END;

 57   2             USART_STATUS=0;  /* Software power-on reset for 8251A */
 58   2             USART_STATUS=0;
 59   2             USART_STATUS=0;
 60   2             USART_STATUS=40H;

 61   2             TEMP=0AH;        /* Determine the parity and # of stop bits */
 62   2             TEMP=TEMP OR (P1 AND 30H);
 63   2             IF STOP_BIT=1
                        THEN TEMP=TEMP OR 0C0H;
 65   2             ELSE TEMP=TEMP OR 40H;

 66   2             USART_STATUS=TEMP;  /* USART Mode Word */
 67   2             USART_STATUS, USART_CMD=27H; /*USART Command Word RTS, RxE, DTR, TxEN=1*/

 68   2             STAD=0FFH;
```

```
  69   2             SEND_DATA=0; /* Intialize Flags */

  70   2            IN_PTR_T, OUT_PTR_T, IN_PTR_R, OUT_PTR_R = 0;/*Initialize FIFO PTRs*/

  71   2             BUFFER_STATUS_T, BUFFER_STATUS_R= EMPTY;

  72   2             CALL POWER_ON_D;

  73   2             IP=01H;          /* USART's RxRdy is the highest priority */
                                        /* Both external interrupts are level triggered*/
  74   2             IE=93H;            /* Enable USART RxRdy, SI, and Timer 0 interrupts*/

  75   2             ERROR_FLAG=0;

  76   1     END POWER_ON;

  77   2     FIFO_R_IN: PROCEDURE (CHAR) USING 1;
  78   2         DECLARE CHAR    BYTE;

  79   2         FIFO_R(IN_PTR_R)=CHAR;
  80   2         IN_PTR_R=IN_PTR_R+1;

  81   2         IF BUFFER_STATUS_R=EMPTY
                     THEN DO;
  83   3                     EA=0;
  84   3                     BUFFER_STATUS_R=INUSE;
  85   3                     EX1=1;      /* Enable USART's TxD interrupt */
  86   3                     EA=1;
  87   3                  END;
  88   2         ELSE IF ((BUFFER_STATUS_R=INUSE) AND (IN_PTR_R=OUT_PTR_R))
                         THEN BUFFER_STATUS_R=FULL;

  90   1     END FIFO_R_IN;

  91   2     FIFO_R_OUT: PROCEDURE BYTE USING 1;

  92   2         DECLARE CHAR    BYTE    AUXILIARY;

  93   2         CHAR=FIFO_R(OUT_PTR_R);
  94   2         OUT_PTR_R=OUT_PTR_R+1;
  95   2         IF OUT_PTR_R=IN_PTR_R
                     THEN DO;
  97   3                     EX1=0;  /* Shut off TxD interrupt */
  98   3                     BUFFER_STATUS_R=EMPTY;
  99   3                  END;
 100   2         ELSE IF ((BUFFER_STATUS_R=FULL) AND (OUT_PTR_R-20=IN_PTR_R))
                         THEN BUFFER_STATUS_R=INUSE;

 102   2         RETURN CHAR;

 103   1     END FIFO_R_OUT;


 104   2     USART_XMIT_INT: PROCEDURE INTERRUPT 2 USING 1;
 105   2            DECLARE
                            MESSAGE BASED ERR_MESSAGE_PTR(1)     BYTE     CONSTANT;

 106   2            IF ERROR_FLAG
                        THEN DO;
 108   3                        IF MESSAGE(ERR_MESSAGE_INDEX)<>0  /* Then continue to send the message */
                                    THEN DO;
 110   4                                    USART_DATA = MESSAGE(ERR_MESSAGE_INDEX);
 111   4                                    ERR_MESSAGE_INDEX=ERR_MESSAGE_INDEX+1;
 112   4                                 END;

 113   4                        ELSE DO; /* If message is done reset ERROR_FLAG and shut off interrupt if FIFO is empty */
 114   4                                ERROR_FLAG=0;
 115   4                                IF BUFFER_STATUS_R = EMPTY
                                            THEN EX1=0;
 117   4                             END;
 118   3                     END;

 119   2            ELSE USART_DATA=FIFO_R_OUT;

 120   1        END USART_XMIT_INT;


 121   2        SIU_RECV: PROCEDURE (LENGTH) PUBLIC USING 1;

 122   2            DECLARE LENGTH  BYTE,
                            I       BYTE    AUXILIARY;

 123   3            DO I=0 TO LENGTH-1;
 124   4                DO WHILE BUFFER_STATUS_R=FULL;/* Check to see if fifo is full */
 125   4                END;
 126   3                CALL FIFO_R_IN(SIU_RECV_BUFFER(I));
 127   3            END;

 128   1        END SIU_RECV;


 129   2        FIFO_T_IN: PROCEDURE (CHAR) USING 2;

 130   2            DECLARE CHAR         BYTE;

 131   2            FIFO_T(IN_PTR_T)=CHAR;
 132   2            IN_PTR_T=IN_PTR_T+1;
 133   2            IF CHAR=LF
                        THEN SEND_DATA=SEND_DATA+1;

 135   2            IF BUFFER_STATUS_T=EMPTY
                        THEN BUFFER_STATUS_T=INUSE;
 137   2            ELSE IF ((BUFFER_STATUS_T=INUSE) AND (IN_PTR_T+20=OUT_PTR_T))
                             THEN DO; /* Stop reception using CTS */
 139   3                             USART_STATUS, USART_CMD=USART_CMD AND NOT(20H);
 140   3                             BUFFER_STATUS_T=FULL;
 141   3                             IF SEND_DATA=0
                                         THEN SEND_DATA=1;/*If the buffer is full and no LF
                                                            has been received then send data */
 143   3                          END;
 144   1        END FIFO_T_IN;
 145   2       FIFO_T_OUT: PROCEDURE BYTE ;

 146   2           DECLARE CHAR    BYTE    AUXILIARY;

 147   2           CHAR=FIFO_T(OUT_PTR_T);
 148   2           OUT_PTR_T=OUT_PTR_T+1;
 149   2           IF OUT_PTR_T=IN_PTR_T   /* Then FIFO_T is empty */
                       THEN DO;
 151   3                       EA=0;
 152   3                       BUFFER_STATUS_T=EMPTY;
 153   3                       SEND_DATA=0;
 154   3                       EA=1;
 155   3                    END;
 156   2           ELSE IF ((BUFFER_STATUS_T=FULL) AND (OUT_PTR_T-80=IN_PTR_T))
                           THEN DO;
 158   3                           USART_STATUS, USART_CMD=USART_CMD OR 20H;
 159   3                           BUFFER_STATUS_T=INUSE;
 160   3                        END;
 161   2           IF (CHAR=LF AND SEND_DATA>0) THEN SEND_DATA=SEND_DATA-1;
 163   2           RETURN CHAR;
 164   1       END FIFO_T_OUT;

 165   2       ERROR: PROCEDURE (STATUS) USING 2;

 166   2               DECLARE STATUS      BYTE;

 167   2               IF (STATUS AND 08H)<>0
                           THEN ERR_MESSAGE_PTR=.PARITY;
 169   2               ELSE IF (STATUS AND 10H)<>0
                           THEN ERR_MESSAGE_PTR=.OVER_RUN;
 171   2               ELSE IF (STATUS AND 20H)<>0
                           THEN ERR_MESSAGE_PTR=.FRAME;

 173   2               USART_STATUS=(USART_CMD OR 10H); /* Reset error flags on USART */

 174   2               ERR_MESSAGE_INDEX = 0;
 175   2               ERROR_FLAG=1;
 176   2               EX1=1;          /* Turn on Tx Interrupt */

 177   1       END ERROR;

 178   2       LINK_DISC: PROCEDURE ;

           /* This procedure sends the message 'Unable to Get Online' to the terminal */

 179   2           DECLARE MESSAGE_PTR WORD    AUXILIARY,
                           MESSAGE     BASED   MESSAGE_PTR(1)      BYTE    CONSTANT,
                           J           BYTE    AUXILIARY,
                           EX1_STORE   BIT;


 180   2           EX1_STORE=EX1;     /* Shut off async transmit interrupt */
 181   2           EX1=0;
 182   2           MESSAGE_PTR=.LINK;
 183   2           J=0;
 184   3           DO WHILE (MESSAGE(J)<>0);
 185   4                DO WHILE (USART_STATUS AND 01H)=0;   /* Wait for TxRDY on USART */
 186   4                END;
 187   3                USART_DATA=MESSAGE(J);
 188   3                J=J+1;
 189   3            END;
 190   2            EX1=EX1_STORE;     /* Restore async transmit interrupt */
 191   1        END LINK_DISC;


 192   2        CO: PROCEDURE (CHAR) USING 2;
 193   2                DECLARE CHAR    BYTE;

 194   3                DO WHILE (USART_STATUS AND 01H) = 0;
 195   3                END;
 196   2                USART_DATA=CHAR;

 197   1        END CO;

 198   2        CI: PROCEDURE BYTE USING 2;

 199   3            DO WHILE (USART_STATUS AND 02H) = 0;
 200   3            END;
 201   2            RETURN USART_DATA;

 202   1        END CI;


 203   2        GET_HEX: PROCEDURE BYTE USING 2;

 204   2            DECLARE CHAR    BYTE    AUXILIARY,
                            I       BYTE    AUXILIARY;

 205   2            LO: CHAR=CI;

 206   3                DO I=0 TO 15;
 207   3                    IF CHAR=HEX_TABLE(I)
                                THEN GOTO L1;
 209   3                END;

 210   2            L1: CALL CO(HEX_TABLE(I));
 211   2                IF I=16
                            THEN GOTO LO;

 213   2                RETURN I;

 214   1        END GET_HEX;

 215   2        OUTPUT_MESSAGE: PROCEDURE (MESSAGE_PTR) USING 2;
 216   2            DECLARE MESSAGE_PTR WORD,
                            MESSAGE     BASED   MESSAGE_PTR(1)  BYTE CONSTANT,
                            I           BYTE    AUXILIARY;

 217   2            I=0;

 218   3            DO WHILE MESSAGE(I) <> 0;
 219   3                CALL CO(MESSAGE(I));
 220   3                I=I+1;
 221   3          END;

 222   1      END OUTPUT_MESSAGE;


 223   2      MENU: PROCEDURE USING 2;

 224   2          DECLARE I               BYTE    AUXILIARY,
                          CHAR            BYTE    AUXILIARY,
                          STATION_ADDRESS BYTE    AUXILIARY;

 225   2      START:
                  CALL OUTPUT_MESSAGE(.SIGN_ON);

 226   2          MO: CHAR=CI;    /* Read a character */

 227   3              DO I=0 TO 4;
 228   3                  IF CHAR=MENU_CHAR(I)
                              THEN GOTO M1;
 230   3              END;

 231   2          M1: CALL CO(MENU_CHAR(I));
 232   2              IF I=5
                          THEN GOTO MO;

 234   3          DO CASE I;

 235   4              DO;
 236   4                  CALL OUTPUT_MESSAGE(.STAT_ADDR);

 237   4                  STATION_ADDRESS=SHL(GET_HEX,4);

 238   4                  STATION_ADDRESS=(STATION_ADDRESS OR GET_HEX);

 239   4                  STAD=STATION_ADDRESS;

 240   4                  CALL OUTPUT_MESSAGE(.S_ADDR_ACK);

 241   4                  CALL CO(HEX_TABLE(SHR(STATION_ADDRESS,4)));
 242   4                  CALL CO(HEX_TABLE(0FH AND STATION_ADDRESS));

 243   4                  CALL OUTPUT_MESSAGE(.ADDR_ACK_FIN);
 244   4              END;

 245   4              DO;

 246   4                  CALL OUTPUT_MESSAGE(.DEST_ADDR);

 247   4                  DESTINATION_ADDRESS=SHL(GET_HEX,4);

 248   4                  DESTINATION_ADDRESS=(DESTINATION_ADDRESS OR GET_HEX );

 249   4                  CALL OUTPUT_MESSAGE(.D_ADDR_ACK);
 250   4                   CALL CO(HEX_TABLE(SHR(DESTINATION_ADDRESS,4)));
 251   4                   CALL CO(HEX_TABLE(OFH AND DESTINATION_ADDRESS));

 252   4                   CALL OUTPUT_MESSAGE(.ADDR_ACK_FIN);
 253   4               END;

 254   4               DO;
 255   4                   CALL OUTPUT_MESSAGE(.FIN);
 256   4                   CALL OPEN;
 257   4               END;


 258   4               DO;
 259   4                   CALL OUTPUT_MESSAGE(.FIN);
 260   4                   CALL CLOSE;
 261   4               END;

 262   3               CALL OUTPUT_MESSAGE(.FIN);

 263   3           END; /* DO CASE */

 264   1       END MENU;


 265   2       USART_RECV_INT: PROCEDURE INTERRUPT 0 USING 2;

 266   2           DECLARE CHAR          BYTE    AUXILIARY,
                           STATUS        BYTE    AUXILIARY;


 267   2           CHAR=USART_DATA;
 268   2           STATUS=USART_STATUS AND 38H;
 269   2           IF STATUS<>0
                       THEN CALL ERROR(STATUS);
 271   2           ELSE IF CHAR=ESC
                           THEN CALL MENU;
 273   3           ELSE DO;
 274   3                   CALL FIFO_T_IN(CHAR);
 275   3                   IF ECHO=0
                               THEN CALL CO(CHAR);
 277   3                END;

 278   1       END USART_RECV_INT;


 279   1       BEGIN:
               CALL POWER_ON;

 280   2           DO FOREVER;
 281   2               IF SEND_DATA>0
                           THEN DO;
 283   4                           DO WHILE NOT(XMIT_BUFFER_EMPTY); /*Wait until SIU_XMIT_BUFFer
                                                                     is empty */
 284   4                           END;
 285   3                           LENGTH, CHAR =1;
 286   3                           SIU_XMIT_BUFFER(0)=DESTINATION_ADDRESS;
 287   4                           DO WHILE ((CHAR<>LF) AND (LENGTH<BUFFER_LENGTH) AND (BUFFER_STATUS_T<>EMPTY));
 288   4                        CHAR=FIFO_T_OUT;
 289   4                        SIU_XMIT_BUFFER(LENGTH)=CHAR;
 290   4                        LENGTH=LENGTH+1;
 291   4                    END;

             /* If the line entered at the terminal is greater than BUFFER_LENGTH char, send the
                first BUFFER_LENGTH char, then send the rest; since the SIU buffer is only BUFFER_LENGTH bytes */
 292   3     L1:                  I=0; /* Use I to count the number of unsuccesful
                                          transmits */

 293   3     RETRY:          RESULT=TRANSMIT(LENGTH); /* Send the message */
 294   3                          IF RESULT<>DATA_TRANSMITTED
                                       THEN DO;
                              /* Wait 50 msec for link to connect then try again */
 296   4                                    WAIT=1;
 297   4                                    TH0=3CH;
 298   4                                    TL0=0AFH;
 299   4                                    TR0=1;
 300   5                                    DO WHILE WAIT;
 301   5                                    END;
 302   4                                    TR0=0;
 303   4                                    I=I+1;
 304   5                                    IF I>100 THEN DO;  /* Wait 5 sec to get on line else
                                                                    send error message to terminal
                                                                    and try again  */
 306   5                                                    CALL LINK_DISC;
 307   5                                                    GOTO L1;
 308   5                                                 END;
 309   4                                    GOTO RETRY;
 310   4                                END;
 311   3                       END;

 312   2         END;
 313   1     END USER$MOD;



WARNINGS:
    2 IS THE HIGHEST USED INTERRUPT


MODULE INFORMATION:                    (STATIC+OVERLAYABLE)
    CODE SIZE                          = 06B2H        1714D
    CONSTANT SIZE                      = 01CFH         463D
    DIRECT VARIABLE SIZE               =    00H+05H      0D+  5D
    INDIRECT VARIABLE SIZE             =    00H+00H      0D+  0D
    BIT SIZE                           =    02H+01H      2D+  1D
    BIT-ADDRESSABLE SIZE               =    00H+00H      0D+  0D
    AUXILIARY VARIABLE SIZE            = 021FH         543D
    MAXIMUM STACK SIZE                 = 0028H          40D
    REGISTER-BANK(S) USED:               0 1 2
    713 LINES READ
    0 PROGRAM ERROR(S)
END OF PL/M-51 COMPILATION
```

```
PL/M-51 COMPILER    RUPI-44 Primary Station                                              20:47:13  09/26/83  PAGE   1

ISIS-II PL/M-51 V1.0
COMPILER INVOKED BY:   :F2:PLM51 :F2:PNOTE.SRC


               $TITLE      ('RUPI-44 Primary Station')
               $DEBUG
               $REGISTERBANK(0)
   1    1      MAIN$MOD:DO;

               /* To save paper the RUPI registers are not listed, but this is the statement
                  used to include them:  $INCLUDE (:f2:REG44.DCL) */

               $NOLIST

   5    1      DECLARE LIT             LITERALLY    'LITERALLY',
                       TRUE            LIT          'OFFH',
                       FALSE           LIT          'OOH',
                       FOREVER         LIT          'WHILE 1';

               /* SDLC COMMANDS AND RESPONSES */

   6    1      DECLARE SNRM            LIT          '93H',
                       UA              LIT          '73H',
                       DISC            LIT          '53H',
                       DM              LIT          '1FH',
                       FRMR            LIT          '97H',
                       REQ_DISC        LIT          '53H',
                       UP              LIT          '33H',
                       TEST            LIT          'OF3H',
                       RR              LIT          '11H',
                       RNR             LIT          '15H',


                               /* REMOTE STATION BUFFER STATUS */

                       BUFFER_READY        LIT          '0',
                       BUFFER_NOT_READY    LIT          '1',

                               /* STATION STATES */
                       DISCONNECT_S    LIT          'OOH', /* LOGICALLY DISCONNECTED STATE*/
                       GO_TO_DISC      LIT          '01H',
                       I_T_S           LIT          '02H', /* INFORMATION TRANSFER STATE */

                       /* PARAMETERS PASSED TO XMIT_I_T_S */
                       T_I_FRAME           LIT      'OOH',
                       T_RR                LIT      '01H',
                       T_RNR               LIT      '02H',

                       /* SECONDARY STATION IDENTIFICATION */

                       NUMBER_OF_STATIONS  LIT      '2',
                       SECONDARY_ADDRESSES(NUMBER_OF_STATIONS)
                                           BYTE     CONSTANT(55H,43H),
PL/M-51 COMPILER    RUPI-44 Primary Station                                            20:47:13  09/26/83  PAGE   2


                         /* Remote Station Database */

                    RSD(NUMBER_OF_STATIONS) STRUCTURE
                    (STATION_ADDRESS     BYTE,
                     STATION_STATE       BYTE,
                     NS                  BYTE,
                     NR                  BYTE,
                     BUFFER_STATUS       BYTE,  /* The status of the secondary stations buffer */
                     INFO_LENGTH         BYTE,
                     DATA(64)            BYTE)   AUXILIARY,


                         /* VARIABLES */
                    STATION_NUMBER       BYTE    AUXILIARY,
                    RECV_FIELD_LENGTH    BYTE    AUXILIARY,
                    WAIT                 BIT,

                         /* BUFFERS */
                    SIU_XMIT_BUFFER(64)      BYTE    IDATA,
                    SIU_RECV_BUFFER(64)      BYTE;


   7   2        POWER_ON: PROCEDURE ;

   8   2            DECLARE I   BYTE    AUXILIARY;

   9   2                TBS=.SIU_XMIT_BUFFER(0);
  10   2                RBS=.SIU_RECV_BUFFER(0);
  11   2                RBL=64;      /* 64 Byte receive buffer */
  12   2                RBE=1;       /* Enable the SIU's receiver */

  13   3                DO I= 0 TO NUMBER_OF_STATIONS-1;

  14   3                    RSD(I).STATION_ADDRESS=SECONDARY_ADDRESSES(I);
  15   3                    RSD(I).STATION_STATE=DISCONNECT_S;
  16   3                    RSD(I).BUFFER_STATUS=BUFFER_NOT_READY;
  17   3                    RSD(I).INFO_LENGTH=0;


  18   3                END;

  19   2                SMD=54H; /* Using DPLL, NRZI, PFS, TIMER 1, @ 62.5 Kbps */
  20   2                TMOD=21H;
  21   2                TH1=0FFH;
  22   2                TCON=40H; /* Use timer 0 for receive time out interrupt */
  23   2                IE=82H;

  24   1        END POWER_ON;

  25   2        XMIT: PROCEDURE (CONTROL_BYTE);

  26   2            DECLARE CONTROL_BYTE    BYTE;

  27   2                TCB=CONTROL_BYTE;
  28   2                TBF=1;
```

```
 29   2             RTS=1;
 30   3             DO WHILE NOT SI;
 31   3             END;
 32   2             SI=0;

 33   1     END XMIT;


 34   2     TIMER_0_INT: PROCEDURE INTERRUPT 1  USING 1;
 35   2         WAIT=0;
 36   1     END TIMER_0_INT;


 37   2     TIME_OUT: PROCEDURE BYTE;          /* Time_out returns true if there wasn't
                                                   a frame received within 200 msec.
                                                   If there was a frame received within
                                                   200 msec then time_out returns false. */
 38   2         DECLARE     I   BYTE    AUXILIARY;

 39   3         DO I=0 TO 3;

 40   3             WAIT=1;
 41   3             TH0=3CH;
 42   3             TL0=0AFH;
 43   3             TR0=1;
 44   4                 DO WHILE WAIT;
 45   4                     IF SI=1
                                THEN GOTO T_01;
 47   4                 END;
 48   3         END;

 49   2         RETURN TRUE;

 50   2     T_01:
                    SI=0;
 51   2             RETURN FALSE;

 52   1     END TIME_OUT;

 53   2     SEND_DISC: PROCEDURE;

 54   2                 TBL=0;
 55   2                 CALL XMIT(DISC);
 56   2                 IF TIME_OUT=FALSE
                            THEN IF RCB=UA OR RCB=DM
 57   3                             THEN DO;
 59   3                                     RSD(STATION_NUMBER).BUFFER_STATUS=BUFFER_NOT_READY;
 60   3                                     RSD(STATION_NUMBER).STATION_STATE=DISCONNECT_S;
 61   3                                 END;
 62   2                 RBE=1;

 63   1     END SEND_DISC;


 64   2     SEND_SNRM: PROCEDURE;

 65   2         TBL=0;
```

```
  66   2          CALL XMIT(SNRM);
  67   2          IF (TIME_OUT=FALSE) AND (RCB=UA)
                        THEN DO;
  69   3                        RSD(STATION_NUMBER).STATION_STATE=I_T_S;
  70   3                        RSD(STATION_NUMBER).NS=0;
  71   3                        RSD(STATION_NUMBER).NR=0;
  72   3                     END;
  73   2          RBE=1;

  74   1      END SEND_SNRM;


  75   2      CHECK_NS: PROCEDURE BYTE;

               /* Check the Ns Field of the received frame. If Nr(P)=Ns(S) return true */

  76   2          IF (RSD(STATION_NUMBER).NR=(SHR(RCB,1) AND 07H))
                      THEN RETURN TRUE;
  78   2              ELSE RETURN FALSE;

  79   1      END CHECK_NS;

  80   2      CHECK_NR: PROCEDURE BYTE;

              /* Check the Nr field of the received frame. If Ns(P)+1=Nr(S) then the frame
                  has been acknowledged, else if Ns(P)=Nr(S) then the frame has not been
                  acknowledged, else reset the secondary */

  81   2          IF (((RSD(STATION_NUMBER).NS + 1) AND 07H) = SHR(RCB,5))
                      THEN DO;
  83   3                    RSD(STATION_NUMBER).NS=((RSD(STATION_NUMBER).NS+1) AND 07H);
  84   3                    RSD(STATION_NUMBER).INFO_LENGTH=0;
  85   3                 END;
  86   2          ELSE IF (RSD(STATION_NUMBER).NS <> SHR(RCB,5))
                          THEN RETURN FALSE;

  88   2          RETURN TRUE;

  89   1      END CHECK_NR;


  90   2      RECEIVE: PROCEDURE ;

  91   2              DECLARE      I    BYTE    AUXILIARY;

  92   2              RSD(STATION_NUMBER).BUFFER_STATUS=BUFFER_READY;

               /* If an RNR was received buffer_status will be changed in the supervisory
                  frame decode section futher down in this procedure, any other response
                  means the remote stations buffer is ready  */

  93   2              IF (RCB AND 01H)=0
                          THEN DO;   /* I Frame Received */
  95   3                        IF (CHECK_NS=TRUE AND BOV=0 AND CHECK_NR=TRUE)
                                     THEN DO;
  97   4                                    RSD(STATION_NUMBER).NR=((RSD(STATION_NUMBER).NR+1) AND 07H);
  98   4                                    RBP=1;
  99   4                                      RECV_FIELD_LENGTH=RFL-1;
 100   4                                   END;

 101   3                          ELSE RSD(STATION_NUMBER).STATION_STATE=GO_TO_DISC;
 102   3                       END;
 103   2              ELSE IF (RCB AND 03H)=01H
                              THEN DO;  /* Supervisory frame received */
 105   3                              IF CHECK_NR=FALSE
                                          THEN RSD(STATION_NUMBER).STATION_STATE=GO_TO_DISC;

 107   3                              ELSE IF ((RCB AND 0FH)=05H) /* then RNR */
                                              THEN RSD(STATION_NUMBER).BUFFER_STATUS=BUFFER_NOT_READY;
 109   3                          END;

 110   3              ELSE DO;        /* Unnumbered frame or unknown frame received */
 111   3                       IF RCB=FRMR
                                  THEN DO;    /* If FRMR was received check Nr for an
                                                acknowledged I frame  */
 113   4                                  RCB=SIU_RECV_BUFFER(1);
 114   4                                  I=CHECK_NR;
 115   4                               END;
 116   3                       RSD(STATION_NUMBER).STATION_STATE=GO_TO_DISC;
 117   3                   END;

 118   2              RBE=1;

 119   1      END RECEIVE;



 120   2      XMIT_I_T_S: PROCEDURE (TEMP);

 121   2          DECLARE     TEMP    BYTE;

 122   2              IF TEMP=T_I_FRAME
                          THEN DO;   /* Transmit I frame */
                                  /* Transfer the station buffer into internal ram */

 124   4                          DO TEMP=0 TO RSD(STATION_NUMBER).INFO_LENGTH-1;
 125   4                          SIU_XMIT_BUFFER(TEMP)=RSD(STATION_NUMBER).DATA(TEMP);
 126   4                          END;

                                  /* Build the I frame control field */

 127   3                          TEMP=(SHL(RSD(STATION_NUMBER).NR,5) OR SHL(RSD(STATION_NUMBER).NS,1) OR 10H);
 128   3                          TBL=RSD(STATION_NUMBER).INFO_LENGTH;
 129   3                          CALL XMIT(TEMP);
 130   3                          IF TIME_OUT=FALSE
                                      THEN CALL RECEIVE;
 132   3                      END;

 133   3              ELSE DO;    /* Transmit RR or RNR*/
 134   3                      IF TEMP=T_RR
                                  THEN TEMP=RR;
 136   3                      ELSE TEMP=RNR;
```

```
PL/M-51 COMPILER    RUPI-44 Primary Station                                                          20:47:13  09/26/83  PAGE    6

 137   3                   TEMP=(SHL(RSD(STATION_NUMBER).NR,5) OR TEMP);
 138   3                   TBL=0;
 139   3                   CALL XMIT(TEMP);
 140   3                   IF TIME_OUT=FALSE
                              THEN CALL RECEIVE;
 142   3                 END;
 143   1       END XMIT_I_T_S;

 144   2       BUFFER_TRANSFER: PROCEDURE;

 145   2           DECLARE      I        BYTE    AUXILIARY,
                                J        BYTE    AUXILIARY;


 146   3           DO I=0 TO NUMBER_OF_STATIONS-1;
 147   3               IF RSD(I).STATION_ADDRESS=SIU_RECV_BUFFER(0)
                           THEN GOTO T1;
 149   3           END;

 150   2         T1:   IF I=NUMBER_OF_STATIONS  /* If the addressed station does not exits,
                                                    then discard the data */
                           THEN DO;
 152   3                       RBP=0;
 153   3                       RETURN;
 154   3                    END;
 155   2               ELSE IF RSD(I).INFO_LENGTH=0
                           THEN DO;
 157   3                       RSD(I).INFO_LENGTH=RECV_FIELD_LENGTH;
 158   4                       DO J=1 TO RECV_FIELD_LENGTH;
 159   4                           RSD(I).DATA(J-1)=SIU_RECV_BUFFER(J);
 160   4                       END;
 161   3                       RBP=0;
 162   3                    END;

 163   1       END BUFFER_TRANSFER;


 164   1       BEGIN:
                   CALL POWER_ON;

 165   2           DO FOREVER;

 166   3               DO STATION_NUMBER=0 TO NUMBER_OF_STATIONS-1;
 167   3                   STAD=RSD(STATION_NUMBER).STATION_ADDRESS;
 168   3                   IF RSD(STATION_NUMBER).STATION_STATE = DISCONNECT_S
                               THEN CALL SEND_SNRM;
 170   3                   ELSE IF RSD(STATION_NUMBER).STATION_STATE = GO_TO_DISC
                               THEN CALL SEND_DISC;
 172   3                   ELSE IF ((RSD(STATION_NUMBER).INFO_LENGTH>0) AND
                                    (RSD(STATION_NUMBER).BUFFER_STATUS=BUFFER_READY))
                               THEN CALL XMIT_I_T_S(T_I_FRAME);
 174   3                   ELSE IF RBP=0
                               THEN CALL XMIT_I_T_S(T_RR);
 176   3                   ELSE CALL XMIT_I_T_S(T_RNR);

 177   3                   IF RBP=1
                               THEN CALL BUFFER_TRANSFER;
```

```
PL/M-51 COMPILER    RUPI-44 Primary Station                                                          20:47:13  09/26/83  PAGE    7

 179   3               END;

 180   2           END;

 181   1       END MAIN$MOD;



