

13

Interfacing Data Converters

The microprocessor is a logic device; it processes digital signals that are binary and discontinuous. On the other hand, the real-world physical quantities such as temperature and pressure are continuous. These are represented by equivalent electrical quantities called analog signals. Even though an analog signal may represent a real physical parameter with accuracy, it is difficult to process or store the analog signal for later use without introducing considerable error. Therefore, in microprocessor-based industrial products, it is necessary to translate an analog signal into a digital signal. The electronic circuit that translates an analog signal into a digital signal is called an analog-to-digital (A/D) converter (ADC). Similarly, a digital signal needs to be translated into an analog signal to represent a physical quantity (e.g., to regulate a machine). This translator is called a digital-to-analog (D/A) converter (DAC). Both A/D and D/A are also known as data

converters and are now available as integrated circuits.

This chapter focuses on interfacing data converters with the 8085 microprocessor. First, D/A converters are discussed, including the basic concepts in the conversion process and their interfacing applications. A/D converters are discussed later because some A/D conversion techniques include D/A converters in the conversion process. The chapter includes illustrations of successive approximation converters.

OBJECTIVES

- Explain the functions of data converters.
- Explain the basic circuit of a D/A converter and define the terms *resolution* and *settling time*.
- Calculate the analog output of a D/A converter for a given digital input signal.

- Design a circuit to interface an 8-bit D/A converter with the 8085 microprocessor, and verify the analog output for a digital signal.
- Interface a 10- or 12-bit D/A converter with an 8-bit microprocessor.
- Explain the basic concepts underlying the successive-approximation A/D converter.
- Interface an 8-bit A/D converter with the 8085, using status check and interrupt.

13.1 DIGITAL-TO-ANALOG (D/A) CONVERTERS

Digital-to-Analog converters can be broadly classified in three categories: **current output**, **voltage output**, and **multiplying type**. The current output DAC, as the name suggests, provides current as the output signal. The voltage output DAC internally converts the current signal into the voltage signal. The voltage output DAC is slower than the current output DAC because of the delay in converting the current signal into the voltage signal. However, in many applications, it is necessary to convert current into voltage by using an external operational amplifier. The multiplying DAC is similar to the other two types except its output represents the product of the input signal and the reference source (Figure 13.3 will explain the reference source), and the product is linear over a broad range. Conceptually, there is not much difference between these three types; any DAC can be viewed as a multiplying DAC.

D/A converters are available as **integrated circuits**. Some are specially designed to be compatible with the microprocessor. Typical applications include digital voltmeters, peak detectors, panel meters, programmable gain and attenuation, and stepping motor drive.

13.1.1 Basic Concepts

Figure 13.1(a) shows a block diagram of a 3-bit D/A converter; it has three digital input lines (D_2 , D_1 , and D_0) and one output line for the analog signal. The three input lines can assume eight ($2^3 = 8$) input combinations from 000 to 111, D_2 being the most significant bit (MSB) and D_0 being the least significant bit (LSB). If the input ranges from 0 to 1 V, it can be divided into eight equal parts ($\frac{1}{8}$ V); each successive input is $\frac{1}{8}$ V higher than the previous combination, as shown in Figure 13.1(b).

The following points can be summarized from the graph:

1. The 3-bit D/A converter has eight possible combinations. If a converter has n input lines, it can have 2^n input combinations.
2. If the full-scale analog voltage is 1 V, the smallest unit or the LSB (001_2) is equivalent to $\frac{1}{2^n}$ of 1 V. This is defined as resolution. In this example, the LSB = $\frac{1}{8}$ V.
3. The MSB represents half of the full-scale value. In this example, the MSB (100_2) = $\frac{1}{2}$ V.
4. For the maximum input signal (111_2), the output signal is equal to the value of the full-scale input signal minus the value of the 1 LSB input signal. In this example, the maximum input signal (111_2) represents $\frac{7}{8}$ V.

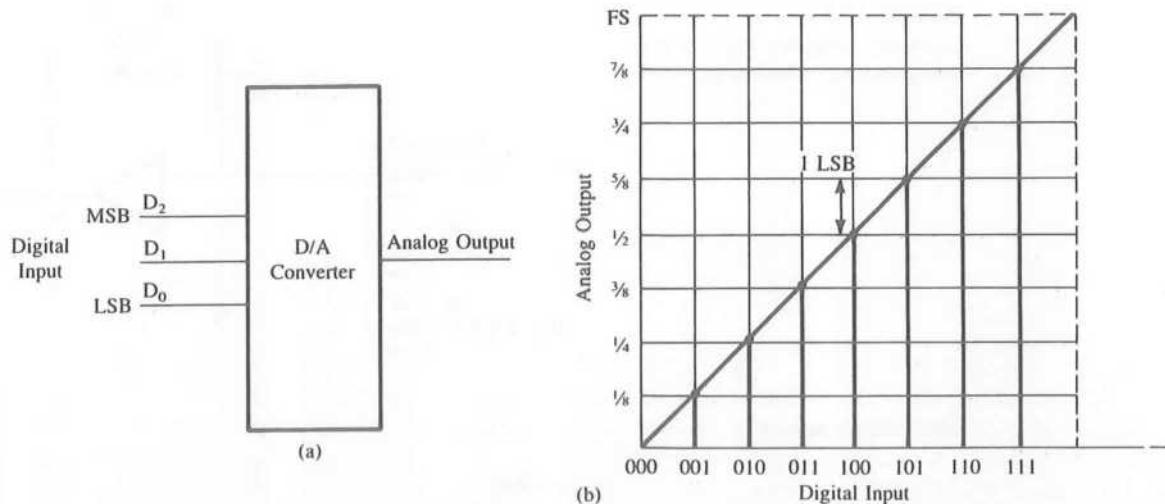


FIGURE 13.1

A 3-Bit D/A Converter: Block Diagram (a) and Digital Input vs. Analog Output (b)

Calculate the values of the LSB, MSB, and full-scale output for an 8-bit DAC for the 0 to 10 V range.

Example
13.1

$$1. \text{ LSB} = \frac{1}{2^8} = \frac{1}{256}$$

For 10 V, $\text{LSB} = 10 \text{ V}/256 = 39 \text{ mV}$

Solution

$$2. \text{ MSB} = \frac{1}{2} \text{ full scale} = 5 \text{ V}$$

$$\begin{aligned} 3. \text{ Full-Scale Output} &= (\text{Full-Scale Value} - 1 \text{ LSB}) \\ &= 10 \text{ V} - 0.039 \text{ V} \\ &= 9.961 \text{ V} \end{aligned}$$

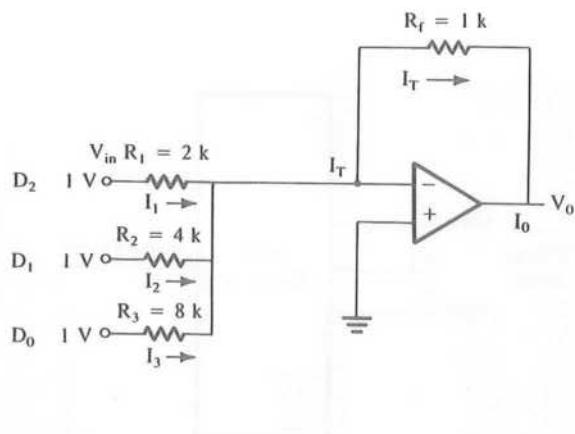
13.1.2 D/A Converter Circuits

Input signals representing appropriate binary values can be simulated by an operational amplifier with a summing network, as shown in Figure 13.2.

The input resistors R_1 , R_2 , and R_3 are selected in binary weighted proportion; each has double the value of the previous resistor. If all three inputs are 1 V, the total output current is

$$\begin{aligned} I_O &= I_T = I_1 + I_2 + I_3 \\ &= \frac{V_{in}}{R_1} + \frac{V_{in}}{R_2} + \frac{V_{in}}{R_3} \\ &= \frac{V_{in}}{1 \text{ k}} \left(\frac{1}{2} + \frac{1}{4} + \frac{1}{8} \right) \\ &= 0.875 \text{ mA} \end{aligned}$$

FIGURE 13.2
Summing Amplifier with Binary
Weighted Input Resistors



The voltage output is

$$\begin{aligned} V_O &= -R_f I_T \\ &= -(1 \text{ k})(0.875 \text{ mA}) \\ &= -0.875 \text{ V} \\ &= |7/8 \text{ V}| \end{aligned}$$

This example shows that for the input = 111₂, the output is equal to either 7/8 mA or 7/8 V, representing the D/A conversion process.

Now we can redraw Figure 13.2 as shown in Figure 13.3(a), where input voltage V_{in} is replaced by V_{Ref} , which can be turned on or off by the switches. The output current I_O can be generalized for any number of bits as

$$I_O = \frac{V_{Ref}}{R} \left(\frac{A_1}{2} + \frac{A_2}{4} + \dots + \frac{A_n}{2^n} \right)$$

where A_1 to A_n can be zero or one.

Figure 13.3(b) shows an illustration of the transistorized switch for input D_0 ; when bit D_0 is high, it will drive the transistor into saturation, and the current is determined by the resistor R_3 with appropriate binary weighting. When bit D_0 is low, the transistor is turned off. The switching speed of the transistor, shown in Figure 13.3(b), determines the settling time of a D/A converter, which is defined as the time necessary for the output to stabilize within $\pm 1/2$ LSB of its final value. The accuracy of the output is dependent on the tolerance of resistor values, and it is generally specified in terms of relative accuracy (also known as linearity)—the difference between the actual output and the expected fraction for a given digital input. The accuracy of the converter is also specified in terms of monotonicity, which guarantees that analog output increases in magnitude with increasing digital code. Most commercial D/A converters are specified as monotonic; this limits the output error within $\pm 1/2$ LSB at each digital input.

The following points can be inferred from the above example:

1. A D/A converter circuit requires three elements: resistor network with appropriate weighting, switches, and a reference source.

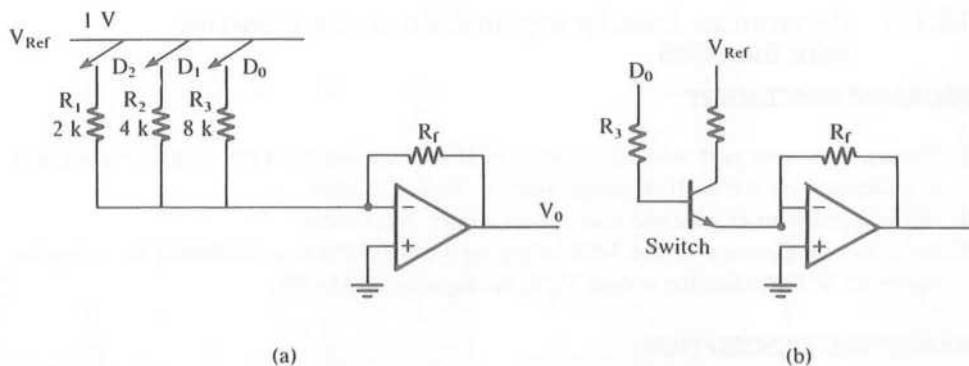


FIGURE 13.3
Simulated D/A Converter (a) and Transistor Switch to Turn On/Off Bit D_0 (b)

2. The output can be a current signal or converted into a voltage signal using an operational amplifier.
3. The time required for conversion, called settling time, is dependent on the response time of the switches and the output amplifier (for a voltage output DAC).

R/2R LADDER NETWORK

One major drawback of designing a DAC as shown in Figure 13.3 is the requirement for various precision resistors. The R/2R ladder network shown in Figure 13.4 uses only two resistor values. The resistors are connected in such a way that for any number of inputs, the total current I_T is in binary proportion. The R/2R ladder network (or a similar network called an inverted ladder) is used commonly in designing integrated D/A converters. The interfacing of 8-bit and 10-bit integrated D/A converters is illustrated in the next two sections.

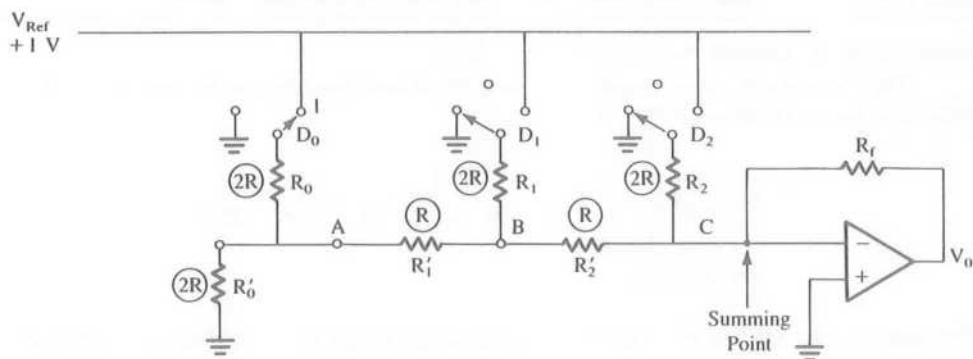


FIGURE 13.4
R/2R Ladder Network

13.1.3 Illustration: Interfacing an 8-Bit D/A Converter with the 8085

PROBLEM STATEMENT

1. Design an output port with the address FFH to interface the 1408 D/A converter that is calibrated for a 0 to 10 V range; refer to Figure 13.5(a).
2. Write a program to generate a continuous ramp waveform.
3. Explain the operation of the 1408 in Figure 13.5(b), which is calibrated for a bipolar range ± 5 V. Calculate the output V_O if the input is 10000000_2 .

HARDWARE DESCRIPTION

The circuit shown in Figure 13.5(a) includes an 8-input NAND gate and a NOR gate (negative AND) as the address decoding logic, the 74LS373 as a latch, and an industry-standard 1408 D/A converter. The address lines A_7 - A_0 are decoded using the 8-input NAND gate, and the output of the NAND gate is combined with the control signal IOW . When the microprocessor sends the address FFH, the output of the negative AND gate enables the latch, and the data bits are placed on the input lines of the converter for conversion.

The 1408 is an 8-bit D/A converter compatible with TTL and CMOS logic, with the settling time around 300 ns. It has eight input data lines A_1 (MSB) through A_8 (LSB); the convention of labeling MSB to LSB is opposite to that of what is normally used for the data bus in the microprocessor. It requires 2 mA reference current for full-scale input and two power supplies $V_{CC} = +5$ V and $V_{EE} = -15$ V (V_{EE} can range from -5 V to -15 V).

The total reference current source is determined by the resistor R_{14} and the voltage V_{Ref} . The resistor R_{15} is generally equal to R_{14} to match the input impedance of the reference source. The output I_O is calculated as follows:

$$I_O = \frac{V_{Ref}}{R_{14}} \left(\frac{A_1}{2} + \frac{A_2}{4} + \frac{A_3}{8} + \frac{A_4}{16} + \frac{A_5}{32} + \frac{A_6}{64} + \frac{A_7}{128} + \frac{A_8}{256} \right)$$

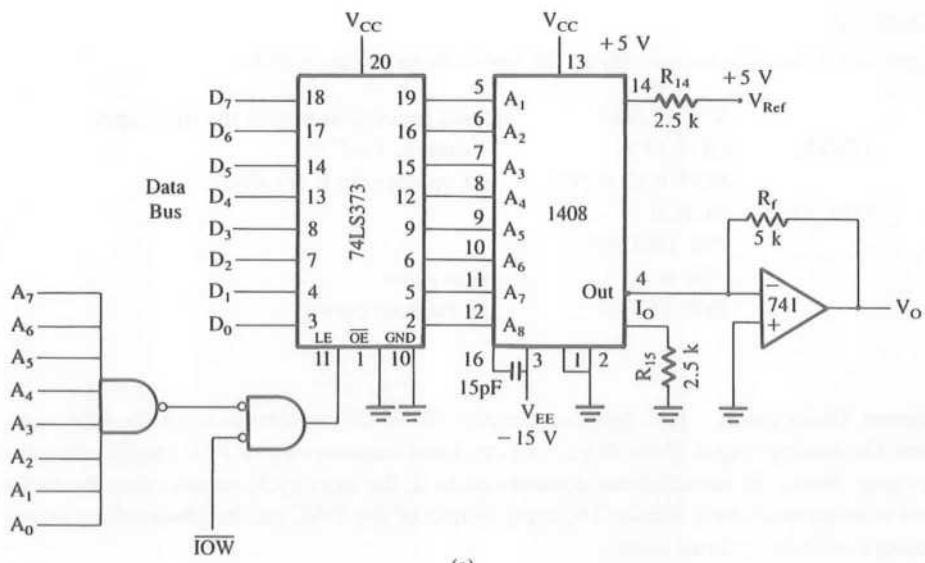
where inputs A_1 through $A_8 = 0$ or 1 .

This formula is an application of the generalized formula for the current I_O . For full-scale input (D_7 through $D_0 = 1$),

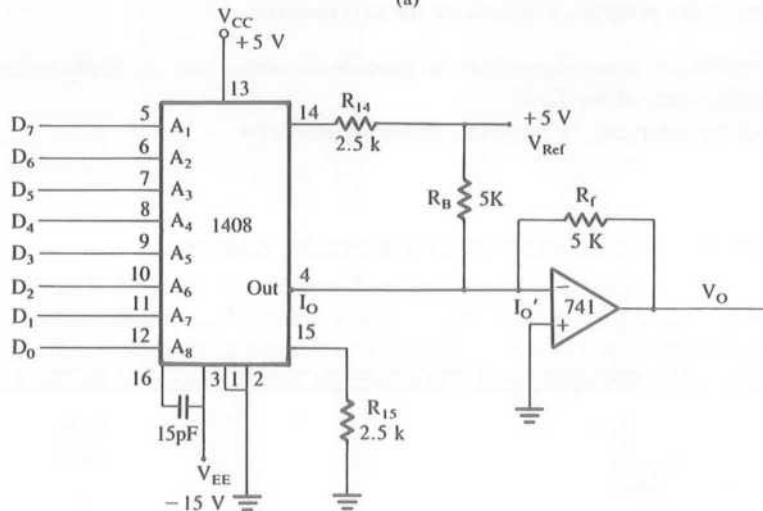
$$\begin{aligned} I_O &= \frac{5 \text{ V}}{2.5 \text{ k}} \left(\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \frac{1}{64} + \frac{1}{128} + \frac{1}{256} \right) \\ &= 2 \text{ mA } (255/256) \\ &= 1.992 \text{ mA} \end{aligned}$$

The output is 1 LSB less than the full-scale reference source of 2 mA. The output voltage V_O for the full-scale input is

$$\begin{aligned} V_O &= 2 \text{ mA } (255/256) \times 5 \text{ k} \\ &= 9.961 \text{ V} \end{aligned}$$



(a)



(b)

FIGURE 13.5

Interfacing the 1408 D/A Converter: Voltage Output in Unipolar Range (a) and in Bipolar Range (b)



ATILIM
UNIVERSITY
LIBRARY

PROGRAM

To generate a continuous waveform, the instructions are as follows:

	MVI A,00H	;Load accumulator with the first input
DTOA:	OUT FFH	;Output to DAC
	MVI B,COUNT	;Set up register B for delay
DELAY:	DCR B	
	JNZ DELAY	
	INR A	;Next input
	JMP DTOA	;Go back to output

Program Description This program outputs 00 to FF continuously to the D/A converter. The analog output of the DAC starts at 0 and increases up to 10 V (approximately) as a ramp. When the accumulator contents go to 0, the next cycle begins; thus the ramp signal is generated continuously. The ramp output of the DAC can be observed on an oscilloscope with an external sync.

The delay in the program is necessary for two reasons:

1. The time needed for a microprocessor to execute an output loop is likely to be less than the settling time of the DAC.
2. The slope of the ramp can be varied by changing the delay.

OPERATING THE D/A CONVERTER IN A BIPOLAR RANGE

The 1408 in Figure 13.5(b) is calibrated for the bipolar range from -5 V to $+5\text{ V}$ by adding the resistor R_B (5.0 k) between the reference voltage V_{Ref} and the output pin 4. The resistor R_B supplies 1 mA (V_{Ref}/R_B) current to the output in the opposite direction of the current generated by the input signal. Therefore, the output current for the bipolar operation I_O' is

$$\begin{aligned} I_O' &= I_O - \frac{V_{Ref}}{R_B} \\ &= \frac{V_{Ref}}{R_{14}} \left(\frac{A_1}{2} + \frac{A_2}{4} + \frac{A_3}{8} + \frac{A_4}{16} + \frac{A_5}{32} + \frac{A_6}{64} + \frac{A_7}{128} + \frac{A_8}{256} \right) - \frac{V_{Ref}}{R_B} \end{aligned}$$

When the input signal is equal to zero, the output V_O is

$$\begin{aligned} V_O &= I_O' R_f \\ &= \left(I_O - \frac{V_{Ref}}{R_B} \right) R_f \\ &= \left(0 - \frac{5\text{ V}}{5\text{ k}} \right) (5\text{ k}) \quad (I_O = 0 \text{ for input} = 0) \\ &= -5\text{ V} \end{aligned}$$

When the input = 1000 0000, output V_O is

$$\begin{aligned}
 V_O &= \left(I_O - \frac{V_{Ref}}{R_B} \right) R_f \\
 &= \left(\frac{V_{Ref}}{R_{14}} \times \frac{A_1}{2} - \frac{V_{Ref}}{R_B} \right) R_f \quad (A_2 - A_8 = 0) \\
 &= \left(\frac{5 \text{ V}}{2.5 \text{ k}} \times \frac{1}{2} - \frac{5 \text{ V}}{5 \text{ k}} \right) 5 \text{ k} \\
 &= (1 \text{ mA} - 1 \text{ mA})5 \text{ k} = 0 \text{ V}
 \end{aligned}$$

13.1.4 Microprocessor-Compatible D/A Converters

In response to the growing need for interfacing data converters with the microprocessor, specially designed microprocessor-compatible D/A converters are now available. These D/A converters generally include a latch on the chip (see Figure 13.6), thus eliminating the need for an external latch as in Figure 13.5.

Figure 13.6 shows the block diagram of Analog Devices AD558, which includes a latch and an output op amp internal to the chip. It can be operated with one power supply voltage between +4.5 V to +16.5 V. To interface the AD558 with the microprocessor, two signals are required: Chip Select (CS) and Chip Enable (CE).

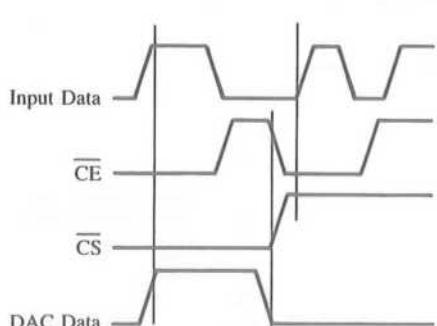
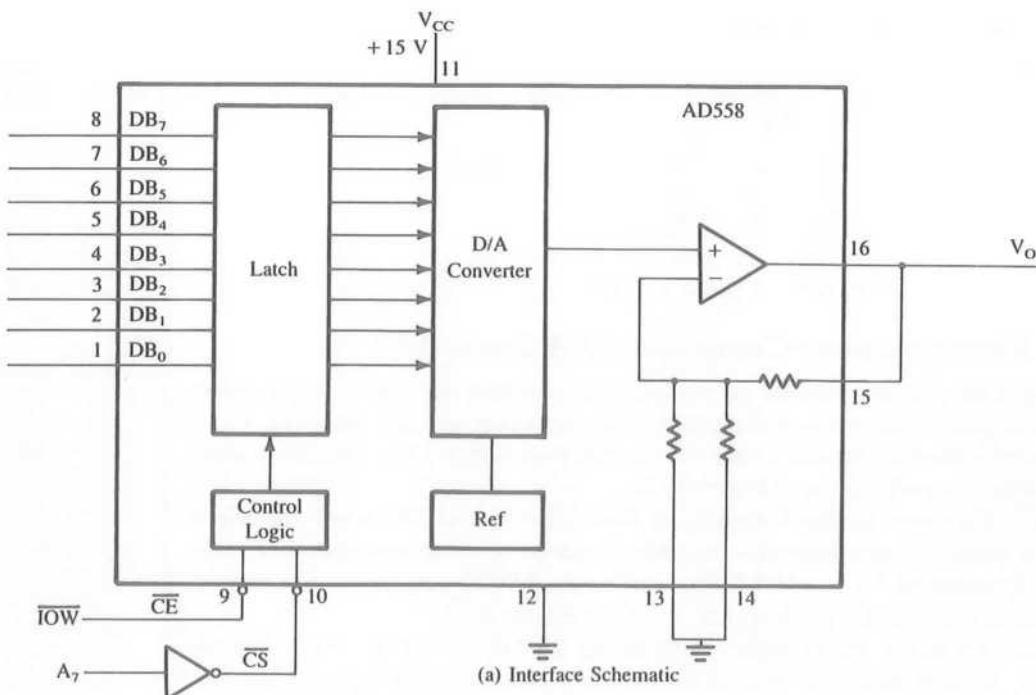
Figure 13.6 shows one example of interfacing the AD558 with the 8085. The address line A_7 through an inverter is used for the Chip Select, which assigns the port address 80H (assuming all other address lines are at logic 0) to the DAC port. The control signal IOW is used for the Chip Enable. The program shown in Section 13.1.3 can be used to generate ramp waveforms.

Figure 13.6(b) shows the timing of latching data in relation to the control signals; Figure 13.6(c) shows the truth table of the control logic. When both signals CS and CE are at logic 0, the latch is transparent, meaning the input is transferred to the DAC section. When either CS or CE goes to logic 1, the input is latched in the register and held until both control signals go to logic 0.

13.1.5 Interfacing a 10-Bit D/A Converter

In many D/A converter applications, 10- or 12-bit resolution is required. However, the 8-bit microprocessor has only eight data lines. Therefore, to transfer ten bits, the data bus is time-shared by using two output ports: one for the first eight bits and the second for the remaining two bits. A disadvantage of this method is that the DAC output assumes an intermediate value between the two output operations. This difficulty can be circumvented by using a double-buffered DAC such as Analog Devices AD7522, as shown in Figure 13.7(a).

The AD7522 is a CMOS 10-bit D/A converter with an input buffer and a holding register. The ten bits are loaded into the input register in two steps using two output ports. The low-order eight bits are loaded with the control line LBS, and the remaining two bits are loaded with the control line HBS. Then all ten bits are switched into a holding register for the conversion by enabling the line LDAC. The last operation (enabling the line LDAC) can be combined with the loading of the second byte as shown in Figure 13.7(a); otherwise, this operation will require three output ports.



Input Data	\overline{CE}	\overline{CS}	DAC Data	Latch Condition
0	0	0	0	"transparent"
1	0	0	1	"transparent"
0	$\frac{1}{2}$	0	0	latching
1	$\frac{1}{2}$	0	1	latching
0	0	$\frac{1}{2}$	0	latching
1	0	$\frac{1}{2}$	1	latching
X	1	X	previous data	latched
X	X	1	previous data	latched

Notes:
X = Does not matter
 $\frac{1}{2}$ = Logic Threshold at Positive-Going Transition

(c) Truth Table

FIGURE 13.6

Interfacing the AD558 (Microprocessor-Compatible D/A Converter) with the 8085.

SOURCE: (b) and (c): Analog Devices, Inc., *Data Converter Reference Manual* (Norwood, Mass.: Author, 1992), pp. 2, 52.

HARDWARE DESCRIPTION

Figure 13.7(a) shows a schematic of interfacing the AD7522 with the 8085. It is a memory-mapped I/O with multiple addresses. The attempt here is to minimize the chip count.

The three input signals to the decoder are address line A_0 and two control signals $\overline{IO/M}$ and WR . To enable the line LBS, the input to the decoder should be 000; this re-

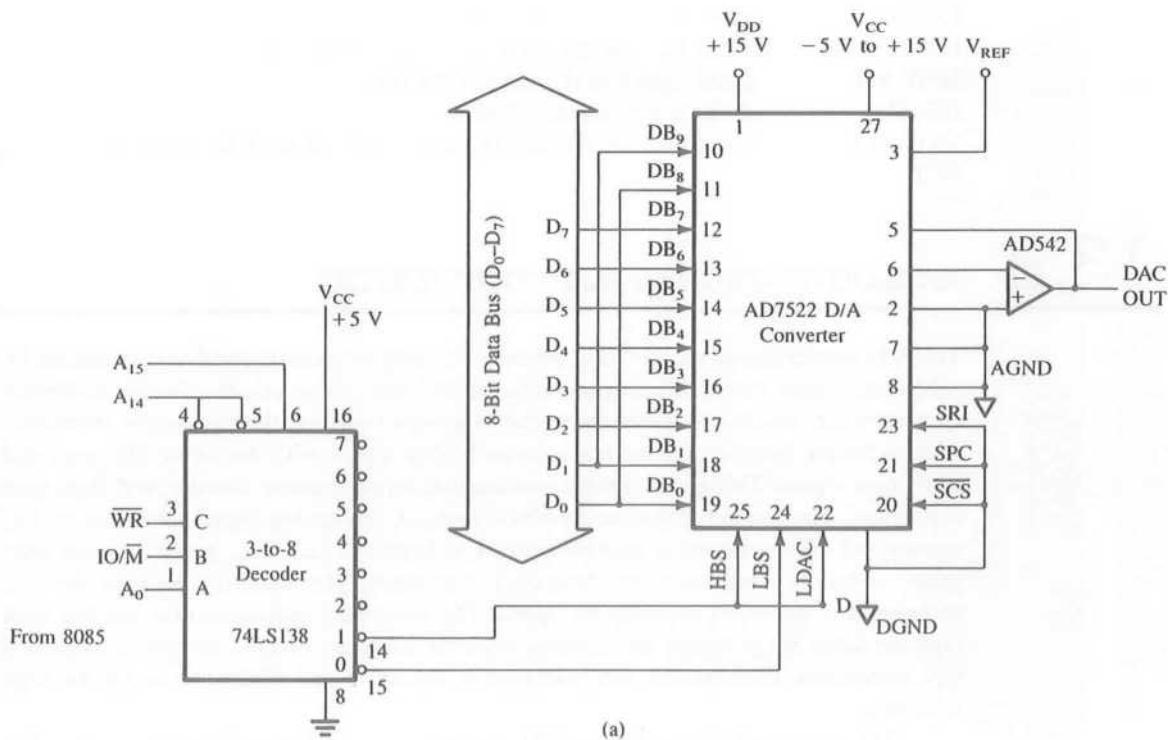
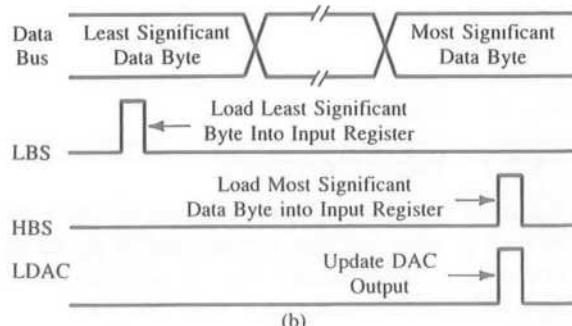


FIGURE 13.7
Interfacing 10-Bit DAC (AD7522) with the 8085 (a) and a Timing Diagram for Loading the Input Data (b)

SOURCE: Analog Devices, Inc., *Data Acquisition Components and Subsystems* (Norwood, Mass.: Author, 1980), pp. 9-73.



sults in the port address 8000H. ($A_{15} = 1$, $A_{14} = 0$, and the lines A_{13} to A_1 are assumed at logic 0.) When a data byte is sent to the port address 8000H in a memory-mapped I/O, the WR and IO/M signals go low along with A_0 , and the line LBS is enabled. Similarly, the address 8001H enables the lines HBS and LDAC. Figure 13.7(b) shows the timing diagram for loading input data into the converters.

The following instructions illustrate how to load the maximum input of ten bits (all 1s) into the D/A converter.

LXI B,03FFH	;Load ten bits at logic 1 in BC register
LXI H,8000H	;Load HL with port address for low-order 8-bits
MOV M,C	;Load eight bits (D_7 – D_0) in the DAC
INX H	;Point to port address 8001H
MOV M,B	;Load two bits (D_9 and D_8) and switch all ten bits for conversion
HLT	

13.2

ANALOG-TO-DIGITAL (A/D) CONVERTERS

The A/D conversion is a quantizing process whereby an analog signal is represented by equivalent binary states; this is opposite to the D/A conversion process. Analog-to-Digital converters can be classified into two general groups based on the conversion technique. One technique involves comparing a given analog signal with the internally generated equivalent signal. This group includes successive-approximation, counter, and flash-type converters. The second technique involves changing an analog signal into time or frequency and comparing these new parameters to known values. This group includes integrator converters and voltage-to-frequency converters. The trade-off between the two techniques is based on accuracy vs. speed. The successive-approximation and the flash type are faster but generally less accurate than the integrator and the voltage-to-frequency type converters. Furthermore, the flash type is expensive and difficult to design for high accuracy.

The successive-approximation A/D converters are used in applications such as data loggers and instrumentation, where conversion speed is important. On the other hand, integrating-type converters are used in applications such as digital meters, panel meters, and monitoring systems, where the conversion accuracy is critical. The most commonly used A/D converters—the successive-approximation—is discussed in this section with several interfacing examples.

13.2.1 Basic Concepts

Figure 13.8(a) shows a block diagram of a 3-bit A/D converter. It has one input line for an analog signal and three output lines for digital signals. Figure 13.8(b) shows the graph of the analog input voltage (0 to 1 V) and the corresponding digital output signal. It shows eight (2^3) discrete output states from 000_2 to 111_2 , each step being $1/8$ V apart. This is defined as the resolution of the converter. The LSB, the MSB, and the full-scale output are calculated the same way as in D/A converters.

In A/D conversion, another critical parameter is **conversion time**. This is defined as the total time required to convert an analog signal into its digital output and is determined by the conversion technique used and by the propagation delay in various circuits.

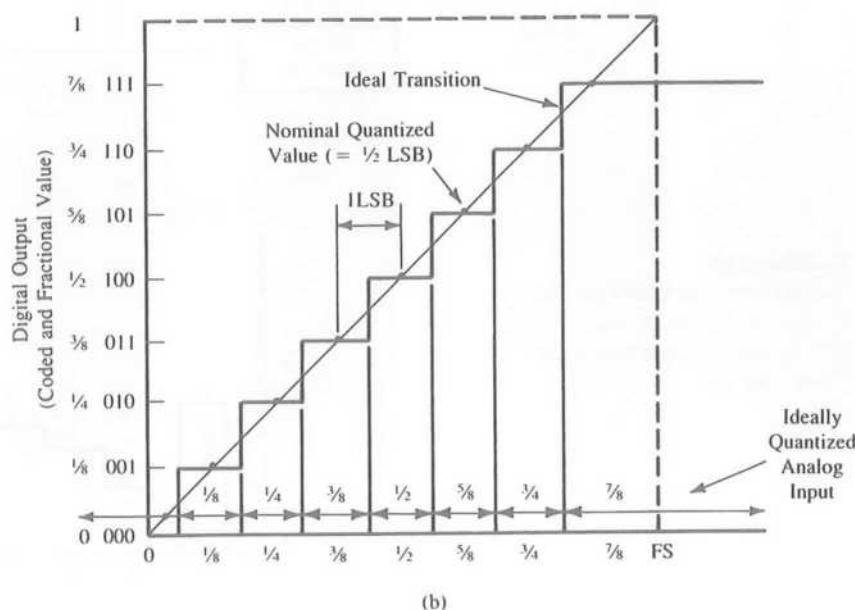
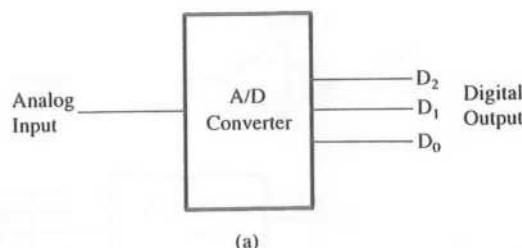
13.2.2 Successive-Approximation A/D Converter

Figure 13.9(a) shows the block diagram of a successive approximation A/D converter. It includes three major elements: the D/A converter, the successive approximation register

FIGURE 13.8

A 3-Bit A/D Converter: Block Diagram (a) and Analog Input vs. Digital Output (b)

SOURCE: Analog Devices, Inc., *Integrated Circuit Converters, Data Acquisition Systems, and Analog Signal Conditioning Components* (Norwood, Mass.: Author, 1979), pp. 1-18.



(SAR), and the comparator. The conversion technique involves comparing the output of the D/A converter V_O with the analog input signal V_{in} . The digital input to the DAC is generated using the successive-approximation method (explained below). When the DAC output matches the analog signal, the input to the DAC is the equivalent digital signal.

The successive-approximation method of generating input to the DAC is similar to weighing an unknown material (e.g., less than 1 gram) on a chemical balance with a set of such fractional weights as $\frac{1}{2}$ g, $\frac{1}{4}$ g, $\frac{1}{8}$ g, etc. The weighing procedure begins with the heaviest weight ($\frac{1}{2}$ g), and subsequent weights (in decreasing order) are added until the balance is tipped. The weight that tips the balance is removed, and the process is continued until the smallest weight is used. In the case of a 4-bit A/D converter, bit D₃ is turned on first and the output of the DAC is compared with an analog signal. If the comparator changes the state, indicating that the output generated by D₃ is larger than the analog signal, bit D₃ is turned off in the SAR and bit D₂ is turned on. The process continues until the input reaches bit D₀. Figure 13.9(b) illustrates a 4-bit conversion process. When bit D₃

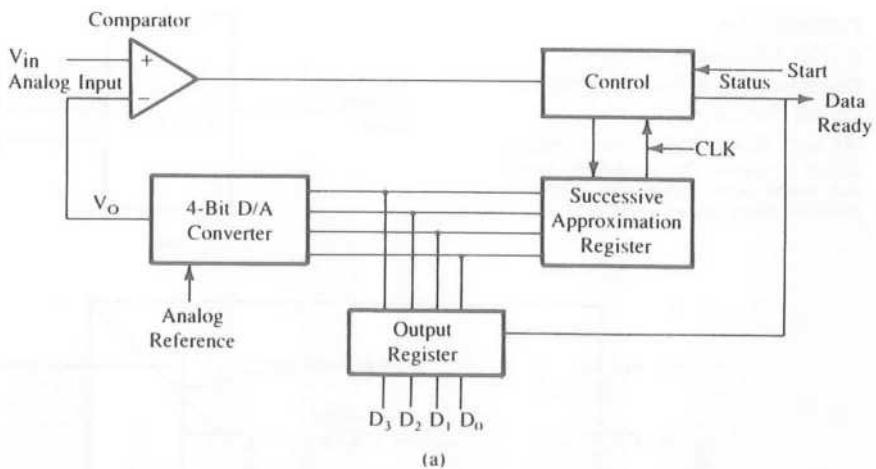
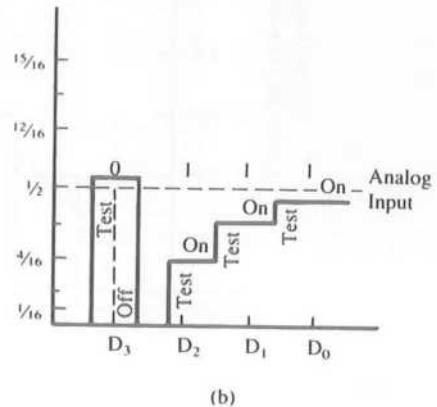


FIGURE 13.9
Successive-Approximation A/D
Converter: Block Diagram (a) and
Conversion Process for a 4-Bit
Converter (b)



is turned on, the output exceeds the analog signal and, therefore, bit D_3 is turned off. When the next three successive bits are turned on, the output becomes approximately equal to the analog signal.

The successive-approximation conversion process can be accomplished through either the software or hardware approach. In the software approach, an A/D converter is designed using a D/A converter, and the microprocessor plays the role of the counter and the SAR. For the hardware approach, a complete ADC, including a tri-state buffer, is now available as an integrated circuit on a chip. The interfacing of both types of A/D converters is illustrated in the next section.

13.2.3 Interfacing 8-Bit A/D Converters

As an integrated circuit, the A/D converter includes all three elements—SAR, DAC, and comparator—on a chip (Figure 13.10). In addition, it has a tri-state output buffer.

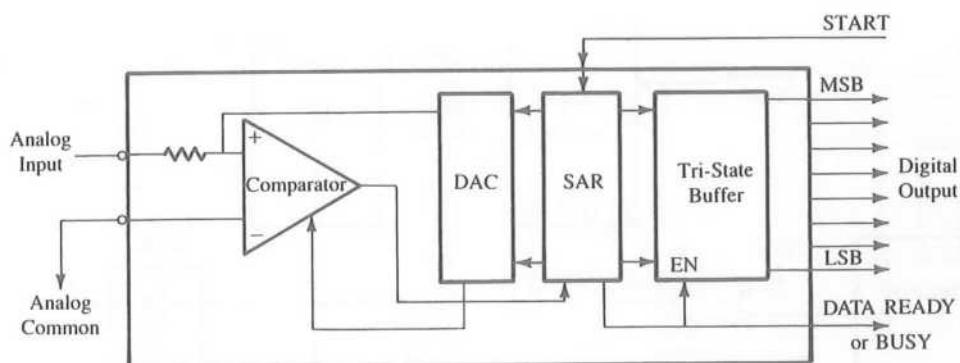


FIGURE 13.10
Block Diagram of a Typical Successive-Approximation A/D Converter as an Integrated Circuit

Typically, it has two control lines, START (or CONVERT) and DATA READY (or BUSY); they are TTL-compatible and can be active low or high depending upon the design.

A pulse to the START pin begins the conversion process and disables the tri-state output buffer. At the end of the conversion period, DATA READY becomes active and the digital output is made available at the output buffer. To interface an A/D converter with the microprocessor, the microprocessor should

1. send a pulse to the START pin. This can be derived from a control signal such as Write (WR).
2. wait until the end of the conversion. The end of the conversion period can be verified either by status checking (polling) or by using the interrupt.
3. read the digital signal at an input port.

Two examples of interfacing A/D converters are discussed in the following sections. The first example demonstrates the interfacing of a typical A/D converter using the status check approach, and the second example demonstrates the interfacing of the National Semiconductor ADC0801 using the interrupt. An example of using handshake signals is given in Chapter 14.

INTERFACING AN 8-BIT A/D CONVERTER USING STATUS CHECK

Figure 13.11 shows a schematic of interfacing a typical A/D converter using status check. The A/D converter has one input line for analog signal and eight output lines for converted digital signals. Typically, the analog signal can range from 0 to 10 V, or ± 5 V. In addition, the converter shows two lines START and DR (Data Ready), both active low (the active logic level of these lines can be either low or high depending upon the design of a particular converter). When an active low pulse is sent to the START pin, the DA

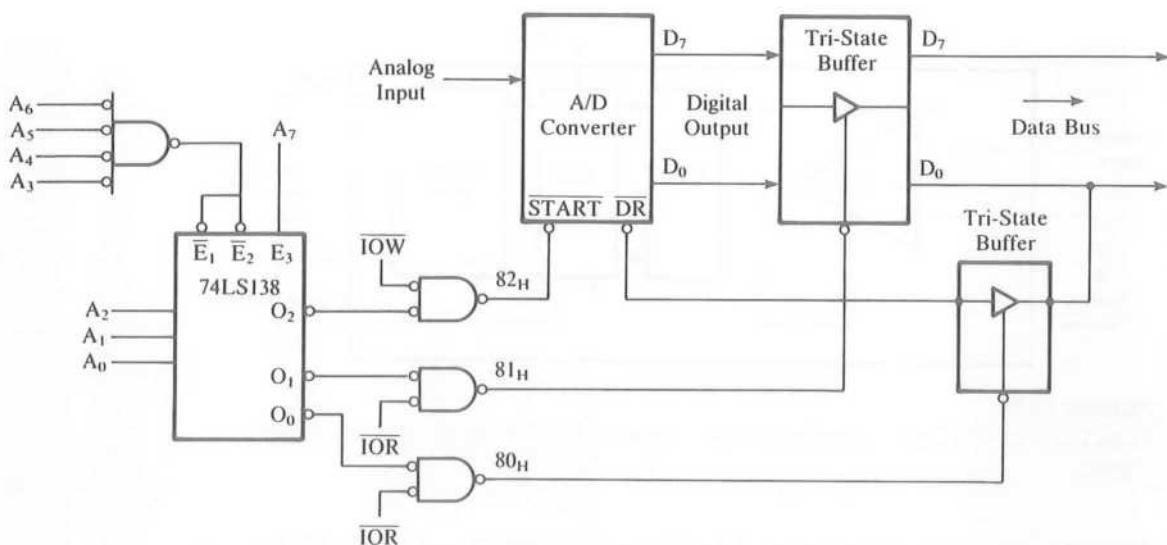


FIGURE 13.11
Interfacing an A/D Converter Using the Status Check

goes high and the output lines go into the high impedance state. The rising edge of the START pulse initiates the conversion. When the conversion is complete, the DR goes low, and the data are made available on the output lines that can be read by the microprocessor. To interface this converter, we need one output port to send a START pulse and two input ports: one to check the status of the DR line and the other to read the output of the converter.

In Figure 13.11, the address decoding is performed by using the 3-to-8 decoder (74LS138), the 4-input NAND gate, and inverters. Three output lines of the decoder are combined with appropriate control signals (\overline{IOW} and \overline{IOR}) to assign three port addresses from 80H to 82H. The output port 82H is used to send a START pulse by writing the OUT instruction; in this case, we are interested in getting a pulse from the microprocessor, and the contents of the accumulator are irrelevant to start the conversion. However, for some converters the \overline{IOW} pulse from the microprocessor may not be long enough to start the conversion. When the conversion begins, the DR (Data Ready) goes high and stays high until the conversion is completed. The status of the DR line is monitored by connecting the line to bit D₀ of the data bus through a tri-state buffer with the input port address 80H. The processor will continue to read port 80H until bit D₀ goes low. When the DR goes active, the data are available on the output lines of the converter, and the processor can access that data by reading the input port 81H. The subroutine instructions, to initiate the conversion and to read output data, and the flowchart are shown in Figure 13.12.

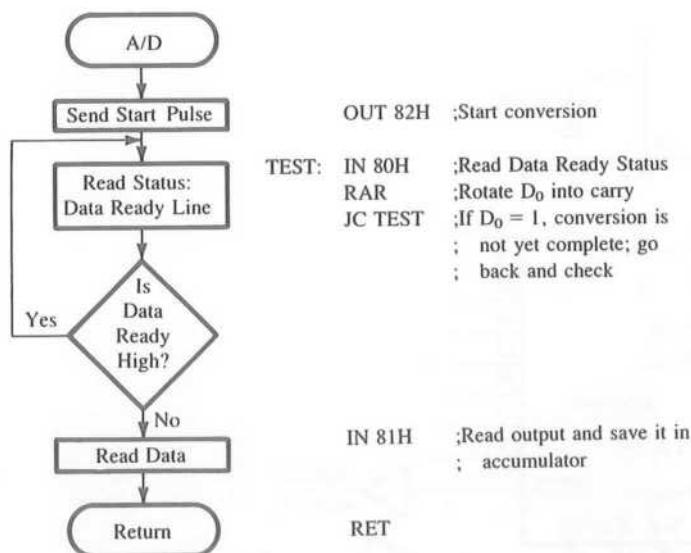


FIGURE 13.12
Flowchart of A/D Conversion Process

13.2.4 Illustration: Interfacing an 8-Bit A/D Converter Using the Interrupt

PROBLEM STATEMENT

1. Interface the National Semiconductor ADC0801 converter with the 8085 MPU using memory-mapped I/O and the interrupt RST 6.5.
2. Write an interrupt routine to read the output data of the converter, store it in memory, and continue to collect data for the specified number of times.

HARDWARE DESCRIPTION

The ADC0801 is a CMOS 8-bit successive-approximation A/D converter housed in a 20-pin DIP package. The input voltage can range from 0 to 5 V and operates with a single power supply of +5 V. It has two inputs $V_{IN}(+)$ and $V_{IN}(-)$ for the differential analog signal. When the analog signal is single-ended positive, $V_{IN}(+)$ is used as the input and the $V_{IN}(-)$ pin is grounded; when the signal is single-ended negative, $V_{IN}(-)$ is used as the input and the $V_{IN}(+)$ pin is grounded. The converter requires a clock at CLK IN; the frequency range can be from 100 kHz to 800 kHz. The user has two options: either to connect an external clock at CLK IN or to use the built-in internal clock by connecting a resistor and a capacitor externally at pins 19 and 4, respectively, as shown in Figure 13.13. The frequency is calculated by using the formula $f = 1/(1.1(RC))$. Typically, the clock frequency is designed for 640 kHz to provide 100 μ s of conversion time.

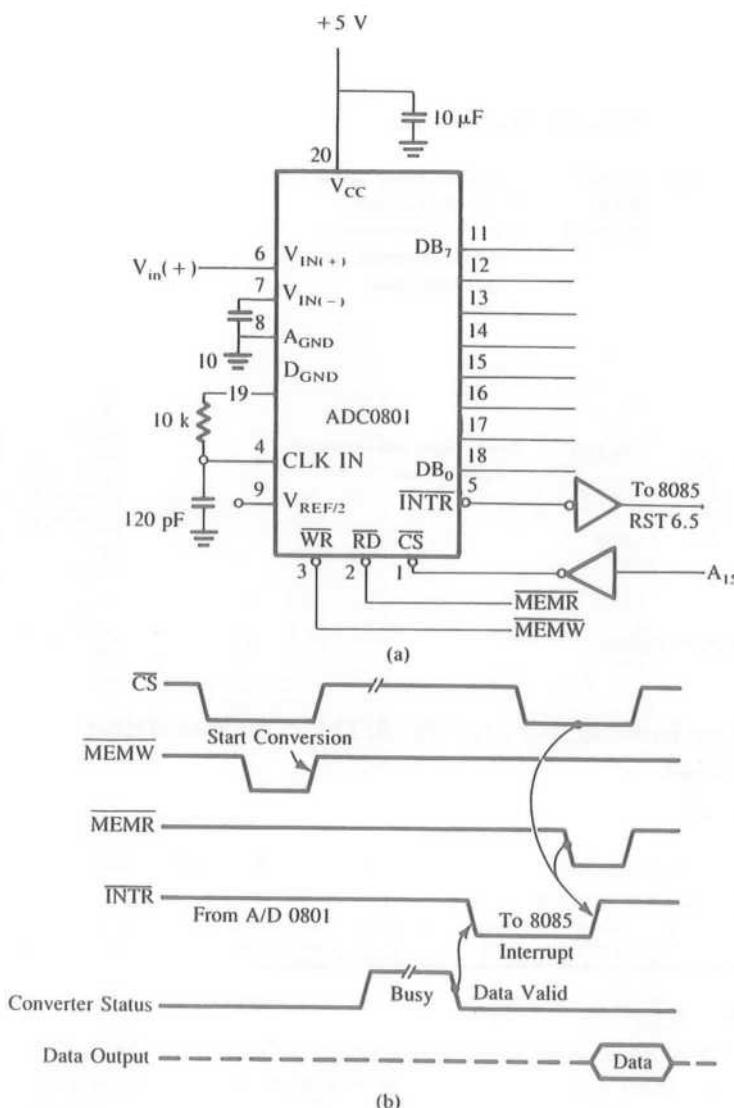


FIGURE 13.13

The ADC 0801 Using the Interrupt: Interface (a) and Timing Diagram for Reading Data from A/D Converter (b)

SOURCE: (b): National Semiconductor, *Data Conversion/Acquisition Data Book* (Santa Clara, Calif.: Author, 1980), pp. 5-25.

The ADC0801 is designed to be microprocessor-compatible. In our previous example of Figure 13.11, we needed external ports to access data and monitor the Data Ready signal; however, in this converter, the necessary control logic is built inside the chip. It has three control signals—CS, WR, and RD—that are used for interfacing. To start conversion, the CS and WR signals are asserted low. When WR goes low, the internal SAR

(Successive-Approximation Register) is reset, and the output lines go into the high impedance state. When WR makes the transition from low to high, the conversion begins. When the conversion is completed, the INTR is asserted low and the data are placed on the output lines. The INTR signal can be used to interrupt the processor. When the processor reads the data by asserting RD, the INTR is reset.

When V_{CC} is +5 V, the input voltage can range from 0 V to 5 V and the corresponding output will be from 00 to FFH. However, the full-scale output can be restricted to the lower range of inputs by using pin 9 ($V_{REF/2}$). For example, if we connect a 0.5 V dc source at pin 9, we can obtain full-scale output FFH for a 1 V input signal (this is twice the voltage of pin 9).

The ADC0801 can be operated in a continuous mode by connecting WR to INTR and grounding CS and RD. However, our focus here is to use the interrupt RST 6.5 to collect data from the converter.

INTERFACING CIRCUIT

Figure 13.13(a) shows the interfacing of the ADC0801 with the 8085 MPU, using the interrupt. Address line A_{15} with an inverter is used for Chip Select (CS), and the control signals MEMR and MEMW are connected to RD and WR signals, respectively. This is a memory-mapped port with the address 8000H (assuming all don't care lines at logic 0).

The conversion is initiated when the CS and WR signals go low; see Figure 13.13(b). At the end of the conversion, the INTR signal goes low and is used to interrupt the MPU through an inverter. When the service routine reads the data byte, the RD signal causes the INTR to go high, as shown in the timing diagram, Figure 13.13(b). This chip includes the control logic to set INTR at the end of a conversion and to reset it when data are read; by including this logic on the converter chip, extra components necessary for interfacing are eliminated.

To implement data transfer using the interrupt, as shown in Figure 13.13(a), the main program should initialize the stack, enable the microprocessor interrupts (EI), unmask the RST 6.5 (SIM), and initiate a conversion by writing to port 8000H. In addition, the main program should include the initialization of the memory pointer for storing data and the counter to count the readings. At the end of the conversion, the microprocessor is interrupted by using RST 6.5, which transfers the program control to location 0034H and then to the service routine.

SERVICE ROUTINE

LDA 8000H	;Read data
MOV M,A	;Store data in memory
INX H	;Next memory location
DCR B	;Next count
STA 8000H	;Start next conversion
EI	;Enable interrupt again
RNZ	;Go back to main if counter ≠ 0
HLT	;End

The service routine reads the output data by using the instruction LDA, stores the byte in memory, and updates the memory pointer and the counter. The routine assumes

that the information concerning the memory pointer (HL) and the counter (B) is supplied by the main program. The memory pointer specifies the location where the data should be stored, and the counter specifies the number of bytes to be collected. The STA instruction starts the next conversion by asserting the MEMW signal; this instruction should not be interpreted to mean that it is storing the contents of the accumulator in the converter. Then the service routine sets the interrupt flip-flop for subsequent interrupts and returns to the main program if the counter is not zero. When the counter goes to zero, the program completes the data collection.

SUMMARY

- D/A converters transform a digital signal into an equivalent analog signal, and A/D converters transform an analog signal into an equivalent digital signal.
- Resolution of a converter determines the degree of accuracy in conversion. It is equal to $\frac{1}{2^n}$ (n is the number of bits).
- Settling time (for DAC) and conversion time (for ADC) are important parameters in selecting data converters; they suggest the speed of data conversion.
- D/A converters are classified into three categories according to their output function: current, voltage, and multiplying.
- The successive-approximation A/D converter is a high-speed converter and uses a D/A converter to compare an analog signal with an internally generated signal.
- A successive-approximation A/D converter is available as an integrated circuit on a chip or can be designed by using software and a D/A converter.
- The integrating-type A/D converter is a high-accuracy converter. It is based on the principle of converting an analog signal into a time period and measuring the time period with a digital counter.
- A/D converters can serve as input devices and D/A converters can serve as output devices to microprocessor-based systems.
- A/D converters can be interfaced with the microprocessor by using techniques such as status check and interrupt.
- When 12-bit (or 16-bit) converters are interfaced with an 8-bit microprocessor, data are transferred in two (or three) stages using multiple ports.

QUESTIONS, PROBLEMS, AND PROGRAMMING ASSIGNMENTS

1. Calculate the output current for the 1408, Figure 13.5(a), if the input is 82H and the converter is calibrated for a 0 to 2 mA current range.
2. What is the voltage output in Problem 1?
3. Figure 13.5(b) shows a circuit that is calibrated for -5 V to +5 V. Calculate the output voltage if the input is 45H.

4. What changes are necessary in the ramp program, Figure 13.5(a), to limit the peak voltage to 7.5 V?
5. Modify the program of Figure 13.5(a) to generate a square wave with the amplitude of 5 V and a 1 kHz frequency.
6. Calculate the resolution of a 12-bit D/A converter.
7. A 12-bit D/A converter is calibrated over the range 0 to 10 V. Calculate the outputs if the input is 01H and 82H.
8. Calculate the analog voltages corresponding to the LSB and the MSB for a 12-bit A/D converter calibrated for a 0 to 5 V range.
9. In Figure 13.11, assume that the data line D_7 is connected to the line (\overline{DR}) , instead of D_0 , to check the status, and make the necessary changes in the instructions.
10. Change the circuit in Figure 13.11 from the peripheral I/O to the memory-mapped I/O, and assign the port addresses as 8000H to 8002H. Assume that the unused address lines are at logic 0. Rewrite the instructions.
11. Modify the circuit in Figure 13.11 to interface a 12-bit A/D converter.
12. Write a main program for the service routine in Section 13.2.4.

EXPERIMENTAL ASSIGNMENTS

1.
 - a. Connect the circuit as shown in Figure 13.5(a) and calibrate the D/A converter for 0 to 10 V.
 - b. Write a DELAY routine for a 100 μ s delay, and enter the program.
 - c. Measure the frequency and the slope of the ramp on an oscilloscope.
 - d. Change the DELAY routine to 500 μ s. Measure the frequency and the slope of the ramp.
 - e. Modify the program to limit the maximum peak voltage of the ramp to 5 V.
 - f. Modify the program to generate a triangular waveform.
 - g. Modify the program as suggested below:
 - (1) Store some random data at locations XX50H to XX5FH.
 - (2) Change instructions to call out the data in sequence. Display on an oscilloscope.
 - (3) Continue displaying the data to observe a stable pattern.
 - (4) Eliminate the DELAY routine and observe the output.
2.
 - a. Modify the interfacing circuit in Figure 13.13 to change memory-mapped I/O to peripheral I/O and assign the port address F8H to the A/D converter by using a 3-to-8 decoder and a NAND gate.
 - b. Calculate the clock frequency and measure at CLK IN.
 - c. Record digital output readings for five different V_{IN} values from 0 V to +5 V, and store them in memory. Rewrite the interrupt routine to record one reading at a time.
 - d. Repeat step c by connecting +1 V dc at pin 9.