WARNINGS:
    1 IS THE HIGHEST USED INTERRUPT


MODULE INFORMATION:                     (STATIC+OVERLAYABLE)
    CODE SIZE                           = 053DH       1341D
    CONSTANT SIZE                       = 0002H          2D
    DIRECT VARIABLE SIZE                =   40H+02H     64D+   2D
    INDIRECT VARIABLE SIZE              =   40H+00H     64D+   0D
    BIT SIZE                            =   01H+00H      1D+   0D
    BIT-ADDRESSABLE SIZE                =   00H+00H      0D+   0D
    AUXILIARY VARIABLE SIZE             = 0093H        147D
    MAXIMUM STACK SIZE                  = 0019H         25D
    REGISTER-BANK(S) USED:                 0 1
    456 LINES READ
    0 PROGRAM ERROR(S)
END OF PL/M-51 COMPILATION
```

A HIGH PERFORMANCE NETWORK USING THE 8044

20.2.1 Introduction

This section describes the design of an SDLC data link using the 8044 (RUPI) to implement a primary station and a secondary station. The design was implemented and tested. The following discussion assumes that the reader understands the 8044 and SDLC. This section is divided into two parts. First the data link design example is discussed. Second the software modules used to implement the data link are described. To help the reader understand the discussion of the software, flow charts and software listings are displayed in Appendix A and Appendix B, respectively.

Application Description

This particular data link design example uses a two wire half-duplex multidrop topology as shown in figure 20-4. In an SDLC multidrop topology the primary station communicates with each secondary station. The secondary stations communicate only to the primary. Because of this hierarchial architecture, the logical topology for an SDLC multidrop is a star as shown in figure 20-5. Although the physical topology of this data link is multidrop, the easiest way to understand the information flow is to think of the logical (star) topology. The term data link in this case refers to the logical communication pathways between the primary station and the secondary stations. The data links are shown in figure 20-5 as two way arrows.

The application example uses dumb async terminals to interface to the SDLC network. Each secondary station has an async terminal connected to it. The secondary stations are in effect protocol converters which allows any async terminal to communicate with any other async terminal on the network. The secondary stations use an 8044 with a UART to convert SDLC to async. Figure 20-6 displays a block diagram of the data link. The primary station, controls the data link. In addition to data link control the primary provides a higher level layer which is a path control function or networking layer. The primary serves as a message exchange or switch. It receives information from one secondary station and retransmits it to another secondary station. Thus a virtual end to end connection is made between any two secondary stations on the network.

Three separate software modules were written for this network. The first module is a Secondary Station Driver (SSD) which provides an SDLC data link interface and a user interface. This module is a general purpose driver which requires application software to run it. The user interface to the driver provides four functions: OPEN, CLOSE, TRANSMIT, and SIU_RECV. Using these four functions properly will allow any application software to communicate over this SDLC data link without knowing the details of SDLC. The secondary station driver uses the 8044's AUTO mode.

The second module is an example of application software which is linked to the secondary station driver. This module drives the 8251A, buffers data, and interfaces with the secondary station driver's user interface.

The third module is a primary station, which is a standalone program (i.e., it is not linked to any other module). The primary station uses the 8044's NON-AUTO or FLEXIBLE mode. In addition to controlling the data link it acts as a message switch. Each time a secondary station transmits a frame, it places the destination address of the frame in the first byte of the information or I field. When the primary station receives a frame, it removes the first byte in the I field and retransmits the frame to the secondary station whose address matches this byte.

This network provides two complete layers of the OSI (Open Systems Interconnection) reference model: the physical layer and the data link layer. The physical layer implementation uses the RS-422 electrical interface. The mechanical medium consists of ribbon cable and connectors. The data link layer is defined by SDLC. SDLC's use of acknowledgements and frame numbering guarantees that messages will be received in the same order in which they were sent. It also guarantees message integrity over the data link. However this network will not guarantee secondary to secondary message delivery, since there are acknowledgements between secondary stations.

20.2.2 Hardware

The schematic of the hardware is given in figure 20-7. The 8251A is used as an async communications controller, in support of the 8044. TxRDY and RxRDY on the 8251A are both tied to the two available external interrupts of the 8044 since the secondary station driver is totally interrupt driven. The 8044 buffers the data and some variables in a 2016 (2K x 8 static RAM). The 8254 programmable interval timer is employed as a programmable baud rate generator and system clock driver for the 8251A. The third output from the 8254 could be used as an external baud rate generator for the 8044. The 2732A shown in the diagram was not used since the software for both the primary and secondary stations used far less than the 4 Kbytes provided on the 8744. For the async interface, the standard RS-232

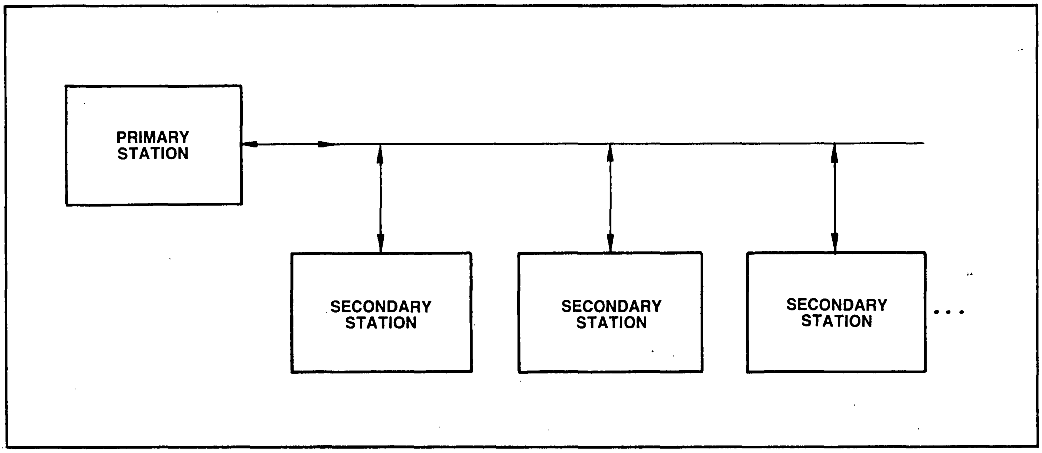


Figure 20-4. SDLC Multidrop Topology

mechanical and electrical interface was used. For the SDLC channel, a standard two wire three state RS-422 driver is used. A DIP switch connected to one of the available ports on the 8044 allows the baud rate, parity, and stop bits to be changed on the async interface. The primary station hardware does not use the USART, 8254, nor the RS-232 drivers.

20.2.3 SDLC Basic Repertoire

The SDLC commands and responses implemented in the data link include the SDLC Basic Repertoire as defined in the IBM SDLC General Information manual. Table 20-3 shows the commands and responses that the primary and the secondary station in this data link design recognize and send.

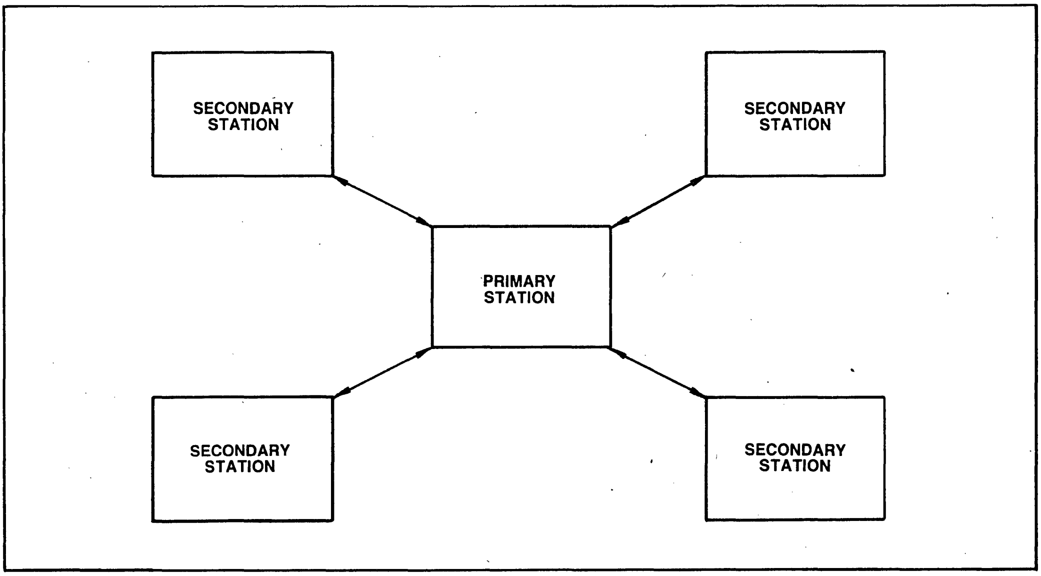


Figure 20-5. SDLC Logical Topology

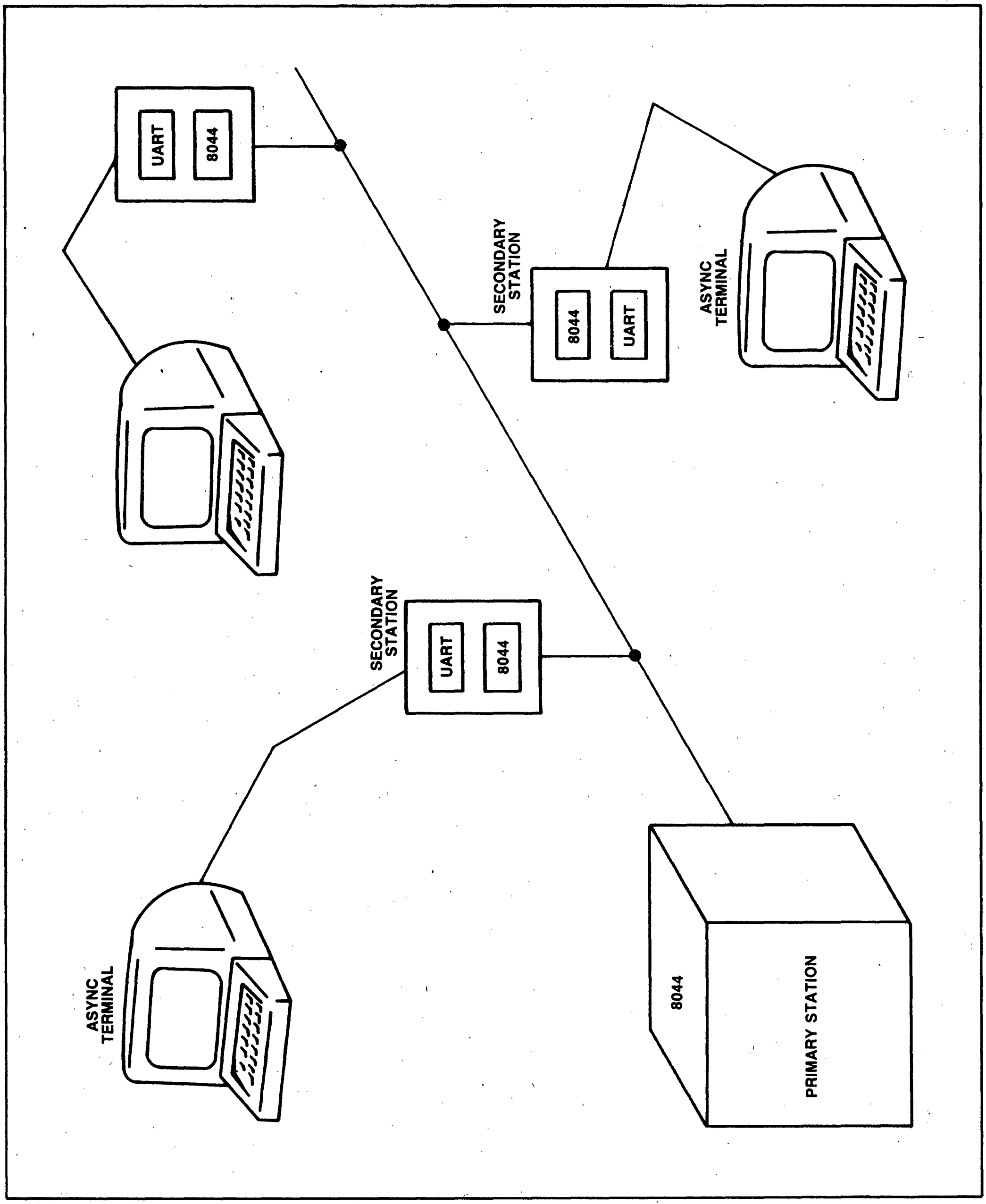


Figure 20-6. Block Diagram of the Data Link Application Example

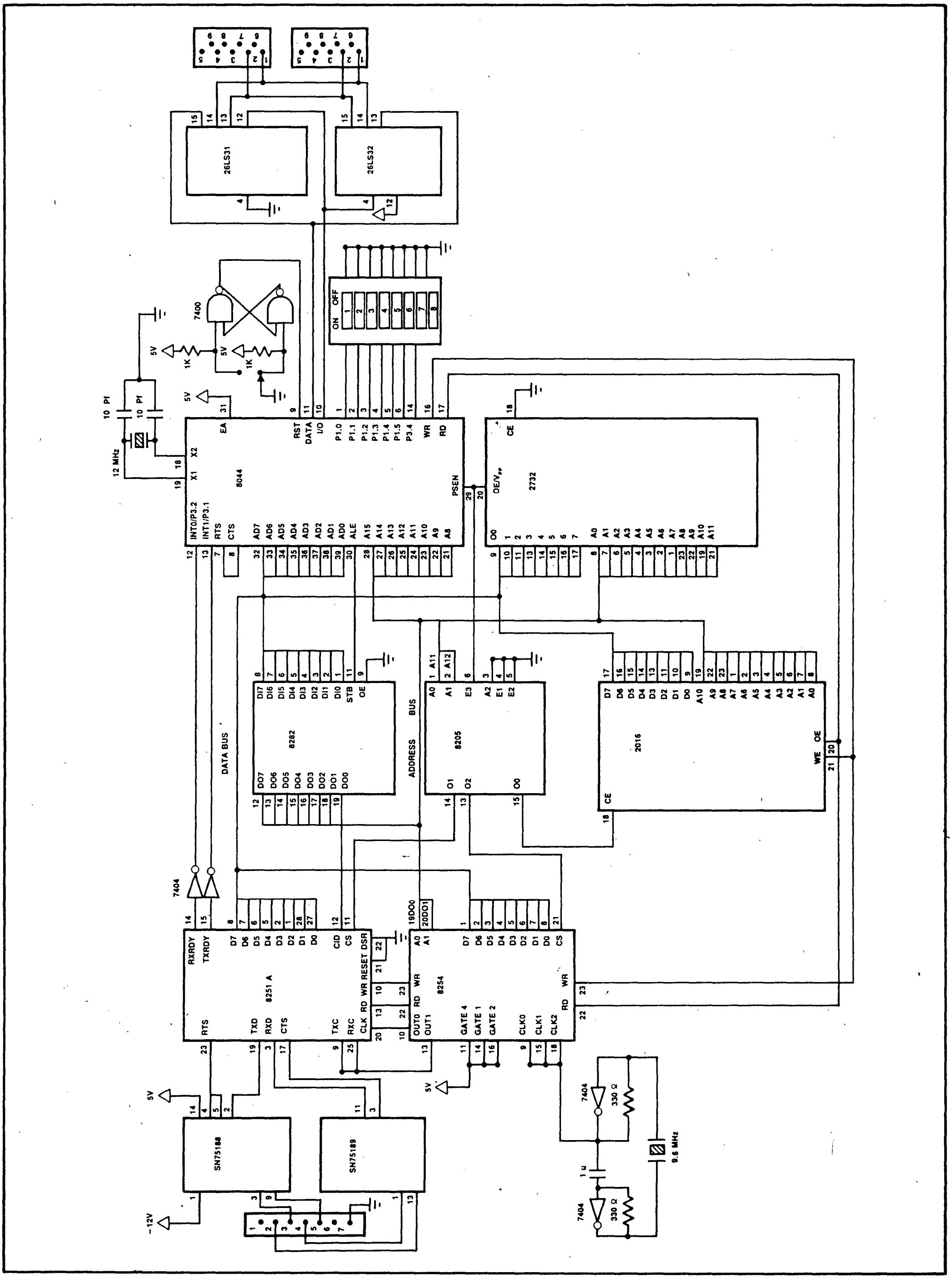


Figure 20-7. Schematic of Async/SDLC Secondary Station Protocol Converter

Table 20-3. Data Link Commands and Responses Implemented for This Design

PRIMARY STATION

	Responses Recognized	Commands Sent
Unnumbered	UA DM FRMR *RD	SNRM DISC
Supervisory	RR RNR	RR RNR
Information	I	I

SECONDARY STATION

	Commands Recognized	Responses Sent
Unnumbered	SNRM DISC *TEST	UA DM FRMR *RD *TEST
Supervisory	RR RNR REJ	RR RNR
Information	I	I

*not included in the SDLC Basic Repertoire

The term command specifically means all frames which the primary station transmits and the secondary stations receive. Response refers to frames which the secondary stations transmit and the primary station receives.

Number of Outstanding Frames

This particular data link design only allows one outstanding frame before it must receive an acknowledgement. Immediate acknowledgement allows the secondary station drivers to use the AUTO mode. In addition, one outstanding frame uses less memory for buffering, and the software becomes easier to manage.

20.2.4 Secondary Station Driver using AUTO Mode

The 8044 secondary station driver (SSD) was written as a general purpose SDLC driver. It was written to be linked to an application module. The application software implements the actual application in addition to interfacing to the SSD. The main application could be, a printer or plotter, a medical intrument, or a terminal. The SSD is independent of the main application, it just provides the SDLC communications. Existing 8051 applications could add high performance SDLC communications capability by linking the SSD to the existing software and providing additional software to be able to communicate with the SSD.

Data Link Interface and User Interface States

The SSD has two software interfaces: a data link interface and a user interface as show in Figure 20-8. The data link interface is the part of the software which controls the SDLC communications. It handles link access, command recognition/response, acknowledgements, and error recovery. The user interface provides four functions: OPEN, CLOSE, TRANSMIT, and SIU_RECV. These are the only four functions which the application software has to interface in order to communicate using SDLC. These four functions are common to many I/O drivers like floppy and hard disks, keyboard/CRT, and async communication drivers.

The data link and the user interface each have their own states. Each interface can only be in one state at any time. The SSD uses the states of these two interfaces to help synchronize the application module to the data link.

There are three states which the secondary station data link interface can be in: Logical Disconnect State (L_D_S). Frame Reject State (FRMR_S), and the Information Transfer State (I_T_S). The Logical Disconnect State is when a station is physically connected to the channel but either the primary or secondary have not agreed to enter the Information Transfer State. Both the primary and the secondary stations synchronize to enter into the Information Transfer State. Only when the secondary station is in the I_T_S is it able to transfer data or information to the primary. The Frame Reject State (FRMR_S) indicates that the secondary station has lost software synchronization with the primary or encountered some kind of error condition. When the secondary station is in the FRMR_S, the primary station must reset the secondary to resynchronize.

The user interface has two states, open or closed. In the closed state the user program does not want to communicate over the network. The communications channel is closed and not available for use. The secondary station tells the primary this by responding to all commands with DM. The primary continues to poll the secondary in case it wants to enter the I_T_S state. When the user program begins communication over the data link it goes into the open state. It does this by calling the OPEN procedure. When the user interface is in the open state it may transfer information to the primary.

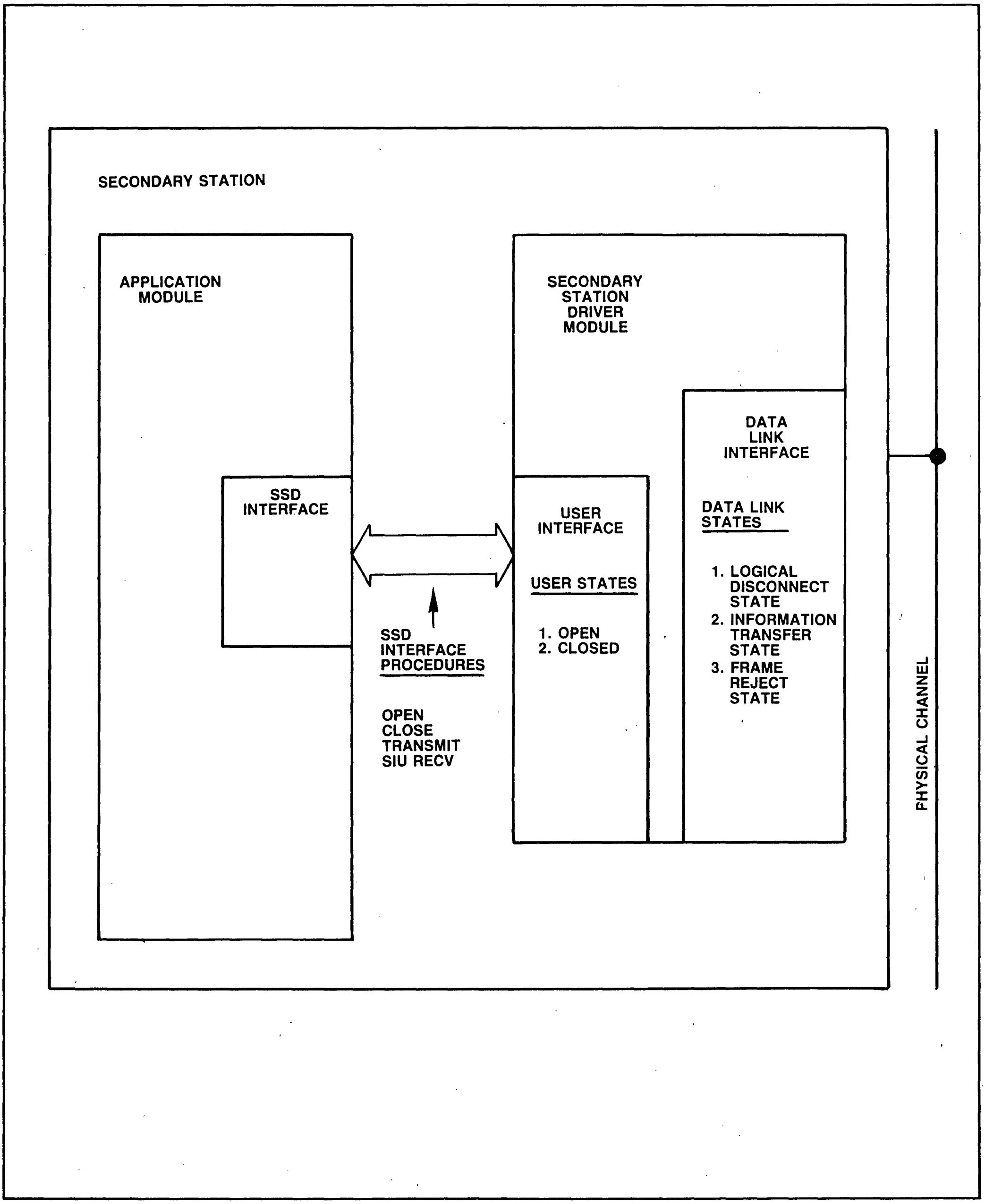


Figure 20-8. Secondary Station Software Modules

Secondary Stations Commands, Responses and State Transitions

Table 20-4 shows the commands which the secondary station recognizes and the responses it generates. The first row in table 20-4 displays commands the secondary station recognizes and each column shows the potential responses with respect to secondary station. For example, if the secondary is in the Logical Disconnect State it will only respond with DM, unless it receives a SNRM command and the user state is open. If this is the case, then the response will be UA and the secondary station will move into the I_T_S.

Figure 20-9 shows the state diagram of the secondary station. When power is first applied to the secondary station, it goes into the Logical Disconnect State. As mentioned above, the I_T_S is entered when the secondary station receives a SNRM command and the user state is open. The secondary responds with UA to let the primary know that it has accepted the SNRM and is entering the I_T_S. The I_T_S can go into either the L_D_S or the FRMR_S. The I_T_S goes into the L_D_S if the primary sends the secondary DISC. The secondary has to respond with UA, and then goes into the L_D_S. If the user interface changes from open to close state, then the secondary sends RD. This causes the primary to send a DISC.

The FRMR_S is entered when a secondary station is in the I_T_S and either one of the following conditions occurs.

— A command can not be recognized by the secondary station.

— There is a buffer overrun.

— The Nr that was received from the primary station is invalid.

The secondary station cannot leave the FRMR_S until it receives a SNRM or a DISC command.

Software description of the SSD

To aid in following the description of the software, the reader may either look at the flow charts which are given for each procedure, or read the PL/M-51 listing provided in Appendix A.

A block diagram of the software structure of the SSD is given in figure 20-10. A complete module is identified by the dotted box, and a procedure is identified by the solid box. Therefore the SIU_RECV procedure is not included in the SSD module, it exists in the application software. Two or more procedures connected by a solid line means the procedure above calls the procedure below. Transmit, Power_on_D, Close, and Open are all called by the application software. Procedures without any solid lines connected above are interrupt procedures. The only interrupt procedure in the SSD module is the SIU_INT.

The entire SSD module is interrupt driven. Its design allows the application program could handle real time events or just dedicate more CPU time to the application program. The SIU_INT is the only interrupt pro-

Table 20-4. Secondary Station Responses to Primary Station Commands

Data Link	Primary Station Commands					
States	I	RR	RNR	SNRM	DISC	TEST
Information transfer state	I RR RNR RD FRMR	I RR RNR RD FRMR	I RR RNR RD FRMR	RNR RD UA	UA	RD Test
Logical disconnect state	DM	DM	DM	DM UA	DM	DM
Frame reject state	FRMR	FRMR	FRMR	UA	UA	FRMR

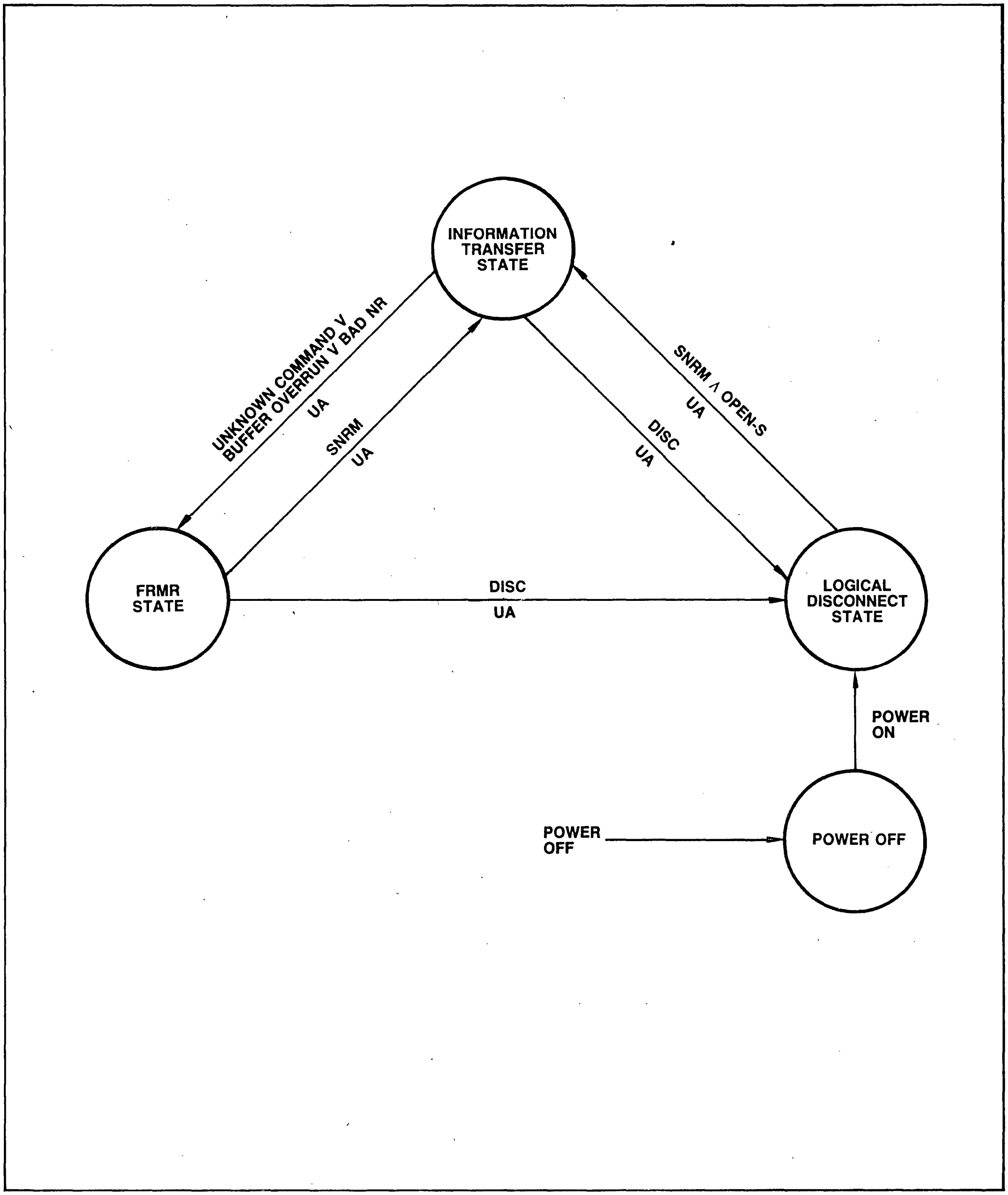


Figure 20-9. State Diagram of Secondary Station

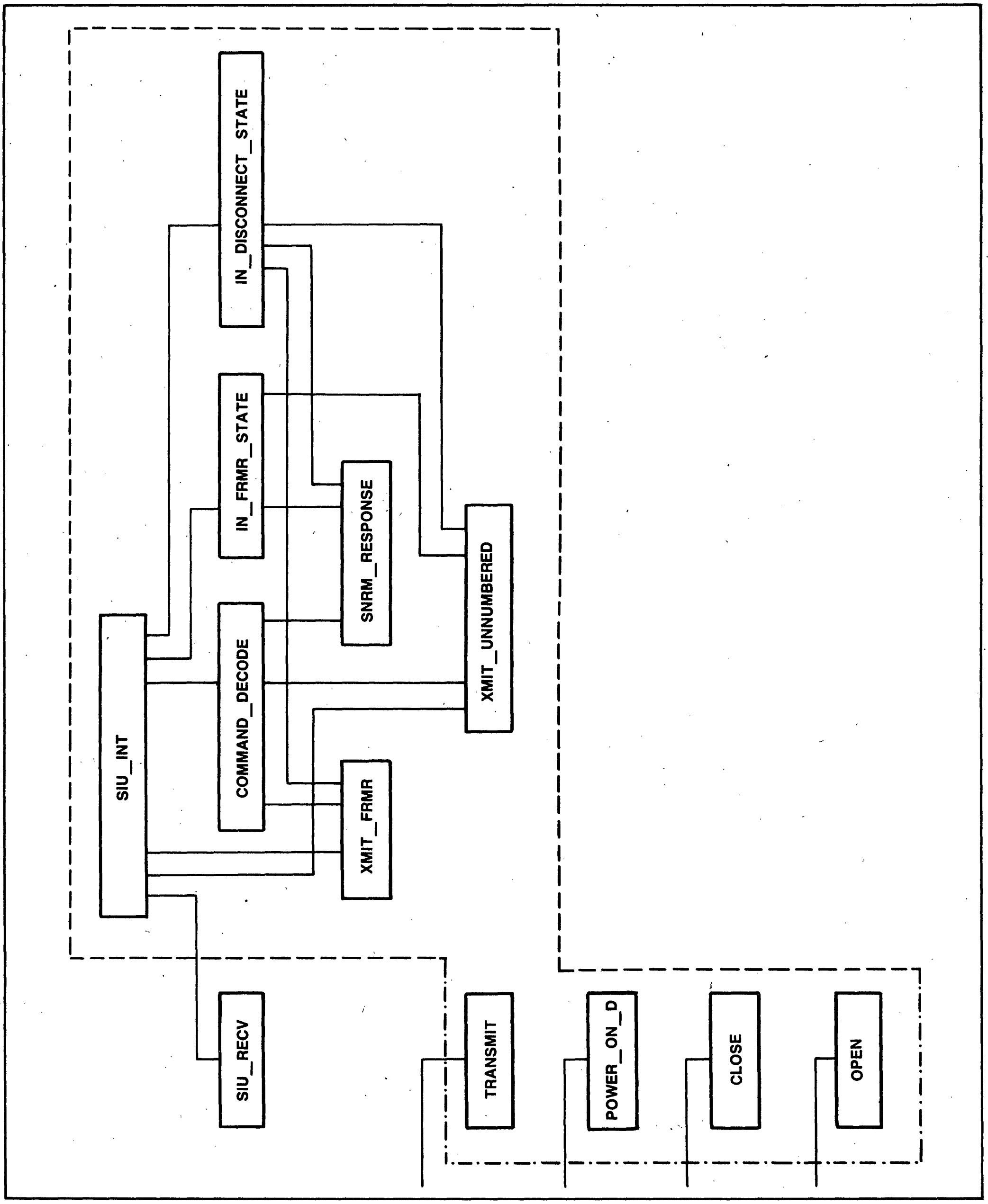


Figure 20-10. Secondary Station Driver

cedure in the SSD. It is automatically entered when an SIU interrupt occurs. This particular interrupt can be the lowest priority interrupt in the system.

SSD Initialization

Upon reset the application software is entered first. The application software initializies its own variables then calls Power__On__D which is the SSD's initialization routine. The SSD's initialization sets up the transmit and receive data buffer pointers (TBS and RBS), the receive buffer length (RBL), and loads the State variables. The STATION__STATE begins in the L__D__S state, and the USER__STATE begins in the closed state. Finally Power__On__D initializes XMIT__BUFFER__EMPTY which is a bit flag. This flag serves as a semaphore between the SSD and the application software to indicate the status of the on chip transmit buffer. The SSD does not set the station address. It is the application software's responsibility to do this. After initialization, the SSD is ready to respond to all of the primary stations commands. Each time a frame is received with a matching station address and a good CRC, the SIU__INT procedure is entered.

SIU__INT Procedure

The first thing the SIU__INT procedure clears the serial interrupt __bit (SI) in the STS register. If the SIU__INT procedure returns with this bit set, another SI interrupt will occur.

The SIU__INT procedure is branches three independent cases. The first case is entered if the STATION__STATE is not in the I__T__S. If this is true, then the SIU is not in the AUTO mode, and the CPU will have to respond to the primary on its own. (Remember that the AUTO mode is entered when the STATION__STATE enters into I__T__S.) If the STATION__STATE is in the I__T__S, then either the SIU has just left the AUTO mode, or is still in the AUTO mode. This is the second and third case respectively.

In the first case, if the STATION__STATE is not in the I__T__S, then it must be in either the L__D__S or the FRMR__S. In either case a separate procedure is called based on which state the station is in. The In__Disconnect__State procedure sends to the primary a DM response, unless it received a SNRM command and the USER__STATE equals open. In that case the SIU sends an UA and enters into the I__T__S. The In__FRMR__State procedure will send the primary the FRMR response unless it received either a DISC or an SNRM. If the primary's command was a DISC, then the secondary will send an UA and enter into the L__D__S. If the primary's command was a SNRM, then the secondary will send an UA, enter into the I__T__S, and clear NSNR register.

For the second case, if the STATION__STATE is in the I__T__S but the SIU left the AUTO mode, then the CPU must determine why the AUTO mode was exited, and generate a response to the primary. There are four reasons for the SIU to automatically leave the AUTO mode. The following is a list of these reasons, and the responses given by the SSD based on each reason.

1. The SIU has received a command field it does not recognize.

Response: If the CPU recognizes the command, it generates the appropriate response. If neither the SIU nor the CPU recognize the command, then a FRMR response is sent.

2. The SIU has received a Sequence Error Sent (SES=1 in NSNR register). Nr(P) ≠ Ns(S)+1, and Nr(P) ≠ Ns(S).

Response: Send FRMR.

3. A buffer overrun has occured. BOV=1 in STS register.

Response: Send FRMR.

4. An I frame with data was received while RPB=1.

Response: Go back into AUTO mode and send an AUTO mode response.

In addition to the above reasons, there is one condition where the CPU forces the SIU out of the AUTO mode. This is discussed in the SSD's User Interface Procedures section in the CLOSED procedure description.

Finally, case three is when the STATION__STATE is in the I__T__S and the AUTO mode. The CPU first looks at the TBF bit. If this bit is 0 then the interrupt may have been caused by a frame which was transmitted and acknowledged. Therefore the XMIT__BUFFER__EMPTY flag is set again indicating that the application software can transmit another frame.

The other reason this section of code could be entered is if a valid I frame was received. When a good I frame is received the RBE bit equals 0. This means that the receiver is disabled. If the primary were to poll the 8044 while RBE=0, it would time out since no response would given. Time outs reduce network throughput. To improve network performance, the CPU first sets RBP, then sets RBE. Now when the primary polls the 8044 an immediate RNR response is given. At this point the SSD calls the application software procedure SIU__RECV and passes the length of the data as a parameter. The SIU__RECV procedure reads the data out of the receive buffer then returns to the SSD module. Now that the receive information has been transfered, RBP can be cleared.

Command__Decode Procedure

The Command__Decode procedure is called from the SIU__INT procedure when the STATION__STATE = I__T__S and the SIU left the AUTO mode as a result

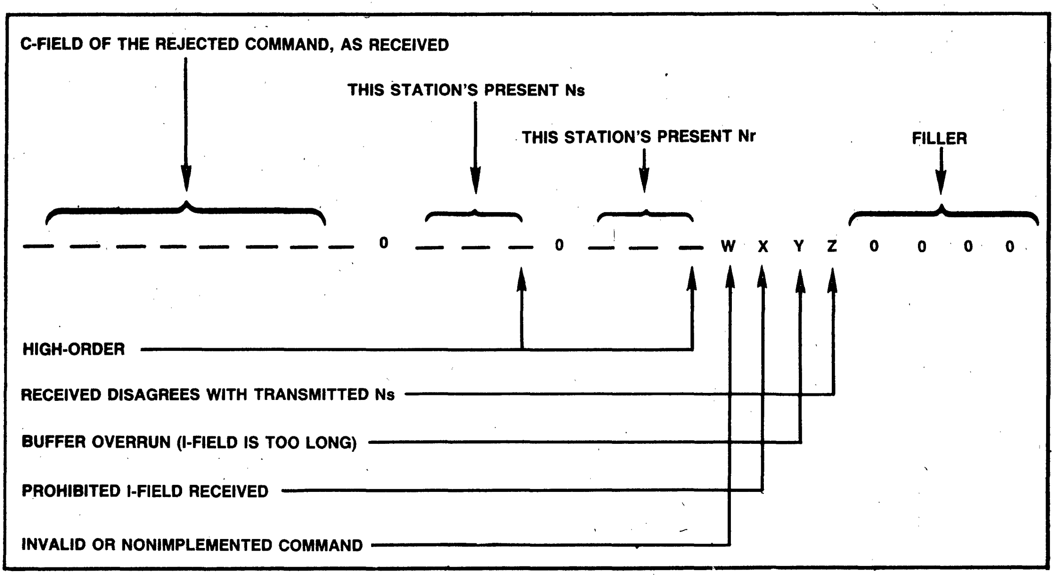

C-FIELD OF THE REJECTED COMMAND, AS RECEIVED
THIS STATION'S PRESENT Ns
THIS STATION'S PRESENT Nr
FILLER
0
0
W X Y Z 0 0 0 0
HIGH-ORDER
RECEIVED DISAGREES WITH TRANSMITTED Ns
BUFFER OVERRUN (I-FIELD IS TOO LONG)
PROHIBITED I-FIELD RECEIVED
INVALID OR NONIMPLEMENTED COMMAND

Figure 20.11. Information Field of the FRMR Response, as Transmitted

of not being able to recognize the receive control byte. Commands which the SIU AUTO mode does not recognize are handled here. The commands recognized in this procedure are: SNRM, DISC, and TEST. Any other command received will generate a Frame Reject with the nonimplemented command bit set in the third data byte of the FRMR frame. Any additional unnumbered frame commands which the secondary station is going to implement, should be implemented in this procedure.

If a SNRM is received the command__decode procedure calls the SNRM__Response procedure. The SNRM__ Response procedure sets the STATION__STATE = I__ T__S, clears the NSNR register and responds with an UA frame. If a DISC is received, the command__decode procedure sets the STATION__STATE = L__D__S, and responds with an UA frame. When a TEST frame is received, and there is no buffer overrun, the command__decode procedure responds with a TEST frame retransmitting the same data it received. However if a TEST frame is received and there is a buffer overrun, then a TEST frame will be sent without any data, instead of a FRMR with the buffer overrun bit set.

Frame Reject Procedures

There are two procedures which handle the FRMR state: XMIT__FRMR and IN__FRMR__STATE. XMIT__ FRMR is entered when the secondary station first goes into the FRMR state. The frame reject response frame contains the FRMR response in the command field plus three additional data bytes in the I field. Figure 20-11 displays the format for the three data byte in the I field of a FRMR response. The XMIT__FRMR procedure sets up the Frame Reject response frame based on the parameter REASON which is passed to it. Each place in the SSD code that calls the XMIT__FRMR procedure, passes the REASON that this procedure was called, which in turn is communicated to the primary station. The XMIT__FRMR procedure uses three bytes of internal RAM which it initializes for the correct response. The TBS and TBL registers are then changed to point to the FRMR buffer so that when a response is sent these three bytes will be included in the I field.

The IN__FRMR__STATE procedure is called by the SIU__INT procedure when the STATION__STATE already is in the FRMR state and a response is required. The IN__FRMR__STATE procedure will only allow two commands to remove the secondary station from the FRMR state: SNRM and DISC. Any other command which is received while in the FRMR state will result a FRMR response frame.

XMIT__UNNUMBERED Procedure

This is a general purpose transmit procedure, used only in the FLEXIBLE mode, which sends unnumbered responses to the primary. It accepts the control byte as a parameter, and also expects the TBL register to be set before the procedure is called. This procedure waits until the frame has been transmitted before returning. If this procedure returned before the transmit interrupt was generated, the SIU__INT routine would be entered. The SIU__INT routine would not be able to distinguish this condition.

SSD's User Interface Procedures -- OPEN, CLOSE, TRANSMIT, SIU_RECV -- are discussed in the following section.

The OPEN procedure is the simplest of all, it changes the USER_STATE to OPEN_S then returns. This lets the SSD know that the user wants to open the channel for communications. When the SSD receives a SNRM command, it checks the USER_STATE. If the USER_STATE is open, then the SSD will respond with an UA, and the STATION_STATE enters the I_T_S.

The CLOSE procedure is also simple, it changes the USER_STATE to CLOSED_S and sets the AM bit to 0. Note that when the CPU sets the AM bit to 0 it puts the SIU out of the AUTO mode. This event is asynchronous to the events on the network. As a result an I frame can be lost. This is what can happen.

1. AM is set to 0 by the CLOSE Procedure.
2. An I frame is received and a SI interrupt occurs.
3. The SIU_INT procedure enters case 2. (STATION_STATE = I_T_S, and AM = 0)
4. Case 2 detects that the USER_STATE = CLOSED_S, sends a RD response and ignores the fact that an I frame was received.

Therefore it is advised to never call the CLOSE procedure or take the SIU out of the AUTO mode when it is receiving I frames or an I frame will be lost.

For both the TRANSMIT and SIU_RECV procedures, it is the application software's job to put data into the transmit buffer, and take data out of the receive buffer. The SSD does not transfer data in or out of its transmit or receive buffers because it does not know what kind of buffering the application software is implementing. What the SSD does do is notify the application software when the transmit buffer is empty, XMIT_BUFFER_EMPTY = 1, and when the receive buffer is full.

One of the functions that the SSD performs to synchronize the application software to the SDLC data link. However some of the synchronization must also be done by the application software. Remember that the SSD does not want to hang up the application software waiting for some event to occur on the SDLC data link, therefore the SSD always returns to the application software as soon as possible.

For example, when the application software calls the OPEN procedure, the SSD returns immediately. The application software thinks that the SDLC channel is now open and it can transmit. This is not the case. For the channel to be open, the SSD must receive a SNRM from the primary and respond with a UA. However, the SSD does not want to hang up the application software waiting for a SNRM from the primary before returning from the OPEN procedure. When the TRANSMIT procedure is called, the SSD expects the STATION_STATE to be in the I_T_S. If it isn't, the SSD refuses to transmit the data. The TRANSMIT procedure first checks to see if the USER_STATE is open, if not the USER_STATE_CLOSED parameter is passed back to the application module. The next thing TRANSMIT checks is the STATION_STATE. If this is not open, then TRANSMIT passes back LINK_DISCONNECTED. This means that the USER_STATE is open, but the SSD hasn't received a SNRM commmand from the primary yet. Therefore, the application software should wait awhile and try again. Based on network performance, one knows the maximum amount of time it will take for a station to be polled. If the application software waits this length of time and tries again but still gets a LINK_DISCONNECTED parameter passed back, higher level recovery must be implemented.

Before loading the transmit buffer and calling the TRANSMIT procedure, the application software must check to see that XMIT_BUFFER_EMPTY = 1. This flag tells the application software that it can write new data into the transmit buffer and call the TRANSMIT procedure. After the application software has verified that XMIT_BUFFER_EMPTY = 1, it fills the transmit buffer with the data and calls the TRANSMIT procedure passing the length of the buffer as a parameter. The TRANSMIT procedure checks for three reasons why it might not be able to transmit the frame. If any of these three reasons are true, the TRANSMIT procedure returns a parameter explaining why it couldn't send the frame. If the application software receives one of these responses, it must rectify the problem and try again. Assuming these three conditions are false, then the SSD clears XMIT_BUFFER_EMPTY, attempts to send the data and returns the parameter DATA_TRANSMITTED. XMIT_BUFFER_EMPTY will not be set to 1 again until the data has been transmitted and acknowledged.

The SIU_RECV procedure must be incorporated into the application software module. When a vaild I frame is received by the SIU, it calls the SIU_RECV procedure and passes the length of the received data as a parameter. The SIU_RECV procedure must remove all of the data from the receive buffer before returning to the SIU_INT procedure.

Linking up to the SSD

Figure 20-12 shows necessary parts to include in a PL/M-51 application program that will be linked to the SSD module. RL51 is used to link and locate the SSD and application modules. The command line used to do this is:

```
RL51 SSD.obj,filename.obj,PLM51.LIB TO filename
& RAMSIZE(192)

$registerbank(0)
user$mod: do;
$include (reg44.dcl)
declare
  lit                      literally     'literally',
  buffer__length           lit             '60',
  siu__xmit__buffer
  (buffer__length)         byte        external idata,
  siu__recv__buffer
  (buffer__length)         byte            external,
  xmit__buffer__empty      bit             external;

/* external procedures */

power__on__d: procedure       external;
end power__on__d;

close: procedure              external     using 1;
end close;

open: procedure               external     using 1;
end open;

transmit: procedure
(xmit__buffer__length)     byte            external;
    declare     xmit__buffer__length       byte;
end transmit;

/* local procedures */

siu__recv: procedure (length) public       using 1;
    declare     length      byte,
                     .
                     .
                     .
end siu__recv;
```

Figure 20-12. Applications Module Link Information

PL/M-51 and Register Banks

The 8044 has four register banks. PL/M-51 assumes that an interrupt procedure never uses the same bank as the procedure it interrupts. The USING attribute of a procedure, or the $REGISTERBANK control, can be used to ensure that.

The SSD module uses the $REGISTERBANK(1) attribute. Some procedures are modified with the USING attribute based on the register bank level of the calling procedure.

20.2.5 APPLICATION MODULE; Async to SDLC protocol converter

One of the purposes of this application module is to demonstrate how to interface software to the SSD. Another purpose is to implement and test a pratical application. This application software performs I/O with an async terminal through a USART, buffers data, and also performs I/O with the SSD. In addition, it allows the user on the async terminal to: set the station address, set the destination address, and go online and offline. Setting the station address sets the byte in the STAD register. The destination address is the first byte in the I field. Going online or offline results in either calling the OPEN or CLOSE procedure respectively.

After the secondary station powers up, it enters the 'terminal mode', which accepts data from the terminal. However, before any data is sent, the user must configure the station. The station address and destination address must be set, and the station must be placed online. To configure the station the ESC character is entered at the terminal which puts the protocol converter into the 'configure mode'. Figure 20-13 shows the menu which appears on the terminal screen.

```
   ( /) 8044 Secondary Station
     /


1 - Set the Station Address
2 - Set the Destination Address
3 - Go Online
4 - Go Offline
5 - Return to terminal mode