14

Programmable Interface Devices: - 8155 I/O and Timer - 8279 Keyboard/Display Interface

A programmable interface device is designed to perform various input/output functions. Such a device can be set up to perform specific functions by writing an instruction (or instructions) in its internal register, called the control register. Furthermore, functions can be changed anytime during execution of the program by writing a new instruction in the control register. These devices are flexible, versatile, and economical; they are widely used in microprocessor-based products.

In Chapter 5, we used simple integrated circuits, such as latches and tri-state buffers for I/O functions. However, they are limited in their capabilities; each device can perform one function, and they are hard-wired.

In a programmable device, on the other hand, functions are determined through software instructions. A programmable interface device can be viewed as multiple I/O devices, but it also performs many other functions, such as time delays, counting, and interrupts. In fact, it consists of many devices on

a single chip, interconnected through a common bus. This is a hardware approach through software control to performing the I/O functions discussed earlier. This approach, a trade-off between hardware and software, should reduce programming.

This chapter describes two programmable devices: the 8155/8156 I/Os and timer, and the 8279 Keyboard/Display Interface. The 8155 is specifically designed to be compatible with the 8085 and the 8279 is a general-purpose device. The Intel family of support devices includes several other general-purpose devices that will be discussed in the next chapter.

This chapter first describes the basic concepts underlying these programmable devices. On the basis of these concepts the 8155 and the 8279 are discussed in the context of a single-board microcomputer. The 8155 is a multipurpose chip. Its memory section has been discussed already in Chapter 3; only the programmable I/Os and the timer are discussed in this chapter.

OBJECTIVES

- List the elements and characteristics of a typical programmable device.
- Explain the functions of handshake signals.
- Explain the block diagram of the 8155 I/O section and timer.
- Design an interfacing circuit for the 8155 I/O ports and the timer, and write initialization instructions.

- Set up the 8155 I/O ports in the handshake mode and write initialization instructions.
- Set up the 8155 timer to generate a pulse after a given time delay or a continuous waveform.
- Explain the block diagram of the 8279 Keyboard/Display Interface and its operation.
- Write instructions to initialize the 8279 in a given mode.

14.1

BASIC CONCEPTS IN PROGRAMMABLE DEVICES

In Chapter 5, we discussed the interfacing of simple input (switches) and output (LEDs) devices. In the illustrations, we assumed that the I/O devices were always ready for data transfer. In fact, the assumption may not be valid in many data transfer situations. The MPU needs to check whether a peripheral is ready before it reads from or writes into a device because the execution speed of the microprocessor is much faster than the response of a peripheral such as a printer. For example, when the MPU sends data bytes (characters) to a printer, the microprocessor can execute the instructions to transfer a byte in microseconds; on the other hand, the printer can take 10 to 25 ms to print a character. After transferring a character to the printer, the MPU should wait until the printer is ready for the next character; otherwise data will be lost. To prevent the loss of data or the MPU reading the same data more than once, signals are exchanged between the MPU and a peripheral prior to actual data transfer; these signals are called handshake signals. To provide such signals in the illustrations of Chapter 5, we need to build additional logic circuitry.

In Chapter 13, we interfaced the A/D converter (ADC0801) using the interrupt I/O; however, the interrupt signal was generated by the internal logic of the data converter. Many peripherals may not have that capability; such signals may have to be provided by the interfacing circuitry. In some applications, data flow is bidirectional (such as data transfer between two computers). In such a situation, the interfacing device should be capable of handling bidirectional data flow. On the basis of the above discussion and the illustrations of Chapters 5 and 13, we can summarize the requirements for a programmable interfacing device as follows:

The device should include

1. input and output registers (a group of latches to hold data).
2. tri-state buffers.
3. capability for bidirectional data flow.
4. handshake and interrupt signals.

5. control logic.
6. chip select logic.
7. interrupt control logic.

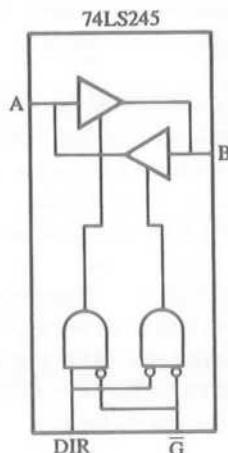
To understand the programmability of such a device, we will, in the next section, illustrate a simple example of building a programmable device using a transceiver (bidirectional buffer).

14.1.1 Making the 74LS245 Transceiver Programmable

The 74LS245 is a bidirectional tri-state octal buffer, and the direction of the data flow is determined by the signal DIR. Figure 14.1 shows the logic diagram of the 74LS245; it shows one buffer (rather than eight) in each direction. The buffer is enabled when \bar{G} is active low; however, the direction of the data flow is determined by the DIR signal. When the DIR is high, data flow from A to B, and when it is low, data flow from B to A. In fact, this is a hard-wired programmable device; the direction of the data flow is programmed through DIR. However, we are interested in a device that can be programmed by writing an instruction through the MPU. This can be accomplished by adding a register called the control register, as shown in Figure 14.2, and by connecting the DIR signal to bit D_0 of the control register. When $D_0 = 1$, data flow from A to B as output, and when $D_0 = 0$, data flow in the opposite direction as input.

Now the question is: How would the MPU write into the control register? The same way it would with any other I/O port—through a port address. Figure 14.2 shows that the address lines A_7-A_1 are used to select the chip through a NAND gate, and A_0 is used to differentiate between the control register and the transceiver. When A_0 is high, the control register is enabled; when A_0 is low, the transceiver is enabled. Thus, the MPU could access the control register through the port address FFH, and the transceiver through FEH. To set up the transceiver as an output device, the control word would be 01H, and to set it up as an input device the control word would be 00H.

FIGURE 14.1
Logic Symbol of 74LS245
Bidirectional Buffer



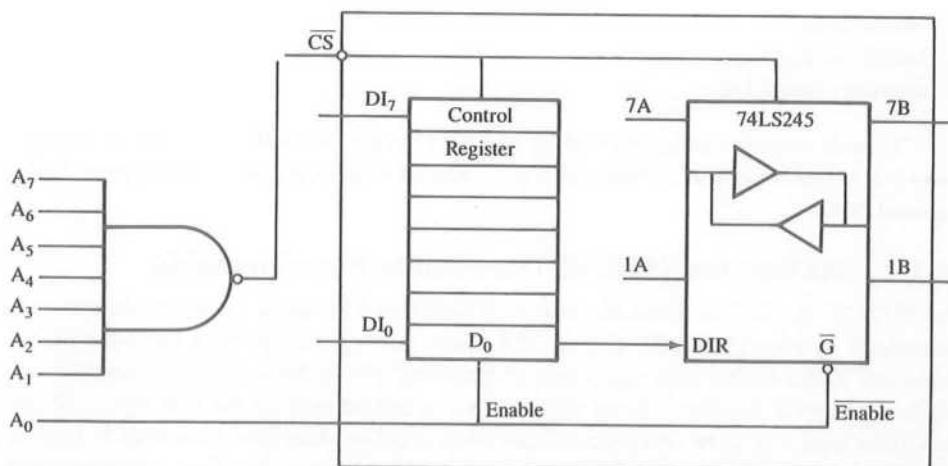


FIGURE 14.2
Making 74LS245 Programmable

**Example
14.1**

Write instructions to initialize the hypothetical chip (Figure 14.2) as an output buffer.

Solution

Instructions

MVI A,01H	;Set D ₀ = 1; D ₁ through D ₇ are don't care lines
OUT FFH	;Write in the control register
MVI A,BYTE1	;Load data byte
OUT FEH	;Send data out

In the last example, we used the 74LS245 as a bidirectional buffer. However, in microprocessor applications, we often need registers that can be used as I/O ports. We can build a bidirectional latch with an input and an output buffer, and by controlling the enable signals of the latch and buffers, we can program it to function as an input port or an output port. Figure 14.3 shows a data register (representing eight latches in each direction); **the I/O mode of this register is determined by bit D₀ in the control register. If bit D₀ is 0, it functions as an output port; if D₀ is 1, it functions as an input port; thus, by programming bit D₀, the device can function as an input port or an output port.** When the device is programmed as an output device, the MPU can write to the port by using the WR control signal that enables the tri-state buffer (not shown) and send out a byte. When bit D₀ = 1, the input latch is enabled, the output latch is disabled, and the MPU can read by using the RD signal.

14.1.2 Programmable Device with a Status Register

Figure 14.3 shows an additional input register called the status register. In the discussion of interfacing A/D converters in Chapter 13, we needed to build an external input port (Figure 13.11) to monitor the status of the Data Ready signal. In a programmable device, we can build such a register internally that can monitor the data lines of the data register as shown in Figure 14.3. Now we have three registers that can be accessed as ports. The decode logic for Chip Select is similar to that in Figure 14.2, with FFH as the port address for the control and the status registers and FEH as the port address for the data register. The control and the status registers are differentiated by the WR and RD control signals even if they have the same port address. The port addresses and the functions of these registers can be summarized as follows:

Control register (output only) = FFH (A_1 and $A_0 = 1$)

Status register (input only) = FFH (A_1 and $A_0 = 1$)

Data register (input or output) = FEH ($A_1 = 1$, $A_0 = 0$)

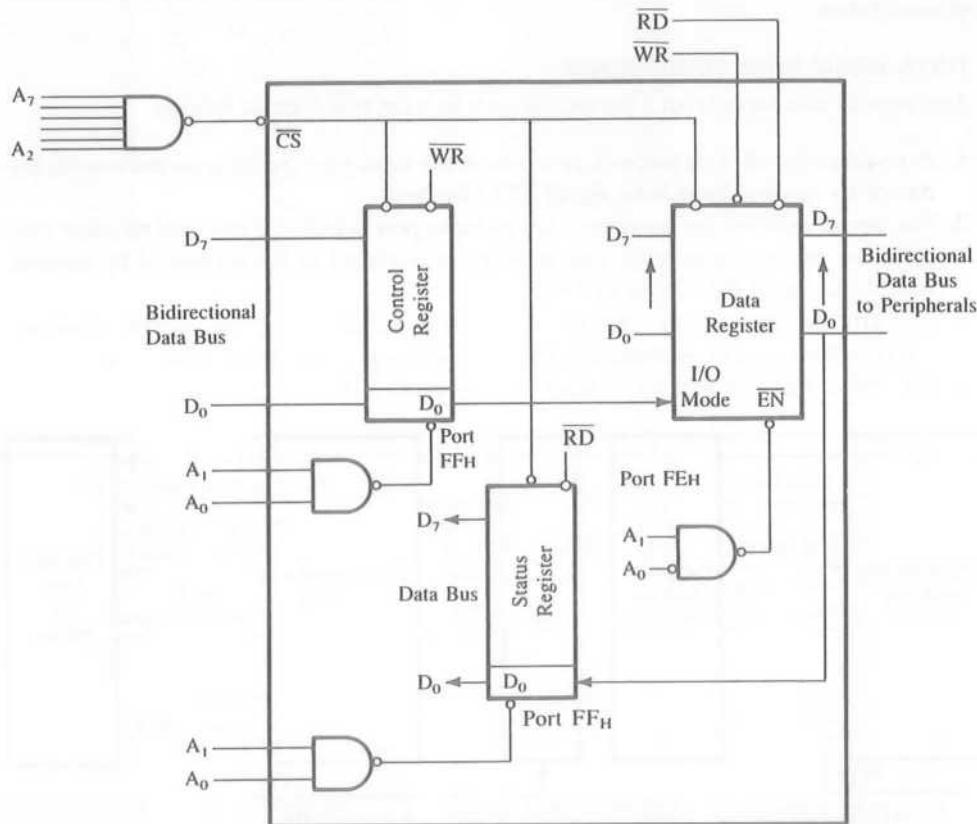


FIGURE 14.3

A Hypothetical Programmable Device with a Status Register

Thus we can build additional ports, and other bits in the control register can be used to define the functions of these registers. Similarly, we can add an interrupt logic and handshake signals.

14.1.3 Programmable Devices with Handshake Signals

The MPU and peripherals operate at different speeds; therefore, signals are exchanged prior to data transfer between the fast-responding MPU and slow-responding peripherals such as printers and data converters. These signals are called **handshake signals**. The exchange of handshake signals prevents the MPU from writing over the previous data before a peripheral has had a chance to accept it or from reading the same data before a peripheral has had time to send the next data byte. These signals are generally provided by programmable devices. Figure 14.4(a) shows a programmable device in the input mode, with two handshake signals (STB and IBF) and one interrupt signal (INTR). Now the MPU has two ways of finding out whether a peripheral is ready: either by checking the status of a handshake signal or through the interrupt technique, as explained below.

DATA INPUT WITH HANDSHAKE

The steps in data input from a peripheral such as a keyboard are as follows:

1. A peripheral strobes or places a data byte in the input port and informs the interfacing device by sending handshake signal STB (Strobe).
2. The device informs the peripheral that its input port is full—do not send the next byte until this one has been read. This message is conveyed to the peripheral by sending handshake signal IBF (Input Buffer Full).
3. The MPU keeps checking the status until a byte is available. Or the interfacing device informs the MPU, by sending an interrupt, that it has a byte to be read.
4. The MPU reads the byte by sending control signal RD.

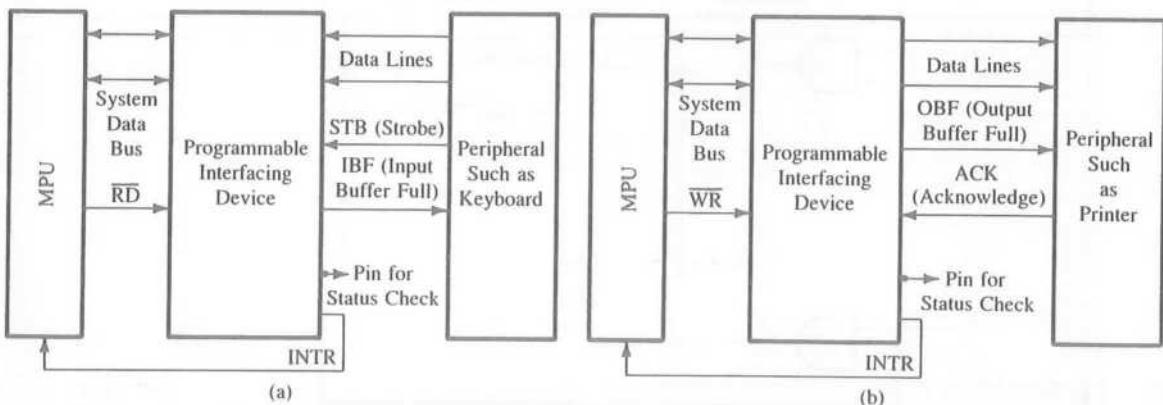


FIGURE 14.4

Interfacing Device with Handshake Signals for Data Input (a) and Data Output (b)

DATA OUTPUT WITH HANDSHAKE

Figure 14.4(b) shows the programmable device in the output mode using the same two handshake signals, except that they are labeled differently. The steps in data output to a peripheral such as a printer are as follows:

1. The MPU writes a byte into the output port of the programmable device by sending control signal WR.
2. The device informs the peripheral, by sending handshake signal OBF (Output Buffer Full), that a byte is on the way.
3. The peripheral acknowledges the byte by sending back the ACK (Acknowledge) signal to the device.
4. The device interrupts the MPU to ask for the next byte, or the MPU finds out that the byte has been acknowledged through the status check.

Examination of Figure 14.4 shows the following similarities among the handshake signals:

1. Handshake signals ACK and STB are input signals to the device and perform similar functions, although they are called by different names.
2. Handshake signals OBF and IBF are output signals from the device and perform similar functions (Buffer Full).

The active levels and labels of these signals are arbitrary and vary considerably from device to device. For example, in the 8155 (R/W memory with I/O) both input signals are called STB (Strobe); in the 8255 (Programmable Peripheral Interface) one input signal is called STB (Strobe) and the other is called ACK (Acknowledge).

14.1.4 Review of Important Concepts

The above examples suggest that a programmable I/O device is likely to have the following elements:

1. A control register in which the MPU can write an instruction
2. A status register that can be read by the MPU
3. I/O devices or registers
4. Control logic
5. Chip Select logic
6. Bidirectional data bus
7. Handshake signals and Interrupt logic

A programmable I/O device is programmed by writing a specific word, called the **control word**, according to the internal logic; its status can be verified by reading the **status register**. This I/O device can be expanded to include elements such as multiple I/O ports, counters, and parallel-to-serial registers. Two programmable devices commonly used in 8085 single-board systems—the 8155 and the 8279—are described in detail in the next sections.

14.2

THE 8155: MULTIPURPOSE PROGRAMMABLE DEVICE

The 8155 is a multipurpose programmable device specifically designed to be compatible with the 8085 microprocessor. The ALE, IO/M, RD, and WR signals from the 8085 can be connected directly to the device; this eliminates the need for external demultiplexing of the low-order bus AD₇–AD₀ and generation of the control signals such as MEMR, MEMW, IOR, and IOW.

The 8155 includes 256 bytes of R/W memory, three I/O ports, and a timer. The programmable I/O sections of this device are illustrated in the following sections.

14.2.1 The 8155 Programmable I/O Ports and Timer

The 8155 is a device with two sections: the first is 256 bytes of R/W memory, and the second is a programmable I/O. Functionally, these two sections can be viewed as two independent chips. The I/O section includes two 8-bit parallel I/O ports (A and B), one 6-bit port (C), and a timer (Figure 14.5). All the ports can be configured simply as input/output ports. Ports A and B also can be programmed in the handshake mode, each port using three signals as handshake signals from port C. The timer is a 14-bit down-counter and has four modes. Pins PA, PB, and PC, shown in Figure 14.5, correspond to ports A, B, and C.

CONTROL LOGIC

The control logic of the 8155 is specifically designed to eliminate the need for externally demultiplexing lines AD₇–AD₀ and generating separate control signals for memory and I/O. Figure 14.5 shows five control signals; all except the Chip Enable (CE) are input signals directly generated by the 8085.

- CE—Chip Enable: This is a master Chip Select signal connected to the decoded high-order bus.
- IO/M—When this signal is low, the memory section is selected, and when it is high, the I/O section (including timer) is selected.
- ALE—Address Latch Enable: This signal latches the low-order address AD₇–AD₀, CE, and IO/M into the chip.
- RD and WR—These are control signals to read from and write into the chip registers and memory.
- RESET—This is connected to RESET OUT of the 8085 and this resets the chip and initializes I/O ports as input.

THE 8155 I/O PORTS

The I/O section of the 8155 includes a control register, three I/O ports, and two registers for the timer (Figure 14.6). The expanded block diagram of the I/O section (Figure 14.6) represents a typical programmable I/O, as discussed in Section 14.1.2. In that section, two address lines plus the Chip Select logic were used to determine port addresses. The 8155 I/O section requires three address lines—AD₂ to AD₀ (A₂–A₀ internally)—and the Chip Enable logic to specify one of the seven registers. In addition, two control signals, RD and WR, are necessary to read from and write into these I/O registers.

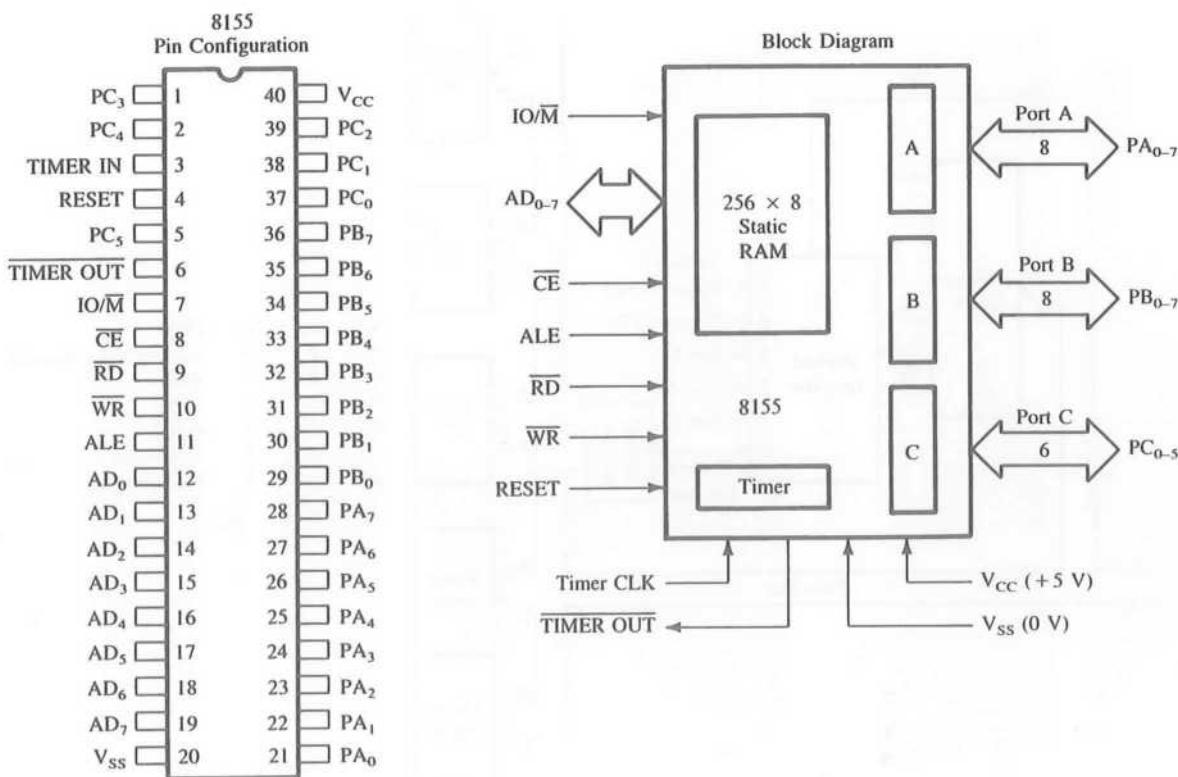


FIGURE 14.5

8155 Pin Configuration and Block Diagram

SOURCE: Intel Corporation, *Embedded Microprocessors* (Santa Clara, Calif.: Author, 1994), pp. 1-31.

To communicate with peripherals through the 8155 the following steps are necessary:

1. Determine the addresses (port numbers of the registers and I/Os) based on the Chip Enable logic and address lines AD₀, AD₁, and AD₂.
2. Write a control word in the control register to specify I/O functions of the ports and the timer characteristics.
3. Write I/O instructions to port addresses to communicate with peripherals.
4. Read the status register, if necessary, to verify the status of the I/O ports and the timer.
In simple applications, this step is not necessary.

CHIP ENABLE LOGIC AND PORT ADDRESSES

Address lines AD₂-AD₀, also shown as A₂-A₀ after internal demultiplexing, select one of the registers, as shown in Figure 14.6(b). Address lines A₃-A₇ are don't care lines; however, the logic levels on the corresponding high-order lines, A₁₁-A₁₅, will be duplicated on lines A₃-A₇, as explained in the next example.

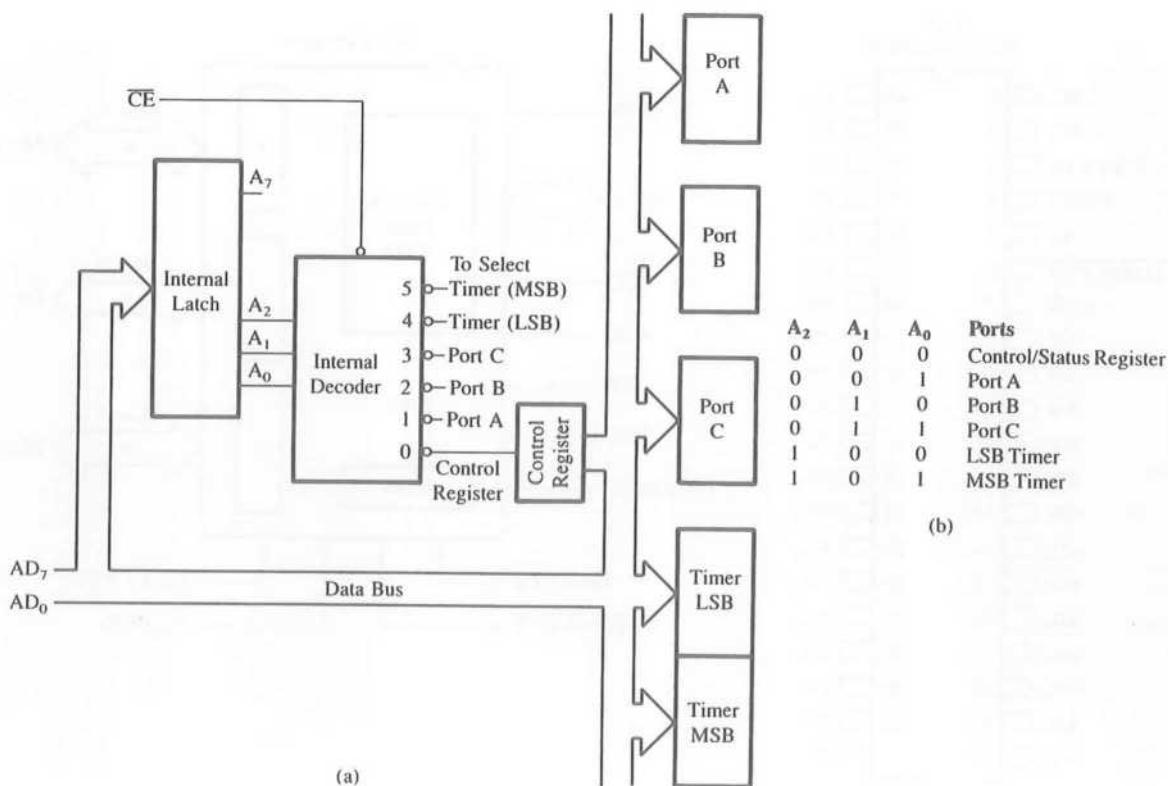


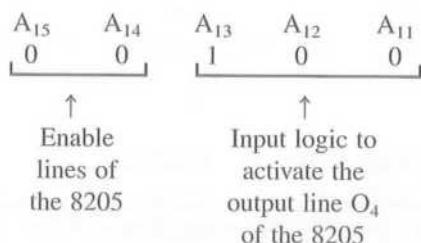
FIGURE 14.6
Expanded Block Diagram of the 8155 (a) and Its I/O Address: Selection (b)

Example 14.2

Determine the addresses of the control/status register, I/O ports, and timer registers in Figure 14.7.

Solution

To select the chip, the output line O₄ of the 8205* (3-to-8) decoder (Figure 14.7) should go low. Therefore, the logic levels of A₁₅–A₁₁ should be as follows:



*The 8205 is a 3-to-8 decoder similar to the 74LS138 except with higher current capacity and speed.

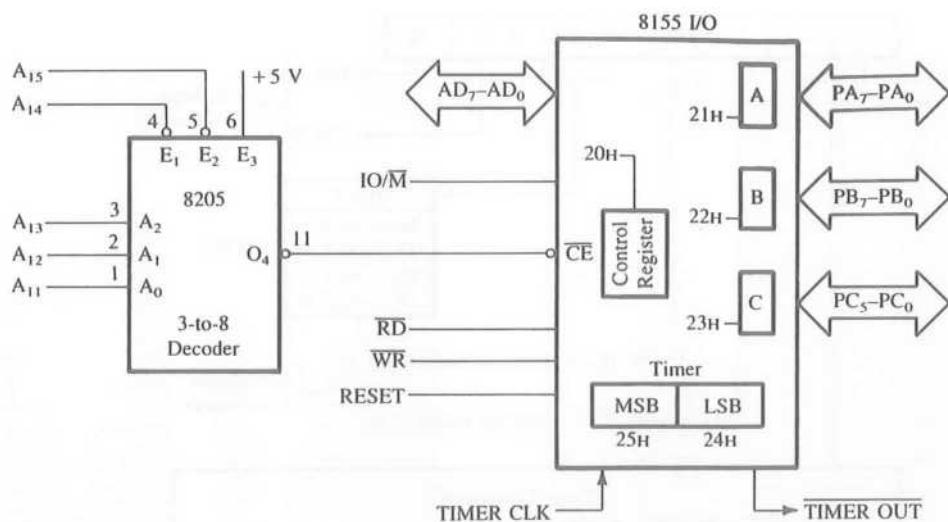


FIGURE 14.7
Interfacing 8155 I/O Ports (Schematic from the SDK-85 System)

By combining five high-order address lines with three low-order address lines (A_2-A_0), the port numbers in Figure 14.7 will range from 20H to 25H, as shown below.

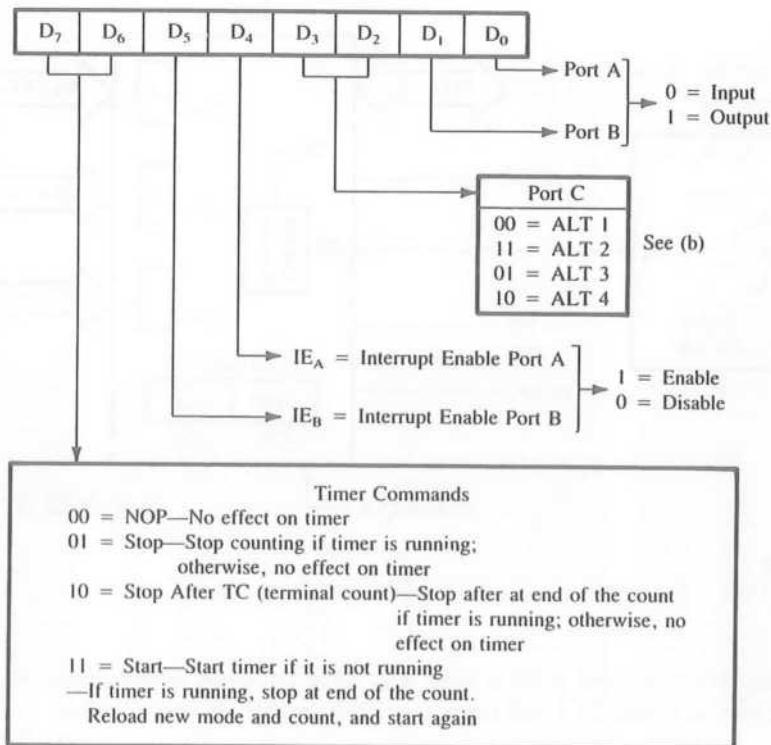
A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	AD_2	AD_1	AD_0	= Addresses	Ports
0	0	1	0	0	0	0	0	= 20H—Control or status register	
					0	0	1	= 21H—Port A	
					0	1	0	= 22H—Port B	
					0	1	1	= 23H—Port C	
					1	0	0	= 24H—Timer (LSB)	
↓					1	0	1	= 25H—Timer (MSB)	
A_7	A_6	A_5	A_4	A_3					

This raises a question: How is it possible to combine five high-order address lines with three low-order address lines to generate a port address? To find an answer to this question, examine the execution of either the **IN** or the **OUT** instruction. When these instructions are executed, the high-order and low-order address buses carry the same information. In this case, the logic levels required on lines $A_{15}-A_{11}$ for the Chip Enable are also duplicated on the address lines from A_7 through A_3 , as shown above.

CONTROL WORD

The I/O ports and the timer can be configured by writing a control word in the control register. The control register bits are defined as shown in Figure 14.8.

In this control word, outputs are defined with logic 1 and inputs with logic 0. The first two LSBs, D_0 and D_1 , determine I/O functions of ports A and B; and the MSBs, D_7



(a)

Table: ALT 1–ALT 4: Port C Bit Assignments, Defined by Bits D₃ and D₂ in the Control Register

ALT	D ₃	D ₂	PC ₅	PC ₄	PC ₃	PC ₂	PC ₁	PC ₀
ALT 1	0	0	I	I	I	I	I	I
ALT 2	1	1	O	O	O	O	O	O
ALT 3	0	1	O	O	O	STB _A	BF _A	INTR _A
ALT 4	1	0	STB _B	BF _B	INTR _B	STB _A	BF _A	INTR _A

I = Input, STB = Strobe, INTR = Interrupt Request

O = Output, BF = Buffer Full, Subscript A = Port A

B = Port B

(b)

FIGURE 14.8

Control Word Definition in the 8155 (a) and Table of Port C Bit Assignments (b)

and D₆, determine timer functions. Bits D₂ and D₃ determine the functions of port C; their combination specifies one of the four alternatives, from simple I/O to interrupt I/O, as shown in Figure 14.8(b). Bits D₄ and D₅ are used only in the interrupt mode to enable or disable internal flip-flops of the 8155. These bits do not have any effect on the Interrupt Enable flip-flop (INTE) of the MPU.

The next section shows an application of the 8155 to design two output ports. An application of the 8155 in the handshake mode is illustrated later.

14.2.2 Illustration: Interfacing Seven-Segment-LED Output Ports Using the 8155

PROBLEM STATEMENT

1. Design two seven-segment-LED displays using ports A and B of the 8155.
2. Write initialization instructions and display data bytes at each port.

HARDWARE DESCRIPTION

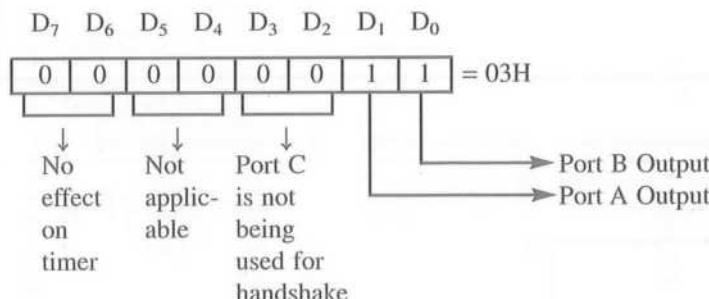
Figure 14.9 shows two seven-segment output ports: port A with the Hewlett-Packard HP 5082/7340, and port B with the 9370 Hex decoders and common-anode seven-segment LEDs. The HP 5082 includes an internal decoder/driver. Both are functionally similar; however, a seven-segment display with an internal built-in decoder/driver is more expensive.

The decode logic is the same as that used in the previous discussion; therefore, the port addresses are as follows:

Control Register = 20H
Port A = 21H
Port B = 22H

CONTROL WORD

To configure ports A and B as outputs, the control word is as follows:



PROGRAM

```

MCI A,03H      ;Initialize ports A and B as output ports
OUT 20H
MVI A,BYTE1
OUT 21H      ;Display BYTE1 at port A
MVI A,BYTE2
OUT 22H      ;Display BYTE2 at port B
HLT

```

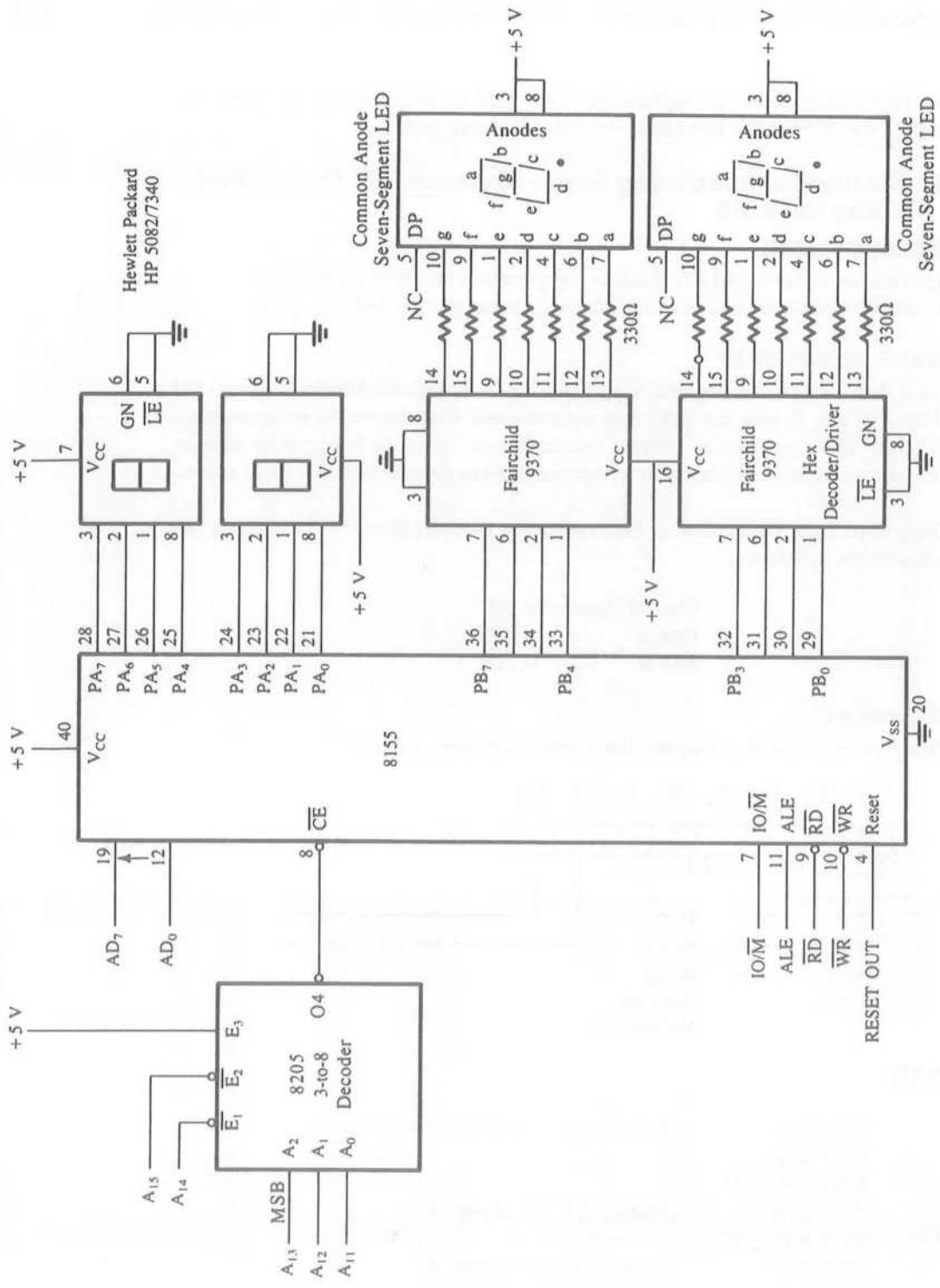


FIGURE 14.9
Interfacing 8155 I/O Ports with Seven-Segment LEDs

PROGRAM DESCRIPTION

The instruction MVI A,03H initializes ports A and B as simple output ports, and the following instructions display data BYTE1 and data BYTE2 at ports A and B, respectively.

14.2.3 The 8155 Timer

The timer section of the 8155 has two 8-bit registers; 14 bits are used for the counter, two bits for the timer mode, and it requires a clock as an input. This 14-bit down-counter provides output in four different modes, as described below.

Figure 14.10(a) shows two registers for a 14-bit count, one for LSB (low significant byte) and one for MSB (most significant byte). The most significant bits M_2 and M_1 are used to specify the timer mode. To operate the timer, a 14-bit count and mode bits are loaded in the registers. An appropriate control word starts the counter, which decrements the count by two at each clock pulse. The timer outputs vary according to the mode specified; see Figure 14.10(b).

The timer can be stopped either in the midst of counting or at the end of a count (applicable to Modes 1 and 3). In addition, the actual count at a given moment can be obtained by reading the status register. These details will be described later.

14.2.4 Illustration: Designing a Square-Wave Generator Using the 8155 Timer

PROBLEM STATEMENT

Design a square-wave generator with a pulse width of 100 μ s by using the 8155 timer. Set up the timer in Mode 1 if the clock frequency is 3 MHz. Use the same decode logic and the port addresses as in Example 14.2 (Figure 14.7).

PROBLEM ANALYSIS

Timer Count The pulse width required is 100 μ s; therefore, the count should be calculated for the period of 200 μ s. The timer output stays high for only half the count.

$$\begin{aligned} \text{Clock Period} &= \frac{1}{f} = \frac{1}{3} \times 10^6 = 330 \text{ ns} \\ \text{Timer Count} &= \frac{\text{Pulse Period}}{\text{Clock Period}} = \frac{200 \times 10^{-6}}{330 \times 10^{-9}} = 606 \\ \text{Count} &= 025EH \end{aligned}$$

Assuming the same decode logic for the 8155 Chip Enable line as in Example 14.2, the port addresses for the timer registers are

Timer LSB = 24H

Timer MSB = 25H

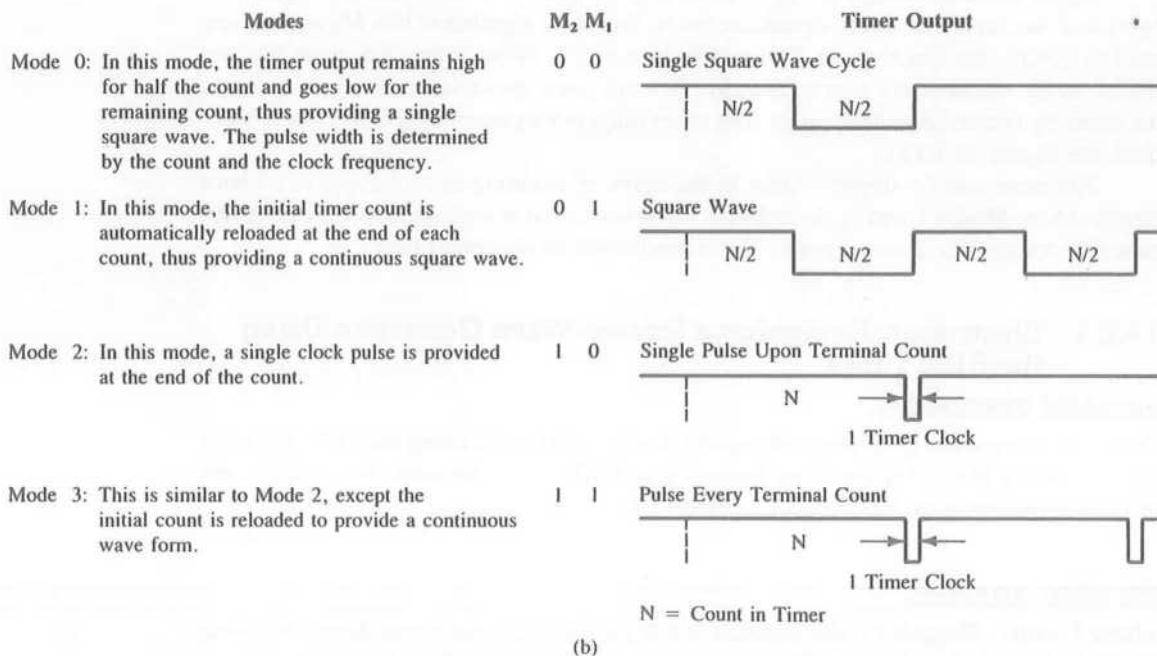
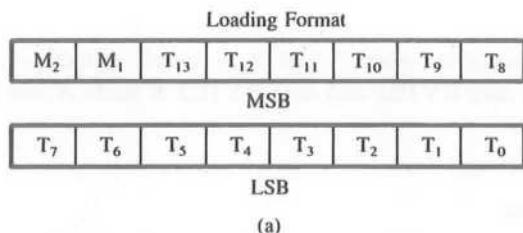
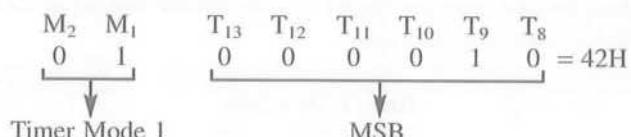


FIGURE 14.10
Timer Loading Format (a) and Modes (b)

The least significant byte, 5EH (of the count 025EH), should be loaded in the timer register with address 24H. The most significant byte is determined as follows:



Therefore, 42H should be loaded in the timer register with the address 25H.

Control Word Assuming the same configuration for ports A and B as before, only bits D₇ and D₆ should be set to 1 to start the counter (see control word definition in Figure 14.8).

Therefore, Control Word: 1100 0011 = C3H

Initialization Instructions

MVI A,5EH	;LSB of the count
OUT 24H	;Load the LSB timer register
MVI A,42H	;MSB of the count
OUT 25H	;Load the MSB timer register
MVI A,C3H	
OUT 20H	;Start the timer
HLT	

14.2.5 The 8155 I/O Ports in Handshake Mode

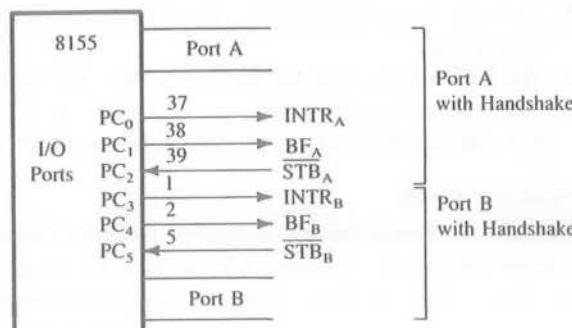
In the handshake mode, data transfer occurs between the MPU and peripherals using control signals called handshake signals. Two I/O ports of the 8155, A and B, can be configured in the handshake mode; each uses three signals from port C as control signals (Figure 14.11). Another alternative (ALT 3 in the table in Figure 14.8) available in the 8155 is to configure port A in the handshake mode with three control signals from port C, configure port B as simple I/O, and configure the remaining three bits of port C as outputs. The details of configuring ports A and B in the handshake mode by using the pins of port C are given below.

CONTROL SIGNALS IN HANDSHAKE MODE

When both ports A and B are configured in the handshake mode, port A uses the lower three signals of port C (PC₀, PC₁, and PC₂), and port B uses the upper three signals (PC₃, PC₄, and PC₅), as shown in Figure 14.11. The functions of these signals are as follows:

- **STB** (Strobe Input): This is an input handshake signal from a peripheral to the 8155. The low on this signal informs the 8155 that data are strobed into the input port.
- **BF** (Buffer Full): This is an active high signal, indicating the presence of a data byte in the port.

FIGURE 14.11
8155 with Handshake Mode



- **INTR (Interrupt Request):** This signal is generated by the rising edge of the STB signal if the interrupt flip-flop (INTE) is enabled. This signal can be used to interrupt the MPU.
- **INTE (Interrupt Enable):** This is an internal flip-flop used to enable or disable the interrupt capability of the 8155. The interrupts for port A and port B are controlled by bits D₄ and D₅, respectively, in the control register.

These control signals can be used to implement either interrupt I/O or status check I/O.

INPUT

Figure 14.12(a) shows the sequence of events and timing in data input to the 8155; they can be described as follows:

1. An external peripheral places data in the input port and informs the 8155 by causing the STB signal to go low.
2. The falling edge of the STB sets signal BF (Buffer Full) high, informing the peripheral to wait.
3. When the STB goes high, the rising edge of the STB can generate signal INTR if the internal interrupt flip-flop INTE is set. The interrupt flip-flops are set or reset by the control word.
4. The last step is to transfer data from the 8155 input port to the MPU. This can be done either by interrupting the MPU with the INTR signal or by checking the status of signal BF. The MPU can check the status by reading the status register (described later). When the MPU reads data, the INTR and BF signals are reset. When the BF signal goes low, it informs the peripheral that the port is empty, and the device is ready for the next byte.

OUTPUT

The sequence of events and timing in data output from the 8155 port to a peripheral are as follows; see Figure 14.12(b):

1. When the output is empty, the MPU writes a byte in the port.
2. The falling edge of the WR signal resets the INTR signal and the rising edge sets the BF (Buffer Full) signal high, informing the peripheral that a byte is available in the port.
3. After receiving the data byte, the peripheral acknowledges by sending the STB signal (active low).
4. The STB signal resets the BF signal low and generates the interrupt request by setting INTR high. Now the MPU can be informed by the interrupt signal to send the next byte, or the MPU can sense that the port is empty through a status check.

STATUS WORD

The MPU can read the status register to check the status of the ports or the timer. The control register and the status register have the same port address; they are differentiated only by the RD and WR signals. The status register bits are defined in Figure 14.13.

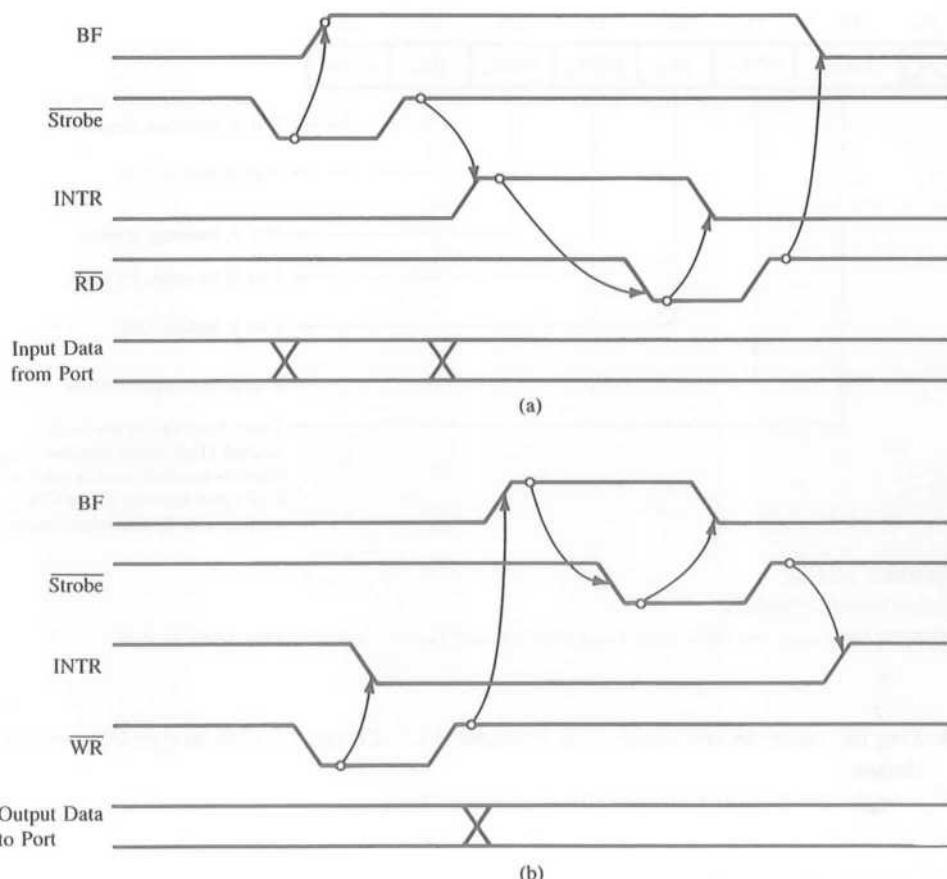


FIGURE 14.12

Timing Waveforms of the 8155 I/O Ports with Handshake: Input Mode (a) and Output Mode (b)

SOURCE: Intel Corporation, *Embedded Microprocessors* (Santa Clara, Calif.: Author, 1994), pp. 1-43.

14.2.6 Illustration: Interfacing I/O Ports in Handshake Mode Using the 8155

PROBLEM STATEMENT

Design an interfacing circuit using the 8155 to read and display from an A/D converter to meet the following requirements:

1. Set up port A in the handshake mode to read data from an A/D converter.
2. Set up port B as an output port to display data at seven-segment LEDs.
3. Use line PC₃ from port C to initiate a conversion.

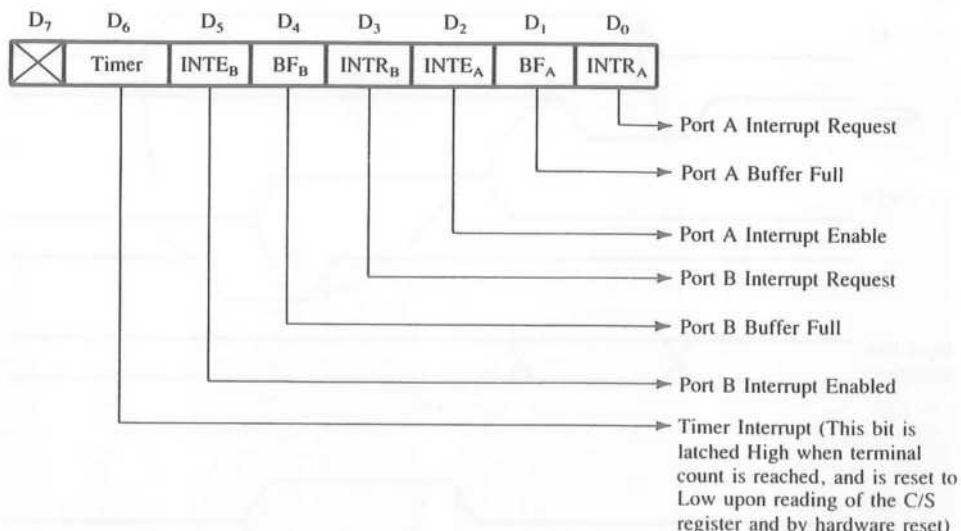


FIGURE 14.13
Status Word Definition

SOURCE: Intel Corporation, *MCS—80/85 Family User's Manual* (Santa Clara, Calif.: Author, 1979), pp. 6–20.

4. Use the same decode logic as in Example 14.2 (Figure 14.7) to assign I/O port addresses.
5. Use the 8155 timer to record the conversion time.

PROBLEM ANALYSIS

Figure 14.14 shows an interfacing circuit that uses the 8155 I/O ports as follows:

1. Port A is configured as an input port in the handshake mode for reading data from the A/D converter.
2. Port B is configured as a simple output port for seven-segment LEDs.
3. The upper half of port C is a simple output port, and bit PC₃ is being used to start conversion.
4. The lower half of port C provides handshake signals for port A. Bit PC₂ is being used as a strobe (STB) to inform the 8155 that the conversion is complete and that the output of the converter has been placed in port A.

INPUT WITH STATUS CHECK

The circuit shows that the INTR signal (bit PC₀) is not being used. This suggests that port A is configured for status check and not for interrupt I/O. Therefore, the control word (see

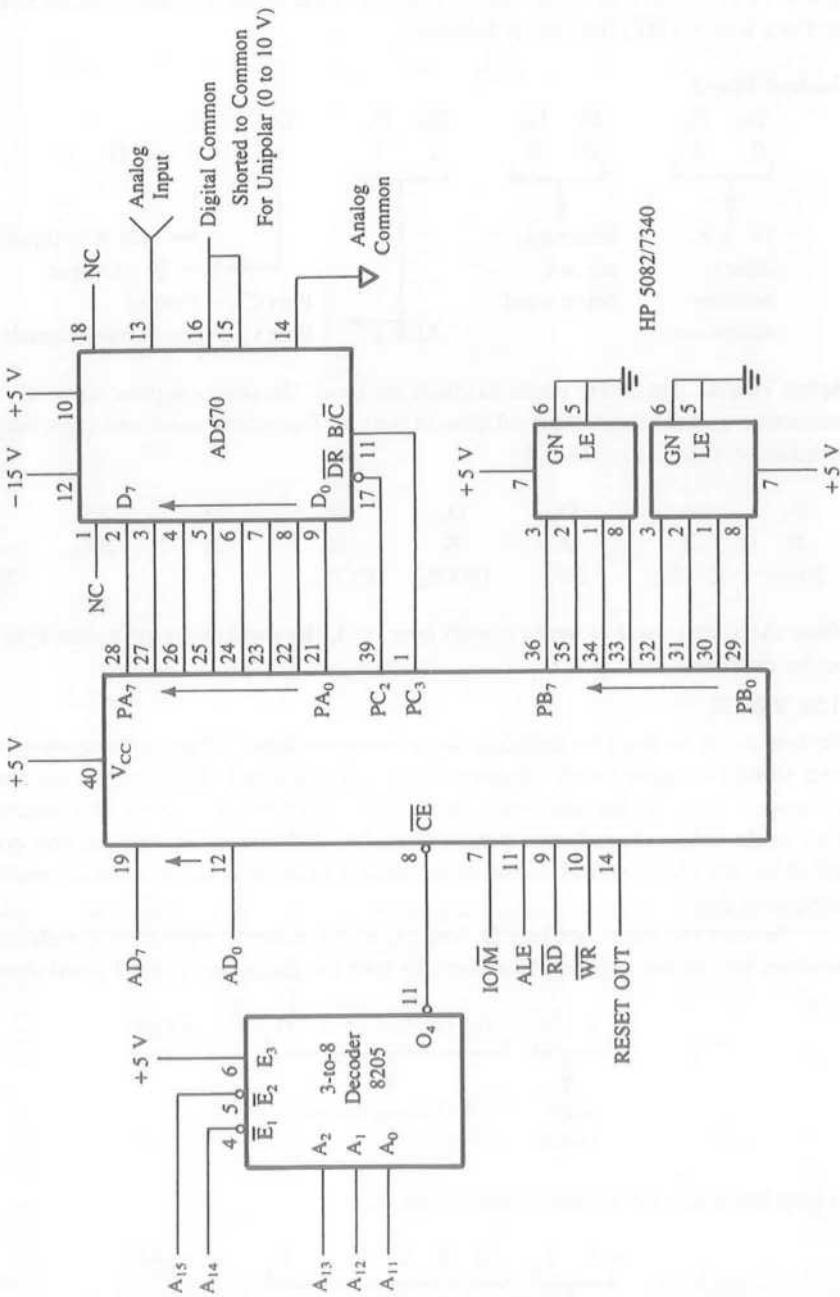
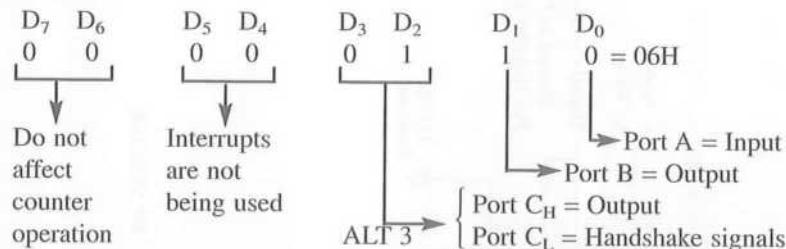


FIGURE 14.14
Interfacing the A/D Converter AD570 in the Handshake Mode

Figure 14.8) required to set up the ports as specified above and the masking byte to check the Data Ready (DR) line are as follows:

Control Word



Status Word The MPU needs to check bit D₁ of the status register to verify the end of conversion and the availability of data in port A. The status word will have the following information (see Figure 14.13):

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
X	X	X	X	X	X	BF _A	X

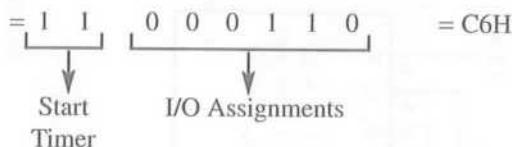
Timer INTE_B BF_B INTR_B INTE_A INTR_A

When the status word is masked with byte 02H, the availability of a data byte in port A can be verified.

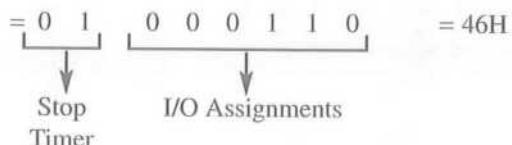
8155 TIMER

The timer can be used to calculate the conversion time. When a conversion begins, the timer should be started with a known count, and at the end of conversion the timer should be stopped. To calculate the count, the reading must be divided by two because at each clock cycle the count is decremented twice. The difference between the two counts multiplied by the clock period of the timer should provide a fairly accurate reading of the conversion time.

To start the timer, set bits D₇ and D₆ of the control register to 1 without affecting the other bits of the register. Therefore, to start the timer, the control word should be



To stop the timer, the control word should be



POR T ADDRESSES

The decode logic is the same as in Example 14.2; therefore, the I/O port addresses range from 20H (control register) to 25H (Timer—MSB).

PROGRAM

MVI A,06H	;Control word for I/O ports
OUT 20H	;Set up ports as specified
MVI A,00H	;Load 0000H in the timer registers
OUT 24H	
OUT 25H	
MVI A,08H	;Byte to set $PC_3 = 1$
OUT 23H	;Send START pulse
MVI A,C6H	;Control word to start timer
OUT 20H	;Start timer
MVI A,00H	;Byte to set $PC_3 = 0$
OUT 23H	;Start conversion
STATUS: IN 20H	;Read status register
ANI 02H	;Check status of DR
JZ STATUS	;If $BF_A = 0$, wait in the loop until a data byte is available
MVI A,46H	;Byte to stop counter
OUT 20H	;Stop counter
IN 21H	;Read A/D converter output
OUT 22H	;Display data at port B
IN 24H	;Read LSB of timer count
MOV L,A	;Save timer count in register L
IN 25H	;Read MSB of timer count
ANI 3FH	;Delete D_7 and D_6 from the MSB; they represent ; timer mode
MOV H,A	;Save MSB timer count in H
LHLD RWM	;Store timer count from HL register in R/W memory ; locations
HLT	

PROGRAM DESCRIPTION

The comments are self-explanatory; however, some explanation is needed for the timer count, start conversion (convert) pulse, and status check.

The program loads 0000H in the timer register, and after the first decrement, the count becomes 3FFFH. This is a 14-bit counter, with bits D_{15} and D_{14} reserved to specify the mode. However, in this particular problem, the counter mode is irrelevant. This program assumes that the A/D conversion time is less than the time period given by the maximum count. The difference between the initial count and the final count will provide the necessary value to calculate the conversion time. The program does not perform this subtraction; it just stores the final count in two consecutive memory locations labeled as RWM.

The second item needing explanation is the start conversion (convert) pulse. This is an active high pulse provided by turning on and off bit PC₃ in port C.

Finally, instruction IN 20H reads the status register, and instructions ANI 02 and JZ check whether the buffer in port A (BF_A) is full. The program stays in the loop until the BF_A goes high, indicating the availability of data.

INTERRUPT I/O

This example illustrates all the important I/O operations of the 8155 except the interrupt I/O in the handshake mode. To implement the interrupt I/O in the above example, the INTR_A—the output bit PC₀—should be connected to a vectored interrupt such as RST 6.5 and the control word should be changed accordingly (see Problem 14 at the end of this chapter).

14.2.7 Interfacing I/O Devices with Multiple Addresses

Figure 14.15 shows a decoding technique using the 74139, a 2-to-4 decoder. This device has two 2-to-4 decoders inside; one is used for interfacing I/O ports and the second is used for interfacing memory (not shown here). In this section, we will focus on I/O interfacing.

In Figure 14.15, inputs to the decoder are the address lines A₇ and A₆, and the decoder is enabled by the IO/M signal through an inverter. When the processor asserts IO/M

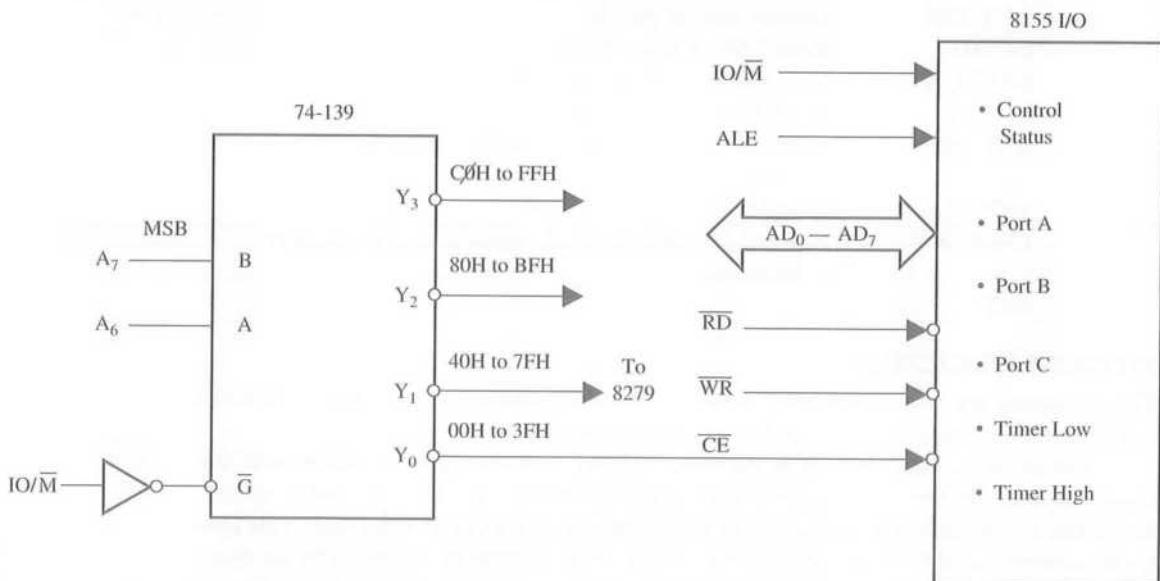


FIGURE 14.15

Interfacing 8155 I/O Ports and 8279 with Multiple Addresses (This figure is extracted from the PRIMER schematic in Appendix B.)

high to access an I/O port, the decoder is enabled. The address lines A₅–A₀ are not decoded in this schematic; some of them are connected to programmable I/O devices such as the 8155 and 8279.

ADDRESSES FOR 8155 I/O PORTS

8155 I/O Addresses

A ₇	A ₆	A ₅	A ₄	A ₃	A ₂	A ₁	A ₀	
0	0	X	X	X	0	0	0	= 00 Control or status register
					0	0	1	= 01 Port A
					0	1	0	= 02 Port B
					0	1	1	= 01 Port C
					1	0	0	= 04 Timer LSB
					1	0	1	= 05 Timer MSB

The 8155 I/O ports are accessed by the output line Y₀ of the decoder; therefore, the address lines A₇ and A₆ must be 0. In these addresses we are assuming that the don't care address lines A₅, A₄, and A₃ are at logic 0. However, these three lines can assume eight different combinations. If we assume these lines to be at logic 1, the addresses will range from 38H to 3DH.

8279 I/O Addresses In this keyboard/display device (discussed in the next section), the address line A₀ is connected to 8279; when A₀ is 0, it accesses the internal data port and when it is 1, it accesses the internal command port. The output line Y₁ of the decoder accesses 8279; therefore, the address lines A₇ and A₆ must be 0 and 1, respectively. Therefore, the range of port addresses of the 8279 is as follows:

8279 I/O Addresses

A ₇	A ₆	A ₅	A ₄	A ₃	A ₂	A ₁	A ₀	
0	1	X	X	X	X	X	0	= 40H Command/status Port
							1	= 41H Data Port

If we assume the don't care address lines A₅ through A₁ at 0, the possible logic combination, these addresses can range from 40H to 7FH.

Multiple I/O Addresses The 8279 needs only two addresses, but because of the five don't care lines, it occupies the space of 64 I/O addresses. In the case of the 8155, three addresses lines are don't care; therefore, it has eight sets of address ranges for its I/O ports and the timer. The advantage of such a technique is in saving costs and space on a printed circuit board. By using one two-input decoder, we can decode only two address lines. To decode the remaining lines, we need additional chips and space on the board. In small systems, the space of the printed circuit, called real estate, is of prime importance. The price we pay for this technique is the assignment of multiple addresses. However, if we do not need all the I/O space in a given system, wasting I/O addresses for a given device to reduce cost is of no consequence.

14.3 THE 8279 PROGRAMMABLE KEYBOARD/DISPLAY INTERFACE

The 8279 is a hardware approach to interfacing a matrix keyboard and a multiplexed display. The software approach to interfacing a matrix keyboard and a multiplexed display of seven-segment LEDs is illustrated in Chapter 17. The disadvantage of the software approach is that the microprocessor is occupied for a considerable amount of time in checking the keyboard and refreshing the display. The 8279 relieves the processor from these two tasks. The disadvantage of using the 8279 is the cost. The trade-offs between the hardware approach and the software approach are the production cost vs. the processor time and the software development cost.

The 8279 (Figure 14.16) is a 40-pin device with two major segments: keyboard and display. The keyboard segment can be connected to a 64-contact key matrix. Keyboard entries are debounced and stored in the internal FIFO (First-In–First-Out) memory; an interrupt signal is generated with each entry. The display segment can provide a 16-character scanned display interface with such devices as LEDs. This segment has 16×8 R/W memory (RAM), which can be used to read/write information for display purposes. The display can be set up in either right-entry or left-entry format.

14.3.1 Block Diagram of the 8279

The block diagram (Figure 14.17) shows four major sections of the 8279: keyboard, scan, display, and MPU interface. The functions of these sections are described below.

KEYBOARD SECTION

This section has eight lines (RL_0 – RL_7) that can be connected to eight columns of a keyboard, plus two additional lines: Shift and CNTL/STB (Control/Strobe). The status of the SHIFT key and the Control key can be stored along with a key closure. The keys are automatically debounced, and the keyboard can operate in two modes: two-key lockout or N-key rollover. In the two-key lockout mode, if two keys are pressed almost simultaneously, only the first key is recognized. In the N-key rollover mode, simultaneous keys are recognized and their codes are stored in the internal buffer; it can also be set up so that no key is recognized until only one key remains pressed.

The keyboard section also includes 8×8 FIFO (First-In–First-Out) RAM. The FIFO RAM consists of eight registers that can store eight keyboard entries; each is then read in the order of entries. The status logic keeps track of the number of entries and provides an IRQ (Interrupt Request) signal when the FIFO is not empty.

SCAN SECTION

The scan section has a scan counter and four scan lines (SL_0 – SL_3). These four scan lines can be decoded using a 4-to-16 decoder to generate 16 lines for scanning. These lines can be connected to the rows of a matrix keyboard and the digit drivers of a multiplexed display.

Pin Configuration

		Pin Number	Pin Name	Description
1	RL ₂	40	V _{CC}	
2	RL ₃	39	RL ₁	
3	CLK	38	RL ₀	
4	IRQ	37	CNTL/STB	
5	RL ₄	36	SHIFT	DB ₀₋₇
6	RL ₅	35	SL ₃	I/O Data Bus (Bidirectional)
7	RL ₆	34	SL ₂	CLK
8	RL ₇	33	SL ₁	RESET
9	RESET	32	SL ₀	CS
10	RD	8279	OUT B ₀	Control Input
11	WR	31	OUT B ₀	Reset Input
12	DB ₀	30	OUT B ₁	Chip Select
13	DB ₁	29	OUT B ₂	Read Input
14	DB ₂	28	OUT B ₃	Write Input
15	DB ₃	27	OUT A ₀	Buffer Address
16	DB ₄	26	OUT A ₁	Interrupt Request Output
17	DB ₅	25	OUT A ₂	Scan Lines
18	DB ₆	24	OUT A ₃	Return Lines
19	DB ₇	23	BD	Shift Input
20	V _{SS}	21	A ₀	Control Strobe Input
				OUT A ₀₋₃
				Display (A) Outputs
				OUT B ₀₋₃
				Display (B) Outputs
				Blank Display Output
				OUT A ₀₋₃
				OUT B ₀₋₃
				Display Data

Logic Symbol

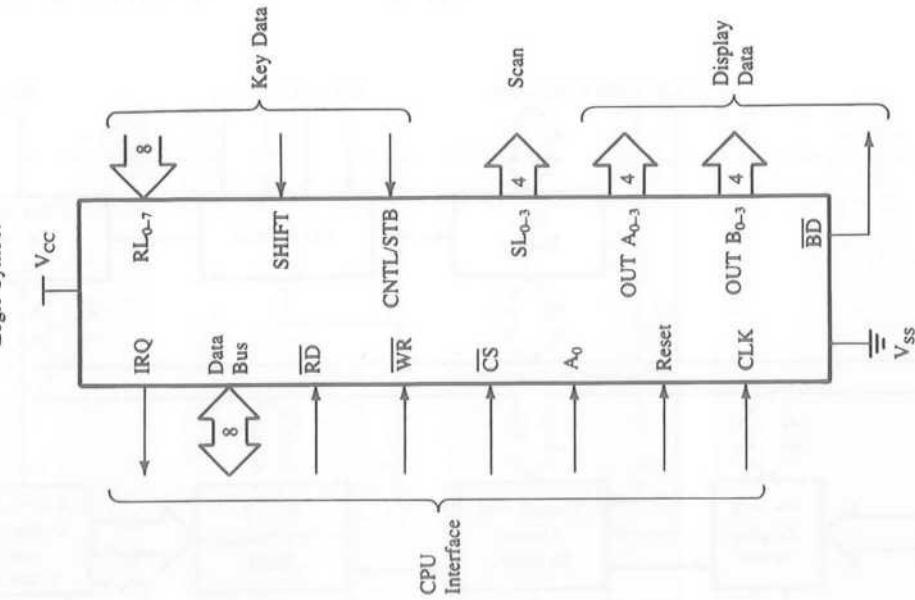


FIGURE 14.16
The 8279 Logic Pinout

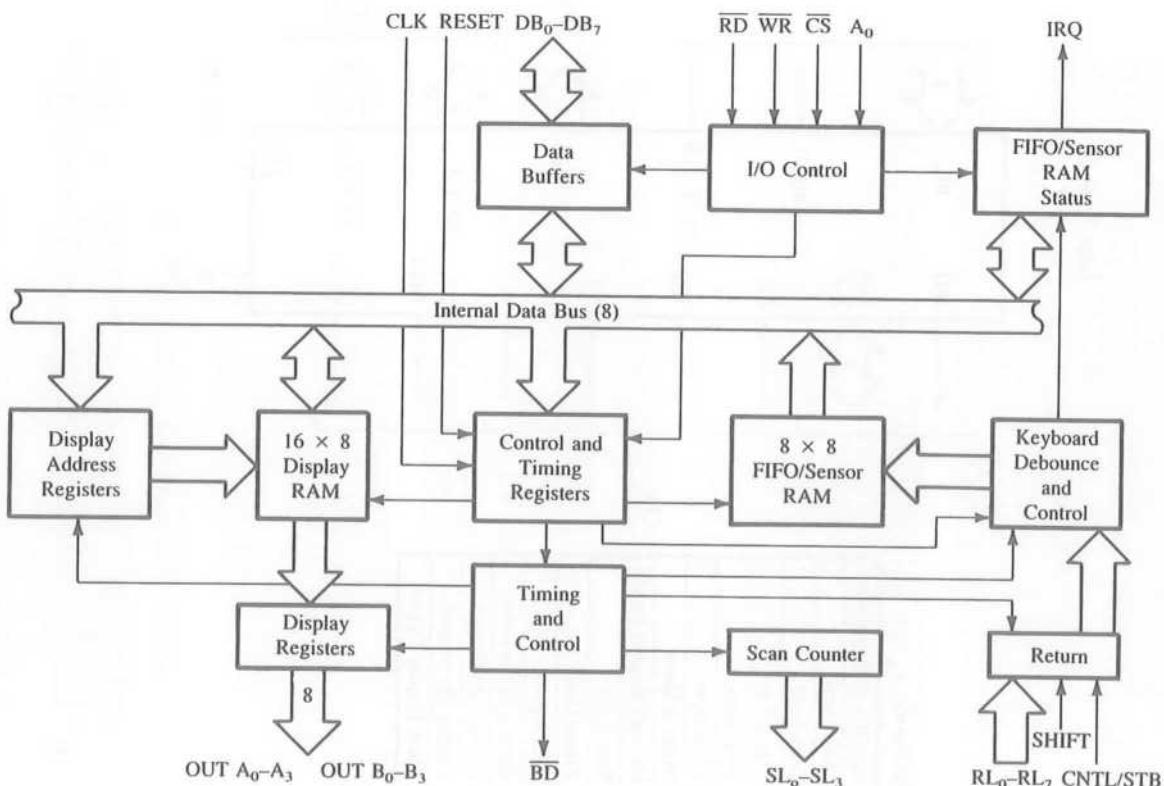


FIGURE 14.17

The 8729 Logic Block Diagram

SOURCE: Intel Corporation, *Peripheral Components* (Santa Clara, Calif.: Author, 1993), pp. 3–218.

DISPLAY SECTION

The display section has eight output lines divided into two groups A₀-A₃ and B₀-B₃. These lines can be used, either as a group of eight lines or as two groups of four, in conjunction with the scan lines for a multiplexed display. The display can be blanked by using the BD line. This section includes 16 × 8 display RAM. The MPU can read from or write into any of these registers.

MPU INTERFACE SECTION

This section includes eight bidirectional data lines (DB₀-DB₇), one Interrupt Request line (IRQ), and six lines for interfacing, including the buffer address line (A₀).

When A₀ is high, signals are interpreted as control words or status; when A₀ is low, signals are interpreted as data. The IRQ line goes high whenever data entries are stored in the FIFO. This signal is used to interrupt the MPU to indicate the availability of data.

14.3.2 Programming the 8279

The 8279 is a complex device that can accept eight different commands to perform various functions.*

The initialization commands can specify

1. left or right entry and key rollover.
2. clock frequency prescaler.
3. starting address and incrementing mode of the FIFO RAM.
4. RAM address to read and write data and incrementing mode.
5. blanking format.

To illustrate important command words, the next section will illustrate the keyboard/display circuit shown in Figure 14.18.

14.3.3 Illustration: Keyboard/Display Interfacing Using the 8279

Figure 14.18 shows the keyboard/display circuit of a single-board system.

1. Explain the functions of various components in the circuit.
2. Explain the decoding logic, and identify the port addresses of the 8279 registers.
3. Explain the initialization instructions given later.

CIRCUIT DESCRIPTION

Figure 14.18 shows the following components:

- The 8279 Programmable Keyboard/Display Interface
- A matrix keyboard with 20 keys: one matrix of 16 keys (4×4) and the other with four keys (1×4)
- Six seven-segment LEDs in groups of two
- A 3-to-8 decoder to generate additional scan lines
- A 2981A, a current driver for anodes and transistors as current drivers for cathodes

Lines RL_0 – RL_3 (Return Lines) of the 8279 are connected to the columns of the matrix keyboard, and the output lines (A_0 – A_3 and B_0 – B_3) are connected to drive the LED segments. The three scan lines are connected to the decoder, the 74HC138, to generate eight decoded signals. In this circuit, six output lines of the decoder are connected as digit drivers to turn on six seven-segment LEDs; two output lines are unused. The 8279 has four scan lines that can be decoded to generate 16 output lines to drive 16 displays. The data lines of the 8279 are connected to the data bus of the 8085, and the IRQ (Interrupt Request) is connected to the RST 5.5 of the system.

Four signals—RD, WR, CLK, and RESET OUT—are connected directly from the 8085. The system has a 3.072 MHz clock; when the 8279 is reset, the clock prescaler is

*For a complete description of the 8279 see Appendix D or refer to Intel's *MCS-80/85 Family User's Manual*.

KEYPAD BUS

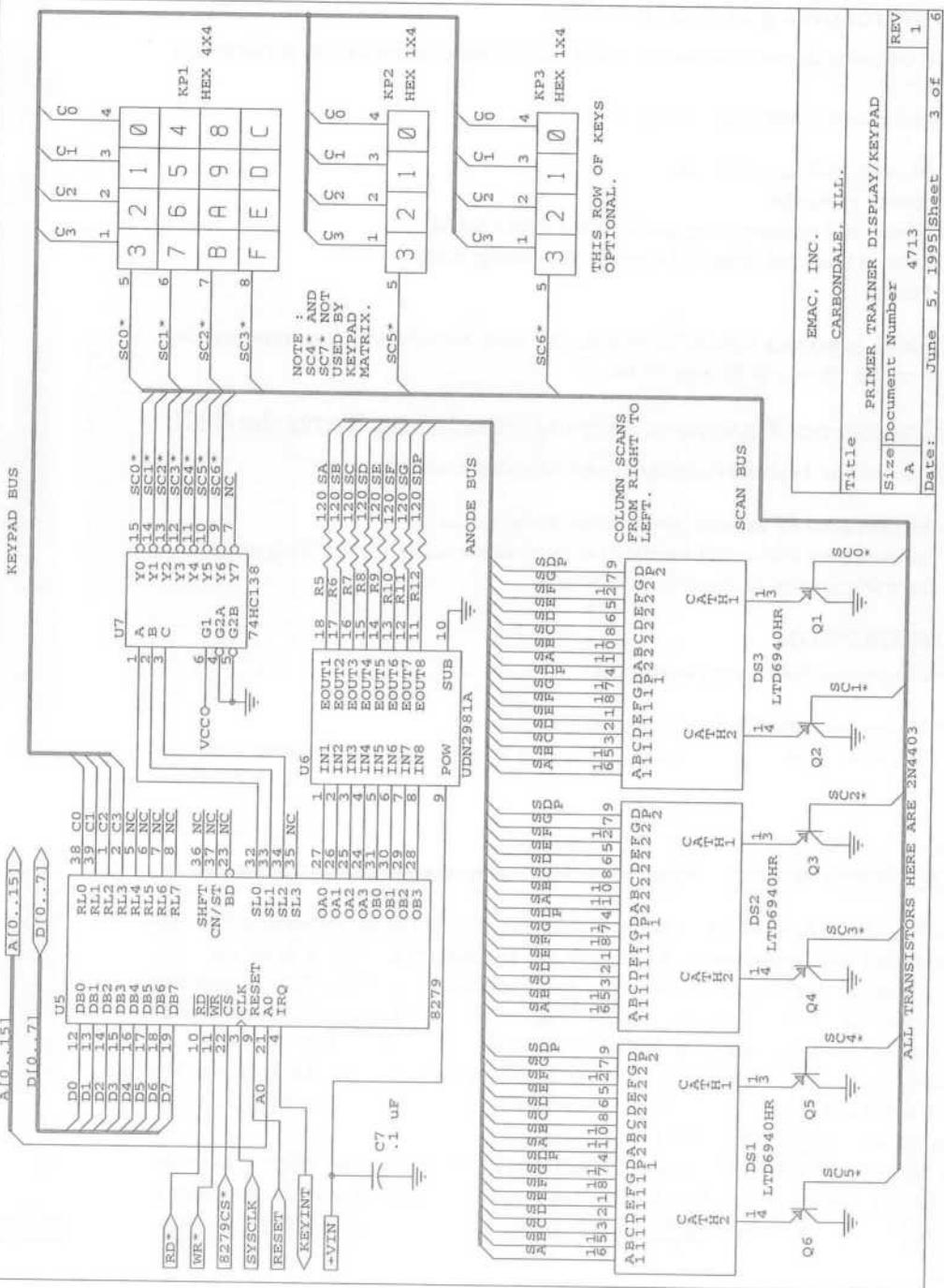


FIGURE 14.18

Keyboard/display circuit
SOURCE: Courtesy of EMAC Inc.

set to 31. This divides the clock frequency by 31 to provide the scan frequency of approximately 100 kHz. The RESET signal also sets the 8279 in the mode of 16-character display with two-key lockout keyboard.

After the initialization of the 8279, the respective codes are sent to the display RAM to display any characters. The 8279 takes over the task of displaying the characters by outputting the codes and digit strobes. To read the keyboard, the 8279 scans the columns; if a key closure is detected, it debounces the key. If a key closure is valid, it loads the key code into the FIFO, and the IRQ line goes high to interrupt the system.

DECODING LOGIC AND PORT ADDRESSES

The port addresses of the 8279 registers are determined by two signals: \overline{CS} and A_0 . The \overline{CS} of the 8279 is connected to the Y_1 signal generated by the address decoding circuit in Figure 14.15 and A_0 of the 8279 is directly connected to the address line A_0 of the 8085 processor. For commands and status, A_0 should be high and for data transfer A_0 should be low. The addresses of the Data port and the Command/Status port are 40H and 41H, respectively, as discussed in Section 14.2.7 (Figure 14.15).

INITIALIZATION INSTRUCTIONS

In Figure 14.18, we need to initialize the 8279 to read a key and display the key at the seven-segment display. The definitions of various control words necessary for initialization are given in Appendix D. The RESET signal sets the clock prescaler to 31. This prescaler divides the system clock (3.072 MHz) to set the scan frequency at approximately 100 kHz. When the 8279 detects a key closure, the IRQ signal interrupts the 8085, using RST 5.5. The interrupt service routine sends the command word to read from the keyboard (Command Port 41H), reads the character data from the keyboard (Data Port 40H), and stores it in the system's R/W memory location, the input buffer (IBUFF). The following instructions illustrate the initialization and the interrupt service routine.

Initialization

		Keyboard/Display Mode
MVI A, 00000000	;Set mode: Left entry, 8-character,	Code: 0 0 0 D D K K K
	;2-key lockout, encoded scan	
	;	
OUT 41H	;Send to Command Port	

Program Clock

MVI A, 00111111	;Divide clock by 31—this is	Code: 0 0 1 P P P P P
	;	
OUT 41H	; unnecessary	
	;	
	shown here for illustration	

Clear

MVI A, 11000001	;Clear Display RAM and FIFO status	Code: 1 1 0 C C C C C
OUT 41H		

Write Display RAM

MVI A, 10000000 ;Set up 8279 for display

Code:

1	0	0	A	A	A	A	A
---	---	---	---	---	---	---	---

Interrupt Routine to Read a Keyboard

PUSH H ;Save registers
 PUSH PSW

Read FIFO RAM

MVI A, 01000000 ;Control word to read FIFO RAM

Code:

0	1	0	A	X	A	A	A
---	---	---	---	---	---	---	---

OUT 41H

IN 40H ;Read Data

Data Format

ANI 00111111 ;Mask CNTL & SHFT keys

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
CNTL	SHFT	ROW			COL		

STA IBUFF ;Store in R/W memory

POP PSW ;Restore registers

POP H

RET

The six seven-segment LED display is divided into two segments: left four LEDs for memory address and right two LEDs for data value. The software instructions should determine whether it is a four-digit or two-digit display and send the control word to write into the display RAM.

For example, to display a 4-digit memory address, the control word instructions are as follows:

Write Display RAM Control Word

MVI A,90H ;Control word to write
 ; starting at first RAM
 ; location

1	0	0	A ₁	A	A	A	A
---	---	---	----------------	---	---	---	---

OUT 41H

MVI A,CODE ;Load seven-segment code
 OUT 40H ;Output the code

To display a 2-digit data value, the control instructions are as follows:

MVI A,94H ;Control word to display data
 OUT 41H

In this example, the control word 94H points to the fifth memory location in the display RAM; the first four locations are reserved for memory addresses.

SUMMARY

This chapter was concerned with the basic concepts (such as control register, control logic, chip select logic, and handshake signals) underlying a programmable device. The characteristics of a programmable device were discussed using the bidirectional buffer 74LS245, and the important concepts related to a programmable device were reviewed in Section 14.1.4. On the basis of these concepts, two programmable devices—the 8155 (R/W memory with I/O) and the 8279 (keyboard/display interface)—were discussed.

The 8155 is a multipurpose device that includes memory, timer, and I/O ports. Interfacing applications of the I/O ports in various modes (including handshake) and the timer were illustrated with examples. Similarly, interfacing and initialization of the 8279 (keyboard/display interface) was illustrated, using the circuit from a single-board system.

QUESTIONS, PROBLEMS, AND PROGRAMMING ASSIGNMENTS

1. List the internal components generally found in a programmable device.
2. In a programmable device, how does the MPU differentiate between the control register and the status register if both registers have the same port address?
3. Explain the functions of handshake signals.
4. Explain the difference between setting the 8155 I/O ports in ALT 1 and ALT 3.
5. Specify the handshake signals for port B of the 8155 if port B is connected as an input port in the interrupt mode. Explain the function of each handshake signal.
6. Port B of the 8155 is set up in the handshake mode, and the reading of the status word is 20H. Is the port set up for status check or interrupt I/O?
7. List the major components of the 8279 keyboard/display interface, and explain their functions.
8. In Figure 14.7, specify all the port addresses if the output line 7 of the decoder is connected to CE.
9. In Figure 14.7, assume that the decoder is eliminated and address line A₁₅ is connected to CE through an inverter. Specify the addresses of ports A, B, and C, assuming all don't care lines are at logic 0.
10. Can any port be accessed with port address FDH in Problem 10?
11. Write the instructions to set up the 8155 timer in Mode 3 with count 3FF8H.
12. In Problem 11, specify the output if the clock frequency is 3 MHz and the count is 3080H.
13. Calculate the count for the 8155 timer to obtain the square wave of the 500 µs period if the clock frequency is 3.072 MHz.
14. Modify the circuit in Figure 14.14 to connect the A/D converter with interrupt I/O. Use RST 6.5 for the interrupt.

15. In Problem 14, write the instructions in the main program to enable the RST 6.5 interrupt. Write a service routine to read a data byte, store it in memory location XX70H, and start the next conversion. (Ignore all the specifications related to the timer in the illustration.)

EXPERIMENTAL ASSIGNMENTS

1.
 - a. Connect the A/D converter AD570 as an input (Figure 14.14).
 - b. Set up port A as an input port in ALT 3 with interrupt I/O. Use RST 6.5 for the interrupt.
 - c. Write a main program to
 - initialize port A as an input and port C for handshake signals.
 - start conversion.
 - display data at an output port.
 - set up a continuous loop for displaying data.
 - d. Write a service routine to
 - read a data byte.
 - start conversion for the next reading.
 - e. Record data for various analog signals.
2.
 - a. Set up the 8155 timer as shown in Section 14.2.4.
 - b. Enter and execute the given program.
 - c. Measure the square-wave output on an oscilloscope.
 - d. Calculate the frequency and the pulse width of the square wave if bits T₁₃–T₀ all = 0.
 - e. Load the count from step (d), start the counter, and measure the frequency and the pulse width of the output.

15

General-Purpose Programmable Peripheral Devices

This chapter is an extension of Chapter 14, except that the programmable devices discussed in this chapter are designed for general-purpose use. This chapter describes several programmable devices from the Intel family: the 8255A Peripheral Interface, the 8254 Interval Timer, the 8259A Interrupt Controller, and the 8237 DMA controller.

The 8255A and the 8254, two widely used general-purpose programmable devices, can be compatible with any microprocessor. The 8255A includes three programmable ports, one of which can be used for bidirectional data transfer. This is an important additional feature in comparison with the 8155 I/O ports discussed in the last chapter. The 8254 timer is similar to the 8155 timer, except that it has three 16-bit independent timers with various modes.

The next two devices—the 8259A Interrupt Controller and the 8237 DMA controller—were introduced briefly in Chapter 12. These devices illustrate the implementation of interrupts and of Direct Memory Access by using programmable devices. If you are not familiar with the concepts underlying programmable devices and handshake signals, you are strongly advised to read Section 14.1 before reading this chapter.

OBJECTIVES

- List the elements of the 8255A Programmable Peripheral Interface (PPI) and explain its various operating modes.
- Set up the 8255A I/O ports in the simple I/O and Bit Set/Reset (BSR) mode.

- Design an interfacing circuit to set up the 8255A in the handshake mode (Mode 1) and write instructions to transfer data under status check I/O and interrupt I/O.
- List operating modes of the 8254 timer and write instructions to set up the timer in the various modes.
- Explain the functions of the 8259A interrupt controller and its operation in the fully nested mode.
- Explain the process of the Direct Memory Access (DMA) and the functions of various elements of the 8237.

15.1

THE 8255A PROGRAMMABLE PERIPHERAL INTERFACE

The 8255A is a widely used, programmable, parallel I/O device. It can be programmed to transfer data under various conditions, from simple I/O to interrupt I/O. It is flexible, versatile, and economical (when multiple I/O ports are required), but somewhat complex. It is an important general-purpose I/O device that can be used with almost any microprocessor.

The 8255A has 24 I/O pins that can be grouped primarily in two 8-bit parallel ports: A and B, with the remaining eight bits as port C. The eight bits of port C can be used as individual bits or be grouped in two 4-bit ports: C_{UPPER} (C_U) and C_{LOWER} (C_L), as in Figure 15.1(a). The functions of these ports are defined by writing a control word in the control register.

Figure 15.1(b) shows all the functions of the 8255A, classified according to two modes: the Bit Set/Reset (BSR) mode and the I/O mode. The BSR mode is used to set or reset the bits in port C. The I/O mode is further divided into three modes: Mode 0, Mode 1, and Mode 2. In Mode 0, all ports function as simple I/O ports. Mode 1 is a handshake mode whereby ports A and/or B use bits from port C as handshake signals. In the handshake mode, two types of I/O data transfer can be implemented: status check and interrupt. In Mode 2, port A can be set up for bidirectional data transfer using handshake signals from port C, and port B can be set up either in Mode 0 or Mode 1.

15.1.1 Block Diagram of the 8255A

The block diagram in Figure 15.2(a) shows two 8-bit ports (A and B), two 4-bit ports (C_U and C_L), the data bus buffer, and control logic. Figure 15.2(b) shows a simplified but expanded version of the internal structure, including a control register. This block diagram includes all the elements of a programmable device; port C performs functions similar to that of the status register in addition to providing handshake signals.

CONTROL LOGIC

The control section has six lines. Their functions and connections are as follows:

- **RD (Read):** This control signal enables the Read operation. When the signal is low, the MPU reads data from a selected I/O port of the 8255A.
- **WR (Write):** This control signal enables the Write operation. When the signal goes low, the MPU writes into a selected I/O port or the control register.

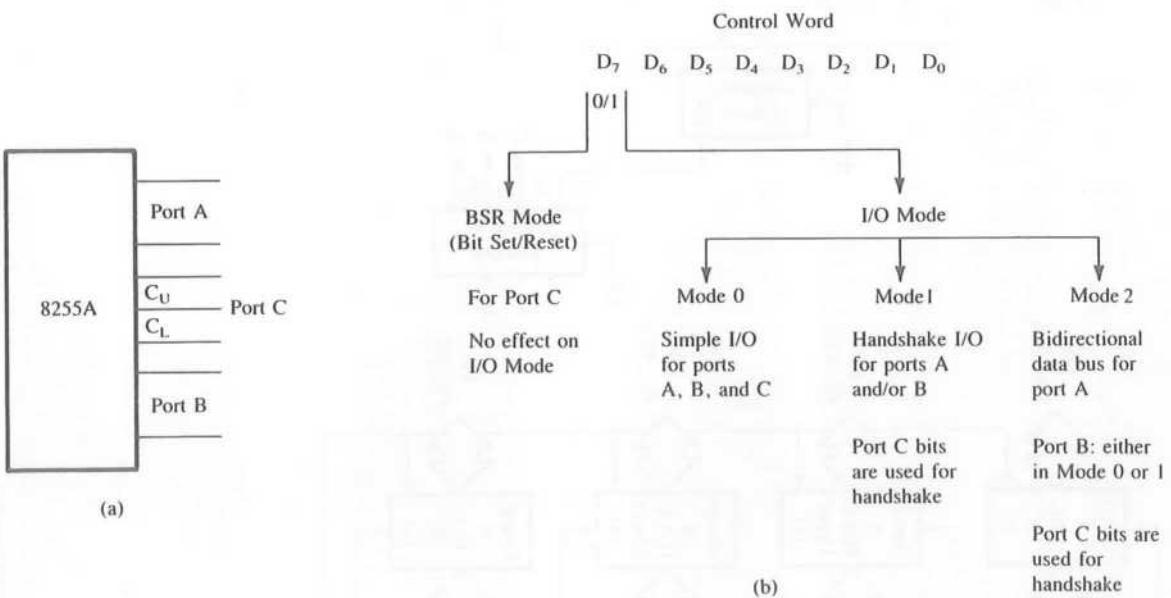


FIGURE 15.1
8255A I/O Ports (a) and Their Modes (b)

- **RESET (Reset):** This is an active high signal; it clears the control register and sets all ports in the input mode.
- **CS, A_0 , and A_1 :** These are device select signals. \overline{CS} is connected to a decoded address, and A_0 and A_1 are generally connected to MPU address lines A_0 and A_1 , respectively.

The \overline{CS} signal is the master Chip Select, and A_0 and A_1 specify one of the I/O ports or the control register as given below:

\overline{CS}	A_1	A_0	Selected
0	0	0	Port A
0	0	1	Port B
0	1	0	Port C
0	1	1	Control Register
1	X	X	8255A is not selected.

As an example, the port addresses in Figure 15.3(a) are determined by the \overline{CS} , A_0 , and A_1 lines. The CS line goes low when $A_7 = 1$ and A_6 through A_2 are at logic 0. When these signals are combined with A_0 and A_1 , the port addresses range from 80H to 83H, as shown in Figure 15.3(b).

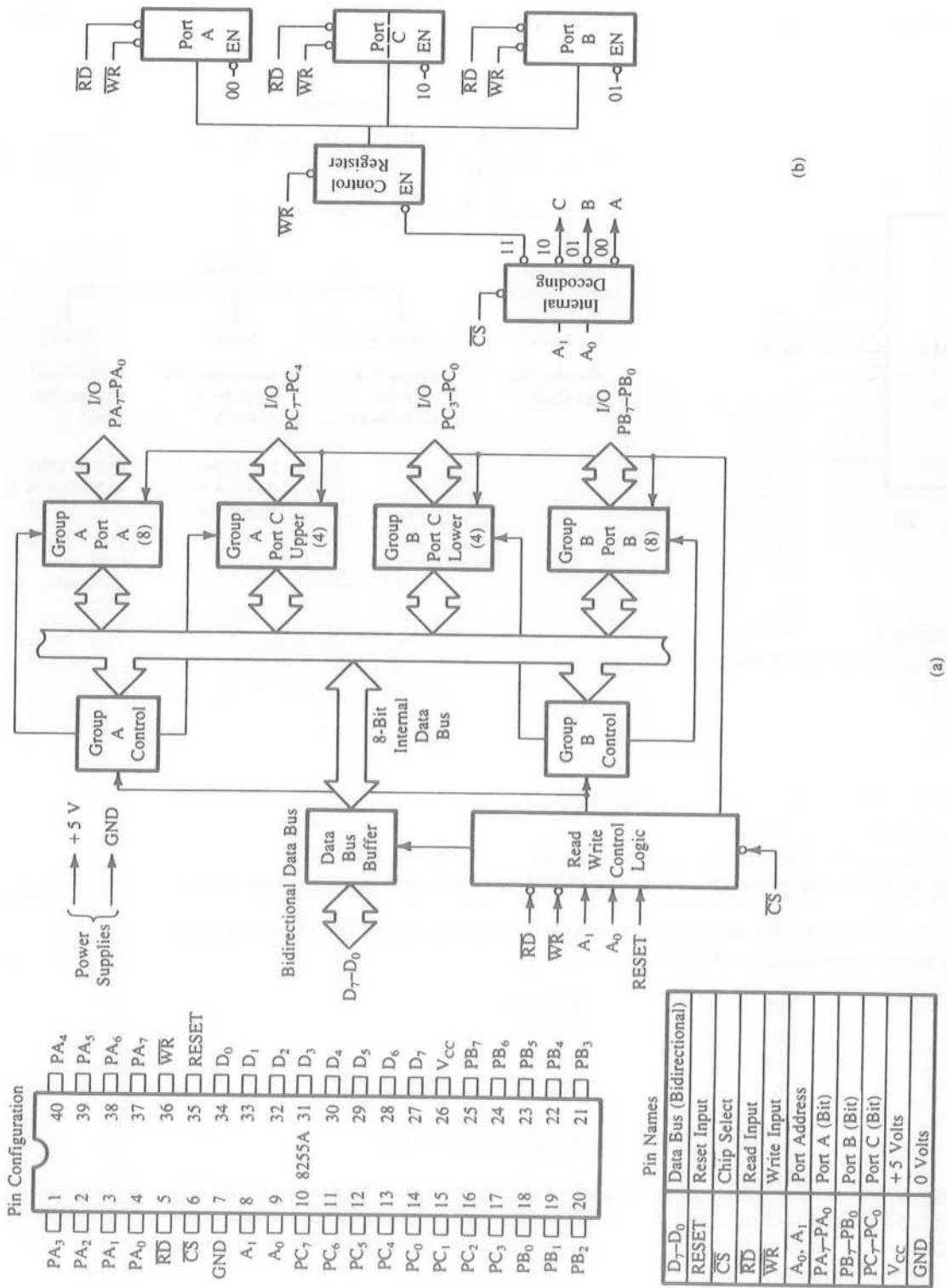


FIGURE 15.2

8255A Block Diagram (a) and an Expanded Version of the Control Logic and I/O Ports (b)
SOURCE: A: Intel Corporation, *Peripheral Components* (Santa Clara, Calif.: Author, 1993), p. 3-100.

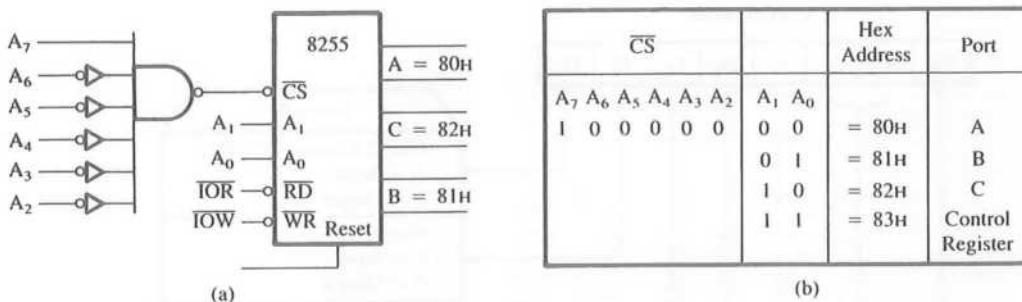


FIGURE 15.3
8255A Chip Select Logic (a) and I/O Port Addresses (b)

CONTROL WORD

Figure 15.2(b) shows a register called the **control register**. The contents of this register, called the **control word**, specify an I/O function for each port. This register can be accessed to write a control word when A₀ and A₁ are at logic 1, as mentioned previously. The register is not accessible for a Read operation.

Bit D₇ of the control register specifies either the I/O function or the Bit Set/Reset function, as classified in Figure 15.1(b). If bit D₇ = 1, bits D₆–D₀ determine I/O functions in various modes, as shown in Figure 15.4. If bit D₇ = 0, port C operates in the Bit Set/Reset (BSR) mode. The BSR control word does not affect the functions of ports A and B (the BSR mode will be described later).

To communicate with peripherals through the 8255A, three steps are necessary:

1. Determine the addresses of ports A, B, and C and of the control register according to the Chip Select logic and address lines A₀ and A₁.
2. Write a control word in the control register.
3. Write I/O instructions to communicate with peripherals through ports A, B, and C.

Examples of the various modes are given in the next section.

15.1.2 Mode 0: Simple Input or Output

In this mode, ports A and B are used as two simple 8-bit I/O ports and port C as two 4-bit ports. Each port (or half-port, in case of C) can be programmed to function as simply an input port or an output port. The input/output features in Mode 0 are as follows:

1. Outputs are latched.
2. Inputs are not latched.
3. Ports do not have handshake or interrupt capability.

-
1. Identify the port addresses in Figure 15.5.
 2. Identify the Mode 0 control word to configure port A and port C_U as output ports and port B and port C_L as input ports.

Example
15.1

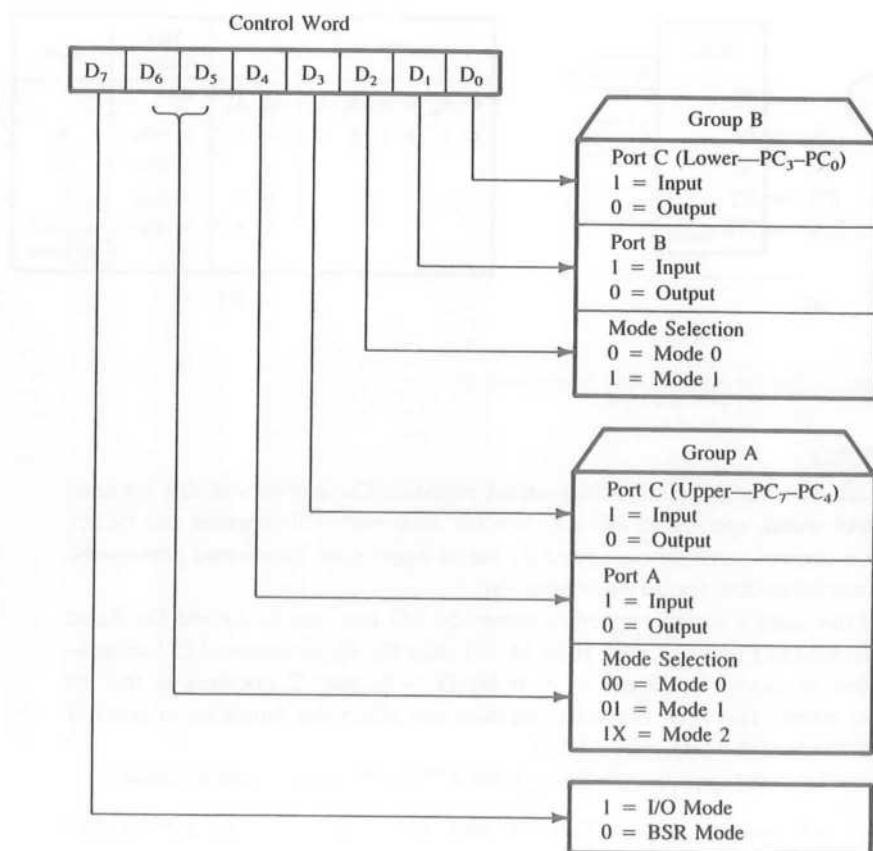


FIGURE 15.4

8255A Control Word Format for I/O Mode

SOURCE: Adapted from Intel Corporation, *Peripheral Components* (Santa Clara, Calif.: Author, 1993), p. 3-104.

3. Write a program to read the DIP switches and display the reading from port B at port A and from port C₁ at port C_H.

Solution

- 1. Port Addresses** This is a memory-mapped I/O; when the address line A_{15} is high, the Chip Select line is enabled. Assuming all don't care lines are at logic 0, the port addresses are as follows:

Port A	=	8000H ($A_1 = 0, A_0 = 0$)
Port B	=	8001H ($A_1 = 0, A_0 = 1$)
Port C	=	8002H ($A_1 = 1, A_0 = 0$)
Control Register	=	8003H ($A_1 = 1, A_0 = 1$)

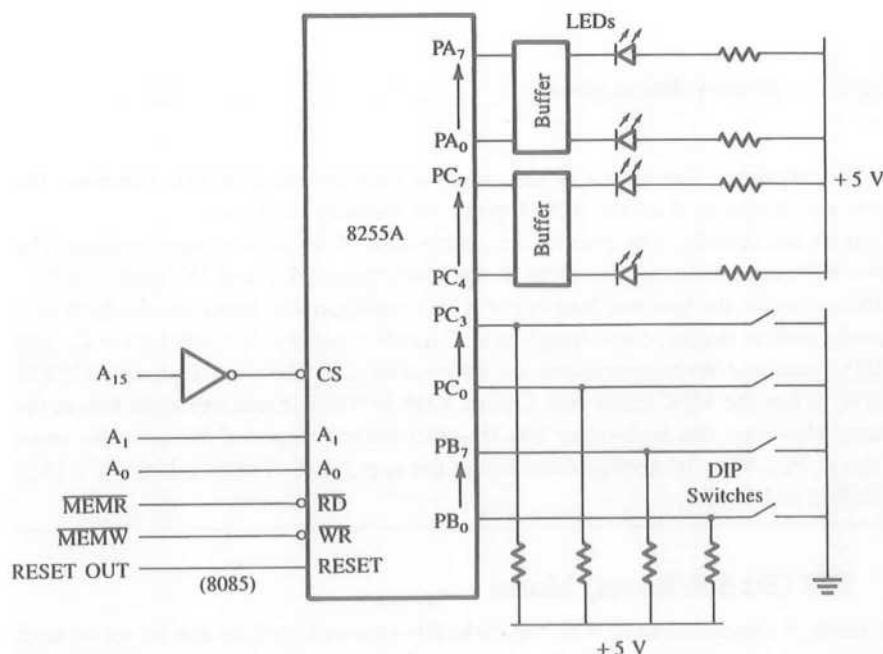
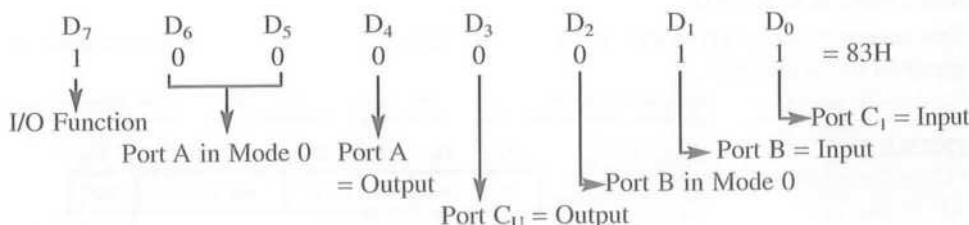


FIGURE 15.5
Interfacing 8255A I/O Ports in Mode 0

2. Control Word



3. Program

```

MVI A,83H      ;Load accumulator with the control word
STA 8003H      ;Write word in the control register to initialize the ports
LDA 8001H      ;Read switches at port B
STA 8000H      ;Display the reading at port A
LDA 8002H      ;Read switches at port C
ANI 0FH        ;Mask the upper four bits of port C; these bits are not input data
RLC            ;Rotate and place data in the upper half of the accumulator
RLC

```

```

RLC
RLC
STA 8002H      ;Display data at port CU
HLT

```

Program Description The circuit is designed for memory-mapped I/O; therefore, the instructions are written as if all the 8255A ports are memory locations.

The ports are initialized by placing the control word 83H in the control register. The instructions STA and LDA are equivalent to the instructions OUT and IN, respectively.

In this example, the low four bits of port C are configured as input and the high four bits are configured as output; even though port C has one address for both halves C_U and C_L (8002H), Read and Write operations are differentiated by the control signals MEMR and MEMW. When the MPU reads port C (e.g., LDA 8002H), it receives eight bits in the accumulator. However, the high-order bits (D_7 - D_4) must be ignored because the input data bits are in PC_3 - PC_0 . To display these bits at the upper half of port C, bits (PC_3 - PC_0) must be shifted to PC_7 - PC_4 .

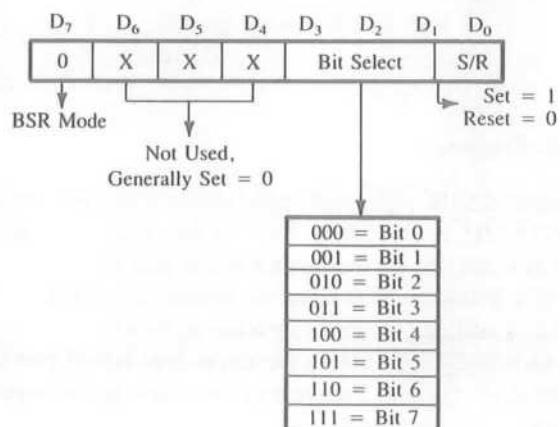
15.1.3 BSR (Bit Set/Reset) Mode

The BSR mode is concerned only with the eight bits of port C, which can be set or reset by writing an appropriate control word in the control register. A control word with bit $D_7 = 0$ is recognized as a BSR control word, and it does not alter any previously transmitted control word with bit $D_7 = 1$; thus the I/O operations of ports A and B are not affected by a BSR control word. In the BSR mode, individual bits of port C can be used for applications such as an on/off switch.

BSR CONTROL WORD

This control word, when written in the control register, sets or resets one bit at a time, as specified in Figure 15.6.

FIGURE 15.6
8255A Control Word Format in the
BSR Mode



Write a BSR control word subroutine to set bits PC₇ and PC₃ and reset them after 10 ms.
Use the schematic in Figure 15.3 and assume that a delay subroutine is available.

Example
15.2

BSR CONTROL WORDS

Solution

	D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀	=	
To set bit PC ₇	=	0	0	0	1	1	1	1	=	0FH
To reset bit PC ₇	=	0	0	0	0	1	1	0	=	0EH
To set bit PC ₃	=	0	0	0	0	0	1	1	=	07H
To reset bit PC ₃	=	0	0	0	0	0	1	1	=	06H

PART ADDRESS

Control register address = 83H; refer to Figure 15.3(b).

SUBROUTINE

```

BSR:    MVI A,0FH      ;Load byte in accumulator to set PC7
        OUT 83H       ;Set PC7 = 1
        MVI A,07H      ;Load byte in accumulator to set PC3
        OUT 83H       ;Set PC3 = 1
        CALL DELAY    ;This is a 10-ms delay
        MVI A,06H      ;Load accumulator with the byte to reset PC3
        OUT 83H       ;Reset PC7
        MVI A,0EH      ;Load accumulator with the byte to reset PC7
        OUT 83H       ;Reset PC7
        RET

```

From an analysis of the above routine, the following points can be noted:

1. To set/reset bits in port C, a control word is written in the control register and not in port C.
2. A BSR control word affects only one bit in port C.
3. The BSR control word does not affect the I/O mode.

15.1.4 Illustration: Interfacing I/O Devices for the MCTS Project Using an A/D Converter

In the MCTS project (described in Chapter 1), the system is expected to read the temperature in a room, display the temperature on a liquid crystal display (LCD) panel, turn on a fan if the temperature is above a set point, and turn on a heater if the temperature is below a set point. This illustration includes the interfacing of a temperature sensor using an A/D converter and I/O devices such as a fan and a heater; the LCD display is illustrated in Chapter 17.

PROBLEM STATEMENT

1. Interface a temperature sensor using an A/D converter and port A of the 8255.
2. Interface a fan and a heater using optocouplers and triacs to drive the I/O devices.
3. Write instructions to read the temperature. If the temperature is less than 10°C, turn on the heater, and if the temperature is higher than 35°C, turn on the fan.

The following components are specified for interfacing:

1. 8255—Port A in Mode 0 and Port C in BSR mode
2. 8-bit A/D Converter—National ADC0801 and Temperature Sensor—National LM135
3. Optoisolator—MOC 3011 and Triac 2N6071

CIRCUIT ANALYSIS

Figure 15.7 shows the interfacing circuit. In this illustration, we will use the symbolic addresses for ports A, C, and Control when we write instructions. The specific port addresses can be the same as in Figure 15.3 (80H to 83H). Figure 15.7 shows that the temperature sensor LM135 is connected as input to the A/D converter, and Port A is set up as an input port for the A/D converter. Port C is set up in BSR mode. Signal line, PC₀ is set up as an input—PC₀ is connected to the INTR. Four signal lines, PC₄–PC₇, are set up as outputs. PC₄ is used to start conversions, and PC₅ and PC₆ are connected to optoisolators that are used to drive the fan and the heater, respectively, using triacs.

TEMPERATURE SENSOR—LM 135

This is a three-terminal integrated circuit temperature sensor that can operate as a two-terminal zener device, and its third terminal can be used for calibration if necessary. Its output voltage is directly proportional to absolute temperature in Kelvin; it changes 10 mV/K and operates over the temperature range from -55°C to +150°C. In Figure 15.7, the output of the sensor is connected to the input V_{IN}(+) of the A/D converter ADC0801. At 0°C, which is 273°K, the output voltage of the sensor is 2.73 V, and at 25°C room temperature, the output voltage is 2.98 V. See the data sheet in Appendix D for additional information on the LM135. Temperature sensors are also available that have outputs proportional to temperatures in Fahrenheit and Centigrade.

A/D CONVERTER (NATIONAL SEMICONDUCTOR ADC0801)

The basic concepts underlying the operation of an A/D converter are discussed in Chapter 13. In Section 13.2.4, the operation of the ADC0801 and its interfacing are described.