   Enter option _
```

Figure 20-13. Menu for the Protocol Converter

In the terminal mode data is buffered up in the secondary station. A Line Feed character 'LF' tells the secondary station to send an I frame. If more than 60 bytes are buffered in the secondary station when a 'LF' is received, the applications software packetizes the data into 60 bytes or less per frame. If a LF is entered when the station is offline, an error message comes on the screen which says 'Unable to Get Online'.

The secondary station also does error checking on the async interface for Parity, Framing Error, and Overrun Error. If one of these errors are detected, an error message is displayed on the terminal screen.

Buffering

There are two separate buffers in the application module: a transmit buffer and a receive buffer. The transmit buffer receives data from the USART, and sends data to the SSD. The receive buffer receives data from the SSD, and transmits data to the USART. Each buffer is a 256 byte software FIFO. If the tranmsit FIFO becomes full and no 'LF' character is received, the secondary station automatically begins sending the data. In addition, the application module will shut off the terminal's transmitter using CTS until the FIFO has been partially emptied. A block diagram of the buffering for the protocol converter is given in Figure 20-14.

Application Module Software

A block diagram of the application module software is given in Figure 20-15. There are three interrupt

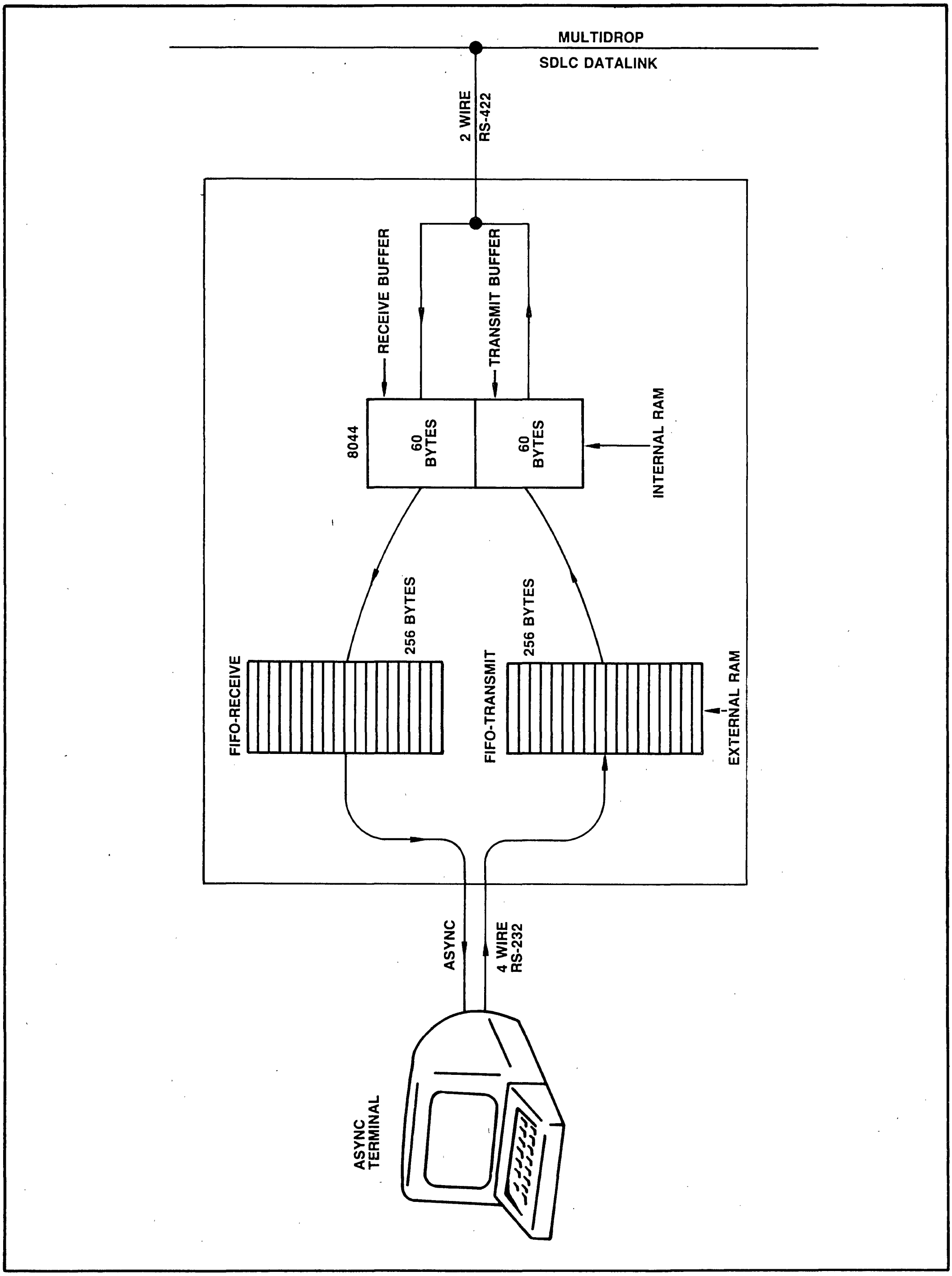


Figure 20-14. Block Diagram of Secondary Station Protocol Converter Illustrating Buffering

routines in this module: USART_RECV_INT, USART_XMIT_INT, and TIMER_0_INT. The first two are for servicing the USART. TIMER_0_INT is used if the TRANSMIT procedure in the SSD is called and does not return with the DATA_TRANSMITTED parameter. TIMER_0_INT employs Timer 0 to wait a finite amount of time before tring to transmit again. The highest priority interrupt is USART_RECV_INT. The main program and all the procedures it calls use register bank 0, USART_XMIT_INT and TIMER_0_INT and FIFO_R_OUT use bank 1, while USART_RECV_INT and all the procedures it calls use register bank 2.

Power_On Procedure

The Power_On procedure initializes all of the chips in the system including the 8044. The 8044 is initialized to use the on-chip DPLL with NRZI coding, PreFrame Sync, and Timer 1 auto reload at a baud rate of 62.5 Kbps. The 8254 and the 8251A are initialized next based on the DIP switch values attached to port 1 on the 8044. Variables and pointers are initialized, then the SSD's Power-Up Procedure, Power_On_D, is called. Finally the interrupt system is enabled and the main program is entered.

Main Program

The main program is a simple loop which waits for a frame transmit command. A frame transmit command is indicated when the variable SEND_DATA is greater than 0. The value of SEND_DATA equals the number of 'LF' characters in the transmit FIFO, hence it also indicates the number of frames pending transmission. Each time a frame is sent, SEND_DATA is decremented by one. Thus when SEND_DATA is greater than 0, the main program falls down into the next loop which polls the XMIT_BUFFER_EMPTY bit. When XMIT_BUFFER_EMPTY equals 1, the SIU_XMIT_BUFFER can be loaded. The first byte in the buffer is loaded with the destination address while the rest of the buffer is loaded with the data. Bytes are removed from the transmit FIFO and placed into the SIU_XMIT_BUFFER until one of three things happen: 1. a 'LF' character is read out of the FIFO, 2. the number of bytes loaded equals the size of the SIU_XMIT_BUFFER, or 3. the transmit FIFO is empty.

After the SIU_XMIT_BUFFER is filled, the SSD TRANSMIT procedure is called and the results from the procedure are checked. Any result other than DATA_TRANSMITTED will result in several retries within a finite amount of time. If all the retries fail then the LINK_DISC procedure is called which sends a message to the terminal, 'Unable to Get Online'.

USART_RECV_INT Procedure

When the 8251A receives a character,the RxRDY pin on the 8251A is activated,and this interrupt procedure is entered. The routine reads the USART status register to determine if there are any errors in the character received. If there are, the character is discarded and the ERROR procedure is called which prints the type of error on the screen. If there are no errors, the received character is checked to see if it's an ESC. If it is an ESC, the MENU procedure is called which allows the user to change the configuration. If neither one of these two conditions exits the received character is inserted into the transmit FIFO. The received character may or may not be echoed back to the terminal based on the dip switch settings.

Transmit FIFO

The transmit FIFO consists of two procedures: FIFO_T_IN and FIFO_T_OUT. FIFO_T_IN inserts a character into the FIFO,and FIFO_T_OUT removes a character from the FIFO. The FIFO itself is an array of 256 bytes called FIFO_T. There are two pointers used as indexes in the array to address the characters: IN_PTR_T and OUT_PTR_T. IN_PTR_T points to the location in the array which willstore the next byte of data inserted. OUT_PTR_T points to the next byte of data removed from the array. Both IN_PTR_T and OUT_PTR_T are declared as bytes. The FIFO_T_ IN procedure receives a character from the USART_RECV_INT procedure and stores it in the array location pointed to by IN_PTR_T, then IN_PTR_T is incremented. Similarly, when FIFO_T_OUT is called by the main program, to load the SIU_XMIT_BUFFER, the byte in the array pointed to by OUT_PTR_T is read, then OUT_PTR_T is incremented. Since IN_PTR_T and OUT_PTR_T are always incremented, they must be able to roll over when they hit the top of the 256 byte address space. This is done automatically by having both IN_PTR_T and OUT_PTR_T declared as bytes. Each character inserted into the transmit FIFO is tested to see if it's a LF. If it is a LF, the variable SEND_DATA is incremented which lets the main program know that it is time to send an I frame. Similarily each character removed from the FIFO is tested. SEND_DATA is decremented for every LF character removed from the FIFO.

IN_PTR_T and OUT_PTR_T are also used to indicate how many bytes are in the FIFO, and whether it is full or empty. When a character is placed into the FIFO and IN_PTR_T is incremented, the FIFO is full if IN_PTR_T equals OUT_PTR_T. When a character is read from the FIFO and OUT_PTR_T is incremented, the FIFO is empty if OUT_PTR_T equals IN_PTR_T. If the FIFO is neither full nor empty, then it is in use. A byte called BUFFER_STATUS_T is used to indicate one of these three conditions. The application module uses the buffer status information to control the flow of data into and out of the FIFO. When the transmit FIFO is empty, the main program must stop loading bytes into the SIU_

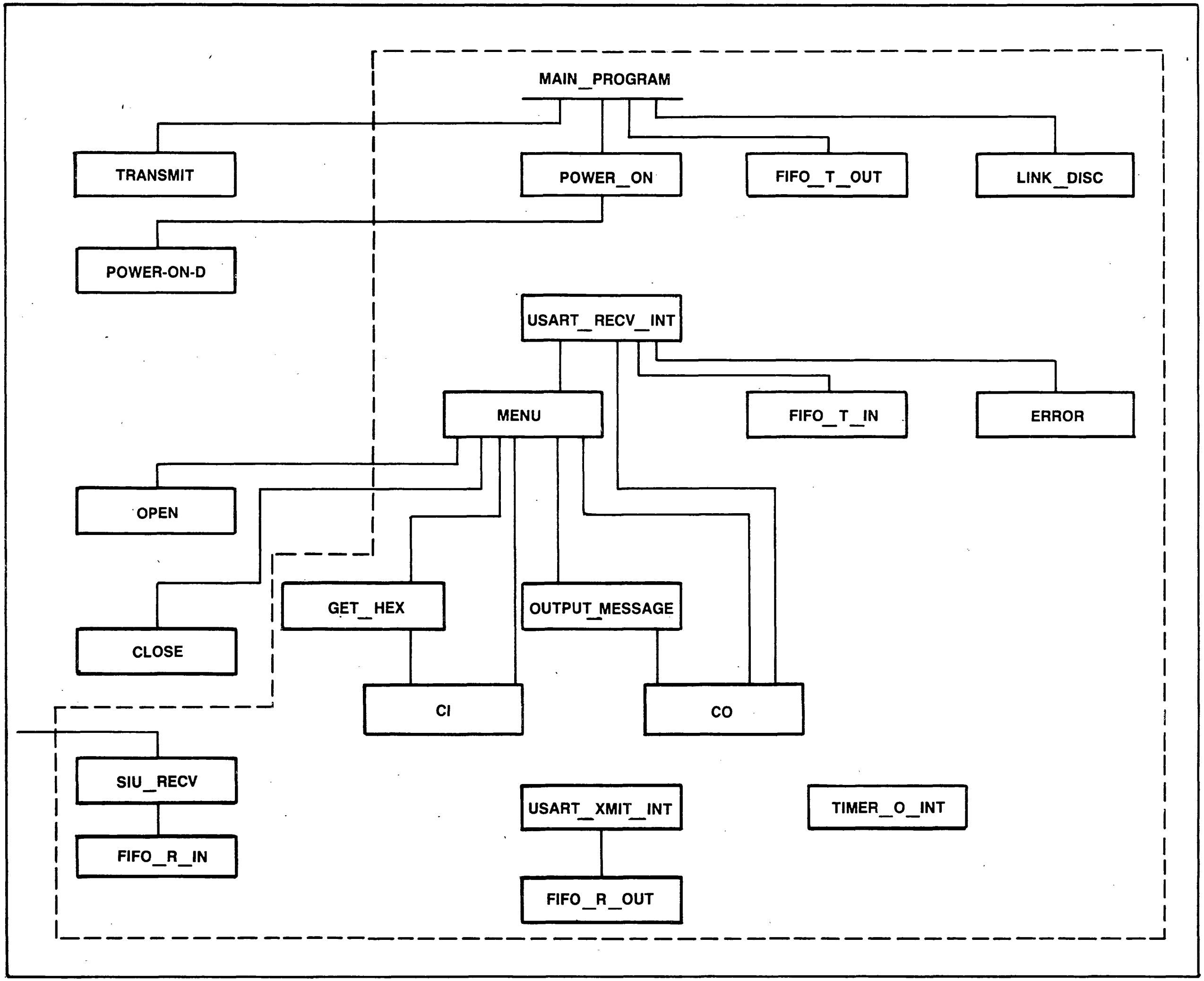


Figure 20-15. Block Diagram of User Software

XMIT__BUFFER. Just before the FIFO is full, the async input must be shut off using CTS. Also if the FIFO is full and SEND__DATA=0, then SEND__DATA must be incremented to automatically send the data without a LF.

Receive FIFO

The receive FIFO operates in a similiar fashion as the transmit FIFO does. Data is inserted into the receive FIFO from the SIU__RECV procedure. The SIU__RECV procedure is called by the SIU__INT procedure when a valid I frame is received. The SIU__RECV procedure mearly polls the receive FIFO status to see if it's full before transfering each byte from the SIU__RECV__BUFFER into the receive FIFO. If the receive FIFO is full, the SIU__RECV procedure remains polling the FIFO status until it can insert the rest of the data. In the meantime, the SIU AUTO mode is responding to all polls from the primary with a RNR supervisory frame. The USART__XMIT__INT interrupt procedure removes data from the receive FIFO and transmits it to the terminal. The USART transmit interrupt remains enabled while the receive FIFO has data in it. When the receive FIFO becomes empty, the USART transmit interrupt is disabled.

20.2.6 PRIMARY STATION

The primary station is responsible for controlling the data link. It issues commands to the secondary stations and receives responses from them. The primary station controls link access, link level error recovery, and the flow of information. Secondaries can only transmit when polled by the primary.

Most primary stations are either micro/minicomputers, or front end processors to a mainframe computer. The example primary station in this design is standalone. It is possible for the 8044 to be used as an intelligent front end processor for a microprocessor, implementing the primary station functions. This latter type of design would extensively off-load link control functions for the microprocessor. The code listed in this paper can be used as the basis for this primary station design. Additional software is required to interface to the microprocessor. A hardware design example for interfacing the 8044 to a microprocessor can be found in the applications section of this handbook.

The primary station must know the addresses of all the stations which will be on the network. The software for this primary needs to know this before it is compiled, however a more flexible system would down load these parameters.

From the listing of the software it can be seen that the variable NUMBER__OF__STATIONS is a literal declaration, which is 2 in this design example. There were three stations tested on this data link, two secondaries and one primary. Following the NUMBER__OF__STATIONS declaration is a table, loaded into the object code file at compile time, which lists the addresses of each secondary station on the network.

Remote Station Database

The primary station keeps a record of each secondary station on the network. This is called the Remote Station Database (RSD). The RSD in this software is an array of structures, which can be found in the listing and also in Figure 20-16. Each RSD stores the necessary information about that secondary station.

To add additional secondary stations to the network, one simply adjusts the NUMBER__OF__STATIONS declaration, and adds the additional addresses to the SECONDARY__ADDRESSES table. The number of RSDs is automatically allocated at compile time, and the primary automatically polls each station whose address is in the SECONDARY__ADDRESSES table.

Memory for the RSDs resides in external RAM. Based on memory requirements for each RSD, the maximum number of stations can be easily buffered in external RAM. (254 secondary stations is the maximum number SDLC will address on the data link; i.e. 8 bit address, FF H is the broadcast address, and 0 is the nul address. Each RSD uses 70 bytes of RAM. 70 x 254 = 17,780.)

The station state, in the RSD structure, maintain the status of the secondary. If this byte indicates that the secondary is in the DISCONNECT__S, then the primary tries to put the station in the I__T__S by sending a SNRM. If the response is an UA then the station state changes into the I__T__S. Any other frame received results in the station state remaining in the DISCONNECT__S. When the RSD indicates that the station state is in the I__T__S, the primary will send either a I, RR, or RNR command, depending on the local and remote buffer status. When the station state equals GO__TO__DISC the primary will send a DISC command. If the response is an UA frame, the station state will change to DISCONNECT__S, else the station state will remain in GO__TO__DISC. The station state is set to GO__TO__DISC when one of the following responses occur:

1. A receive buffer overrun in the primary.
2. An I frame is received and Nr(P) ≠Ns(S).
3. An I frame or a Supervisory frame is received and Ns(P) + 1 ≠ Nr(S) and Ns(P) ≠Nr(S).
4. A FRMR response is received.
5. An RD response is received.
6. An unknown response is received.

The send count (Ns) and receive count (Nr) are also maintained in the RSD. Each time an I frame is sent by the primary and acknowledged by the secondary, Ns is incremented. Nr is incremented each time a valid I frame is received. BUFFER__STATUS indicates the status of the secondary stations buffer. If a RR response is received, BUFFER__STATUS is set to BUFFER__

READY. If a RNR response is received, BUFFER__STATUS is set to BUFFER__NOT__READY.

Buffering

The buffering for the primary station is as follows: within each RSD is a 64 byte array buffer which is initially empty. When the primary receives an I frame, it looks for a match between the first byte of the I frame and the addresses of the secondaries on the network. If a match exits, the primary places the data in the RSD buffer of the destination station. The INFO__LENGTH in the RSD indicates how many bytes are in the buffer. If INFO__LENGTH equals 0,then the buffer is empty. The primary can buffer only one I frame per station. If a second I frame is received while the addressed secondary's RSD buffer is full, the primary cannot receive any more I frames. At this point the primary continues to poll the secondaries using RNR supervisory frame.

Primary Station Software

A block diagram of the primary station software is shown in Figure 20-17 The primary station software consists of a main program, one interrupt routine, and several procedures. The POWER__ON procedure begins by initializing the SIU's DMA and enabling the receiver. Then each RSD is initialized. The DPLL and the timers are set, and finally the TIMER 0 interrupt is enabled.

The main program consists of an iterative do loop within a do forever loop. The iterative do loop polls each secondary station once through the do loop. The variable STATION__NUMBER is the counter for the iterative do statement which is also used as an index to the array of RSD structures. The primary station issues one command and receives one response from every secondary station each time through the loop. The first statement in the loop loads the secondary station address, indexed by STATION__NUMBER into the array of the RSD structures. Now when the primary sends a command, it will have the secondary's address in the address field of the frame. The automatic address recognition feature is used by the primary to recognize the response from the secondary.

Next the main program determines the secondary stations state. Based on this state, the primary knows what command to send. If the station is in the DISCONNECT__S, the primary calls the SNRM__P

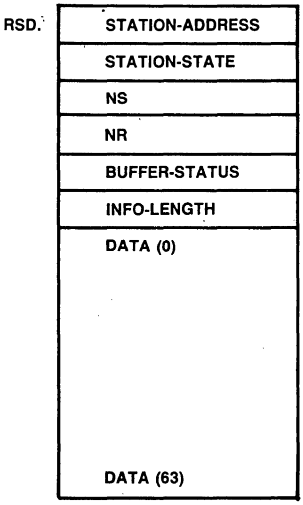

RSD:
STATION-ADDRESS
STATION-STATE
NS
NR
BUFFER-STATUS
INFO-LENGTH
DATA (0)
DATA (63)

Figure 20-16. Remote Station Database Structure

procedure to try and put the secondary in the I_T_S. If the station state is in the GO_TO_DISC state, the DISC_P is called to try and put the secondary in the L_D_S. If the secondary is in neither one of the above two states, then it is in the I_T_S. When the secondary is in the I_T_S, the primary could send one of three commands: I, RR, or RNR. If the RSD's buffer has data in it, indicated by INFO_LENGTH being greater than zero, and the secondary's BUFFER_STATUS equals BUFFER_READY, then an I frame will be sent. Else if RPB=0, a RR supervisory frame will be sent. If neither one of these cases is true, then an RNR will be sent. The last statement in the main program checks the RPB bit. If set to one, the BUFFER_TRANSFER procedure is called, which transfers the data from the SIU receive buffer to the appropriate RSD buffer.

Receive Time Out

Each time a frame is transmitted, the primary sets a receive time out timer; Timer 0. If a response is not received within a certain time, the primary returns to the main program and continues polling the rest of the stations. The minimum length of time the primary should wait for a response can be calculated as the sum of the following parameters.

1. propagation time to the secondary station
2. clear-to-send time at the secondary station's DCE
3. appropriate time for secondary station processing
4. propagation time from the secondary station
5. maximum frame length time

The clear-to-send time and the propagation time are negligible for a local network at low bit rates. However, the turnaround time and the maximum frame length time are significant factors. Using the 8044 secondaries in the AUTO mode minimizes turnaround time. The maximum frame length time comes from the fact the 8044 does not generate an interrupt from a received frame until it has been completely received, and the CRC is verified as correct. This means that the time-out is bit rate dependent.

Ns and Nr check Procedures

Each time an I frame or supervisory frame is received, the Nr field in the control byte must be checked. Since this data link only allows one outstanding frame, a valid Nr would satisfy either one of two equations; Ns(P) + 1 = Nr(S) the I frame previously sent by the primary is acknowledged, Ns(P) = Nr(S) the I frame previously sent is not acknowledged. If either one of these two cases is true, the CHECK_NR procedure returns a parameter of TRUE; otherwise a FALSE parameter is returned. If an acknowledgement is received, the Ns byte in the RSD structure is incremented, and the Information buffer may be cleared. Otherwise the information buffer remains full.

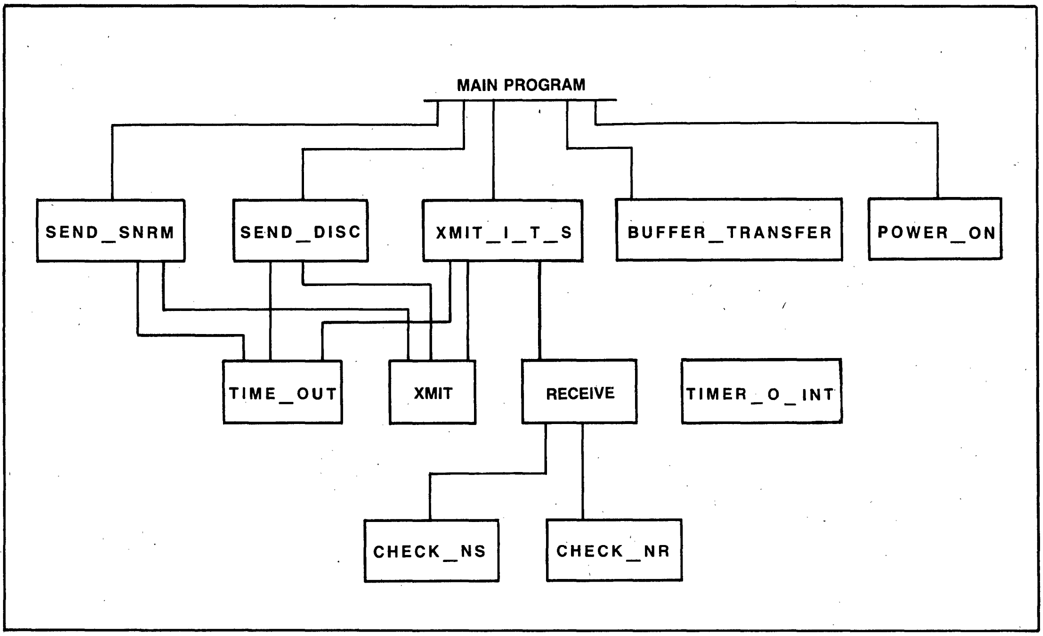


Figure 20.17. Block Diagram of Primary Station Software Structure

When an I frame is received, the Ns field has to be checked also. If Nr(P) = Ns(S), then the procedure returns TRUE, otherwise a FALSE is returned.

Receive Procedure

The receive procedure is called when a supervisory or information frame is sent, and a response is received before the time-out period. The RECEIVE procedure can be broken down into three parts. The first part is entered if an I frame is received. When an I frame is received, Ns, Nr and buffer overrun are checked. If there is a buffer overrun, or there is an error in either Ns or Nr, then the station state is set to GO_TO_DISC. Otherwise Nr in the RSD is incremented, the receive field length is saved, and the RPB bit is set. By incrementing the Nr field, the I frame just received is acknowledged the next time the primary polls the secondary with an I frame or a supervisory frame. Setting RBP protects the received data, and also tells the main program that there is data to transfer to one of the RSD buffers.

If a supervisory frame is received, the Nr field is checked. If a FALSE is returned, then the station state is set to GO_TO_DISC. If the supervisory frame received was an RNR, buffer status is set to not ready. If the response is not an I frame, nor a supervisory frame, then it must be an Unnumbered frame.

The only Unnumbered frames the primary recognizes are UA, DM, and FRMR. In any event, the station state is set to GO_TO_DISC. However if the frame received is a FRMR, Nr in the second data byte of the I field is checked to see if the secondary acknowledged an I frame received before it went into the FRMR state. If this is not done and the secondary acknowledged an I frame which the primary did not recognize, the primary transmits, the I frame when the secondary returns to the I_T_S. In this case, the secondary would receive duplicate I frames.

APPENDIX A
8044 SOFTWARE FLOWCHARTS

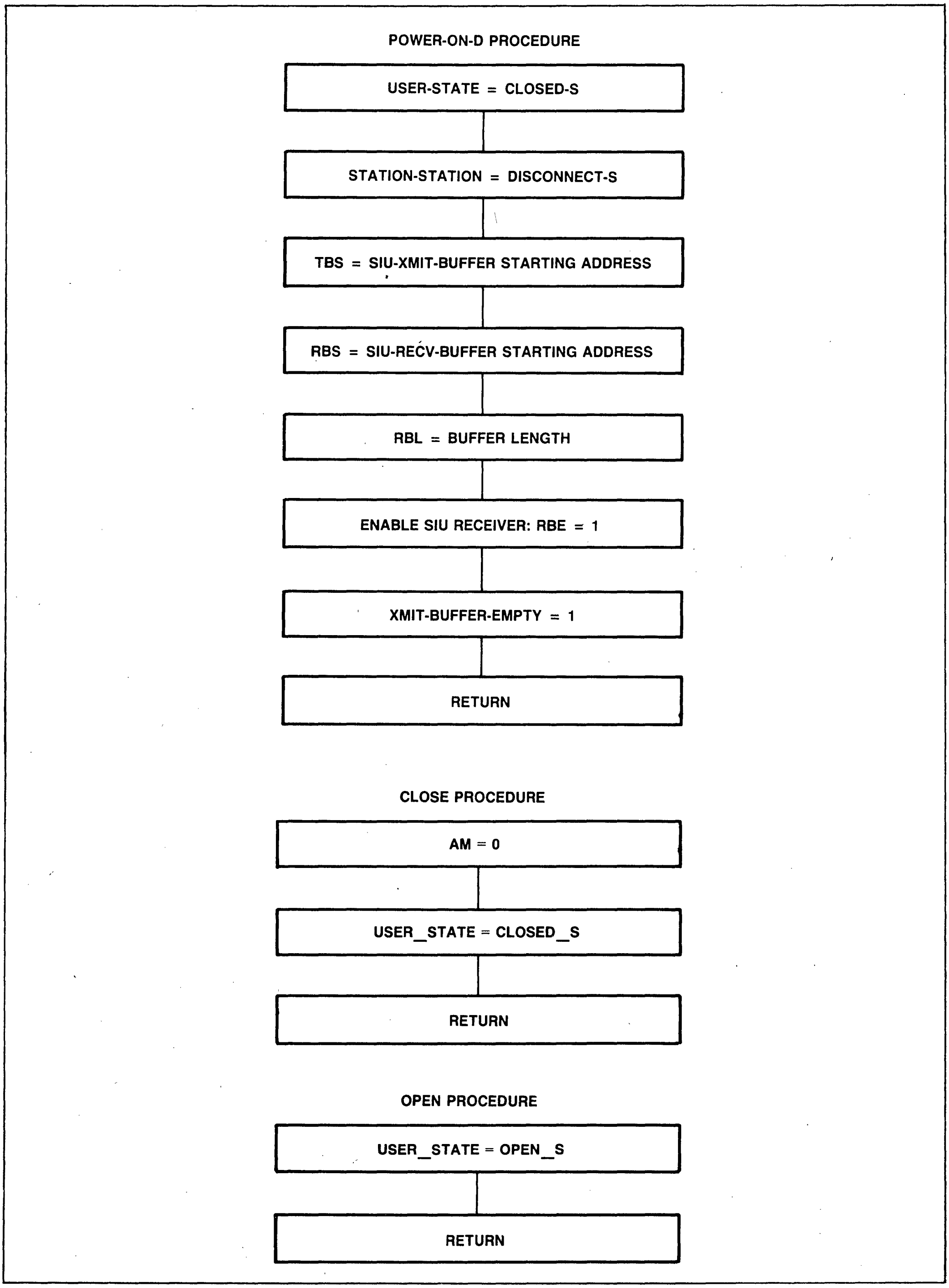


Figure 20-18. Secondary Station Driver Flow Chart

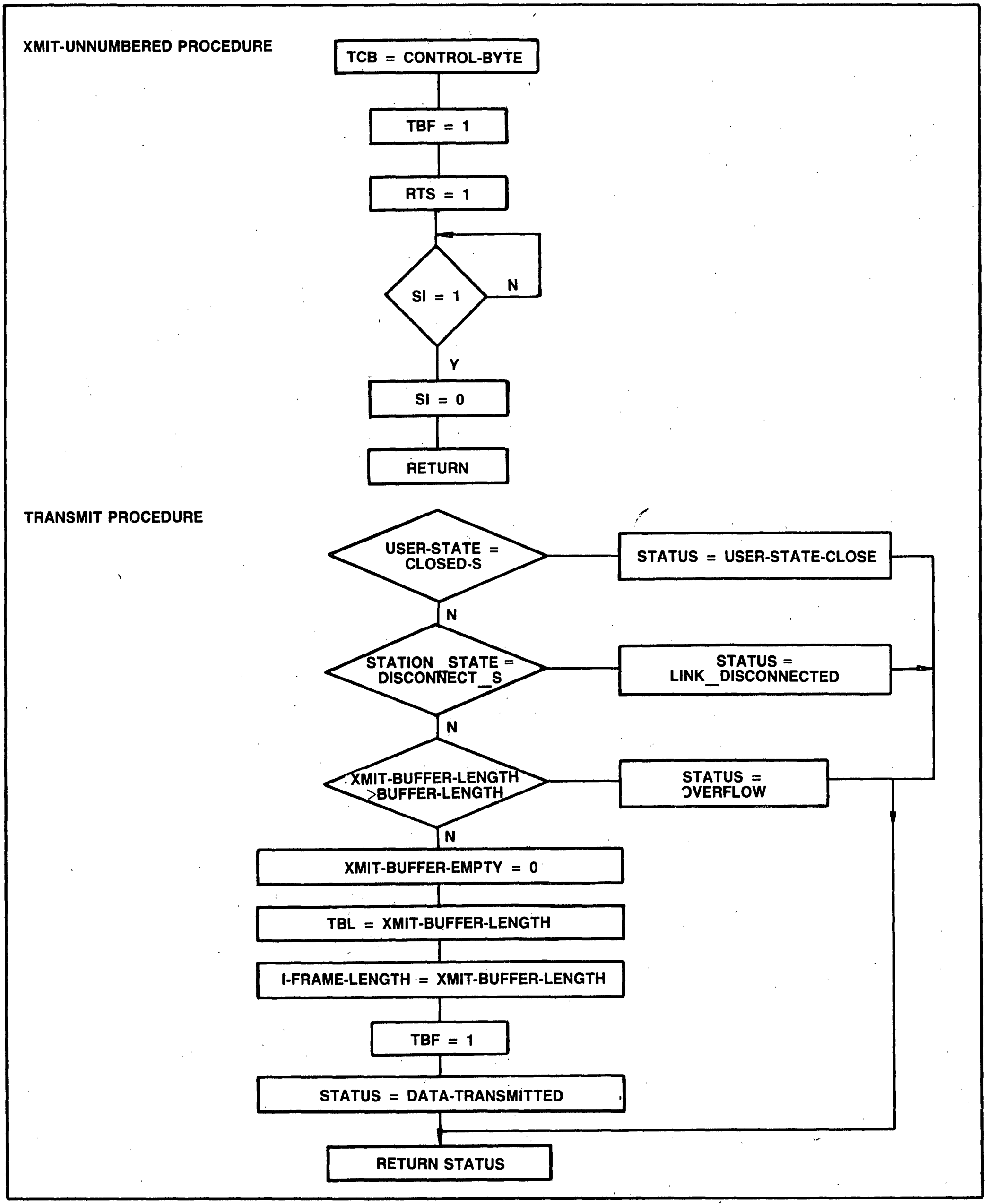

XMIT-UNNUMBERED PROCEDURE
TCB = CONTROL-BYTE
TBF = 1
RTS = 1
SI = 1
N
Y
SI = 0
RETURN
TRANSMIT PROCEDURE
USER-STATE = CLOSED-S
STATUS = USER-STATE-CLOSE
N
STATION_STATE = DISCONNECT_S
STATUS = LINK_DISCONNECTED
N
XMIT-BUFFER-LENGTH >BUFFER-LENGTH
STATUS = OVERFLOW
N
XMIT-BUFFER-EMPTY = 0
TBL = XMIT-BUFFER-LENGTH
I-FRAME-LENGTH = XMIT-BUFFER-LENGTH
TBF = 1
STATUS = DATA-TRANSMITTED
RETURN STATUS

Figure 20-20. Secondary Station Driver Flow Chart

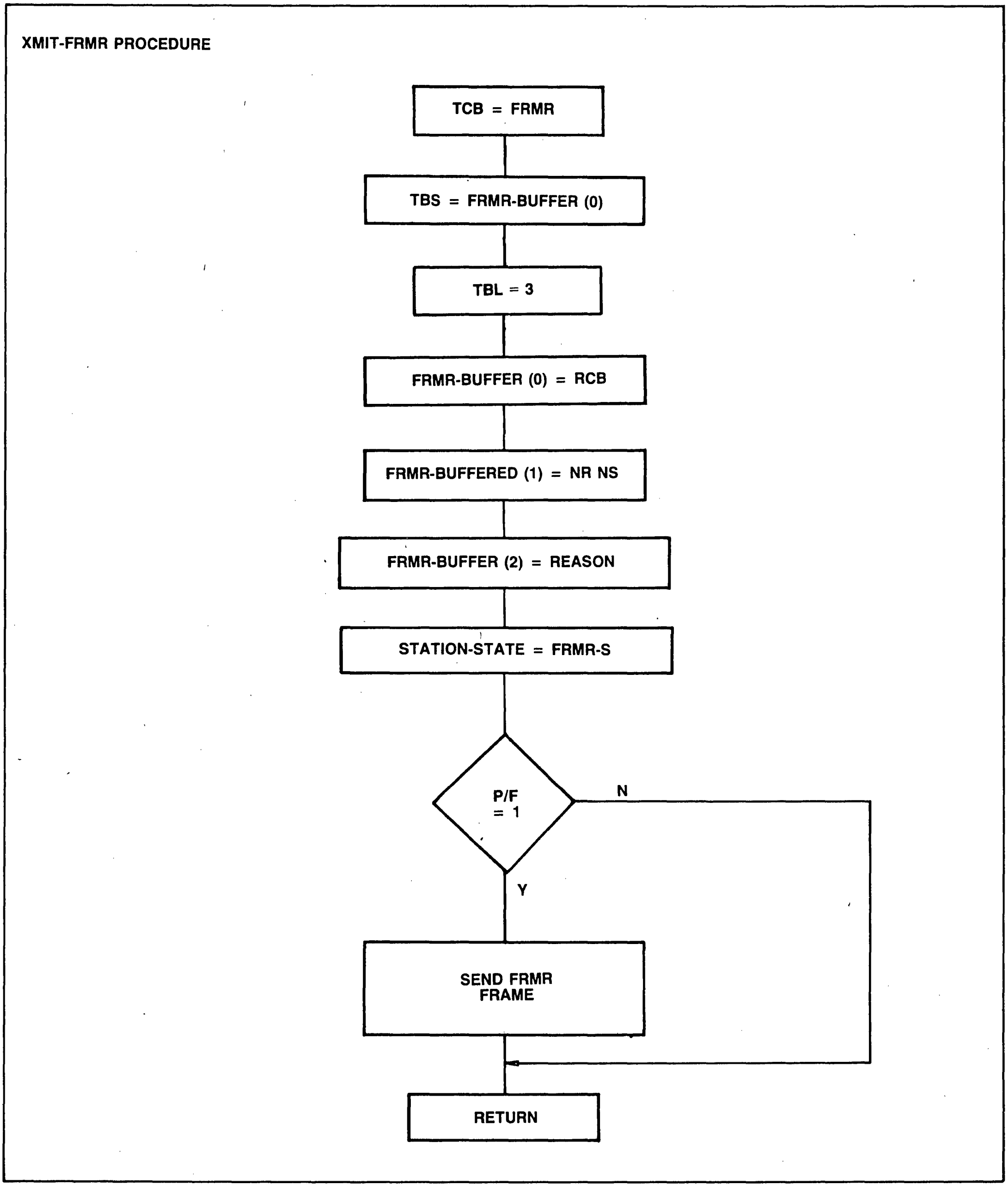


Figure 20-21. Secondary Station Driver Flow Chart

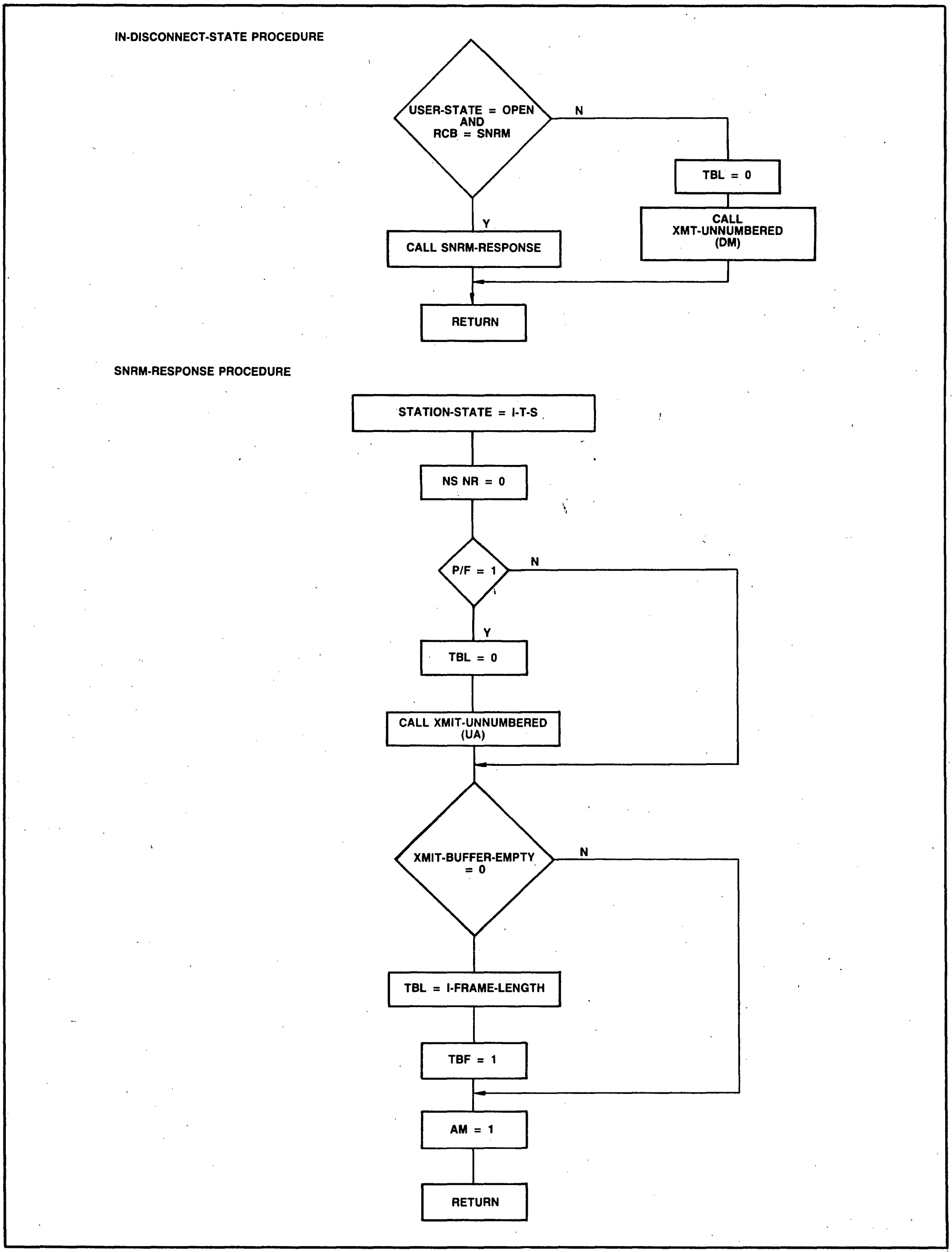


Figure 20-22. Secondary Station Driver Flow Chart

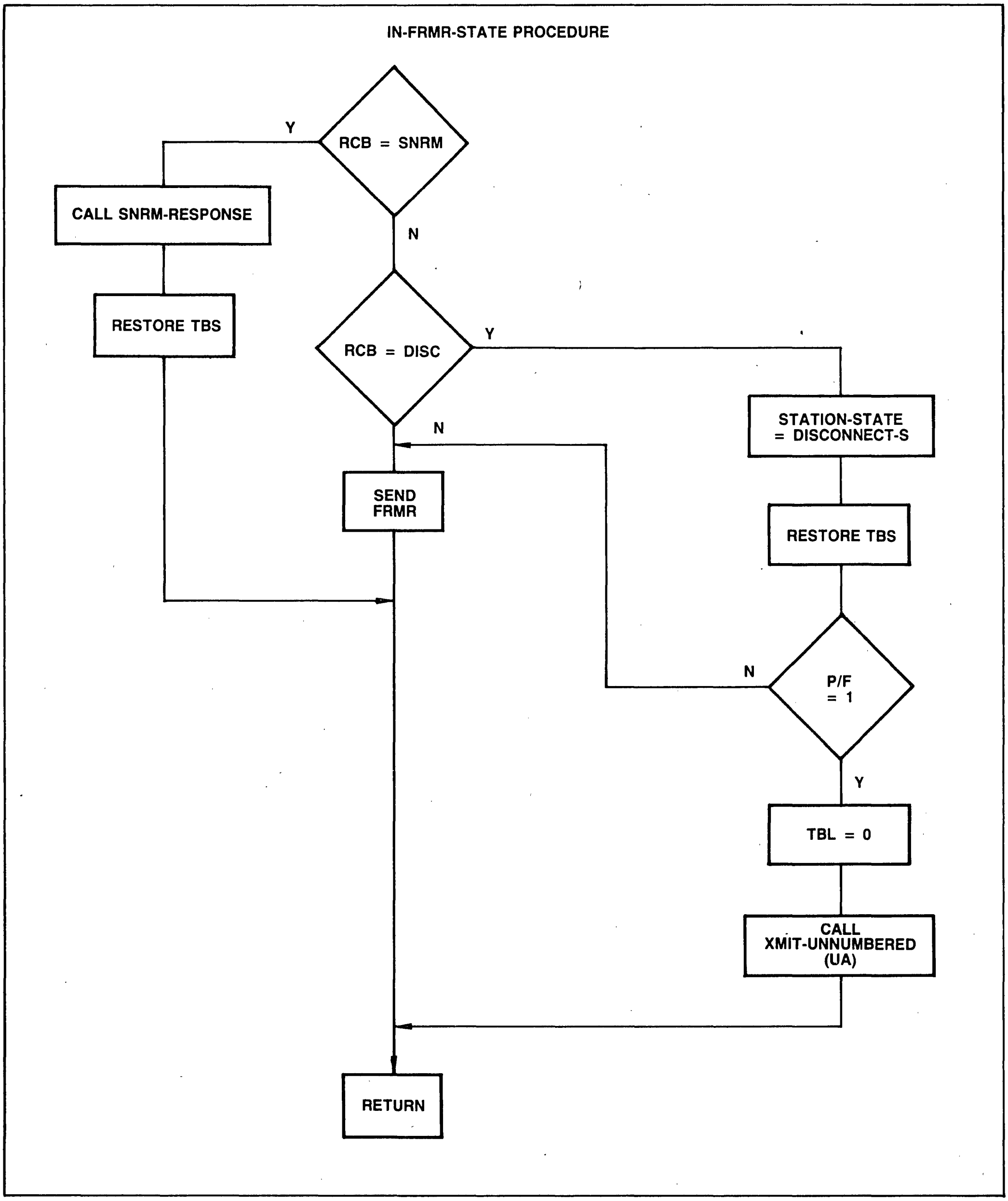


Figure 20-24. Secondary Station Driver Flow Chart

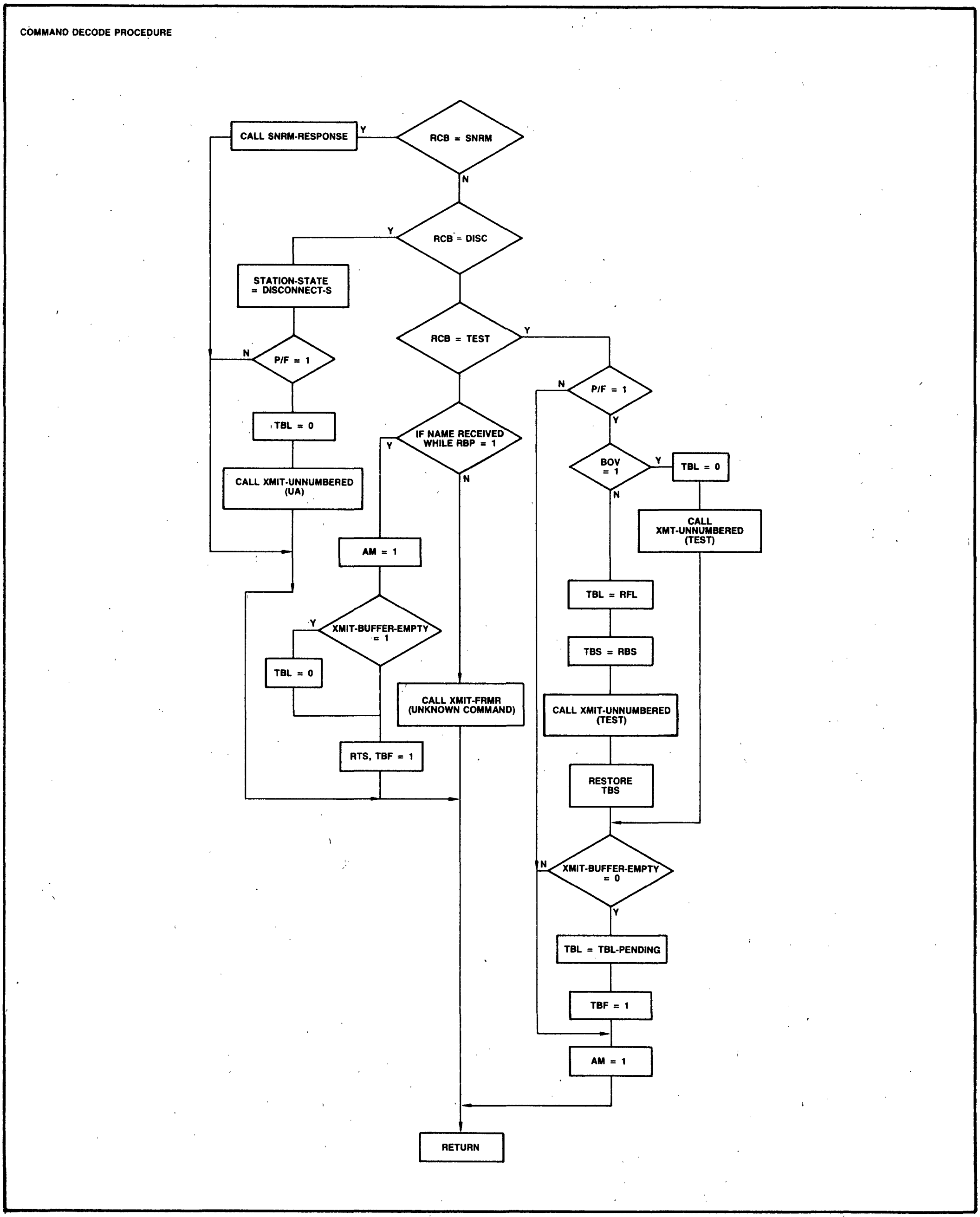


Figure 20-25. Secondary Station Driver Flow Chart

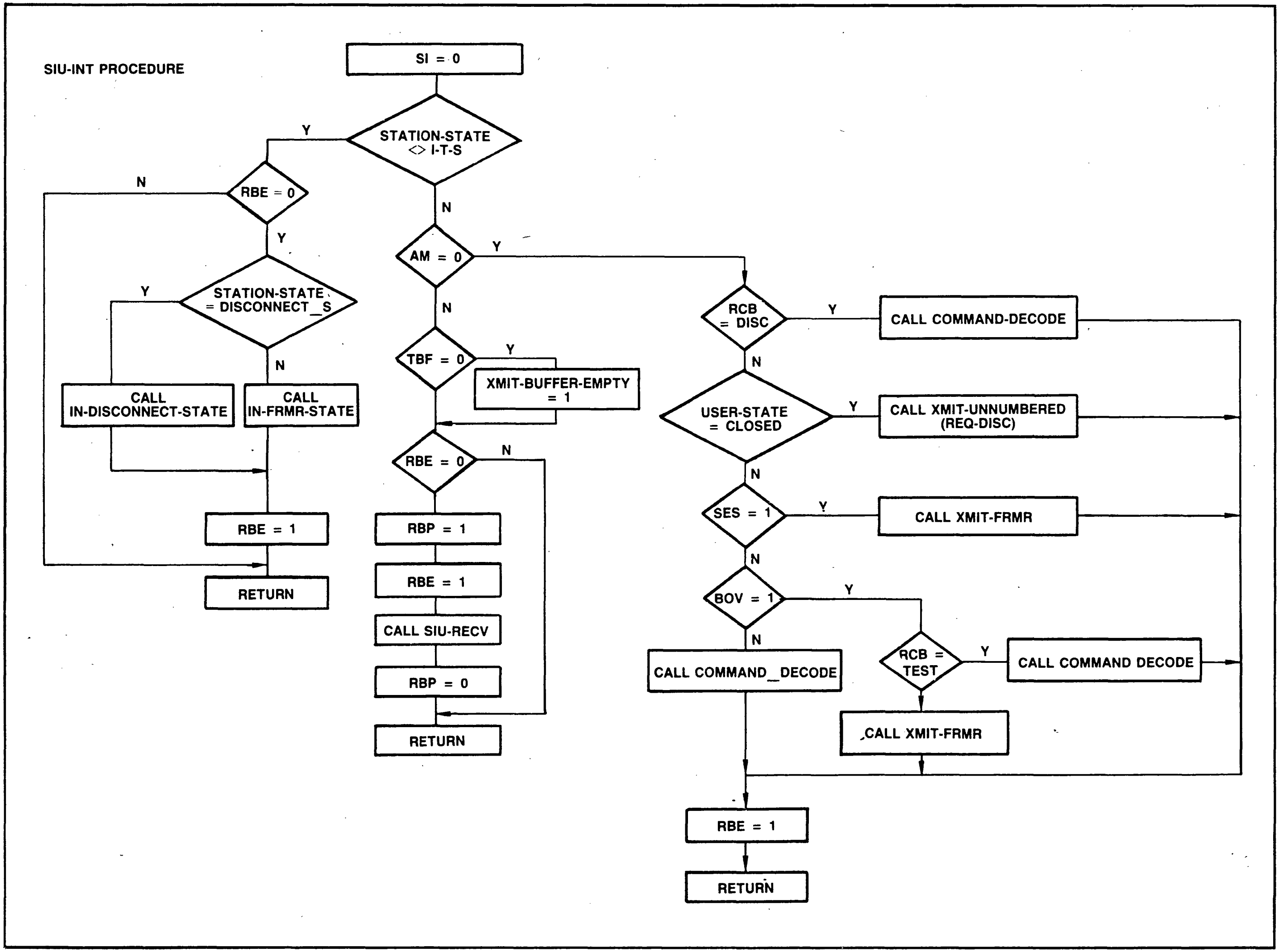


Figure 20-26. Secondary Station Driver Flow Chart

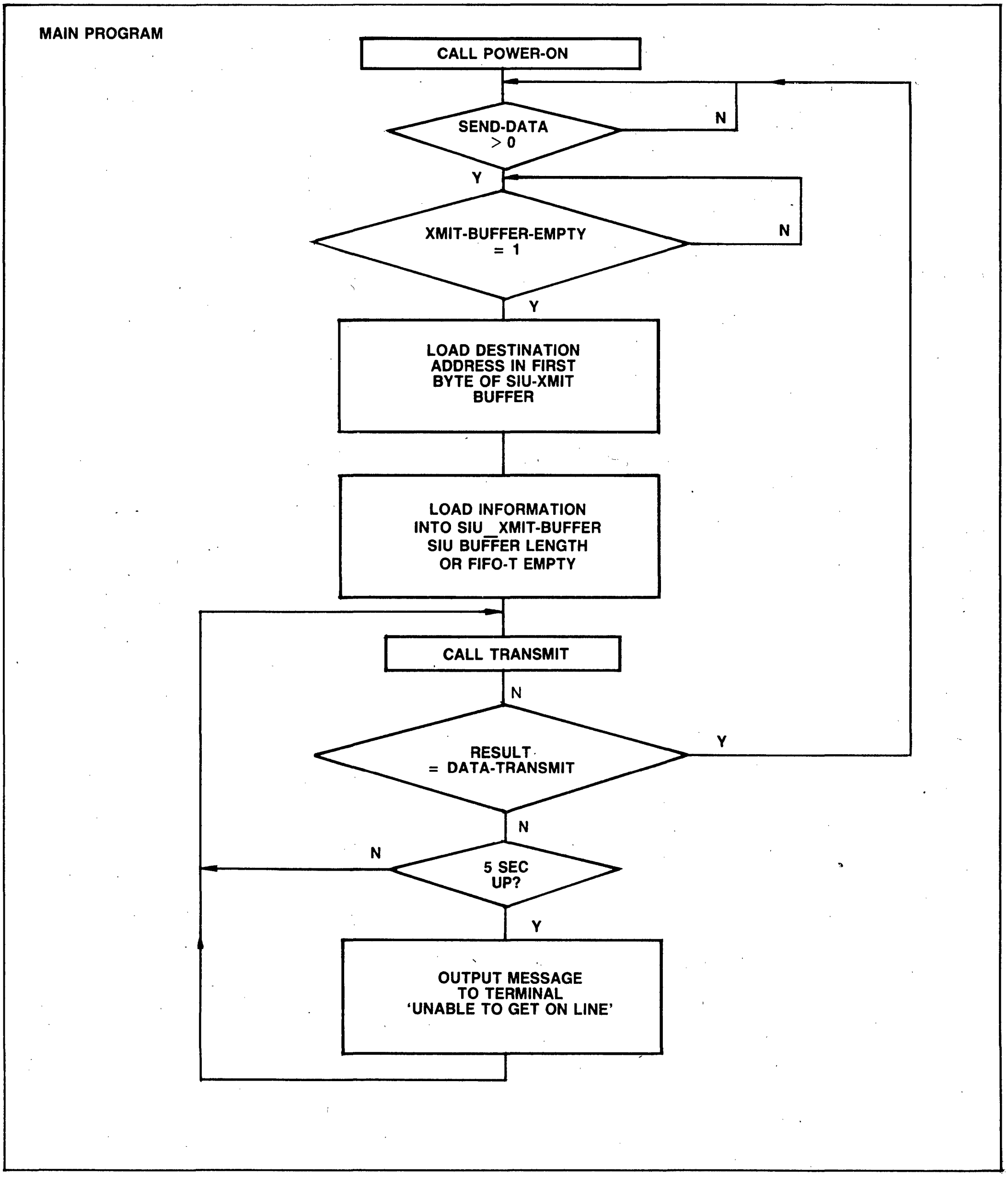

MAIN PROGRAM
CALL POWER-ON
SEND-DATA > 0
N
Y
XMIT-BUFFER-EMPTY = 1
N
Y
LOAD DESTINATION ADDRESS IN FIRST BYTE OF SIU-XMIT BUFFER
LOAD INFORMATION INTO SIU_XMIT-BUFFER SIU BUFFER LENGTH OR FIFO-T EMPTY
CALL TRANSMIT
N
RESULT = DATA-TRANSMIT
Y
N
N
5 SEC UP?
Y
OUTPUT MESSAGE TO TERMINAL 'UNABLE TO GET ON LINE'

Figure 20-27. Application Module Flow Chart

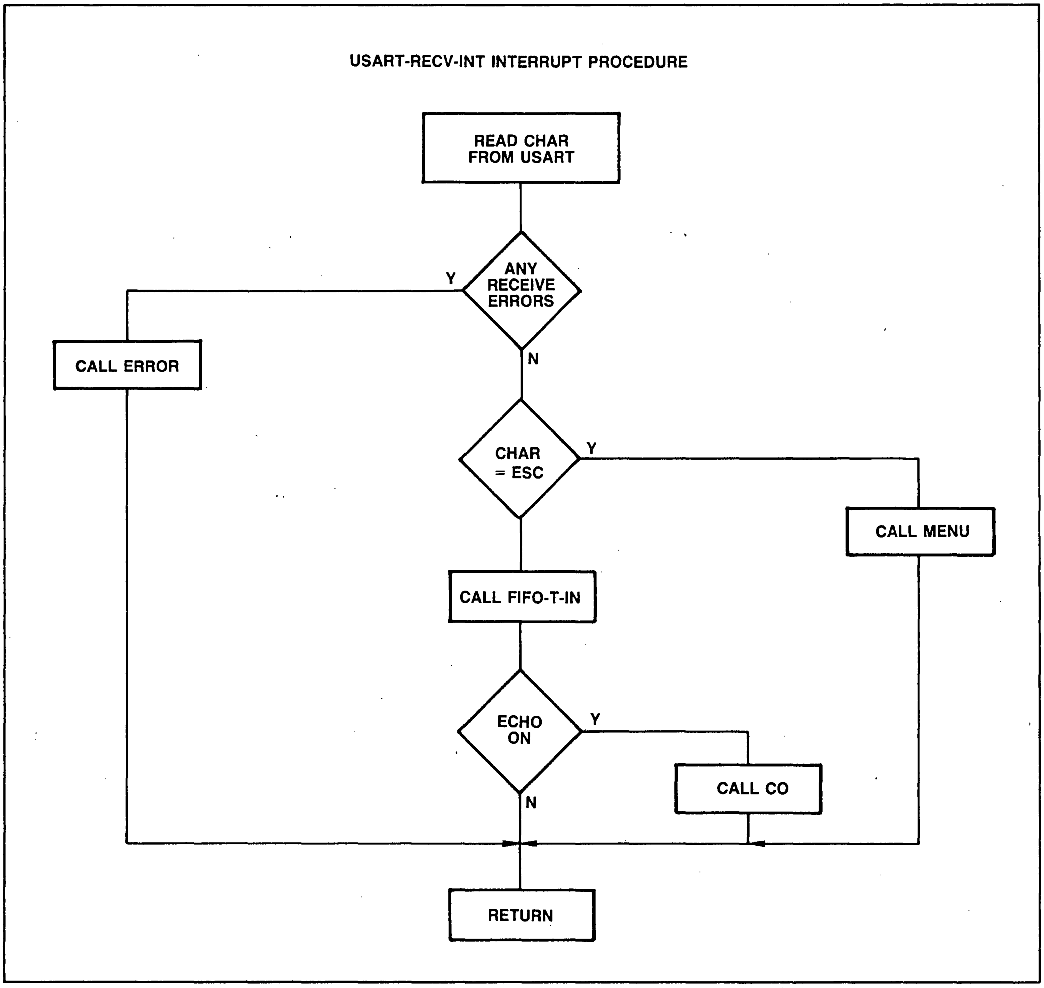


Figure 20-28. Application Module Flow Chart

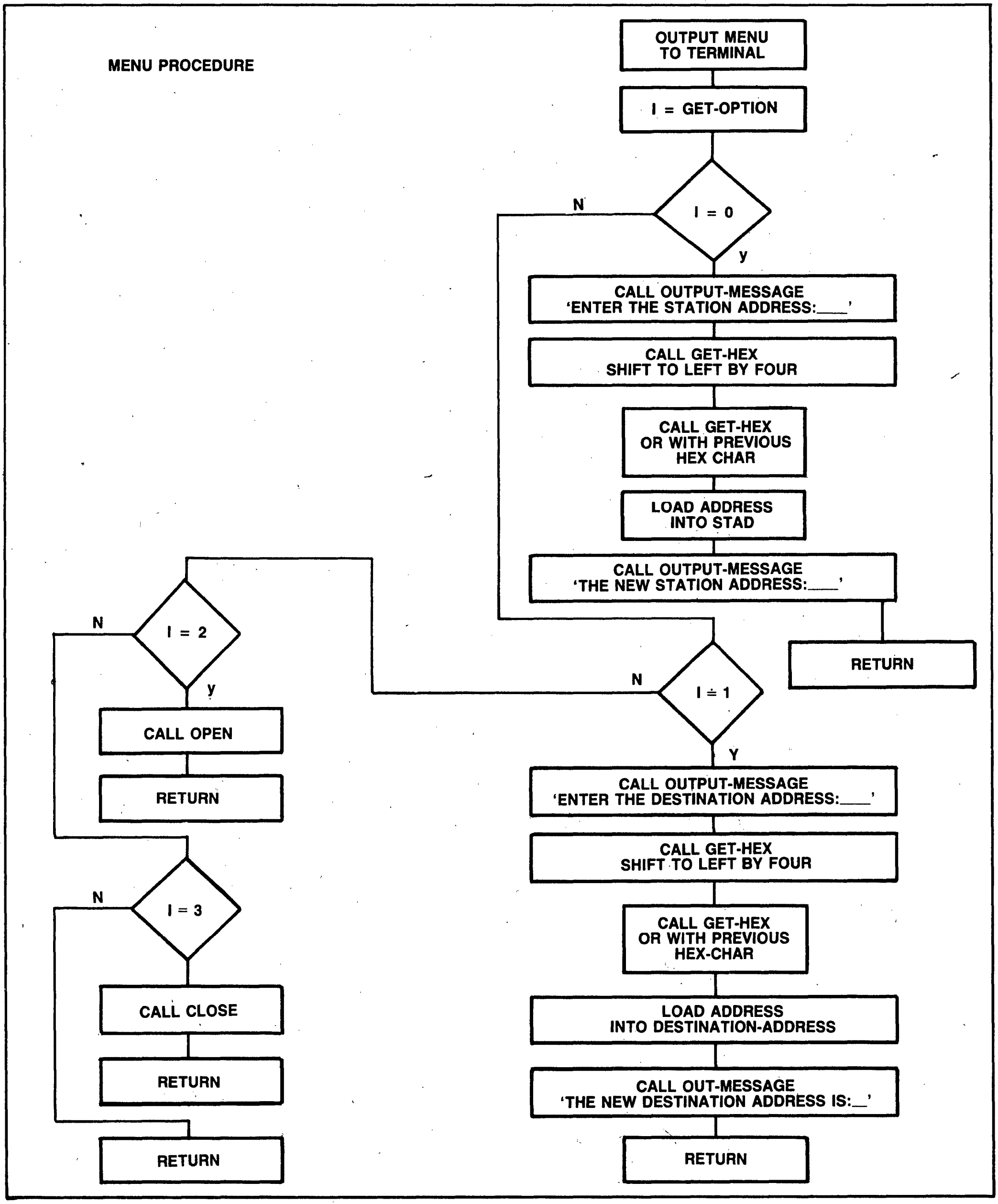

MENU PROCEDURE
OUTPUT MENU TO TERMINAL
I = GET-OPTION
I = 0
N
y
CALL OUTPUT-MESSAGE 'ENTER THE STATION ADDRESS:___'
CALL GET-HEX SHIFT TO LEFT BY FOUR
CALL GET-HEX OR WITH PREVIOUS HEX CHAR
LOAD ADDRESS INTO STAD
CALL OUTPUT-MESSAGE 'THE NEW STATION ADDRESS:___'
RETURN
I ≐ 1
N
Y
CALL OUTPUT-MESSAGE 'ENTER THE DESTINATION ADDRESS:___'
CALL GET-HEX SHIFT TO LEFT BY FOUR
CALL GET-HEX OR WITH PREVIOUS HEX-CHAR
LOAD ADDRESS INTO DESTINATION-ADDRESS
CALL OUT-MESSAGE 'THE NEW DESTINATION ADDRESS IS:_'
RETURN
I = 2
N
y
CALL OPEN
RETURN
I = 3
N
CALL CLOSE
RETURN
RETURN

Figure 20-29. Application Module Flow Chart

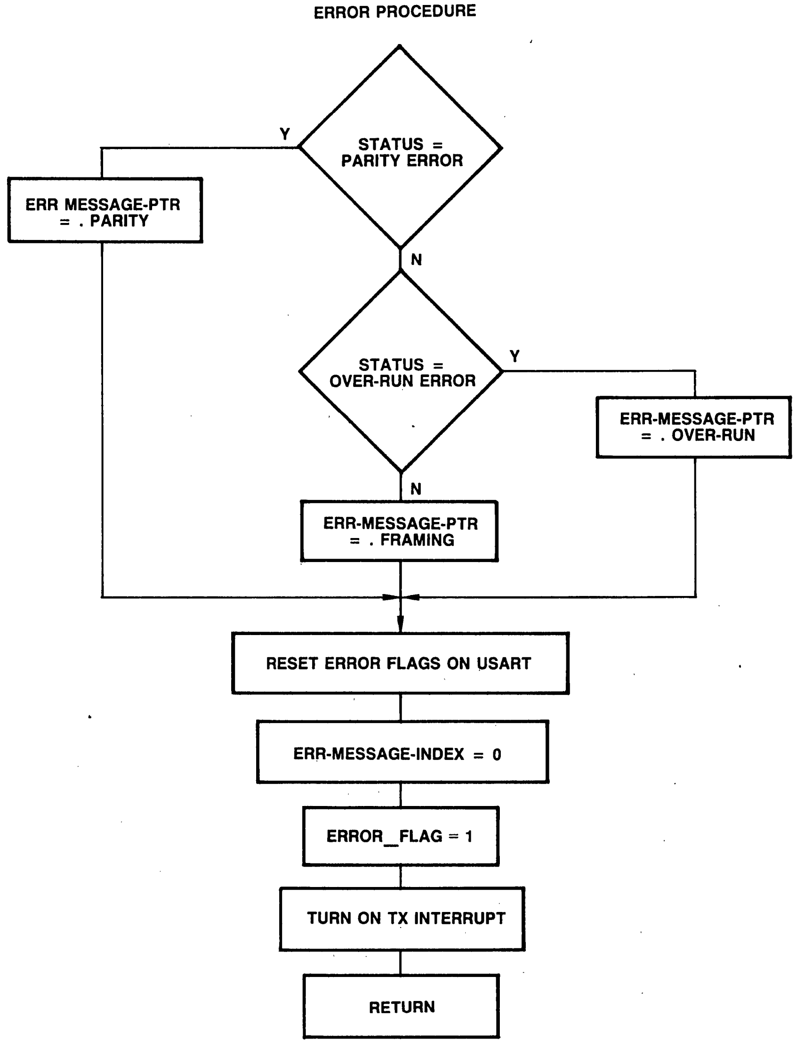


Figure 20-30. Application Module Flow Chart

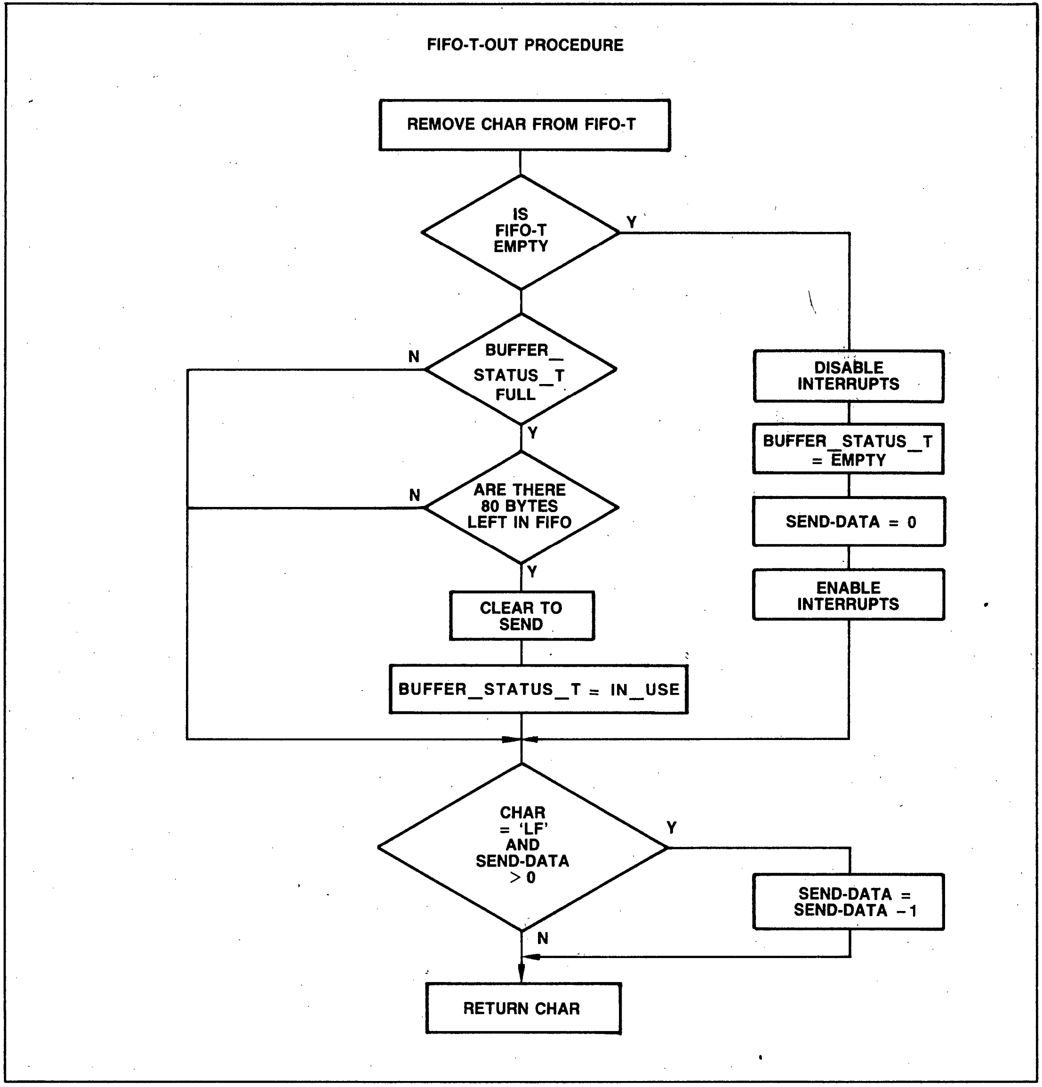


Figure 20-31. Application Module Flow Chart

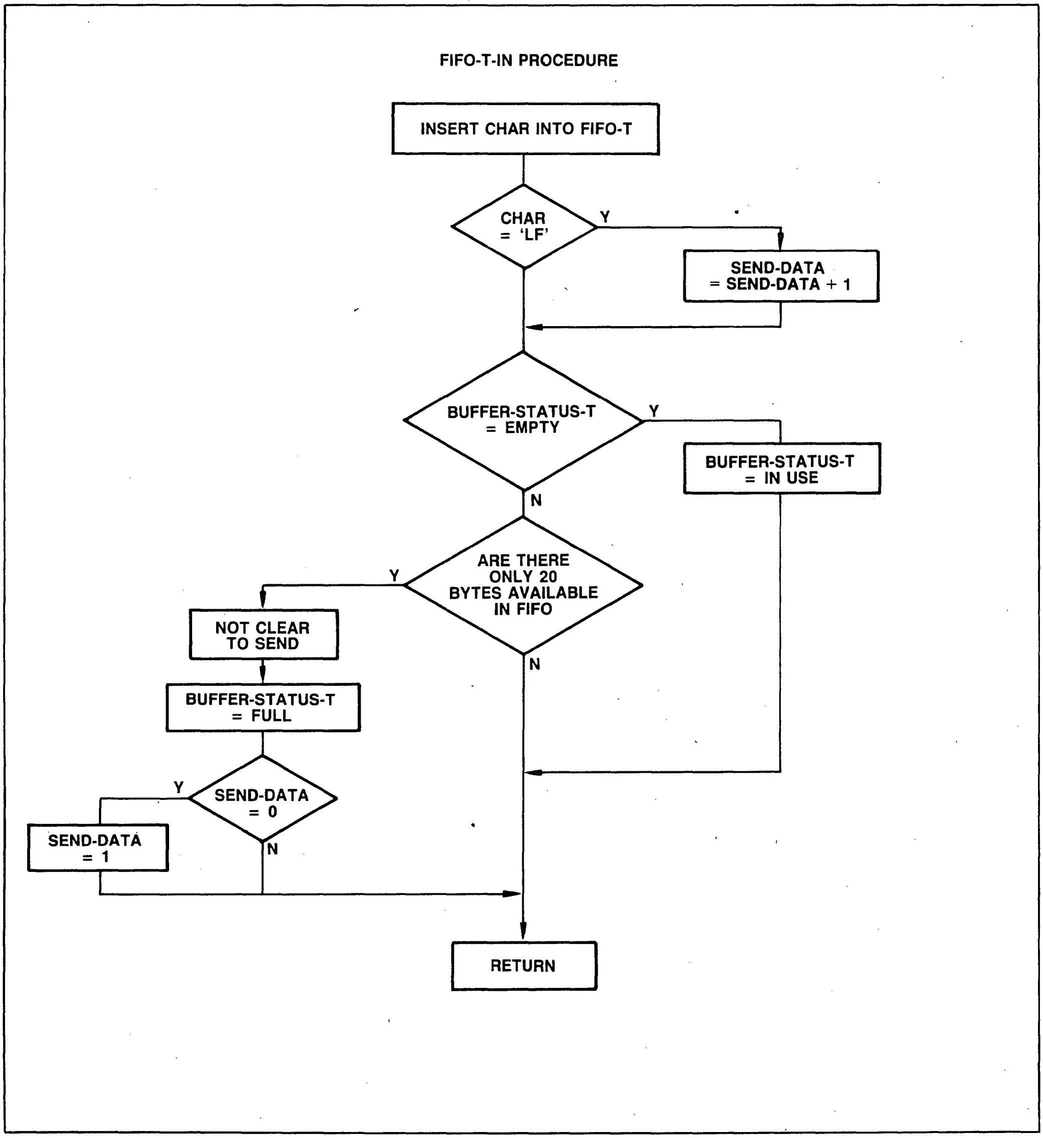


Figure 20-32. Application Module Flow Chart

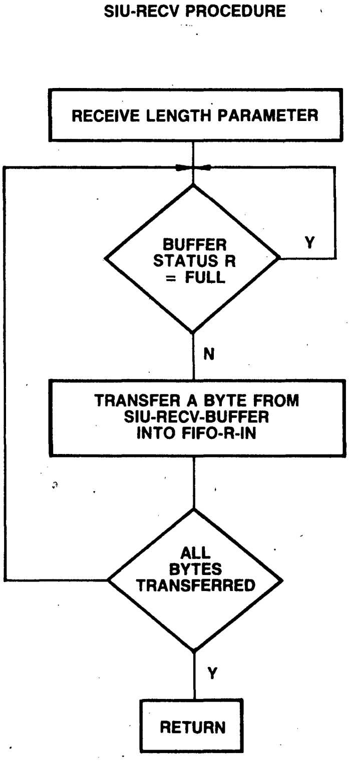


Figure 20-33. Application Module Flow Chart

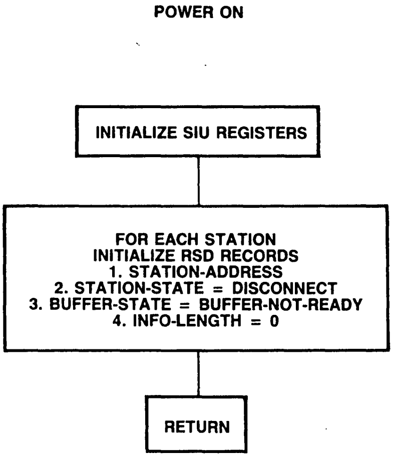


Figure 20-34. Primary Station Flow Charts

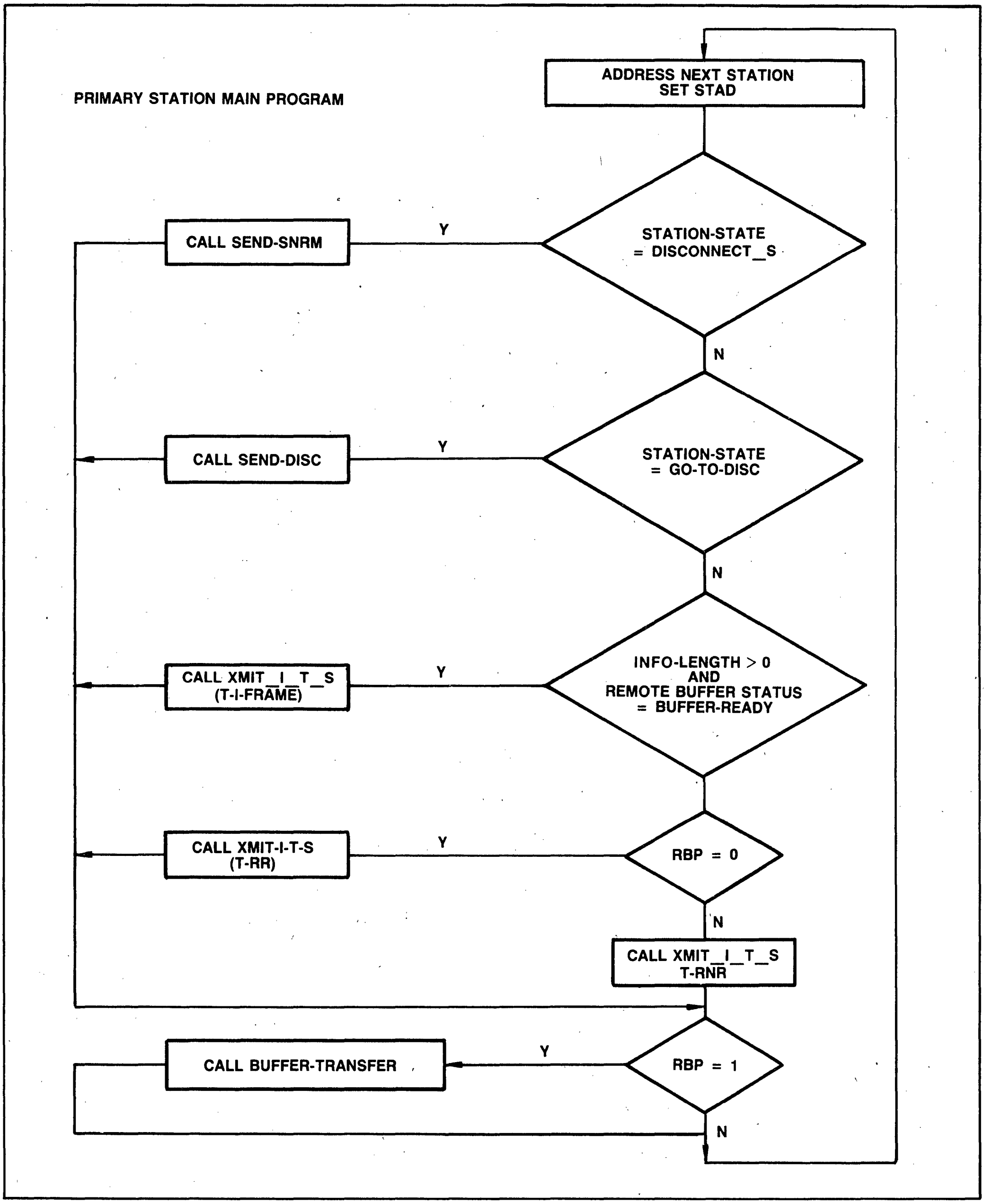


Figure 20-35. Primary Station Flow Charts

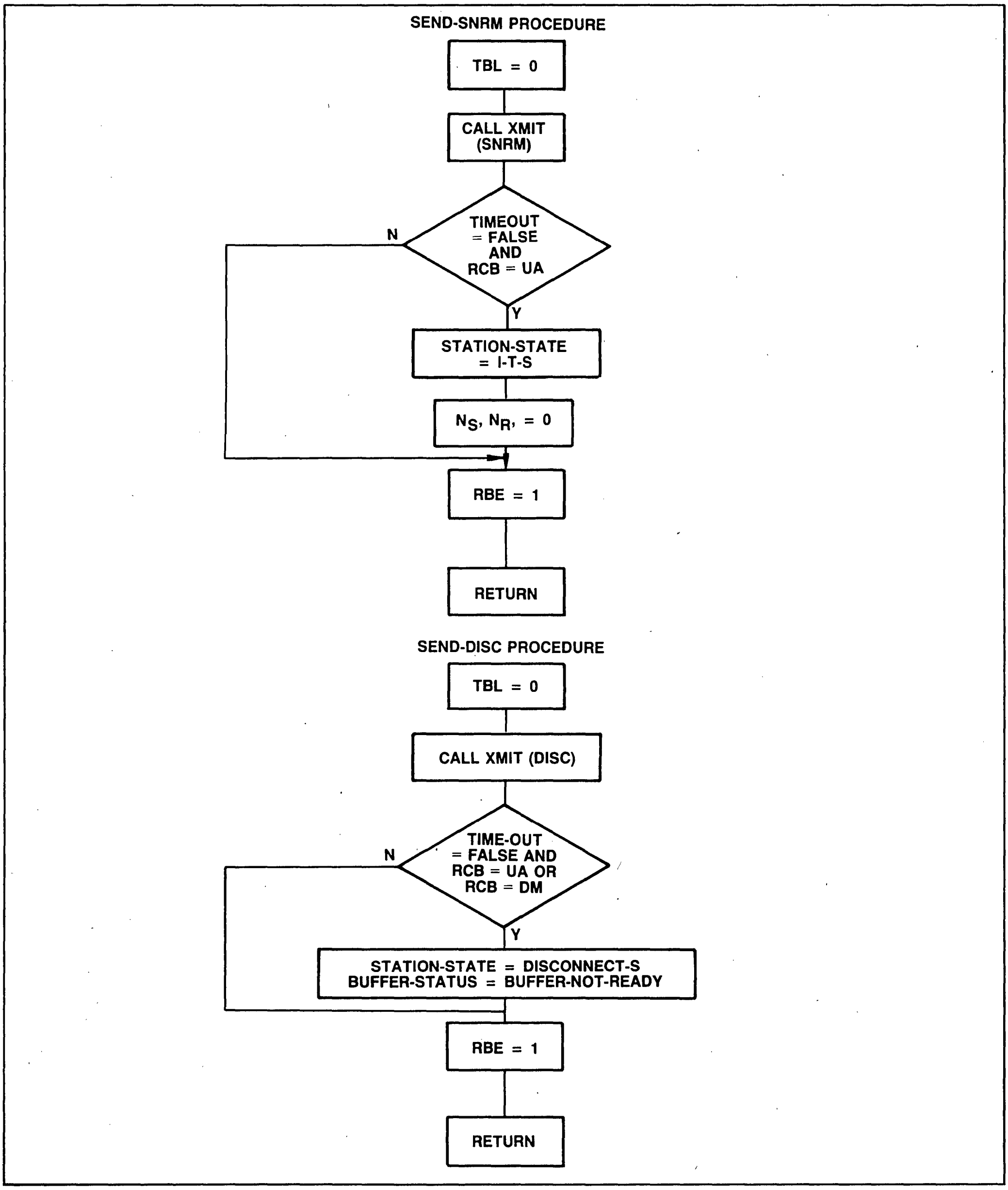


Figure 20-36. Primary Station Flow Charts

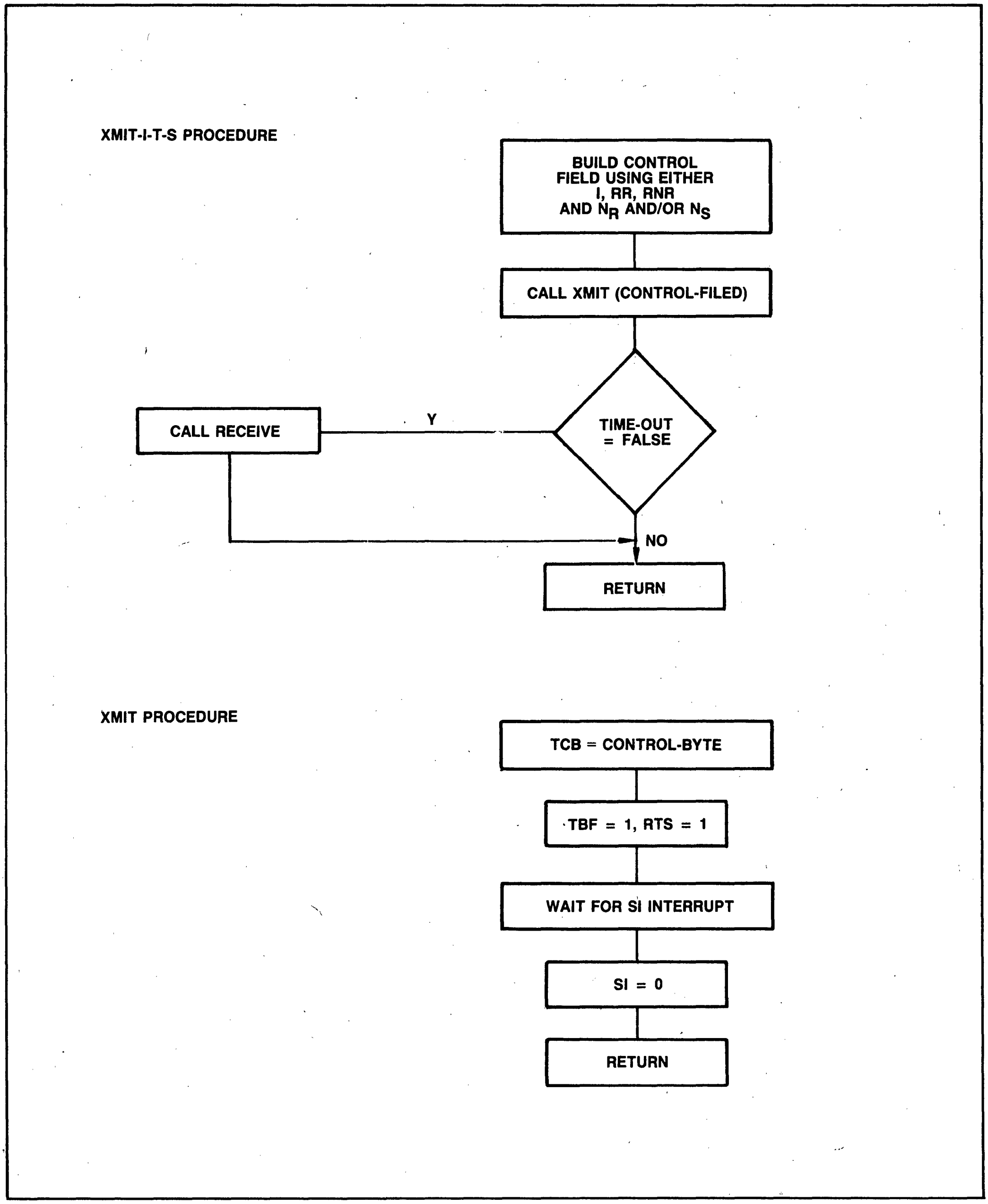


Figure 20-38. Primary Station Flow Charts

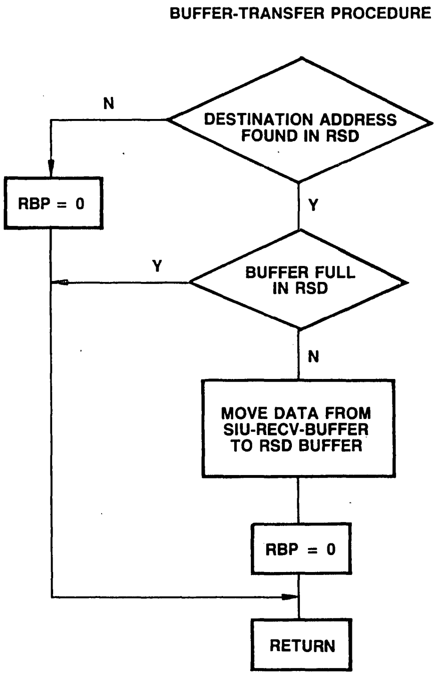

BUFFER-TRANSFER PROCEDURE
N
DESTINATION ADDRESS FOUND IN RSD
RBP = 0
Y
Y
BUFFER FULL IN RSD
N
MOVE DATA FROM SIU-RECV-BUFFER TO RSD BUFFER
RBP = 0
RETURN

Figure 20-39. Primary Station Flow Charts

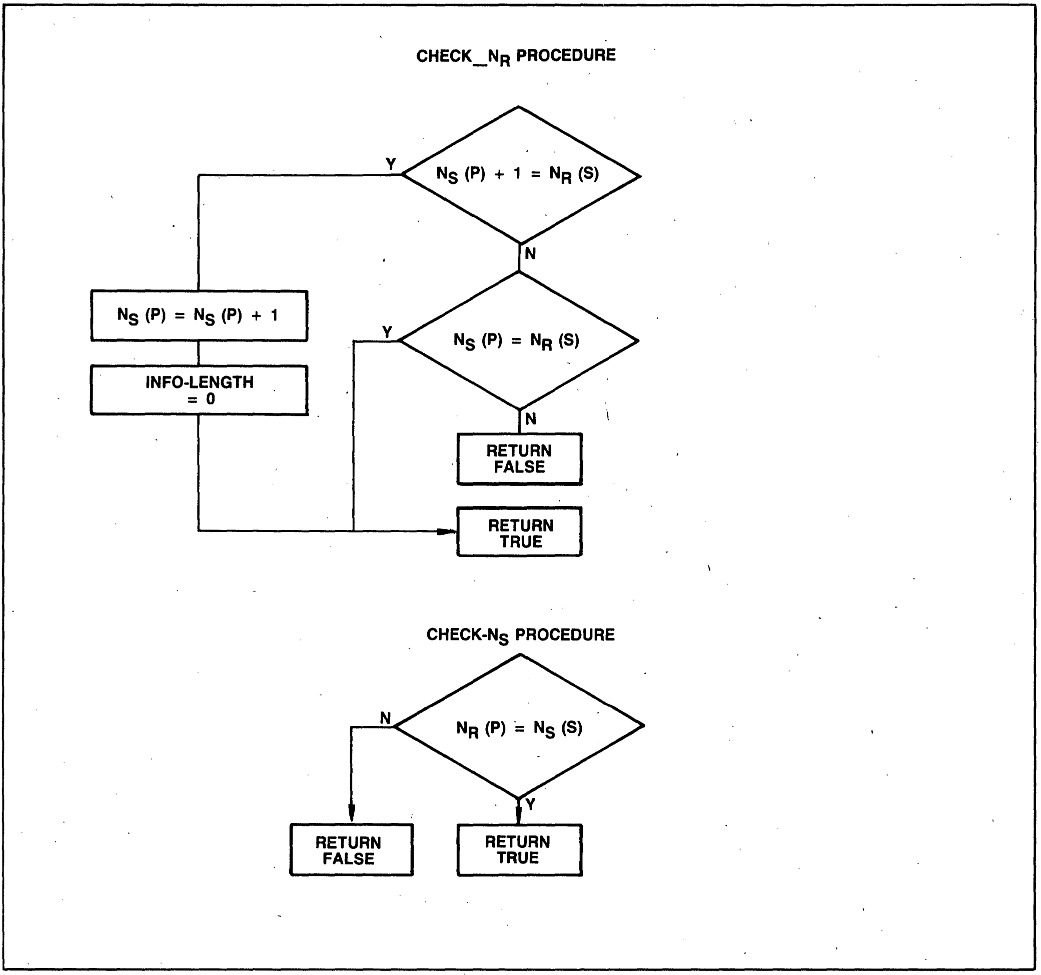


Figure 20-40. Primary Station Flow Charts

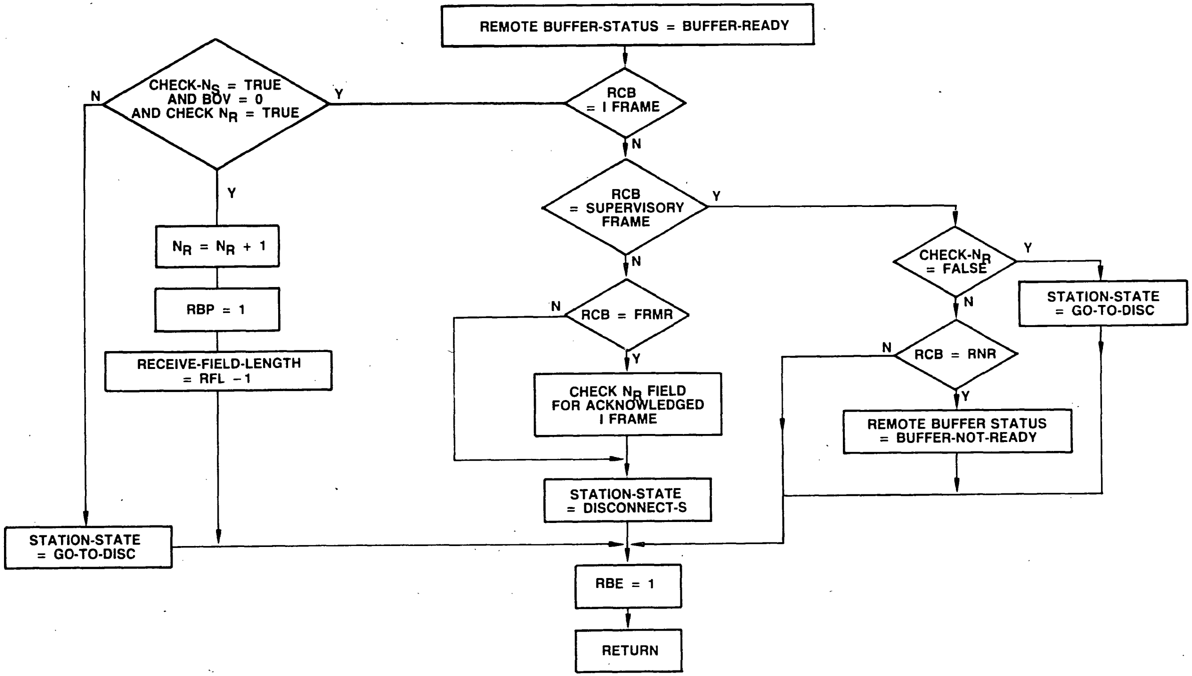

REMOTE BUFFER-STATUS = BUFFER-READY
RCB = I FRAME
Y
N
CHECK-N_S = TRUE AND BOV = 0 AND CHECK N_R = TRUE
N
Y
$N_R = N_R + 1$
RBP = 1
RECEIVE-FIELD-LENGTH = RFL −1
STATION-STATE = GO-TO-DISC
RCB = SUPERVISORY FRAME
Y
N
RCB = FRMR
N
Y
CHECK N_R FIELD FOR ACKNOWLEDGED I FRAME
STATION-STATE = DISCONNECT-S
CHECK-N_R = FALSE
Y
N
STATION-STATE = GO-TO-DISC
RCB = RNR
N
Y
REMOTE BUFFER STATUS = BUFFER-NOT-READY
RBE = 1
RETURN

Figure 20-41. Primary Station Flow Charts

RUPI™ Data Sheets 21

intel®

ADVANCE INFORMATION

8044AH/8344AH HIGH PERFORMANCE 8-BIT MICROCONTROLLER WITH ON-CHIP SERIAL COMMUNICATION CENTER

- **8044AH — CPU/SIU with Factory Mask Programmable ROM**
- **8344AH — An 8044AH used with External Program Memory**
- **8744H — An 8044AH with User Programmable/Erasable EPROM**

8051 MICROCONTROLLER CORE

- **Optimized for Real Time Control 12 MHz Clock, Priority Interrupts, 32 Programmable I/O lines, Two 16-bit Timer/Counters**