This is an 8-bit A/D converter that can convert analog input voltage from 0 to 5 V. It has two inputs, V_{IN}(+) and V_{IN}(-), for differential analog input. If an analog input signal has a smaller range than 0 to 5 V, this converter provides a reference voltage (pin 9) V_{REF}/2 that can adjust lower analog voltages to full resolution. The dc voltage applied to pin 9 has a gain of 2. In Figure 15.7, V_{REF}/2 is connected to 1.28 V by adjusting the 10K pot; therefore, the full-scale range is calibrated for 0 to 2.56 V, and the output of the

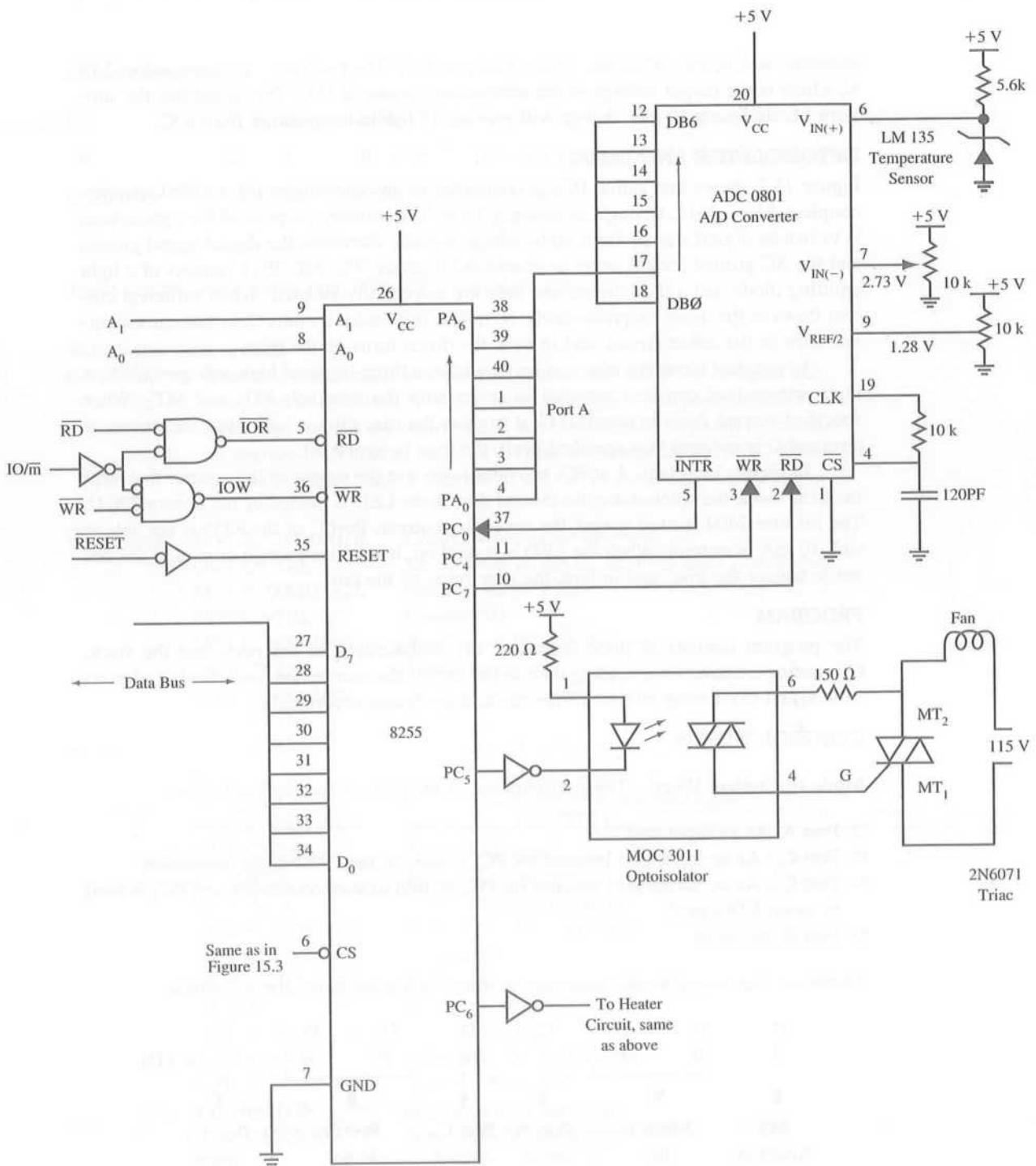


FIGURE 18.7
I/O Schematic for MCTS

```

MVI A, 00001011 ;Word to set PC5 to turn on the fan
OUT CNTRL ;Turn on the fan
POP PSW ;Retrieve A and flags
RET

FANOFF: PUSH PSW ;Save A and flags
MVI A, 00001010 ;Word to reset PC5 to turn off the fan
OUT CNTRL ;Turn off the fan
POP PSW ;Retrieve A and flags
RET

```

HEATON and HEATOFF are subroutines similar to FANON and FANOFF except that they turn on/off the bit PC₆.

PROGRAM DESCRIPTION

The subroutine initializes port A and the lower half of port C as inputs and the upper half of Port C as outputs. It disables RD and asserts WR signals. When WR goes low, the conversion begins and the INTR goes high. When the conversion is complete, the INTR goes low, indicating that the data byte is ready for reading (see Figure 13.13). The INTR signal can be read as input in a loop or it can be used to interrupt the processor.

The subroutine reads port C and places the status of PC₀ in the carry flag by rotating the bit right. If the CY flag is logic 1, the subroutine goes back to the label READ and reads Port C again. The subroutine stays in the loop until PC₀ is at logic zero. Once PC₀ goes low, the processor asserts RD low and reads the data byte. This data byte is already hardware calibrated to be read in degrees Centigrade. The subroutine compares the reading in A with the high set point (35°C). If the comparison does not generate a carry, it means the reading in A is higher than 35°C, and it calls a subroutine FANON to turn on the fan. If it generates a carry, it means the reading in A is lower than 35°C, and it turns off the fan by calling the subroutine FANOFF. However, the subroutine does not check whether a fan is already off or on; in some instances this may be a redundant action. Next it compares the reading with the low set point (10°C). If the reading is lower than 10°C, it calls the subroutine HEATON and if it is higher than 10°C, it calls the subroutine HEATOFF. The subroutines FANON, FANOFF, HEATON, and HEATOFF turn on/off bits PC₅ and PC₆ connected to the fan and the heater.

15.1.5 Mode 1: Input or Output with Handshake

In Mode 1, handshake signals are exchanged between the MPU and peripherals prior to data transfer. The features of this mode include the following:

1. Two ports (A and B) function as 8-bit I/O ports. They can be configured either as input or output ports.
2. Each port uses three lines from port C as handshake signals. The remaining two lines of port C can be used for simple I/O functions.
3. Input and output data are latched.
4. Interrupt logic is supported.

In the 8255A, the specific lines from port C used for handshake signals vary according to the I/O function of a port. Therefore, input and output functions in Mode 1 are discussed separately.

MODE 1: INPUT CONTROL SIGNALS

Figure 15.8(a) shows the associated control signals used for handshaking when ports A and B are configured as input ports. Port A uses the upper three signals: PC₃, PC₄, and PC₅. Port B uses the lower three signals: PC₂, PC₁, and PC₀. The functions of these signals are as follows:

- **STB (Strobe Input):** This signal (active low) is generated by a peripheral device to indicate that it has transmitted a byte of data. The 8255A, in response to STB, generates IBF and INTR, as shown in Figure 15.9.
- **IBF (Input Buffer Full):** This signal is an acknowledgment by the 8255A to indicate that the input latch has received the data byte. This is reset when the MPU reads the data (Figure 15.9).
- **INTR (Interrupt Request):** This is an output signal that may be used to interrupt the MPU. This signal is generated if STB, IBF, and INTE (Internal flip-flop) are all at logic 1. This is reset by the falling edge of the RD signal (Figure 15.9).

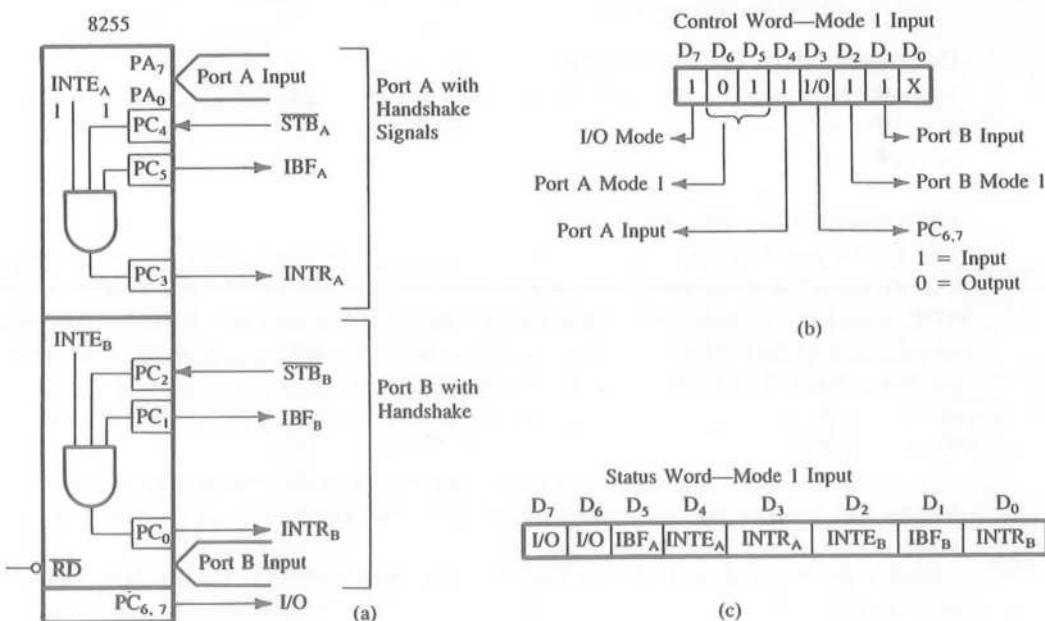


FIGURE 15.8
8255A Mode 1: Input Configuration

SOURCE: Adapted from Intel Corporation, *Peripheral Components* (Santa Clara, Calif.: Author, 1993), p. 3-110.

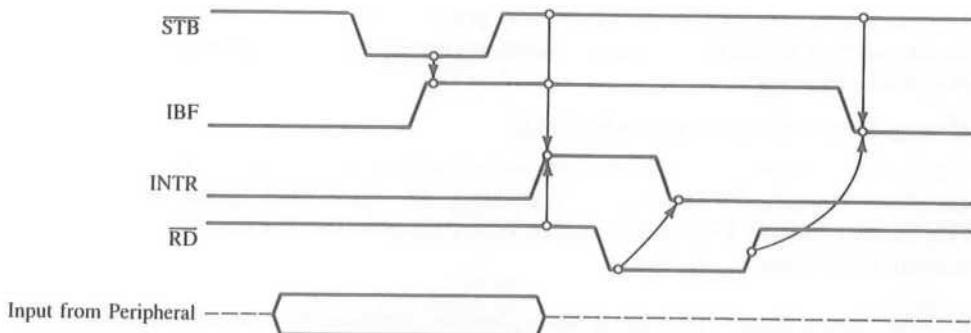


FIGURE 15.9

8255A Mode 1: Timing Waveforms for Strobed Input (with Handshake)

SOURCE: Adapted from Intel Corporation, *Peripheral Components* (Santa Clara, Calif.: Author, 1993), p. 3-110.

- **INTE (Interrupt Enable):** This is an internal flip-flop used to enable or disable the generation of the INTR signal. The two flip-flops INTE_A and INTE_B are set/reset using the BSR mode. The INTE_A is enabled or disabled through PC_4 , and INTE_B is enabled or disabled through PC_2 .

CONTROL AND STATUS WORDS

Figure 15.8(b) uses control words derived from Figure 15.4 to set up port A and port B as input ports in Mode 1. Similarly, Figure 15.8(c) also shows the status word, which will be placed in the accumulator if port C is read.

PROGRAMMING THE 8255A IN MODE 1

The 8255A can be programmed to function using either status check I/O or interrupt I/O. Figure 15.10(a) shows a flowchart for the status check I/O. In this flowchart, the MPU continues to check data status through the IBF line until it goes high. This is a simplified flowchart; however, it does not show how to handle data transfer if two ports are being used. The technique is similar to that of Mode 0 combined with the BSR mode. The disadvantage of the status check I/O with handshake is that the MPU is tied up in the loop.

The flowchart in Figure 15.10(b) shows the steps required for the interrupt I/O, assuming that vectored interrupts are available. The confusing step in the interrupt I/O is to set INTE either for port A or port B. Figure 15.8(a) shows that the STB signal is connected to pin PC_4 and the INTE_A is also controlled by the pin PC_4 . (In port B, pin PC_2 is used for the same purposes.) However, the INTE_A is set or reset in the BSR mode and the BSR control word has no effect when ports A and B are set in Mode 1.

In case the INTR line is used to implement the interrupt, it may be necessary to read the status of INTR_A and INTR_B to identify the port requesting an interrupt service and to determine the priority through software, if necessary.

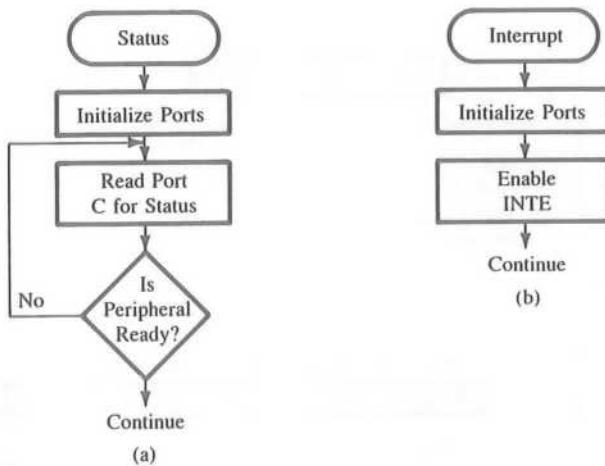


FIGURE 15.10

Flowcharts: Status Check I/O (a) and Interrupt I/O (b)

MODE 1: OUTPUT CONTROL SIGNALS

Figure 15.11 shows the control signals when ports A and B are configured as output ports. These signals are defined as follows:

- OBF (Output Buffer Full):** This is an output signal that goes low when the MPU writes data into the output latch of the 8255A. This signal indicates to an output peripheral that new data are ready to be read (Figure 15.12). It goes high again after the 8255A receives an ACK from the peripheral.
- ACK (Acknowledge):** This is an input signal from a peripheral that must output a low when the peripheral receives the data from the 8255A ports (Figure 15.12).
- INTR (Interrupt Request):** This is an output signal, and it is set by the rising edge of the ACK signal. This signal can be used to interrupt the MPU to request the next data byte for output. The INTR is set when OBF, ACK, and INTE are all one (Figure 15.12) and reset by the falling edge of WR.
- INTE (Interrupt Enable):** This is an internal flip-flop to a port and needs to be set to generate the INTR signal. The two flip-flops INTE_A and INTE_B are controlled by bits PC_6 and PC_2 , respectively, through the BSR mode.
- PC_{4,5}:** These two lines can be set up either as input or output.

CONTROL AND STATUS WORDS

Figure 15.11(b) shows the control word used to set up ports A and B as output ports in Mode 1. Similarly, Figure 15.11(c) also shows the status word, which will be placed in the accumulator if port C is read.

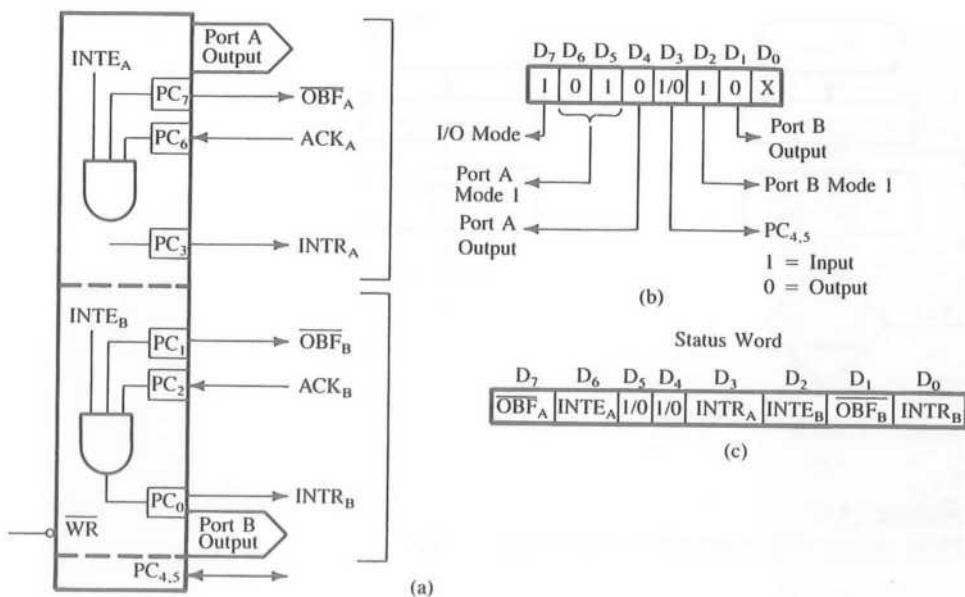


FIGURE 15.11

8255A Mode 1: Output Configuration

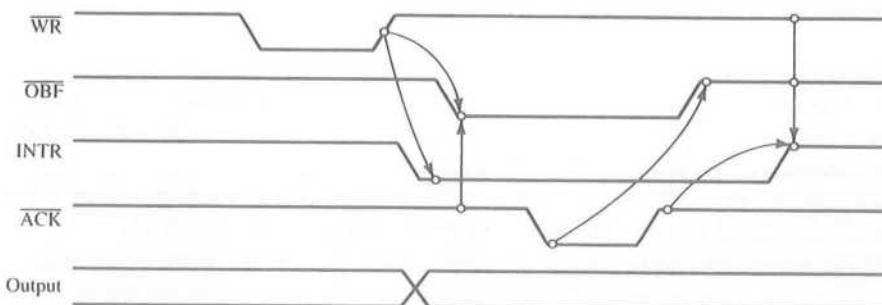
SOURCE: Adapted from Intel Corporation, *Peripheral Components* (Santa Clara, Calif.: Author, 1993), p. 3-111.

FIGURE 15.12

8255 Mode 1: Timing Waveforms for Strobed (with Handshake) Output

SOURCE: Adapted from Intel Corporation, *Peripheral Components* (Santa Clara, Calif.: Author, 1993), p. 3-111.

15.1.6 Illustration: An Application of the 8255A in the Handshake Mode (Mode 1)

PROBLEM STATEMENT

Figure 15.13 shows an interfacing circuit using the 8255A in Mode 1. Port A is designed as the input port for a keyboard with interrupt I/O, and port B is designed as the output port for a printer with status check I/O.

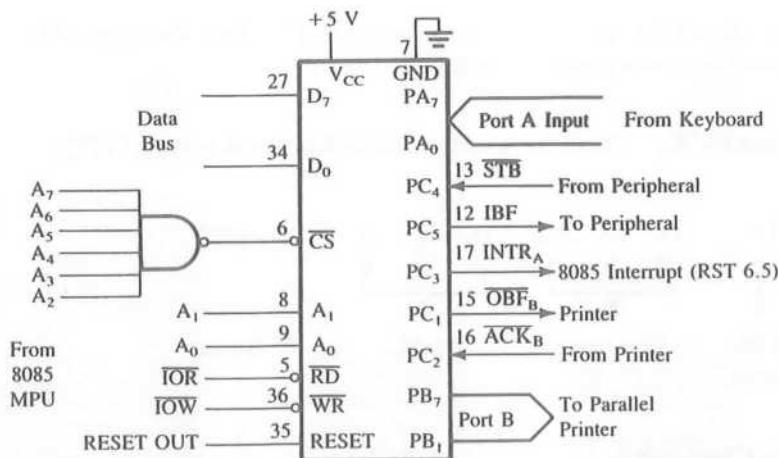


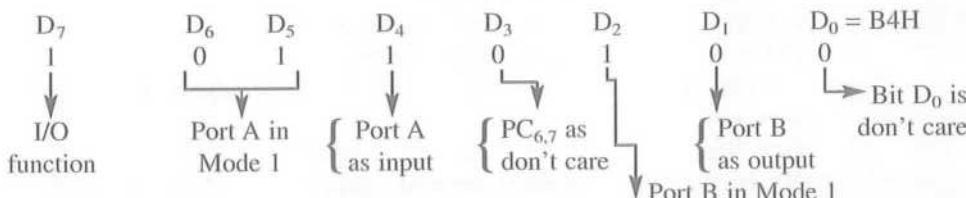
FIGURE 15.13
Interfacing the 8255A in Mode 1 (Strobed Input/Output)

1. Find port addresses by analyzing the decode logic.
2. Determine the control word to set up port A as input and port B as output in Mode 1.
3. Determine the BSR word to enable INTR_A (port A).
4. Determine the masking byte to verify the $\overline{\text{OBF}}_B$ line in the status check I/O (port B).
5. Write initialization instructions and a printer subroutine to output characters that are stored in memory.

1. Port Addresses The 8255A is connected as peripheral I/O. When the address lines A_7 – A_2 are all 1, the output of the NAND gate goes low and selects the 8255A. The individual ports are selected as follows:

$$\begin{aligned}
 \text{Port A} &= \text{FCH} \quad (A_1 = 0, A_0 = 0) \\
 \text{Port B} &= \text{FDH} \quad (A_1 = 0, A_0 = 1) \\
 \text{Port C} &= \text{FEH} \quad (A_1 = 1, A_0 = 0) \\
 \text{Control Register} &= \text{FFH} \quad (A_1 = 1, A_0 = 1)
 \end{aligned}$$

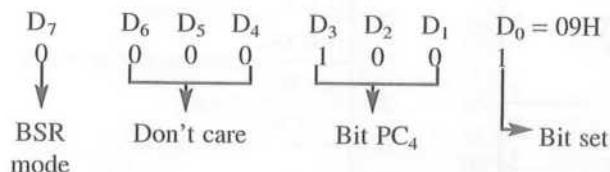
2. Control Word to Set Up Port A as Input and Port B as Output in Mode 1



In the above control word, all bits are self-explanatory, and bits D₃ and D₀ are in a don't care logic state. To generate interrupt signal INTR_A , flip-flop INTE_A should be set to 1, which can be accomplished by using the BSR Mode to set PC₄.

The output to the printer (port B) is status-controlled. Therefore, the status of line $\overline{\text{OBF}_B}$ can be checked by reading bit D_1 of port C_L .

3. BSR Word to Set INTE_A To set the Interrupt Enable flip-flop of port A (INTE_A), bit PC_4 should be 1.



4. Status Word to Check $\overline{\text{OBF}_B}$

D_7 X	D_6 X	D_5 X	D_4 X	D_3 X	D_2 X	D_1 $\overline{\text{OBF}_B}$	D_0 X
------------	------------	------------	------------	------------	------------	------------------------------------	------------

Masking byte: 02H

5. Initialization Program

MVI A,B4H	;Word to initialize port A as input, ; port B as output in Mode 1
OUT FFH	
MVI A,09H	;Set INTE_A (PC_4)
OUT FFH	;Using BSR Mode
EI	;Enable interrupts
CALL PRINT	;Continue other tasks

Print Subroutine

PRINT:	LXI H, MEM	;Point index to location of stored characters
	MVI B,COUNT	;Number of characters to be printed
NEXT:	MOV A,M	;Get character from memory
	MOV C,A	;Save character
STATUS:	IN FEH	;Read port C for status of $\overline{\text{OBF}}$
	ANI 02H	;Mask all bits except D_1
	JZ STATUS	;If it is low, the printer is not ready; wait in the loop
	MOV A,C	
	OUT FDH	;Send a character to port B
	INX H	;Point to the next character
	DCR B	
	JNZ NEXT	
	RET	

PROGRAM DESCRIPTION

This I/O design using the 8255A in Mode 1 allows two operations: outputting to the printer and data entry through the keyboard. The printer interfacing is designed with the status check and the keyboard interfacing with the interrupt.

In the PRINT subroutine, the character is placed in the accumulator, and the status is read by the instruction IN FEH. Initially, port B is empty, bit PC₁ ($\overline{\text{OBF}}_B$) is high, and the instruction OUT FDH sends the first character to port B. The rising edge of the WR signal sets signal OBF low, indicating the presence of a data byte in port B, which is sent out to the printer (Figure 15.12). After receiving a character, the printer sends back an acknowledge signal (ACK), which in turn sets OBF_B high, indicating that port B is ready for the next character, and the PRINT subroutine continues.

If a key is pressed during the PRINT, a data byte is transmitted to port A and the STB_A goes low, which sets IBF_A high. The initialization routine should set the INTE_A flip-flop. When the STB_A goes high, all the conditions (i.e., IBF_A = 1, INTE_A = 1) to generate INTR_A are met. This signal, which is connected to the RST 6.5, interrupts the MPU, and the program control is transferred to the service routine. This service routine would read the contents of port A, enable the interrupts, and return to the PRINT routine (the interrupt service routine is not shown here).

15.1.7 Mode 2: Bidirectional Data Transfer

This mode is used primarily in applications such as data transfer between two computers or floppy disk controller interface. In this mode, port A can be configured as the bidirectional port and port B either in Mode 0 or Mode 1. Port A uses five signals from port C as handshake signals for data transfer. The remaining three signals from port C can be used either as simple I/O or as handshake for port B. Figure 15.14 shows two configurations of Mode 2. This mode is illustrated in Section 15.3.

ILLUSTRATION: INTERFACING KEYBOARD AND SEVEN-SEGMENT DISPLAY

15.2

This illustration is concerned with interfacing a pushbutton keyboard and a seven-segment LED display using the 8255A. The emphasis in this illustration is not particularly on the features of 8255A but on how to integrate hardware and software. When a key is pressed, the binary reading of the key has almost no relationship to what it represents. Similarly, to display a number at a seven-segment LED, the binary value of the number needs to be converted into the seven-segment code, which is primarily decided by the hardware consideration. This illustration demonstrates how the microprocessor monitors the changes in hardware reading and converts into appropriate binary reading using its instruction set.

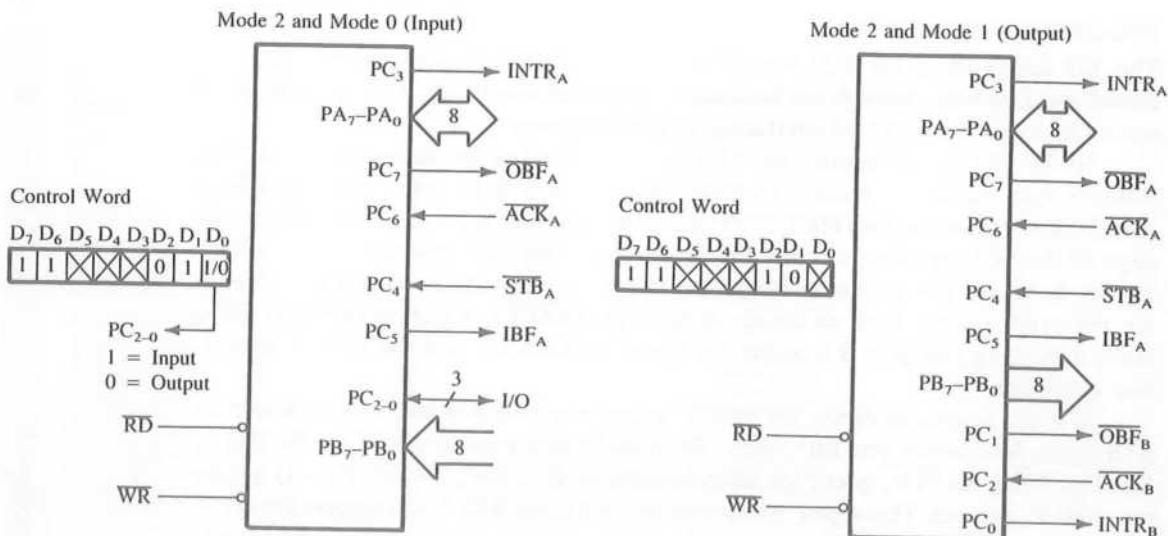


FIGURE 15.14
8255A Mode 2: Bidirectional Input/Output

SOURCE: Intel Corporation, *Peripheral Components* (Santa Clara, Calif.: Author, 1993), p. 3-113.

15.2.1 Problem Statement

A pushbutton keyboard is connected to port A and a seven-segment LED is connected to port B of the 8255A, as shown in Figure 15.15. Port A should be configured as an input port and port B as an output port; this is a simple I/O configuration in Mode 0 without the use of handshake signals or the interrupt.

Write a program to monitor the keyboard to sense a key pressed and display the number of the key at the seven-segment LED. For example, when the key K₇ is pressed, the digit 7 should be displayed at port B.

15.2.2 Problem Analysis

In this problem, the address decoding circuit is the same as in Figure 15.13; therefore, the port addresses range from FCH to FFH. The keyboard circuit shown in Figure 15.15 is similar to that in Figure 4.11 except that the DIP switches are replaced by pushbutton keys and the buffer is replaced by port A of the 8255A. When a pushbutton key is pressed, it bounces (makes and breaks contact) a few times before it makes a firm contact. To prevent multiple readings of the same key, it is necessary to debounce the key. The hardware solution to this problem is to use a key debounce circuit (cross-coupled NAND or NOR gates), and the software solution is to wait for 10 to 20 ms until the key is settled and then check the key again. The display circuit in Figure 15.15 uses a common-cathode seven-segment LED, connected to port B of the 8255A. To display a digit, it is necessary to turn on the appropriate segments of the LED. The appropriate binary code can be obtained by using the table look-up technique, described in Section 15.2.4. The programming of this problem can be divided into the following categories:

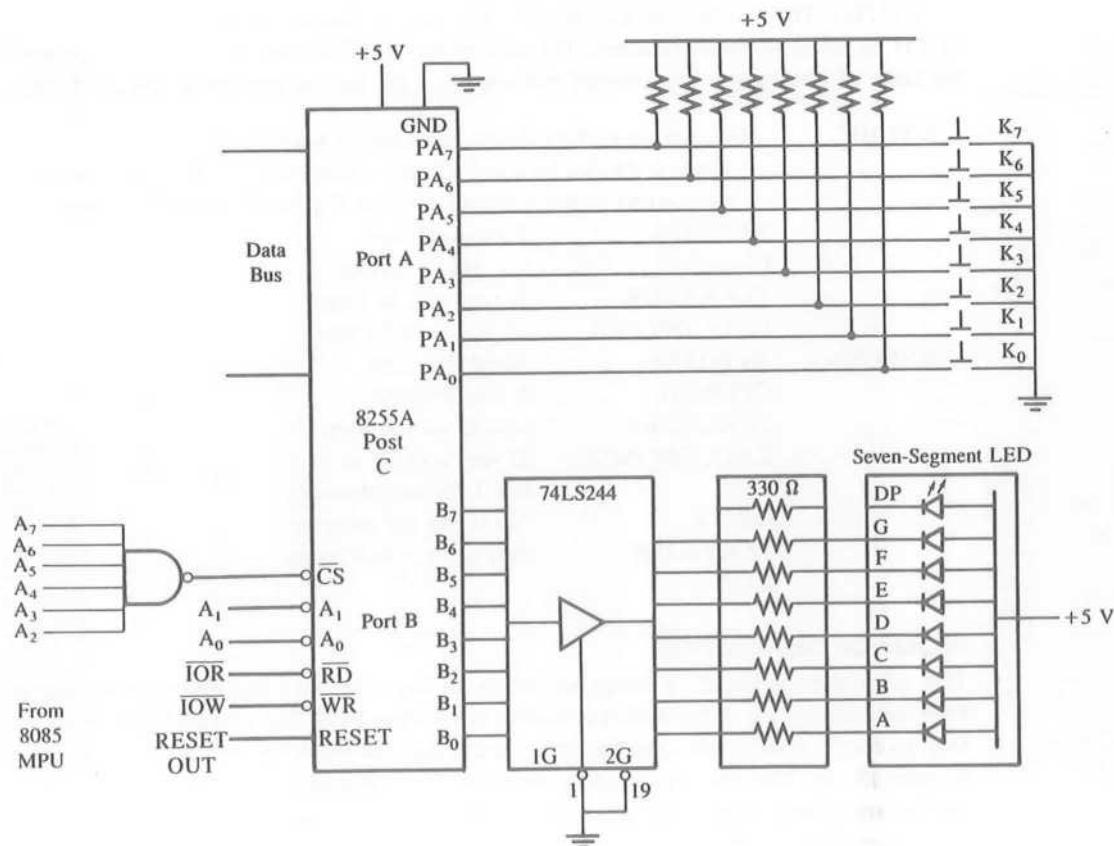


FIGURE 15.15
Interfacing a Keyboard and a Seven-Segment LED

1. Check if a key is pressed.
2. Debounce the key.
3. Identify and encode the key in appropriate binary format.
4. Obtain the seven-segment code and display it.

The instructions for these steps can be written in separate modules, as shown in the next section.

15.2.3 Keyboard

The keys K₇–K₀ are tied high through 10 kΩ resistors, and when a key is pressed, the corresponding line is grounded. When all keys are open and if the 8085 reads port A, the reading on the data bus will be FFH. When any key is pressed, the reading will be less than FFH. For example, if K₇ is pressed, the output of port A will be 0111

1111 (7FH). This reading should be encoded into the binary equivalent of the digit 7 (0111) by using software routines. The subroutines KYCHK and KYCODE accomplish the tasks of checking a key pressed and encoding the key in appropriate binary format.

KYCHK:	;This subroutine first checks whether all keys are open.
	; Then, it checks for a key closure, debounces the key, and places
	; the reading in the accumulator. See Figure 15.16 for flowchart.
	IN PORTA ;Read keyboard
	CPI 0FFH ;Are all keys open?
	JNZ KYCHK ;If not, wait in loop
	CALL DBONCE ;If yes, wait 20 ms
KYPUSH:	IN PORTA ;Read keyboard
	CPI 0FFH ;Is key pressed?
	JZ KYPUSH ;If not, wait in loop
	CALL DBONCE ;If yes, wait 20 ms
	CMA ;Set 1 for key closure
	ORA A ;Set 0 flag for an error
	JZ KYPUSH ;It is error, check again
	RET

PROGRAM DESCRIPTION

This subroutine is based on hardware; when all keys are open the keyboard reading is FFH, and when a key is pressed, the reading is less than FFH. The routine begins with the loop to check whether all keys are open, and it stays in the loop until all keys are open (Figure 15.16). This prevents reading the same key repeatedly if someone were to hold the key for a long time. When the routine finds that a key has been released, it waits for 20 ms for a key debounce.

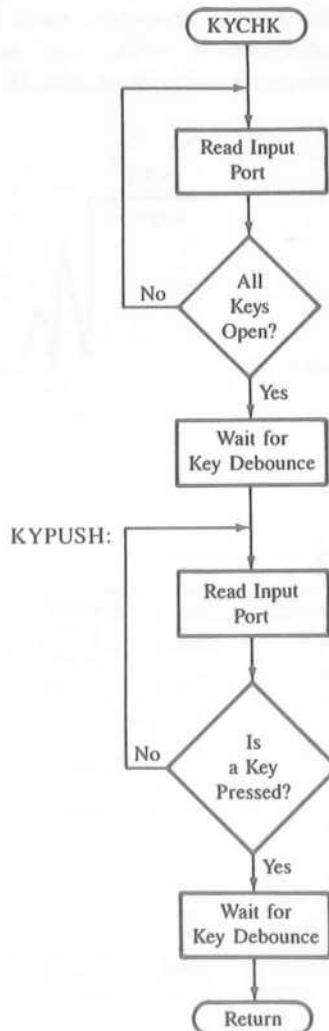
The loop starting at KYPUSH (Figure 15.16) checks whether a key is pressed. When a key is pressed, the reading is less than FFH; thus, the compare instruction does not set the Z flag and the program goes to the next instruction for a key debounce. The CMA instruction complements the accumulator reading; thus, the reading of the key pressed is set to 1, and other bits are set to 0. The next two instructions check for an error. If it is a momentary contact (false alarm), all bits will be 0s. The ORA instruction sets the Z flag, and the Jump instruction takes the program back to checking keys.

KYCODE:	;This routine converts (encodes) the binary hardware reading of the key
	; pressed into appropriate binary format according to the number of the
	; key.
	MVI C,08H ;Set code encounter
NEXT:	DCR C ;Adjust key code
	RAL ;Place MSB in CY
	JNC NEXT ;If bit = 0, go back to check next bit
	MOV A,C ;Place key code in the accumulator
	RET

PROGRAM DESCRIPTION

Conceptually, this is an important routine; it establishes the relationship between the hardware and the number of a key. For example, if key K₇ is pressed, the reading from the routine KYCHK in the accumulator will be 1000 0000 (the reading is already complemented). The KYCODE routine sets register C for the count of eight and immediately decrements the count to seven. The instruction RAL places bit D₇ in the CY flag, and the next instruction checks for the CY flag. If it is set, the key K₇ must be pressed, and the key code (digit 7) is

FIGURE 15.16
Flowchart: Key Check Subroutine



in register C. If CY = 0, the program loops back to check the next bit (D_6). The loop is repeated until 1 is found in CY, and at every iteration of the loop the key code in register C is adjusted for the next key. If more than one key is pressed, this routine ignores the low-order key. Finally, the subroutine places the key code in the accumulator and returns.

KEY DEBOUNCE

When a mechanical pushbutton key, shown in Figure 15.17(a), is pressed or released, the metal contacts of the key momentarily bounce before giving a steady-state reading, as shown in Figure 15.17(b). Therefore, it is necessary that the bouncing of the key should not be read as an input. The key bounce can be eliminated from input data by the **key-debounce technique**, using either hardware or software.

Figure 15.17(c) shows a key debounce circuit. In this circuit, the outputs of the NAND gates do not change even if the key is released from position A₁. The outputs change when the key makes a contact with position B₁. When the key is connected to A₁,

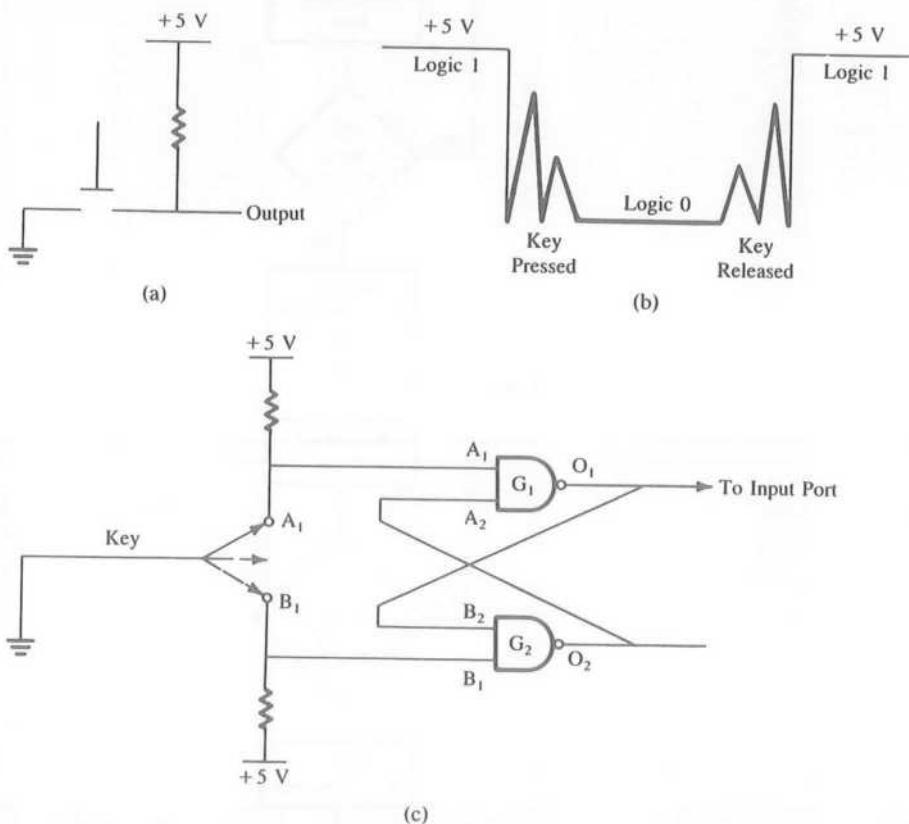


FIGURE 15.17

Pushbutton Key (a), Key Bounce (b), and Key Debounce Circuit Using NAND Gates (c)

A_1 goes low. If one of the inputs to gate G_1 is low, the output O_1 becomes 1, which makes B_2 high. Because line B_1 is already high, the output of O_2 goes low, which makes A_2 low. When the key connection is released from A_1 , it goes high, but because A_2 is low the output doesn't change. When the key makes contact with B_1 , the outputs change. This means when the key goes from one contact (+5 V) to another contact (ground), the output does not change during the transition period, thus eliminating multiple readings.

In the software technique, when a key closure is found, the microprocessor waits for 10 to 20 ms before it accepts the key as an input. The delay routine is as follows:

```

DBONCE: ;This is a 20 ms delay routine
        ;The delay COUNT should be calculated based on system frequency
        ;This does not destroy any register contents
        ;Input and Output = None
        PUSH B          ;Save register contents
        PUSH PSW
        LXI B,COUNT    ;Load delay count
LOOP:   DCX B          ;Next count
        MOV A,C
        ORA B          ;Set Z flag if (BC) = 0
        JNZ LOOP
        POP PSW         ;Restore register contents
        POP BC
        RET

```

PROGRAM DESCRIPTION

This is a simple delay routine similar to the delay routines discussed in Chapter 8. The first instruction loads the BC register with a 16-bit number, and the loop is repeated until $BC = 0$. In this routine, the 16-bit number (COUNT) should be calculated on the basis of the clock frequency of the system and the T-states in the loop (see Chapter 8 for details).

15.2.4 Seven-Segment Display

Figure 15.15 shows that a common-anode seven-segment LED is connected to port B through the driver 74LS244. The driver is necessary to increase the current capacity of port B; each LED segment requires 15–20 mA of current. The code for each Hex digit from 0 to F can be determined by examining the connections of the data lines to the segments and the logic requirements.

The driver 74LS244 (Figure 15.15) is an octal noninverting driver with tri-state output and current sinking capacity of 24 mA. It has two active low enable lines ($1G$ and $2G$), and the driver is permanently enabled by grounding these lines. In this circuit, this driver functions simply as a current amplifier; whatever logic is at port B will be at the output of the driver.

To display the number of the key pressed, a routine is necessary that will send an appropriate code to port B. The routine KYCODE supplies the binary number of the key pressed; however, there is no relationship between the binary value of a digit and its

seven-segment code. Therefore, the table look-up technique (refer to Chapter 10, Section 10.3) will have to be used to find the code for the digit supplied by the KYCODE; this is shown in the next routine, DSPLAY.

```

DSPLAY: ;This routine takes the binary number and converts into its common-
        ; anode seven-segment LED code. The codes are stored in memory
        ; sequentially, starting from the address CACODE
;Input: Binary number in accumulator
;Output: None
;Modifies contents of HL and A
LXI H,CACODE    ;Load starting address of code table in HL
ADD L           ;Add digit to low-order address in L
MOV L,A         ;Place code address in L
MOV A,M         ;Get code from memory
OUT PORTB       ;Send code to port B
RET

CACODE: ;Common-anode seven-segment codes are stored sequentially in memory
DB 40H,79H,24H,30H,19H,12H ;Codes for digits from 0 to 5
DB 02H,78H,00H,18H,08H,03H ;Codes for digits from 6 to B
DB 46H,21H,06H,0EH        ;Codes from digits from C to F

```

PROGRAM DESCRIPTION

In this routine the HL register is used as a memory pointer to code location. The digit to be displayed is in the accumulator, supplied by the routine KYCODE, and the seven-segment code is stored sequentially in memory, starting from location CACODE. The basic concept in this routine is to modify the memory pointer by adding the value of the digit to the base address and get the code location. For example, let us assume that the starting address of CACODE is 2050H and the digit 7 is in the accumulator. The code for digit 0 is in location 2050H; consequently, the code for digit 7 is in location 2057H. Therefore, to display digit 7, the routine adds the contents of the accumulator (7) to the low-order byte 50H in register L, resulting in the sum 57H. By transferring 57H in register L, the memory pointer in HL is modified to 2057H. Thus, the code for digit 7 is obtained by using this memory pointer.

15.2.5 Main Program

Now to monitor the keyboard and display the key pressed, we need to initialize the 8255A ports and combine the software modules discussed below:

```

KYBORD: ;This program initializes the 8255A ports; port A and port B in Mode 0
        ; and then calls the subroutine modules discussed
        ; previously to monitor the keyboard
PORTA EQU FCH      ;Port A address
PORTB EQU FDH      ;Port B address

```

CNTRL	EQU FFH	;Control register
CNWORD	EQU 90H	;Mode 0 control word, port A input and port B output
STACK	EQU 20AFH	;Beginning stack address
	LXI SP,STACK	
PPI:	MVI A,CNWORD	
	OUT CNTRL	;Set up port A in Mode 1
NEXTKY:	CALL KYCHK	;Check if a key is pressed
	CALL KYCODE	;Encode the key
	CALL DISPLAY	;Display key pressed
	JMP NEXTKY	;Check the next key pressed

PROGRAM DESCRIPTION

This is the main program, which involves the initialization of the 8255A and the stack pointer. The port addresses defined here are from Figure 15.13, and the address of STACK (20AFH) is shown as an illustration; it has no specific significance. Because the problem is divided into small modules, the main program consists primarily of calling these modules.

15.2.6 Comments and Alternative Approaches

The interfacing of the pushbutton keyboard and seven-segment display is a simplified illustration of industrial applications. The illustration is deliberately kept simple to emphasize the conceptual framework between hardware and software. However, as an application, it has several limitations, as follows:

1. The method of connecting the keyboard demands the number of I/O ports be in proportion to the number of keys; only eight keys can be connected to an 8-bit port. Generally, keys are connected in a matrix format (discussed in Chapter 17). For example, in the matrix format, 16 keys can be connected to one 8-bit port or 64 keys can be connected to two 8-bit ports.
2. The method of connecting a seven-segment LED needs excessive hardware, one port per seven-segment LED and a driver. Furthermore, it consumes a large amount of current (100 to 150 mA per display). To minimize hardware and power consumption, the technique of multiplexing is generally used (discussed in Chapter 17).

In this illustration, the approach is primarily software. For example, in the keyboard, the debouncing and encoding are performed by using instructions. However, nowadays, interfacing chips are available commercially that can sense a key closure, debounce the key, and encode the key. In addition, the chip can generate an interrupt signal when a key is pressed. Similarly, in the seven-segment display, the table look-up can be replaced by a decoder/driver. However, the hardware approach increases unit price. On the other hand, the software approach involves considerable labor (programming and debugging) cost. The choice is generally determined by the production volume and the total unit price.

15.3

ILLUSTRATION: BIDIRECTIONAL DATA TRANSFER BETWEEN TWO MICROCOMPUTERS

Advances in the VLSI technology make it economical to design dedicated microcomputers to perform or monitor specific tasks where the speed of data processing is less important. These dedicated microcomputers are usually controlled by a high-speed computer; this approach is called distributed data processing. The high-speed computer is known as a master and the dedicated computer is called a slave. Parallel I/O with handshake is used to transfer data between a master and a slave, and the data transfer is bidirectional. The bidirectional communication between two microcomputers can be accomplished using the 8255 PPI in Mode 2, as shown in the next illustration.

15.3.1 Problem Statement

Design an interfacing circuit to set up bidirectional data communication in the master-slave format between two 8085A microcomputers. Use the 8255A as the interfacing device with the master and a tri-state buffer with the slave microcomputer. Write necessary software to transfer a block of data from the master to the slave.

15.3.2 Problem Analysis

Figure 15.18 shows a block diagram to set up the bidirectional communication between the master and the slave MPUs. The block diagram shows two bidirectional data buses interconnected through the 8255A, which serves as a peripheral device of the master MPU. Port A of the 8255A is used for bidirectional data transfer, and four signals from port C are used for handshaking. The communication process is similar to that of Mode 1 of the 8255A. When the master MPU writes a data byte in the 8255A, the OBF signal goes low to inform the slave that a byte is available, and the slave acknowledges when it reads the byte. Similarly, two other handshake signals are used when the slave transfers a data byte to the master.

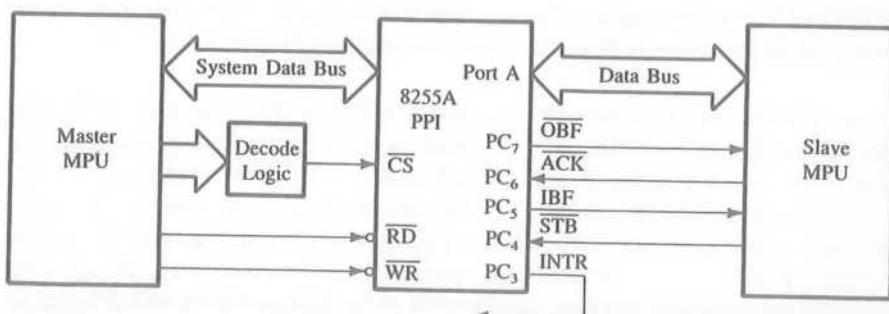


FIGURE 15.18

Block Diagram of Bidirectional Communication between Two Computers Using the 8255A

The master requires I/O ports to read and write data and to check the status of handshake signals. Similarly, the slave MPU requires I/O ports to perform Read and Write operations. Therefore, it is necessary to analyze carefully these I/O functions between the MPUs. Data transfer can be accomplished either by status check or interrupt. The speed of handling data is of more importance to the master MPU than to the slave MPU. Therefore, the master MPU is generally set up in the interrupt mode and the slave MPU in the status check mode. However, for this example, both MPUs are set up under the status check mode; the interrupt mode is left as an assignment. The data transfer operations between the two MPUs under the status check I/O are listed in the following sections.

DATA TRANSFER FROM MASTER MPU TO SLAVE MPU

1. The master MPU reads the status of OBF to verify whether the previous byte has been read by the slave MPU. This is an input function for the master MPU.
2. The master writes data into port A and the 8255A informs the slave by causing the signal OBF to go low. This is an output function for the master MPU.
3. The slave checks the OBF signal from the master for data availability. This is an input function for the slave MPU.
4. The slave MPU reads data from port A and acknowledges the reading at the same time by making the signal ACK low. This is an input function for the slave MPU.

DATA TRANSFER FROM SLAVE TO MASTER MPU

5. The slave MPU checks the handshake signal IBF (Input Buffer Full) to find out whether port A is available (empty) to transfer a data byte. This is an input function for the slave MPU.
6. The slave MPU places a data byte on the data bus and informs the 8255A by enabling the STB (Strobe) signal. This is an output function for the slave MPU.
7. The 8255A causes the IBF (Input Buffer Full) to go high, and the master MPU reads the signal to find out whether a data byte is available. This is an input function for the master MPU.
8. Finally, the master reads the data byte. This is an input function for the master MPU.

This analysis leads to certain hardware requirements that are discussed in the next section.

15.3.3 Hardware Description

In the first four steps described in the previous section, the master MPU performs one input and one output operation, and the slave MPU performs two input operations. They use two handshake signals: OBF (Output Buffer Full) and ACK (Acknowledge). Steps 5 through 8 are mirror images of the first four steps. The slave MPU performs one input and one output operation, and the master MPU performs two input operations. They use two additional handshake signals: IBF (Input Buffer Full) and STB (Strobe). These steps suggest that the master MPU and the slave MPU require three input ports and one output port each. However, if port A is a bidirectional port and port C is a status port, they will meet all the Read/Write requirements of the master MPU. Additional ports need to be designed for the slave MPU.

Figure 15.19 shows the complete schematic of the necessary ports and their decoding logic. The address bus of the master MPU is decoded by using an 8-input NAND gate, and the 8255A is selected when all lines are high, thus assigning the following port addresses:

$$\begin{aligned} \text{Control Register} &= \text{FFH} & (A_1 \text{ and } A_0 = 1) \\ \text{Port A} &= \text{FCH} & (A_1 \text{ and } A_0 = 0) \\ \text{Port C} &= \text{FEH} & (A_1 = 1, A_0 = 0) \end{aligned}$$

Port A is configured in Mode 2 using the four signals from port C as shown in the schematic. The INTR signals are unnecessary and, therefore, are not shown. The master MPU checks the ACK and the STB signals by reading the status bits of OBF and IBF in port C.

The other two handshake signals—OBF and IBF—are tied, respectively, to bits D_7 and D_0 of the slave data bus through a tri-state buffer so that they can be read by the slave MPU. The decode logic for three input ports and one output port is generated by using

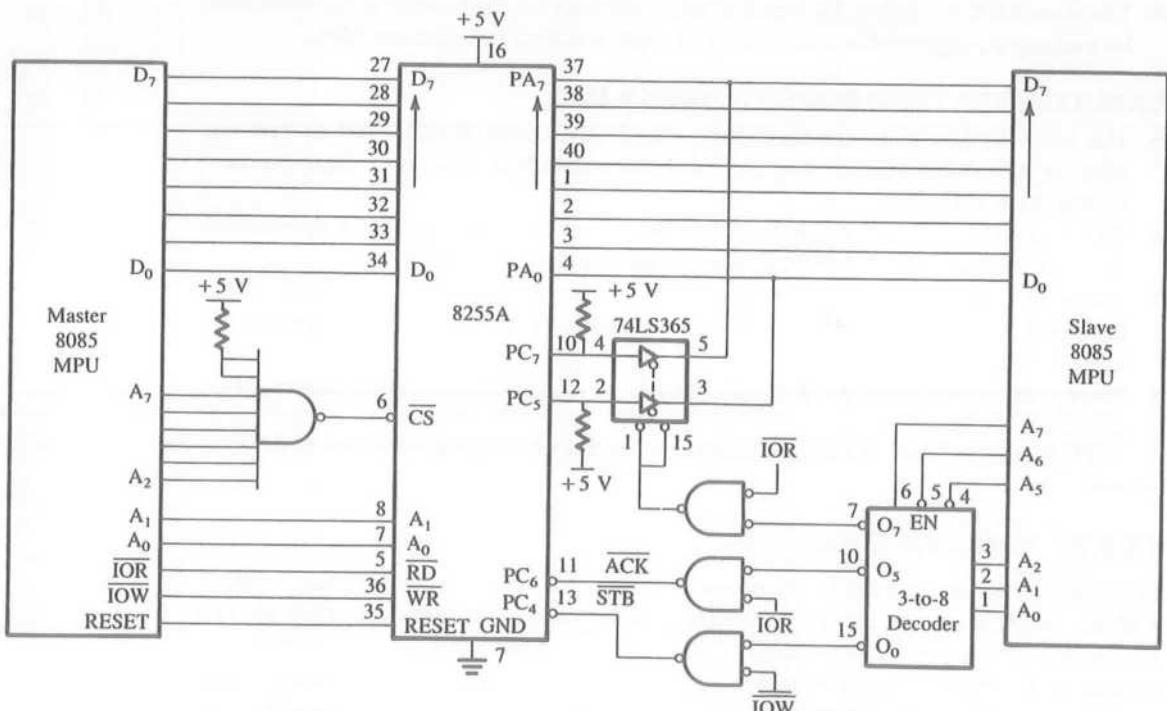


FIGURE 15.19

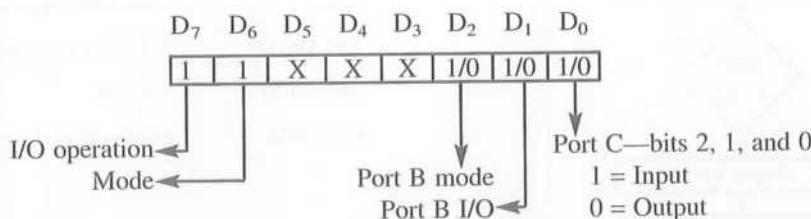
Schematic: Bidirectional Communication between the Master and Slave MPUs

SOURCE: Adapted from Peter Rony, "Interfacing Fundamentals: Bidirectional I/Os Using Two Semaphores." Reprinted with permission from the April 1981 issue of *Computer Design*, copyright 1981 Computer Design Publishing Company.

the 74LS138 (3-to-8) decoder. Assuming the don't care address lines (A_4 and A_3) are at logic 0, the eight output lines of the decoder can be enabled with the addresses from 80H to 87H. Two output lines of the decoder are combined with the \overline{IOR} control signal to generate two input device select pulses (85H and 87H). Input device select pulse 87H is used to read status on the data lines D_7 and D_0 . The decoder line with address 80H is combined with the IOW signal to generate the STB signal.

CONTROL WORD—MODE 2

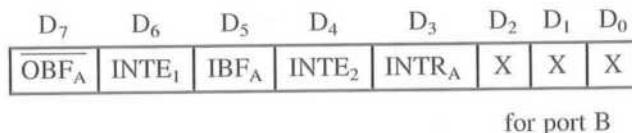
To set up the 8255A in the bidirectional mode (Mode 2), the bits of the control word are defined as follows:



Examination of the control word definition shows that bits D₂ to D₀ are irrelevant in this example because port B and the remaining bits of port C are not being used. Therefore, the required control word is COH.

STATUS WORD—MODE 2

The status of the I/O operation in Mode 2 can be verified by reading the contents of port C. The status word format is as follows:



The status of the signal OBF can be checked by rotating bit D₇ into the Carry, and the status of the signal IBE can be checked by ANDing the status word with data byte 20H.

READ AND WRITE OPERATIONS OF THE SLAVE MPU

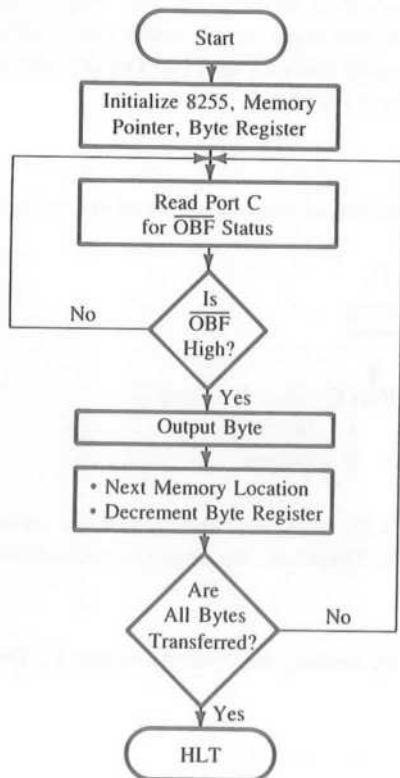
A data byte can be read by the slave MPU from port A simply by sending an active low device select pulse to the ACK signal; there is no need to build an input port. Similarly, a data byte can be written by the slave MPU into port A by causing the STB signal to go low.

15.3.4 Program

PROGRAM COMMENTS

1. The flowcharts in Figure 15.20 and Figure 15.21 show that both programs check for the status line OBF. The master program waits until the OBF goes high, then writes a byte in port A. On the other hand, the slave program waits until the OBF goes low, then reads data.