- **Boolean Processor**
- **4K x 8 ROM, 192 x 8 RAM**
- **64K Accessible External Program Memory**
- **64K Accessible External Data Memory**
- **4 μs Multiply and Divide**

SERIAL INTERFACE UNIT (SIU)

- **Serial Communication Processor that Operates Concurrently to C.P.U.**
- **2.4 Mbps Maximum Data Rate**
- **375 Kbps using On-Chip Phase Locked Loop**
- **Communication Software in Silicon:**
  - **— Complete Data Link Functions**
  - **— Automatic Station Responses**
- **Operates as an SDLC Primary or Secondary Station**

The RUPI-44 family integrates a high performance 8-bit Microcontroller, the Intel 8051 Core, with an intelligent/high performance HDLC/SDLC serial communication controller, called the Serial Interface Unit (SIU). See Figure 1. This dual architecture allows complex control and high speed data communication functions to be realized cost effectively.

Specifically, the 8044's Microcontroller features: 4K byte On-Chip program memory space; 32 I/O lines; two 16-bit timer/event counters; a 5-source; 2-level interrupt structure; a full duplex serial channel; a Boolean processor; and on-chip oscillator and clock circuitry. Standard TTL and most byte-oriented MCS-80 and MCS-85 peripherals can be used for I/O and memory expansion.

The Serial Interface Unit (SIU) manages the interface to a high speed serial link. The SIU offloads the On-Chip 8051 Microcontroller of communication tasks, thereby freeing the CPU to concentrate on real time control tasks.

The RUPI-44 family consists of the 8044, 8744, and 8344. All three devices are identical except in respect of on-chip program memory. The 8044 contains 4K bytes of mask-programmable ROM. User programmable EPROM replaces ROM in the 8744. The 8344 addresses all program memory externally.

The RUPI-44 devices are fabricated with Intel's reliable +5 volt, silicon-gate HMOSII technology and packaged in a 40-pin DIP.

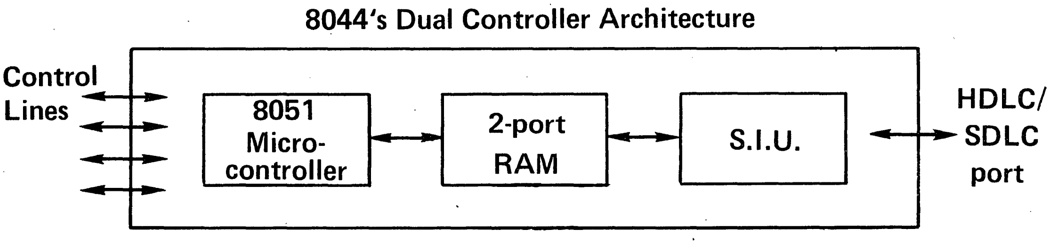


Figure 1. Dual Controller Architecture

Table 1. RUPI™ -44 Family Pin Description

VSS
Circuit ground potential.

VCC
+5V power supply during operation and program verification.

PORT 0
Port 0 is an 8-bit open drain bidirectional I/O port. It is also the multiplexed low-order address and data bus when using external memory. It is used for data output during program verification. Port 0 can sink/source eight LS TTL loads.

PORT 1
Port 1 is an 8-bit quasi-bidirectional I/O port. It is used for the low-order address byte during program verification. Port 1 can sink/source four LS TTL loads.

In non-loop mode two of the I/O lines serve alternate functions:

—$\overline{\text{RTS}}$ (P1.6). Request-to-Send output. A low indicates that the RUPI-44 is ready to transmit.
—$\overline{\text{CTS}}$ (P1.7) Clear-to-Send input. A low indicates that a receiving station is ready to receive.

PORT 2
Port 2 is an 8-bit quasi-bidirection I/O port. It also emits the high-order address byte when accessing external memory. It is used for the high-order address and the control signals during program verification. Port 2 can sink/source four LS TTL loads.

PORT 3
Port 3 is an 8-bit quasi-bidirectional I/O port. 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 secondary function must be programmed to a one (1) for that function to operate. Port 3 can sink/source four LS TTL loads.

In addition to I/O, some of the pins also serve alternate functions as follows:

—I/$\overline{\text{O}}$ RxD (P3.0). In point-to-point or multipoint configurations, this pin controls the direction of pin P3.1. Serves as Receive Data input in loop and diagnostic modes.
—DATA TxD (P3.1) In point-to-point or multipoint configurations, this pin functions as data input/output. In loop mode, it serves as transmit pin. A '0' written to this pin enables diagnostic mode.
—$\overline{\text{INT0}}$ (P3.2). Interrupt 0 input or gate control input for counter 0.
—$\overline{\text{INT1}}$ (P3.3). Interrupt 1 input or gate control input for counter 1.
—TO(P3.4). Input to counter 0.
—SCLK T1 (P3.5). In addition to I/O, this pin provides input to counter 1 or serves as SCLK (serial clock) input.
—$\overline{\text{WR}}$ (P3.6). The write control signal latches the data byte from Port 0 into the External Data Memory.
—$\overline{\text{RD}}$ (P3.7). The read control signal enables External Data Memory to Port 0.

RST/VPD
A high level on this pin resets the RUPI-44. A small internal pulldown resistor permits power-on reset using only a capacitor connected to VCC. If VPD is held within its spec while VCC drops below spec, VPD will provide standby power to the RAM. When VPD is low, the RAM's current is drawn from VCC.

ALE/$\overline{\text{PROG}}$
Provides Address Latch Enable output used for latching the address into external memory during normal operation. It is activitated every six oscillator periods except during an external data memory access. It also receives the program pulse input for programming the EPROM version.

$\overline{\text{PSEN}}$
The Program Store Enable output is a control signal that enables the external Program Memory to the bus during external fetch operations. It is activated every six oscillator periods, except during external data memory accesses. Remains high during internal program execution.

$\overline{\text{EA}}$/VPP
When held at a TTL high level, the RUPI-44 executes instructions from the internal ROM when the PC is less than 4096. When held at a TTL low level, the RUPI-44 fetches all instructions from external Program Memory. The pin also receives the 21V EPROM programming supply voltage on the 8744.

XTAL 1
Input to the oscillator's high gain amplifier. Required when a crystal is used. Connect to VSS when external source is used on XTAL 2.

XTAL 2
Output from the oscillator's amplifier. Input to the internal timing circuitry. A crystal or external source can be used.

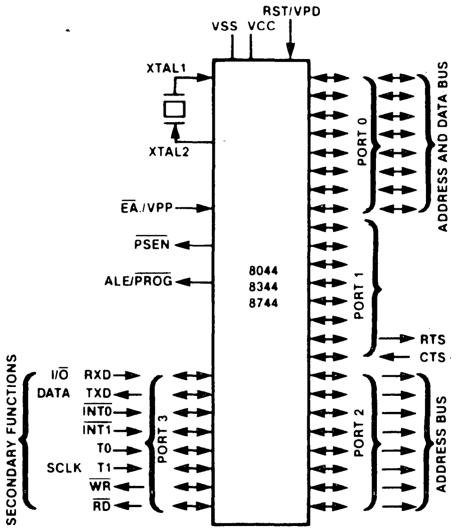


Figure 2. Logic Symbol

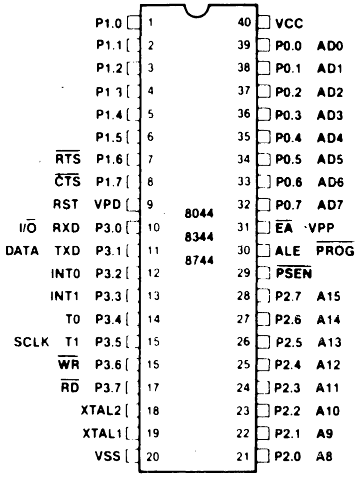


Figure 3. Pin Configuration

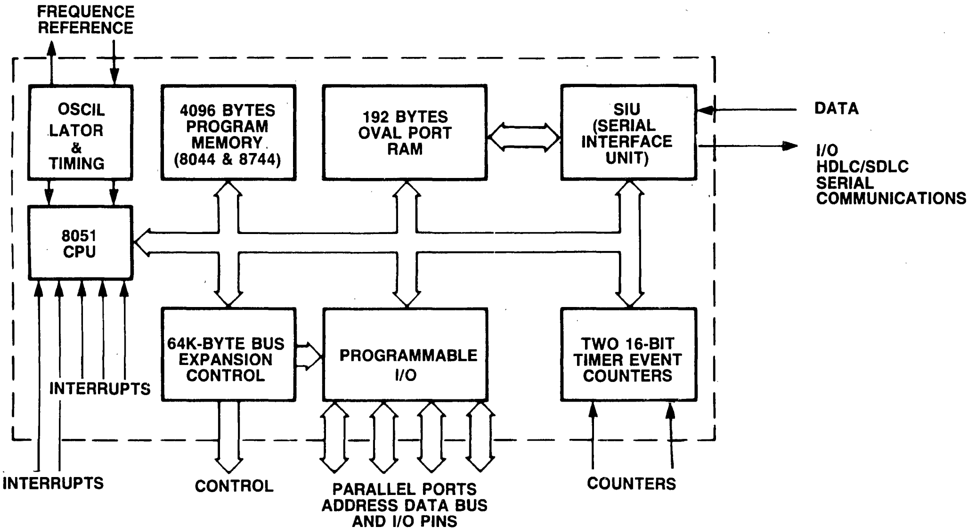

FREQUENCE REFERENCE
OSCIL LATOR & TIMING
4096 BYTES PROGRAM MEMORY (8044 & 8744)
192 BYTES OVAL PORT RAM
SIU (SERIAL INTERFACE UNIT)
DATA
I/O
HDLC/SDLC SERIAL COMMUNICATIONS
8051 CPU
INTERRUPTS
64K-BYTE BUS EXPANSION CONTROL
PROGRAMMABLE I/O
TWO 16-BIT TIMER EVENT COUNTERS
INTERRUPTS
CONTROL
PARALLEL PORTS ADDRESS DATA BUS AND I/O PINS
COUNTERS

Figure 4. Block Diagram

Functional Description

General

The 8044 integrates the powerful 8051 microcontroller with an intelligent Serial Communication Controller to provide a single-chip solution which will efficiently implement a distributed processing or distributed control system. The microcontroller is a self-sufficient unit containing ROM, RAM, ALU, and its own peripherals. The 8044's architecture and instruction set are identical to the 8051's. The 8044 replaces the 8051's serial interface with an intelligent SDLC/HDLC Serial Interface Unit (SIU). 64 more bytes of RAM have been added to the 8051 RAM array. The SIU can communicate at bit rates up to 2.4 M bps. The SIU works concurrently with the Microcontroller so that there is no throughput loss in either unit. Since the SIU possesses its own intelligence, the CPU is off-loaded from many of the communications tasks, thus dedicating more of its computing power to controlling local peripherals or some external process.

The Microcontroller

The microcontroller is a stand-alone high-performance single-chip computer intended for use in sophisticated real-time application such as instrumentation, industrial control, and intelligent computer peripherals.

The major features of the microcontroller are:

- 8-bit CPU
- on-chip oscillator
- 4K bytes of ROM
- 192 bytes of RAM
- 32 I/O lines
- 64K address space for external Data Memory
- 64K address space for external Program Memory
- two fully programmable 16-bit timer/counters
- a five-source interrupt structure with two priority levels
- bit addressability for Boolean processing
- 1 μsec instruction cycle time for 60% of the instructions 2 μsec instruction cycle time for 40% of the instructions
- 4 μsec cycle time for 8 by 8 bit unsigned Multiply/Divide

Internal Data Memory

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 5.

The Internal Data RAM address space is 0 to 255. Four 8-Register Banks occupy locations 0 through 31. The stack can be located anywhere in the Internal Data RAM address space. In addition, 128 bit locations of the on-chip RAM are accessible through Direct Addressing. These bits reside in Internal Data RAM at byte locations 32 through 47. Currently locations 0 through 191 of the Internal Data RAM address space are filled with on-chip RAM.

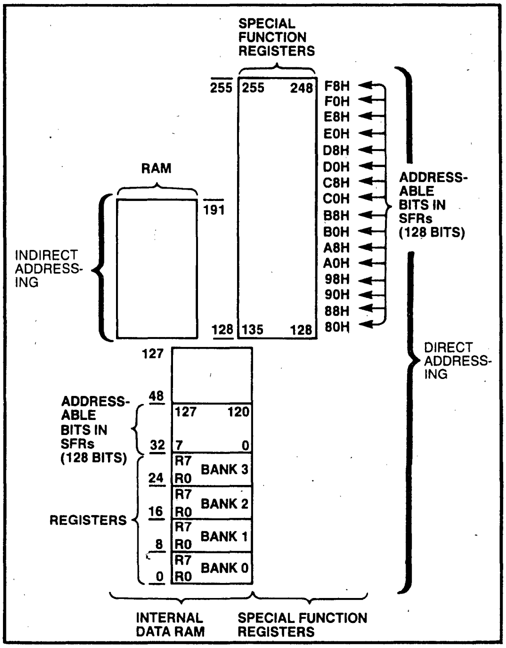


Figure 5. Internal Data Memory Address Space

Parallel I/O

The 8044 has 32 general-purpose I/O lines which are arranged into four groups of eight lines. Each group is called a port. Hence there are four ports; Port 0, Port 1, Port 2, and Port 3. Up to five lines from 1 and Port 2 are dedicated to supporting the serial channel when the SIU is invoked. Due to the nature of the serial port, two of Port 3's I/O lines (P3.0 and P3.1) do not have latched outputs. This is true whether or not the serial channel is used.

Table 1. MCS®-51 Instruction Set Description

ARITHMETIC OPERATIONS

Mnemonic		Description	Byte	Cyc
ADD	A,Rn	Add register to Accumulator	1	1
ADD	A,direct	Add direct byte to Accumulator	2	1
ADD	A,@Ri	Add indirect RAM to Accumulator	1	1
ADD	A,#data	Add immediate data to Accumulator	2	1
ADDC	A,Rn	Add register to Accumulator with Carry	1	1
ADDC	A,direct	Add direct byte to A with Carry flag	2	1
ADDC	A,@Ri	Add indirect RAM to A with Carry flag	1	1
ADDC	A,#data	Add immediate data to A with Carry flag	2	1
SUBB	A,Rn	Subtract register from A with Borrow	1	1
SUBB	A,direct	Subtract direct byte from A with Borrow	2	1
SUBB	A,@Ri	Subtract indirect RAM from A with Borrow	1	1
SUBB	A,#data	Subtract immed data from A with Borrow	2	1
INC	A	Increment Accumulator	1	1
INC	Rn	Increment register	1	1
INC	direct	Increment direct byte	2	1
INC	@Ri	Increment indirect RAM	1	1
INC	DPTR	Increment Data Pointer	1	2
DEC	A	Decrement Accumulator	1	1
DEC	Rn	Decrement register	1	1
DEC	direct	Decrement direct byte	2	1
DEC	@Ri	Decrement indirect RAM	1	1
MUL	AB	Multiply A & B	1	4
DIV	AB	Divide A by B	1	4
DA	A	Decimal Adjust Accumulator	1	1

LOGICAL OPERATIONS

Mnemonic		Destination	Byte	Cyc
ANL	A,Rn	AND register to Accumulator	1	1
ANL	A,direct	AND direct byte to Accumulator	2	1
ANL	A,@Ri	AND indirect RAM to Accumulator	1	1
ANL	A,#data	AND immediate data to Accumulator	2	1
ANL	direct,A	AND Accumulator to direct byte	2	1
ANL	direct,#data	AND immediate data to direct byte	3	2
ORL	A,Rn	OR register to Accumulator	1	1
ORL	A,direct	OR direct byte to Accumulator	2	1

LOGICAL OPERATIONS (CONTINUED)

Mnemonic		Destination	Byte	Cyc
ORL	A,@Ri	OR indirect RAM to Accumulator	1	1
ORL	A,#data	OR immediate data to Accumulator	2	1
ORL	direct,A	OR Accumulator to direct byte	2	1
ORL	direct,#data	OR immediate data to direct byte	3	2
XRL	A,Rn	Exclusive-OR register to Accumulator	1	1
XRL	A,direct	Exclusive-OR direct byte to Accumulator	2	1
XRL	A,@Ri	Exclusive-OR indirect RAM to A	1	1
XRL	A,#data	Exclusive-OR immediate data to A	2	1
XRL	direct,A	Exclusive-OR Accumulator to direct byte	2	1
XRL	direct,#data	Exclusive-OR immediate data to direct	3	2
CLR	A	Clear Accumulator	1	1
CPL	A	Complement Accumulator	1	1
RL	A	Rotate Accumulator Left	1	1
RLC	A	Rotate A Left through the Carry flag	1	1
RR	A	Rotate Accumulator Right	1	1
RRC	A	Rotate A Right through Carry flag	1	1
SWAP	A	Swap nibbles within the Accumulator	1	1

DATA TRANSFER

Mnemonic		Description	Byte	Cyc
MOV	A,Rn	Move register to Accumulator	1	1
MOV	A,direct	Move direct byte to Accumulator	2	1
MOV	A,@Ri	Move indirect RAM to Accumulator	1	1
MOV	A,#data	Mov immediate data to Accumulator	2	1
MOV	Rn,A	Move Accumulator to register	1	1
MOV	Rn,direct	Move direct byte to register	2	2
MOV	Rn,#data	Move immediate data to register	2	1
MOV	direct,A	Move Accumulator to direct byte	2	1
MOV	direct,Rn	Move register to direct byte	2	2
MOV	direct,direct	Move direct byte to direct	3	2
MOV	direct,@Ri	Move indirect RAM to direct byte	2	2

Table 1. (Cont.)

DATA TRANSFER (CONTINUED)

Mnemonic		Description	Byte	Cyc
MOV	direct,#data	Move immediate data to direct byte	3	2
MOV	@Ri,A	Move Accumulator to indirect RAM	1	1
MOV	@Ri,direct	Move direct byte to indirect RAM	2	2
MOV	@Ri,#data	Move immediate data to indirect RAM	2	1
MOV	DPTR,#data16	Load Data Pointer with a 16-bit constant	3	2
MOVC	A,@A+DPTR	Move Code byte relative to DPTR to A	1	2
MOVC	A,@A+PC	Move Code byte relative to PC to A	1	2
MOVX	A,@Ri	Move External RAM (8-bit addr) to A	1	2
MOVX	A,@DPTR	Move External RAM (16-bit addr) to A	1	2
MOVX	@Ri,A	Move A to External RAM (8-bit addr)	1	2
MOVX	@DPTR,A	Move A to External RAM (16-bit addr)	1	2
PUSH	direct	Push direct byte onto stack	2	2
POP	direct	Pop direct byte from stack	2	2
XCH	A,Rn	Exchange register with Accumulator	1	1
XCH	A,direct	Exchange direct byte with Accumulator	2	1
XCH	A,@Ri	Exchange indirect RAM with A	1	1
XCHD	A,@Ri	Exchange low-order Digit ind RAM w A	1	1

BOOLEAN VARIABLE MANIPULATION

Mnemonic		Description	Byte	Cyc
CLR	C	Clear Carry flag	1	1
CLR	bit	Clear direct bit	2	1
SETB	C	Set Carry flag	1	1
SETB	bit	Set direct Bit	2	1
CPL	C	Complement Carry flag	1	1
CPL	bit	Complement direct bit	2	1
ANL	C,bit	AND direct bit to Carry flag	2	2
ANL	C,/bit	AND complement of direct bit to Carry	2	2
ORL	C/bit	OR direct bit to Carry flag	2	2
ORL	C,/bit	OR complement of direct bit to Carry	2	2
MOV	C,/bit	Move direct bit to Carry flag	2	1
MOV	bit,C	Move Carry flag to direct bit	2	2

PROGRAM AND MACHINE CONTROL

Mnemonic		Description	Byte	Cyc
ACALL	addr11	Absolute Subroutine Call	2	2
LCALL	addr16	Long Subroutine Call	3	2
RET		Return from subroutine	1	2
RETI		Return from interrupt	1	2
AJMP	addr11	Absolute Jump	2	2
LJMP	addr16	Long Jump	3	2
SJMP	rel	Short Jump (relative addr)	2	2
JMP	@A+DPTR	Jump indirect relative to the DPTR	1	2
JZ	rel	Jump if Accumulator is Zero	2	2
JNZ	rel	Jump if Accumulator is Not Zero	2	2
JC	rel	Jump if Carry flag is set	2	2
JNC	rel	Jump if No Carry flag	2	2
JB	bit,rel	Jump if direct Bit set	3	2
JNB	bit,rel	Jump if direct Bit Not set	3	2
JBC	bit,rel	Jump if direct Bit is set & Clear bit	3	2
CJNE	A,direct,rel	Compare direct to A & Jump if Not Equal	3	2
CJNE	A,#data,rel	Comp, immed, to A & Jump if Not Equal	3	2
CJNE	Rn,#data,rel	Comp, immed, to reg & Jump if Not Equal	3	2
CJNE	@Ri,#data,rel	Comp, immed, to ind, & Jump if Not Equal	3	2
DJNZ	Rn,rel	Decrement register & Jump if Not Zero	2	2
DJNZ	direct,rel	Decrement direct & Jump if Not Zero	3	2
NOP		No operation	1	1

Notes on data addressing modes:

Rn —Working register R0-R7

direct —128 internal RAM locations, any I/O port, control or status register

@Ri —Indirect internal RAM location addressed by register R0 or R1

#data —8-bit constant included in instruction

#data16 —16-bit constant included as bytes 2 & 3 of instruction

bit —128 software flags, any I/O pin, control or status bit

Notes on program addressing modes:

addr16 —Destination address for LCALL & LJMP may be anywhere within the 64-K program memory address space

Addr11 —Destination address for ACALL & AJMP will be within the same 2-K page of program memory as the first byte of the following instruction

rel —SJMP and all conditional jumps include an 8-bit offset byte, Range is +127–128 bytes relative to first byte of the following instruction

Port 0 and Port 2 also have an alternate dedicated function. When placed in the external access mode, Port 0 and Port 2 become the means by which the 8044 communicates with external program memory. Port 0 and Port 2 are also the means by which the 8044 communicates with external data memory. Peripherals can be memory mapped into the address space and controlled by the 8044.

Timer/Counters

The 8044 contains two 16-bit counters which can be used for measuring time intervals, measuring pulse widths, counting events, generating precise periodic interrupt requests, and clocking the serial communications. Internally the Timers are clocked at 1/12 of the crystal frequency, which is the instruction cycle time. Externally the counters can run up to 500 KHz.

Interrupt System

External events and the real-time driven on-chip peripherals require service by the CPU asynchronous to the execution of any particular section of code. To tie the asynchronous activities of these functions to normal program execution, a sophisticated multiple-source, two priority level, nested interrupt system is provided. Interrupt response latency ranges from 3 μsec to 7 μsec when using a 12 MHz clock.

All five interrupt sources can be mapped into one of the two priority levels. Each interrupt source can be enabled or disabled individually or the entire interrupt system can be enabled or disabled. The five interrupt sources are: Serial Interface Unit, Timer 1, Timer 2, and two external interrupts. The external interrupts can be either level or edge triggered.

Serial Interface Unit (SIU)

The Serial Interface Unit is used for HDLC/SDLC communications. It handles Zero Bit Insertion/Deletion, Flags, automatic address recognition, and a 16-bit cyclic redundancy check. In addition it implements in hardware a subset of the SDLC protocol such that it responds to many SDLC frames without CPU intervention. In certain applications it is advantageous to have the CPU control the reception or transmission of every single frame. For this reason the SIU has two modes of operation: "AUTO" and "FLEXIBLE" (or "NON-AUTO"). It is in the AUTO mode that the SIU responds to SDLC frames without CPU intervention; whereas, in the FLEXIBLE mode the reception or transmission of every single frame will be under CPU control.

There are three control registers and eight parameter registers that are used to operate the serial interface. These registers are shown in Figure 5 and Figure 6. The control registers set the modes of operation and provide status information. The eight parameter registers buffer the station address, receive and transmit control bytes, and point to the on-chip transmit and receive buffers.

Data to be received or transmitted by the SIU must be buffered anywhere within the 192 bytes of on-chip RAM. Transmit and receive buffers are not allowed to "wrap around" in RAM; a "buffer end" is generated after address 191 is reached.

AUTO Mode

In the AUTO mode the SIU implements in hardware a subset of the SDLC protocol such that it responds to many SDLC frames without CPU intervention. All AUTO mode responses to the primary station will conform to IBM's SDLC definition. The advantages of the AUTO mode are that less software is required to implement a secondary station, and the hardware generated response to polls is much faster than doing it in software. However, the Auto mode can not be used at a primary station.

To transmit in the AUTO mode the CPU must load the Transmit Information Buffer, Transmit Buffer Start register, Transmit Buffer Length register, and set the Transmit Buffer Full bit. The SIU automatically responds to a poll by transmitting an information frame with the P/F bit in the control field set. When the SIU receives a positive acknowledgement from the primary station, it automatically increments the Ns field in the NSNR register and interrupts the CPU. A negative acknowledgement would cause the SIU to retransmit the frame.

To receive in the AUTO mode, the CPU loads the Receive Buffer Start register, the Receive Buffer Length register, clears the Receive Buffer Protect bit, and sets the Receive Buffer Empty bit. If the SIU is polled in this state, and the TBF bit indicates that the Transmit Buffer is empty, an automatic RR response will be generated. When a valid information frame is received the SIU will automatically increment Nr in the NSNR register and interrupt the CPU.

While in the AUTO mode the SIU can recognize and respond to the following commands without CPU intervention: I (Information), RR (Receive Ready), RNR (Receive Not Ready), REJ (Reject), and UP (Unnumbered Poll). The SIU can generate

REGISTER NAMES	SYMBOLIC ADDRESS	BIT ADDRESS	BYTE ADDRESS
B REGISTER	B	247 through 240	240 (F0H)
ACCUMULATOR	ACC	231 through 224	224 (E0H)
*THREE BYTE FIFO	FIFO		223 (DFH)
	FIFO		222 (DEH)
	FIFO		221 (DDH)
TRANSMIT BUFFER START	TBS		220 (DCH)
TRANSMIT BUFFER LENGTH	TBL		219 (DBH)
TRANSMIT CONTROL BYTE	TCB		218 (DAH)
* SIU STATE COUNTER	SIUST		217 (D9H)
SEND COUNT RECEIVE COUNT	NSNR	223 through 216	216 (D8H)
PROGRAM STATUS WORD	PSW	215 through 208	208 (D0H)
*DMA COUNT	DMA CNT		207 (CFH)
STATION ADDRESS	STAD		206 (CEH)
RECEIVE FIELD LENGTH	RFL		205 (CDH)
RECEIVE BUFFER START	RBS		204 (CCH)
RECEIVE BUFFER LENGTH	RBL		203 (CBH)
RECEIVE CONTROL BYTE	RCB		202 (CAH)
SERIAL MODE	SMD		201 (C9H)
STATUS REGISTER	STS	207 through 200	200 (C8H)
INTERRUPT PRIORITY CONTROL	IP	191 through 184	184 (B8H)
PORT 3	P3	183 through 176	176 (B0H)
INTERRUPT ENABLE CONTROL	IE	175 through 168	168 (A8H)
PORT 2	P2	167 through 160	160 (A0H)
PORT 1	P1	151 through 144	144 (90H)
TIMER HIGH 1	TH1		141 (8DH)
TIMER HIGH 0	TH0		140 (8CH)
TIMER LOW 1	TL1		139 (8BH)
TIMER LOW 0	TL0		138 (8AH)
TIMER MODE	TMOD		137 (89H)
TIMER CONTROL	TCON	143 through 136	136 (88H)
DATA POINTER HIGH	DPH		131 (83H)
DATA POINTER LOW	DPL		130 (82H)
STACK POINTER	SP		129 (81H)
PORT 0	P0	135 through 128	128 (80H)

SFR's CONTAINING DIRECT ADDRESSABLE BITS

*ICE Support Hardware registers. Under normal operating conditions there is no need for the CPU to access these registers.

Figure 5. Mapping of Special Function registers

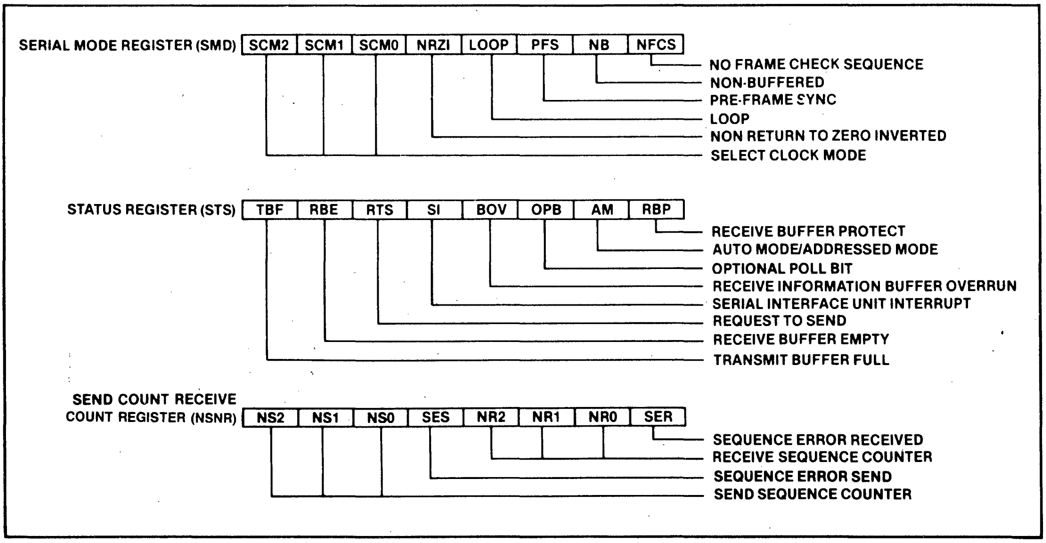


Figure 6. Serial Interface Unit Control registers

the following responses without CPU intervention: I (Information), RR (Receive Ready), and RNR (Receive Not Ready).

When the Receive Buffer Empty bit (RBE) indicates that the Receive Buffer is empty, the receiver is enabled, and when the RBE bit indicates that the Receive Buffer is full, the receiver is disabled. Assuming that the Receive Buffer is empty, the SIU will respond to a poll with an I frame if the Transmit Buffer is full. If the Transmit Buffer is empty, the SIU will respond to a poll with a RR command if the Receive Buffer Protect bit (RBP) is cleared, or an RNR command if RBP is set.

FLEXIBLE (or NON-AUTO) Mode

In the FLEXIBLE mode all communications are under control of the CPU. It is the CPU's task to encode and decode control fields, manage acknowledgements, and adhere to the requirements of the HDLC/SDLC protocols. The 8044 can be used as a primary or a secondary station in this mode.

To receive a frame in the FLEXIBLE mode, the CPU must load the Receive Buffer Start register, the Receive Buffer Length register, clear the Receive Buffer Protect bit, and set the Receive Buffer Empty bit. If a valid opening flag is received and the address field matches the byte in the Station Address register or the address field contains a broadcast address, the 8044 loads the control field in the receive control byte register, and loads the I field in the receive buffer. If there is no CRC error, the SIU interrupts the CPU, indicating a frame has just been received. If there is a CRC error, no interrupt occurs. The Receive Field Length register provides the number of bytes that were received in the information field.