Master Program



LXI SP,STACK1	
LXI H,MASTR	;Memory pointer for data
MVI B,BYTES	;Number of bytes to be sent
MVI A,CTRL	;Control word
OUT FFH	;Initialize 8255
OBFLO: IN FEH	;Read port C for status
RAL	;Place $\overline{\text{OBF}}$ status in CY
JNC OBFLO	;If $\overline{\text{OBF}}$ is low, wait
MOV A,M	;Get byte
OUT FCH	;Send byte to port A
INX H	;Next memory location
DCR B	;Reduce byte count by 1
JNZ OBFLO	;If all bytes are not transferred, go back to get next byte
HLT	

FIGURE 15.20
Flowchart of Master Program

2. When the master MPU writes a data byte, it is latched by port A, and the data byte is placed on the slave data bus when the ACK goes low. The timing diagram in Figure 15.22 shows that when the slave MPU reads data, the ACK signal goes low, and the falling edge of the ACK signal sets the OBF signal high for the master MPU to write the next byte.
3. The programs given above can transfer a block of data from the master MPU to the slave MPU but not vice versa. To transfer a block of data from the slave MPU to the master MPU, additional instructions are necessary (see Problem 7 at the end of this chapter). These instructions should monitor the IBF signal in both programs and Read/Write operations should be interchanged. The master MPU should wait until IBF goes high to read a data byte, and the slave MPU should wait until IBF goes low to write a byte.
4. The timing diagram in Figure 15.22 shows an INTR signal to implement data transfer using the interrupt. However, the signal is irrelevant in this illustration; it is given in Figure 15.22 for Problem 8 at the end of this chapter.

Slave Program

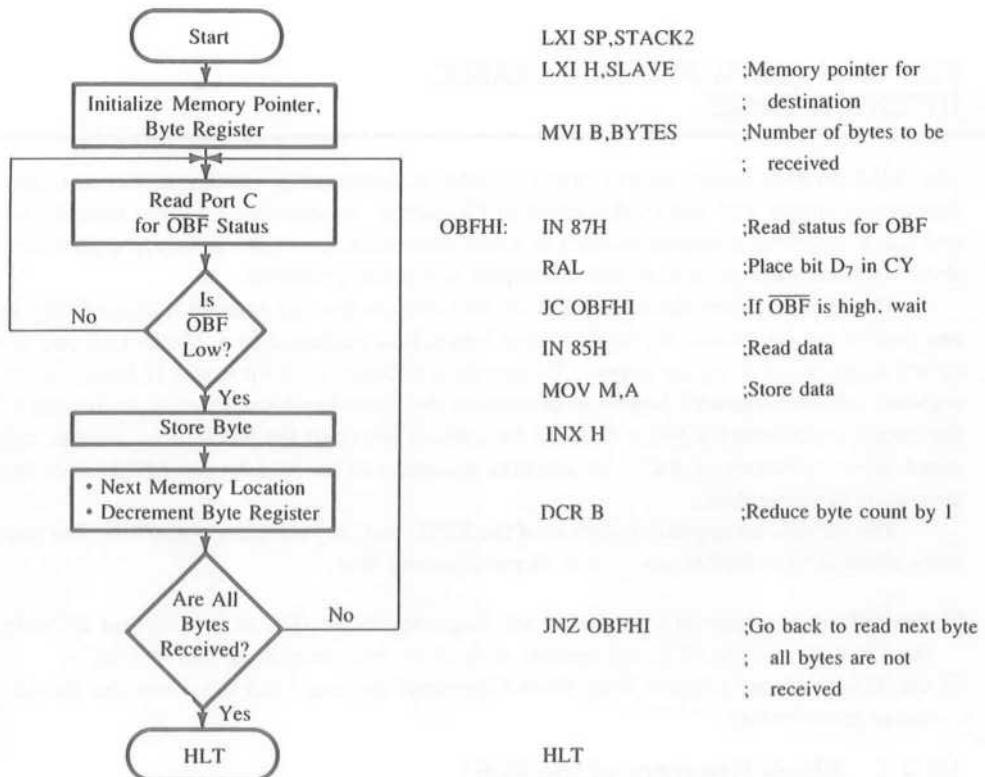


FIGURE 15.21
Flowchart of Slave Program

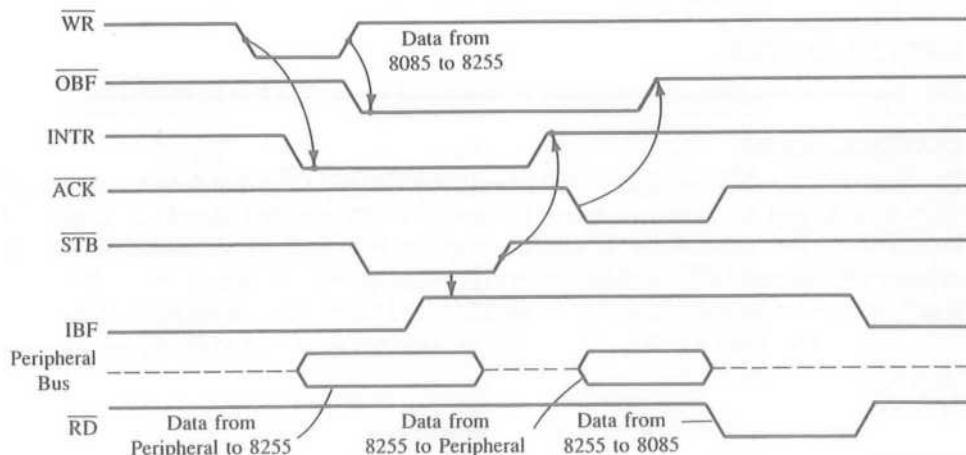


FIGURE 15.22

Timing Diagram Mode 2

SOURCE: Adapted from Intel Corporation, *Peripheral Components* (Santa Clara, Calif.: Author, 1993), p. 3-113.

15.4

THE 8254 (8253) PROGRAMMABLE INTERVAL TIMER

The 8254 programmable interval timer/counter is functionally similar to the software-designed counters and timers described in Chapter 8. It generates accurate time delays and can be used for applications such as a real-time clock, an event counter, a digital one-shot, a square-wave generator, and a complex waveform generator.

The 8254 includes three identical 16-bit counters that can operate independently in any one of the six modes (to be described later). It is packaged in a 24-pin DIP and requires a single +5 V power supply. To operate a counter, a 16-bit count is loaded in its register and, on command, begins to decrement the count until it reaches 0. At the end of the count, it generates a pulse that can be used to interrupt the MPU. The counter can count either in binary or BCD. In addition, a count can be read by the MPU while the counter is decrementing.

The 8254 is an upgraded version of the 8253, and they are pin-compatible. The features of these two devices are almost identical except that

- the 8254 can operate with higher clock frequency range (DC to 8 MHz and 10 MHz for 8254-2), and the 8253 can operate with clock frequency from DC to 2 MHz.
- the 8254 includes a Status Read-Back Command that can latch the count and the status of the counters.

15.4.1 Block Diagram of the 8254

Figure 15.23 is the block diagram of the 8254; it includes three counters (0, 1, and 2), a data bus buffer, Read/Write control logic, and a control register. Each counter has two input signals—Clock (CLK) and GATE—and one output signal—OUT.

DATA BUS BUFFER

This tri-state, 8-bit, bidirectional buffer is connected to the data bus of the MPU.

CONTROL LOGIC

The control section has five signals: RD (Read), WR (Write), CS (Chip Select), and the address lines A₀ and A₁. In the peripheral I/O mode, the RD and WR signals are connected to IOR and IOW, respectively. In memory-mapped I/O, these are connected to MEMR (Memory Read) and MEMW (Memory Write). Address lines A₀ and A₁ of the MPU are usually connected to lines A₀ and A₁ of the 8254, and CS is tied to a decoded address.

The control word register and counters are selected according to the signals on lines A₀ and A₁, as shown below:

A ₁	A ₀	Selection
0	0	Counter 0
0	1	Counter 1
1	0	Counter 2
1	1	Control Register

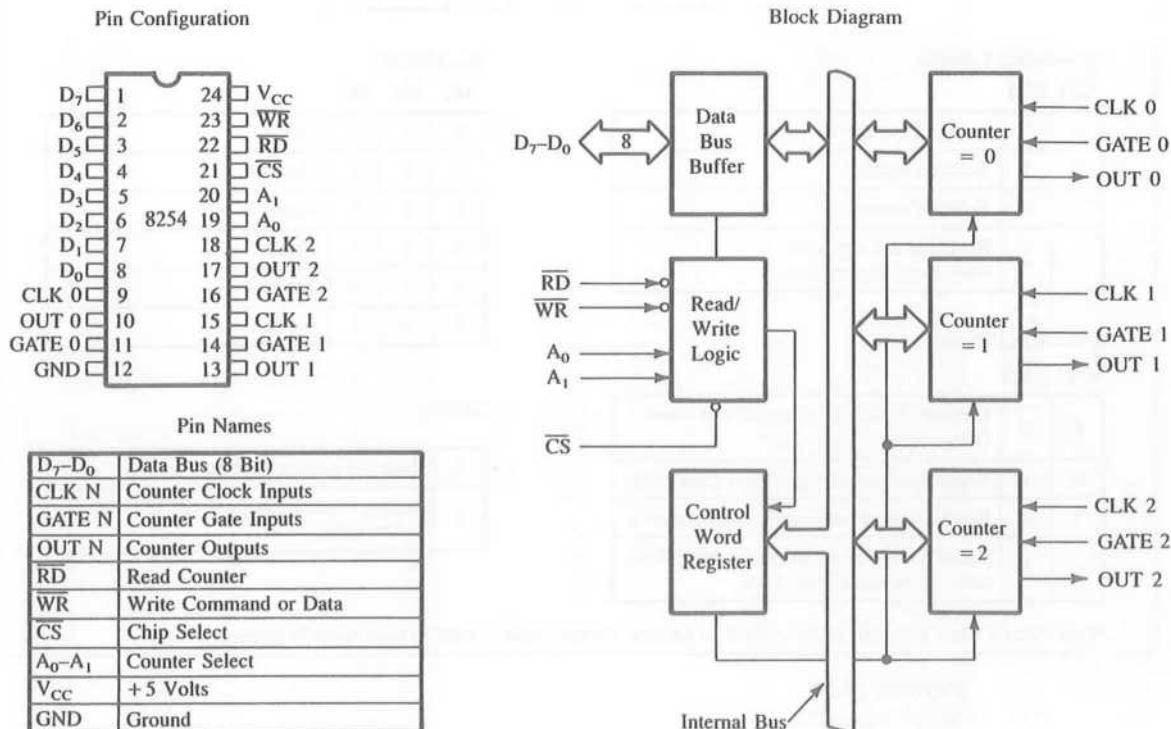


FIGURE 15.23

8254 Block Diagram

SOURCE: Intel Corporation, *Peripheral Components* (Santa Clara, Calif.: Author, 1993), p. 3-62.

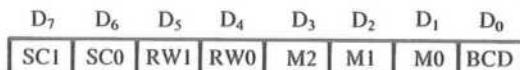
CONTROL WORD REGISTER

This register is accessed when lines A₀ and A₁ are at logic 1. It is used to write a command word which specifies the counter to be used, its mode, and either a Read or a Write operation. The control word format is shown in Figure 15.24.

MODE

The 8254 can operate in six different modes, and the gate of a counter is used either to disable or enable counting, as shown in Figure 15.25. However, to maintain clarity, only one mode (Mode 0) is illustrated first, and details of the remaining modes are discussed in Section 15.4.4.

In Mode 0, after the count is written and if the gate is high, the count is decremented every clock cycle. When the count reaches zero, the output goes high and remains high until a new count or mode word is loaded.



SC—Select Counter:

SC1 SC0

0	0	Select Counter 0
0	1	Select Counter 1
1	0	Select Counter 2
1	1	Read-Back Command (See Read Operations)

RW—Read/Write:

RW1 RW0

0	0	Counter Latch Command (see Read Operations)
0	1	Read/Write least significant byte only.
1	0	Read/Write most significant byte only.
1	1	Read/Write least significant byte first, then most significant byte.

M—MODE:

M2 M1 M0

0	0	0	Mode 0
0	0	1	Mode 1
X	1	0	Mode 2
X	1	1	Mode 3
1	0	0	Mode 4
1	0	1	Mode 5

BCD:

0	Binary Counter 16-bits
1	Binary Coded Decimal (BCD) Counter (4 Decades)

Note: Don't Care Bits (X) Should Be 0 to Ensure Compatibility with Future Intel Products.

FIGURE 15.24
8254 Control Word Format

SOURCE: Intel Corporation, *Peripheral Components* (Santa Clara, Calif.: Author, 1993), p. 3-67.

Modes \ Signal Status	Low or Going Low	Rising	High
0	Disables counting	—	Enables counting
1	—	(1) Initiates counting (2) Resets output after next clock	—
2	(1) Disables counting (2) Sets output immediately high	(1) Reloads counter (2) Initiates counting	Enables counting
3	(1) Disables counting (2) Sets output immediately high	Initiates counting	Enables counting
4	Disables counting	—	Enables counting
5	—	Initiates counting	—

FIGURE 15.25
Gate Settings of a Counter

SOURCE: Intel Corporation, *Peripheral Components* (Santa Clara, Calif.: Author, 1993), p. 3-78.

15.4.2 Programming the 8254

The 8254 can be programmed to provide various types of output (see Section 15.4.4, Figure 15.27) through Write operations, or to check a count while counting through Read operations. The details of these operations are given below.

WRITE OPERATIONS

To initialize a counter, the following steps are necessary.

1. Write a control word into the control register.
2. Load the low-order byte of a count in the counter register.
3. Load the high-order byte of a count in the counter register.

With a clock and an appropriate gate signal to one of the counters, the above steps should start the counter and provide appropriate output according to the control word.

READ OPERATIONS

In some applications, especially in event counters, it is necessary to read the value of the count in progress. This can be done by either of two methods. One method involves reading a count after inhibiting (stopping) the counter to be read. The second method involves reading a count while the count is in progress (known as reading on the fly).

In the first method, counting is stopped (or inhibited) by controlling the gate input or the clock input of the selected counter, and two I/O read operations are performed by the MPU. The first I/O operation reads the low-order byte, and the second I/O operation reads the high-order byte.

In the second method, an appropriate control word is written into the control register to latch a count in the output latch, and two I/O Read operations are performed by the MPU. These Read/Write operations are illustrated below.

15.4.3 Illustration: The 8254 as a Counter

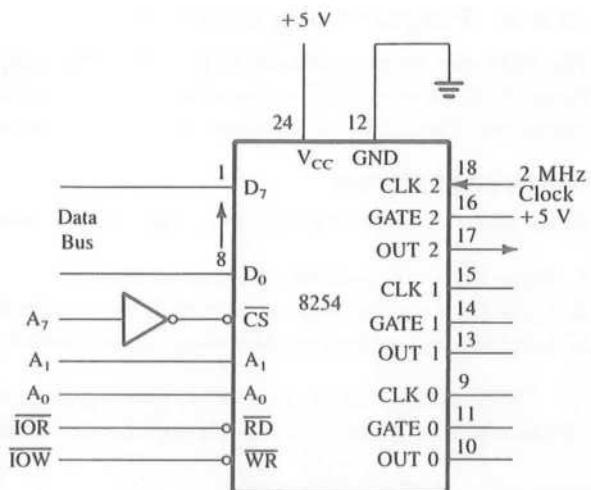
PROBLEM STATEMENT

1. Identify the port addresses of the control register and counter 2 in Figure 15.26.
2. Write a subroutine to initialize counter 2 in Mode 0 with a count of $50,000_{10}$. The subroutine should also include reading counts on the fly; when the count reaches zero, it should return to the main program.
3. Write a main program to display seconds by calling the subroutine as many times as necessary.

1. Port Addresses The Chip Select is enabled when $A_7 = 1$ (see Figure 15.26), and the control register is selected when A_1 and $A_0 = 1$. Similarly, counter 2 is selected when $A_1 = 1$ and $A_0 = 0$. Assuming that the unused address lines A_6 to A_2 are at logic 0, the port addresses will be as follows:

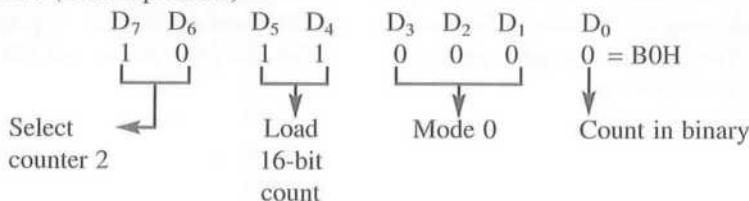
Control Register = 83H
Counter 2 = 82H

FIGURE 15.26
Schematic: Interfacing the 8254



2. Subroutine Counter To initialize the 8254 for counter 2 in Mode 0, the following control word is necessary (see Figure 15.24):

Control Word (Load Operation)



Control Word for Latching: Bits D_5 and D_4 should be 0 = 80H

Subroutine

COUNTER:	MVI A,B0H	;Control word to initialize counter 2
	OUT 83H	;Write in the control register
	MVI A,LOBYTE	;Low-order byte of the count 50000
	OUT 82H	;Load counter 2 with the low-order byte
	MVI A,HIBYTE	;High-order byte of the count 50000
	OUT 82H	;Load counter 2 with the high-order byte
READ:	MVI A,80H	;Control word to latch a count
	OUT 83H	;Write in the control register
	IN 82H	;Read low-order byte
	MOV D,A	;Store low-order byte in register D
	IN 82H	;Read high-order byte
	ORA D	;OR low- and high-order bytes to set Z flag
	JNZ READ	;If counter $\neq 0$, go back to read next count
	RET	

Subroutine Description The subroutine has two segments. In the first segment, counter 2 is initialized by writing a control word in the control register and a 16-bit count specified as LOBYTE and HIBYTE in the counter register. The hexadecimal value equivalent to 50000_{10} must be calculated.

In the second segment (beginning at READ), a control word is written into the control register to sample a count, and the 16-bit count is read by performing two input operations. The reading of the counter is repeated until the counter reaches 0; the Zero flag is checked by ORing the low- and high-order bytes.

3. Main Program The subroutine COUNTER provides 25 ms ($50000 \times 0.5 \mu\text{s}$ Clock) delay; if this routine is called 40 times, the total delay will be one second.

Program

LXI SP,STACK	;Initialize stack pointer
MVI B,00H	;Clear register B to save number of seconds
SECOND: MVI C,28H	;Set up register C to count 40_{10}
WAIT: CALL COUNTER	;Wait for 25 ms
DCR C	
JNZ WAIT	;Is this one second? If not, go back and wait
MVI A,B	
ADI 01	;Add one second
DAA	
OUT PORT1	
MOV B,A	;Save seconds
JMP SECOND	;Go back and start counting the next second

Program Description The main program initializes the stack pointer; loads register C with the count of 28H (40_{10}) and sets up the WAIT loop. The loop calls the COUNTER subroutine 40 times to generate a one-second delay. At the end of the loop, it increments the seconds in register B, decimal-adjusts the byte, and displays seconds. The sequence is repeated until register B reaches 99_{BCD} . After the 99th second, register B is cleared and the clock sequence is repeated.

This program is just to demonstrate the Read and Write operations of the 8254; this clock design does not take into account the errors caused by the delay in executing the program instructions. A better way of designing a real-time clock is to interrupt the MPU at the end of a count (see Problem 15 at the end of this chapter).

15.4.4 Modes

As mentioned earlier, the 8254 can operate in six different modes; we already illustrated Mode 0 in Section 15.2.3. Now we will describe briefly various modes of the 8254 including Mode 0.

MODE 0: INTERRUPT ON TERMINAL COUNT

In this mode, initially the OUT is low. Once a count is loaded in the register, the counter is decremented every cycle, and when the count reaches zero, the OUT goes high. This

can be used as an interrupt. The OUT remains high until a new count or a command word is loaded. Figure 15.27 also shows that the counting ($m = 5$) is temporarily stopped when the Gate is disabled ($G = 0$), and continued again when the Gate is at logic 1.

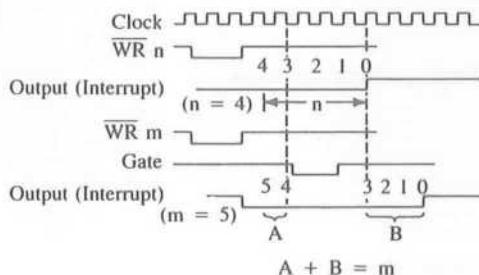
MODE 1: HARDWARE-RETRIGGERABLE ONE-SHOT

In this mode, the OUT is initially high. When the Gate is triggered, the OUT goes low, and at the end of the count, the OUT goes high again, thus generating a one-shot pulse (Figure 15.27, Mode 1).

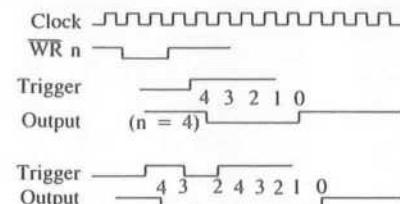
MODE 2: RATE GENERATOR

This mode is used to generate a pulse equal to the clock period at a given interval. When a count is loaded, the OUT stays high until the count reaches 1, and then the OUT goes

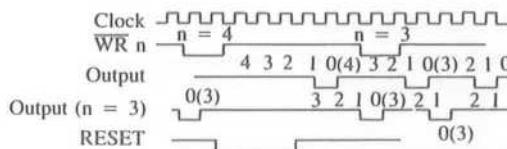
Mode 0: Interrupt on Terminal Count



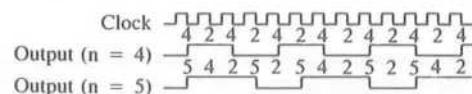
Mode 1: Programmable One-Shot



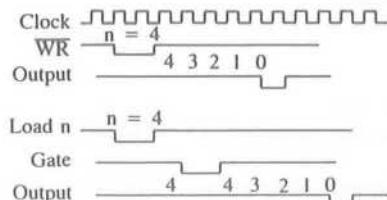
Mode 2: Rate Generator Clock



Mode 3: Square Wave Generator



Mode 4: Software Triggered Strobe



Mode 5: Hardware Triggered Strobe

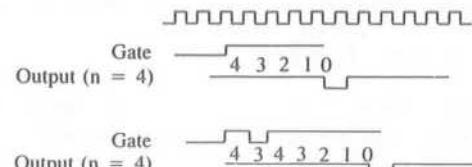


FIGURE 15.27
Six Modes of the 8254

SOURCE: Intel Corporation, *Peripheral Components* (Santa Clara, Calif.: Author, 1993), pp. 3-72-3-75 (adapted).

low for one clock period. The count is reloaded automatically, and the pulse is generated continuously. The count = 1 is illegal in this mode.

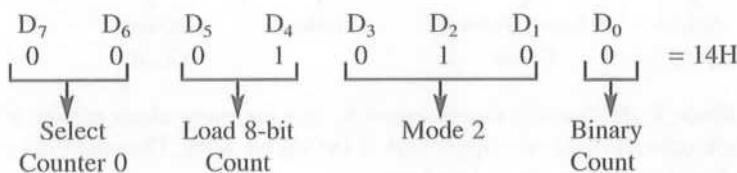
Write instructions to generate a pulse every 50 µs from Counter 0 (refer to Figure 15.26).

**Example
15.3**

To generate a pulse every 50 µs, from Counter 0, it should be initialized in Mode 2 (refer to Figure 15.27 for modes), and Gate 0 should be high.

Solution

Control Word (Refer to Figure 15.24):



Count In Mode 2, the count is decremented every clock period, and at the last count, the counter generates a pulse equivalent to the clock period of the timer. Here the clock frequency of the 8254 is 2 MHz (0.5 µs clock period), and the pulse should be generated every 50 µs. Therefore, the count is calculated as follows:

$$\text{Count} = \frac{50 \times 10^{-6}}{0.5 \times 10^{-6}} = 100 = 64H$$

In this example, the frequency of the pulse is 20 kHz (1/50 µs). This count can also be calculated by dividing the clock frequency by the frequency of the pulse (2 MHz/20 kHz = 100).

Instructions

PULSE:	MVI A, 00010100B	;Control word Mode 2 and Counter 0
	OUT 83H	;Write in 8254 control register
	MVI A,64H	;Low-order byte of the count
	OUT 80H	;Load Counter 0 with low-order byte
	HALT	

MODE 3: SQUARE-WAVE GENERATOR

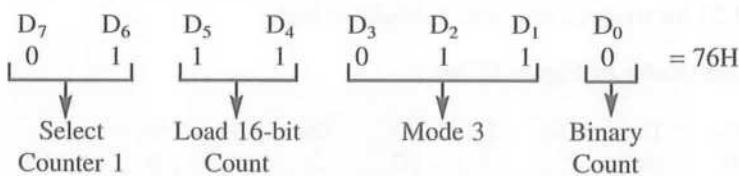
In this mode, when a count is loaded, the OUT is high. The count is decremented by two at every clock cycle, and when it reaches zero, the OUT goes low, and the count is reloaded again. This is repeated continuously; thus, a continuous square wave with period equal to the period of the count is generated. In other words, the frequency of the square wave is equal to the frequency of the clock divided by the count. If the count (N) is odd, the pulse stays high for $(N + 1)/2$ clock cycles and stays low for $(N - 1)/2$ clock cycles.

**Example
15.4**

Write instructions to generate a 1 kHz square wave from Counter 1 (refer to Figure 15.26). Assume the gate of Counter 1 is tied to +5 V through a 10 k resistor. Explain the significance of connecting the gate to +5 V.

Solution

To generate a square wave from Counter 1, it should be initialized in Mode 3 (refer to Figure 15.27 for modes).

Control Word (Refer to Figure 15.24)

Count In Mode 3, the count is decremented by two for every clock period. If the count is N, $N/2$ clock pulses provide the upper half of the square wave. The count is loaded again and $N/2$ clock pulses provide the lower half.

In this example, the clock frequency of the 8254 is 2 MHz (0.5 μ s clock period), and the square wave frequency is 1 kHz (1 ms clock period). Therefore, we need a count for 1 ms delay.

$$\text{Count} = \frac{1 \times 10^{-3}}{0.5 \times 10^{-6}} = 2000 = 07D0H$$

This count can also be calculated by dividing the clock frequency by the square wave frequency ($2 \text{ MHz}/1 \text{ kHz} = 2000$).

Instructions

SQWAVE:	MVI A, 01110110B	;Control word Mode 3 and Counter 1
	OUT 83H	;Write in 8254 control register
	MVI A, D0H	;Low-order byte of the count
	OUT 81H	;Load Counter 1 with low-order byte
	MVI A, 07H	;High-order byte of the count
	OUT 81H	;Load Counter 1 with high-order byte
	HALT	

To run Counter 1, the gate of that counter must be tied high; otherwise the counter action is inhibited.

MODE 4: SOFTWARE-TRIGGERED STROBE

In this mode, the OUT is initially high; it goes low for one clock period at the end of the count. The count must be reloaded for subsequent outputs.

MODE 5: HARDWARE-TRIGGERED STROBE

This mode is similar to Mode 4, except that it is triggered by the rising pulse at the gate. Initially, the OUT is low, and when the Gate pulse is triggered from low to high, the count begins. At the end of the count, the OUT goes low for one clock period.

READ-BACK COMMAND

The Read-Back Command in the 8254 allows the user to read the count and the status of the counter; this command is not available in the 8253. The format of the command is shown in Figure 15.28(a).

The command is written in the control register, and the count of the specified counter(s) can be latched if COUNT (bit D₅) is 0. A counter or a combination of counters is specified by keeping the respective CNT bits (D₁, D₂, and D₃) high. For example, the control word 1 1 0 1 0 1 1 0 (D6H) written in the control register will latch the counts of Counter 0 and Counter 1, and these counts can be obtained by reading respective counter

(a) Read-Back Command Format

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
1	1	COUNT	STATUS	CNT 2	CNT 1	CNT 0	0

D₅: 0 = Latch Count of Selected Counter(s)

A₀, A₁ = 11

D₄: 0 = Latch Status of Selected Counter(s)

\overline{CS} = 0

D₃: 1 = Select Counter 2

\overline{RD} = 1

D₂: 1 = Select Counter 1

\overline{WR} = 0

D₁: 1 = Select Counter 0

D₀: Reserved for Future Expansion; Must Be 0

(b) Status Byte

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
OUTPUT	NULL COUNT	RW1	RW0	M2	M1	M0	BCD

D₇: 1 = Out Pin is 1

: 0 = Out Pin is 0

D₆: 1 = Null Count

: 0 = Count Available
for Reading

D₅-D₀: Counter Programmed Mode

FIGURE 15.28

Read-Back Command Format (a) and Status Byte (b)

SOURCE: Intel Corporation, *Peripheral Components* (Santa Clara, Calif.: Author, 1993), p. 3-69.

port addresses. The latched counts are held until they are read or the counters are reprogrammed. The Read-Back Command eliminates the need of writing separate counter-latch commands for different counters.

The status of the counter(s) can be read if STATUS bit (D₄) of the Read-Back Command is low. Figure 15.28(b) shows the format of the status byte.

Example 15.5

Write a subroutine to generate an interrupt every 1 sec. Refer to Figure 15.26 for counter addresses.

Solution

The clock frequency shown in Figure 15.26 is 2 MHz; thus, a count is decremented every 0.5 μ s. To obtain a delay of one second, the count should be $(1 \text{ sec}/0.5 \times 10^{-6}) 2 \text{ Meg.}$; this count is too large for one 16-bit counter. We can divide this count, as an example, 50,000 for Counter 1 and 40 for second Counter 2 ($50 \text{ k} \times 40 = 2 \text{ Meg.}$). If we set up Counter 1 in Mode 2 with 50,000 as a count, it will generate a pulse every 25 msec. The output of Counter 1 can be used as a clock input to Counter 2. That means the count in Counter 2 will be decremented every 25 msec. If Counter 2 is also set up in Mode 2 with the count = 40, the output pulse from Counter 2 is generated every second that can be used to interrupt the processor.

Control Word (Refer to Figure 15.24)

Instructions

The following subroutine is an initialization for 8254 timer:

It uses two counters to generate an interrupt every one second.

```

CNT1LO EQU 50H
CNT1HI EQU C3H
COUNT2 EQU 40
SECOND: MVI A, 01110100B
          OUT 83H
          MVI A, 10010100B
          OUT 83H
          MVI A, CNT1LO
          OUT 81H
          MVI A, CNT1HI
          OUT 81H

```

```
;Control word: Mode 2, Counter 1  
;Write in 8254 control register  
;Control word: Mode 2, Counter 2  
;Write in 8254 control register  
;Low-order byte of count 50,000  
;Load Counter 1 with low-order byte  
;High-order byte of count 50,000  
;Load Counter 1 with high-order byte
```

```
MVI A, COUNT2      ;Count for Counter 2
OUT 82H           ;Load Counter 2
RET
```

This subroutine sends two control words in a sequence to initialize Counter 1 and Counter 2. The 8254 differentiates these words according to the specified counter in the control words. We could have initialized these counters in a different sequence; for example, the control word for Counter 1 followed by its count. Initially, the equates CNTILO and CNT1HI are defined in Hex numbers and the equate COUNT2 is specified in decimal. This is based on the assumption that an assembler will convert the decimal count in Hex equivalent. However, this procedure is not feasible for the 16-bit count of 50,000; therefore, the equivalent Hex count (C350H) is loaded into two registers as 50H (low) and C3H (high).

Once this subroutine is called, the output of Counter 2 will generate a pulse every 1 second because the counts are reloaded automatically at the end of each count. This pulse can be used to interrupt the processor, and the processor can update the clock every second. This is a hardware dependent timing and more accurate than the technique used in Section 15.4.3.

THE 8259A PROGRAMMABLE INTERRUPT CONTROLLER

15.5

The 8259A is a programmable interrupt controller designed to work with Intel microprocessors 8085, 8086, and 8088. The 8259A interrupt controller can

1. manage eight interrupts according to the instructions written into its control registers. This is equivalent to providing eight interrupt pins on the processor in place of one INTR (8085) pin.
2. vector an interrupt request anywhere in the memory map. However, all eight interrupts are spaced at the interval of either four or eight locations. This eliminates the major drawback of the 8085 interrupts in which all interrupts are vectored to memory locations on page 00H.
3. resolve eight levels of interrupt priorities in a variety of modes, such as fully nested mode, automatic rotation mode, and specific rotation mode (to be explained later).
4. mask each interrupt request individually.
5. read the status of pending interrupts, in-service interrupts, and masked interrupts.
6. be set up to accept either the level-triggered or the edge-triggered interrupt request.
7. be expanded to 64 priority levels by cascading additional 8259As.
8. be set up to work with either the 8085 microprocessor mode or the 8086/8088 microprocessor mode.

The 8259A is upward-compatible with its predecessor, the 8259. The main difference between the two is that the 8259A can be used with Intel's 8086/88 16-bit microprocessors. It also includes additional features such as the level-triggered mode, buffered

mode, and automatic-end-of-interrupt mode. To simplify the explanation of the 8259A, illustrative examples will not include the cascade mode or the 8086/88 mode and will be limited to modes commonly used with the 8085.

15.5.1 Block Diagram of the 8259A

Figure 15.29 shows the internal block diagram of the 8259A. It includes eight blocks: control logic, Read/Write logic, data bus buffer, three registers (IRR, ISR, and IMR), priority resolver, and cascade buffer. This diagram shows all the elements of a programmable device, plus additional blocks. The functions of some of these blocks need explanation, which is given below.

READ/WRITE LOGIC

This is a typical Read/Write control logic. When the address line A_0 is at logic 0, the controller is selected to write a command or read a status. The Chip Select logic and A_0 determine the port address of the controller.

CONTROL LOGIC

This block has two pins: INT (Interrupt) as an output, and INTA (Interrupt Acknowledge) as an input. The INT is connected to the interrupt pin of the MPU. Whenever a valid interrupt is asserted, this signal goes high. The INTA is the Interrupt Acknowledge signal from the MPU.

INTERRUPT REGISTERS AND PRIORITY RESOLVER

The Interrupt Request Register (IRR) has eight input lines (IR_0 - IR_7) for interrupts. When these lines go high, the requests are stored in the register. The In-Service Register (ISR) stores all the levels that are currently being serviced, and the Interrupt Mask Register (IMR) stores the masking bits of the interrupt lines to be masked. The Priority Resolver (PR) examines these three registers and determines whether INT should be sent to the MPU.

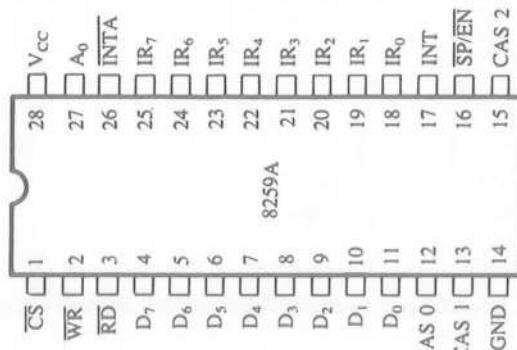
CASCADE BUFFER/COMPARATOR

This block is used to expand the number of interrupt levels by cascading two or more 8259As. To simplify the discussion, this block will not be mentioned again.

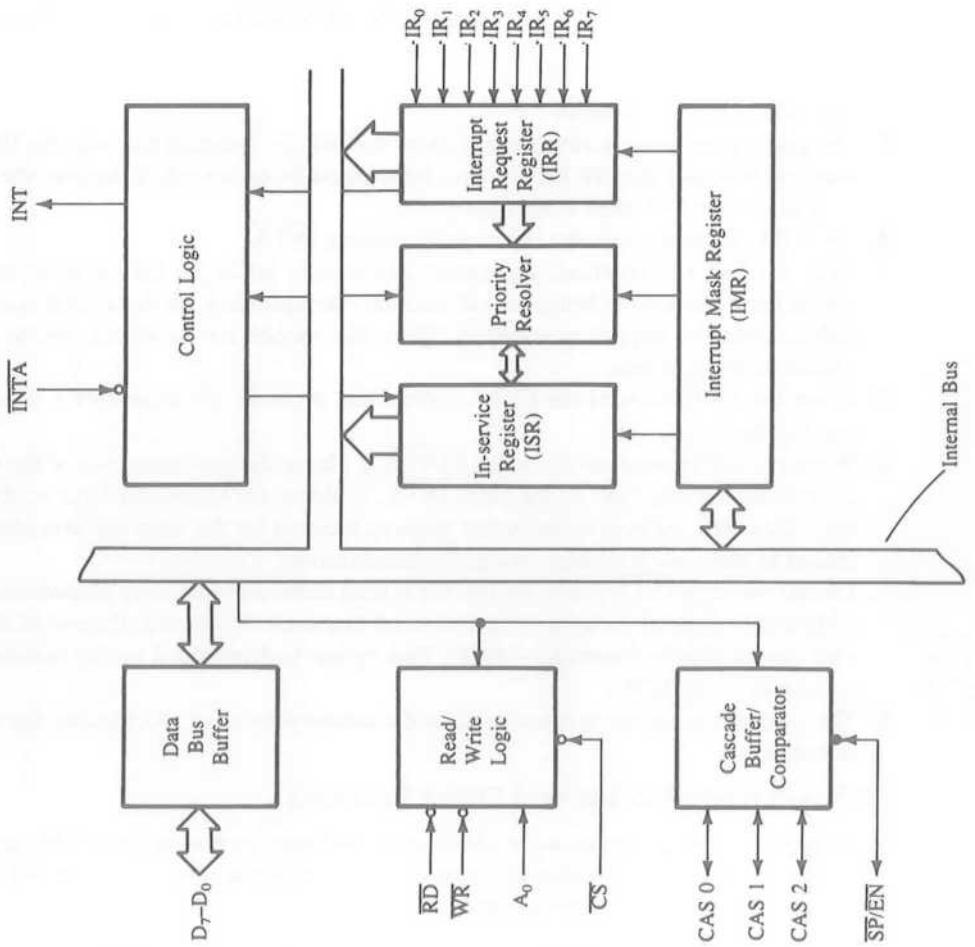
15.5.2 Interrupt Operation

To implement interrupts, the Interrupt Enable flip-flop in the microprocessor should be enabled by writing the EI instruction, and the 8259A should be initialized by writing control words in the control register. The 8259A requires two types of control words: Initialization Command Words (ICWs) and Operational Command Words (OCWs). The ICWs are used to set up the proper conditions and specify RST vector addresses. The OCWs are used to perform functions such as masking interrupts, setting up status-read operations, etc. After the 8259A is initialized, the following sequence of events occurs when one or more interrupt request lines go high:

Pin Configuration



Block Diagram



Pin Names	
D_7-D_0	Data Bus (Bidirectional)
\overline{RD}	Read Input
\overline{WR}	Write Input
A_0	Command Select Address
\overline{CS}	Chip Select
CAS_2-CAS_0	Cascade Lines
$\overline{SP/EN}$	Slave Program/Enable Buffer
INT	Interrupt Output
\overline{INTA}	Interrupt Acknowledge Input
IR_0-IR_7	Interrupt Request Inputs

FIGURE 15.29

The 8259A Block Diagram
SOURCE: Intel Corporation, *Peripheral Components* (Santa Clara, Calif.: Author, 1993), pp. 3-171, 3-172.

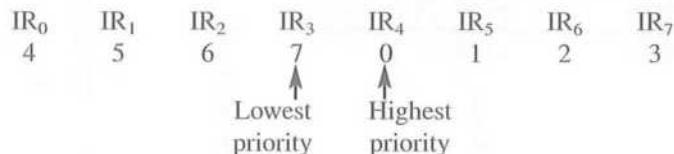
1. The IRR stores the requests.
2. The priority resolver checks three registers: the IRR for interrupt requests, the IMR for masking bits, and the ISR for the interrupt request being served. It resolves the priority and sets the INT high when appropriate.
3. The MPU acknowledges the interrupt by sending INTA.
4. After the INTA is received, the appropriate priority bit in the ISR is set to indicate which interrupt level is being served, and the corresponding bit in the IRR is reset to indicate that the request is accepted. Then, the opcode for the CALL instruction is placed on the data bus.
5. When the MPU decodes the CALL instruction, it places two more INTA signals on the data bus.
6. When the 8259A receives the second INTA, it places the low-order byte of the CALL address on the data bus. At the third INTA, it places the high-order byte on the data bus. The CALL address is the vector memory location for the interrupt; this address is placed in the control register during the initialization.
7. During the third INTA pulse, the ISR bit is reset either automatically (Automatic-End-of-Interrupt—AEOI) or by a command word that must be issued at the end of the service routine (End-of-Interrupt—EOI). This option is determined by the initialization command word (ICW).
8. The program sequence is transferred to the memory location specified by the CALL instruction.

15.5.3 Priority Modes and Other Features

Many types of priority modes are available under software control in the 8259A, and they can be changed dynamically during the program by writing appropriate command words. Commonly used priority modes are discussed below.

1. **Fully Nested Mode** This is a general-purpose mode in which all IRs (Interrupt Requests) are arranged from highest to lowest, with IR₀ as the highest and IR₇ as the lowest.

In addition, any IR can be assigned the highest priority in this mode; the priority sequence will then begin at that IR. In the example below, IR₄ has the highest priority, and IR₃ has the lowest priority:



2. **Automatic Rotation Mode** In this mode, a device, after being serviced, receives the lowest priority. Assuming that the IR₂ has just been serviced, it will receive the seventh priority, as shown below:

IR ₀	IR ₁	IR ₂	IR ₃	IR ₄	IR ₅	IR ₆	IR ₇
1	6	7	0	1	2	3	4

3. Specific Rotation Mode This mode is similar to the automatic rotation mode, except that the user can select any IR for the lowest priority, thus fixing all other priorities.

END OF INTERRUPT

After the completion of an interrupt service, the corresponding ISR bit needs to be reset to update the information in the ISR. This is called the End-of-Interrupt (EOI) command. It can be issued in three formats:

1. Nonspecific EOI Command When this command is sent to the 8259A, it resets the highest priority ISR bit.

2. Specific EOI Command This command specifies which ISR bit to reset.

3. Automatic EOI In this mode, no command is necessary. During the third $\overline{\text{INTA}}$, the ISR bit is reset. The major drawback with this mode is that the ISR does not have information on which IR is being serviced. Thus, any IR can interrupt the service routine, irrespective of its priority, if the Interrupt Enable flip-flop is set.

ADDITIONAL FEATURES OF THE 8259A

The 8259A is a complex device with various modes of operation. These modes are listed below for reference; the user should refer to the *8085 User's Manual* for details.

- **Interrupt Triggering:** The 8259A can accept an interrupt request with either the edge-triggered mode or the level-triggered mode. The mode is determined by the initialization instructions.
- **Interrupt Status:** The status of the three interrupt registers (IRR, ISR, and IMR) can be read, and this status information can be used to make the interrupt process versatile.
- **Poll Method:** The 8259A can be set up to function in a polled environment. The MPU polls the 8259A rather than each peripheral.

15.5.4 Programming the 8259A

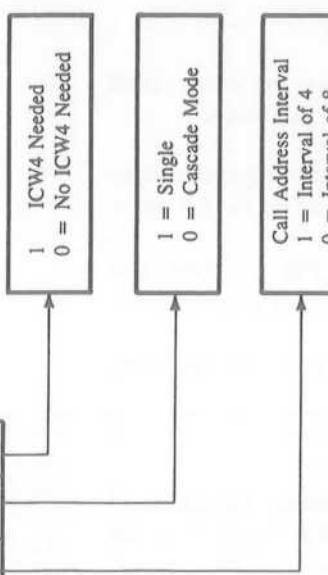
As mentioned before, the 8259A requires two types of command words: Initialization Command Words (ICWs) and Operational Command Words (OCWs). The 8259A can be initialized with four ICWs; the first two are essential, and the other two are optional based on the modes being used. These words must be issued in a given sequence. Once initialized, the 8259A can be set up to operate in various modes by using three different OCWs; however, they no longer need be issued in a specific sequence.

Figure 15.30 shows the bit specification of the first two ICWs. The ICW₁, shown in Figure 15.30(a), specifies

1. single or multiple 8259As in the system.
2. 4- or 8-bit interval between the interrupt vector locations.
3. the address bits A₇–A₅ of the CALL instruction; the rest are supplied by the 8259A, as shown in Figure 15.30(b).

ICW1

A ₀	D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
0	A ₇	A ₆	A ₅	1	LTIM	ADI	SNGL	IC4



(a)

ICW1

IR	D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
7	A ₇	A ₆	A ₅	1	1	1	0	0
6	A ₇	A ₆	A ₅	1	1	0	0	0
5	A ₇	A ₆	A ₅	1	0	1	0	0
4	A ₇	A ₆	A ₅	1	0	0	0	0
3	A ₇	A ₆	A ₅	0	1	1	0	0
2	A ₇	A ₆	A ₅	0	1	0	0	0
1	A ₇	A ₆	A ₅	0	0	1	0	0
0	A ₇	A ₆	A ₅	0	0	0	0	0

(b)

ICW2

A ₀	D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
1	A ₁₅	A ₁₄	A ₁₃	A ₁₂	A ₁₁	A ₁₀	A ₉	A ₈

A₁₅-A₈ of Interrupt Vector Address (MCS80/85 Mode)

(c)

FIGURE 15.30

Initialization Command Words for the 8259A

SOURCE: Intel Corporation, *Peripheral Components* (Santa Clara, Calif.: Author, 1993), p. 3-181.

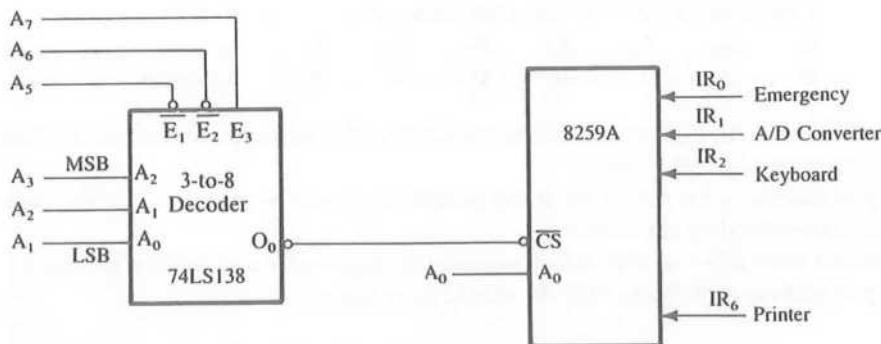


FIGURE 15.31
Schematic of an Interrupt System Using the 8259A

The ICW₂ of Figure 15.30(c) specifies the high-order byte of the CALL instruction.

EXAMPLES

Figure 15.31 shows the schematic of an interrupt-driven system using the 8259A. Four sources are connected to the IR lines of the 8259A: Emergency Signal, Keyboard, A/D Converter, and Printer. Of these, the Emergency Signal has the highest priority and the Printer has the lowest priority.

Explain the following initialization instructions in reference to Figure 15.31.

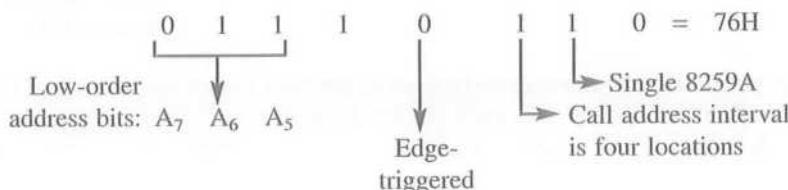
Example
15.6

Initialization Instructions

```
DI
MVI A,76H    ;ICW1
OUT 80H      ;Initialize 8259A
MVI 20H      ;ICW2
OUT 81H      ;Initialize 8259A
```

1. The DI instruction disables the interrupts so that the initialization process will not be interrupted.
2. The command word 76H specifies the following parameters; see Figure 15.30(a):

Solution



Low-Order Byte of the IR_0 Call Address; see Figure 15.22(b):

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
0	1	1	0	0	0	0	0 = 60H

The address bits A_4 – A_0 are supplied by the 8259A. The subsequent addresses are four locations apart (e.g., $IR_1 = 64H$).

3. The port address of the 8259A for ICW_1 is $80H$; A_0 should be at logic 0, and the other bits are determined by the decoder.
4. Command word ICW_2 is $20H$, which specifies the high-order byte of the Call address.
5. The port address of ICW_2 is $81H$; A_0 should be at logic 1.

Example 15.7

Explain the interrupt process in the fully nested mode relative to Figure 15.32. Assume that the 8259A is initialized with the same instructions as in the previous example.

Solution

Figure 15.32 shows that the interrupts are enabled by the main program, and the 8259A is initialized. After the initialization, the 8259A is set in the fully nested mode by default, unless a different Operational Command Word (OCW) is issued. During the main program, the printer has made a request. Because the interrupts are enabled, the program is transferred first to the vectored location $2078H$ for IR_6 , and then to the service routine. The ISR_6 bit is also set to indicate that IR_6 is being serviced.

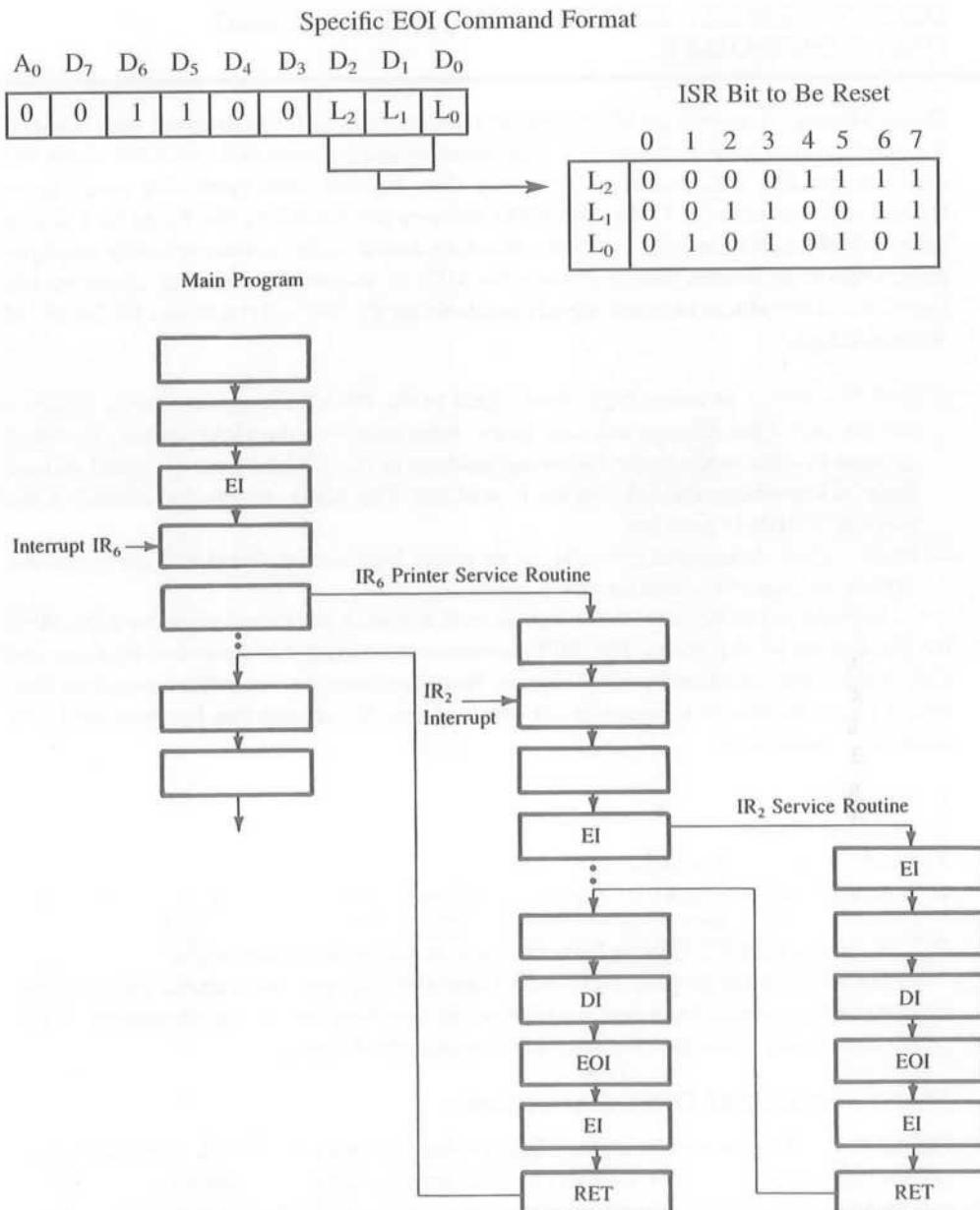
During the IR_6 service routine, the keyboard makes a request (IR_2). Even though IR_2 has a higher priority than the IR_6 , the request is not acknowledged until the IR_6 service routine enables the interrupts through the EI instruction. When IR_2 is acknowledged, bit ISR_2 is set, and the program is vectored to the location $2068H$ and then to the service routine.

At the end of the IR_2 service routine, the instruction EOI (End-of-Interrupt) informs the 8259A that the service has been completed, and the highest ISR bit (ISR_2) has been reset. The program returns to the IR_6 service routine, completes the service, sends the EOI, and then returns to the main program. The EOI in this routine resets the ISR_6 bit.

The format for the nonspecific EOI command is as follows:

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
0	0	0	1	0	0	0	0	0	= $20H$ with port address $80H$

The nonspecific EOI command can be used in the fully nested mode because it always resets the bit of the highest priority; however, in other priority modes, it may reset the wrong bit. It is always safe to use a specific EOI command; the format is as follows:

**FIGURE 15.32**

Interrupt Process: Fully Nested Mode

SOURCE: John Beaston, *Using the Programmable Interrupt Controller*, Intel Application Note AP-31 (Santa Clara, Calif.: Intel Corporation, May 1978), p. 10.

15.6 DIRECT MEMORY ACCESS (DMA) AND THE 8237 DMA CONTROLLER

Direct Memory Access is an I/O technique commonly used for high-speed data transfer; for example, data transfer between system memory and a floppy disk. In status check I/O and interrupt I/O, data transfer is relatively slow because each instruction needs to be fetched and executed. In DMA, the MPU releases the control of the buses to a device called a DMA controller. The controller manages data transfer between memory and a peripheral under its control, thus bypassing the MPU. Conceptually, this is an important I/O technique; it introduces two new signals available on the 8085—HOLD and HLDA (Hold Acknowledge).

- HOLD—This is an active high input signal to the 8085 from another master requesting the use of the address and data buses. After receiving the Hold request, the MPU relinquishes the buses in the following machine cycle. All buses are tri-stated and the Hold Acknowledge (HLDA) signal is sent out. The MPU regains the control of the buses after HOLD goes low.
- HLDA (Hold Acknowledge)—This is an active high output signal indicating that the MPU is relinquishing control of the buses.

A DMA controller uses these signals as if it were a peripheral requesting the MPU for the control of the buses. The MPU communicates with the controller by using the Chip Select line, buses, and control signals. However, once the controller has gained control, it plays the role of a processor for data transfer. To perform this function the DMA controller should have

1. a data bus,
2. an address bus,
3. Read/Write control signals, and
4. control signals to disable its role as a peripheral and to enable its role as a processor.

This process is called switching from the slave mode to the master mode.

For all practical purposes, the DMA controller is a processor capable only of copying data at high speed from one location to another location. As an illustration, a programmable DMA controller, the Intel 8237, is described below.

15.6.1 The 8237 DMA Controller

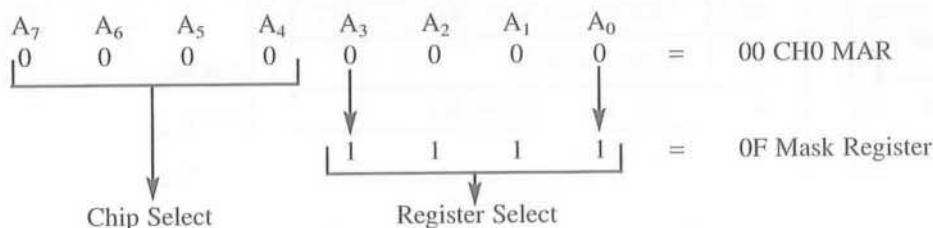
The 8237 is a programmable Direct Memory Access controller (DMA) housed in a 40-pin package. It has four independent channels with each channel capable of transferring 64K bytes. It must interface with two types of devices: the MPU and peripherals such as floppy disks. As mentioned earlier, the DMA plays two roles in a given system: It is an I/O to the microprocessor (slave mode) and it is a data transfer processor to peripherals such as floppy disks (master mode). Many of its signals that are input in the I/O mode become outputs in the processor mode. It also needs additional signal lines to communicate

with the addresses of 64K data bytes, and these signals must be generated externally by using latches and buffers. The 8237 is a complex device. To maintain clarity, the following discussion is divided into five segments: DMA channels and interfacing, DMA signals, system interface, programming, and DMA execution. The specification details of the 8237 are included in Appendix D.

DMA CHANNELS AND INTERFACING

Figure 15.33 shows a logical pin out and internal registers of the 8237. It also shows the interface with the 8085 using a 3-to-8 decoder.

The 8237 has four independent channels, CH0 to CH3. Internally, two 16-bit registers are associated with each channel: One is used to load a starting address of the byte to be copied and the second is used to load a count of the number of bytes to be copied. Figure 15.33 shows eight such registers that can be accessed by the MPU. The addresses of these registers are determined by four address lines, A_3 to A_0 , and the Chip Select (CS) signal. Address 0000 on lines A_3 – A_0 selects CH0 Memory Address Register (MAR) and address 0001 selects the next register, CH0 Count. Similarly, all the remaining registers are selected in sequential order. The last eight registers are used to write commands or read status as shown. In Figure 15.33, the MPU accesses the DMA controller by asserting the signal Y_0 of the decoder. Therefore, the addresses of these internal registers range from 00 to 0FH as follows:



DMA SIGNALS

In Figure 15.33, signals are divided into two groups: (1) one group of signals shown on the left of the 8237 is used for interfacing with the MPU; (2) the second group shown on the right-hand side of the 8237 is for communicating with peripherals. Some of these signals are bidirectional and their functions are determined by the DMA mode of operation (I/O or processor). The signals that are necessary to understand the DMA operation are explained as follows; the remaining signals are listed in Appendix D.

- **DREQ0–DREQ3—DMA Request:** These are four independent, asynchronous input signals to the DMA channels from peripherals such as floppy disks and the hard disk. To obtain DMA service, a request is generated by activating the DREQ line of the channel.
 - **DACK0–DACK3—DMA Acknowledge:** These are output lines to inform the individual peripherals that a DMA is granted. DREQ and DACK are equivalent to hand-shake signals in I/O devices.