To transmit a frame, the CPU must load the transmit information buffer, the Transmit Buffer Start register, the Transmit Buffer Length register, the Transmit Control Byte, and set the TBF and the RFS bit. The SIU, unsolicited by an HDLC/SDLC frame, will transmit the entire information frame, and interrupt the CPU, indicating the completion of transmission. For supervisory frames or unnumbered frames, the transmit buffer length would be 0.

CRC

The FCS register is initially set to all 1's prior to calculating the FCS field. The SIU will not interrupt the CPU if a CRC error occurs (in both AUTO and FLEXIBLE modes). The CRC error is cleared upon receiving of an opening flag.

Frame Format Options

In addition to the standard SDLC frame format, the 8044 will support the frames displayed in Figure 7. The standard SDLC frame is shown at the top of this figure. For the remaining frames the information field will incorporate the control or address bytes and the frame check sequences; therefore these fields will be stored in the Transmit and Receive buffers. For example, in the non-buffered mode the third byte is treated as the beginning of the information field. In the non-addressed mode, the information field begins after the opening flag. The mode bits to set the frame format options are found in the Serial Mode register and the Status register.

EXTENDED ADDRESSING

To realize an extended control field or an extended address field using the HDLC protocol, the FLEXIBLE mode must be used. For an extended control field, the SIU is programmed to be in the non-buffered mode. The extended control field will be the first and second bytes in the Receive and Transmit Buffers. For extended addressing the SIU is placed in the non-addressed mode. In this mode the CPU must implement the address recognition for received frames. The addressing field will be the initial bytes in the Transmit and Receive buffers followed by the control field.

The SIU can transmit and receive only frames which are multiples of 8 bits. For frames received with other than 8-bit multiples, a CRC error will cause the SIU to reject the frame.

SDLC Loop Networks

The SIU can be used in a ADLC loop as a secondary or primary station. When the SIU is placed in the Loop mode it receives the data on pin 10 and transmits the data one bit time delayed on pin 11. It can also recognize the Go ahead signal and change it into a flag when it is ready to transmit. As a secondary station the SIU can be used in the AUTO or FLEXIBLE modes. As a primary station the FLEXIBLE mode is used; however, additional hardware is required for generating the Go Ahead bit pattern. In the Loop mode the maximum data rate is 1 Mbps clocked or 375 Kbps self-clocked.

SDLC Multidrop Networks

The SIU can be used in a SDLC non-loop configuration as a secondary or primary station. When the SIU is placed in the non-loop mode, data is received and transmitted on pin 11, and pin 10 drives a tri-state buffer. In non-loop mode, modem interface pins, RTS and CTS, become available.

FRAME OPTION	NFCS	NB	AM[1]	FRAME FORMAT
Standard SDLC **NON-AUTO Mode**	0	0	0	F A C I FCS F
Standard SDLC **AUTO Mode**	0	0	1	F A C I FCS F
Non-Buffered Mode **NON-AUTO Mode**	0	1	1	F A I FCS F
Non-Addressed Mode **NON-AUTO Mode**	0	1	0	F I FCS F
No FCS Field **NON-AUTO Mode**	1	0	0	F A C I F
No FCS Field **AUTO Mode**	1	0	1	F A C I F
No FCS Field **Non-Buffered Mode** **NON-AUTO Mode**	1	1	1	F A I F
No FCS Field **Non-Addressed Mode** **NON-AUTO Mode**	1	1	0	F I F

Mode Bits:
AM — "AUTO" Mode/Addressed Mode
NB — Non-Buffered Mode
NFCS — No FCS Field Mode

Key to Abbreviations:

F = Flag (01111110)	**I = Information Field**
A = Address Field	**FCS = Frame Check Sequence**
C = Control Field	

Note 1: The AM bit function is controlled by the NB bit. When NB = 0, AM becomes AUTO mode select, when NB = 1, AM becomes Address mode select.

Figure 7. Frame Format Options

Data Clocking Options

The 8044's serial port can operate in an externally clocked or self clocked system. A clocked system provides to the 8044 a clock synchronization to the data. A self-clocked system uses the 8044's on-chip Digital Phase Locked Loop (DPLL) to recover the clock from the data, and clock this data into the Serial Receive Shift Register.

In this mode, a clock synchronized with the data is externally fed into the 8044. This clock may be generated from an External Phase Locked Loop, or possibly supplied along with the data. The 8044 can transmit and receive data in this mode at rates up to 2.4 Mbps.

This self clocked mode allows data transfer without a common system data clock. An on-chip Digital Phase Locked Loop is employed to recover the data clock which is encoded in the data stream. The DPLL will converge to the nominal bit center within eight bit transitions, worst case. The DPLL requires a reference clock of either 16 times (16x) or 32 times (32x) the data rate. This reference clock may be externally applied or internally generated. When internally generated either the 8044's internal logic clock (crystal frequency divided by two) or the timer 1 overflow is used as the reference clock. Using the internal timer 1 clock the data reates can vary from 244 to 62.5 Kbps. Using the internal logic clock at a 16x sampling rate, receive data can either be 187.5 Kbps, or 375 Kbps. When the reference clock for the DPLL is externally applied the data rates can vary from 0 to 375 Kbps at a 16x sampling rate.

To aid in a Phase Locked Loop capture, the SIU has a NRZI (Non Return to Zero Inverted) data encoding and decoding option. Additionally the SIU has a preframe sync option that transmits two bytes of alternating 1's and 0's to ensure that the receive station DPLL will be synchronized with the data by the time it receives the opening flag.

Control and Status Registers

There are three SIU Control and Status Registers:

Serial Mode Register (SMD)

Status/Command Register (STS)

Send/Receive Count Register (NSNR)

The SMD, STS, and NSNR registers are all cleared by system reset. This assures that the SIU will power up in an idle state (neither receiving nor transmitting).

These registers and their bit assignments are described below.

SMD: Serial Mode Register (byte-addressable)

Bit: 7	6	5	4	3	2	1	0
SCM2	SCM1	SCM0	NRZI	LOOP	PFS	NB	NFCS

The Serial Mode Register (Address C9H) selects the operational modes of the SIU. The 8044 CPU can both read and write SMD. The SIU can read SMD but cannot write to it. To prevent conflict between CPU and SIU access to SMD, the CPU should write SMD only when the Request To Send (RTS) and Receive Buffer Empty (RBE) bits (in the STS register) are both false (0). Normally, SMD is accessed only during initialization.

The individual bits of the Serial Mode Register are as follows:

Bit #	Name	Description
SMD.0	NFCS	No FCS field in the SDLC frame.
SMD.1	NB	Non-Buffered mode. No control field in the SDLC frame.
SMD.2	PFS	Pre-Frame Sync mode. In this mode, the 8044 transmits two bytes before the first flag of a frame, for DPLL synchronization. If NRZI is enabled, 00H is sent; otherwise, 55H is sent. In either case, 16 pre-frame transitions are guaranteed.
SMD.3	LOOP	Loop configuration.
SMD.4	NRZI	NRZI coding option.
SMD.5	SCM0	Select Clock Mode — Bit 0
SMD.6	SCM1	Select Clock Mode — Bit 1
SMD.7	SCM2	Select Clock Mode — Bit 2

The SCM bits decode as follows:

SCM 2	1	0	Clock Mode	Data Rate (Bits/sec)*
0	0	0	Externally clocked	0-2.4M**
0	0	1	Reserved	
0	1	0	Self clocked, timer overflow	244-62.5K
0	1	1	Reserved	
1	0	0	Self clocked, external 16x	0-375K
1	0	1	Self clocked, external 32x	0-187.5K
1	1	0	Self clocked, internal fixed	375K
1	1	1	Self clocked, internal fixed	187.5k

*Based on a 12 Mhz crystal frequency
**0-1M bps in loop configuration

STS: Status/Command Register (bit-addressable)

Bit: 7	6	5	4	3	2	1	0
TBF	RBE	RTS	SI	BOV	OPB	AM	RBP

The Status/Command Register (Address C8H) provides operational control of the SIU by the 8044 CPU, and enables the SIU to post status information for the CPU's access. The SIU can read STS, and can alter certain bits, as indicated below. The CPU can both read and write STS asynchronously. However, 2-cycle instructions that access STS during both cycles ('JBC/B, REL' and 'MOV /B,C.') should not be used, since the SIU may write to STS between the two CPU accesses.

The individual bits of the Status/Command Register are as follows:

Bit #	Name	Description
STS.0	RBP	Receive Buffer Protect. Inhibits writing of data into the receive buffer. In AUTO mode, RBP forces an RNR response instead of an RR.
STS.1	AM	AUTO Mode/Addressed Mode. Selects AUTO mode where AUTO mode is allowed. If NB is true, (=1), the AM bit selects the addressed mode. AM may be cleared by the SIU.
STS.2	OPB	Optional Poll Bit. Determines whether the SIU will generate an AUTO response to an optional poll (UP with P=0). OPB may be set or cleared by the SIU.
STS.3	BOV	Receive Buffer Overrun. BOV may be set or cleared by the SIU.
STS.4	SI	SIU Interrupt. This is one of the five interrupt sources to the CPU. The vector location = 23H. SI may be set by the SIU. It should be cleared by the CPU before returning from an interrupt routine.
STS.5	RTS	Request To Send. Indicates that the 8044 is ready to transmit or is transmitting. RTS may be read or written by the CPU. RTS may be read by the SIU, and in AUTO mode may be written by the SIU.

STS.6	RBE	Receive Buffer Empty. RBE can be thought of as Receive Enable. RBE is set to one by the CPU when it is ready to receive a frame, or has just read the buffer, and to zero by the SIU when a frame has been received.
STS.7	TBF	Transmit Buffer Full. Written by the CPU to indicate that it has filled the transmit buffer. TBF may be cleared by the SIU.

NSNR: Send/Receive Count Register (bit-addressable)

Bit:	7	6	5	4	3	2	1	0
	NS2	NS1	NS0	SES	NR2	NR1	NR0	SER

The Send/Receive Count Register (Address D8H) contains the transmit and receive sequence numbers, plus tally error indications. The SIU can both read and write NSNR. The 8044 CPU can both read and write NSNR asynchronously. However, 2-cycle instructions that access NSNR during both cycles ('JBC /B, REL', and 'MOV /B,C') should not be used, since the SIU may write to NSNR between the two 8044 CPU accesses.

The individual bits of the Send/Receive Count Register are as follows:

Bit #	Name	Description
NSNR.0	SER	Receive Sequence Error: NS (P) ≠ NR (S)
NSNR.1	NR0	Receive Sequence Counter—Bit 0
NSNR.2	NR1	Receive Sequence Counter—Bit 1
NSNR.3	NR2	Receive Sequence Counter—Bit 2
NSNR.4	SES	Send Sequence Error: NP (P) ≠ NS (S) and NP (P) ≠ NS (S) + 1
NSNR.5	NS0	Send Sequence Counter — Bit 0
NSNR.6	NS1	Send Sequence Counter — Bit 1
NSNR.7	NS2	Send Sequence Counter — Bit 2

Parameter Registers

There are eight parameter registers that are used in connection with SIU operation. All eight registers may be read or written by the 8044 CPU. RFL and RCB are normally loaded by the SIU.

The eight parameter registers are as follows:

STAD: Station Address Register (byte-addressable)

The Station Address register (Address CEH) contains the station address. To prevent access conflict, the CPU should access STAD only when the SIU is idle (RTS=0 and RBE=0). Normally, STAD is accessed only during initialization.

TBS: Transmit Buffer Start Address Register (byte-addressable)

The Transmit Buffer Start address register (Address DCH) points to the location in on-chip RAM for the beginning of the I-field of the frame to be transmitted. The CPU should access TBS only when the SIU is not transmitting a frame (when TBF=0).

TBL: Transmit Buffer Length Register (byte-addressable)

The Transmit Buffer Length register (Address DBH) contains the length (in bytes) of the I-field to be transmitted. A blank I-field (TBL=0) is valid. The CPU should access TBL only when the SIU is not transmitting a frame (when TBF=0).

NOTE: The transmit and recieve buffers are not allowed to "wrap around" in the on-chip RAM. A "buffer end" is automatically generated if address 191 (BFH) is reached.

TCB: Transmit Control Byte Register (byte-addressable)

The Transmit Control Byte register (Address DAH) contains the byte which is to be placed in the control field of the transmitted frame, during NON-AUTO mode transmission. The CPU should access TCB only when the SIU is not transmitting a frame (when TBF=0). The N_S and N_R counters are not used in the NON-AUTO mode.

RBS: Receive Buffer Start Address Register (byte-addressable)

The Receive Buffer Start address register (Address CCH) points to the location in on-chip RAM where the beginning of the I-field of the frame being received is to be stored. The CPU should write RBS only when the SIU is not receiving a frame (when RBE=0).

RBL: Receive Buffer Length Register (byte-addressable)

The Receive Buffer Length register (Address CBH) contains the length (in bytes) of the area in on-chip RAM allocated for the received I-field. RBL=0 is valid. The CPU should write RBL only when RBE=0.

RFL: Receive Field Length Register (byte-addressable)

The Received Field Length register (Address CDH) contains the length (in bytes) of the received I -field that has just been loaded into on-chip RAM. RFL is loaded by the SIU. RFL=0 is valid. RFL should be accesssed by the CPU only when RBE=0.

RCB: Receive Control Byte Register (byte-addressable)

The Received Control Byte register (Address CAH) contains the control field of the frame that has just been received. RCB is loaded by the SIU. The CPU can only read RCB, and should only access RCB when RBE=0.

ICE Support Registers

The 8044 In-Circuit Emulator (ICE-44) allows the user to exercise the 8044 application system and monitor the execution of instructions in real time.

The emulator operates with Intel's Intellec® development system. The development system interfaces with the user's 8044 system through an in-cable buffer box. The cable terminates in a 8044 pin-compatible plug, which fits into the 8044 socket in the user's system. With the emulator plug in place, the user can excercise his system in real time while collecting up to 255 instruction cycles of real-time data. In addition, he can single-step the program.

Static RAM is available (in the in-cable buffer box) to emulate the 8044 internal and external program memory and external data memory. The designer can display and alter the contents of the replacement memory in the buffer box, the internal data memory, and the internal 8044 registers, including the SFRs.

SIUST: SIU State Counter (byte-addressable)

The SIU State Counter (Address D9H) reflects the state of the internal logic which is under SIU control. Therefore, care must be taken not to write into this register. This register provides a useful means for debugging 8044 receiver problem.

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias .... 0 to 70°C
Storage Temperature .... −65°C to +150°C
Voltage on Any Pin With
Respect to Ground (Vss) .... −0.5V to +7V
Power Dissipation .... 2 Watts

*Notice: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

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

Symbol	Parameter	Min	Typ	Max	Units	Test Conditions
VIL	Input Low Voltage	−0.5		0.8	V	
VIH	Input High Voltage (Except RST/VPD and XTAL2)	2.0		VCC + 0.5	V	
VIH1	Input High Voltage To RST/VPD For Reset, XTAL2	2.5			V	XTAL1 to VSS
VPD	Power Down Voltage To RST/VPD	4.5		5.5	V	VCC = 0V
VOL	Output Low Voltage Ports 1, 2, 3 (Note 1)			0.45	V	IOL = 1.6mA
VOL1	Output Low Voltage Port 0 ALE, \$\overline{\text{PSEN}}$ (Note 1)			0.45	V	IOL = 3.2mA
VOH	Outpt High Voltage Ports 1, 2, 3	2.4			V	IOH = −80μA
VOH1	Output High Voltage Port 0, ALE, \$\overline{\text{PSEN}}$	2.4			V	IOH = −400μA
IIL	Logical 0 Input Current Ports 1, 2, 3			−800	μA	XTAL1 at VSS VIL = 0.45V
IIH1	Input High Current,T0 RST/VPD For Reset			500	μA	Vin = VCC − 1.5V
ILI	Input Leakage Current To Port 0, \$\overline{\text{EA}}$			10	μA	0.45V < Vin < VCC
ICC	Power Supply Current		125	200	mA	TA = 25°C
IPD	Power Down Current		15	30	mA	
CIO	Capacitance of $\overline{\text{I/O}}$ Buffer			10	pF	fc = 1MHz
IIL2	Logical 0 Input Current XTAL 2			−2.5	mA	XTAL1 = VSS VIL = 0.45V

Note 1: VOL is degraded when the RUPI-44 rapidly discharges external capacitance. This A.C. noise is most pronounced during emission of address data. When using external memory, locate the latch or buffer as close to the RUPI-44 as possible.

Datum	Emitting Ports	Degraded I/O Lines	VOL (peak) (max)
Address	P2, P0	P1, P3	.8V
Write Data	P0	P1, P3, ALE	.8V

A.C. CHARACTERISTICS (TA 0°C to 70°C, VCC = 5V ± 10%, VSS = OV, CL for Port 0, ALE and $\overline{\text{PSEN}}$ Outputs = 100pF; CL for All Other Outputs = 80 pF)

Program Memory

		12 MHz Clock			Variable Clock 1/TCLCL = 1.2 MHz to 12 MHz		
Symbol	**Parameter**	**Min**	**Max**	**Units**	**Min**	**Max**	**Units**
TLHLL	ALE Pulse Width	127		ns	2TCLCL-40		ns
TAVLL	Address Setup to ALE	43		ns	TCLCL-30		ns
TLLAX[1]	Address Hold After ALE	48		ns	TCLCL-35		ns
TLLIV	ALE To Valid Instr In		233	ns		4TCLCL-100	ns
TLLPL	ALE To $\overline{\text{PSEN}}$	58		ns	TCLCL-25		ns
TPLPH	$\overline{\text{PSEN}}$ Pulse Width	215		ns	3TCLCL-35		ns
TPLIV	$\overline{\text{PSEN}}$ To Valid Instr In		125	ns		3TCLCL-125	ns
TPXIX	Input Instr Hold After $\overline{\text{PSEN}}$	0		ns	0		ns
TPXIZ[2]	Input Instr Float After $\overline{\text{PSEN}}$		63	ns		TCLCL-20	ns
TPXAV[2]	Address Valid After $\overline{\text{PSEN}}$	75		ns	TCLCL-8		ns
TAVIV	Address To Valid Instr In		302	ns		5TCLCL-115	ns
TAZPL	Address Float To $\overline{\text{PSEN}}$	0		ns	0		ns

Notes:

1. TLLAX for access to program memory is different from TLLAX for data memory.
2. Interfacing RUPI-44 devices with float times up to 75ns is permissible. This limited bus contention will not cause any damage to Port 0 drivers.

External Data Memory

		12 MHz Clock			Variable Clock 1/TCLCL = 1.2 MHz to 12 MHz		
Symbol	**Parameter**	**Min**	**Max**	**Units**	**Min**	**Max**	**Units**
TRLRH	$\overline{\text{RD}}$ Pulse Width	400		ns	6TCLCL-100		ns
TWLWH	$\overline{\text{WR}}$ Pulse Width	400		ns	6TCLCL-100		ns
TLLAX[1]	Address Hold After ALE	48		ns	TCLCL-35		
TRLDV	$\overline{\text{RD}}$ To Valid Data In		250	ns		5TCLCL-165	ns
TRHDX	Data Hold After $\overline{\text{RD}}$	0		ns	0		ns
TRHDZ	Data Float After $\overline{\text{RD}}$		97	ns		2TCLCL-70	ns
TLLDV	ALE To Valid Data In		517	ns		8TCLCL-150	ns
TAVDV	Address To Valid Data In		585	ns		9TCLCL-165	ns
TLLWL	ALE To $\overline{\text{WR}}$ or $\overline{\text{RD}}$	200	300	ns	3TCLCL-50	3TCLCL + 50	ns
TAVWL	Address To $\overline{\text{WR}}$ or $\overline{\text{RD}}$	203		ns	4TCLCL-130		ns
TWHLH	$\overline{\text{WR}}$ or $\overline{\text{RD}}$ High To ALE High	43	123	ns	TCLCL-40	TCLCL + 40	ns
TDVWX	Data Valid To $\overline{\text{WR}}$ Transition	33		ns	TCLCL-50		ns
TQVWH	Data Setup Before $\overline{\text{WR}}$	433		ns	7TCLCL-150		ns
TWHQX	Data Hold After $\overline{\text{WR}}$	33		ns	TCLCL-50		ns
TRLAZ	Address Float After $\overline{\text{RD}}$		0	ns		0	ns

Note 1. TLLAX for access to program memory is different from TLLAX for access data memory.

Serial Interface

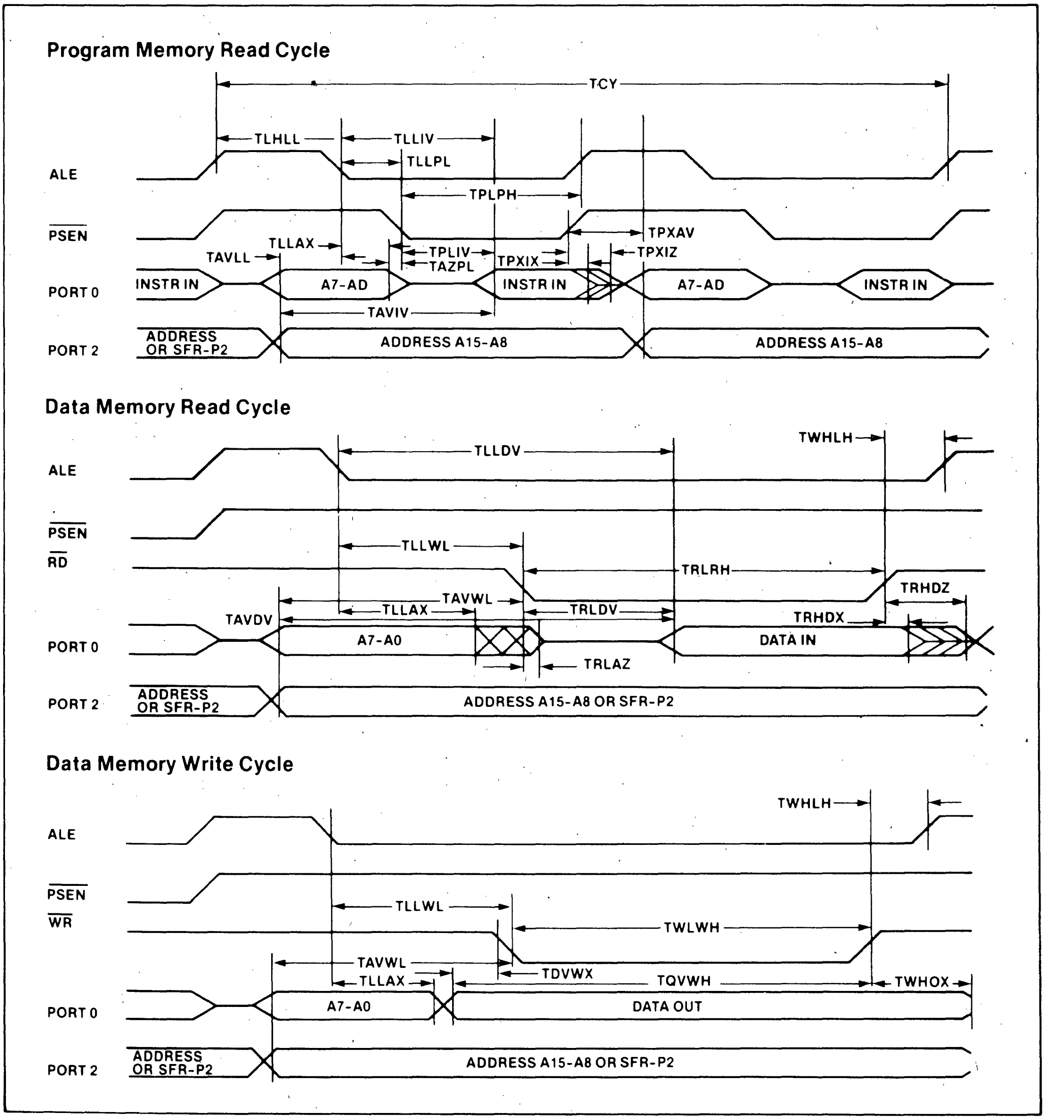
Symbol	Parameter	Min	Max	Units	
TDCY	Data Clock	420		ns	
TDCL	Data Clock Low	180		ns	
TDCH	Data Clock High	120		ns	

Serial Interface (Continued)

tTD	Transmit Data Delay		80	ns	
tDSS	Data Setup Time	60		ns	
tDHS	Data Hold Time	40		ns	

WAVEFORMS

Memory Access

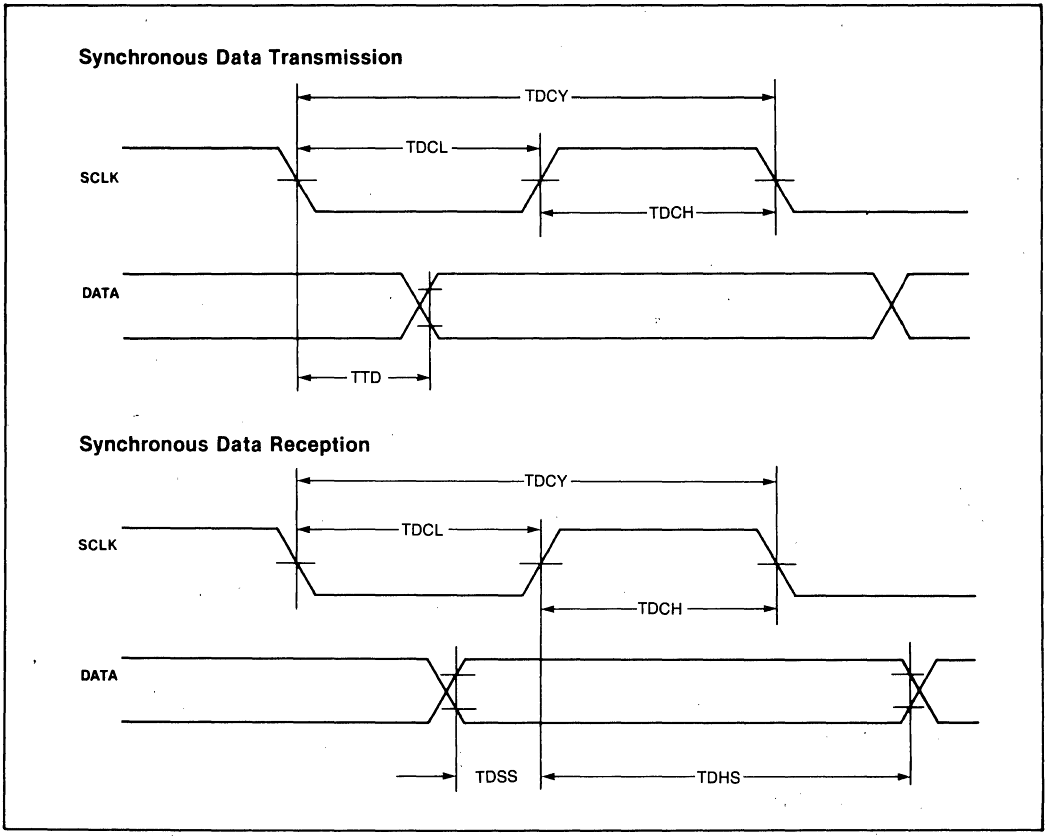


Program Memory Read Cycle

Data Memory Read Cycle

Data Memory Write Cycle

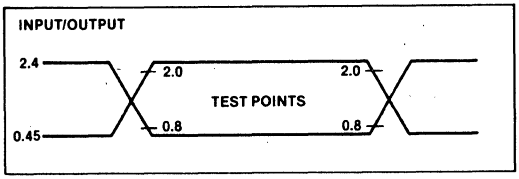
SERIAL I/O WAVEFORMS

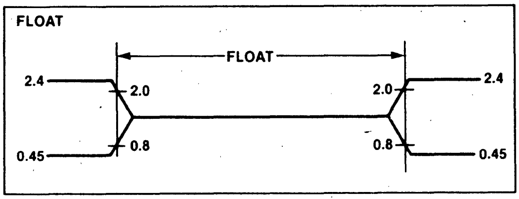


Synchronous Data Transmission

Synchronous Data Reception

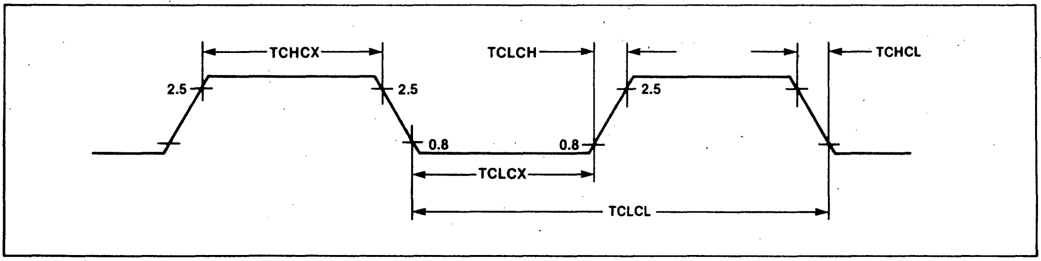
AC TESTING INPUT, OUTPUT, FLOAT WAVEFORMS





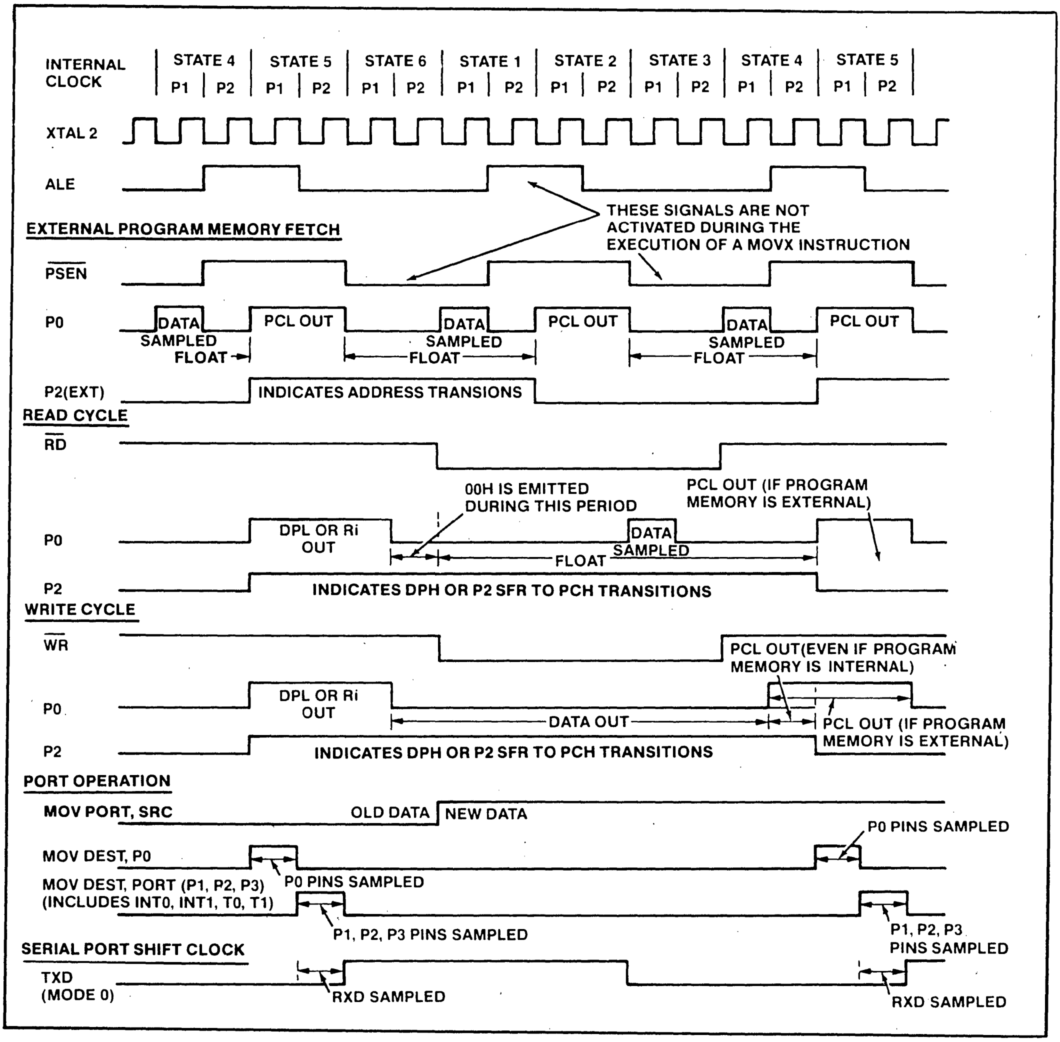
AC testing inputs are driven at 2.4V for a logic "1" and 0.45V for a logic "0."
Timing measurements are made at 2.0V for a logic "1" and 0.8V for a logic "0."

EXTERNAL CLOCK DRIVE XTAL2



Symbol	Parameter	Variable Clock Freq = 1.2 MHz to 12 MHz Min	 Max	Unit
TCLCL	Oscillator Period	83.3	833.3	ns
TCHCX	High Time	20	TCLCL-TCLCX	ns
TCLCX	Low Time	20	TCLCL-TCHCX	ns
TCLCH	Rise Time		20	ns
TCHCL	Fall Time		20	ns

CLOCK WAVEFORMS



This diagram indicates when signals are clocked internally. The time it takes the signals to propagate to the pins, however, ranges from 25 to 125 ns. This propagation delay is dependent on variables such as temperature and pin loading. Propagation also varies from output to output and component to component. Typically though, (T_A = 25° C, fully loaded) RD and WR propagation delays are approximately 50 ns. The other signals are typically 85 ns. Propagation delays are incorporated in the AC specifications.

8744
HIGH PERFORMANCE 8-BIT MICROCONTROLLER WITH On-CHIP SERIAL COMMUNICATION CENTER

8044AH — CPU/SIU with Factory Mask Programmable ROM
8344AH — An 8044AH used with External Program Memory
8744H — An 8044AH with User Programmable/Erasable EPROM

8051 MICROCONTROLLER CORE

- **Optimized for Real Time Control 12 MHz Clock, Priority Interrupts, 32 Programmable I/O lines, Two 16-bit Timer/Counters**
- **Boolean Processor**
- **4K x 8 EPROM, 192 x 8 RAM**
- **64K Accessible External Program Memory**
- **64K Accessible External Data Memory**
- **4 μs Multiply and Divide**

SERIAL INTERFACE UNIT (SIU)

- **Serial Communication Processor that Operates Concurrently to C.P.U.**
- **2.4 Mbps Maximum Data Rate**
- **375 Kbps using On-Chip Phase Locked Loop**
- **Communication Software in Silicon:**
  - **— Complete Data Link Functions**
  - **— Automatic Station Responses**
- **Operates as an SDLC Primary or Secondary Station**

The RUPI-44 family integrates a high performance 8-bit Microcontroller, the Intel 8051 Core, with an intelligent/high performance HDLC/SDLC serial communication controller, called the Serial Interface Unit (SIU). See Figure 1. This dual architecture allows complex control and high speed data communication functions to be realized cost effectively.

Specifically, the 8044's Microcontroller features: 4K byte On-Chip program memory space; 32 I/O lines; two 16-bit timer/event counters; a 5-source; 2-level interrupt structure; a full duplex serial channel; a Boolean processor; and on-chip oscillator and clock circuitry. Standard TTL and most byte-oriented MCS-80 and MCS-85 peripherals can be used for I/O and memory expansion.

The Serial Interface Unit (SIU) manages the interface to a high speed serial link. The SIU offloads the On-Chip 8051 Microcontroller of communication tasks, thereby freeing the CPU to concentrate on real time control tasks.

The RUPI-44 family consists of the 8044, 8744, and 8344. All three devices are identical except in respect of on-chip program memory. The 8044 contains 4K bytes of mask-programmable ROM. User programmable EPROM replaces ROM in the 8744. The 8344 addresses all program memory externally.

The RUPI-44 devices are fabricated with Intel's reliable +5 volt, silicon-gate HMOSII technology and packaged in a 40-pin DIP.

8044's Dual Controller Architecture

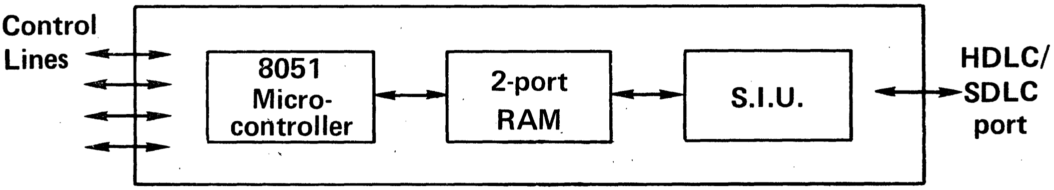


Figure 1. Dual Controller Architecture

Intel Corporation Assumes No Responsibility for the Use of Any Circuitry Other Than Circuitry Embodied in an Intel Product. No Other Circuit Patent Licenses are Implied.

Table 1. RUPI™-44 Family Pin Description

VSS
Circuit ground potential.

VCC
+5V power supply during operation and program verification.

PORT 0
Port 0 is an 8-bit open drain bidirectional I/O port. It is also the multiplexed low-order address and data bus when using external memory. It is used for data output during program verification. Port 0 can sink/source eight LS TTL loads.

PORT 1
Port 1 is an 8-bit quasi-bidirectional I/O port. It is used for the low-order address byte during program verification. Port 1 can sink/source four LS TTL loads.

In non-loop mode two of the I/O lines serve alternate functions:

—$\overline{RTS}$ (P1.6). Request-to-Send output. A low indicates that the RUPI-44 is ready to transmit.
—$\overline{CTS}$ (P1.7) Clear-to-Send input. A low indicates that a receiving station is ready to receive.

PORT 2
Port 2 is an 8-bit quasi-bidirection I/O port. It also emits the high-order address byte when accessing external memory. It is used for the high-order address and the control signals during program verification. Port 2 can sink/source four LS TTL loads.

PORT 3
Port 3 is an 8-bit quasi-bidirectional I/O port. 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 secondary function must be programmed to a one (1) for that function to operate. Port 3 can sink/source LS TTL loads.

In addition to I/O, some of the pins also serve alternate functions as follows:

—I/$\overline{O}$ RxD (P3.0). In point-to-point or multipoint configurations, this pin controls the direction of pin P3.1. Serves as Receive Data input in loop and diagnostic modes.
—DATA TxD (P3.1) In point-to-point or multipoint configurations, this pin functions as data input/output. In loop mode, it serves as transmit pin. A '0' written to this pin enables diagnostic mode.
—$\overline{INT0}$ (P3.2). Interrupt 0 input or gate control input for counter 0.
—$\overline{INT1}$ (P3.3). Interrupt 1 input or gate control input for counter 1.
—TO(P3.4). Input to counter 0.
—SCLK T1 (P3.5). In addition to I/O, this pin provides input to counter 1 or serves as SCLK (serial clock) input.
—$\overline{WR}$ (P3.6). The write control signal latches the data byte from Port 0 into the External Data Memory.
—$\overline{RD}$ (P3.7). The read control signal enables External Data Memory to Port 0.

RST/VPD
A high level on this pin resets the RUPI-44. A small internal pulldown resistor permits power-on reset using only a capacitor connected to VCC. If VPD is held within its spec while VCC drops below spec, VPD will provide standby power to the RAM. When VPD is low, the RAM's current is drawn from VCC.

ALE/$\overline{PROG}$
Provides Address Latch Enable output used for latching the address into external memory during normal operation. It is activitated every six oscillator periods except during an external data memory access. It also receives the program pulse input for programming the EPROM version.

$\overline{PSEN}$
The Program Store Enable output is a control signal that enables the external Program Memory to the bus during external fetch operations. It is activated every six oscillator periods, except during external data memory accesses. Remains high during internal program execution.

$\overline{EA}$/VPP
When held at a TTL high level, the RUPI-44 executes instructions from the internal ROM when the PC is less than 4096. When held at a TTL low level, the RUPI-44 fetches all instructions from external Program Memory. The pin also receives the 21V EPROM programming supply voltage on the 8744.

XTAL 1
Input to the oscillator's high gain amplifier. Required when a crystal is used. Connect to VSS when external source is used on XTAL 2.

XTAL 2
Output from the oscillator's amplifier. Input to the internal timing circuitry. A crystal or external source can be used.

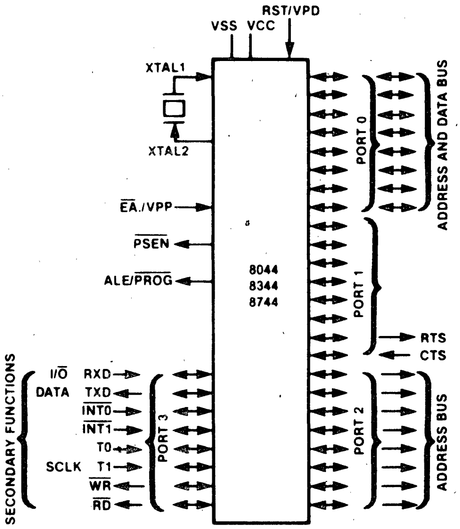


Figure 2.
Logic Symbol

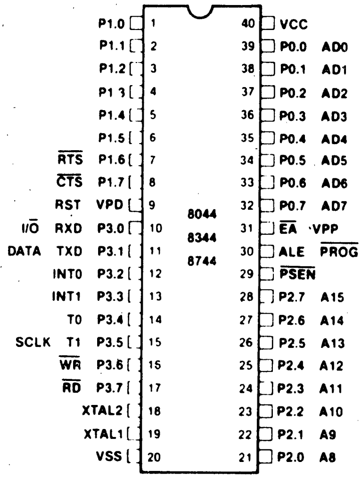


Figure 3. Pin
Configuration

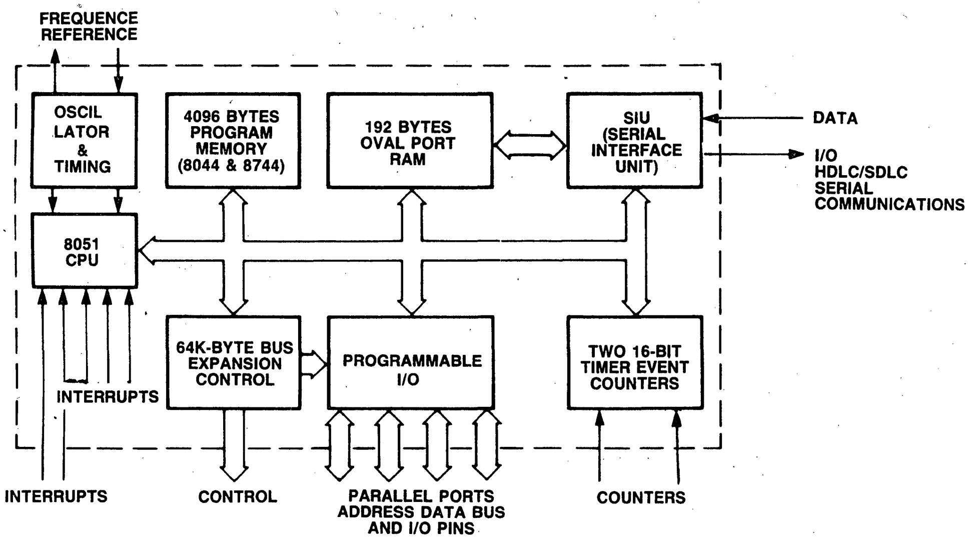


Figure 4.
Block Diagram

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias . . . 0°C to 70°C

Storage Temperature . . . . -65°C to +150°C

Voltage On Any Pin to V_{SS}
(Except V_{PP}) . . . . -0.5V to +7V

Voltage from V_{PP} to V_{SS} 21.5V

Power Dissipation . . . . 2 W

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS: (T_A = 0°C to 70°C; V_{CC} = 4.5V to 5.5V, V_{SS} = 0V)

Symbol	Parameter	Min	Max	Unit	Test Conditions
VIL	Input Low Voltage	-0.5	0.8	V	
VIH	Input High Voltage (Except XTAL2, RST)	2.0	V_{CC} + 0.5	V	
VIH1	Input High Voltage to XTAL2, RST	2.5	V_{CC} + 0.5	V	XTAL1 = V_{SS}
VPD	Power Down Voltage to RST/V_{PD}	4.5	5.5	V	V_{CC} = 0V
VOL	Output Low Voltage Ports 1, 2, 3 (Note 1)		0.45	V	IOL = 1.6mA
VOL1	Output Low Voltage Port 0, ALE, $\overline{PSEN}$ (Note 1)		0.60 0.45	V V	IOL = 3.2mA IOL = 2.4mA
VOH	Output High Voltage Ports 1, 2, 3	2.4		V	IOH = -80μA
VOH1	Output High Voltage Port 0 (in External Bus Mode), ALE, $\overline{PSEN}$	2.4		V	IOH = -400μA
IIL	Logical 0 Input Current P1, P2, P3		500	μA	Vin = 0.45V
IIL1	Logical 0 Input Current to $\overline{EA}$/V_{PP}		-15	mA	Vin = 0.45V
IIL2	Logical 0 Input Current to XTAL2		-2.5	mA	XTAL1 = V_{SS} Vin = 0.45V
ILI	Input Leakage Current to Port 0		100	μA	0.45 < Vin < V_{CC}
IIH	Logical Input Current to $\overline{EA}$/V_{PP}		500	μA	Vin = 2.4V
IIH1	Input Current to RST/V_{PD} to activate reset		500	μA	Vin < (V_{CC} - 1.5V)
ICC	Power Supply Current		250	mA	All outputs disconnected, $\overline{EA}$ = V_{CC}
IPD	Power Down Current to RST/V_{PD}		10	mA	V_{CC} = 0V V_{PD} = 5V
CIO	Capacitance of I/O Buffers		10	pF	fc = 1 MHz T_A = 25°C

AC CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 4.5V to 5.5V, V_{SS} = 0V, Load Capacitance for Port 0, ALE, and $\overline{\text{PSEN}}$ = 100 pF; Load Capacitance for all other outputs = 80 pF)

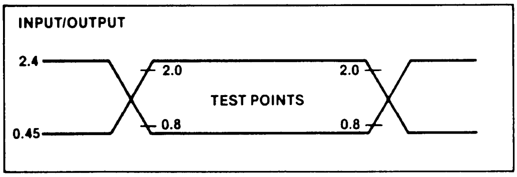
EXTERNAL PROGRAM MEMORY CHARACTERISTICS

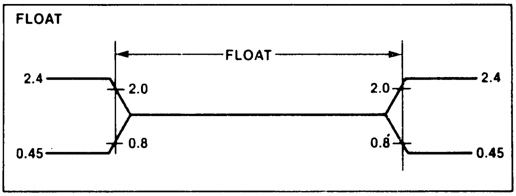
Symbol	Parameter	8744H 12 MHz Osc Min	8744H 12 MHz Osc Max	Variable Oscillator Min	Variable Oscillator Max	Units
TCLCL	Oscillator Period			83.3	833.3	ns
TLHLL	ALE Pulse Width	127		2TCLCL-40		ns
TAVLL	Address Valid to ALE	53		TCLCL-40		ns
TLLAX	Address Hold after ALE	48		TCLCL-35		ns
TLLIV	ALE to Valid Instr In		183		4TCLCL-150	ns
TLLPL	ALE to $\overline{\text{PSEN}}$	58		TCLCL-25		ns
TPLPH	$\overline{\text{PSEN}}$ Pulse Width	190		3TCLCL-60		ns
TPLIV	$\overline{\text{PSEN}}$ to Valid Instr In		100		3TCLCL-150	ns
TPXIX	Input Instr Hold after $\overline{\text{PSEN}}$	0		0		ns
TPXIZ	Input Instr Float after $\overline{\text{PSEN}}$		63		TCLCL-20	ns
TPXAV	$\overline{\text{PSEN}}$ to Address Valid	75		TCLCL-8		ns
TAVIV	Address to Valid Instr In		267		5TCLCL-150	ns
TAZPL	Address Float to $\overline{\text{PSEN}}$	0		0		ns

EXTERNAL DATA MEMORY CHARACTERISTICS

Symbol	Parameter	8744H 12 MHz Osc Min	8744H 12 MHz Osc Max	Variable Oscillator Min	Variable Oscillator Max	Units
TRLRH	$\overline{\text{RD}}$ Pulse Width	400		6TCLCL-100		ns
TWLWH	$\overline{\text{WR}}$ Pulse Width	400		6TCLCL-100		ns
TLLAX	Address Hold After ALE	48		TCLCL-35		ns
TRLDV	$\overline{\text{RD}}$ to Valid Data In		252		5TCLCL-165	ns
TRHDX	Data Hold after $\overline{\text{RD}}$	0		0		ns
TRHDZ	Data Float after $\overline{\text{RD}}$		97		2TCLCL-70	ns
TLLDV	ALE to Valid Data In		517		8TCLCL-150	ns
TAVDV	Address to Valid Data In		585		9TCLCL-165	ns
TLLWL	ALE to $\overline{\text{WR}}$ or $\overline{\text{RD}}$	200	300	3TCLCL-50	3TCLCL+50	ns
TAVWL	Address to $\overline{\text{WR}}$ or $\overline{\text{RD}}$	203		4TCLCL-130		ns
TQVWX	Data Valid to $\overline{\text{WR}}$ Transition	13		TCLCL-70		ns
TQVWH	Data Setup to $\overline{\text{WR}}$ High	433		7TCLCL-150		ns
TWHQX	Data Held after $\overline{\text{WR}}$	33		TCLCL-50		ns
TRLAZ	$\overline{\text{RD}}$ Low to Address Float		0		0	ns
TWHLH	$\overline{\text{RD}}$ or WR High to ALE High	33	133	TCLCL-50	TCLCL+50	ns

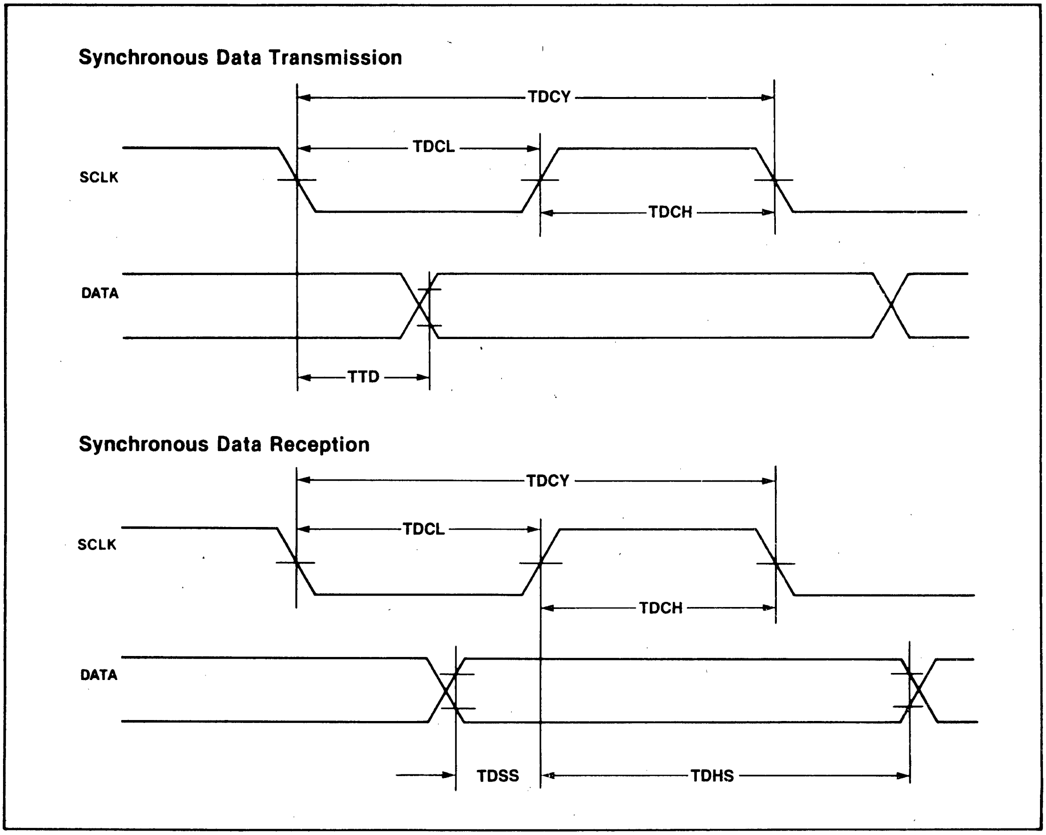
AC TESTING INPUT, OUTPUT, FLOAT WAVEFORMS





AC testing inputs are driven at 2.4V for a logic 1 and 0.45V for a logic 0
Timing measurements are made at 2.0V for a logic 1 and 0.8V for a logic 0
For timing purposes, the float state is defined as the point at which a P0 pin sinks 3.2mA or sources 400μA at the voltage test levels

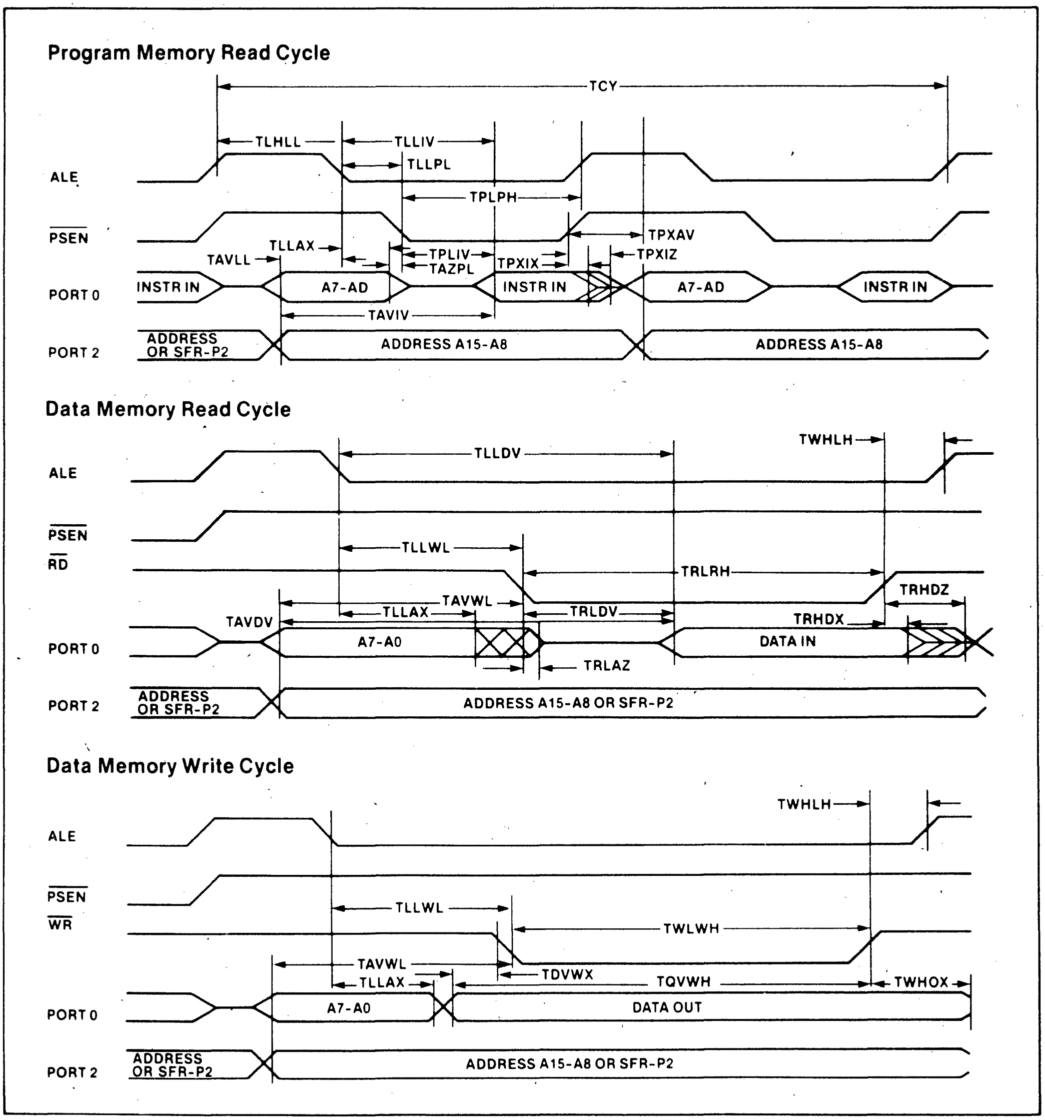
SERIAL I/O WAVEFORMS



Serial Interface

Symbol	Parameter	Min	Max	Units	
TDCY	Data Clock	420		ns	
TDCL	Data Clock Low	180		ns	
TDCH	Data Clock High	120		ns	
TTD	Transmit Data Delay		80	ns	
TDSS	Data Setup Time	60		ns	
TDHS	Data Hole Time	40		ns	

Memory Access

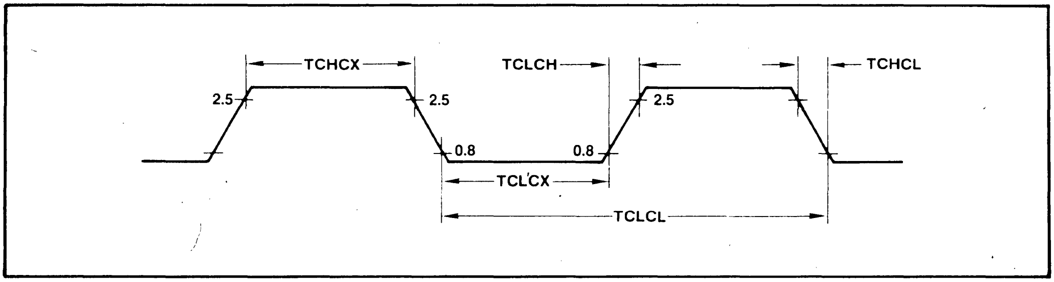


NOTE: VOL is degraded when the 8744H rapidly discharges external capacitance. This AC noise is most pronounced during emission of address data. When using external memory, locate the latch or buffer as close to the 8744H as possible.

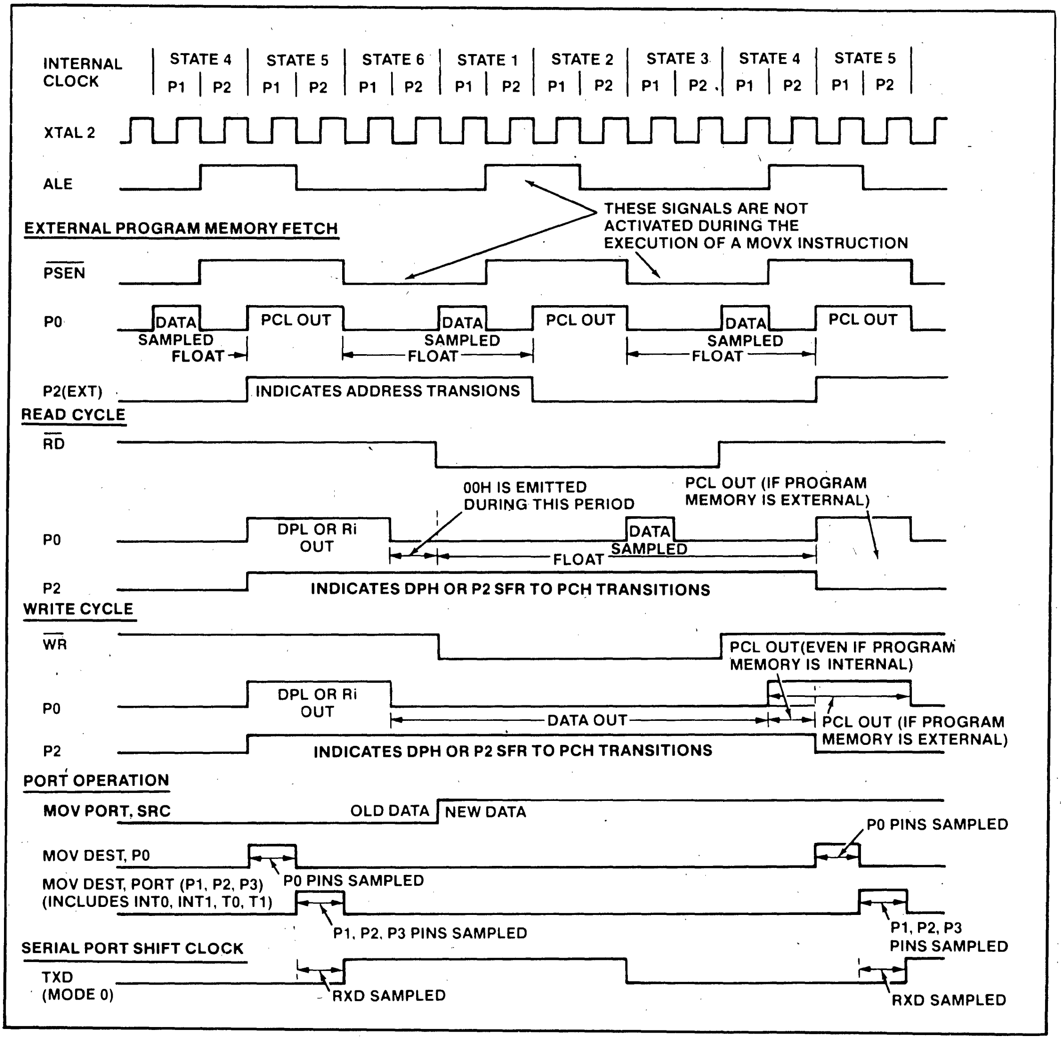
Datum	Emitting Ports	Degraded I/O Lines	VOL (peak) (max)
Address	P2, P0	P1, P3	0.8V
Write Data	P0	P1, P3, ALE	0.8V

EXTERNAL CLOCK DRIVE CHARACTERISTICS (XTAL2)

Symbol	Parameter	Min	Max	Units
TCLCL	Oscillator Period: 8751H	83.3	833.3	ns
TCHCX	High Time	20		ns
TCLCX	Low Time	20		ns
TCLCH	Rise Time		20	ns
TCHCL	Fall Time		20	ns



CLOCK WAVEFORMS


INTERNAL CLOCK
STATE 4 STATE 5 STATE 6 STATE 1 STATE 2 STATE 3 STATE 4 STATE 5
P1 P2 P1 P2 P1 P2 P1 P2 P1 P2 P1 P2 P1 P2 P1 P2
XTAL 2
ALE
THESE SIGNALS ARE NOT ACTIVATED DURING THE EXECUTION OF A MOVX INSTRUCTION
EXTERNAL PROGRAM MEMORY FETCH
PSEN
P0
DATA SAMPLED
FLOAT
PCL OUT
P2(EXT)
INDICATES ADDRESS TRANSIONS
READ CYCLE
RD
00H IS EMITTED DURING THIS PERIOD
PCL OUT (IF PROGRAM MEMORY IS EXTERNAL)
P0
DPL OR Ri OUT
FLOAT
DATA SAMPLED
P2
INDICATES DPH OR P2 SFR TO PCH TRANSITIONS
WRITE CYCLE
WR
PCL OUT(EVEN IF PROGRAM MEMORY IS INTERNAL)
P0
DPL OR Ri OUT
DATA OUT
PCL OUT (IF PROGRAM MEMORY IS EXTERNAL)
P2
INDICATES DPH OR P2 SFR TO PCH TRANSITIONS
PORT OPERATION
MOV PORT, SRC
OLD DATA
NEW DATA
MOV DEST, P0
P0 PINS SAMPLED
MOV DEST, PORT (P1, P2, P3) (INCLUDES INT0, INT1, T0, T1)
P1, P2, P3 PINS SAMPLED
SERIAL PORT SHIFT CLOCK
TXD (MODE 0)
RXD SAMPLED

This diagram indicates when signals are clocked internally. The time it takes the signals to propagate to the pins, however, ranges from 25 to 125 ns. This propagation delay is dependent on variables such as temperature and pin loading. Propagation also varies from output to output and component to component. Typically though, (T_A = 25° C, fully loaded) RD and WR propagation delays are approximately 50 ns. The other signals are typically 85 ns. Propagation delays are incorporated in the AC specifications.

8744H EPROM CHARACTERISTICS

Erasure Characteristics

Erasure of the 8744H Program Memory begins to occur when the chip is exposed to light with wavelengths shorter than approximately 4,000 Ångstroms. Since sunlight and fluorescent lighting have wavelengths in this range, constant exposure to these light sources over an extended period of time (about 1 week in sunlight, or 3 years in room-level fluorescent lighting) could cause unintentional erasure. If an application subjects the 8744H to this type of exposure, it is suggested that an opaque label be placed over the window.

The recommended erasure procedure is exposure to ultraviolet light (at 2537 Ångstroms) to an integrated dose of at least 15 W-sec/cm^2 rating for 20 to 30 minutes, at a distance of about 1 inch, should be sufficient.

Erasure leaves the array in an all 1s state.

Programming the EPROM

To be programmed, the 8744H must be running with a 4 to 6 MHz oscillator. (The reason the oscillator needs to be running is that the internal bus is being used to transfer address and program data to appropriate registers.) The address of an EPROM location to be programmed is applied to Port 1 and pins P2.0-P2.3 of Port 2, while the data byte is applied to Port 0. Pins P2.4-P2.6 and $\overline{\text{PSEN}}$ should be held low, and P2.7 and RST high. (These are all TTL levels except RST, which requires 2.5V for high.) $\overline{\text{EA}}$/VPP is held normally high, and is pulsed to +21V. While EA/VPP is at 21V, the ALE/$\overline{\text{PROG}}$ pin, which is normally being held high, is pulsed low for 50 msec. Then EA/VPP is returned to high. This is illustrated in Figure 3. Detailed timing specifications are provided in the EPROM Programming and Verification Characteristics section of this data sheet.

Program Memory Security

The program memory security feature is developed around a "security bit" in the 8744H EPROM array. Once this "hidden bit" is programmed, electrical access to the contents of the entire program memory array becomes impossible. Activation of this feature is accomplished by programming the 8744H as described in "Programming the EPROM" with the exception that P2.6 is held at a TTL high rather than a TTL low. In addition, Port 1 and P2.0-P2.3 may be in any state. Figure 4 illustrates the security bit programming configuration. Deactivating the security feature, which again allows programmability of the EPROM, is accomplished by exposing the EPROM to ultraviolet light. This exposure, as described in "Erasure Characteristics," erases the entire EPROM array. Therefore, attempted retrieval of "protected code" results in its destruction.

Program Verification

Program Memory may be read only when the "security feature" has not been activated. Refer to Figure 5 for Program Verification setup. To read the Program Memory, the following procedure can be used. The unit must be running with a 4 to 6 MHz oscillator. The address of a Program Memory location to be read is applied to Port 1 and pins P2.0-P2.3 of Port 2. Pins P2.4-P2.6 and $\overline{\text{PSEN}}$ are held at TTL low, while the ALE/$\overline{\text{PROG}}$, RST, and $\overline{\text{EA}}$/VPP pins are held at TTL high. (These are all TTL levels except RST, which requires 2.5V for high.) Port 0 will be the data output lines. P2.7 can be used as a read strobe. While P2.7 is held high, the Port 0 pins float. When P2.7 is strobed low, the contents of the addressed location will appear at Port 0. External pullups (e.g., 10K) are required on Port 0 during program verification.

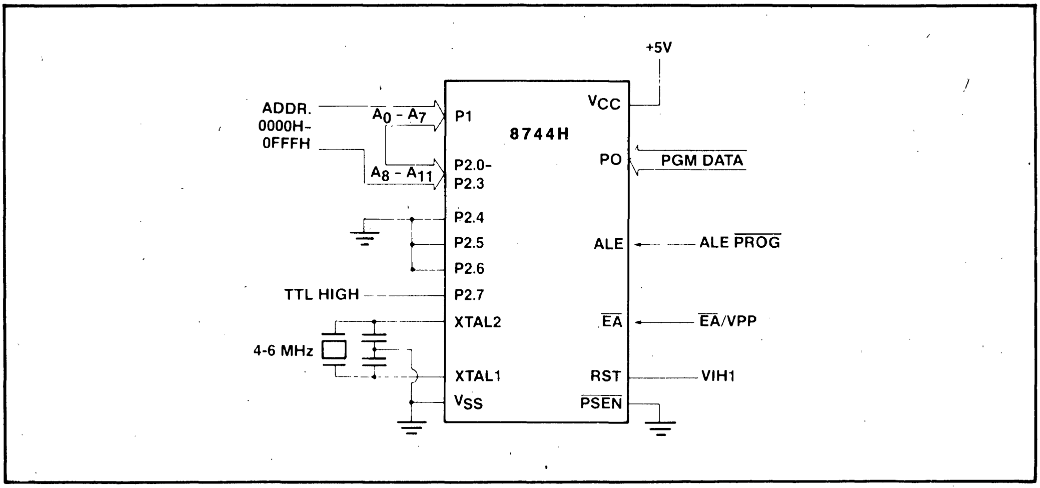


Figure 3. Programming Configuration

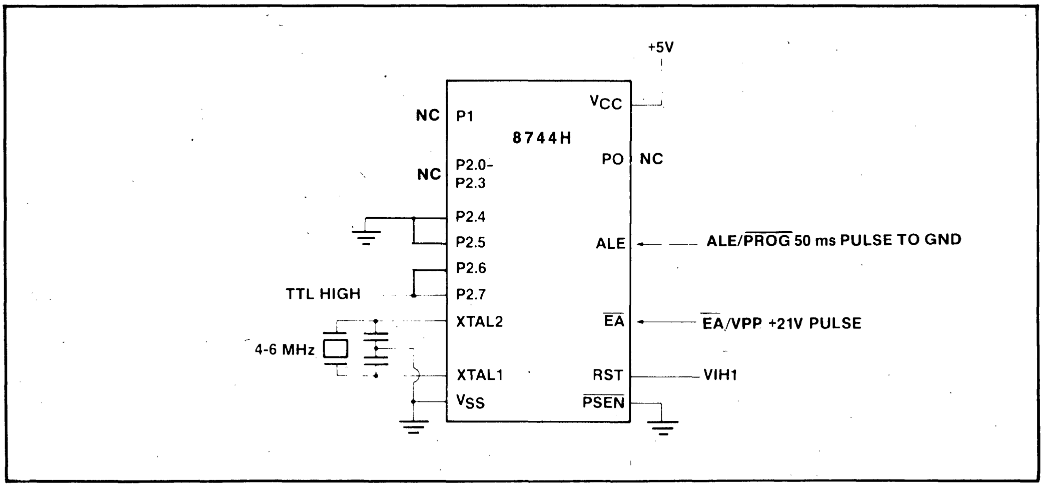


Figure 4. Security Bit Programming Configuration

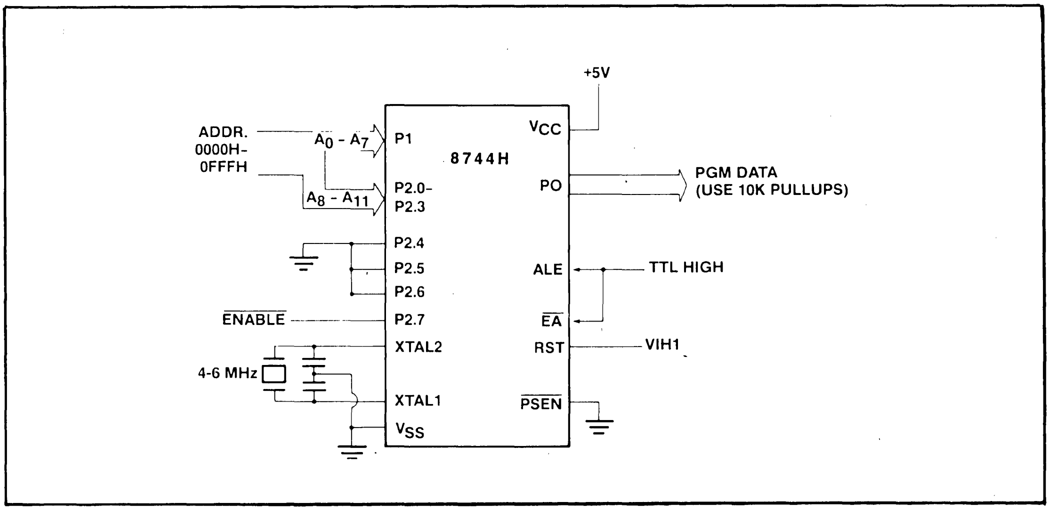


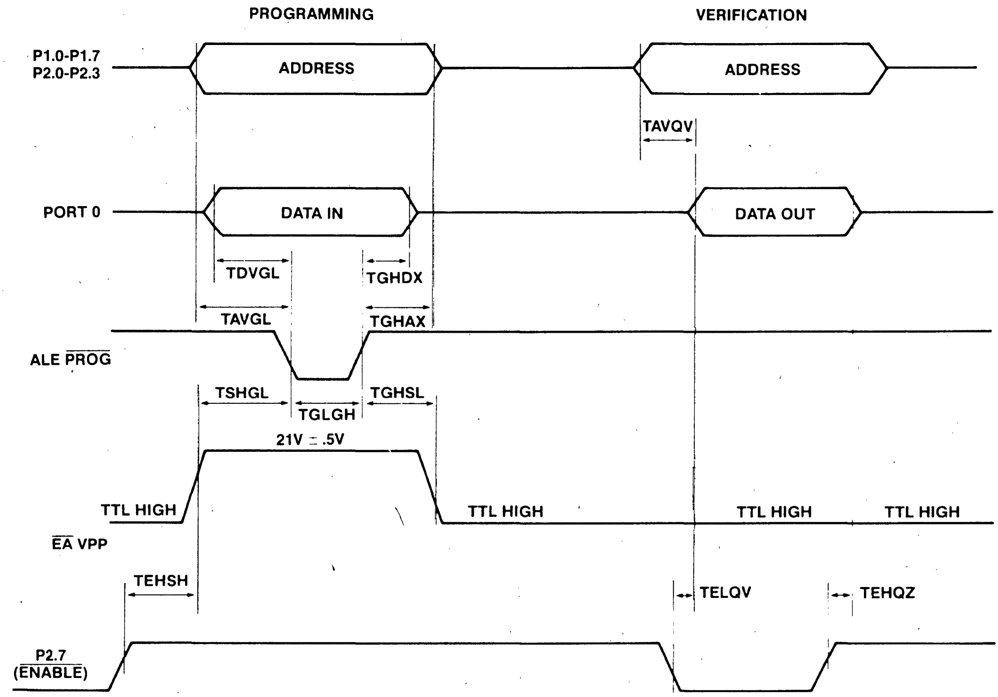
Figure 5. Program Verification Configuration

EPROM PROGRAMMING, SECURITY BIT PROGRAMMING AND VERIFICATION CHARACTERISTICS

(TA = 21°C to 27°C, V_{CC} = 4.5V to 5.5V, V_{SS} = 0V)

Symbol	Parameter	Min	Max	Units
V_{PP}	Programming Supply Voltage	20.5	21.5	V
IPP	Programming Current		30	mA
1/TCLCL	Oscillator Frequency	4	6	MHz
TAVGL	Address Setup to $\overline{\text{PROG}}$	48TCLCL		
TGHAX	Address Hold after $\overline{\text{PROG}}$	48TCLCL		
TDVGL	Data Setup to $\overline{\text{PROG}}$	48TCLCL		
TGHDX	Data Hold after $\overline{\text{PROG}}$	48TCLCL		
TEHSH	$\overline{\text{ENABLE}}$ High to V_{PP}	48TCLCL		
TSHGL	V_{PP} Setup to $\overline{\text{PROG}}$	10		μsec
TGHSL	V_{PP} Hold after $\overline{\text{PROG}}$	10		μsec
TGLGH	$\overline{\text{PROG}}$ Width	45	55	msec
TAVQV	Address to Data Valid		48TCLCL	
TELQV	$\overline{\text{ENABLE}}$ to Data Valid		48TCLCL	
TEHQZ	Data Float after $\overline{\text{ENABLE}}$	0	48TCLCL	

EPROM PROGRAMMING, SECURITY BIT PROGRAMMING AND VERIFICATION WAVEFORMS



FOR PROGRAMMING CONDITIONS SEE FIGURE 3.
FOR SECURITY BIT PROGRAMMING CONDITIONS SEE FIGURE 4.
FOR VERIFICATION CONDITIONS SEE FIGURE 5.

RUPI™ Article Reprints

22

ARTICLE
REPRINT

AR-307

NOVEMBER 1983

MICROCONTROLLER WITH INTEGRATED HIGH PERFORMANCE COMMUNICATIONS INTERFACE

CHARLES YAGER
APPLICATIONS ENGINEER

ORDER NUMBER 230876-001

SUMMARY

The 8044 offers a lower cost and higher performance solution to networking microcontrollers than conventional solutions. The system cost is lowered by integrating an entire microcomputer with an intelligent HDLC/SDLC communication processor onto a single chip. The higher performance is realized by integrating two processors running concurrently on one chip; the powerful 8051 microcontroller and the Serial Interface Unit. The 8051 microcontroller is substantially off-loaded from the communication tasks when using the AUTO mode. In the AUTO mode the SIU handles many of the data link functions in hardware. The advantages of the AUTO mode are: less software is required to implement a secondary station data link, the 8051 CPU is offloaded, and the turn-around time is reduced, thus increasing the network throughput. Currently the 8044 is the only microcontroller with a sophisticated communications processor on-chip. In the future there will be more microcontrollers available following this trend.

INTRODUCTION

Today microcontrollers are being designed into virtually every type of equipment. For the household, they are turning up in refrigerators, thermostats, burglar alarms, sprinklers, and even water softeners. At work they are found in laboratory instruments, copiers, elevators, hospital equipment, and telephones. In addition, a lot of microcomputer equipment contains more than one microcontroller. Applications using multiple microcontrollers as well, as the office and home, are now faced with the same requirements that laboratory instfuments were faced with 12 years ago — they need to connect them together and have them communicate. This need was satisfied in the laboratory with the IEEE-488 General Purpose Instrumentation Bus (GPIB). However, GPIB does not meet the current design objectives for networking microcontrollers.

Today there are many communications schemes and protocols available; some of the popular ones are GPIB, Async, HDLC/SDLC, and Ethernet. Common design objectives of today's networks are: low cost, reliable, efficient throughput, and expandable. In examining available solutions, GPIB does not meet these design objectives; first, the cable is too expensive (parallel communications), second, it can only be used over a limited distance (20 meters), and third, it can only handle a limited number of stations. For general networking, serial communications is preferable because of lower cable costs and higher reliability (fewer connections). While Ethernet provides very high performance, it is more of a system backbone rather than a microcontroller interconnect. Async, on the other hand, is inexpensive but it is not an efficient protocol for data block or file transfers. Even with some new modifications such as a 9 bit protocol for addressing, important functions such as acknowledgements, error checking/recovery, and data transparency are not standardized nor supported by available data comm chips.

SDLC, Synchronous Data Link Control, meets the requirements for communications link design. The physical medium can be used on two or four wire twisted pair with inexpensive transceivers and connectors. It can also be interfaced through modems, which allows it to be used on broadband networks, leased or switched telephone lines. VLSI controllers have been available from a number of vendors for years; higher performance and more user friendly SDLC controllers continue to appear. SDLC has also been designed to be very reliable. A 16 bit CRC checks the integrity of the received data, while frame numbering and acknowledgements are also built in. Using SDLC, up to 254 stations can be uniquely addressed, while HDLC addressing is unlimited. If an RS-422 only requires a single +5 volt power supply.

What will the end user pay for the added value provided by communications? The cost of the communications hardware is not the only additional cost. There will be performance degradation in the main application because the microcontroller now has additional tasks to perform. There are two extremes to the cost of adding communication capability. One could spend very little by adding an I/O port and have the CPU handle everything from the baud rate to the protocol. Of course the main application would be idle while the CPU was communicating. The other extreme would be to add another microcontroller to the system dedicated to communications. This communications processor could interface to the main CPU through a high speed parallel link or dual port RAM. This approach would maintain system performance, but it would be costly.

Adding HDLC/SDLC Netowrking Capability

Figure 1 shows a microcomputer system with a conventional HDLC/SDLC communications solution. The additional hardware needed to realize the conventional design is: an HDLC/SDLC communication chip, additional ROM for the communication software, part of an interrupt controller, a baud rate generator, a phase locked loop, NRZI encoded/decoder, and a cable driver. The NRZI encoder/decoder, and phase locked loop are used when the transmitter does not send the clock on a separate line from the data (i.e. over telephone lines, or two wire cable). The NRZI encoder/decoder is used in HDLC/SDLC to combine the clock into the data line. A phase locked loop is used to recover the clock from the data line.

The majority of the available communication chips provide a limited number of data link control functions. Most of them will handle Zero Bit Insertion/Deletion (ZBI/D), Flags, Aborts, Automatic

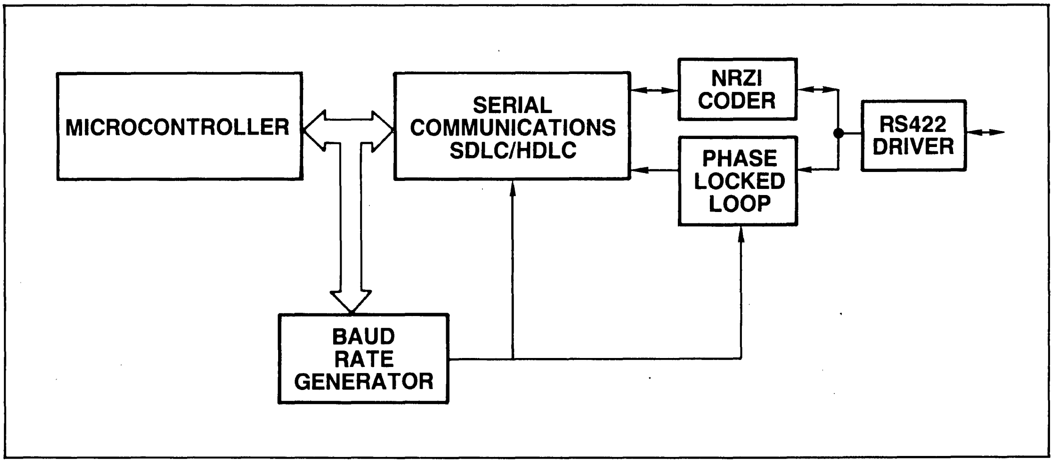


Figure 1. Conventional microcontroller networking solution

address recognition, and CRC generation and checking. It is the CPU's responsibility to manage link access, command recognition and response, acknowledgements and error recovery. Handling these tasks can take a lot of CPU time. In addition, servicing the transmission and reception of data bytes can also be very time consuming depending on the method used.

The Using a DMA controller can increase the overall system performance, since it can transfer a block of data in fewer clock cycles than a CPU. In addition, the CPU and the DMA controller can multiplex their access to the bus so that both can be running at virtually the same time. However, both the DMA controller and the CPU are sharing the same bus, therefore, neither one get to utilize 100% of the bus bandwidth. Microcontrollers available today do not support DMA, therefore, they would have to use interrupts, since polling is unacceptable in a multitasking environment.

In an interrupt driven, the CPU has overhead in addition to servicing the interrupt. During each interrupt request the CPU has to save all of the important registers, transfer a byte, update pointers and counters, then restore all of its registers. At low bit rates this overhead may be insignificant. However, the percentage of overhead increases linearly with the bit rate. At high bit rates this overhead would consume all of the CPU's time. There is another nuisance factor associated with interrupt driven systems, interrupt latency. Too much interrupt latency will cause data to be lost from underrun and overrun errors.

The additional hardware necessary to implement the communications solution, as shown in Figure 1, would require 1 LSI chip and about 10 TTL chips. The cost of CPU throughput degradation can be even greater. The percentage of time the CPU has to spend servicing the communication tasks can be anywhere from 10-100%, depending on the serial bit rate. These high costs will prevent consumer acceptance of networking microcomputer equipment.

A Highly Integrated, High Performance Solution

The 8044 reduces the cost of networking microcontrollers without compromising performance. It contains all of the hardware components necessary to implement a microcomputer system with communications capability, plus it reduces the CPU and software overhead of implementing HDLC/SDLC. Figure 2 shows a functional block diagram of the 8044.