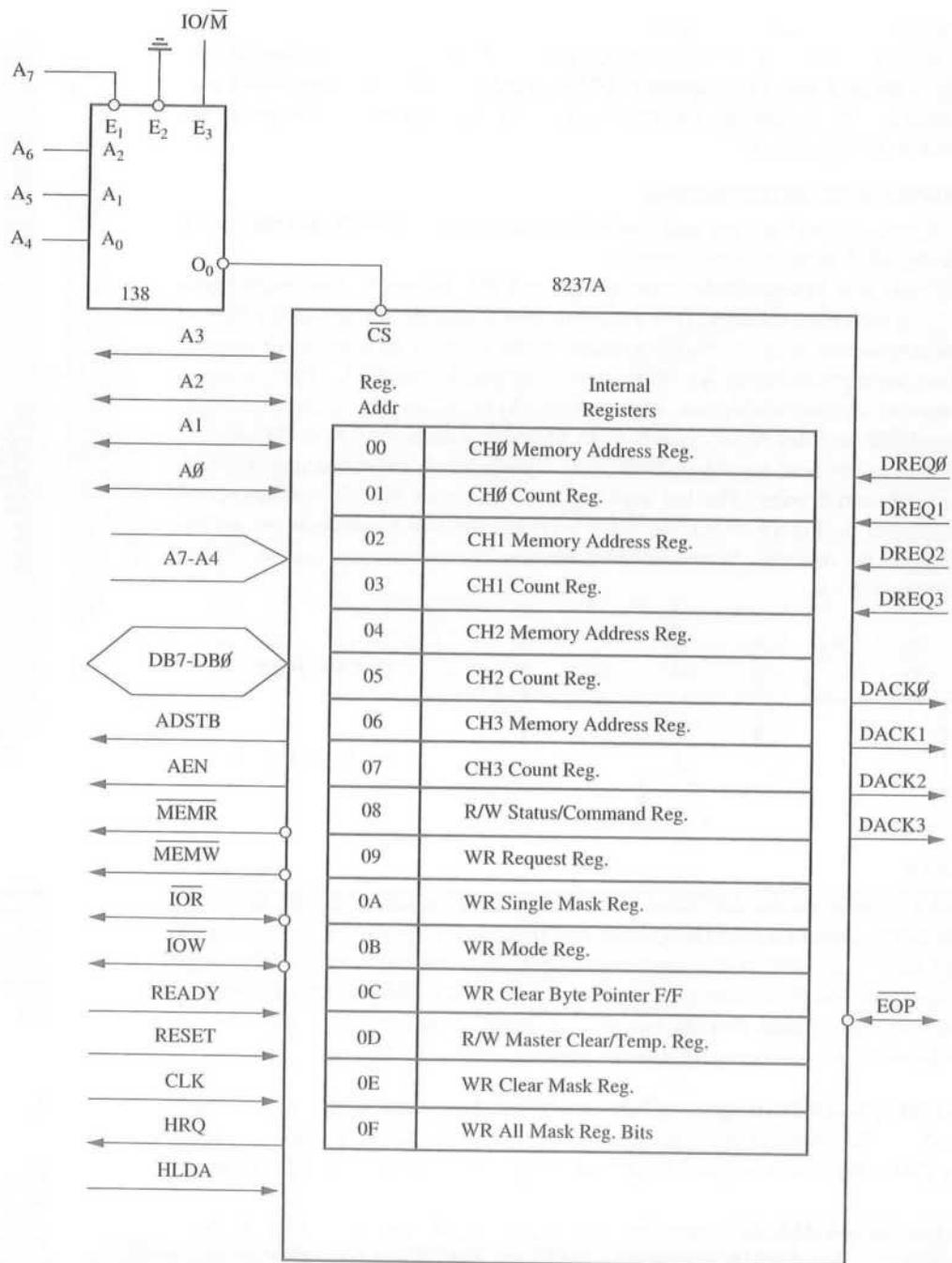


FIGURE 15.33
8237A—DMA Controller with Internal Registers

- **AEN and ADSTB—Address Enable and Address Strobe:** These are active high output signals that are used to latch a high-order address byte to generate a 16-bit address.
- **MEMR and MEMW—Memory Read and Memory Write:** These are output signals used during the DMA cycle to write and read from memory.
- **A₃–A₀ and A₇–A₄—Address:** A₃–A₀ are bidirectional address lines. They are used as inputs to access control registers as shown in the previous section. During the DMA cycle, these lines are used as output lines to generate a low-order address that is combined with the remaining address lines A₇–A₄.
- **HRQ and HLDA—Hold Request and Hold Acknowledge:** HRQ is an output signal used to request the MPU control of the system bus. After receiving the HRQ, the MPU completes the bus cycle in process and issues the HLDA signal.

SYSTEM INTERFACE

The DMA is used to transfer data bytes between I/O (such as a floppy disk) and system memory (or from memory to memory) at high speed. It includes eight data lines, four control signals (IOR, IOW, MEMR, and MEMW), and eight address lines (A₇–A₀). However, it needs 16 address lines to access 64K bytes. Therefore, an additional eight lines must be generated as shown in Figure 15.34.

When a transfer begins, the DMA places the low-order byte on the address bus and the high-order byte on the data bus and asserts AEN (Address Enable) and ADSTB (Address Strobe). These two signals are used to latch the high-order byte from the data bus; thus, it places the 16-bit address on the system bus. After the transfer of the first byte, the latch is updated when the lower byte generates a carry (or borrow). Figure 15.34 shows two latches: one latch (373 #1) to latch a high-order address from the data bus by using the AEN and ADSTB signals, and the second latch (373 #2) to demultiplex the 8085 bus and generate the low-order address bus by using the ALE (Address Latch Enable from the 8085) signal. The AEN signal is connected to the OE signal of the second latch to disable the low-order address bus from the 8085 when the first latch is enabled to latch the high-order byte of the address.

PROGRAMMING THE 8237

To implement the DMA transfer, the 8237 should be initialized by writing into various control registers discussed earlier in the DMA channels and interfacing section. To initialize the 8237, the following steps are necessary.

1. Write a control word in the Mode register that selects the channel and specifies the type of transfer (Read, Write, or Verify) and the DMA mode (block, single-byte, etc.).
2. Write a control word in the Command register that specifies parameters such as priority among four channels, DREQ and DACK active levels, and timing, and enables the 8237.
3. Write the starting address of the data block to be transferred in the channel Memory Address Register (MAR).
4. Write the count (the number of the bytes in the data block) in the channel Count register.

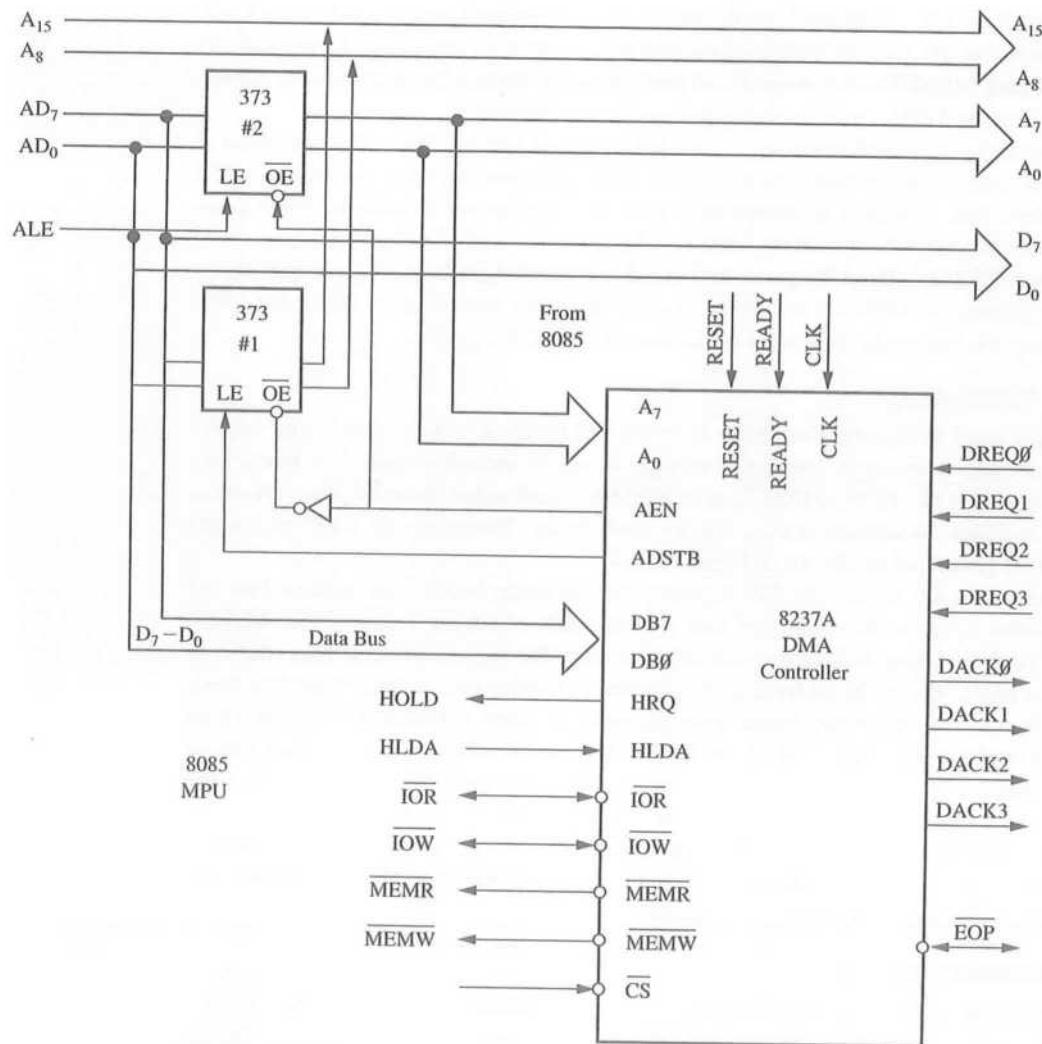


FIGURE 15.34
Interfacing 8237A—DMA Controller with the 8085

These steps are illustrated in the following example.

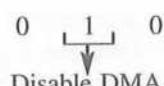
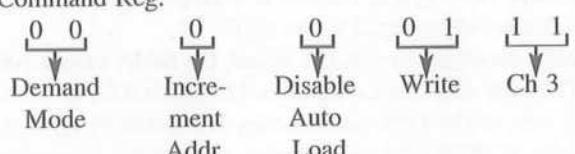
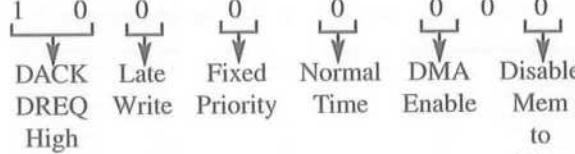
**Example
15.8**

Write initialization instructions for the DMA controller in Figure 15.33 to meet the following specifications. Use the same port (register) addresses (00 to OFH) as in Figure 15.33.

1. Disable the DMA controller and begin writing initialization instructions.
2. Initialize Channel #3 (CH3) to transfer 1K of bytes from the system memory to the floppy disk assigned to CH3.

3. The starting address of the data block is 4075H and subsequent data bytes have memory addresses in increasing order.
4. The Command parameters should be: normal timing, fixed priority, late write. DREQ and DACK are both active low.
5. Set up the demand mode whereby the DMA can complete the data transfer without any interruption.

The initialization instructions to set up the DMA controller are as follows (for a detailed explanation see Appendix D):

MVI A, 00000100B ;Command:	0	0	0	0	<u>1</u>	0	0	Solution
								
OUT 08H ;Send to Command Reg.								
MVI A, 00000111B ;Mode:	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>	<u>1</u>	<u>0</u>	<u>1</u>	
								
	Demand Mode	Increment Addr.	Disable Auto Load	Write	Ch 3			
OUT 0BH ;Send to Mode Reg.								
MVI A, 75H ;Low-order byte of starting address								
OUT 06H ;Output to CH3 Memory Address Reg.								
MVI A, 40H ;High-order byte of starting address								
OUT 06H ;Output to CH3 Memory Address Reg.								
MVI A, FFH ;Low-order byte of the count 03FFH								
OUT 07H ;Output to CH3 Count Reg.								
MVI A, 03H ;High-order byte of the count 03FFH								
OUT 07H ;Output to CH3 Count Reg.								
MVI A, 10000000B ;Command:	<u>1</u>	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>	<u>0</u>	
								
	DACK	Late	Fixed Priority	Normal Time	DMA Enable	Disable Mem to Mem		
OUT 08H ;Send to Command Reg.								

DMA EXECUTION

The process of data transfer from the peripheral to the system memory under the DMA controller can be classified under two modes: the slave mode and the master mode.

Slave Mode In the slave mode, the DMA controller is treated as a peripheral, using the following steps:

1. The MPU selects the DMA controller through Chip Select.
2. The MPU writes the control words as illustrated in Example 15.8 in channel registers and command/status registers by using control signals $\overline{\text{IOW}}$ and $\overline{\text{IOR}}$.

In this mode, the output signals of the 8237, such as A_7-A_4 , $\overline{\text{MEMW}}$, and $\overline{\text{MEMR}}$, are in tri-state.

Master Mode After the initialization, the 8237 in master mode keeps checking for a DMA request, and the steps in data transfer can be listed as follows:

1. When the peripheral is ready for data transfer, it sends a high signal to DRQ.
2. When the DRQ has been received and the channel enabled, the control logic sets HRQ (Hold Request) high. (HRQ is connected to the HOLD signal of the 8085.)
3. In the next cycle, the MPU relinquishes the buses and sends the HLDA (Hold Acknowledge) signal to the 8237.
4. After receiving the HLDA signal, the DMA asserts AEN (Address Enable) signal high. The high AEN signal disables 373 Latch #2, thus disconnecting the demultiplexed bus A_7-A_0 of the MPU and enables 373 Latch #1 through an inverter. Next, the DMA asserts ADSTB (Address Strobe) high that is connected to Latch Enable (LE) of 373 Latch #1 and places the contents of the data bus, which is a high-order byte of the starting address, on $A_{15}-A_8$. At the same time, the DMA also outputs the low order address A_7-A_0 on the low-order address bus.
5. When the entire address $A_{15}-A_0$ is available on the address bus, the DMA sends DACK to the peripheral.
6. The DMA controller continues the data transfer by asserting the necessary control signals ($\overline{\text{IOR}}$, $\overline{\text{IOW}}$, $\overline{\text{MEMR}}$, or $\overline{\text{MEMW}}$) until DACK remains high.
7. At the end of the data transfer, the DMA asserts EOP (End of Process) signal low that can be used to inform the peripheral that the data transfer is complete. The DMA data transfer can also be terminated by sending a low signal to EOP from outside.

SUMMARY

In this chapter, four programmable devices were described: the 8255A (PPI), the 8254 (timer), the 8259A (interrupt controller), and the 8237 (DMA controller). These are general-purpose devices, and each is designed to serve different purposes in the I/O communication process. The common element among them is that the functions of these devices can be programmed by writing instructions in their control registers. Applications of these devices were demonstrated with illustrations and examples.

QUESTIONS, PROBLEMS, AND PROGRAMMING ASSIGNMENTS

1. List the operating modes of the 8255A Programmable Peripheral Interface.
2. Specify the handshake signals and their functions if port A of the 8255A is set up as an output port in Mode 1.
3. Specify the bit of a control word for the 8255, which differentiates between the I/O mode and the BSR mode.
4. Specify the two control words that are necessary to set bit PC₆ (assume that the other ports are not being used).
5. Port A of the 8255A is set up in Mode 1, and the status word is read as 18H. Is there an error in the status word?
6. Refer to Section 15.1.4. Write subroutines HEATON and HEATOFF.
7. Describe a software technique that can be used to check whether the fan or the heater is already on or off.
8. List the necessary conditions to generate INTR when port A of the 8255A is set up as an output port in Mode 1.
9. Write necessary software to transfer 100 bytes of data from the slave MPU to the master MPU, using the status check I/O (see Figure 15.19).
10. Connect the INTR signal to RST 6.5 (8085 system) to interrupt the master MPU when data transfer is required. Modify the master program to implement data transfer under the interrupt I/O mode.

Hints: The main program should enable INT_{E1} and INT_{E2}, using the BSR mode. The Interrupt service routine should verify whether it is a Read or a Write request.

11. Specify the conditions to start the timer 8254.
12. List the major components of the 8259A interrupt controller and explain their functions.
13. Explain how the 8237 DMA controller transfers 64K bytes of data per channel with eight address lines.
14. Write initialization instructions for the 8255A to set up

- port A as an output port in Mode 0.
- port B as an output port in Mode 1 for interrupt I/O.
- port C_U as an output port in Mode 0.

15. In Figure 15.5, connect the system address lines A₉ and A₈ to A₁ and A₀ lines of the 8255A, respectively. Specify the port addresses.
16. Set up the 8254 as a square-wave generator with a 1 ms period, if the input frequency to the 8254 is 1 MHz.
17. Design a five-minute clock (timer) using the 8254 and the interrupt technique. Display minutes and seconds.

18. Write initialization instructions for the 8259A Interrupt Controller to meet the following specifications:

- Interrupt vector address: 2090H
- Call address interval of eight bits
- Nested mode

EXPERIMENTAL ASSIGNMENTS

1.
 - a. Connect the circuit as shown in Figure 15.7.
 - b. Write initialization instructions, and store binary readings in memory buffer for five different analog signals.
 - c. Modify the circuit to record the data, using the interrupt RST 6.5.
 - d. Modify the initialization instructions.
 - e. Write the service routine to record the data.
 - f. Store data in memory for five different analog signals.
2.
 - a. Set up the 8254 timer as shown in Figure 15.26.
 - b. Write instructions to obtain a square wave with the period of 500 μ s.
 - c. Enter the instructions and execute the program.
 - d. Measure the square wave output on an oscilloscope.
 - e. Calculate the pulse width if the count is 8000H.
 - f. Load the count in step e, start the counter, and measure the pulse width of the output.

16

Serial I/O and Data Communication

The 8085 microprocessor is a parallel device; it transfers eight bits of data simultaneously over eight data lines. This is the **parallel I/O mode** discussed in previous chapters. However, in many situations, the parallel I/O mode is either impractical or impossible. For example, parallel data communication over a long distance can become very expensive. Similarly, devices such as a CRT terminal or a cassette tape are not designed for parallel I/O. In these situations, the serial I/O mode is used, whereby one bit at a time is transferred over a single line.

In serial transmission (from the MPU to a peripheral), an 8-bit parallel word should be converted into a stream of eight serial bits; this is known as **parallel-to-serial conversion**. After the conversion, one bit at a time is transmitted over a single line at a given rate called the baud (bits per second). On the other hand, in serial reception, the MPU receives a stream of eight bits, and they should be converted into an 8-bit parallel word; this is known as **serial-to-parallel conversion**. In addition to the conver-

sion, information such as the beginning and the end of transmission and error check is necessary in serial transmission. This process raises several questions about the serial I/O mode.

- In serial I/O, how does the MPU identify a peripheral and what are the conditions of data transfer: unconditional, using the status check, or using the interrupt?
- What are the codes for alphanumeric data?
- What are the requirements of transmission: synchronization, speed, error check, etc.?
- What are the standards for interfacing various types of equipment?
- What are the trade-offs between software and hardware approaches in implementing serial I/O?

These questions concern basic concepts in serial data communication and are discussed in this chapter. The illustrations include both software- and hardware-controlled serial I/O, as well as uses of the SOD (Serial Output Data) and SID (Serial Input

Data) pins—specially designed signals for serial I/O in the 8085.

OBJECTIVES

- Explain how data transfer occurs in the serial I/O mode and how it differs from the parallel I/O mode.
- Explain the terms *synchronous* and *asynchronous transmission*, *simplex*, *half* and *full duplex transmission*; *ASCII code*; *baud (rate)*; and *parity check*.
- Explain how data bits are transmitted (or received) in the asynchronous format, and calculate the delay required between two successive bits for a given baud.
- Explain the RS-232C serial I/O standard, and compare it with the RS-422A and 423A standards.
- Write instructions to transmit and receive data using the serial I/O lines (SID and SOD) in an 8085 system.
- Explain the block diagram and the functions of each block of the Intel 8251 USART (Programmable Communication Interface).
- Design an interfacing circuit using the 8251, and write initialization instructions to set up data communication between a microcomputer and a serial peripheral.

16.1 BASIC CONCEPTS IN SERIAL I/O

The basic concepts concerning the serial I/O mode can be classified into the following categories as discussed in the next sections.

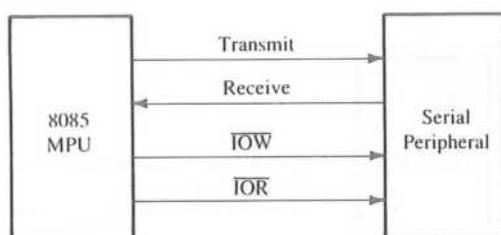
1. Interfacing requirements
2. Alphanumeric codes
3. Transmission format
4. Error checks in data communication
5. Data communication over telephone lines
6. Standards in serial I/O
7. Software vs. programmable hardware approaches

16.1.1 Interfacing Requirements

The interfacing requirements for a serial I/O peripheral are the same as for a parallel I/O device. The microprocessor identifies the peripheral through a port address and enables it using the Read or Write control signal. The primary difference between parallel I/O and serial I/O is in the number of lines used for data transfer—the parallel I/O uses the entire data bus and the serial I/O uses one data line. Figure 16.1 shows a typical configuration of serial I/O transmission; the MPU selects the peripheral through Chip Select and uses the control signals Read to receive data and Write to transmit data. The address decoding can be either peripheral I/O or memory-mapped I/O. Similarly, a serial peripheral can be interfaced under either program control (status check) or interrupt control.

FIGURE 16.1

Block Diagram: Serial I/O
Interfacing



16.1.2 Alphanumeric Codes

A computer is a binary machine; to communicate with the computer in alphabetic letters and decimal numbers, translation codes are necessary. The commonly used code is known as ASCII, the American Standard Code for Information Interchange. It is a 7-bit code with 128 (2^7) combinations, and each combination from 00H to 7FH is assigned to a letter, a decimal number, a symbol, or a machine command (see Appendix E). For example, hexadecimals 30H to 39H represent numerals 0 to 9; 41H to 5AH represent capital letters A through Z; 21H to 2FH represent various symbols; and the initial codes 00H to 1FH represent machine commands such as Carriage Return (CR) or Line Feed (LF). Devices that use ASCII characters include ASCII terminals, teletype machines (TTY), and printers. When key 9 is pressed on an ASCII terminal, the computer receives 39H in binary, and the system programs (as shown in Chapter 10) translate ASCII characters into appropriate binary or BCD numbers.

This topic was discussed briefly in Chapter 1 and repeated here because of its direct relevance to serial communication.

16.1.3 Transmission Format

A transmission format is concerned with issues such as synchronization, direction of data flow, speed, errors, and medium of transmission (telephone lines, for example). These topics are described briefly below.

SYNCHRONOUS VS. ASYNCHRONOUS TRANSMISSION

Serial communication occurs in either synchronous or asynchronous format. In the synchronous format, a receiver and a transmitter are synchronized; a block of characters is transmitted along with the synchronization information, as in Figure 16.2(a). This format is generally used for high-speed transmission (more than 20 k bits/second).

The asynchronous format is character-oriented. Each character carries the information of the Start and the Stop bits, shown in Figure 16.2(b). When no data are being transmitted, a receiver stays high at logic 1, called Mark; logic 0 is called Space. Transmission begins with one Start bit (low), followed by a character and one or two Stop bits (high). This is also known as **framing**. Figure 16.2(b) shows the transmission of 11 bits for an ASCII character in the asynchronous format: one Start bit, eight character bits, and two Stop bits. The format shown in Figure 16.2(b) is similar to Morse code, but the dots and dashes are replaced by logic 0s and 1s. The asynchronous format is generally used in low-speed transmission (less than 20 k bits/second).

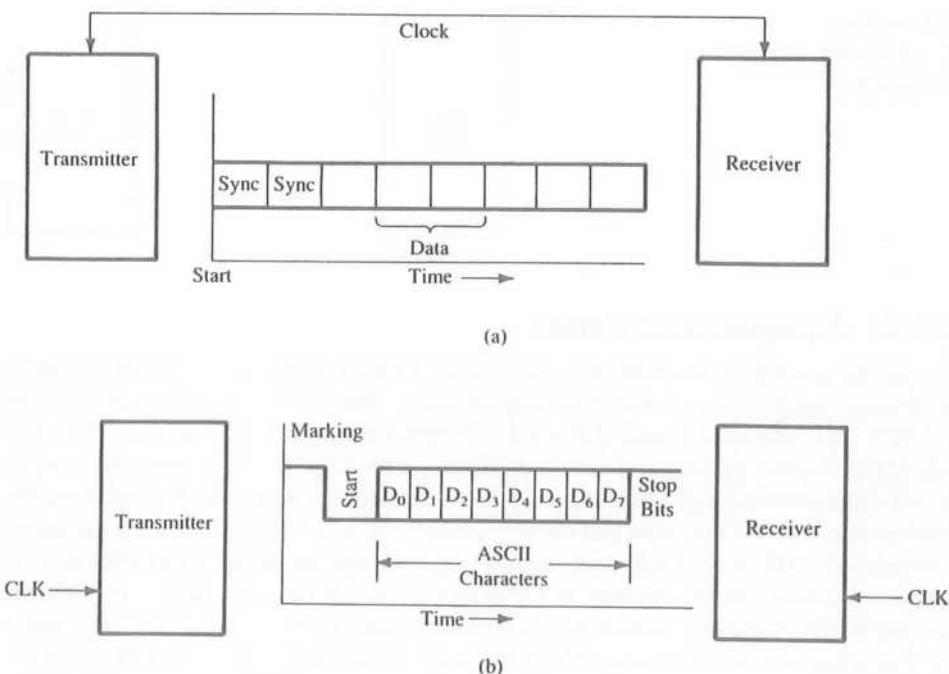


FIGURE 16.2
Transmission Format: Synchronous (a) and Asynchronous (b)

SIMPLEX AND DUPLEX TRANSMISSION

Serial communication also can be classified according to the direction and simultaneity of data flow.

In **simplex transmission**, data are transmitted in only one direction. A typical example is transmission from a microcomputer to a printer.

In **duplex transmission**, data flow in both directions. However, if the transmission goes one way at a time, it is called **half duplex**; if it goes both ways simultaneously, it is called **full duplex**. Generally, transmission between two computers or between a computer and a terminal is full duplex.

RATE OF TRANSMISSION (BAUD)

In parallel I/O, data bits are transferred when a control signal enables the interfacing device; the transfer takes place in less than three T-states. However, in serial I/O, one bit is sent out at a time; therefore, how long the bit stays on or off is determined by the speed at which the bits are transmitted. Furthermore, the receiver should be set up to receive the bits at the same rate as the transmission; otherwise, the receiver may not be able to differentiate between two consecutive 0s or 1s.

The rate at which the bits are transmitted—bits/second—is called a baud; technically, however, it is defined as the number of signal changes/second. Each piece of equip-

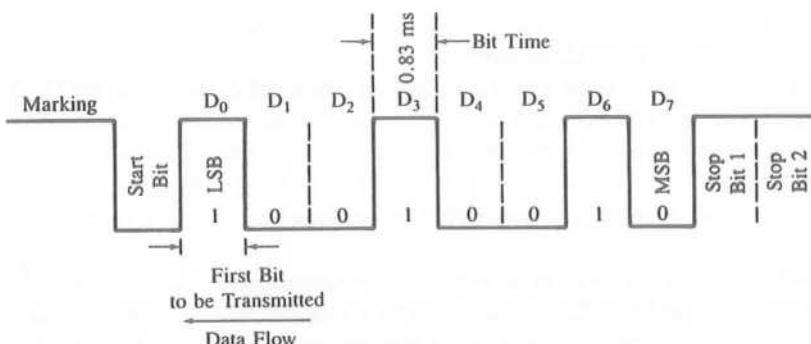


FIGURE 16.3

Serial Bit Format for ASCII Character I at 1200 Baud

ment has its own baud requirement. For example, a teletype (TTY) generally runs at 110 baud. However, in most terminals and printers, the baud is adjustable, typically, in the range of 50 to 9600 baud. Figure 16.3 shows how the ASCII character I (49H) will be transmitted with 1200 baud with the framing information of one Start and two Stop bits. The transmission begins with an active low Start bit, followed by the LSB-bit D₀. The bit time—the delay between any two successive bits—is 0.83 ms; this is determined by the baud as follows:

$$\begin{aligned} 1200 \text{ bits} &= 1 \text{ second} \\ \text{For 1 bit} &= 1/1200 = 0.83 \text{ ms} \end{aligned}$$

Therefore, to transmit one character, a parallel byte (49H) should be converted into a stream of 11 bits by adding framing bits (one Start and two Stop bits), and each bit must be transmitted at the interval of 0.83 ms. This can be implemented either through software or through programmable hardware chips. To receive a character in the serial mode, the process is reversed—one bit at a time is received and the bits are converted into a parallel word.

16.1.4 Error Checks in Data Communication

During transmission, various types of errors can occur. For example, data bits may change because of noise or can be misunderstood by the receiver because of differences in receiver and transmitter clocks. These errors need to be checked; therefore, additional information for error checking is sent during the transmission. The receiver can check the received data against the error check information, and if an error is detected, the receiver can request the retransmission of that data segment. Three methods are generally in common practice; they are parity check, checksum, and cyclic redundancy check.

PARITY CHECK

This is used to check each character by counting the number of 1s in the character; in the ASCII code transmission, bit D₇ is used to transmit parity check information. The tech-

nique is based on the principle that, in a given system, each character is transmitted with either an even number of 1s or an odd number of 1s.

In an even parity system, when a character has an odd number of 1s, the bit D₇ is set to 1 and an even number of 1s is transmitted. For example, the code for the character I is 49H (0100 1001), with three 1s. When the character I is transmitted in an even parity system, the bit D₇ is set to 1, making the code C9H (1100 1001). On the other hand, in an odd parity system, the character I is transmitted by keeping bit D₇ = 0; thus, the code remains 49H.

In the 8085 system, the parity check is easy to implement and detect because the 8085 has the parity flag, and this flag can be used to check parity information in each character. However, the parity check cannot detect multiple errors in any given character.

CHECKSUM

The checksum technique is used when blocks of data are transferred. It involves adding all the bytes in a block without carries. Then, the 2's complement of the sum (negative of the sum) is transmitted as the last byte. The receiver adds all the bytes, including the 2's complement of the sum; thus, the result should be zero if there is no error in the block (refer to Section 16.4.4 for an illustration of the checksum technique).

CYCLIC REDUNDANCY CHECK (CRC)

This technique is commonly used when data are transferred from and to a floppy disk and in a synchronous data communication. The technique is based on mathematical relationships of polynomials. A stream of data can be represented as a polynomial that is divided by a constant polynomial, and the remainder, unique to that set of bits, is generated. The remainder is sent out as a check for errors. The receiver checks the remainder to detect an error in the transmission. This is a somewhat complex technique and will not be discussed here.

16.1.5 Data Communication over Telephone Lines

The serial I/O technique can be used to send data over long distance through telephone lines. However, telephone lines are designed to handle voice; the bandwidth of telephone lines ranges from 300 Hz to 3300 Hz. The digital signal with rise time in nanoseconds requires a bandwidth of several megahertz. Therefore, data bits should be converted into audio tones; this is accomplished through modems.

A modem (Modulator/Demodulator) is a circuit that translates digital data into audio tone frequencies for transmission over the telephone lines and converts audio frequencies into digital data for reception. At the present time, modems are available that can transfer data at rates of 300–2400 bps (bits per second). Generally, two types of modulation techniques are used: frequency-shift keying (FSK) for low-speed modems and phase-shift keying (PSK) for high-speed modems.

Computers can exchange information over telephone lines by using two modems—one on each side (Figure 16.4). A calling computer (or a terminal), also known as an originator, contacts a receiving computer (also known as answering) through a telephone number, and a communication link is established after control signals have been exchanged between computers and modems.

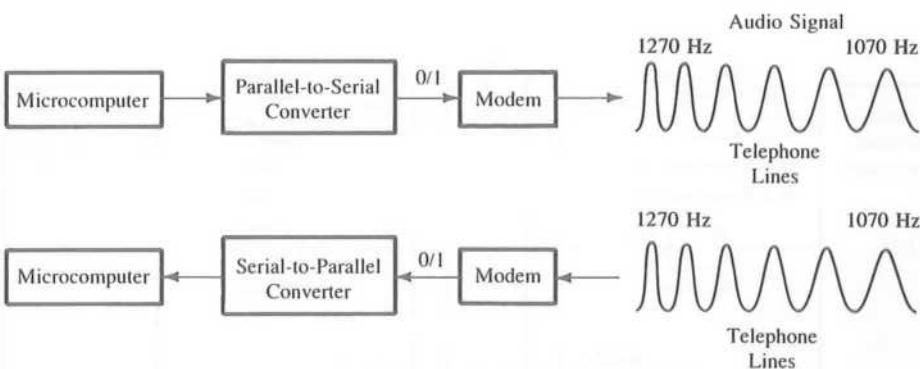


FIGURE 16.4
Communication over Telephone Lines Using Modems

A typical process of communication for a 300 bps modem is shown in Figure 16.4. A parallel word is converted into serial bits; in turn, the originator modem generates two audio frequencies—1070 Hz for logic 0 (Space) and 1270 Hz for logic 1 (Mark). These audio frequencies are transmitted over telephone lines. At the answering end, audio frequencies are converted back into 0s and 1s, and serial bits are converted into a parallel word that can be read by the computer. When the answering-end computer needs to transmit, it transmits on 2025 Hz (Space) and 2225 Hz (Mark).

16.1.6 Standards in Serial I/O

The serial I/O technique is commonly used to interface terminals, printers, and modems. These peripherals and computers are designed and manufactured by various manufacturers. Therefore, a common understanding must exist, among various manufacturing and user groups, that can ensure compatibility among different equipment. When this understanding is defined and generally accepted in industry (and by users), it is known as a standard. A standard is normally defined by a professional organization (such as IEEE—Institute of Electrical and Electronics Engineers); however, occasionally, a widespread practice can become a de facto standard. A standard may include such items as assignment of pin positions for signals, voltage levels, speed of data transfer, length of cables, and mechanical specifications.

In serial I/O, data can be transmitted as either current or voltage. Typically, 20 mA (or 60 mA) current loops are used in teletype equipment. When a teletype is marking or at logic 1, current flows; when it is at logic 0 (or Space), the current flow is interrupted. The advantage of the current loop method is that signals are relatively noise-free and are suitable for transmission over a distance.

When data are transmitted as voltage, the commonly used standard is known as RS-232C. It is defined in reference to Data Terminal Equipment (DTE) and Data Communication Equipment (DCE)—terminal and modem—as shown in Figure 16.5(a); however, its voltage levels are not compatible with TTL logic levels. The rate of data transmission in RS-232C is restricted to a maximum of 20 kbaud and a distance of 50 ft. For high-

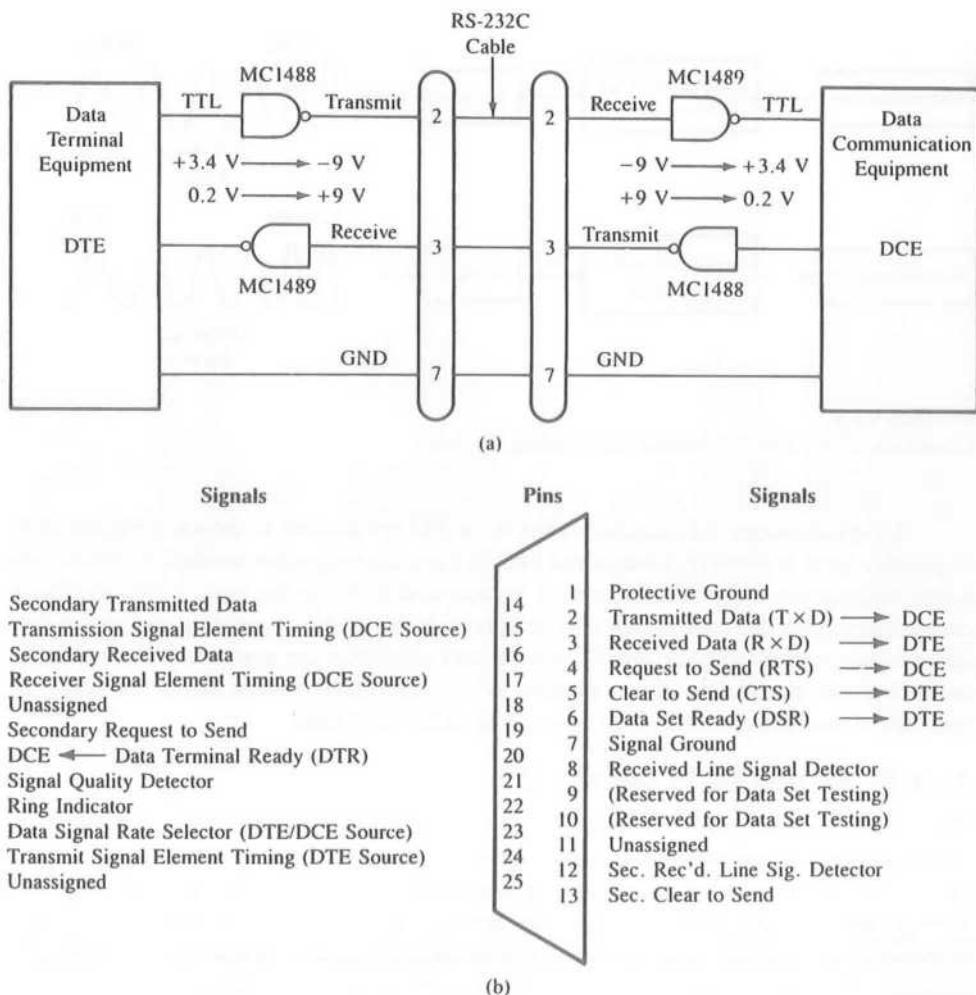


FIGURE 16.5

Minimum Configuration of RS-232C Signals and Voltage Levels (a) and RS-232C Signal Definitions and Pin Assignments (b)

SOURCE: Courtesy of Electronic Industries Association.

speed data transmission, two new standards—RS-422A and RS-423A—have been developed in recent years; however, they are not yet widely used.

To appreciate the difficulties and confusion in this standard, one has to examine its historical background. The RS-232 standard was developed during the initial days of computer timesharing, long before the existence of TTL logic, and its primary focus was to have compatibility between a terminal and a modem. However, the same standard is now being used for communications between computers and peripherals, and the roles of a data terminal and a modem have become ambiguous. Should a computer be considered

a terminal or a modem? The answer is that it can be either. Therefore, the lines used for transmission and reception will differ, depending on the manufacturer's role-definition of its equipment.

RS-232C

Figure 16.5(b) shows the RS-232C 25-pin connector and its signals. The signals are divided into four groups: data signals, control signals, timing signals, and grounds. For data lines, the voltage level +3 V to +15 V is defined as logic 0; from -3 V to -15 V is defined as logic 1 (normally, voltage levels are ± 12 V). This is negative true logic. However, other signals (control and timing) are compatible with the TTL level. Because of incompatibility of the data lines with the TTL logic, voltage translators, called line drivers and line receivers, are required to interface TTL logic with the RS-232 signals, as shown in Figure 16.5(a). The line driver, MC1488, converts logic 1 into approximately -9 V and logic 0 into +9 V, as shown in Figure 16.5(a). Before it is received by the DCE, it is again converted by the line receiver, MC1489, into TTL-compatible logic. This raises the question: If the received signal is to be converted back to the TTL level, what is the reason, in the first place, to convert the transmitted signal to the higher level? The primary reason is that the standard was defined before the TTL levels came into existence; before 1960, most equipment was designed to handle higher voltages. The other reason is that this standard provides a higher level of noise margin—from -3 V to +3 V.

The minimum interface between a computer and a peripheral requires three lines: pins 2, 3, and 7, as shown in Figure 16.5(a). These lines are defined in relation to the DTE; the terminal transmits on pin 2 and receives on pin 3. On the other hand, the DCE transmits on pin 3 and receives on pin 2. Now the dilemma is: How does a manufacturer define the role of its equipment? For example, the user may connect its microcomputer to a serial printer configured as a DTE. Therefore, to remain compatible with the defined signals of the RS-232C, the RS-232 cable must be reconfigured as shown in Figure 16.6(b). In Figure 16.6, the microcomputer is defined as a DTE, and it can be connected to the modem, defined as a DCE, without any modification in the RS-232 cable, as shown in Figure 16.6(a). However, when it is connected to the printer, the transmit and the re-

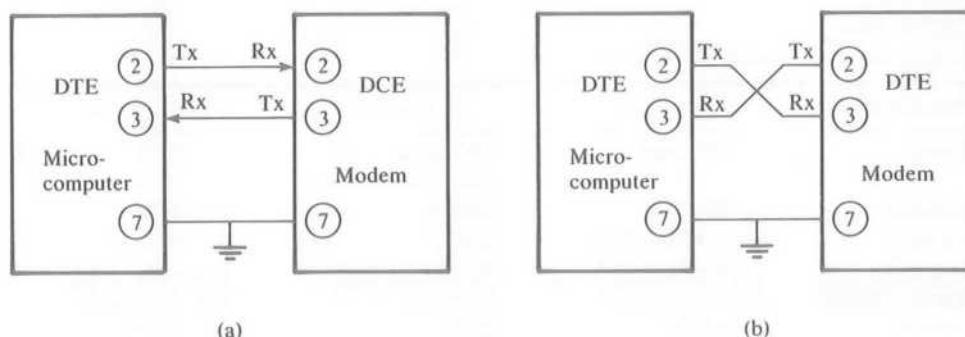


FIGURE 16.6
RS-232C Connections: DTE to DCE (a) and DTE to DTE (b)

ceive lines must be crossed as shown in Figure 16.6(b); this is known as a null-modem connection.

Typically, data transmission with a handshake requires eight lines, listed in Table 16.1. Specific functions of handshake lines differ in different peripherals and, therefore, should be referred to in the manufacturers' manuals.

For high-speed transmission, the standards RS-422A and RS-423A are used. These standards use differential amplifiers to reject noise levels and can transmit data at higher speed with longer cable. The RS-422A allows a maximum speed of 10 kbaud for a 40-ft distance and 100 kbaud for 4000 ft. The RS-423A is limited to 100 kbaud for a 30-ft distance and 10 kbaud for 300 ft. See Table 16.2 for comparison of the three standards: RS-232C, RS-422A, and RS-423A.

TABLE 16.1
RS-232C Signals Used with Handshake Data Communication

Pin No.	Signals ^a	Functions	
2	Transmitted Data	TxD	Output; transmits data from DTE to DCE
3	Received Data	RxD	Input; DTE receives data from DCE
4	Request to Send	RTS	General-purpose output from DTE
5	Clear to Send	CTS	General-purpose input to DTE; can be used as a handshake signal
6	Data Set Ready	DSR	General-purpose input to DTE; can be used to indicate that DCE is ready
7	Signal Ground	GND	Common reference between DTE and DCE
8	Data Carrier Detect	DCD	Generally used by DTE to disable data reception
20	Data Terminal Ready	DTR	Output; generally used to indicate that DTE is ready

^aSignals are referenced to DTE.

TABLE 16.2
Comparison of Serial I/O Standards

Specifications	RS-232C	RS-422A	RS-423A
Speed	20 kbaud	10 Mbaud at 40 ft 100 kbaud at 4000 ft	100 kbaud at 30 ft 1 kbaud at 4000 ft
Distance	50 ft	4000 ft	4000 ft
Logic 0	> +3 to +25 V	B > A ^a	+4 to +6 V
Logic 1	< -3 to -25 V	B < A	-4 to -6 V
Receiver Input			
Voltage	±15 V	±7 V	±12 V

^aB and A are differential input to the op amp.

16.1.7 Review of Serial I/O Concepts and Approaches to Implementation

The serial data transmission can be implemented through either software or programmable I/O devices. Conceptually, the software and the hardware approaches are similar. In the asynchronous data transmission, the steps can be summarized as follows:

1. Inform the receiver of the beginning and the end of the transmission and the parity check.
2. Convert a parallel word into a stream of serial bits.
3. Transmit one bit at a time with appropriate time delay, using one data line of an output port. The time delay is determined by the speed of the transmission.

In data reception, the above process is reversed. The receiver needs to

1. Recognize the beginning of the transmission.
2. Receive serial bits, one at a time, and convert them into a parallel byte.
3. Check for errors and recognize the end of the transmission.

In the software approach, the speed of transmission is set up by using an appropriate delay between the transmission of two consecutive bits, and the entire word is converted into a serial stream by rotating the byte and outputting one bit at a time, using one of the data lines of an output port. The software provides the time delay between the two consecutive bits and adds framing bits and the parity bit; this is discussed in the next section.

In the hardware approach, the above functions are performed by a programmable device (chip). The device contains a parallel-to-serial register and 1-bit output port for transmission and a serial-to-parallel register and 1-bit input port for reception. The rate of transmission and reception is determined by the clock. The programmable chip also includes a control register that can be programmed to add framing and error-check information and to specify the number of bits to be transferred. The microprocessor transfers a parallel byte using the data bus, and the programmable chip performs the remaining functions for serial I/O.

The software approach is suitable for slow-speed asynchronous data communication where timing requirements are not critical. The approach is simple and inexpensive. The hardware approach is suitable for both asynchronous and synchronous formats. The approach is flexible, and chips can be programmed to accommodate changing requirements. In industrial and commercial products, the hardware approach has become almost universal. This chapter includes detailed discussion and illustrations of a widely used serial I/O device: the Intel 8251. However, to understand the basic concepts in serial I/O, the software approach is more suitable than the hardware approach; thus, the software approach is described here prior to discussion of programmable serial I/O devices. We will limit our discussion to the asynchronous communication mode commonly used in the microcomputer. Synchronous communication, being a specialized technique, will not be discussed here.

TABLE 16.3

Summary of Synchronous and Asynchronous Serial Transmission

Format	Synchronous	Asynchronous
Data Format	Groups of characters	One character at a time
Speed	High (20 k bits/s or higher)	20 k bits/s or lower
Framing Information	Sync characters are sent with each group	Start and Stop bits with every character
Implementation	Hardware	Hardware or software
Data Direction	Simplex, Half and Full Duplex	Simplex, Half and Full Duplex

The basic concepts concerning serial I/O, discussed in the previous sections, are summarized in Table 16.3.

16.2

SOFTWARE-CONTROLLED ASYNCHRONOUS SERIAL I/O

In the software-controlled asynchronous serial mode, the program should perform the following tasks, as discussed in the previous section:

1. Output a Start bit.
2. Convert the character to be sent in a stream of serial bits with appropriate delay.
3. Add parity information if necessary.
4. Output one or two Stop bits.

Figure 16.7 shows the accumulator with the code for the ASCII character I, and it is converted into a stream of 11 bits; it includes one Start bit and two Stop bits. After the Start bit, the character bits are transmitted with bit D₀ first and bit D₇ last; in ASCII characters, bit D₇ can be used to add parity information. The bit time—the delay between two successive bits—is determined by the transmission baud (rate). Figure 16.7 shows the transmission with 1200 baud; the delay between the two consecutive bits is 0.833 ms.

Data reception in the serial mode involves the reverse process: receiving one bit at a time and forming an 8-bit parallel word. The receiving program should continue to read the input port until it receives the Start bit and, then, begin to count character bits with appropriate delay.

16.2.1 Serial Data Transmission

Figure 16.8(a) shows a flowchart to transmit an ASCII character; it can be explained in the context of the block diagram shown in Figure 16.7(a). When no character is being transmitted, the transmit line of the output port stays high—in Mark position. The transmission begins with the Start bit, active low. The initialization block of the flowchart includes setting up a counter to count eight character bits; Start and Stop bits are sent out

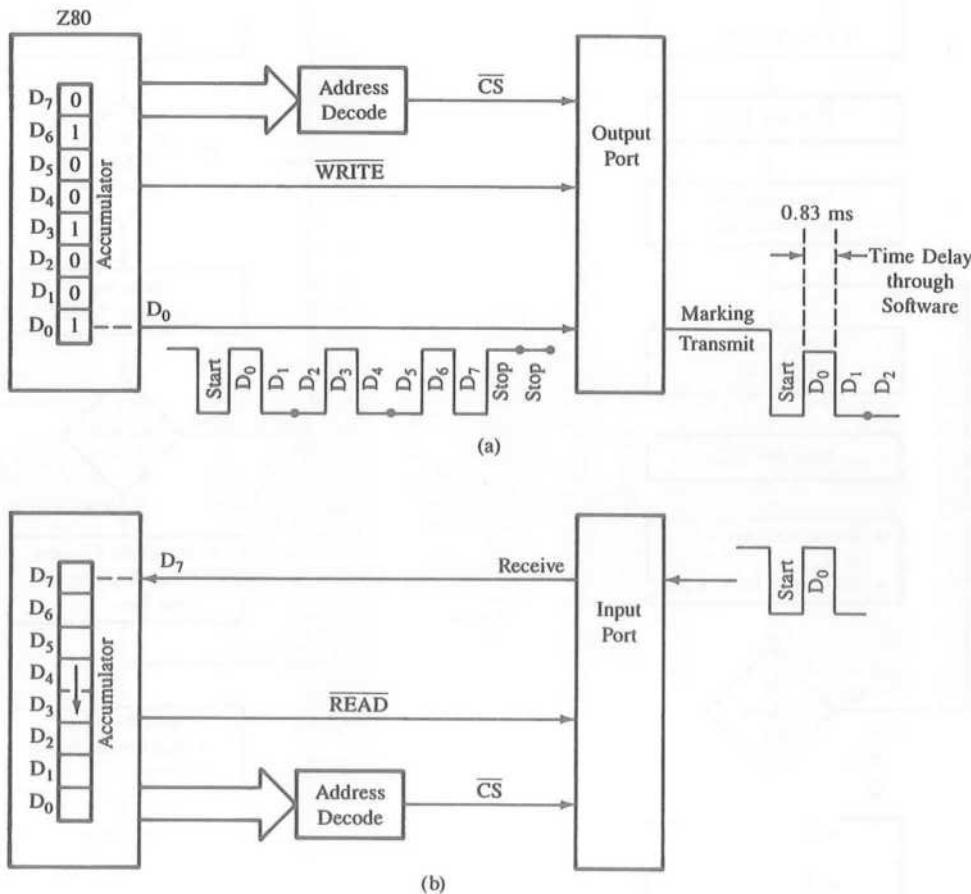


FIGURE 16.7
Serial Data Transmission (a) and Serial Data Reception under Software Control (b)

separately. The program waits for bit time—0.833 ms for 1200 baud—and begins to send one character-bit at a time over the data line D₀ at the interval of 0.833 ms. To get ready for the next bit, the program rotates the bits—for example, D₁ into D₀. It repeats the loop eight times and finally sends out two Stop bits. Assuming that the character is being sent out to a printer, the printer waits until it receives all the bits serially, forms a character, and prints it during the Stop bits. The Stop bits perform two functions: they allow sufficient time for the printer to print the character, and they leave the transmit line in Mark position at the end of the character.

16.2.2 Serial Data Reception

In serial data reception, the program begins with reading the input port. When no character is being received, the input line stays high. The program stays in the loop and contin-

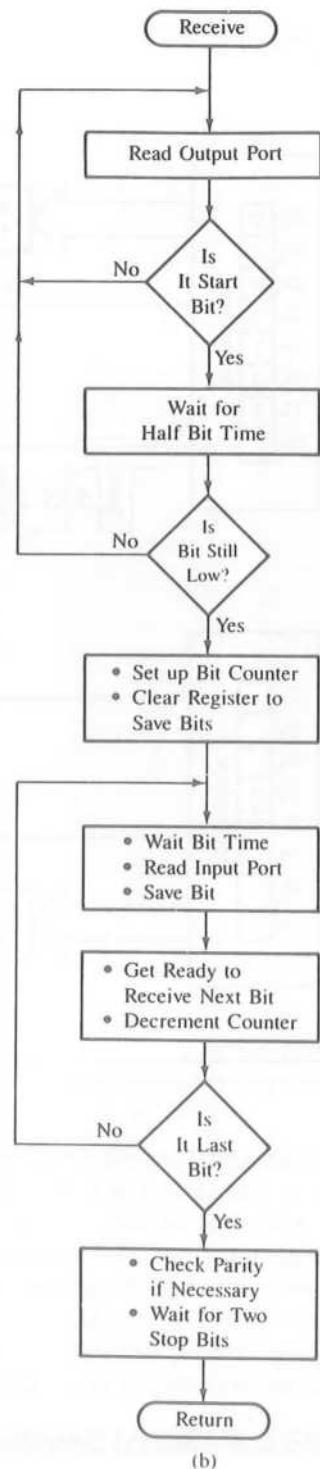
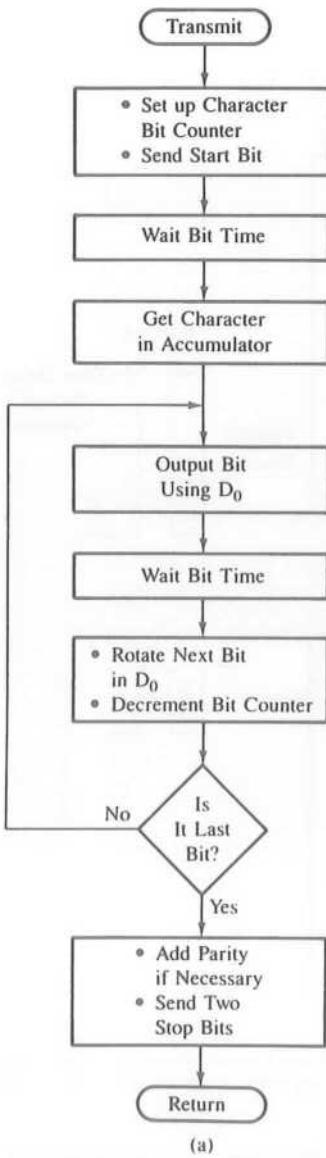


FIGURE 16.8

Flowcharts: Transmission (a) and Reception (b) of an ASCII Character

ues to read the port until the Start bit (active low) is received, as shown in the flowchart of Figure 16.8(b).

When the Start bit is received, it waits for half the bit time and samples the character bits in the middle of the pulse rather than at the beginning to avoid errors in transition. In the next block, it initializes the counter to count eight bits and clears a register to save the partial readings. The program reads the input port at the interval of bit time until it reads all the character bits and ignores the last two bits by just waiting for bit times. The character reception begins also with the LSB; that means the microprocessor will receive bit D₀ first. In Figure 16.7(a), the data line D₇ is used for the reception. The line D₇ provides some programming convenience for serial-to-parallel conversion; the word can be formed by shifting bits to the right whenever a bit is read, and eventually the LSB will reach its proper position.

THE 8085—SERIAL I/O LINES: SOD AND SID

16.3

The 8085 microprocessor has two pins specially designed for software-controlled serial I/O. One is called SOD (Serial Output Data), and the other is called SID (Serial Input Data). Data transfer is controlled through two instructions: SIM and RIM. Instructions SIM and RIM are used for two different processes: interrupt and serial I/O. These instructions have already been discussed in the context of the 8085 interrupt process (Chapter 12); now the focus is on their uses in serial I/O.

SERIAL OUTPUT DATA (SOD)

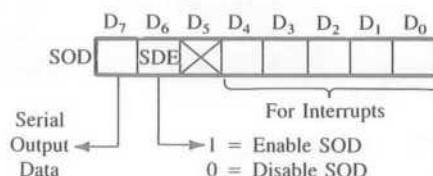
The instruction SIM is necessary to output data serially from the SOD line. It can be interpreted for serial output as in Figure 16.9.

Instructions

MVI A,80H	;Set D ₇ in the accumulator = 1
RAR	;Set D ₆ = 1 and bring Carry into D ₇
SIM	;Output D ₇

In this set of instructions, the serial output line is enabled by rotating 1 into bit position D₆; the instruction SIM outputs the Carry bit through bit position D₇.

FIGURE 16.9
Interpretation of Accumulator
Contents by the SIM Instruction



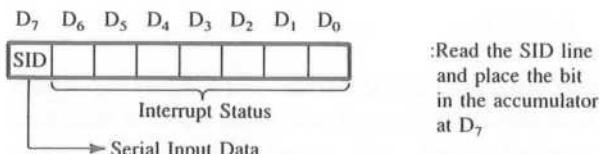


FIGURE 16.10

Interpretation of Accumulator Contents After the RIM Instruction

SERIAL INPUT DATA (SID)

Instruction RIM is used to input serial data through the SID line. Instruction RIM can be interpreted for serial I/O as in Figure 16.10.

In the context of serial I/O, instruction RIM is similar to instruction IN, except RIM reads only one bit and places it in the accumulator at D₇.

The SID and SOD lines in the 8085 eliminate the need for an input port and an output port in the software-controlled serial I/O. Essentially, SID is a 1-bit input port and SOD is a 1-bit output port. Similarly, instruction RIM is equivalent to a 1-bit IN instruction and instruction SIM is equivalent to a conditional 1-bit OUT instruction. The software necessary to implement serial I/O using SID and SOD lines is conceptually similar to that illustrated in Section 16.2.

16.3.1 Illustration: Data Transmission Using the SOD Line

PROBLEM STATEMENT

Write a subroutine to transmit an ASCII character, stored in register B, using the SOD line as a 1-bit output port.

SUBROUTINE*

The following subroutine SODATA (Serial Output Data) follows the flowchart given in Figure 16.8(a), except that it sets the counter for 11 bits and repeats the loop 11 times. The Start and the Stop bits are included with the character, and the Stop bits are set up by using the instruction STC (Set Carry):

	;Input: An ASCII character without parity check in register B	
	;Output: None	
SODATA:	MVI C,0BH	;Set up counter C to count eleven bits
	XRA A	;Reset Carry to 0
NXTBIT:	MVI A,80H	;Set D ₇ = 1 in the accumulator
	RAR	;Bring Carry in D ₇ and set D ₆ = 1
	SIM	;Output D ₇
	CALL BITTIME	;Wait for 9.1 ms
	STC	;Set Carry = 1
	MOV A,B	;Place ASCII character in the accumulator

*This subroutine is adapted from John Wharton, *Using the 8085 Serial Lines*, Application Note AP 29 (Santa Clara, Calif.: Intel Corporation, 1977).

RAR	;Place ASCII D ₀ in the Carry, shift 1 in D ₇ ,
	; and continue shifting for each loop
MOV B,A	;Save
DCR C	;One bit transmitted, decrement counter
JNZ NXTBIT	;If all bits are not transmitted, go back
RET	

Subroutine Description This is an efficient subroutine based on the same serial I/O concepts discussed earlier. However, it uses some programming tricks to output data. To understand these tricks, it is necessary to examine the role of the following instructions, illustrated with an example of the ASCII letter G (47H = 0100 0111):

1. MVI A,10000000 (80H)

RAR

When D₇ is rotated into D₆, the SOD line is enabled for each loop and the contents of the Carry are placed in D₇.