The 8044 integrates the powerful 8051 microcontroller with an intelligent Serial Interface Unit to provide a single chip solution which efficiently implements a distributed processing or distributed control system. The microcontroller is a self sufficient unit containing ROM, RAM, ALU and its own peripherals. The 8044's architecture and instruction set are identical to the 8051's. The Serial Interface Unit (SIU) uses bit synchronous HDLC/SDLC protocol and can communicate at bit rates up to 2.4 Mbps, externally clocked, or up to 375 Kbps using the on-chip digital phase locked loop. The SIU contains its own processor, which operates concurrently with the microcontroller.

The CPU and the SIU, in the 8044, interface through 192 bytes of dual port RAM. There is no hardware arbitration in the dual port RAM. Both processor's memory access cycles are interlaced; each processor has access every other clock cycle. Therefore, there

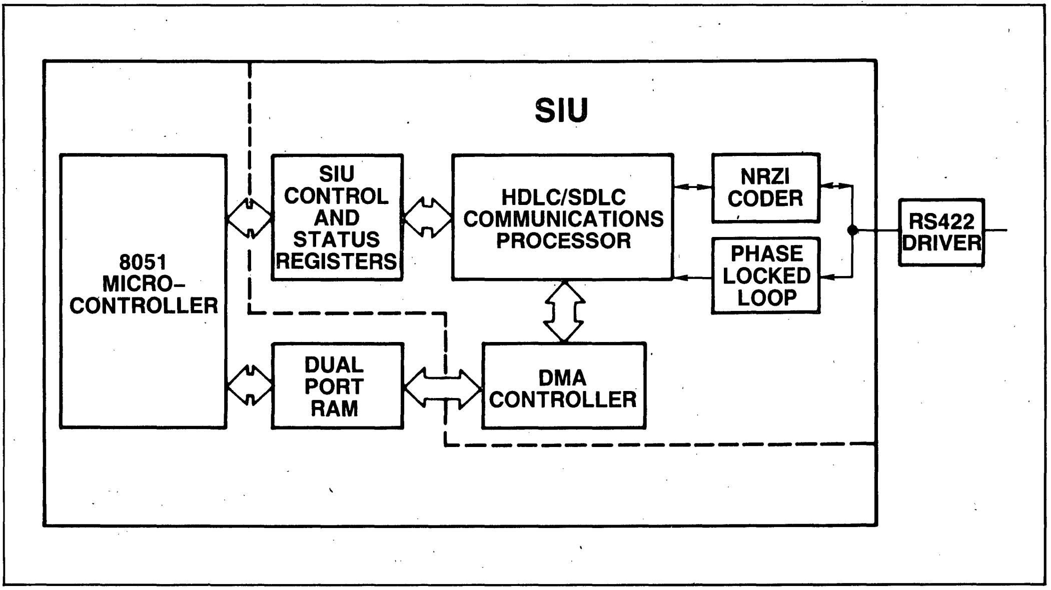


Figure 2. 8044 single chip microcontroller networking solution

is no throughput loss in either processor as a result of the dual port RAM, and execution times are deterministic. Since this has always been the method for memory access on the 8051 microcontroller, 8051 programs have the same execution time in the 8044.

By integrating all of the communication hardware onto the 8051 microcontroller, the hardware cost of the system is reduced. Now several chips have been integrated into a single chip. This means that the system power is reduced, P.C. board space is reduced, inventory and assembly is reduced, and reliability is improved. The improvement in reliability is a result of fewer chips and interconnections on the P.C. board.

As mentioned before, there can be two extremes in a design which adds communications to the microcomputer system. The 8044 solution uses the high end extreme. The SIU on the 8044 contains its own processor which communicates with the 8051 processor through dual port RAM and control/status registers. While the SIU is not a totally independent communications processor, it substantially offloads the 8051 processor from the communication tasks.

The DMA on the 8044 is dedicated to the SIU. It cannot access external RAM. By having a DMA controller in the SIU, the 8051 CPU is offloaded. As a result of the dual port RAM design, the DMA does not share the running at full speed while the frames are being transmitted or received. Also, the nuisance of overrun and underrun errors is totally eliminated since the dedicated DMA controller is guaranteed to meet the maximum data rates. Having a dedicated DMA controller means that the serial channel interrupt can be the lowest priority, thus allowing the CPU to have higher priority real time interrupts.

Figure 3 shows a comparison between the conventional and the 8044 solution on the percentage of time the CPU must spend sending data. This diagram was derived by assuming a 64 byte information frame is being transmitted repeatedly. The conventional solution is interrupt driven, and each interrupt service routine is assumed to take about 15 instructions with a 1 μsec instruction cycle time. At 533 Kbps, an interrupt would occur every 15 usec. Thus, the CPU becomes completely dedicated to servicing the serial communications. The conventional design could not support bit rates higher than this because of underruns and overruns. For the 8044 to repeatedly send 64 byte frames, it simply has to reinitialize the DMA controller. As a result, the 8044 can support bit rates up to 2.4 Mbps.

Some of the other communications tasks the CPU has to perform, such as link access, command recognition/response, and acknowledgements, are performed automatically by the SIU in a mode called "AUTO." The combination of the dedicated DMA controller and the AUTO mode, substantially offload

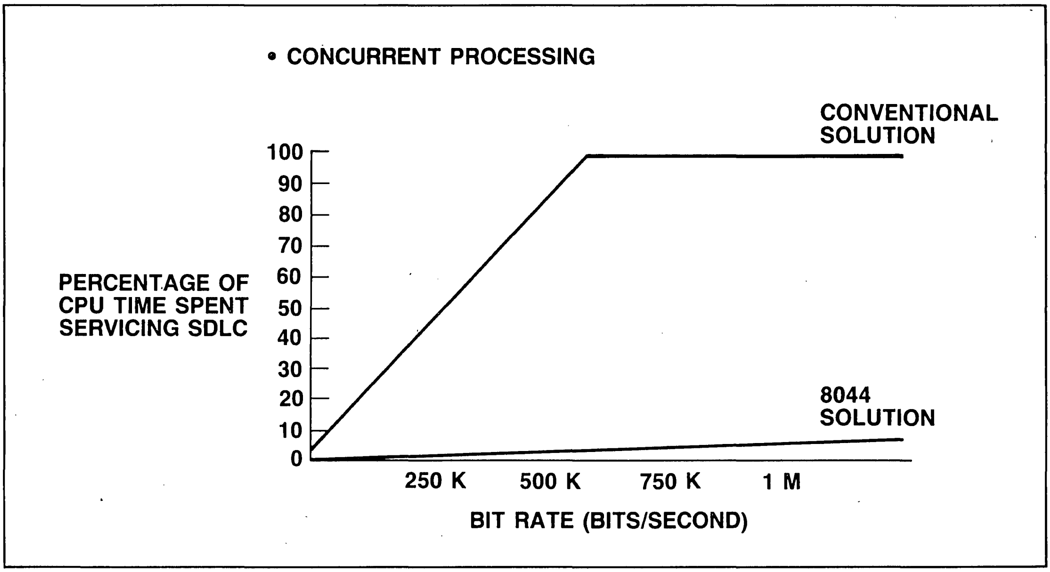


Figure 3. SIU offloads CPU

the CPU, thus allowing it to devote more of its power to other tasks.

8044's Auto Mode

In the AUTO mode the SIU implements in hardware a subset of the SDLC protocol such that it responds to many SDLC commands without CPU intervention. All AUTO mode responses to the primary station conform to IBM's SDLC definition. In the AUTO mode the 8044 can only be a secondary station operating in SDLC specified "Normal Response Mode." Normal Response Mode means that the secondary station can not transmit unless it is polled by the primary station. The SIU in the AUTO mode can recognize and respond to the following SDLC commands without CPU intervention: I (Information), RR (Receive Ready), RNR (Receive Not Ready), REJ (Reject), and for loop mode UP (Unnumbered Poll). The SIU can generate the following responses without CPU intervention: I, RR, and RNR. In addition, the SIU manages Ns and Nr in the control field. If it detects an error in either Ns or Nr, it interrupts the CPU for error recovery.

How does the SIU know what responses to send to the primary? It uses two status bits which are set by the CPU. The two bits are TBF (Transmit Buffer Full) and RBP (Receive Buffer Protect). TBF indicates that the CPU wants to send data, and RBP indicates that the receive data buffer is full. Table 1 shows the responses the SIU will send based on these two status bits. This is an innovative approach to communication design. the CPU in the 8044 with one instruction can directly set a bit which communicates to the primary what its transmit and receive buffering status is.

When the CPU wants to send a frame, it loads the transmit buffer with the data, loads the starting address and the count of the data into the SIU, then sets TBF to transmit the frame. The SIU waits for the primary station to poll it with a RR command. After the SIU is polled, it automatically sends the information frame to the primary with the proper control field. The SIU then waits for a positive acknowledgement from the primary before incrementing the Ns field and interrupting the CPU for more data. If a negative acknowledgement is received, the SIU automatically retransmits the frame.

When the 8044 is ready to receive information, the CPU loads the receive buffer starting address and the buffer length into the SIU, then enables the receiver. When a valid information frame with the correct address and CRC is received, the SIU will increment the Nr field, disable the receiver and interrupt the CPU indicating that a good I frame has been received. The CPU then sets RBP, reenables the receiver and processes the received data. By enabling the receiver with RBP set, the SIU will automatically respond to polls with a Receive Not Ready, thus keeping the link moving rather than timing out the primary from a disabled receiver, or interrupting the CPU with another poll before it has processed the data. After the data has been processed, the CPU clears RBP, returning to the Receive Ready responses.

Table 1. SIU's automatic responses in auto mode

STATUS BITS		RESPONSE
TBF	RBP	
0	0	(RR) Receive ready
0	1	(RNR) Receive not ready
1	0	(I) Information
1	1	(I) Information

SDLC communications can be broken up into four states: Logical Disconnect State, Initialization State, Frame Reject State, and Information Transfer State. Data can only be transferred in the Information Transfer State. More than 90% of the time a station will be in the Information Transfer State, which is where the SIU can run autonomously. In the other states, where error recovery, online/offline, and initialization takes place, the CPU manages the protocol.

In the Information Transfer State there are three common events which occur as illustrated in Figure 4, they are: 1) the primary polls the secondary and the secondary is ready to receive but has nothing to send, 2) the primary sends the secondary information, and 3) the secondary sends information to the primary. Figures 5, 6, and 7 compare the functions the conventional design and the 8044 must execute in order to respond to the primary for the cases in Figure 4.

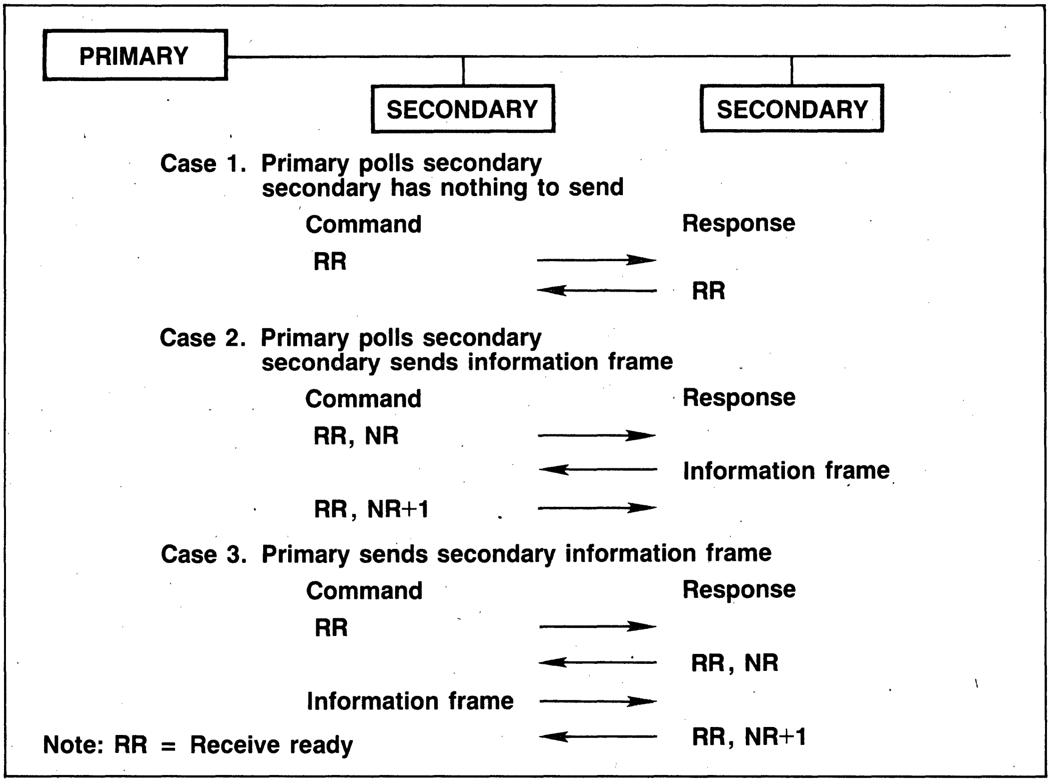
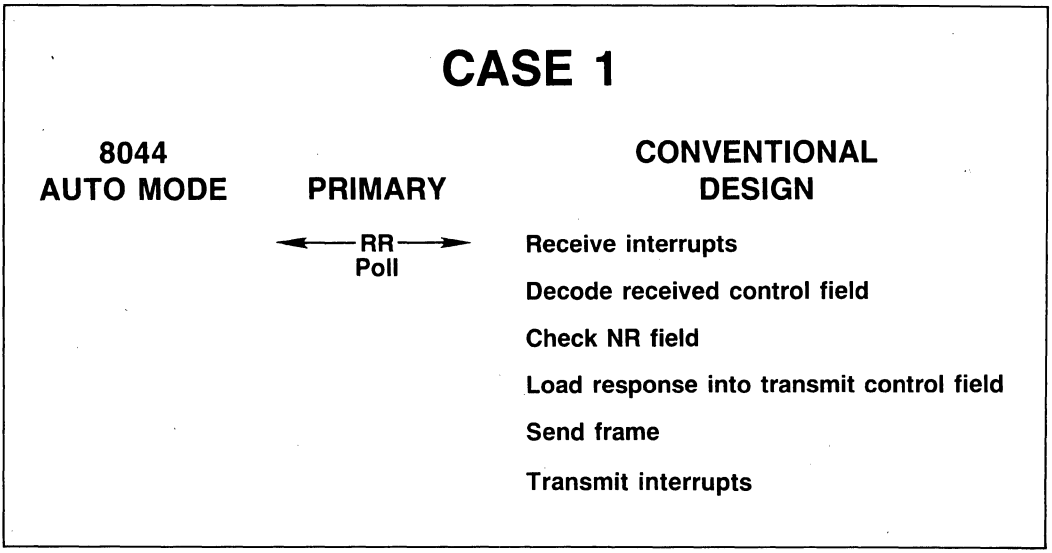

PRIMARY
SECONDARY
SECONDARY
Case 1. Primary polls secondary secondary has nothing to send
Command
Response
RR
RR
Case 2. Primary polls secondary secondary sends information frame
Command
Response
RR, NR
Information frame
RR, NR+1
Case 3. Primary sends secondary information frame
Command
Response
RR
RR, NR
Information frame
RR, NR+1
Note: RR = Receive ready

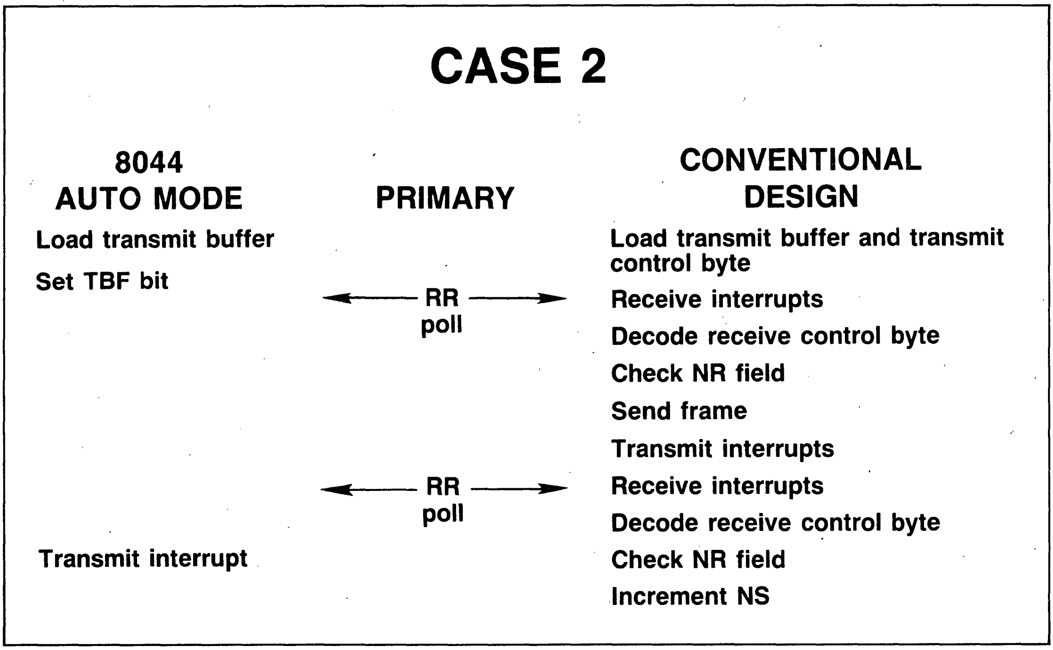
Figure 4. SDLC commands and responses in the information transfer state



CASE 1

8044 AUTO MODE	PRIMARY	CONVENTIONAL DESIGN
	←— RR —→ Poll	Receive interrupts
		Decode received control field
		Check NR field
		Load response into transmit control field
		Send frame
		Transmit interrupts

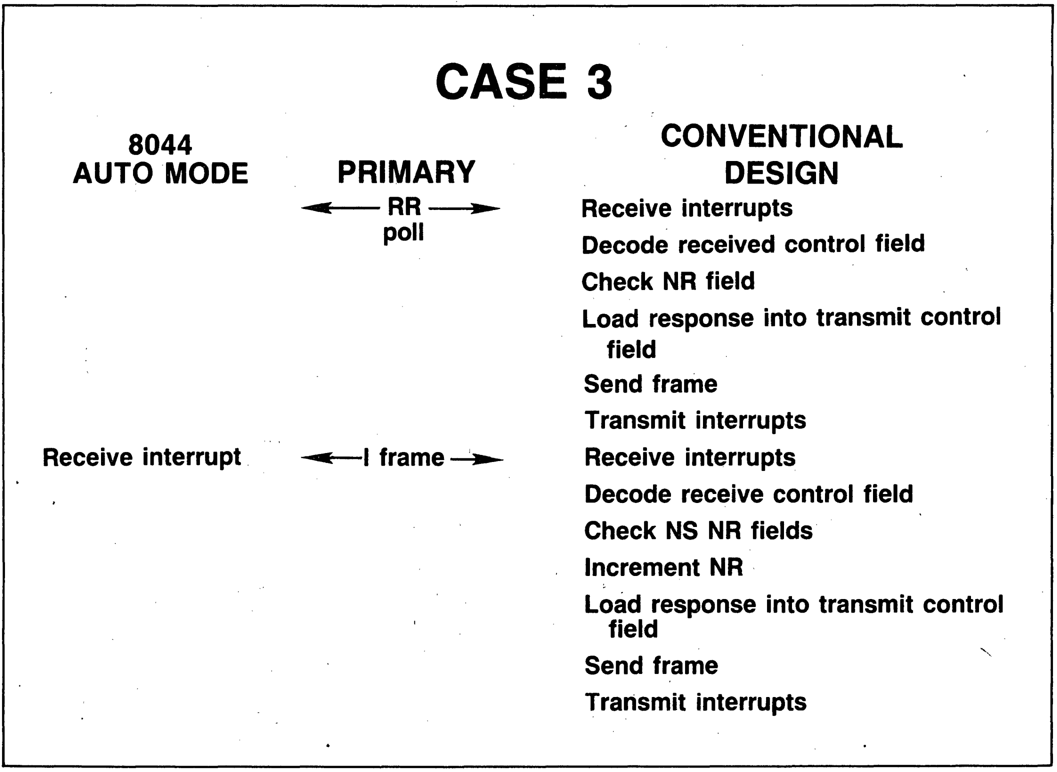
Figure 5. Primary polls secondary, secondary has nothing to send



CASE 2

8044 AUTO MODE	PRIMARY	CONVENTIONAL DESIGN
Load transmit buffer		Load transmit buffer and transmit control byte
Set TBF bit	←— RR —→ poll	Receive interrupts
		Decode receive control byte
		Check NR field
		Send frame
		Transmit interrupts
	←— RR —→ poll	Receive interrupts
		Decode receive control byte
Transmit interrupt		Check NR field
		Increment NS

Figure 6. Primary polls secondary, secondary sends information frame



CASE 3

8044 AUTO MODE

PRIMARY

CONVENTIONAL DESIGN

RR poll

Receive interrupts

Decode received control field

Check NR field

Load response into transmit control field

Send frame

Transmit interrupts

Receive interrupt

I frame

Receive interrupts

Decode receive control field

Check NS NR fields

Increment NR

Load response into transmit control field

Send frame

Transmit interrupts

Figure 7. Primary sends information frame to secondary

Using case 1 as an example, the conventional design first gets receive interrupts bringing the data from the SDLC comm chip into memory. The CPU must then decode the command in the control field and determine the response. In addition, it must check the Nr field for any pending acknowledgements. The CPU loads the transmit buffer with the appropriate address and control field, then transmits the frame. When the 8044 receives this frame in AUTO mode, the CPU never gets an interrupt because the SIU handles the entire frame reception and response automatically.

In SDLC networks, when there is no information transfers, case 1 is the activity on the line. Typically this is 80% of the network traffic. The CPU in the conventional design would constantly be getting interrupts and servicing the communications tasks, even when it has nothing to send or receive. On the other hand, the 8044 CPU would only get involved in communicating when it has data to send or receive.

Having the SIU implement a subset of the SDLC protocol in hardware not only offloads the CPU, but it also improves the throughput on the network. The most critical parameter for calculating throughput on any high speed network is the station turnaround time; the time it takes a station to respond after receiving a frame. Since the 8044 handles all of the commands and responses of the Information Transfer State in hardware, the turnaround time is much faster than handling it in software, hence a higher throughput.

8044's Flexible Mode

In the "NON-AUTO" mode or Flexible mode, the SIU does not recognize or respond to any commands, nor does it manage acknowledgements, which means the CPU must handle link access, command recognition/response, acknowledgements and error recovery by itself. The Flexible mode allows the 8044 to have extended address fields and extended control fields, thus providing HDLC support. In the Flexible mode the 8044 can operate as a primary station, since it can transmit without being polled.

Front End Communications Processor

The 8044 can also be used as an intelligent HDLC/SDLC front end for a microporcessor, capable of extensively off-loading link control functions for the microporcessor. In some applications the 8044 can even be used for communications preprocessing, in addition to data link control. For this type of design the 8044 would communicate to the Host CPU through a FIFO,.or dual port RAM. A block diagram of this design is given in Figure 8. A tightly coupled interface between the 8044 and the Host CPU would be established. The Host CPU would give the 8044 high level commands and data which the 8044 would convert to HDLC/SDLC. This particular type of design would be most appropriate for a primary Station which is normally a micro, mini, or mainframe computer. Sophisticated secondary stations could also take advantage of this design.

Since the 8044 has ROM on chip, all the communications software is non-volatile. The 8044 primary station could down-line-load software to 8044 secondary stations. Once down-line-loading is implemented, software updates to the primary and secondary stations could be done very inexpensively. The only things which would remain fixed in ROM are the HDLC/SDLC communications software and the software interface to the HOST.

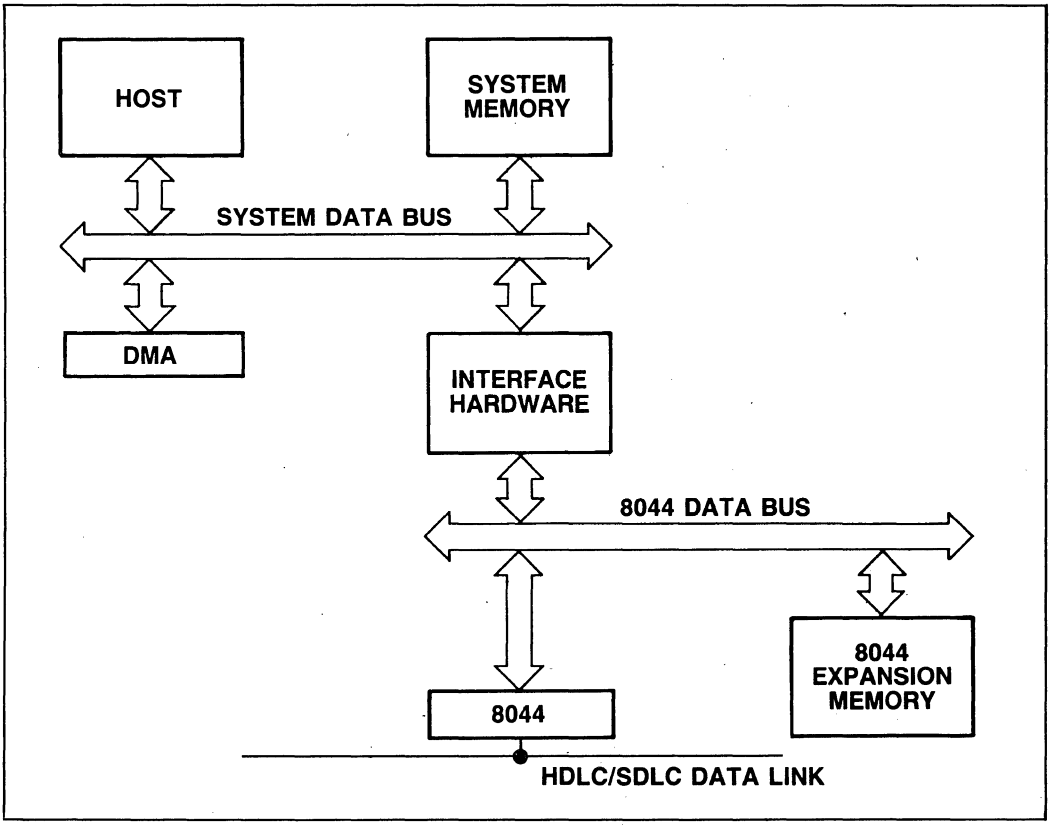


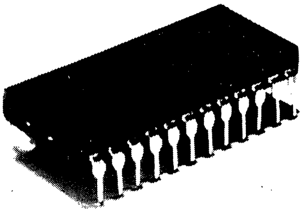
Figure 8. 8044 front end processor

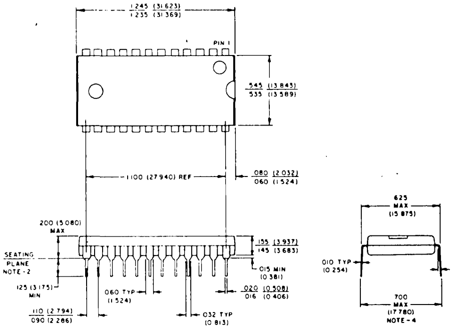
Packaging Information 23

NOTES:

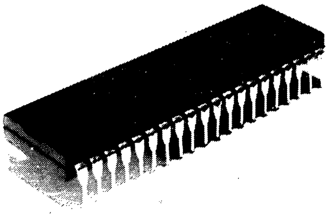
1. All packages drawings not to scale.
2. All packages seating plane defined by .0415 to .0430 PCB holes.
3. Type P packages only. Package length dqes not include end flash burr. Burr is .005 nominal, can be .010 max. at one end.
4. All package drawings end view dimensions are to outside of leads.

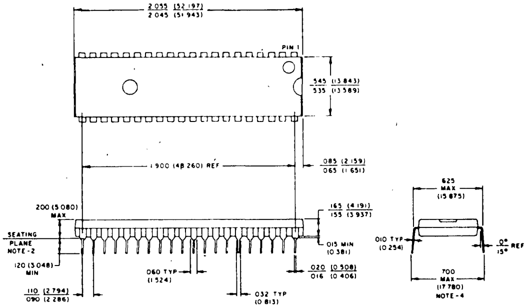
24-LEAD PLASTIC DUAL IN-LINE PACKAGE TYPE P



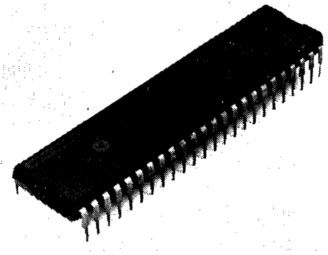


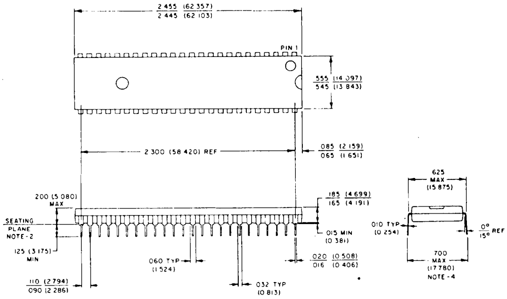
40-LEAD PLASTIC DUAL IN-LINE PACKAGE TYPE P



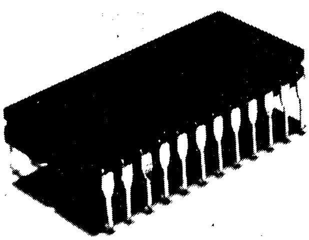


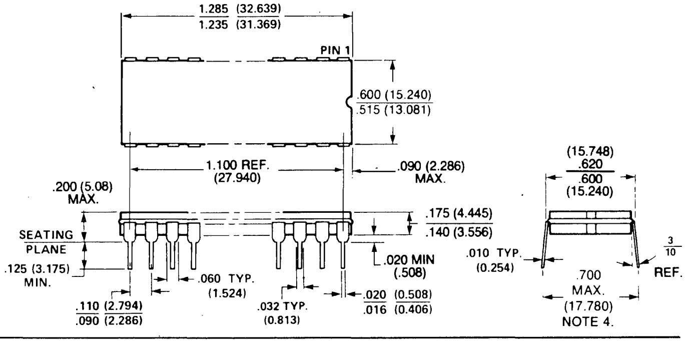
48-LEAD PLASTIC DUAL IN-LINE PACKAGE TYPE P



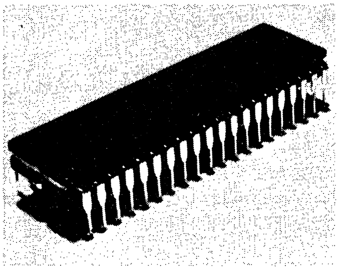


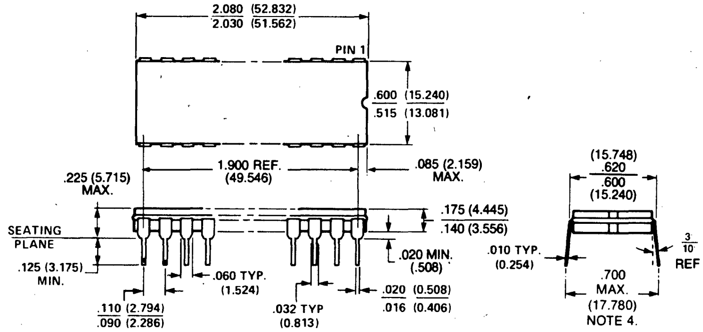
24-LEAD HERMETIC DUAL IN-LINE PACKAGE TYPE D



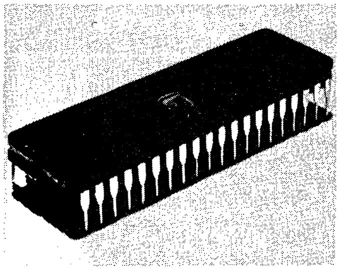


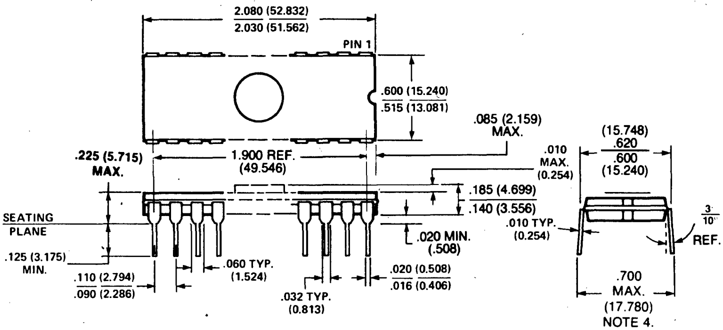
40-LEAD HERMETIC DUAL IN-LINE PACKAGE TYPE D



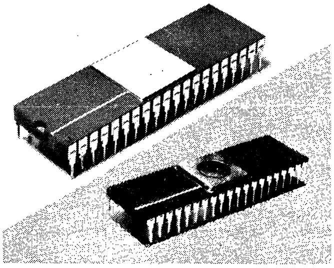


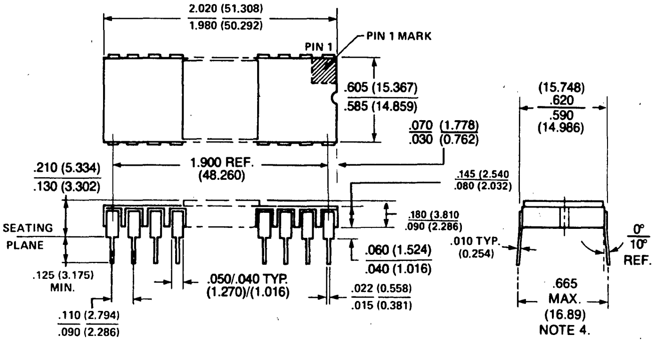
40-LEAD HERMETIC DUAL IN-LINE PACKAGE TYPE D



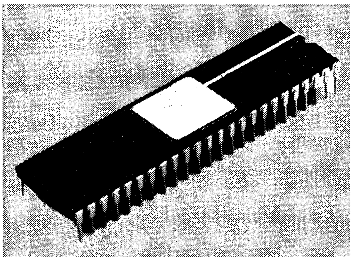


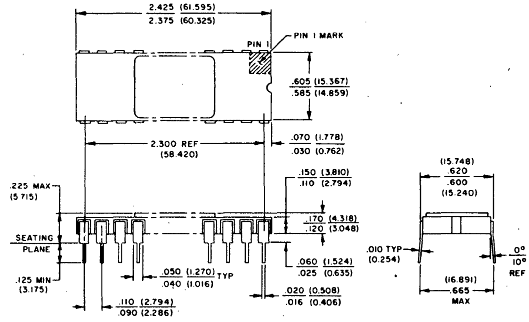
40-LEAD HERMETIC DUAL IN-LINE PACKAGE TYPE C





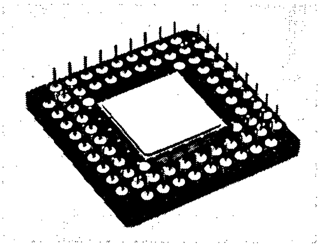
48-LEAD PLASTIC DUAL IN-LINE PACKAGE TYPE C

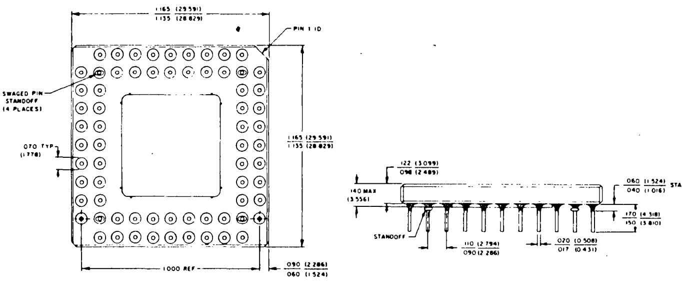




CERAMIC PIN GRID ARRAY PACKAGE TYPE CG

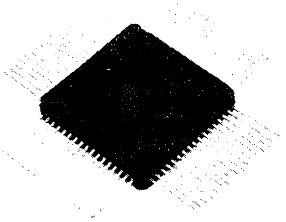
68-LEAD CERAMIC PIN GRID ARRAY PACKAGE TYPE CG

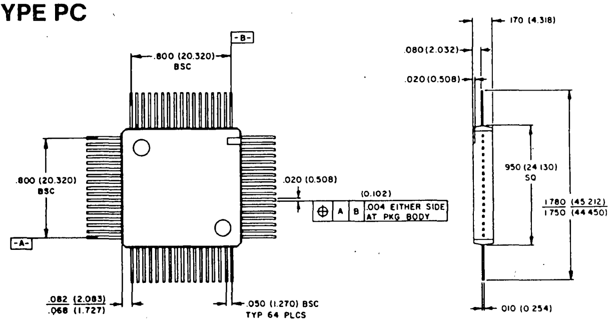




PLASTIC FLAT PACK PACKAGE TYPE PC

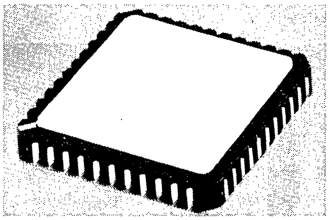
68-LEAD PLASTIC FLAT PACK PACKAGE TYPE PC

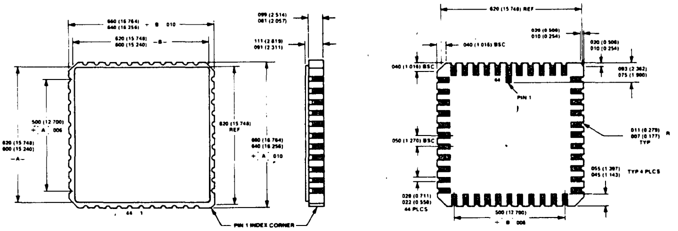




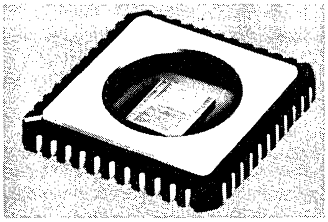
CERAMIC LEADLESS CHIP CARRIERS

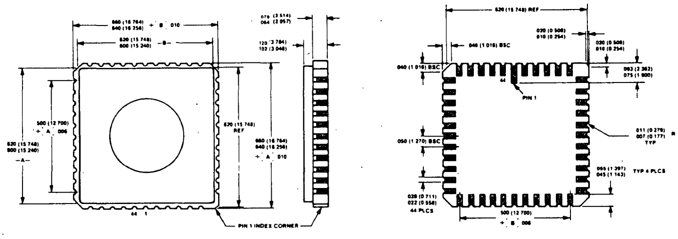
44-LEAD CERAMIC LEADLESS PACKAGE TYPE C





44-LEAD CERAMIC LEADLESS PACKAGE TYPE C





intel

DOMESTIC SALES OFFICES

ALABAMA

Intel Corp.
303 Williams Avenue, S.W.
Suite 1422
Huntsville 35801
Tel: (205) 533-9353

ARIZONA

Intel Corp.
11225 N. 28th Drive
Suite 214D
Phoenix 85029
Tel: (602) 869-4980

CALIFORNIA

Intel Corp.
1010 Hurley Way
Suite 300
Sacramento 95825
Tel: (916) 929-4078

Intel Corp.
7670 Opportunity Road
Suite 135
San Diego 92111
(714) 268-3563

Intel Corp.•
2000 East 4th Street
Suite 100
Santa Ana 92705
Tel: (714) 835-9642
TWX: 910-595-1114

Intel Corp.•
1350 Shorebird Way
Mt. View 94043
Tel: (415) 968-8086
TWX: 910-339-9279
910-338-0255

Intel Corp.•
5530 Corbin Avenue
Suite 120
Tarzana 91356
Tel: (213) 708-0333
TWX: 910-495-2045

COLORADO

Intel Corp.
4445 Northpark Drive
Suite 100
Colorado Springs 80907
Tel: (303) 594-6622

Intel Corp.•
650 S. Cherry Street
Suite 720
Denver 80222
Tel: (303) 321-8086
TWX: 910-931-2289

CONNECTICUT

Intel Corp.
36 Padanaram Road
Danbury 06810
Tel: (203) 792-8366
TWX: 710-456-1199

EMC Corp.
393 Center Street
Wallingford 06492
Tel: (203) 265-6991

FLORIDA

Intel Corp.
1500 N.W. 62nd Street
Suite 104
Ft. Lauderdale 33309
Tel: (305) 771-0600
TWX: 510-956-9407

Intel Corp.
500 N. Maitland
Suite 205
Maitland 32751
Tel: (305) 628-2393
TWX: 810-853-9219

GEORGIA

Intel Corp.
3300 Holcombe Bridge Road
Suite 225
Norcross 30092
Tel: (404) 449-0541

ILLINOIS

Intel Corp.•
2550 Golf Road
Suite 815
Rolling Meadows 60008
Tel: (312) 981-7200
TWX: 910-651-5881

INDIANA

Intel Corp.
9100 Purdue Road
Suite 400
Indianapolis 46268
Tel: (317) 875-0623

IOWA

Intel Corp.
St. Andrews Building
1930 St. Andrews Drive N.E.
Cedar Rapids 52402
Tel: (319) 393-5510

KANSAS

Intel Corp.
8400 W. 110th Street
Suite 170
Overland Park 66210
Tel: (913) 642-8080

LOUISIANA

Industrial Digital Systems Corp.
2332 Severn Avenue
Suite 202
Metairie 70001
Tel: (504) 831-8492

MARYLAND

Intel Corp.•
7257 Parkway Drive
Hanover 21076
Tel: (301) 796-7500
TWX: 710-862-1944

Intel Corp.
1620 Elton Road
Silver Spring 20903
Tel: (301) 431-1200

MASSACHUSETTS

Intel Corp.•
27 Industrial Avenue
Chelmsford 01824
Tel: (617) 256-1800
TWX: 710-343-6333

EMC Corp.
385 Elliot Street
Newton 02164
Tel: (617) 244-4740
TWX: 922531

MICHIGAN

Intel Corp.•
26500 Northwestern Hwy.
Suite 401
Southfield 48075
Tel: (313) 353-0920
TWX: 810-244-4915

MINNESOTA

Intel Corp.
3500 W. 80th Street
Suite 360
Bloomington 55431
Tel: (612) 835-6722
TWX: 910-576-2867

MISSOURI

Intel Corp.
4203 Earth City Expressway
Suite 131
Earth City 63045
Tel: (314) 291-1990

NEW JERSEY

Intel Corp.•
Raritan Plaza III
Raritan Center
Edison 08837
Tel: (201) 225-3000
TWX: 710-480-6238

NEW MEXICO

Intel Corp.
1120 Juan Tabo N.E.
Albuquerque 87112
Tel: (505) 292-8086

NEW YORK

Intel Corp.•
300 Vanderbilt Motor Parkway
Hauppauge 11788
Tel: (516) 231-3300
TWX: 510-227-6236

Intel Corp.
80 Washington Street
Poughkeepsie 12601
Tel: (914) 473-2303
TWX: 510-248-0060

Intel Corp.•
211 White Spruce Boulevard
Rochester 14623
Tel: (716) 424-1050
TWX: 510-253-7391

T-Squared
6443 Ridings Road
Syracuse 13206
Tel: (315) 463-8592
TWX: 710-541-0554

T-Squared
7353 Pittsford
Victor Road
Victor 14564
Tel: (716) 924-9101
TWX: 510-254-8542

NORTH CAROLINA

Intel Corp.
2306 W. Meadowview Road
Suite 206
Greensboro 27407
Tel: (919) 294-1541

OHIO

Intel Corp.•
6500 Poe Avenue
Dayton 45414
Tel: (513) 890-5350
TWX: 810-450-2528

Intel Corp.•
Chagrin-Brainard Bldg., No. 300
28001 Chagrin Boulevard
Cleveland 44122
Tel: (216) 464-6915
TWX: 810-427-9298

OKLAHOMA

Intel Corp.
4157 S. Harvard Avenue
Suite 123
Tulsa 74135
Tel: (918) 749-8688

OREGON

Intel Corp.
10700 S.W. Beaverton
Hillsdale Highway
Suite 22
Beaverton 97005
Tel: (503) 641-8086
TWX: 910-467-8741

PENNSYLVANIA

Intel Corp.•
510 Pennsylvania Avenue
Fort Washington 19034
Tel: (215) 641-1000
TWX: 510-661-2077

Intel Corp.•
201 Penn Center Boulevard
Suite 301W
Pittsburgh 15235
Tel: (412) 823-4970

Q.E.D. Electronics
300 N. York Road
Hatboro 19040
Tel: (215) 674-9600

TEXAS

Intel Corp.•
12300 Ford Road
Suite 380
Dallas 75234
Tel: (214) 241-8087
TWX: 910-860-5617

Intel Corp.•
7322 S.W. Freeway
Suite 1490
Houston 77074
Tel: (713) 988-8086
TWX: 910-881-2490

Industrial Digital Systems Corp.
5925 Sovereign
Suite 101
Houston 77036
Tel: (713)988-9421

Intel Corp.
313 E. Anderson Lane
Suite 314
Austin 78752
Tel: (512) 454-3628

UTAH

Intel Corp.
268 West 400 South
Salt Lake City 84101
Tel: (801) 533-8086

VIRGINIA

Intel Corp.
1603 Santa Rosa Road
Suite 109
Richmond 23288
Tel: (804) 282-5668

WASHINGTON

Intel Corp.
110 110th Avenue N.E.
Suite 510
Bellevue 98004
Tel: (206) 453-8086
TWX: 910-443-3002

WISCONSIN

Intel Corp.
450 N. Sunnyslope Road
Suite 130
Brookfield 53005
Tel: (414) 784-9060

CANADA

ONTARIO

Intel Semiconductor of Canada, Ltd.
39 Hwy. 7, Bell Mews
Nepean K2H 8R2
Tel: (613) 829-9714
TELEX: 053-4115

Intel Semiconductor of Canada, Ltd.
50 Galaxy Boulevard
Suite 12
Rexdale M9W 4Y5
Tel: (416) 675-2105
TELEX: 06983574

Intel Semiconductor of Canada, Ltd.
201 Consumers Road
Suite 200
Willowdale M2J 4G8
Tel: (416) 494-6831
TELEX: 4946831

QUEBEC

Intel Semiconductor of Canada, Ltd.
3860 Cote Vertu Road
Suite 210
St. Laurent H4R 1V4
Tel: (514) 334-0560
TELEX: 05-824172

•Field Application Location

intel

DOMESTIC DISTRIBUTORS

ALABAMA

†Arrow Electronics, Inc.
3611 Memorial Parkway So.
Huntsville 35405
Tel: (205) 882-2730

†Hamilton/Avnet Electronics
4812 Commercial Drive N.W.
Huntsville 35805
Tel: (205) 837-7210
TWX: 810-726-2162

†Pioneer/Huntsville
1207 Putnam Drive N.W.
Huntsville 35805
Tel: (205) 837-9300
TWX: 810-726-2197

ARIZONA

†Hamilton/Avnet Electronics
505 S. Madison Drive
Tempe 85281
Tel: (602) 231-5140
TWX: 910-950-0077

†Wyle Distribution Group
8155 N. 24th Street
Phoenix 85021
Tel: (602) 249-2232
TWX: 910-951-4282

CALIFORNIA

†Arrow Electronics, Inc.
521 Weddell Drive
Sunnyvale 94086
Tel: (408) 745-6600
TWX: 910-339-9371

†Arrow Electronics, Inc.
19748 Dearborn Street
Chatsworth 91311
Tel: (213) 701-7500
TWX: 910-493-2086

†Hamilton/Avnet Electronics
350 McCormick Avenue
Costa Mesa 92626
Tel: (714) 754-6051
TWX: 910-595-1928

†Hamilton/Avnet Electronics
19515 So. Vermont Avenue
Torrance 90502
Tel: (213) 615-3909
TWX: 910-349-6263

†Hamilton/Avnet Electronics
1175 Bordeaux Drive
Sunnyvale 94086
Tel: (408) 743-3300
TWX: 910-339-9332

†Hamilton/Avnet Electronics
4545 Viewridge Avenue
San Diego 92123
Tel: (714) 641-1850
TWX: 910-595-2638

†Hamilton/Avnet Electronics
10912 W. Washington Boulevard
Culver City 20230
Tel: (213) 558-2458
TWX: 910-340-6364

†Hamilton Avnet/Electronics
21050 Erwin Street
Woodland Hills 91367
Tel: (213) 883-0000
TWX: 910-494-2207

†Hamilton Electro Sales
3170 Pullman Street
Costa Mesa 92626
Tel: (714) 641-4109
TWX: 910-595-2638

†Hamilton/Avnet Electronics
4103 Northgate Boulevard
Sacramento 95834
Tel: (916) 920-3150

Kierulff Electronics, Inc.
3969 E. Bayshore Road
Palo Alto 94303
Tel: (415) 968-6292
TWX: 910-379-6430

Kierulff Electronics, Inc.
14101 Franklin Avenue
Tustin 92680
Tel: (714) 731-5711
TWX: 910-595-2599

Kierulff Electronics, Inc.
2585 Commerce Way
Los Angeles 90040
Tel: (213) 725-0325
TWX: 910-580-3666

†Wyle Distribution Group
124 Maryland Street
El Segundo 90245
Tel: (213) 322-8100
TWX: 910-348-7140 or 7111

CALIFORNIA (Cont'd)

†Wyle Distribution Group
9525 Chesapeake Drive
San Diego 92123
Tel: (714) 565-9171
TWX: 910-335-1590

†Wyle Distribution Group
3000 Bowers Avenue
Santa Clara 95051
Tel: (408) 727-2500
TWX: 910-338-0296

†Wyle Distribution Group
17872 Cowan Avenue
Irvine 92714
Tel: (714) 641-1600
TWX: 910-595-1572

COLORADO

†Wyle Distribution Group
451 E. 124th Avenue
Thornton 80241
Tel: (303) 457-9953
TWX: 910-936-0770

†Hamilton/Avnet Electronics
8765 E. Orchard Road
Suite 708
Englewood 80111
Tel: (303) 740-1017
TWX: 910-935-0787

CONNECTICUT

†Arrow Electronics, Inc.
12 Beaumont Road
Wallingford 06492
Tel: (203) 265-7741
TWX: 710-476-0162

†Hamilton/Avnet Electronics
Commerce Industrial Park
Commerce Drive
Danbury 06810
Tel: (203) 797-2800
TWX: 710-456-9974

†Harvey Electronics
112 Main Street
Norwalk 06851
Tel: (203) 853-1515
TWX: 710-468-3373

FLORIDA

†Arrow Electronics, Inc.
1001 N.W. 62nd Street
Suite 108
Ft. Lauderdale 33309
Tel: (305) 776-7790
TWX: 510-955-9456

†Arrow Electronics, Inc.
50 Woodlake Drive W.
Bldg. B
Palm Bay 32905
Tel: (305) 725-1480
TWX: 510-959-6337

†Hamilton/Avnet Electronics
6801 N.W. 15th Way
Ft. Lauderdale 33309
Tel: (305) 971-2900
TWX: 510-956-3097

†Hamilton/Avnet Electronics
3197 Tech. Drive North
St. Petersburg 33702
Tel: (813) 576-3930
TWX: 810-863-0374

†Pioneer/Alta Monte Springs
221 N. Lake Blvd.
Suite 412
Alta Monte Springs 32701
Tel: (305) 859-3600
TWX: 810-853-0284

†Pioneer/Ft. Lauderdale
1500 62nd Street N.W.
Suite 506
Ft. Lauderdale 33309
Tel: (305) 771-7520
TWX: 510-955-9653

GEORGIA

†Arrow Electronics, Inc
2979 Pacific Drive
Norcross 30071
Tel: (404) 449-8252
TWX: 810-766-0439

†Hamilton/Avnet Electronics
5825 D. Peachtree Corners
Norcross 30092
Tel: (404) 447-7500
TWX: 810-766-0432

†Pioneer/Georgia
5835B Peachtree Corners E
Norcross 30092
Norcross 30092
Tel: (404) 448-1711
TWX: 810-766-4515

ILLINOIS

†Arrow Electronics, Inc.
2000 E. Alonquin Street
Schaumberg 60195
Tel: (312) 397-3440
TWX: 910-291-3544

†Hamilton/Avnet Electronics
1130 Thorndale Avenue
Bensenville 60106
Tel: (312) 860-7780
TWX: 910-227-0060

†Pioneer/Chicago
1551 Carmen Drive
Elk Grove Village 60007
Tel: (312) 437-9680
TWX: 910-222-1834

INDIANA

†Arrow Electronics, Inc.
2718 Rand Road
Indianapolis 46241
(317) 243-9353
TWX: 810-341-3119

†Hamilton/Avnet Electronics
485 Gradle Drive
Carmel 46032
Tel: (317) 844-9333
TWX: 810-260-3966

†Pioneer/Indiana
6408 Castleplace Drive
Indianapolis 46250
Tel: (317) 849-7300
TWX: 810-260-1794

KANSAS

†Hamilton/Avnet Electronics
9219 Quivera Road
Overland Park 66215
Tel: (913) 888-8900
TWX: 910-743-0005

MARYLAND

†Hamilton/Avnet Electronics
6822 Oak Hall Lane
Columbia 21045
Tel: (301) 995-3500
TWX: 710-862-1861

†Mesa Technology Corporation
16021 Industrial Drive
Gaithersburg 20877
Tel: (301) 948-4350TWX: 710-828-9702

†Pioneer
9100 Gaither Road
Gaithersburg 20877
Tel: (301) 948-0710
TWX: 710-828-0545

MASSACHUSETTS

†Arrow Electronics, Inc.
1 Arrow Drive
Woburn 01801
Tel: (617) 933-8130
TWX: 710-393-6770

†Hamilton/Avnet Electronics
50 Tower Office Park
Woburn 01801
Tel: (617) 935-9700
TWX: 710-393-0382

†Harvey/Boston
44 Hartwell Avenue
Lexington 02173
Tel: (617) 863-1200
TWX: 710-326-6617

MICHIGAN

†Arrow Electronics, Inc.
3810 Varsity Drive
Ann Arbor 48104
Tel: (313) 971-8220
TWX: 810-223-6020

†Pioneer/Michigan
13485 Stamford
Livonia 48150
Tel: (313) 525-1800
TWX: 810-242-3271

†Hamilton/Avnet Electronics
32487 Schoolcraft Road
Livonia 48150
Tel: (313) 522-4700
TWX: 810-242-8775

†Hamilton/Avnet Electronics
2215 29th Street S.E.
Space A5
Grand Rapids 49508
Tel: (616) 243-8805
TWX: 810-273-6921

MINNESOTA

†Arrow Electronics, Inc.
5230 W. 73rd Street
Edina 55435
Tel: (612) 830-1800
TWX: 910-576-3125

†Hamilton/Avnet Electronics
10300 Bren Road East
Minnetonka 55343
Tel: (612) 932-0600
TWX: (910) 576-2720

†Pioneer/Twin Cities
10203 Bren Road East
Minnetonka 55343
Tel: (612) 935-5444
TWX: 910-576-2738

MISSOURI

†Arrow Electronics, Inc.
2380 Schuetz
St. Louis 63141
Tel: (314) 567-6888
TWX: 910-764-0882

†Hamilton/Avnet Electronics
13743 Shoreline Court
Earth City 63045
Tel: (314) 344-1200
TWX: 910-762-0684

NEW HAMPSHIRE

†Arrow Electronics, Inc.
1 Perimeter Road
Manchester 03103
Tel: (603) 668-6968
TWX: 710-220-1684

NEW JERSEY

†Arrow Electronics, Inc.
Pleasant Valley Avenue
Moorestown 08057
Tel: (215) 928-1800
TWX: 710-897-0829

†Arrow Electronics, Inc.
2 Industrial Road
Fairfield 07006
Tel: (201) 575-5300
TWX: 710-998-2206

†Hamilton/Avnet Electronics
1 Keystone Avenue
Bldg. 36
Cherry Hill 08003
Tel: (609) 424-0110
TWX: 710-940-0262

†Hamilton/Avnet Electronics
10 Industrial
Fairfield 07006
Tel: (201) 575-3390
TWX: 710-734-4388

†Harvey Electronics
45 Route 46
Pinebrook 07058
Tel: (201) 575-3510
TWX: 710-734-4382

†MTI Systems Sales
383 Route 46 W
Fairfield 07006
Tel: (201) 227-5552

NEW MEXICO

†Alliance Electronics Inc.
11030 Cochiti S.E.
Albuquerque 87123
Tel: (505) 292-3360
TWX: 910-989-1151

†Hamilton/Avnet Electronics
2524 Baylor Drive S.E.
Albuquerque 87106
Tel: (505) 765-1500
TWX: 910-989-0614

NEW YORK

†Arrow Electronics, Inc.
900 Broad Hollow Road
Farmingdale 11735
Tel: (516) 694-6800
TWX: 510-224-6126

†Arrow Electronics, Inc.
3000 South Winton Road
Rochester 14623
Tel: (716) 275-0300
TWX: 510-253-4766

†Arrow Electronics, Inc.
7705 Maltage Drive
Liverpool 13088
Tel: (315) 652-1000
TWX: 710-545-0230

†Arrow Electronics, Inc.
20, Oser Avenue
Hauppauge 11788
Tel: (516) 231-1000
TWX: 510-227-6623

†Microcomputer System Technical Demonstrator Centers

DOMESTIC DISTRIBUTORS

NEW YORK (Cont'd)

†Hamilton/Avnet Electronics
333 Metro Park
Rochester 14623
Tel: (716) 475-9130
TWX: 510-253-5470

†Hamilton/Avnet Electronics
16 Corporate Circle
E. Syracuse 13057
Tel: (315) 437-2641
TWX: 710-541-1560

†Hamilton/Avnet Electronics
5 Hub Drive
Melville, Long Island 11747
Tel: (516) 454-6000
TWX: 510-224-6166

†Harvey Electronics
P.O. Box 1208
Binghamton 13902
Tel: (607) 748-8211
TWX: 510-252-0893

†Harvey Electronics
60 Crossway Park West
Woodbury, Long Island 11797
Tel: (516) 921-8700
TWX: 510-221-2184

†Harvey/Rochester
840 Fairport Park
Fairport 14450
Tel: (716) 381-7070
TWX: 510-253-7001

†MTI Systems Sales
38 Harbor Park Drive
Port Washington 11050
Tel: (516) 621-6200
TWX: 510-223-0846

NORTH CAROLINA

†Arrow Electronics, Inc.
938 Burke Street
Winston-Salem 27101
Tel: (919) 725-8711
TWX: 510-931-3169

†Hamilton/Avnet Electronics
3510 Spring Forest Drive
Raleigh 27604
Tel: (919) 878-0819
TWX: 510-928-1836

†Pioneer/Carolina
103 Industrial Avenue
Greensboro 27406
Tel: (919) 273-4441
TWX: 510-925-1114

OHIO

†Arrow Electronics, Inc.
7620 McEwen Road
Centerville 45459
Tel: (513) 435-5563
TWX: 810-459-1611

†Arrow Electronics, Inc.
6238 Cochran Road
Solon 44139
Tel: (216) 248-3990
TWX: 810-427-9409

†Hamilton/Avnet Electronics
954 Senate Drive
Dayton 45459
Tel: (513) 433-0610
TWX: 810-450-2531

†Hamilton/Avnet Electronics
4588 Emery Industrial Parkway
Warrensville Heights 44128
Tel: (216) 831-3500
TWX: 810-427-9452

OHIO (Cont'd)

†Pioneer/Dayton
4433 Interpoint Boulevard
Dayton 45424
Tel: (513) 236-9900
TWX: 810-459-1622

†Pioneer/Cleveland
4800 E. 131st Street
Cleveland 44105
Tel (216) 587-3600
TWX: 810-422-2211

OKLAHOMA

†Arrow Electronics, Inc.
4719 S. Memorial Drive
Tulsa 74145
Tel: (918) 665-7700

OREGON

†Almac Electronics Corporation
8022 S.W. Nimbus, Bldg. 7
Beaverton 97005
Tel: (503) 641-9070
TWX: 910-467-8743

†Hamilton/Avnet Electronics
6024 S.W. Jean Road
Bldg. C, Suite 10
Lake Oswego 97034
Tel: (503) 635-7848
TWX: 910-455-8179

PENNSYLVANIA

†Arrow Electronics, Inc.
650 Seco Road
Monroeville 15146
Tel: (412) 856-7000

†Pioneer/Pittsburgh
259 Kappa Drive
Pittsburgh 15238
Tel: (412) 782-2300
TWX: 710-795-3122

†Pioneer/Delaware Valley
261 Gibralter Road
Horsham 19044
Tel: (215) 674-4000
TWX: 510-665-6778

TEXAS

†Arrow Electronics, Inc.
13715 Gama Road
Dallas 75234
Tel: (214) 386-7500
TWX: 910-860-5377

†Arrow Electronics, Inc.
10899 Kinghurst
Suite 100
Houston 77099
Tel: (713) 530-4700
TWX: 910-880-4439

†Arrow Electronics, Inc. 10125
Metropolitan
Austin 78758
Tel: (512) 835-4100
TWX: 910-874-1348

†Hamilton/Avnet Electronics
2401 Rutland
Austin 78757
Tel: (512) 837-8911
TWX: 910-874-1319

†Hamilton/Avnet Electronics
2111 W. Walnut Hill Lane
Irving 75062
Tel: (214) 659-4100
TWX: 910-860-5929

TEXAS (Cont'd)

†Hamilton/Avnet Electronics
8750 West Park
Hosuton 77063
Tel: (713) 780-1771
TWX: 910-881-5523

†Pioneer/Austin
9901 Burnet Road
Austin 78758
Tel: (512) 835-4000
TWX: 910-874-1323

†Pioneer/Dallas
13710 Omega Road
Dallas 75234
Tel: (214) 386-7300
TWX: 910-850-5563

†Pioneer/Houston
5853 Point West Drive
Houston 77036
Tel: (713) 988-5555
TWX: 910-881-1606

UTAH

†Hamilton/Avnet Electronics
1585 West 2100 South
Salt Lake City 84119
Tel: (801) 972-2800
TWX: 910-925-4018

Wyle Distribution Group
1959 South 4130 West, Unit B
Salt Lake City 84104
Tel: (801) 974-9953

WASHINGTON

†Almac Electronics Corporation
14360 S.E. Eastgate Way
Bellevue 98007
Tel: (206) 643-9992
TWX: 910-444-2067

†Arrow Electronics, Inc.
14320 N.E. 21st Street
Bellevue 98007
Tel: (206) 643-4800
TWX: 910-444-2017

†Hamilton/Avnet Electronics
14212 N.E. 21st Street
Bellevue 98005
Tel: (206) 453-5874
TWX: 910-443-2469

WISCONSIN

†Arrow Electronics, Inc.
430 W. Rausson Avenue
Oakcreek 53154
Tel: (414) 764-6600
TWX: 910-262-1193

†Hamilton/Avnet Electronics
2975 Moorland Road
New Berlin 53151
Tel: (414) 784-4510
TWX: 910-262-1182

CANADA

ALBERTA

†Hamilton/Avnet Electronics
2816 21st Street N.E.
Calgary T2E 6Z3
Tel: (403) 230-3586
TWX: 03-827-642

†L.A. Varah, Ltd.
4742 14th Street N.E.
Calgary T2D 6L7
Tel: (403)230-1235
TWX: 038-258-97

ALBERTA (Cont'd)

Zentronics
Bay #1
3300 14th Avenue N.E.
Calgary T2A 6J4
Tel: (403) 272-1021

BRITISH COLUMBIA

L.A. Varah, Ltd.
2077 Alberta Street
Vancouver V5Y 1C4
Tel: (604) 873-3211
TWX: 610-929-1068

Zentronics
108-11400 Bridgeport Road
Richmond V6X 1T2
Tel: (604) 273-5575
TWX: 04-5077-89

MANITOBA

L.A. Varah, Ltd.
12-1832 King Edward Street
Winnipeg R2R ON1
Tel: (204) 633-6190
TWX: 07-55-365

Zentronics
590 Berry Street
Winnipeg R3H OS1
Tel: (204) 775-8661

ONTARIO

Hamilton/Avnet Electronics
6845 Rexwood Road
Units G & H
Mississauga L4V 1R2
Tel: (416) 677-7432
TWX: 610-492-8867

Hamilton/Avnet Electronics
210 Colonnade Road South
Nepean K2E 7L5
Tel: (613) 226-1700
TWX: 05-349-71

L.A. Varah, Ltd.
505 Kenora Avenue
Hamilton L8E 3P2
Tel: (416) 561-9311
TWX: 061-8349

Zentronics
8 Tilbury Court
Brampton L6T 3T4
Tel: (416) 451-9600
TWX: 06-976-78

Zentronics
564/10 Weber Street North
Waterloo N2L 5C6
Tel: (519) 884-5700

Zentronics
590 Berry Street
Winnipeg R3H OS1
Tel: (204) 775-8661

QUEBEC

Hamilton/Avnet Electronics
2670 Sabourin Street
St. Laurent H4S 1M2
Tel: (514) 331-6443
TWX: 610-421-3731

Zentronics
505 Locke Street
St. Laurent H4T 1X7
Tel: (514) 735-5361
TWX: 05-827-535

†Microcomputer System Technical Demonstrator Centers

intel

EUROPEAN SALES OFFICES

BELGIUM

Intel' Corporation S.A.
Parc Seny
Rue du Moulin a Papier 51
Boite 1
B-1160 Brussels
Tel: (02)661 07 11
TELEX: 28414

DENMARK

Intel Denmark A/S*
Lyngbyvej 32F 2nd Floor
DK-2100 Copenhagen East
Tel: (01) 18 20 00
TELEX: 19567

FINLAND

Intel Finland OY
Hameentie 103
SF - 00550 Helsinki 55
Tel: 0/716 955
TELEX: 123 332

FRANCE

Intel Corporation, S.A.R.L.*
5 Place de la Balance
Silic 223
94528 Rungis Cedex
Tel: (01) 687 22 21
TELEX: 270475

FRANCE (Cont'd)

Intel Corporation, S.A.R.L.
Immeuble BBC
4 Quai des Etroits
69005 Lyon
Tel: (7) 842 40 89
TELEX: 305153

WEST GERMANY

Intel Semiconductor GmbH*
Seidlstrasse 27
D-8000 Muenchen 2
Tel: (89) 53891
TELEX: 05-23177 INTL D

Intel Semiconductor GmbH*
Mainzer Strasse 75
D-6200 Wiesbaden 1
Tel: (6121) 70 08 74
TELEX: 04186183 INTW D

Intel Semiconductor GmbH
Brueckstrasse 61
7012 Fellbach
West Germany
Tel: (711) 58 00 82
TELEX: 7254826 INTS D

Intel Semiconductor GmbH*
Hohenzollern Strasse 5*
3000 Hannover 1
Tel: (511) 34 40 81
TELEX: 923625 INTH D

Intel Semiconductor GmbH
Ober-Ratherstrasse 2
D-4000 Dusseldorf 30
Tel: (211) 65 10 54
TELEX: 08-58977 INTL D

ISRAEL

Intel Semiconductor Ltd.*
P.O. Box 1659
Haifa
Tel: 4/524
TELEX: 46511

ITALY

Intel Corporation Italia Spa*
Milanofiori, Palazzo E
20094 Assago (Milano)
Tel: (02) 824 00 06
TELEX: 315183 INTMIL

NETHERLANDS

Intel Semiconductor Nederland B.V.*
Alexanderpoort Building
Marten Meesweg 93
3068 Rotterdam
Tel: (10) 21 23 77
TELEX: 22283

NORWAY

Intel Norway A/S
P.O. Box 92
Hvamveien 4
N-2013
Skjetten
Tel: (2) 742 420
TELEX: 18018

SWEDEN

Intel Sweden A.B.*
Box 20092
Archimedesvagen 5
S-16120 Bromma
Tel: (08) 98 53 85
TELEX: 12261

SWITZERLAND

Intel Semiconductor A.G.*
Forchstrasse 95
CH 8032 Zurich
Tel: (01) 55 45 02
TELEX: 57989 ICH CH

UNITED KINGDOM

Intel Corporation (U.K.) Ltd.*
5 Hospital Street
Nantwich, Cheshire CW5 5RE
Tel: (0270) 626 560
TELEX: 36620

Intel Corporation (U.K.) Ltd.*
Pipers Way
Swindon, Wiltshire SN3 1RJ
Tel: (0793) 488 388
TELEX: 444447 INT SWN

*Field Application Location

EUROPEAN DISTRIBUTORS/REPRESENTATIVES

AUSTRIA

Bacher Elektronische Geraete GmbH
Rotemuehlgasse 26
A 1120 Vienna
Tel: (222) 83 63 96
TELEX: 11532 BASAT A

BELGIUM

Inelco Belgium S.A.
Ave. des Croix de Guerre 94
B1120 Brussels
Tel: (02) 216 01 60
TELEX: 25441

DENMARK

MultiKomponent A/S
Fabriksparken 31
DK-2600 Gloskrup
Tel: (02) 45 66 45
TX: 33355

Scandinavian Semiconductor
Supply A/S
Nannasgade 18
DK-2200 Copenhagen
Tel: (01) 83 50 90
TELEX: 19037

FINLAND

Oy Fintronic AB
Melkonkatu 24 A
SF-00210
Helsinki 21
Tel: (0) 692 60 22
TELEX: 124 224 Ftron SF

FRANCE

Generim
Z.I. de Courtaboeuf
Avenue de la Baltique
91943 Les Ulis Cedex-B.P.88
Tel: (6) 907 78 79
TELEX: F691700

Jermyn S.A.
rue Jules Ferry 35
93170 Bagnolet
Tel: (1) 859 04 04
TELEX: 21810 F

Metrologie
La Tour d' Asnieres
1, Avenue Laurent Cely
92606-Asnieres
Tel: (1) 791 44 44
TELEX: 611-448

FRANCE (Cont'd)

Tekelec Airtronic
Cite des Bruyeres
Rue Carle Vernet
F-92310 Sevres
Tel: (01) 534 75 35
TELEX: 204552

WEST GERMANY

Electronic 2000 Vertriebs A.G.
Neumarkter Strasse 75
D-8000 Munich 80
Tel: (89) 43 40 61
TELEX 522561 EIEC D

Jermyn GmbH
Postfach 1180
Schulstrasse 48
D-6277 Bad Camberg
Tel: (06434) 231
TELEX: 484426 JERM D

Celdis Enatechnik Systems GmbH
Gutenbergstrasse 4
2359 Henstedt-Ulzburg 1
Tel: (04193) 4026
TELEX: 2180260

Proelectron Vertriebs GmbH
Max Planck Strasse 1-3
6072 Dreieich bei Frankfurt
Tel: (6103) 33564
TELEX: 417983

IRELAND

Micro Marketing
Glenageary Office Park
Glenageary
Co. Dublin
Tel: (1) 85 62 88
TELEX: 31584

ISRAEL

Eastronics Ltd.
11 Rozanis Street
P.O. Box 39300
Tel Aviv 61390
Tel: (3) 47 51 51
TELEX: 33638

ITALY

Eledra 3S S.P.A.
Viale Elvezia, 18
I 20154 Milano
Tel: (2) 34 97 51
TELEX: 332332

ITALY (Cont'd)

Intesi
Milanfiori Pal. E/5
20090 Assago
Milano
Tel: (02) 82470
TELEX: 311351

NETHERLANDS

Koning & Hartman
Koperwerf 30
P.O. Box 43220
2544 EN's Gravenhage
Tel: 31 (70) 210.101
TELEX: 31528

NORWAY

Nordisk Elektronic (Norge) A/S
Postoffice Box 122
Smedsvingen 4
1364 Hvalstad
Tel: (2) 786 210
TELEX: 77546

PORTUGAL

Ditram
Componentes E Electronica LDA
Av. Miguel Bombarda, 133
P1000 Lisboa
Tel: (19) 545 313
TELEX: 14182 Brieks-P

SPAIN

Interface S.A.
Ronda San Pedro 22,3
Barcelona 10
Tel: (3) 301 78 51
TWX: 51508

ITT SESA
Miguel Angel 23-3
Madrid 10
Tel: (1) 419 54 00
TELEX: 27707

SWEDEN

AB Gosta Backstrom
Box 12009
Alstroemergatan 22
S-10221 Stockholm 12
Tel: (8) 541 080
TELEX: 10135

SWEDEN (Cont'd)

Nordisk Electronik AB
Box 27301
Sandhamnsgatan 71
S-10254 Stockholm
Tel: (8) 635 040
TELEX: 10547

SWITZERLAND

Industrade AG
Gemsenstrasse 2
Postcheck 80 - 21190
CH-8021 Zurich
Tel: (01) 363 23 20
TELEX: 56788 INDEL CH

UNITED KINGDOM

Bytech Ltd.
Unit 57
London Road
Earley, Reading
Berkshire
Tel: (0734) 61031
TELEX: 848215

Comway Microsystems Ltd.
Market Street
UK-Bracknell, Berkshire
Tel: 44 (344) 55333
TELEX: 847201

Jermyn Industries
Vestry Estate
Sevenoaks, Kent
Tel: (0732) 450144
TELEX: 95142

M.E.D.L.
East Lane Road
North Wembley
Middlesex HA9 7PP
Tel: (01) 904 93 07
TELEX: 28817

Rapid Recall, Ltd.
Rapid House/Denmark St
High Wycombe
Berks, England · HP11 2ER
Tel: (0494) 26 271
TELEX: 837931

YUGOSLAVIA

H. R. Microelectronics Enterprises
P.O. Box 5604
San Jose, California 95150
Tel: 408/978-8000
TELEX: 278-559

intel

INTERNATIONAL SALES OFFICES

AUSTRALIA

Intel Semiconductor Pty. Ltd.
Spectrum Building
200 Pacific Highway
Level 6
Crows Nest, NSW, 2089
Australia
Tel: 011-61-2-436-2744
TELEX: 790-20097
FAX: 011-61-2-923-2632

HONG KONG

Intel Semiconductor Ltd.
13/F Hong Kong Trade Centre
161-167 Des Voeux Road Central
Tel: 011-852-5-450-885
TELEX: 63869 ISLHKHX

JAPAN

Intel Japan K.K.
5-6 Tokodai, Toyosato-machi
Tsukuba-gun, Ibaraki-ken 300-26
Tel: 029747-8511
TELEX: 03656-160

JAPAN (Cont'd)

Intel Japan K.K.*
2-1-15 Naka-machi
Atsugi, Kanagawa 243
Tel: 0462-23-3511

Intel Japan K.K.*
2-51-2 Kojima-cho
Chofu, Tokyo 182
Tel: 0424-88-3151

Intel Japan K.K.*
2-69 Hon-cho
Kumagaya, Saitama 360
Tel: 0485-24-6871

JAPAN (Cont'd)

Intel Japan K.K.*
2-4-1 Terauchi
Toyonaka, Osaka 560
Tel: 06-863-1091

Intel Japan K.K.
1-5-1 Marunouchi
Chiyoda-ku, Tokyo 100
Tel: 03-201-3621/3681

Intel Japan K.K.*
1-23-9 Shinmachi
Setagaya-ku, Tokyo 154
Tel: 03-426-2231

*Field Application Location

INTERNATIONAL DISTRIBUTORS/REPRESENTATIVES

ARGENTINA

VLC S.R.L.
Sarmiento 1630, 1 Piso
1042 Buenos Aires
Tel: 35-1201/9242
TELEX: Public Booth 9900 or 9901

Mailing Address
Soimex International Corporation
15 Park Row, Room #1730
New York, New York 10038
(212) 406-3052
Attn: Gaston Briones

AUSTRALIA

Total Electronics
9 Harker Street
Burwood
Victoria 3125
Tel: 61 3 288-4044
TELEX: AA 31261

Mailing Address
Private Bag 250
Burwood, Victoria 3125
Australia

Total Electronics
#1 Johnstone Lane
Lane Cove, N.S.W. 2066
TELEX: 26297

BRAZIL

Icotron S.A.
05110 Av. Mutinga 3650-6 Andar
Pirituba Sao Paulo
Tel: 261-0211
TELEX: 1122274/ICOTBR

CHILE

DIN
AV. VIC MCKENNA 204
Casilla 6055
Santiago
Tel: 227 564
TELEX: 352 003

COLUMBIA

International Computer Machines
Carrera 7 No. 72-34
Apdo. Aereo 19403
Bogota 1
Tel: 211-7282
TELEX: 45716 ICM CO

HONG KONG

Schmidt & Co. Ltd.
Wing on Centre, 28th Floor
111 Connaught Road Central
Tel: 5 8521 222
TELEX: 74766 SCHMC HX

INDIA

Micronic Devices
104/109C, Nirmal Industrial Estate
Sion (E)
Bombay 400022
Tel: 486-170
TELEX: 011-71447 MDEV IN

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

JAPAN (Cont'd)

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) 543-7711/541-7311

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
2nd Floor, Government Pension Bldg.
1-589, Yoido-Dong
Youngdungpo-Ku
Seoul 150
Tel: 782-8039 or 8049
TELEX: KODIGIT K25 299

NEW ZEALAND

McLean Information Technology Ltd.
459 Kyber Pass Road, Newmarket,
P.O. Box 9464, Newmarket
Auckland 1, New Zealand
Tel: 501-801, 501-219, 587-037
TELEX: NZ21570 THERMAL

SINGAPORE

General Engineers Corporation Pty. Ltd.
18 Pasir Panjang Road
11-05/08 PSA Multi Storey Complex
Singapore 0511
Tel: 011-65-271-3163
TELEX: RS23987 GENERCO

SOUTH AFRICA

Electronic Building Elements, Pty. Ltd.
P.O. Box 4609
Hazelwood, Pretoria 0001
Tel: 011-27-12-46-9221 or 9227
TELEX: 3-0181 SA

TAIWAN

Taiwan Automation Corp.*
3rd Floor #75, Section 4
Nanking East Road
Taipei
Tel: 771-0940 or 0941
TELEX: 11942 TAIAUTO

YUGOSLAVIA

H. R. Microelectronics Enterprises
P.O. Box 5604
San Jose, California 95150
Tel: (408) 978-8000
TELEX: 278-559

*Field Application Location

intel

U.S. SERVICE OFFICES

CALIFORNIA

Intel Corp.
1350 Shorebird Way
Mt. View 94043
Tel: (415) 968-8211
TWX: 910-339-9279
910-338-0255

Intel Corp.
2000 E. 4th Street
Suite 110
Santa Ana 92705
Tel: (714) 835-5577
TWX: 910-595-2475

Intel Corp.
7670 Opportunity Road
San Diego 92111
Tel: (714) 268-3563

Intel Corp.
5530 N. Corbin Avenue
Suite 120
Tarzana 91356
Tel: (213) 708-0333

COLORADO

Intel Corp.
650 South Cherry
Suite 720
Denver 80222
Tel: (303) 321-8086
TWX: 910-931-2289

CONNECTICUT

Intel Corp.
36 Padanaram Road
Danbury 06810
Tel: (203) 792-8366

FLORIDA

Intel Corp.
1500 N.W. 62nd Street
Suite 104
Ft. Lauderdale 33309
Tel: (305) 771-0600
TWX: 510-956-9407

Intel Corp.
500 N. Maitland Avenue
Suite 205
Maitland 32751
Tel: (305) 628-2393
TWX: 810-853-9219

Intel Corp.
5151 Adanson Street
Orlando 32804
Tel: (305) 628-2393

GEORGIA

Intel Corp.
3300 Holcombe Bridge Road
Suite 225
Norcross 30092
Tel: (404) 441-1171

ILLINOIS

Intel Corp.
2550 Golf Road
Suite 815
Rolling Meadows 60008
Tel: (312) 981-7270
TWX: 910-253-1825

KANSAS

Intel Corp.
8400 W. 110th Street
Suite 170
Overland Park 66210
Tel: (913) 642-8080

MARYLAND

Intel Corp. 7257 Parkway Drive
Hanover 21076
Tel: (301) 796-7500
TWX: 710-862-1944

MASSACHUSETTS

Intel Corp.
27 Industrial Avenue
Chelmsford 01824
Tel: (617) 256-1800
TWX: 710-343-6333

MICHIGAN

Intel Corp.
26500 Northwestern Highway
Suite 401
Southfield 48075
Tel: (313) 354-1540
TWX: 810-244-4915

MINNESOTA

Intel Corp.
7401 Metro Boulevard
Suite 355
Edina 55435
Tel: (612) 835-6722
TWX: 910-567-2867

MISSOURI

Intel Corp.
4203 Earth City Expressway
Suite 143
Earth City 63045
Tel: (314) 291-2015

NEW JERSEY

Intel Corp.
385 Sylvan Avenue
Englewood Cliffs 07632
Tel: (201) 567-0820
TWX: 710-991-8593

NEW YORK

Intel Corp.
2255 Lyell Avenue
Rochester 14606
Tel: (716) 254-6120

NORTH CAROLINA

Intel Corp.
5600 Executive Drive
Suite 113
Charlotte 28212
Tel: (704) 568-8966

Intel Corp.
2306 W. Meadowview Road
Suite 206
Greensboro 27407
Tel: (919) 294-1541

OHIO

Intel Corp.
Chagrin-Brainard Bldg.
Suite 305
28001 Chagrin Boulevard
Cleveland 44122
Tel: (216) 464-6915
TWX: 810-427-9298

Intel Corp.
6500 Poe Avenue
Dayton 45414
Tel: (800) 325-4415
TWX: 810-450-2528

OKLAHOMA

Intel Corp.
4157 S. Harvard
Suite 123
Tulsa 74101
Tel: (918) 744-8068

OREGON

Intel Corp.
10700 S.W. Beaverton-Hillsdale
Highway
Suite 22
Beaverton 97005
Tel: (503) 641-8086
TWX: 910-467-8741

PENNSYLVANIA

Intel Corp.
500 Pennsylvania Avenue
Fort Washington 19034
Tel: (215) 641-1000
TWX: 510-661-2077

Intel Corp.
201 Penn Center Boulevard
Suite 301 W
Pittsburgh 15235
Tel: (313) 354-1540

TEXAS

Intel Corp.
313 E. Anderson Lane
Suite 314
Austin 78752
Tel: (512)454-3628
TWX: 910-874-1347

Intel Corp.
12300 Ford Road
Suite 380
Dallas 75234
Tel: (214) 241-8087
TWX: 910-860-5617

Intel Corp.
7322 S.W. Freeway
Suite 1490
Houston 77074
Tel: (713) 988-8088
TWX: 910-881-2490

VIRGINIA

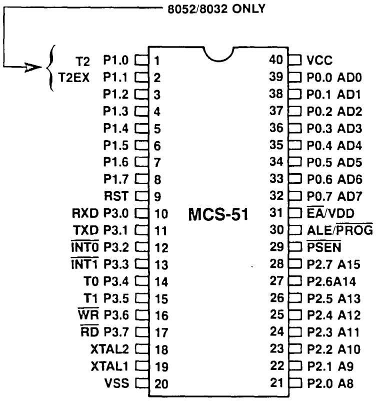
Intel Corp.
7700 Leesburg Pike
Suite 412
Falls Church 22043
Tel: (703) 734-9707
TWX: 710-931-0625

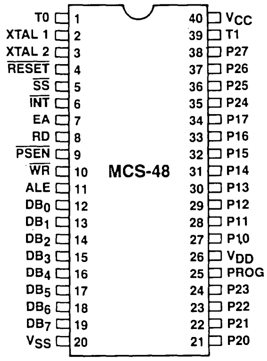
WASHINGTON

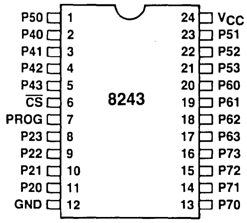
Intel Corp.
110 110th Avenue N.E.
Suite 510
Bellevue 98004
Tel: 1-800-538-0662
TWX: 910-443-3002

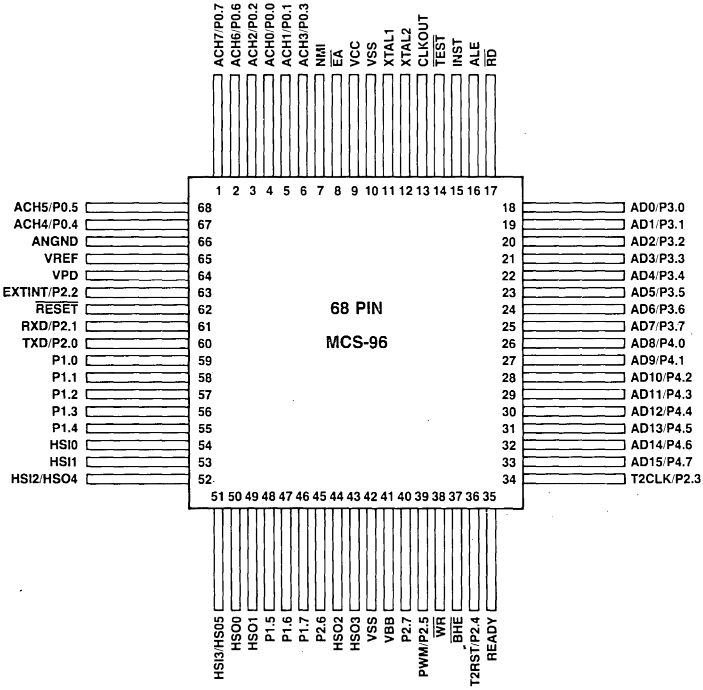
WISCONSIN

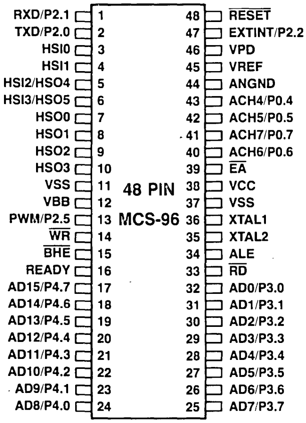
Intel Corp.
150 S. Sunnyslope Road
Suite 148
Brookfield 53005
Tel: (414) 784-9060


8052/8032 ONLY
MCS-51
T2 P1.0 1
T2EX P1.1 2
P1.2 3
P1.3 4
P1.4 5
P1.5 6
P1.6 7
P1.7 8
RST 9
RXD P3.0 10
TXD P3.1 11
INT0 P3.2 12
INT1 P3.3 13
T0 P3.4 14
T1 P3.5 15
WR P3.6 16
RD P3.7 17
XTAL2 18
XTAL1 19
VSS 20
40 VCC
39 P0.0 AD0
38 P0.1 AD1
37 P0.2 AD2
36 P0.3 AD3
35 P0.4 AD4
34 P0.5 AD5
33 P0.6 AD6
32 P0.7 AD7
31 EA/VDD
30 ALE/PROG
29 PSEN
28 P2.7 A15
27 P2.6A14
26 P2.5 A13
25 P2.4 A12
24 P2.3 A11
23 P2.2 A10
22 P2.1 A9
21 P2.0 A8


MCS-48
T0 1
XTAL 1 2
XTAL 2 3
RESET 4
SS 5
INT 6
EA 7
RD 8
PSEN 9
WR 10
ALE 11
DB0 12
DB1 13
DB2 14
DB3 15
DB4 16
DB5 17
DB6 18
DB7 19
VSS 20
40 VCC
39 T1
38 P27
37 P26
36 P25
35 P24
34 P17
33 P16
32 P15
31 P14
30 P13
29 P12
28 P11
27 P10
26 VDD
25 PROG
24 P23
23 P22
22 P21
21 P20


8243
P50 1
P40 2
P41 3
P42 4
P43 5
CS 6
PROG 7
P23 8
P22 9
P21 10
P20 11
GND 12
24 VCC
23 P51
22 P52
21 P53
20 P60
19 P61
18 P62
17 P63
16 P73
15 P72
14 P71
13 P70


68 PIN
MCS-96
1 ACH7/P0.7
2 ACH6/P0.6
3 ACH2/P0.2
4 ACH0/P0.0
5 ACH1/P0.1
6 ACH3/P0.3
7 NMI
8 EA
9 VCC
10 VSS
11 XTAL1
12 XTAL2
13 CLKOUT
14 TEST
15 INST
16 ALE
17 RD
18 AD0/P3.0
19 AD1/P3.1
20 AD2/P3.2
21 AD3/P3.3
22 AD4/P3.4
23 AD5/P3.5
24 AD6/P3.6
25 AD7/P3.7
26 AD8/P4.0
27 AD9/P4.1
28 AD10/P4.2
29 AD11/P4.3
30 AD12/P4.4
31 AD13/P4.5
32 AD14/P4.6
33 AD15/P4.7
34 T2CLK/P2.3
35 READY
36 T2RST/P2.4
37 BHE
38 WR
39 PWM/P2.5
40 P2.7
41 VBB
42 VSS
43 HSO3
44 HSO2
45 P2.6
46 P1.7
47 P1.6
48 P1.5
49 HSO1
50 HSO0
51 HSI3/HSO5
52 HSI2/HSO4
53 HSI1
54 HSI0
55 P1.4
56 P1.3
57 P1.2
58 P1.1
59 P1.0
60 TXD/P2.0
61 RXD/P2.1
62 RESET
63 EXTINT/P2.2
64 VPD
65 VREF
66 ANGND
67 ACH4/P0.4
68 ACH5/P0.5


48 PIN
MCS-96
RXD/P2.1 1
TXD/P2.0 2
HSI0 3
HSI1 4
HSI2/HSO4 5
HSI3/HSO5 6
HSO0 7
HSO1 8
HSO2 9
HSO3 10
VSS 11
VBB 12
PWM/P2.5 13
WR 14
BHE 15
READY 16
AD15/P4.7 17
AD14/P4.6 18
AD13/P4.5 19
AD12/P4.4 20
AD11/P4.3 21
AD10/P4.2 22
AD9/P4.1 23
AD8/P4.0 24
48 RESET
47 EXTINT/P2.2
46 VPD
45 VREF
44 ANGND
43 ACH4/P0.4
42 ACH5/P0.5
41 ACH7/P0.7
40 ACH6/P0.6
39 EA
38 VCC
37 VSS
36 XTAL1
35 XTAL2
34 ALE
33 RD
32 AD0/P3.0
31 AD1/P3.1
30 AD2/P3.2
29 AD3/P3.3
28 AD4/P3.4
27 AD5/P3.5
26 AD6/P3.6
25 AD7/P3.7