2. STC

MOV A,B

RAR

Instruction STC places 1 into the Carry, and instruction MOV A,B places the ASCII character in the accumulator. Instruction RAR brings 1 from the Carry into D₇ and places the ASCII bit into the Carry.

3. In the second iteration, the first RAR (described in step 1) places the ASCII bit from the Carry into D₇, and 1 from 80H into D₆. Instruction SIM outputs the ASCII bit from bit D₇.
4. The logic 1s, set by instruction STC and saved in register B, are shifted right by one position in every iteration. In the ninth iteration, when ASCII D₇ is sent out, register B will have all 1s from D₀ to D₇. In the last two iterations, logic 1s are sent out as Stop bits.
5. The contents of the accumulator after execution of these instructions are as follows (assume ASCII letter G is being transmitted):

Input from Main

Program:	(B) = 47H	0	1	0	0	0	1	1	1	(B)
Subroutine										
SODATA:		CY	D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
XRA		0	0	0	0	0	0	0	0	(A)
MVI A,80H		C	1	0	0	0	0	0	0	(A)
RAR D ₀		0	0	1	0	0	0	0	0	(A)
SIM →		0				SOD				
							-----	→	OUTPUT	
STC		1								
		↓								
MOV A,B		1	0	1	0	0	0	1	1	(A)
RAR		1	1	0	1	0	0	0	1	(A)

16.3.2 Illustration: Data Reception Using the SID Line

PROBLEM STATEMENT

Write a subroutine to receive an ASCII character using the 8085 SID line.

SUBROUTINE*

The subroutine shown in Figure 16.11, SIDATA (Serial Input Data), is conceptually similar to the flowchart in Figure 16.8(b).

SUBROUTINE DESCRIPTION

The flowchart of this subroutine is slightly different from the flowchart in Figure 16.8(b). The difference is in the procedure of saving a bit and forming a parallel word.

After the first bit (D_0) is read, it is saved in the Carry with instruction RAL. The bit is stored and shifted right by using the instruction RAR. This procedure is repeated eight times. In the ninth iteration, the subroutine returns to the main program, ignoring the last two Stop bits.

16.4

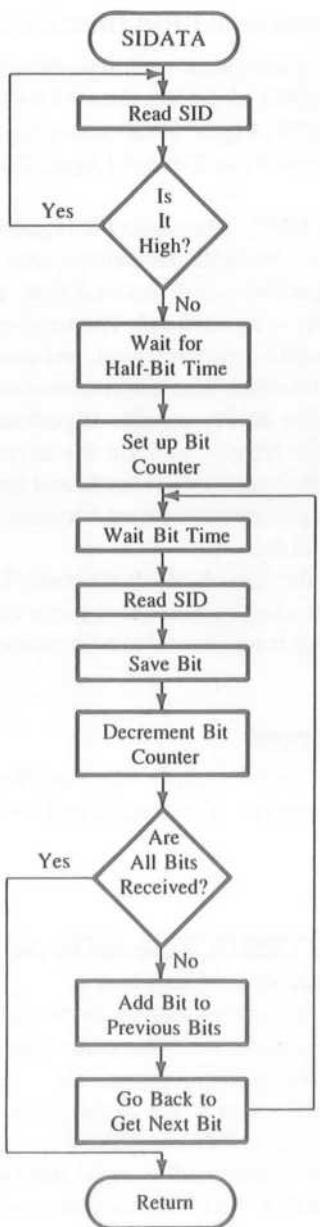
HARDWARE-CONTROLLED SERIAL I/O USING PROGRAMMABLE CHIPS

The hardware approach to serial I/O incorporates the same basic principles and requirements necessary for the software approach. The functions performed separately under software control must be combined in one chip. These functions and requirements for the software approach are summarized below.

1. An input port and an output port are required for interfacing.
2. In data transmission, the MPU converts a parallel word into a stream of serial bits.
3. In data reception, the MPU converts serial bits into a parallel word.
4. Data transfer is synchronized between the MPU and slow-responding peripherals through time delays.

An integrated circuit called **USART** (Universal Synchronous/Asynchronous Receiver/Transmitter) incorporates all the features described above on the chip, as well as many more sophisticated features used for serial data communication. It is a programmable device; i.e., its functions and specifications for serial I/O can be determined by writing instructions in its internal registers. The Intel 8251A USART is a device widely used in serial I/O; the device and its applications are described in the next section.

*This subroutine is adapted from John Wharton, *Using the 8085 Serial Lines*, Application Note AP 29 (Santa Clara, Calif.: Intel Corporation, 1977).



SIDATA:	RIM	;Read SID line
	RAL	;Place D ₇ into CY
	JC SIDATA	;If D ₇ = 1, this is ; not a Start ; bit; go back and ; read again
	CALL HAFBIT	;If D ₇ = 0, this ; is Start bit; ; wait for half-bit ; period
	MVI C,09H	;Counter for ; nine bits
NXTBIT:	CALL BITTIME	;Wait for one bit ; period
	RIM	;Read input bit
	RAL	;Save bit in CY
	DCR C	;One bit is read
	JZ RETURN	;If all bits are ; read, return to ; main program
	MOV A,B	;Place the bits ; saved so far in A
	RAR	;Place the bit ; saved in CY ; into position D ₇ and ; shift all other ; bits by one ; position
	MOV B,A	,Save bits in B
	JMP NXTBIT	;Get the next bit

FIGURE 16.11
Flowchart of Data Reception Using SID Line

16.4.1 The 8251A Programmable Communication Interface

The 8251A is a programmable chip designed for synchronous and asynchronous serial data communication, packaged in a 28-pin DIP. The 8251A is the enhanced version of its predecessor, the 8251, and is compatible with the 8251. Figure 16.12 shows the block diagram of the 8251A. It includes five sections: Read/Write Control Logic, Transmitter, Receiver, Data Bus Buffer, and Modem Control.

The control logic interfaces the chip with the MPU, determines the functions of the chip according to the control word in its register (to be explained below), and monitors the data flow. The transmitter section converts a parallel word received from the MPU into serial bits and transmits them over the TxD line to a peripheral. The receiver section receives serial bits from a peripheral, converts them into a parallel word, and transfers the word to the MPU. The modem control is used to establish data communication through modems over telephone lines. The 8251A is a complex device, capable of performing various functions. For the sake of clarity, this chapter focuses only on the asynchronous mode of serial I/O and excludes any discussion of the synchronous mode and the modem control. The asynchronous mode is often used for data communication between the MPU and serial peripherals such as terminals and floppy disks.

Figure 16.13 shows an expanded version of the 8251A block diagram. The block diagram shows all the elements of a programmable chip; it includes the interfacing signals, the control register, and the status register. The functions of various blocks are described below.

READ/WRITE CONTROL LOGIC AND REGISTERS

This section includes R/W control logic, six input signals, control logic, and three buffer registers: data register, control register, and status register. The input signals to the control logic are as follows.

Input Signals

- CS—Chip Select:** When this signal goes low, the 8251A is selected by the MPU for communication. This is usually connected to a decoded address bus.
- C/D—Control/Data:** When this signal is high, the control register or the status register is addressed; when it is low, the data buffer is addressed. The control register and the status register are differentiated by WR and RD signals, respectively.
- WR—Write:** When this signal goes low, the MPU either writes in the control register or sends output to the data buffer. This is connected to IOW or MEMW.
- RD—Read:** When this signal goes low, the MPU either reads a status from the status register or accepts (inputs) data from the data buffer. This is connected to either IOR or MEMR.
- RESET—Reset:** A high on this input resets the 8251A and forces it into the idle mode.
- CLK—Clock:** This is the clock input, usually connected to the system clock. This clock does not control either the transmission or the reception rate. The clock is necessary for communication with the microprocessor.

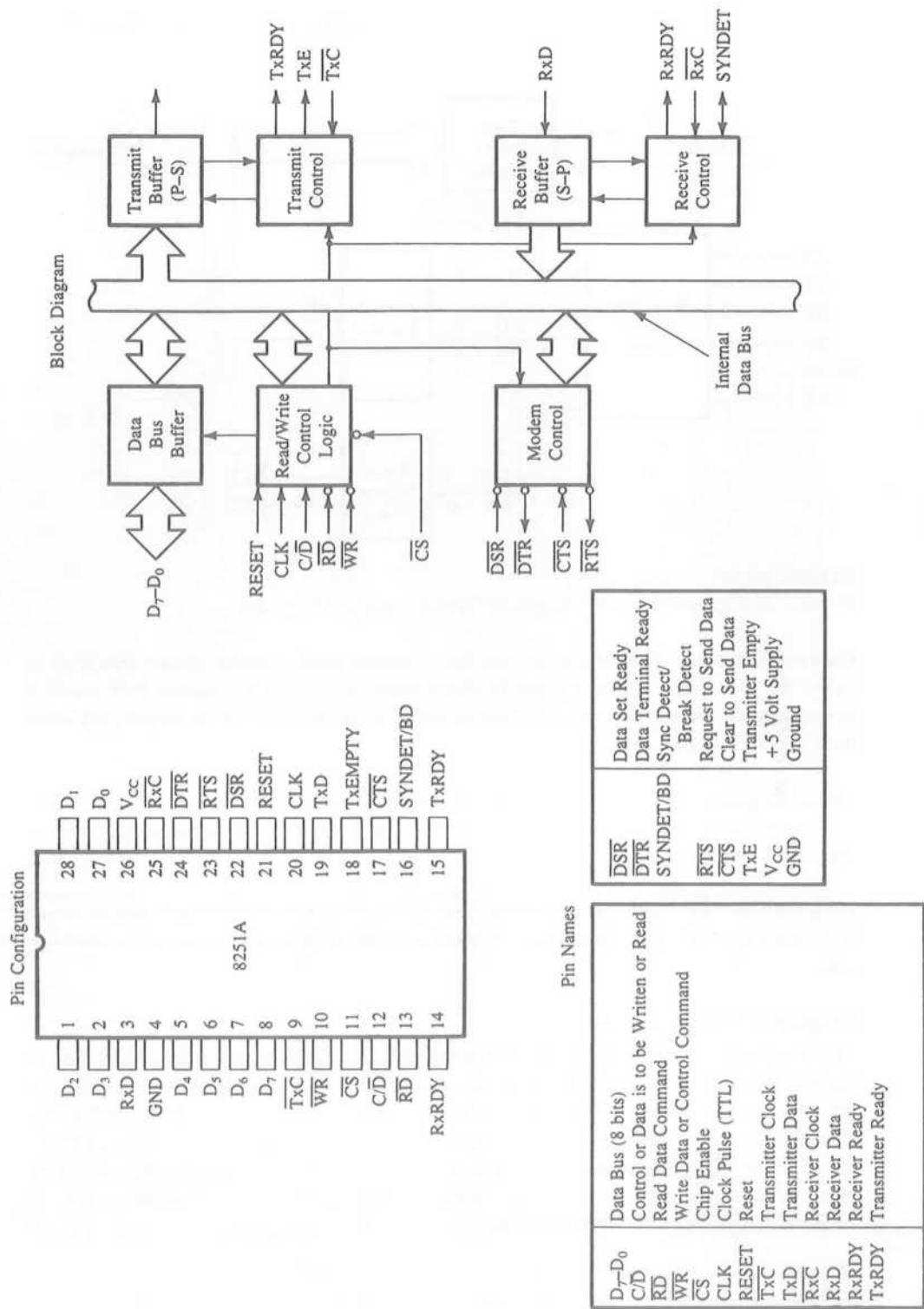


FIGURE 16.12
The 8251A: Block Diagram, Pin Configuration, and Description
SOURCE: Intel Corporation, *Connectivity* (Santa Clara, Calif.: Author, 1993), p. 2-2.

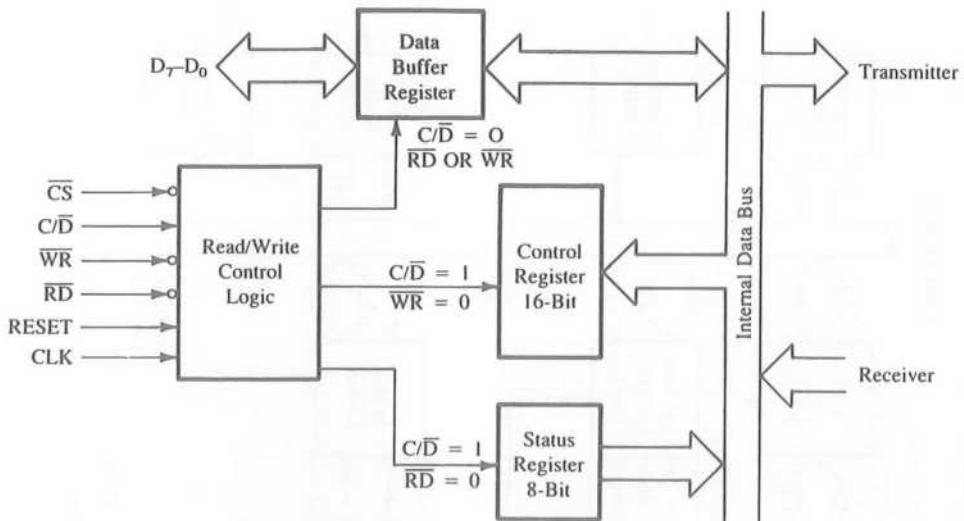


FIGURE 16.13

The 8251A: Expanded Block Diagram of Control Logic and Registers

Control Register This 16-bit register for a control word consists of two independent bytes: the first byte is called the **mode instruction** (word) and the second byte is called the **command instruction** (word). This register can be accessed as an output port when the C/D pin is high.

Status Register This input register checks the ready status of a peripheral. This register is addressed as an input port when the C/D pin is high; it has the same port address as the control register.

Data Buffer This bidirectional register can be addressed as an input port and an output port when the C/D pin is low. Table 16.4 summarizes all the interfacing and control signals.

TRANSMITTER SECTION

The transmitter accepts parallel data from the MPU and converts them into serial data. It has two registers: a buffer register to hold eight bits and an output register to convert eight bits into a stream of serial bits (Figure 16.14). The MPU writes a byte in the buffer register; whenever the output register is empty, the contents of the buffer register are transferred to the output register. This section transmits data on the TxD pin with the appropriate framing bits (Start and Stop). Three output signals and one input signal are associated with the transmitter section.

- TxD—Transmit Data:** Serial bits are transmitted on this line.
- TxC—Transmitter Clock:** This input signal controls the rate at which bits are transmitted by the USART. The clock frequency can be 1, 16, or 64 times the baud.

TABLE 16.4
Summary of Control Signals for the 8251A

CS	C/D	RD	WR	Function
0	1	1	0	MPU writes instructions in the control register
0	1	0	1	MPU reads status from the status register
0	0	1	0	MPU outputs data to the Data Buffer
0	0	0	1	MPU accepts data from the Data Buffer
1	X	X	X	USART is not selected

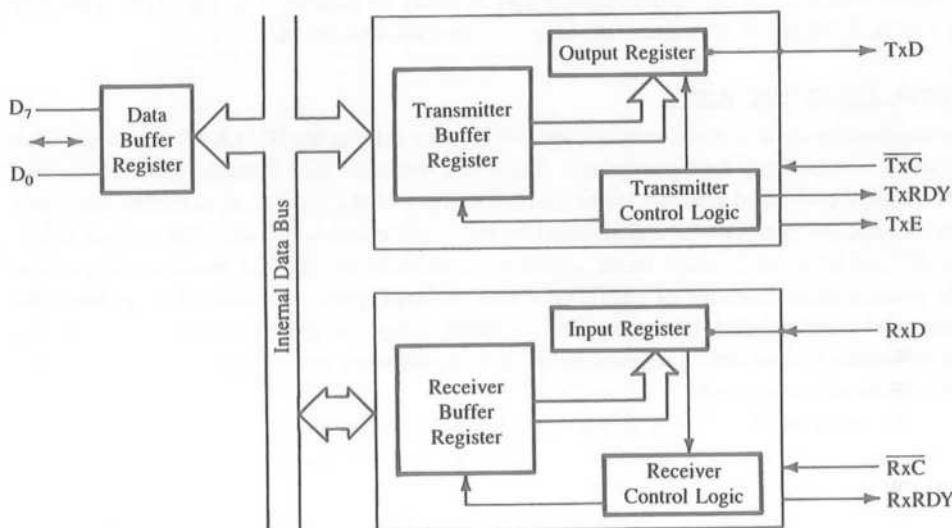


FIGURE 16.14

The 8251A: Expanded Block Diagram of Transmitter and Receiver Sections

- **TxRDY—Transmitter Ready:** This is an output signal. When it is high, it indicates that the buffer register is empty and the USART is ready to accept a byte. It can be used either to interrupt the MPU or to indicate the status. This signal is reset when a data byte is loaded into the buffer.
- **TxE—Transmitter Empty:** This is an output signal. Logic 1 on this line indicates that the output register is empty. This signal is reset when a byte is transferred from the buffer to the output registers.

RECEIVER SECTION

The receiver accepts serial data on the RxD line from a peripheral and converts them into parallel data. The section has two registers: the receiver input register and the buffer register (Figure 16.14).

When the RxD line goes low, the control logic assumes it is a Start bit, waits for half a bit time, and samples the line again. If the line is still low, the input register accepts

the following bits, forms a character, and loads it into the buffer register. Subsequently, the parallel byte is transferred to the MPU when requested. In the asynchronous mode, two input signals and one output signal are necessary, as described below.

- RxD—Receive Data:** Bits are received serially on this line and converted into a parallel byte in the receiver input register.
- RxC—Receiver Clock:** This is a clock signal that controls the rate at which bits are received by the USART. In the asynchronous mode, the clock can be set to 1, 16, or 64 times the baud.
- RxRDY—Receiver Ready:** This is an output signal. It goes high when the USART has a character in the buffer register and is ready to transfer it to the MPU. This line can be used either to indicate the status or to interrupt the MPU.

INITIALIZING THE 8251A

To implement serial communication, the MPU must inform the 8251A of all details, such as mode, baud, Stop bits, parity, etc. Therefore, prior to data transfer, a set of control words must be loaded into the 16-bit control register of the 8251A. In addition, the MPU must check the readiness of a peripheral by reading the status register. The control words are divided into two formats: mode words and command words. The mode word specifies the general characteristics of operation (such as baud, parity, number of Stop bits), the command word enables data transmission and/or reception, and the status word provides the information concerning register status and transmission errors. Figure 16.15 shows the definitions of these words.

To initialize the 8251A in the asynchronous mode, a certain sequence of control words must be followed. After a Reset operation (system Reset or through instruction), a mode word must be written in the control register followed by a command word. Any control word written into the control register immediately after a mode word will be interpreted as a command word; that means a command word can be changed anytime during the operation. However, the 8251A should be reset prior to writing a new mode word, and it can be reset by using the Internal Reset bit (D_6) in the command word.

16.4.2 Illustration: Interfacing an RS-232 Terminal Using the 8251A

PROBLEM STATEMENT

1. Identify the port addresses of the control register, the status register, and the data register in Figure 16.16.
2. Explain the RS-232 signals and the operations of the line driver (MC 1488) and the line receiver (MC 1489) shown in Figure 16.16.
3. Specify the initialization instructions and the status word to transmit characters with the following parameters if the transmitter clock frequency (TxC) is 153.6 kHz.
 - Asynchronous mode with 9600 baud
 - Character length = seven bits and two Stop bits
 - No parity check

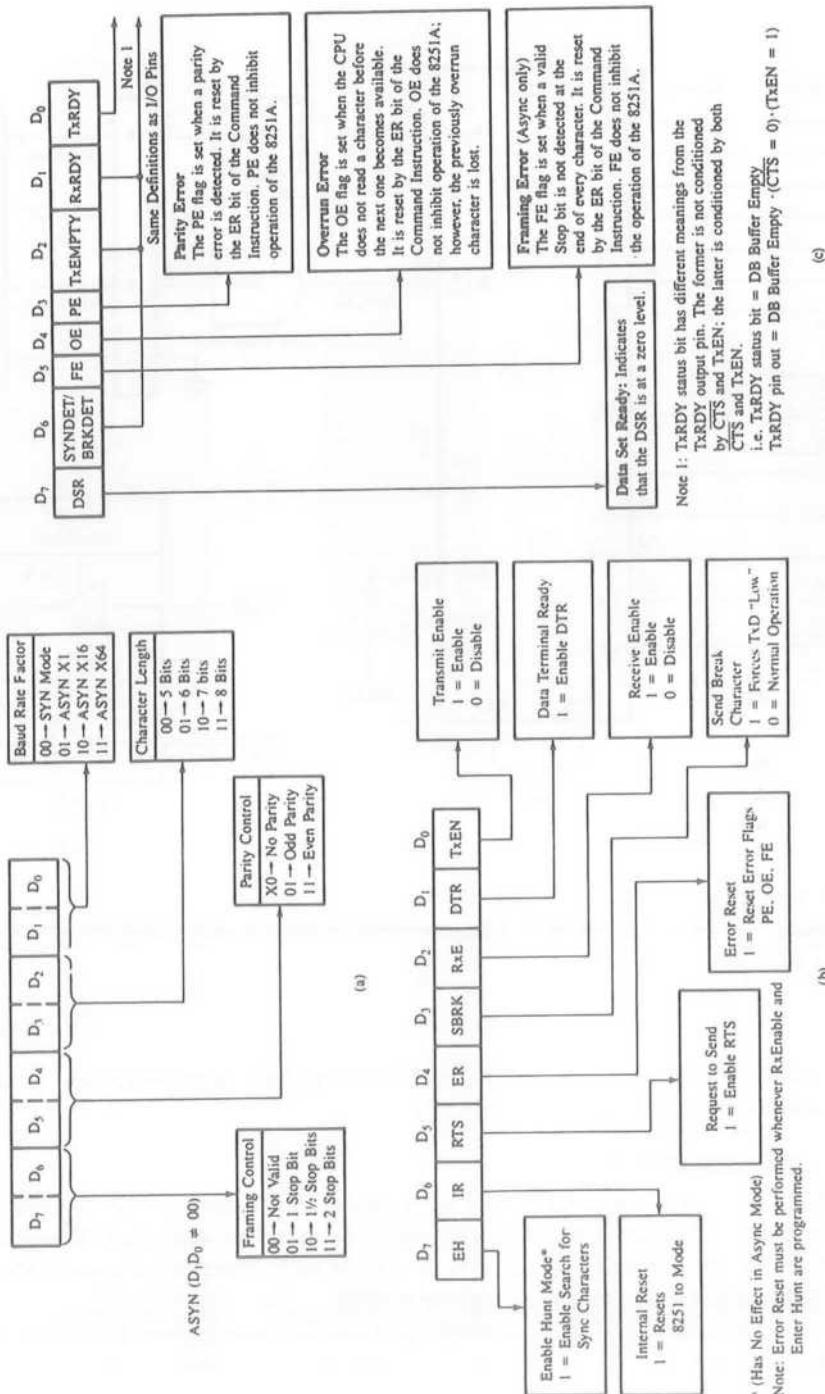


FIGURE 16.15
Mode Word Format (a), Command Word Format (b), and Status Word Format (c)

SOURCE: (a) and (c): Intel Corporation, *Connectivity* (Santa Clara, Calif.: Author, 1993), p. 2-8.

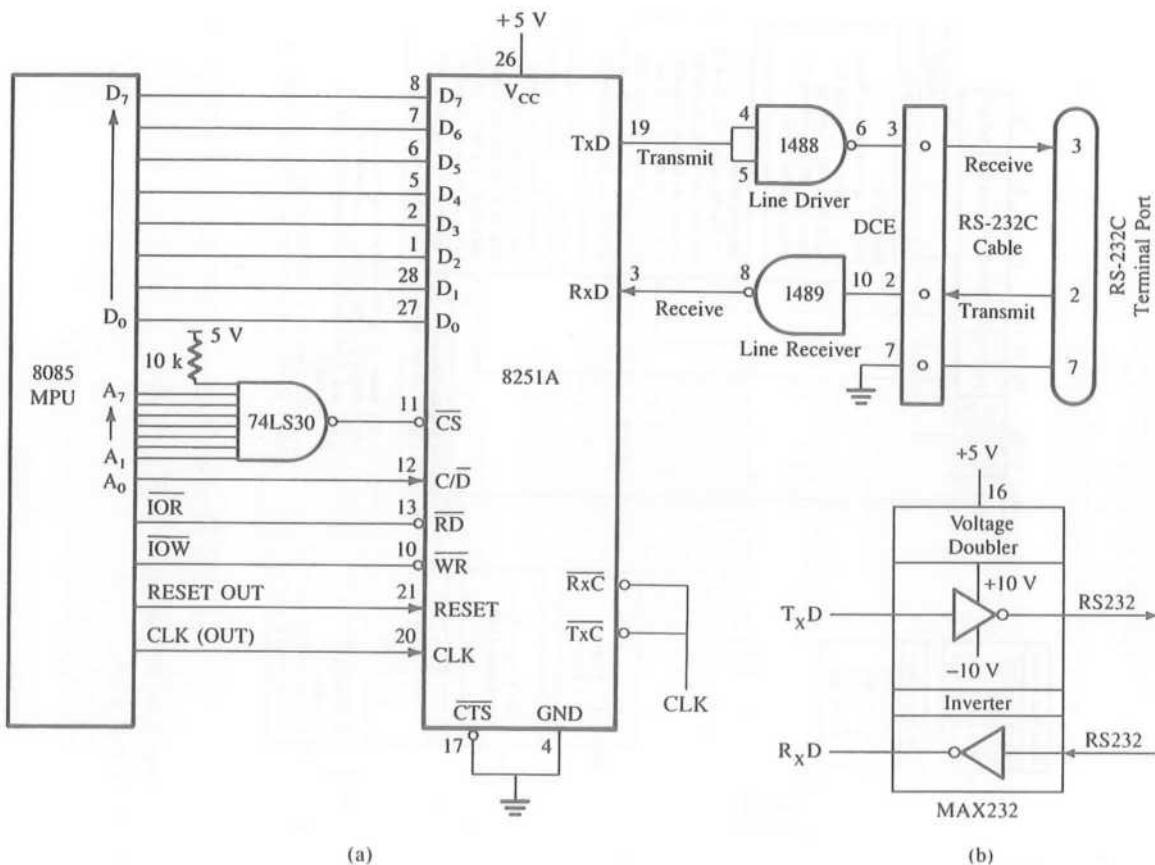


FIGURE 16.16
Schematic of Interfacing an RS-232 Terminal with an 8085 System Using the 8251A (a) and MAX 232 Logic Diagram (b)

4. Write instructions to initialize the 8251A, to read the status word, and set up a loop until the transmitter (TxRDY) is ready.

1. PORT ADDRESSES

- a. The Chip Select line of the 8251A is enabled when the address lines A₇ through A₁ are at logic 1. To select the control register or the status register, the C/D line should be high, which means that address line A₀ should be 1. Therefore, the port address of the control register and the status register = FFH.

The control register is an output port and the status register is an input port; they are identified by WR and RD signals, even if their port addresses are the same.

- b. The data register is selected when the C/D line goes low; thus, A₀ should be low. The port address of the data register = FEH. The register is bidirectional, and the same address is used to receive or transmit data. The input and output functions are identified by RD and WR signals.

2. RS-232C SIGNALS, LINE DRIVERS, AND LINE RECEIVERS

Figure 16.16 shows that three RS-232 signals—TxD, RxD, and Ground—are being used for serial communication between the CRT terminal and the 8085 system. The terminal transmits data on pin 2 and receives on pin 3; on the other hand, the 8085 system receives on pin 2 and transmits on pin 3 using the 8251A. Therefore, the terminal is connected as the DTE and the system plays the role of the DCE; the 8251A is part of the 8085 system.

Data transmitted over the TxD line (pin 19 of the 8251A) are at the TTL logic level. These bits are converted to RS-232 voltage levels and negative logic by line driver MC 1488. Data received by the 8251A over the RxD line (pin 3) should be at the TTL logic level. Therefore, the RS-232 signals at pin 2 of the connector are converted to the positive TTL logic level by line receiver MC 1489. The line driver and receiver are described here briefly.

Line Driver: MC 1488 This is a quad line driver that converts TTL input levels to a maximum +15 V_{DC} output signal. Typically, it is used with a ±12 V power supply. For logic 0 input (< 0.8 V_{DC}) the output is around +10 V, and for logic 1 input (> +2.4 V_{DC}) the output is around -10 V; thus, the positive true logic is converted into negative true logic for RS-232C signals. The internal circuit of the MC 1488 functions much like a comparator. For an input lower than the threshold voltage, the output approaches positive power supply voltage, and for an input higher than the threshold voltage, the output approaches negative power supply voltage.

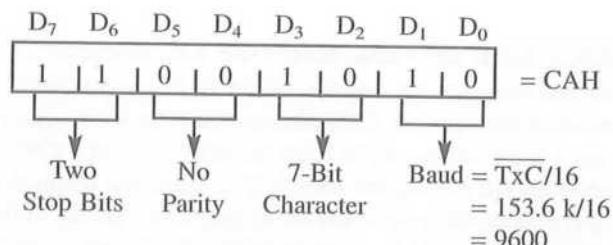
Line Receiver: MC 1489 This is a quad line receiver that converts high voltage signals (+15 V) into TTL logic levels. Output voltages usually range from 0.2 V (low) to 4.0 V (high). The internal circuit functions as an on/off transistor. When the transistor base has a negative input voltage, the transistor is turned off and the collector voltage (the output of the MC 1489) is high. When the transistor base has a positive input voltage, the transistor is driven into saturation to 0.2 V.

RS-232 Drivers/Receivers: MAX 232 One of the drawbacks of the line driver MC 1488 is that it requires additional voltages ±12 V from a power supply. Typically, such voltages are unavailable in systems compatible with the TTL logic. This difficulty can be resolved by using a specialized integrated device, MAX 232 (Figure 16.16(b)). It includes drivers and receivers (two each), and it generates ±10 V internally by using a voltage doubler and inverter circuits. The MAX 232 can replace the 1488 and 1489 and eliminate the need for ±12 V power supplies.

3. INITIALIZATION

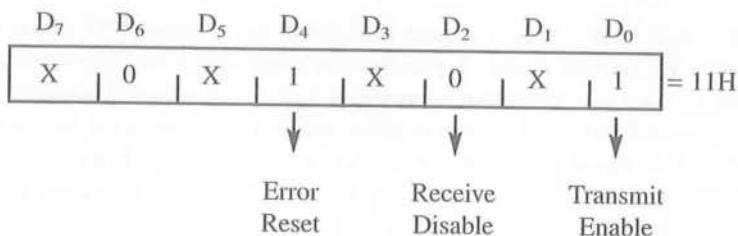
The control words necessary for the given specifications are as follows:

Mode Word Refer to Figure 16.15(a).



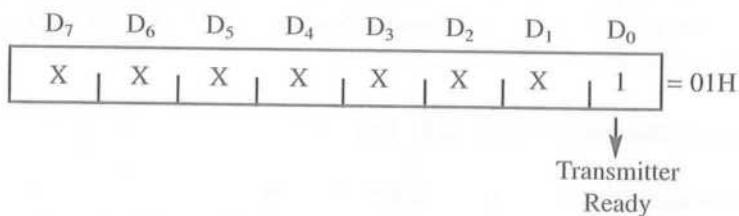
In a mode word, bits D₁ and D₀ can specify a baud factor that divides the clock frequency (TxC) to provide three different transmission rates. In this illustration, TxC is sixteen times the specified baud.

Command Word Refer to Figure 16.15(b).



In this command word, bit D₀ enables the transmitter, bit D₄ ignores any errors, and bit D₆ prevents reset of the 8251A; all other bits are don't care. In this illustration, bit D₄ also can assume don't care logic level.

Status Word Refer to Figure 16.15(c).



The MPU should check bit D₀ before transferring a character to the 8251A; it indicates the status of the pin TxRDY. When a byte is transferred from the transmitter buffer to the output register (see Figure 16.14 for the block diagram), bit D₀ is set to 1, and it is reset when the MPU loads the next byte in the buffer. Bit D₂ (TxEMPTY) indicates the status

of the output register; this bit usually is not used except in applications such as half duplex mode.

4. INITIALIZATION INSTRUCTIONS

SETUP:	MVI A,CAH	;Load the mode word
	OUT FFH	;Write mode word in control register
	MVI A,11H	;Load the command word to enable transmitter
	OUT FFH	;Enable the transmitter
STATUS:	IN FFH	;Read status register
	ANI 01H	;Mask all bits except D ₀
	JZ STATUS	;If D ₀ = 0, the transmitter buffer is full; go back ; and wait

Figure 16.17 shows the contents of various registers after the initialization. Mode word CAH and command word 11H are loaded in the control register by the OUT FFH instructions. The MPU checks the transmitter status by reading the status register and examining bit D₀ (Transmitter Ready).

When a character is transferred from the buffer register to the output (parallel-to-serial) register, bit D₀ is set to 1, indicating to the MPU that the transmitter is ready to accept the next character.

16.4.3 Illustration: Data Transmission to a CRT Terminal Using the 8251A in the Status Check Mode

PROBLEM STATEMENT

Write a program including the initialization of the USART—the 8251A—to transmit a message from an 8085 single-board microcomputer to a CRT terminal (Figure 16.16). The requirements are as follows:

1. A message is stored as ASCII characters (without parity) in memory locations starting at XX70H.
2. The message specifies the number of characters to be transmitted as the first byte (excluding the first byte) and concludes with the characters for the Carriage Return and the Line Feed.

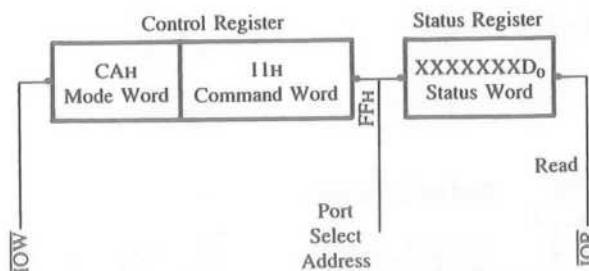


FIGURE 16.17
Control Register Contents After Initialization

3. The initialization instructions are the same as in the previous illustration (Figure 16.16).
4. The program should check status before it transmits a character. See Figure 16.18.

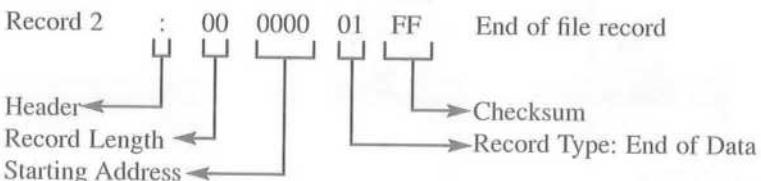
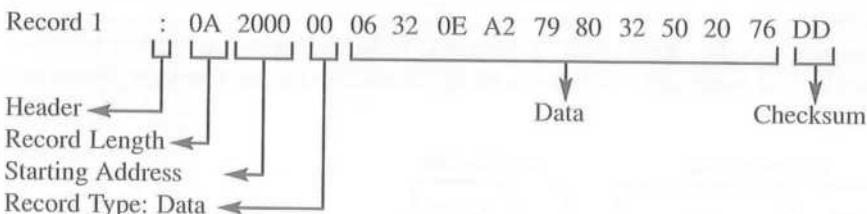
PROGRAM DESCRIPTION

According to the problem statement, the first character of the message specifies the number of characters to be transmitted. Therefore, the instruction `MOV C,M` loads the first character (in this case `08H`) in register C and sets that register as the counter.

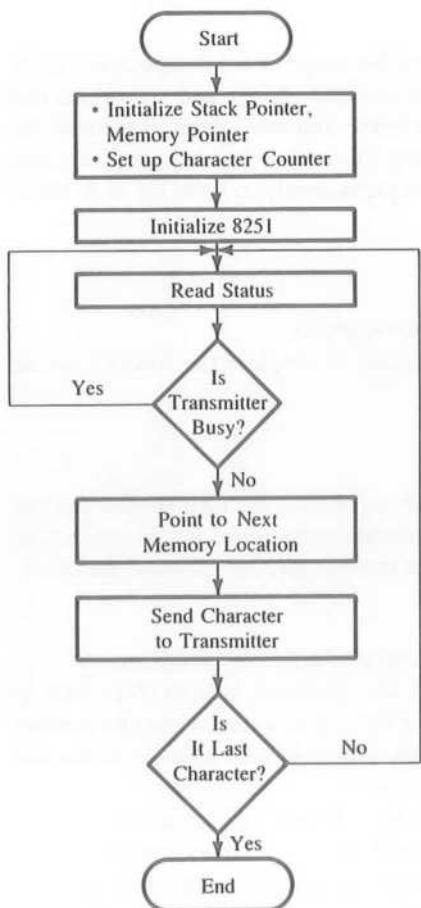
Before the initialization of the 8251A, the dummy mode word and the reset command are sent to the control register. Initially, the control register may have any random command; therefore, it is a good practice to reset the 8251A. However, it expects the first instruction as a mode word followed by a command word. Therefore, the reset command is sent after sending three dummy mode words, which are recommended to avoid problems when it is turned on.

16.4.4 Illustration: Transferring (Downloading) a File from the Microcomputer to the 8085 Single-Board Computer Using the Intel Hex Format

The communication between computers is a common occurrence. For example, to interface an A/D converter with a single-board computer, a program is generally written using an assembler or a cross-assembler on a software development system such as an IBM PC. Then, to test the program and the interfacing hardware, it must be transferred from the IBM PC to R/W memory of the single-board computer. The commonly used format for the Intel family of microprocessors is known as the Intel Hex format. For example, the Hex file of the ADDHEX program illustrated in Chapter 11 (Section 11.4.3) will appear on a CRT or a printer as follows. However, these are sent as ASCII characters. For example, the colon (:) will be sent as `3AH` and zero will be sent as `30H`.



PROGRAM



LXI SP,STACK	;Memory pointer
LXI H,XX70H	; for message
MOV C,M	;Set up register C as counter
MVI A,00H	;Dummy mode word
OUT FFH	;Write dummy word in
OUT FFH	; mode register
MVI A,40H	;Reset word
OUT FFH	;Reset 8251
MVI A,CAH	;Initialize 8251
OUT FFH	See program description
MVI A,11H	
OUT FFH	
IN FFH	
ANI 01H	;Check TxRDY
JZ STATUS:	;Is TxRDY 1? ; If not, wait
INX H	;Point to next character
MOV A,M	;Place character in the accumulator
OUT FEH	;Send character to the transmitter
DCR C	;One character transmitted; ; decrement the counter
JNZ STATUS	;If all characters are not yet transmitted, go back and transmit again
HLT	
:Message—HELLO	
XX70 08	;Number of characters to follow
71 48	;Letter H
72 45	;Letter E
73 4C	;Letter L
74 4C	;Letter L
75 4F	;Letter O
76 21	;Exclamation
77 0D	;Carriage Return
78 0A	;Line Feed

FIGURE 16.18

Flowchart for Transmitting Message from Single-Board Microcomputer Using the 8251A

To transfer this program in the R/W memory of the single-board computer using the serial I/O, we need an interfacing circuit, as shown in Figure 16.16, and a program that can recognize the header of a file, receive the data bytes, and store them in memory locations with the given starting address. The following illustration discusses the Intel Hex format and the receiver program that must be stored permanently in EPROM or ROM of the single-board computer.

PROBLEM STATEMENT

1. Explain the Intel Hex format for the program shown above.
2. Draw flowcharts for a receiver (download) program and explain the function of the DTR signal.

1. INTEL HEX FORMAT

A program or a file is divided into records, and each record has the format shown in the example of the ADDHEX program. Each record has six segments in ASCII characters: the header, the record length, the starting address, the record type, the data, and the checksum. These are described below.

- Header:** The colon is the first byte used to indicate the beginning of the record.
- Record Length:** The number of data bytes in the record; it can be from 00 to 10H. In the above example, the record length in Record 1 is 0AH to indicate that the number of bytes in the record is ten, and in Record 2, the length is zero to indicate the last record in the file.
- Starting Address:** This is the memory address where the data in that record are to be stored. In the example, the starting address is 2000H. If there were additional records, the starting memory address of each record would be automatically calculated and shown. Record 2 is the end of the file, and it does not have any data bytes. Therefore, the memory address shown is 0000H.
- Record Type:** This includes two types of records: 00 means normal data and 01 means end of file record. Record 1 shows 00 to indicate normal data, and Record 2 shows 01 to indicate the end of file record.
- Data:** These are the Hex bytes of the mnemonics in the program. In the example, we have 10 data bytes; the maximum number of bytes allowed in a record is 16 (10H). There are no data bytes in Record 2.
- Checksum:** This is the 2's complement of the sum of all the bytes in the record, excluding the header. In Record 1, the sum of all the bytes, from 0AH to 76H, is 323H. The checksum ignores the carry digit 3 and takes the 2's complement of 23H, which is DDH. The receiver program adds all the bytes, including the checksum, and the result should be zero for an error-free transmission.

2. DESIGN OF DOWNLOAD PROGRAM

After examining the Intel Hex format, the tasks of the receiver program can be divided into the following segments.

- Initialize the 8251 to receive a file.
- Check for the header character.
- Read two ASCII characters at a time.
- Convert the ASCII characters into binary values, and combine them in a byte.
- Extract the information concerning the byte count and the memory address. Add the Hex values in the checksum counter.
- Check for the record type. If it is the end of file, display the successful transfer.
- If the record type is data, store the bytes in memory, update the memory pointer and the byte counter, and add the Hex values in the checksum counter.
- After reading all the data bytes, add the checksum to the value of the checksum received (the last two ASCII characters in the record). If the result is zero, display end of data transfer; if it is other than zero, display error message.

The flowchart in Figure 16.19 shows the above steps. The program is divided into five subroutines, which are explained below:

1. **RDASKY**—Figure 16.20(a). This subroutine enables the 8251 receiver, checks the status, and reads an ASCII character. Before it returns to the main program, it deactivates the DTR signal (Figure 16.21) so that no character can be sent until it is ready to read again. This allows the program to perform other functions, such as ASCII conversion and checksum.
2. **ASCBIN**—Figure 16.20(c). This subroutine converts an ASCII character into binary value and adds the value to the checksum counter. (Refer to Problem 9.)
3. **HEXBYTE**—Figure 16.20(b). This subroutine reads two ASCII characters by calling RDASKY, converts the characters into binary by calling ASCBIN, and combines the binary values to form a byte. The binary value of the first ASCII character is saved as high-order four bits and of the second character as low-order four bits.
4. **CHKSUM**. This subroutine reads the last two ASCII characters by calling the routine HEXBYTE and compares the byte with the checksum counter. If the result is not zero, it calls the error message.
5. **ERROR**. This subroutine displays an error message. The details of this message are dependent on a particular single-board computer and its display output ports.

Figure 16.21 shows that the DTR signal is connected to pin 20 of the RS-232 cable; this signal is used as a handshake signal. Before reading an ASCII character, the DTR is enabled, and after reading the ASCII character, the DTR signal is set high by sending $D_1 = 1$ in the command word. This prevents the transmission of subsequent characters and allows the program to perform ASCII-to-binary conversion and the checksum procedure.

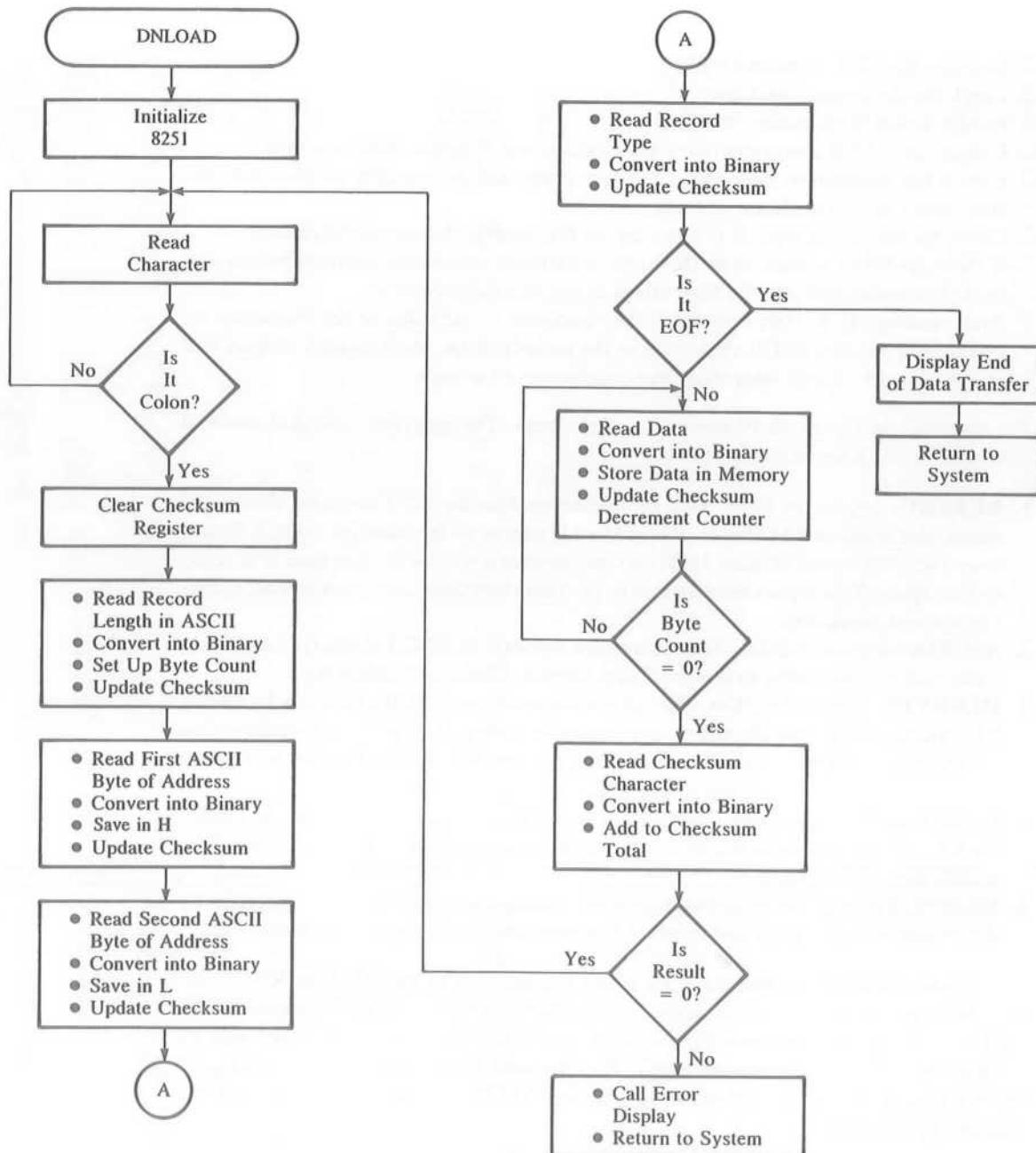


FIGURE 16.19
Flowchart: Download—Main Program

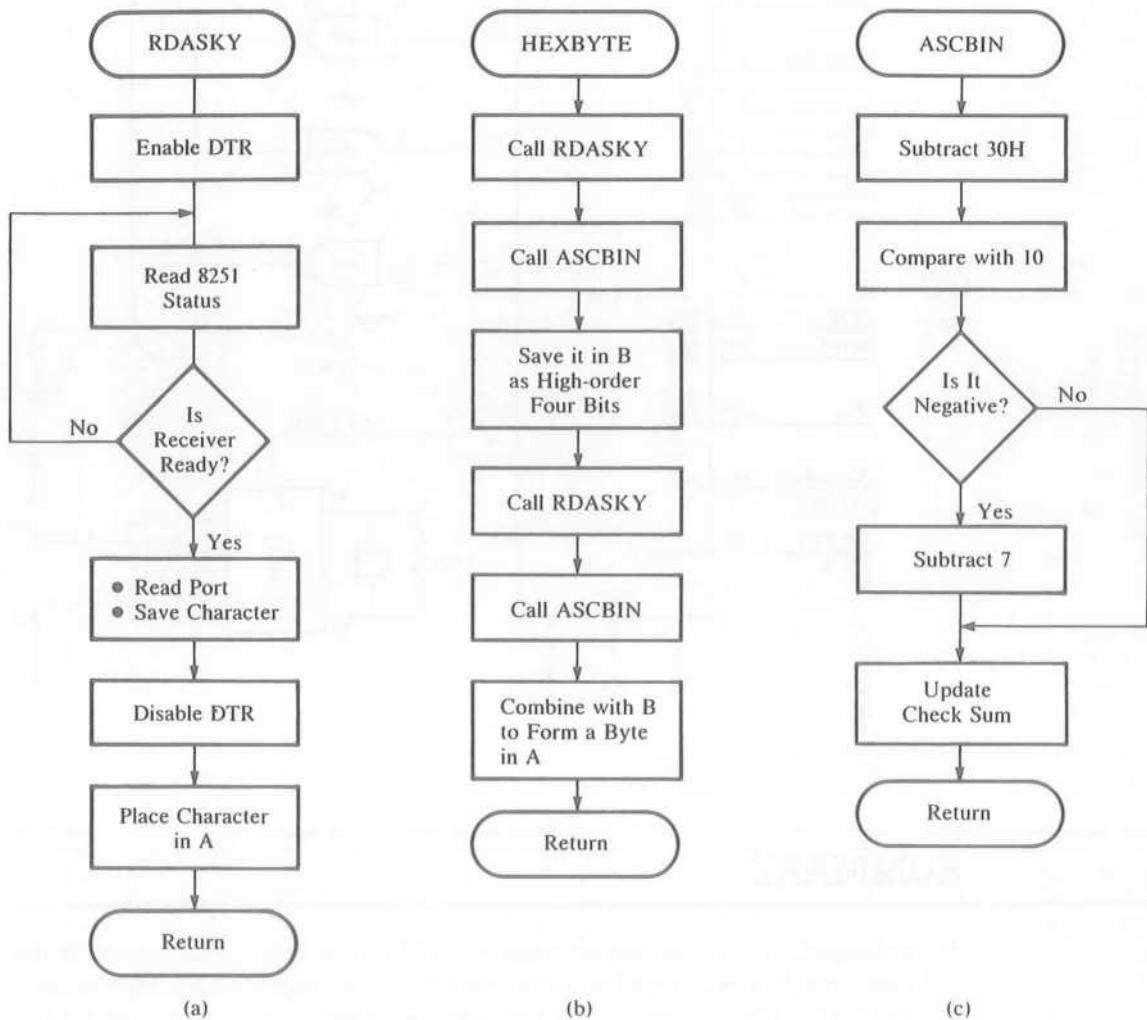


FIGURE 16.20
Flowcharts: Download Program Subroutines

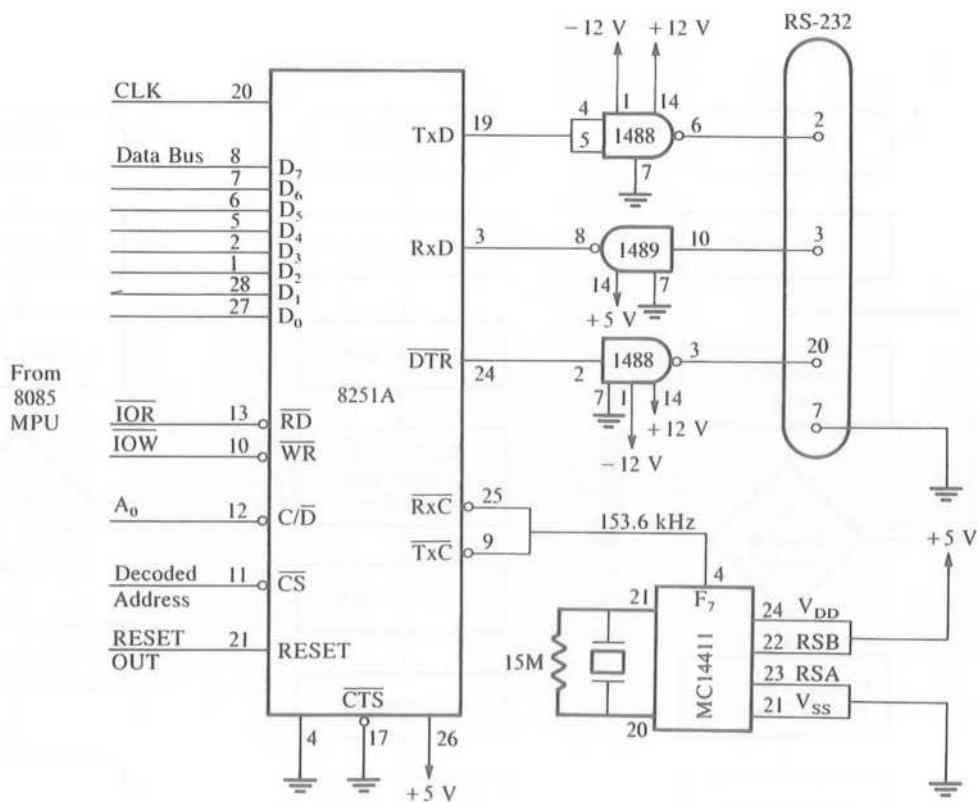


FIGURE 16.21
Using DTR as a Handshake Signal

SUMMARY

In this chapter, we discussed the technique of serial I/O for data communication. In this technique, one bit is transferred over one line rather than using eight data lines to transfer a byte. The serial I/O technique is necessary for certain types of equipment and media, such as magnetic tapes and telephone lines. The rate of data transfer in serial I/O is determined by the time delay between two successive bits. Therefore, a host of issues, such as synchronization of data transfer between the transmitter and the receiver, and error check, need to be resolved. The serial I/O data transfer can be implemented using software techniques; however, programmable devices called USART (Universal Synchronous/Asynchronous Receiver/Transmitter) are commonly used in industrial and commercial products. The basic concepts involved in the serial I/O can be summarized as follows:

1. In serial I/O communication, a word is transmitted one bit at a time over a single line by converting a parallel word into a stream of serial bits. On the other hand, a word is received by converting a stream of bits into a parallel word.
2. Serial data communication can be either synchronous or asynchronous. The synchronous mode is used for high-speed and the asynchronous mode is used for low-speed data communication.
3. The MPU identifies a serial peripheral through a decoded address and an appropriate control signal. Data transfer occurs under various conditions—unconditional, status check, and interrupt—depending upon logic design.
4. In software-controlled serial transmission, the MPU converts a parallel word into serial bits by using time delays and transmits bits over one data line of an output port.
5. In software-controlled serial reception, the MPU converts a serial word into a parallel word by using time delays and receives bits over one data line.
6. In the 8085, two specially designed 1-bit ports are provided for serial I/O: SOD and SID. Data transfer is controlled through two instructions: SIM and RIM.
7. The 8251A is a programmable serial I/O chip known as USART, which can perform all the functions of serial I/O.

DEFINITION OF TERMS

- ASCII (American Standard Code for Information Interchange). A 7-bit alphanumeric code commonly used in computers.
- EBCDIC (Extended Binary Coded Decimal Interchange Code). An 8-bit alphanumeric code used primarily in large IBM computers.
- Asynchronous Serial Data Transmission. In this format, the transmitter is not synchronized with the receiver by the same master clock. A transmitted character includes information about the starting and ending of the character.
- Synchronous Serial Data Transmission. In this format, the transmitter is synchronized with the receiver on the same frequency.
- Simplex Transmission. One-way data communication.
- Duplex Transmission. Two-way data communication. Full duplex is simultaneous in both directions and half duplex is one direction at a time.
- Baud (Rate). The number of signal changes per second. In serial I/O, it is equal to bits per second, the rate of data transmission.
- Current Loop. The transmission of serial data bits as current signals.
- RS-232C. A data communications standard that defines voltage signals in reference to data terminal equipment and data communication equipment.
- RS-422A and 423A. Data communication standards for high-speed data transmission.
- SID (Serial Input Data Line). A specially designed 1-bit input port in the 8085 to receive serial data.

- SOD (Serial Output Data). A specially designed 1-bit output port in the 8085 to transmit serial data.
- USART (Universal Synchronous/Asynchronous Receiver/Transmitter). A programmable chip designed for synchronous/asynchronous serial data communication.

QUESTIONS, PROBLEMS, AND PROGRAMMING ASSIGNMENTS

1. Check whether the following statements are true or false.
 - a. Serial data communication cannot be implemented using the memory-mapped I/O technique. (T/F)
 - b. ASCII is an 8-bit binary code that represents 256 different characters. (T/F)
 - c. In the synchronous serial I/O format, all eight bits are sent simultaneously. (T/F)
 - d. In the half duplex transmission mode, data flow is bidirectional between the MPU and a serial peripheral, but not simultaneously. (T/F)
 - e. In serial transmission from the MPU to a peripheral, bit D₀ is transmitted first after the Start bit. (T/F)
 - f. In serial reception from a peripheral to the MPU, bit D₇ of a character is received first after the Start bit. (T/F)
 - g. In a system with the odd parity check, the letter A is transmitted with the code C1H. (T/F)
 - h. In a system with the even parity check, the letter M is transmitted with the code 4CH. (T/F)
 - i. The delay between the successive bits for 9600 baud rate is approximately 0.1 ms. (T/F)
 - j. RS-232C is a 25-pin serial I/O voltage standard compatible with TTL logic. (T/F)
 - k. To enable the 8085 serial output data line (SOD), bits D₇ and D₆ of the accumulator should be at logic 1. (T/F)
 - l. The instruction RIM is equivalent to the instruction IN with one input data line (D₇). (T/F)
 - m. The 8085 SID and SOD lines receive and transmit characters starting from bit D₇ after the Start bit. (T/F)
2. Write delay loops for BITTIME and half BITTIME for 1200 baud if the system frequency is 3 MHz.
3. Sketch the serial output waveform for the ASCII character A when it is transmitted with 9600 baud and even parity.
4. Sketch the serial output waveform for the ASCII sign + when it is transmitted with 2400 baud and odd parity.
5. Write a program to transmit letters A to Z from the MPU to the terminal in Figure 16.16.

6. Write a subroutine to accept a letter from the CRT terminal (Figure 16.16).
7. Write a program to receive ASCII characters from the terminal to the 8085 system (Figure 16.16) and to transmit the same characters back to the terminal to display on the CRT.
8. Specify the control word and the command word for data communication having the following specifications:
 - a. asynchronous mode
 - b. 1200 baud ($\overline{\text{Tx}}\text{C} = \overline{\text{Rx}}\text{C} = 76.8$ kHz)
 - c. 8-bit character
 - d. even parity
 - e. one Stop bit
9. An ASCII character is supplied in the accumulator. Write the subroutine ASCBIN (Section 16.4.4) to convert the ASCII character into its binary value; if the character is not within Hex values 0 to F, call the Error routine; otherwise, update the checksum counter and return to the main program. (Refer to Section 10.4.2. Modify the illustrative program in this section to check whether the binary value is within the range.)
10. Write the subroutine HEXBYTE (Section 16.4.4) that reads two ASCII characters by calling the subroutine RDASKY, converts them into binary values by calling ASCBIN, and combines the binary values in a byte.

EXPERIMENTAL ASSIGNMENTS

1.
 - a. Connect the circuit shown in Figure 16.16 to receive and transmit ASCII characters.
 - b. Connect the clock with 4.8 kHz frequency to $\overline{\text{Rx}}\text{C}$ and $\overline{\text{Tx}}\text{C}$ pins of the 8251A. (Use a pulse generator or set up the 8155 timer for 4.8 kHz.)
 - c. Write initialization instructions to set up the 8251A for the asynchronous mode, 300 baud, and 7-bit character with no parity.
 - d. Write a program to receive a character from the terminal and echo the character back to the terminal for display. Use the status check technique.
2. Write a download program to transfer a file from a computer system, such as an IBM PC, to the single-board system shown in Figure 16.16.

17

Microprocessor Applications

Microcomputer systems based on the 8085 microprocessor were introduced at the beginning of the book, an overview of the instruction set was given in Chapter 2, and Chapters 6 through 10 were devoted to programming techniques. Various types of I/O data transfer and interfacing concepts were later discussed in detail in Chapters 12 through 16. This chapter is concerned with integrating or synthesizing all the concepts of the microprocessor architecture, software, and interfacing discussed previously, by designing a microprocessor system.

Designing a single-board microcomputer is the best possible choice, since it can incorporate all the important concepts related to the microprocessor. Furthermore, it allows expansion to include various types of interfacing. The chapter begins with interfacing applications commonly used in indus-

trial environments. These applications include such examples as the scanned LED displays, liquid crystal display (LCD), the matrix keyboard, and memory. Later these examples are used as components for a system design that deals primarily with designing a single-board microcomputer. The chapter also includes troubleshooting techniques using an in-circuit emulator and a logic analyzer.

OBJECTIVES

- Illustrate the interfacing of scanned display and liquid crystal display (LCD).
- Illustrate the interfacing of a matrix keyboard using software.
- Illustrate the interfacing of a matrix keyboard using a keyboard encoder.
- Draw a system block diagram of a single-board microcomputer based on the 8085 microprocessor

- and its family of programmable interfacing devices.
- Draw flowcharts to illustrate the software design of a key monitor program and related submodules.
 - List the primary features of an in-circuit emulator and explain its applications in troubleshooting microprocessor-based systems.
 - Explain the functions of a logic analyzer as a troubleshooting instrument.

17.1 INTERFACING SCANNED MULTIPLEXED DISPLAYS AND LIQUID CRYSTAL DISPLAYS

In Chapter 15, when we illustrated an interfacing of a seven-segment LED using the 8255A (Figure 15.15), we needed one I/O port and a driver per LED to display one Hex digit. To display several digits with this technique, additional hardware is required in proportion to the number of digits to be displayed—which can be costly. The number of hardware chips needed for multiple-digit display can be minimized by using the technique called **multiplexing**, whereby the data lines and output ports are time-shared by various seven-segment LEDs. Similarly, liquid crystal displays (LCDs) are also used commonly in low-power consumption systems. The interfacing of scanned displays and LCDs is discussed in the next sections.

17.1.1 Scanned Multiplexed Displays

The basic circuit for multiplexed display is illustrated in Figure 17.1. The circuit has two output ports: one port (P_A) to drive LED segments, and a second port (P_B) to turn on the corresponding cathodes. The output data lines of port P_A are connected to seven segments of each LED, and the output lines of port P_B are connected to the cathodes of each LED. The code of the first digit to be displayed at LED-1 is sent on the data lines by outputting to port P_A . The corresponding seven-segment LED is turned on by sending a bit to the cathode through port P_B . Next, LED-2 is turned on and LED-1 is turned off. Each seven-segment LED is turned on and off sequentially. The cycle is repeated fast enough that the display appears stable. This is also known as scanned display.

In a common-cathode seven-segment LED, all segments are driven by the output lines, which should supply at least 10 mA to 15 mA of current to each segment. The cathode should sink seven or eight times that current. I/O ports of programmable devices (such as the 8255A) are limited in current capacity; therefore, additional transistors or ICs, called **segment** and **digit drivers**, are required, as shown in Figure 17.1 and illustrated in the next problem.

PROBLEM STATEMENT

1. Design a six seven-segment LED display using the technique of multiplexing.
2. Write a program to display the message uP-rdy (microprocessor-ready).

INTERFACING CIRCUIT

Figure 17.2 shows the address-decoding network using the 74LS138, 3-to-8 decoder to assign I/O ports. The address lines A_7-A_2 are connected to the decoder and the remain-

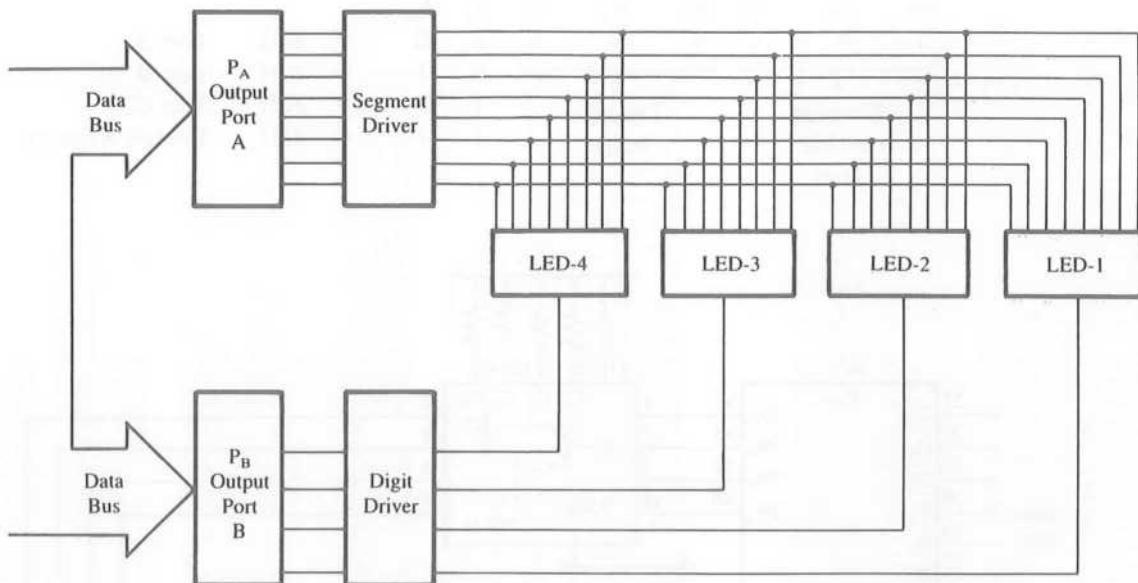
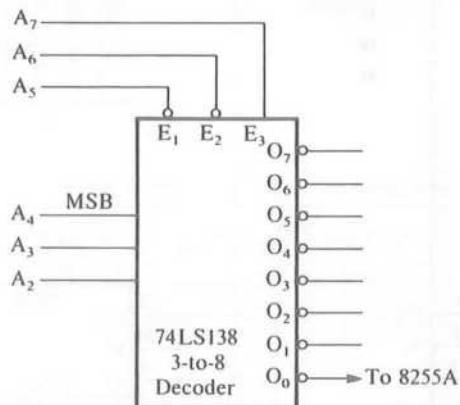


FIGURE 17.1
Block Diagram of Multiplexed Output Display

FIGURE 17.2
Address Decoding



ing two address lines A₁ and A₀ will be connected directly to the 8255A. This decoder provides the capability of eight I/O ports. We will use the output line O₀ of this decoder for the LED scanned display, and we can use the remaining output lines of the decoder for other I/Os. The 8255A is selected when the output line O₀ of the decoder (Figure 17.2) is asserted. Therefore, the port addresses of the 8255A (Figure 17.3) are as follows (refer to Chapter 15, Section 15.1.1).

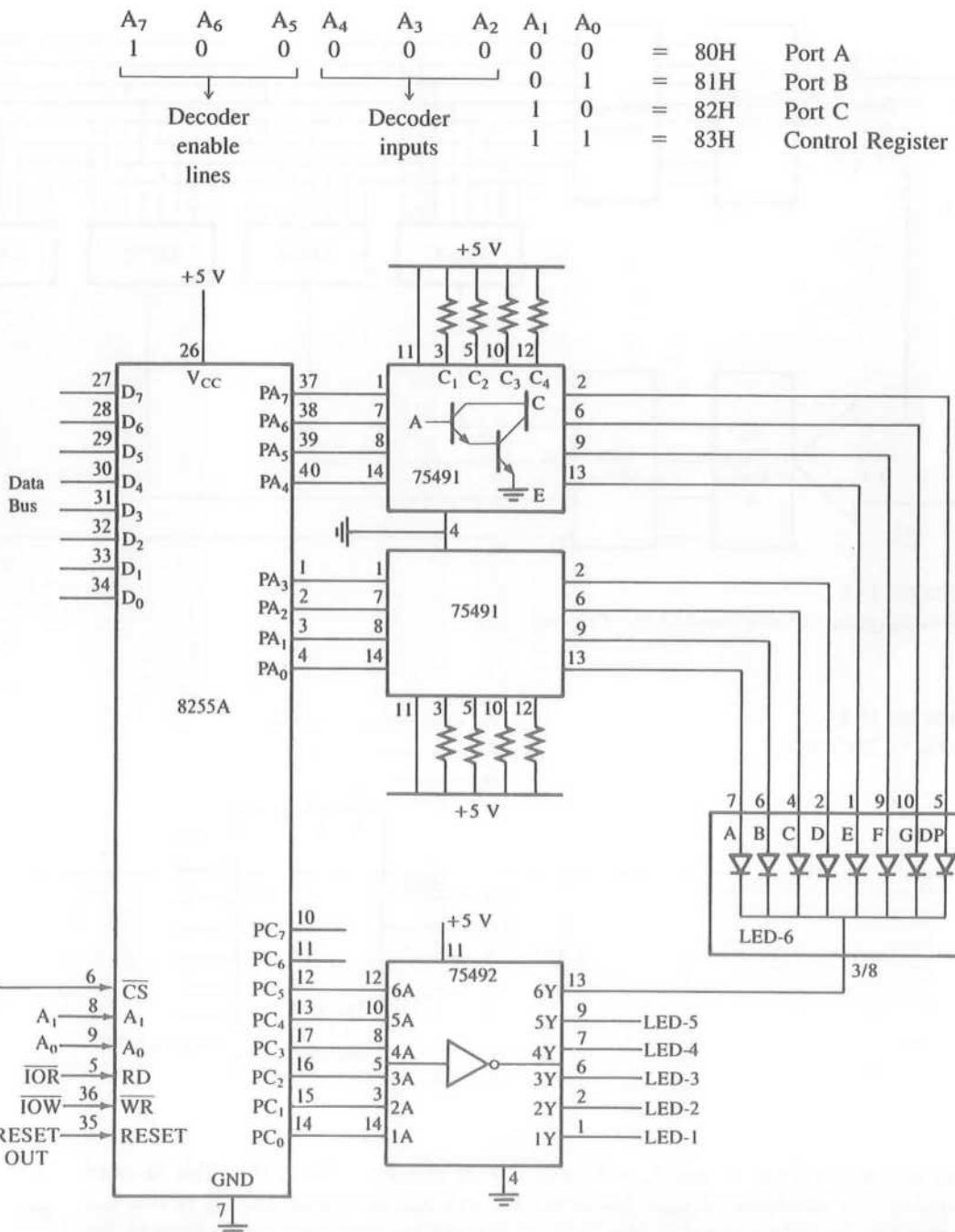


FIGURE 17.3
 Schematic: Scanned Multiplexed Display

Figure 17.3 shows the schematic of a scanned display; it has six common-cathode seven-segment LEDs, one 8255A, and two drivers. Ports A and C are used as output ports: port A with the address 80H for segment codes and port C with the address 82H for digits to turn on/off LEDs. The SN 75491 and the SN 75492 are used as the segment code driver and the digit driver, respectively, to increase the current capacity in the circuit.

SN 75491—SEGMENT DRIVER

The SN 75491 is a quad device that has four Darlington pair transistors in a package. To drive eight data lines, we need two devices, as shown in Figure 17.3. It can source or sink 50 mA current (approx. 12.5 mA/pair). Pin A, the base of the transistor, is connected to one of the data lines of the output port and emitter E is connected to one of the LED segments, as shown in Figure 17.4(a). Similarly, the base of the driver is tied to +5 V through a resistor to turn on the driver properly.

SN 75492—DIGIT DRIVER

The SN 75492 has six Darlington pairs in a package, and can sink 250 mA of total current. The collector (pin Y) is connected to the common cathode of the LED, and the data lines from the port are connected to the base of the transistor, as shown in Figure 17.4(b), to turn on/off the LEDs.

To display a digit, the seven-segment code for the digit is sent to port A, the corresponding cathode is turned on and off in sequence, and the loop is repeated continuously.

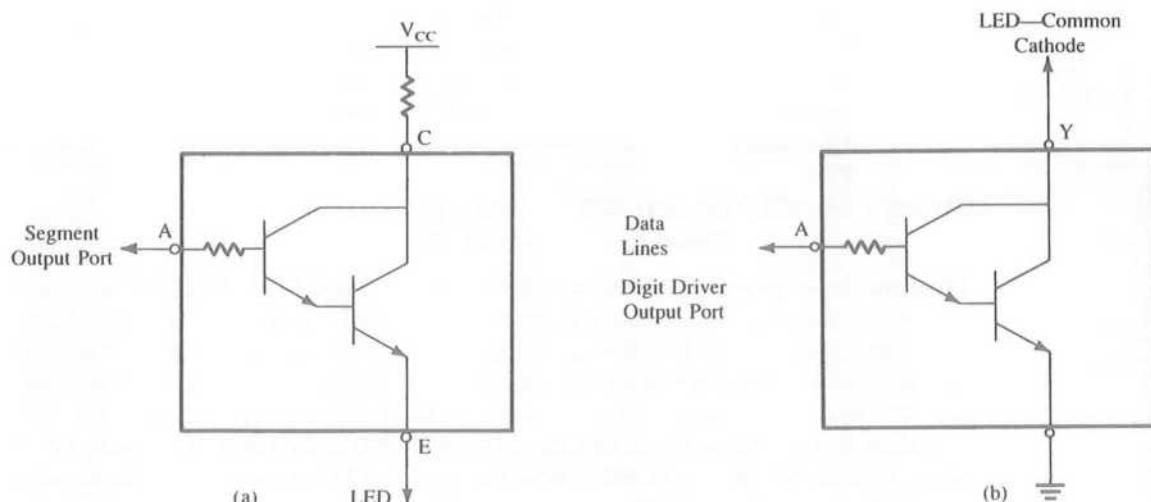


FIGURE 17.4

Simplified Diagram of: SN 75491 (a), SN 75492 (b)

SOURCE: Courtesy of Texas Instruments Incorporated

PROGRAM

;The following program initializes the 8255A ports A and C as output ports
; and displays a constant message stored at memory location SYSRDY (System
; Ready). The message has six codes: uP-rdy (microprocessor-ready). The
; code for the rightmost letter "y" is stored at the first location SYSRDY,
; and the scanning begins at that location.

PORTA EQU 80H	;Port A address—Segment Driver
PORTC EQU 82H	;Port C address—Digit Driver
PORTB EQU 81H	;Port B address, initialized for later use
CONTRL EQU 83H	;Address for Control Register
8255A: MVI A,10000010B	;Control word: Mode 0, ports A and C as output, ; port B as input for later use
OUT CONTRL	;Initialize ports A and C as outputs
READY: MVI B,00000001B	;Initialize digit code
MVI C,06H	;Initialize counter for six LEDs
LXI H,SYSRDY	;Use HL as memory pointer for message
NEXT: MOV A,M	;Get segment code
OUT PORTA	;Output segment code
MOV A,B	;Get digit code
OUT PORTC	;Turn on one LED
CALL DELAY1	;Wait 1 millisecond
XRA A	;Code to turn off segments
OUT PORTA	;Clear segments
MOV A,B	
RLC	;Shift digit code to turn on next LED
MOV B,A	;Save digit code
INX HL	;Point to next code
DCR C	;Next LED count
JNZ NEXT	
RET	
SYSRDY: DB 6EH, 5EH, 50H, 40H	;Message Codes
73H,1CH	;y d r - P u

Program Description This routine initializes ports A and C of the 8255A as output ports by sending the word 82H to the control register (refer to Figure 15.4 for the definition of the control word). Port B is not being used in this illustration; however, it is being initialized as an input port to use in the illustration of the matrix keyboard in the next section. The next instruction initializes the scan routine by placing the digit code 00000001 in register B; this code will turn on LED-1 (the first LED at the right). By rotating bit D₀ (logic 1) to the left, the next LED is turned on and the LED presently being displayed is turned off; thus, only one LED is on at a time. Register C is set up as a counter to scan six LEDs, and the HL register is used as a memory pointer to point where the message is stored.

The scanning begins by sending the first code (the last letter "y" in the message) to port A and the LED-1 is turned on by sending the digit code. This LED is kept on for ap-

proximately 1 ms by calling the delay routine, and the entire display is turned off by clearing the segment code; this eliminates the flicker and the ghost images. The segment codes are sent in a sequence as they are stored in memory, and the corresponding LED is turned on until the counter reaches zero. To keep the display on, the routine should be called repeatedly.

Comments In the scanned display, the hardware is minimized. With two output ports, this scheme can scan eight LEDs. In addition, current consumption is considerably reduced. However, the major disadvantage is that the MPU is kept occupied in scanning the display continuously. To relieve the MPU from the continuous scanning task, the Intel 8279 programmable keyboard/display interface device can be used.

This routine is appropriate for a permanent message; however, there are many situations where the routine is expected to scan only four or two LEDs. In such situations, an appropriate parameter must be passed on to register C by the calling program, instead of loading 06H as shown in the routine.

17.1.2 Interfacing a Liquid Crystal Display (LCD) for the MCTS Project

A liquid crystal display (LCD) is commonly used in systems where low power consumption is necessary. Typical examples include watches, calculators, instrument panels, and consumer electronic displays. An LCD display consists of crystal material placed between two plates. The crystal material is arranged in segments or in the form of a dot matrix. The crystal material can pass or block the light that passes through; thus it creates a display. To display a character, certain segments or dots are driven by a square wave pattern, which is supplied by a built-in driver. The driver is interfaced with the processor using signals such as data lines, enable, read/write, and chip select. In the MCTS project (described in Chapter 1), an LCD is interfaced to display temperature readings. An interfacing of a commonly available LCD is illustrated here that can be very easily adapted by replacing the message with temperature readings.

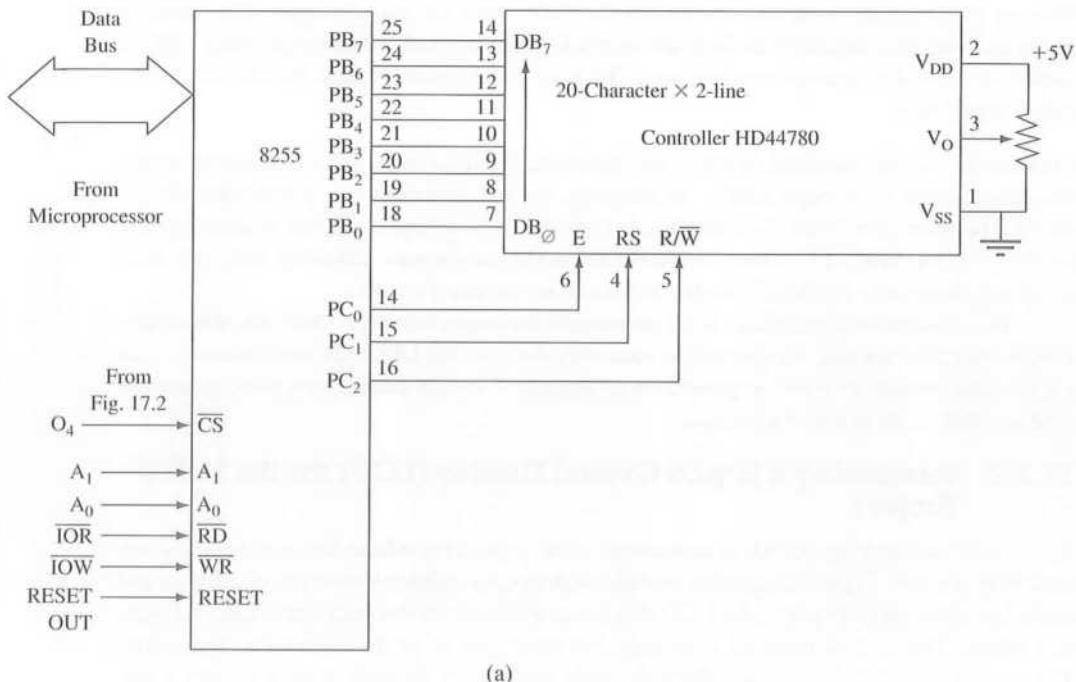
PROBLEM STATEMENT

1. Explain the hardware requirements to interface an LCD using the 8255 peripheral chip.
2. Write instructions to initialize the LCD and display a message, which terminates in the Null (00) character in the center of the display area (begin writing at address 87H of the first line).

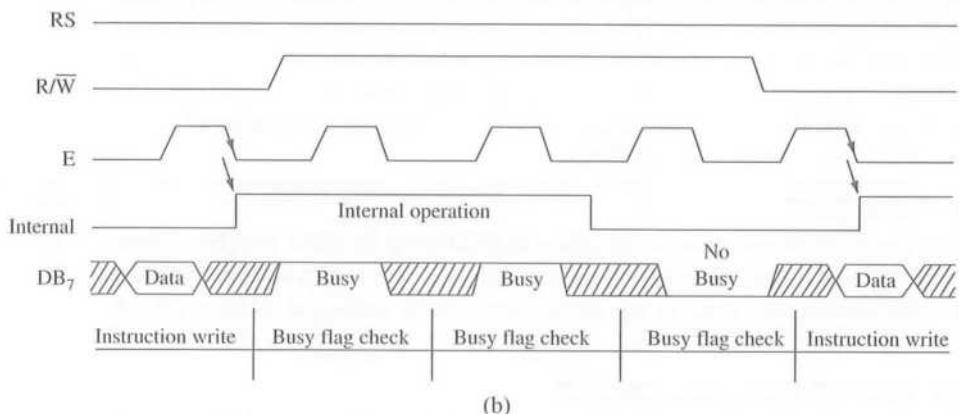
HARDWARE AND INTERFACING CIRCUIT

Figure 17.5(a) shows the interfacing of a 20-character \times 2-line dot matrix LCD module using the 8255. The LCD module has a built-in controller driver (Hitachi HD44780)* with a 14-pin dual row header. It can be interfaced with a 4-bit and 8-bit data bus. The pinout includes 8 data lines (DB7–DB0), three control signals (RS-Register Select, R/W-Read and Write, and E-Enable), and three connections for power supply and ground (V_{ss} , V_{dd} , and V_o). The controller has two 8-bit control registers: an instruction register

*Most manufacturers use this type of controller driver with a similar pin configuration.



(a)



(b)

FIGURE 17.5
Interfacing LCD Using 8255 (a); Data Transfer Timing Diagram (b)

(IR) for commands and a data register (DR) for data. The MPU can write commands when RS is low and data when RS is high. When the MPU writes into one of the registers, the controller sets data line DB7 high as a busy flag that can be read by the MPU. When the controller completes its internal operation, it resets the busy flag.

Figure 17.5(b) shows the timing diagram of the read and write operations. It shows that after selection of a register (IR or DR) by RS, the Read/Write signal is asserted, and the enable signal is pulsed low-high-low. At the trailing edge of the enable signal, the controller sets the flag (data line DB7) and executes the internal operation of either writing a command or displaying a character. During the internal operation, the MPU should continue to check the busy flag (DB7) by asserting the enable signal (E) low-high-low until the controller resets the flag. When DB7 is reset, the MPU can write the next command or data (character).

In Figure 17.5, the LCD is interfaced by using the 8255 that is selected by the signal O_1 of the decoder in Figure 17.2. Therefore, the Hex addresses of ports A, B, C, and the control register are 84, 85, 86, and 87, respectively; however, we will use only the symbolic addresses in writing instructions. Port B is set up in I/O mode (Mode 0), the output mode to send a character to the LCD and the input mode to read the busy flag (DB7). Port C is set up in BSR mode; three signal lines of port C (PC_0 , PC_1 , and PC_2) are used for control signals (E, RS, and R/W).

To display a message, we need to initialize the LCD controller by writing into the IR register; the list of commands is given in Appendix D. The list includes various commands such as clearing the display, returning the cursor home (at the beginning), and shifting the cursor automatically after writing a character. After the initialization, we can write an ASCII character into the R/W memory of the LCD called DD RAM (Display Data R/W Memory) or an internally generated character called CG RAM (Character Generator R/W Memory). Each R/W memory location has an address. For example, the address of the first DD RAM of the first line is 80H and of the second line is C0H. This gives the capability of writing in any specific locations and generating signboard-type displays.

PROGRAM

(See Appendix D for LCD command codes.)

	MVI A, 10010000B	;Control word to set up port A as an input ; and port B as an output in Mode 0
	OUT CNTRL	;Send to 8255 control port
	MVI A, 00000000B	;Reset EN signal PC_0 low
	OUT CNTRL	
INIT:	MVI A, 00110000B	;Precautionary code if power on conditions are ; unmet
	MOV B, A	;Save the code
	CALL OUTPUT	;DB7 cannot be checked before the above in- ; struction
	CALL CMDOUT	;Send to port B two more times

	CALL CMDOUT	
	MVI A, 00111000B	;Function set code (38H) for 8 bits, 2 lines,
	CALL CMDOUT	;
	MVI A, 00001000B	5X7 dots
	CALL CMDOUT	;Code (08H) for display off
	MVI A, 00000001B	
	CALL CMDOUT	;Code (01H) for clear display
	MVI A, 00000010B	
	CALL CMDOUT	;Code (06H) for entry mode set: shift and in-
	MVI A, 0000110B	crement cursor
	CALL CMDOUT	;Code (0EH) to turn on display, cursor, and
	MVI A, 00001110B	;
	CALL CMDOUT	blink
DISPLAY:	MVI A, 87H	;Write to eighth location of DD Ram address
	CALL CMDOUT	
	LXI H, MESAGE	
NEXT:	MOV A, M	;Point to MESAGE
	CPI 00	;Get a character
	JZ END	;Is this the end of characters
	CALL DTAOUT	;If yes, end of character display
	INX H	;Send character to LCD
	JMP NEXT	;Point to the next character
END	HLT	;Go back to get next character
MESAGE	DB "Hello!", 00	
CMDOUT:	MOV B, A	;Message Hello terminated in Null character
	CALL CHKDB7	
OUTPUT:	MVI A, 00000010	;Save the command code
	OUT CNTRL	
	MVI A, 00000100B	;Select command register PC ₁ = RS = 0
	OUT CNTRL	
	MVI A, 00000001B	;Enable Write. PC ₂ = R/W = 0
	OUT CNTRL	
	MOV A, B	;Set EN high
	OUT PORTA	
	MVI A, 00000000B	;Get code
	RET	;Send out on data bus
DTAOUT:	MOV B, A	;Set EN low
	CALL CHKDB7	
	MVI A, 00000011B	;Save data byte
	OUT CNTRL	;Select data register PC ₁ = RS = 1
	MVI A, 00000100B	
	OUT CNTRL	;Enable Write. PC ₂ = R/W = 0
	MVI A, 00000001B	
	OUT CNTRL	;Set EN high
	MOV A, B	
		;Get data byte

	OUT PORTA	
	MVI A, 0000000B	;Set EN low
	RET	
CHKDB7:	MVI A, 10010010B	;Set up port B as input port
	OUT CNTRL	
	MVI A, 00000010B	;Select command register PC ₁ = RS = 0
	OUT CNTRL	
	MVI A, 00000101B	;Enable Read. PC ₂ = R/W = 1
	OUT CNTRL	
READ:	MVI A, 00000001B	;Set EN high
	OUT CNTRL	
	IN PORTB	;Read port B and check DB7
	RLC	;Place DB7 in carry flag
	MVI A, 0000000B	;Set EN low
	JC READ	;If DB7 = 1, go back and read again
	MVI A, 1000000B	;Set up port B as an output again
	OUT CNTRL	
	RET	

Program Description The program initializes port B as an output port and resets the EN signal. The LCD has certain power-on conditions; therefore, initial precautionary code 30H is sent in case those power-on conditions are unmet. Before this code is sent out, bit DB7 (flag) cannot be checked. After the first instruction, the program checks bit DB7 before it sends out any code. The first six commands initialize the LCD for a given operation. The command at location DSPLAY loads 87H in the accumulator. The DD RAM address begins at 80H; this command selects the eighth RAM location (approximately near the center of the 20-character line) to write. The loop beginning at NEXT sends out six ASCII characters in a sequence. After each character, the program checks bit DB7 by calling the subroutine CHKDB7, and when the bit is low, it calls the DTAOUT subroutine. The subroutines CMDOUT and DTAOUT are almost identical except for the RS (Register Select) signal; it must be low for commands and high for data. The subroutine CHKDB7 initializes port B as an input port and checks DB7. It continues to check DB7 until it goes low and then reinitializes port B as an output port.

INTERFACING A MATRIX KEYBOARD

17.2

A matrix keyboard is a commonly used input device when more than eight keys are necessary, rather than a row of keys as illustrated in Chapter 15. A matrix keyboard reduces the number of connections, thus the number of interfacing devices required. For example, a keyboard with 16 keys, arranged in a 4×4 (four rows and four columns) matrix as

shown in Figure 17.6, requires eight lines from the microprocessor to make all the connections instead of 16 lines if the keys are connected in a linear format. When a key is pressed, it shorts one row and column; otherwise, the row and the column do not have any connections. The interfacing of a matrix keyboard requires two ports: one output port and one input port. Rows are connected to the output port, and the columns are connected to the input port. The schematic in Figure 17.6 shows 8-bit I/O ports; they are capable of interfacing a matrix keyboard as large as 64 keys: eight columns and eight rows.

In a matrix keyboard, the major task is to identify a key that is pressed and decode the key in terms of its binary value. This task can be accomplished through either software or hardware. In this section, we will explore both methods: first we will discuss basic concepts in identifying a key pressed and, then, write subroutines to check, identify, and decode (interpret) the key that is pressed. Finally, we will illustrate how these functions can be replaced by a hardware device, such as the National Semiconductor keyboard encoder MM74C923.

17.2.1 Problem Statement

1. Interface a 20-key matrix keyboard using ports B and C of the 8255A shown in Figure 17.3.

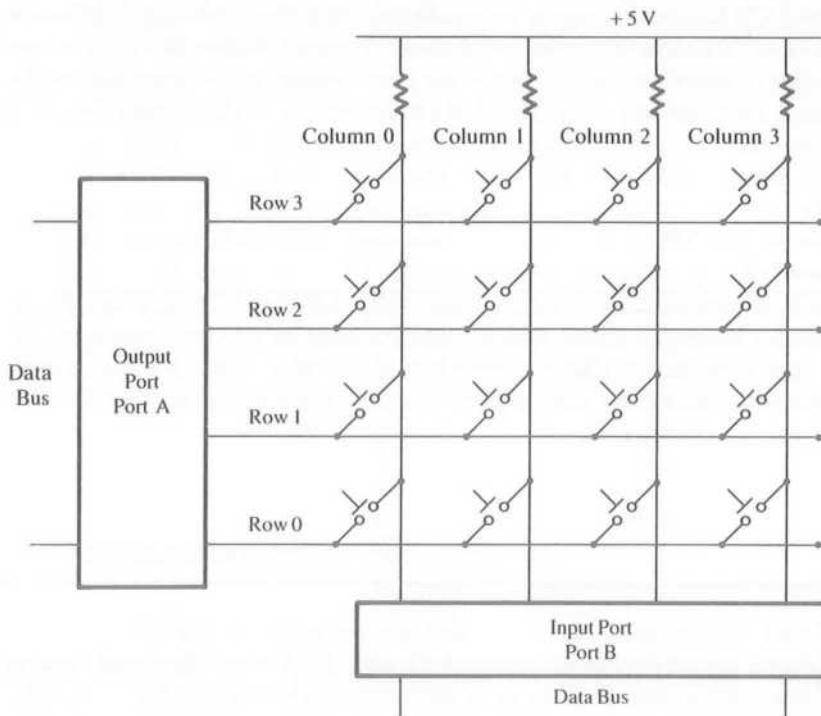


FIGURE 17.6
Matrix Keyboard Connections

2. Use the same address decoding circuit shown in Figure 17.2.
3. Write a keyboard subroutine with a software key debounce to read the keyboard, and return the equivalent binary code of the key pressed in the accumulator.

17.2.2 Interfacing a Matrix Keyboard

Figure 17.7 shows a matrix keyboard with 20 keys; the keyboard has five rows and four columns. The first 16 keys in a sequence will represent data 0 to F in Hex, and the remaining four will represent various functions such as Store and Execute. The circuit shows that the rows are connected to port C and the columns are connected to port B of the 8255A. To avoid any confusion, let us assume that port C in Figure 17.7 is different from port C in Figure 17.3 even if we are using the same address-decoding circuit. Later, we will discuss how the same I/O lines can be used for scanning both LEDs and keys. Furthermore, we will not repeat the discussion of initialization of the 8255A; we have already set up port B as an input port in Mode 0 in the previous illustration.

In Figure 17.7, the columns and rows make contact only when a key is pressed; otherwise, they remain high (+5 V). When a key is pressed, the key must be identified by its

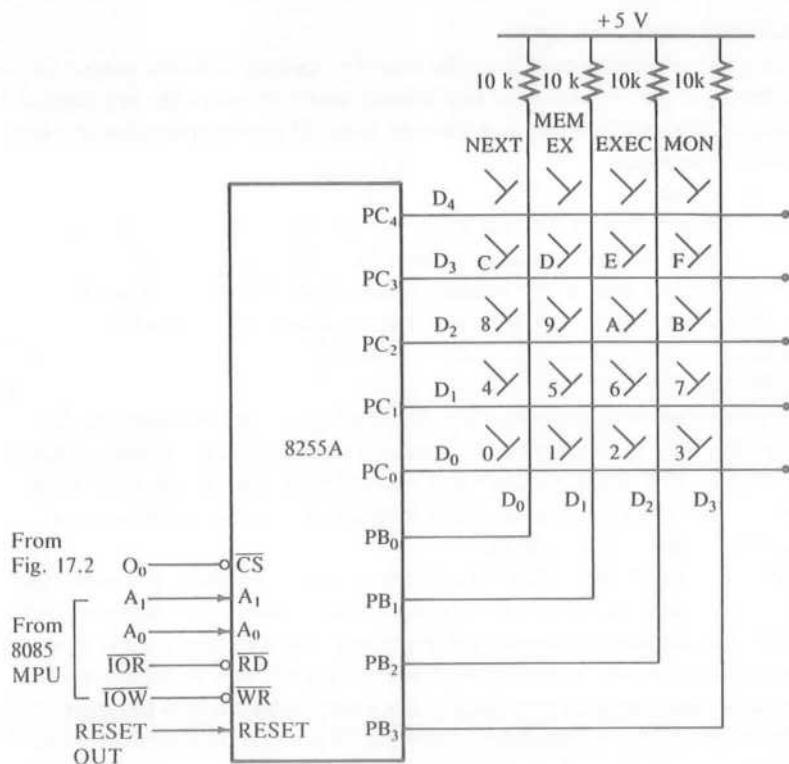


FIGURE 17.7
Interfacing a Matrix Keyboard

column and the row, and the intersection of the column and row must change from high to low. This can be accomplished as explained in the following steps:

1. Ground all the rows by sending logic 0 through the output port.
2. Check the columns by reading the input port. If no key is pressed, all columns remain high. Continue to repeat Steps 1 and 2 until the reading indicates a change.
3. When one of the keys is pressed, the corresponding column goes low; identify and decode the key.

17.2.3 Program

The matrix keyboard routine is conceptually important: it illustrates how to set up relationships between hardware binary readings and expected codes. For example, when key 0 is pressed, the input reading at port B will be XXXX1110 (D_3-D_0); however, the binary code for that key must be 00000000. This conversion is performed by the software routines illustrated below. Similarly, when key NEXT is pressed, the input reading for bits D_3-D_0 will be the same as for the 0 key (1110). Again, the software routines will have to differentiate between data and function keys.

This matrix keyboard problem can be divided into four steps:

Step 1: Check whether all keys are open.

In this step, the program grounds all the rows by sending 0s to the output port. It reads the input port to check the key release, and debounces the key release by waiting for 10 ms. This step is necessary to avoid misinterpretation if a key is held for a long time.

Step 2: Check a key closure.

In this step the program checks for a key closure by reading the input port. If all keys are open, the input reading on data lines D_3-D_0 should be 1111, and if one of the keys is closed, the reading will be less than 1111. (Data lines D_7-D_4 are not connected; therefore, the data on these lines should be masked.)

Step 3: Identify the key.

This is a somewhat complex procedure. Once a key closure is found, the key should be identified by grounding one row at a time and checking each column for zero. Figure 17.8 (Step 3) shows that two loops are set up: the outer loop grounds one row at a time, and the inner loop checks each column for zero.

Step 4: Find the binary key code for the key.

The binary key code is identified through the counter procedure. For each row, the inner loop is repeated four times to check four columns, and for every column check, the counter is incremented. For five rows, the inner loop is repeated 20 times and the counter is incremented from 0 to 13H, thus maintaining the binary code in the counter. Once the key is identified, the code is transferred from the counter to the accumulator. The codes 0 to F are used for data keys and the remaining codes 10H to 13H are assigned various functions, as shown in Figure 17.7.

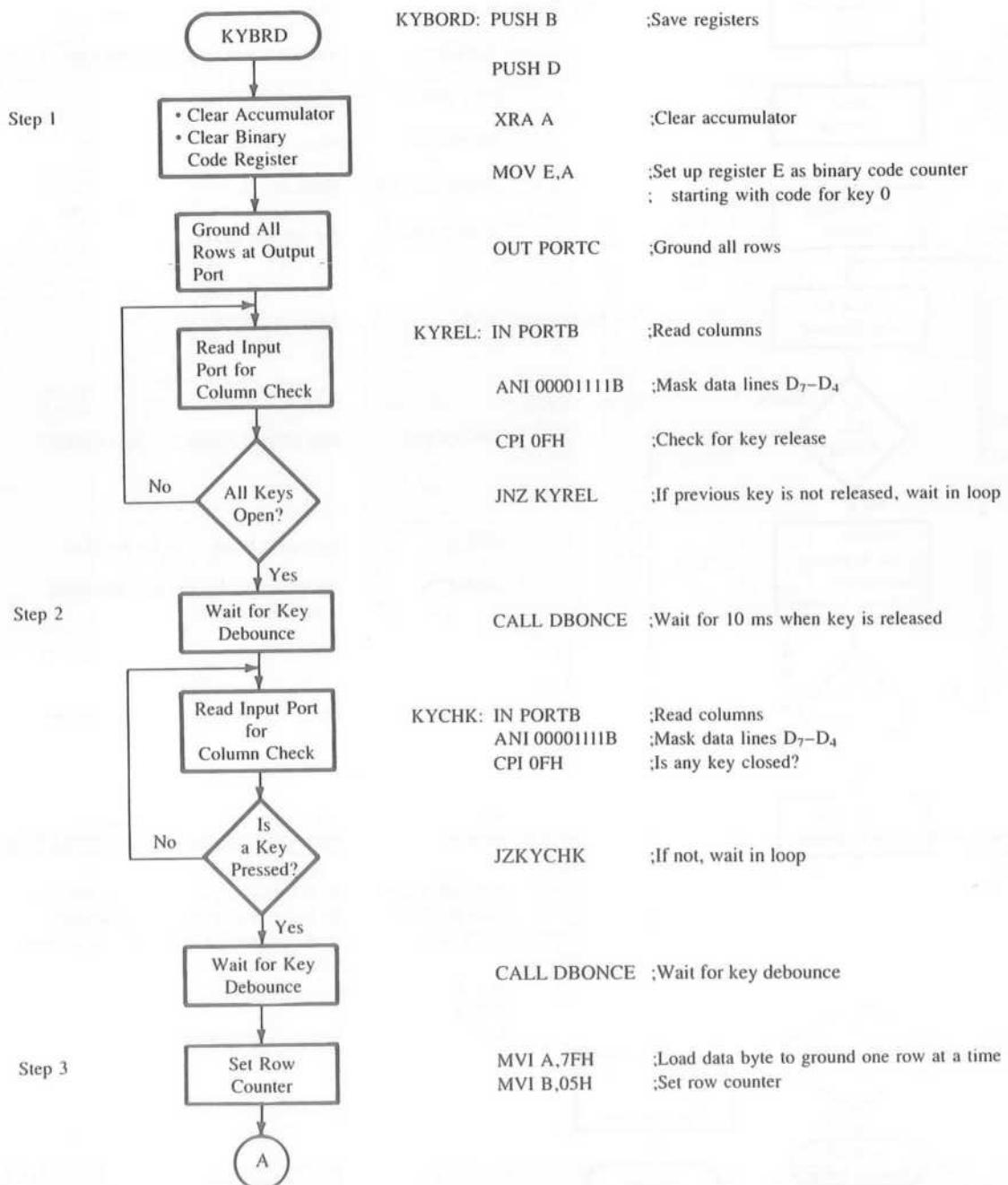


FIGURE 17.8, PART ONE
Flowchart: Matrix Keyboard Subroutine

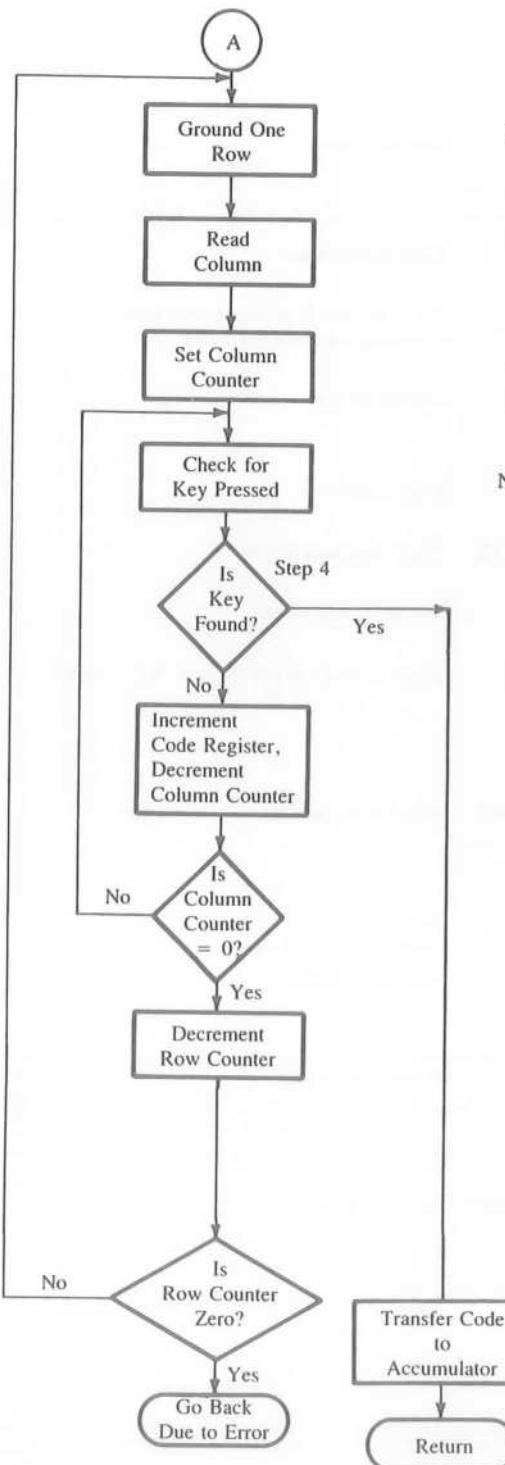


FIGURE 17.8, PART TWO
(continued)

```

DBONCE: ;This is a 10 ms delay routine, does not destroy any register content
;Input: None
;Output: None
PUSH B           ;Save registers
PUSH PSW
LXI B,COUNT    ;Load 10 ms delay count
LOOP: DCX B      ;Repeat loop for delay
        MOV A,C
        ORA B          ;Set zero flag if BC = 0
        JNZ LOOP
        POP PSW
        POP B
        RET

```

FIGURE 17.8, PART THREE

(continued)

KEYBOARD SUBROUTINE

;This subroutine checks a key closure in the keyboard, identifies the key, and supplies the
; corresponding binary code in the accumulator. It does not modify any register
; contents.

;Input: None
;Output: Binary key code in the accumulator
;Calls DBONCE, a 10 ms delay subroutine

PROGRAM DESCRIPTION

This keyboard routine saves register contents of the calling program and clears registers A and E. Register E is used as a binary code counter for the keys; it begins with the code of the 0 key. The OUT instruction grounds all the rows, and the IN instruction reads the columns. The ANI instruction masks the data on lines D₇-D₄ because they are not being used for this keyboard.

The next instruction, CPI 0FH, checks whether the previous key pressed has been released; this is a precautionary step against someone holding a key for a long time. If all keys are open, D₃-D₀ will be high, the reading will be 0FH, and the Compare instruction will set the Zero flag; otherwise, the routine stays in the loop KYREL until all keys are open. The subroutine DBONCE eliminates the key bounce by waiting for 10 ms.

Once all keys are open, the routine reads the columns to check for a key closure. If any of the keys is closed, one of the columns will be at logic 0, and the routine will skip the KYCHK loop. The DBONCE routine will debounce the key closure. At this point, a key closure is found, but the key is not identified. For example, if the reading on data lines D₃-D₀ is 1110, any of the keys in Column 0 may have been pressed. Therefore, the next step is to identify the key.

To identify the key pressed, one row is grounded at a time, beginning at Row 0. The byte 01111111 (7FH) is loaded into the accumulator and rotated left (RLC) by one position; the byte is converted to 11111110. This byte is sent to port C to ground Row 0. Then, port B is read, and each column is checked for logic 0 by rotating the reading into

the CY flag. Register C is set up to count four columns, and by rotating the byte to the left four times, each column is checked for logic 0 in the loop labeled NXTCOLM. As each column is being checked, the code counter (register E) is incremented in each iteration. For example, when Row 0 is grounded, four keys, 0 through 3, are checked, and the code counter is incremented from 00 to 03H.

After checking the columns in Row 0, the program loops back to location NXTROW and grounds the next row by sending the code that was previously saved in register D. Register B is set up as a row counter to count five rows. For each row, the loop NXTCOLM is repeated four times; thus, all 20 keys are checked, and for each iteration the code counter is incremented. When the key closure is found and the key is identified, the program jumps to location CODE. The routine copies the key code into the accumulator and returns to the calling program.

17.2.4 Combining the Matrix Keyboard and the Scanned Multiplexed Display

A display and a keyboard are two devices that are commonly used in microprocessor-based products. To reduce the cost and the chip-count in a product, a matrix keyboard and a scanned display are combined often. For example, in our previous illustrations, we can use port C to scan the display as well as ground the keys. If we use the scanned multiplexed display with the software-driven matrix keyboard, the keyboard subroutine must be coupled with the display; otherwise, the display may go off. For example, when the subroutine is waiting for a key to be pressed, the display cannot be refreshed by turning on and off digits in a sequence at a regular interval. Therefore, the program must alternate between refreshing the display and checking a keyboard to find a key pressed. The time needed for the keyboard subroutine to check a key is relatively short; therefore, it does not affect the display. Another approach is to interface the keyboard using the interrupt technique. In this approach, the program continues to refresh the display until the interrupt signal is received; then, the program checks the keyboard, processes the key, and goes back to the display.

17.2.5 Hardware Approaches to Matrix Keyboard and Scanned Display

The hardware approach reduces the software and allows the MPU to perform other tasks; however, it may increase the unit cost of the product. One approach is to use a logic device, such as the National Semiconductor MM74C923 keyboard encoder. This keyboard encoder can sense a key closure, debounce the key, provide the binary code of the key, and generate an interrupt. The other approach is to use a programmable device, such as the Intel 8279 keyboard/display interface. The 8279 performs two tasks: one task is to detect and encode a key (this is the same as that of the National Semiconductor keyboard encoder), and the other task is to refresh a scanned display (capable of displaying 16 bytes). We discussed the 8279 in Chapter 14 (Section 14.3); now we will illustrate how to interface a matrix keyboard using the MM74C923 keyboard encoder.

MM74C923 KEYBOARD ENCODER

This is a 20-key encoder with columns and rows (Figure 17.9). The respective columns and rows of a matrix keyboard must be connected to the columns and rows of the encoder. The encoder includes the chip select and the interrupt logic. The decoded address line (I/O Select) is connected to the OE signal of the encoder; it does not require an 8255A device or an input buffer. It has five output lines that provide the binary code of a key closure.

Figure 17.10 shows the schematic for interfacing a 20-key matrix keyboard using the encoder. The keyboard is assigned the port address by connecting the output O_5 of the decoder from Figure 17.2. Thus, the keyboard can be accessed by the port addresses 94H to 97H; the address lines A_1 and A_0 are left as don't care.

When a key is pressed, the encoder debounces the key and checks again for a valid key. If a valid key is detected, the encoder generates an interrupt and places the binary code of the key on the output lines. When the MPU acknowledges the interrupt and reads the binary code, the encoder turns off the interrupt. In this interfacing, the keyboard routine is reduced to a few instructions that involve reading the keyboard and storing the code in the input buffer. This technique reduces a considerable software overhead of the MPU. Therefore, the MPU can continue to scan the display until an interrupt request is received. Then, it can process the key and go back to the scanned display routine.

MEMORY DESIGN

17.3

Microprocessor-based products that require inputs from the user generally include two types of memory: Read-Only memory (ROM or EPROM) and R/W memory. ROM is used to store permanent or system programs and R/W memory is used to enter data and parameters and as stack. In memory design, we should be concerned about the size of the memory chips required and their memory maps, future expandability, and access time. Additional consideration is the capability to program Read-Only memories in the laboratory environment. Therefore, during the product development cycle or in instructional laboratories, EPROM is commonly used.

The first consideration of the memory design is the memory size and the memory map. Programs written for instructional projects can be very easily stored in 2K memory, such as the 2716 EPROM. However, the price differences between the 2764 EPROM (8K), the 2732 EPROM (4K), and the 2716 EPROM are negligible; furthermore, the price tends to go down as the usage of a chip in the marketplace increases even if the chip has larger memory size. Therefore, we will use the 2764 EPROM ($8K \times 8$) in our design. The memory map of this EPROM should begin at memory address 0000H because the program counter is cleared to that address whenever the system is reset, and we will assume that the EPROM will be used for the system program. The memory map of EPROM with 8K bytes of memory should be placed in the range from 0000H to 1FFFH, as shown in Figure 17.11. However, there are no such restrictions for the memory map of R/W

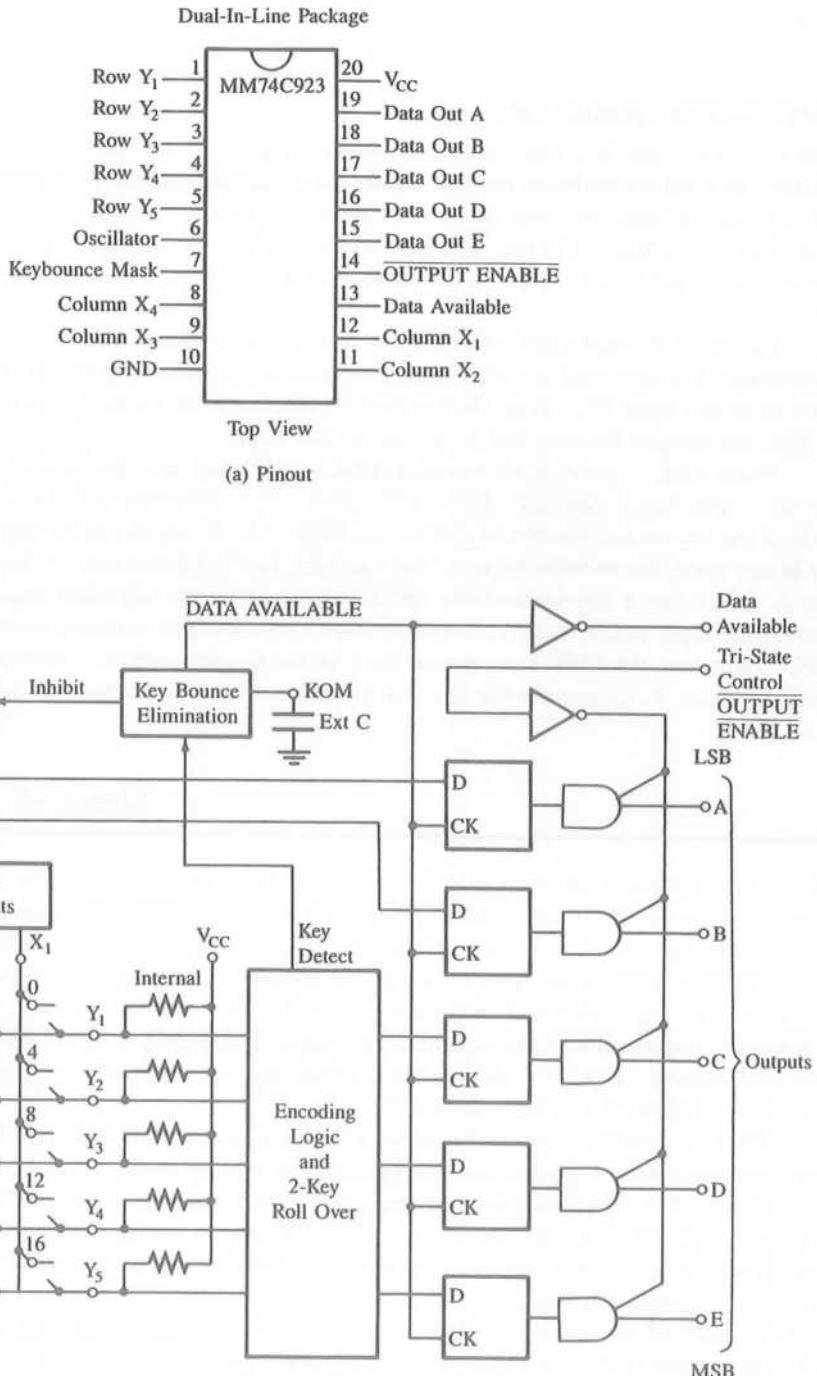


FIGURE 17.9

Keyboard Encoder MM74C923 (National Semiconductor): Pinout (a) and Block Diagram (b)
SOURCE: Reprinted with permission of National Semiconductor Corporation.

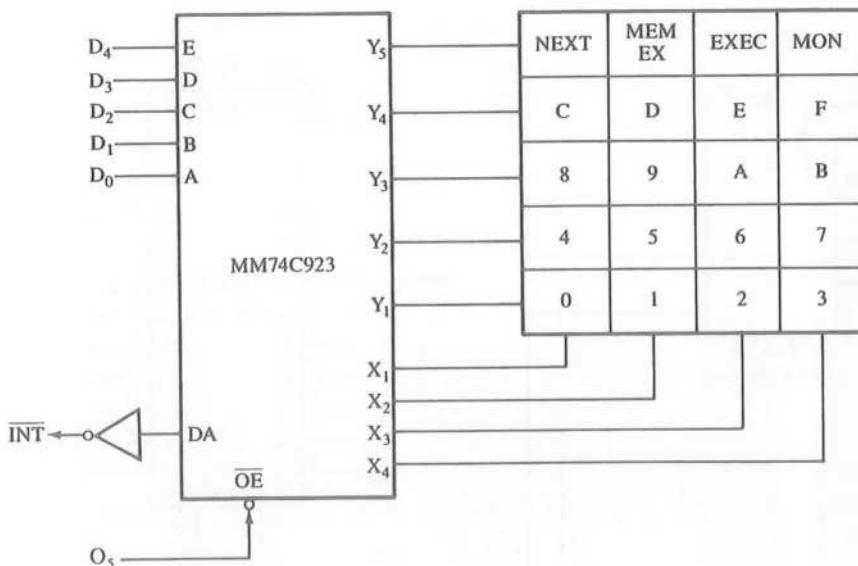


FIGURE 17.10
Interfacing 20-Key Matrix Keyboard Using the MM74C923

memory; it can be mapped anywhere as long as it does not overlap with the map of the EPROM. The other consideration is an appropriate decoding technique for memory devices with different sizes. If we use the same decoding network for devices with different sizes, such as a 3-to-8 decoder, the smaller memory chip will be left with don't care address lines; thus, it will have foldback memory addresses. However, in a small system, the foldback memory is not a serious concern.

The next consideration is future expandability. The 2764 requires a 28-pin socket; however, its pinout is designed in such a way that it is compatible with larger memory chips such as 27128 ($16K \times 8$) and 27256 ($32K \times 8$). However, this type of expansion cannot be easily accomplished by the decoding network shown in Figure 17.11; we will have to use a PROM, which can be reprogrammed to accommodate larger-size memory chips.

The last consideration is the memory access time and whether we need any Wait states in interfacing these memories. In the last decade, the memory access time has improved considerably; memory devices with 200 to 250 ns are commonly available. However, we need to calculate memory response time for a given system frequency to determine the necessity of Wait states.

17.3.1 EPROM Memory

Figure 17.11 shows the design of EPROM using the 2764 (8192×8) and the 74LS138 (3-to-8 decoder). The 13 address lines ($A_{12}-A_0$) of the MPU are directly connected to pins $A_{12}-A_0$ of the 2764 to decode 8192 memory locations. The rest of the address lines

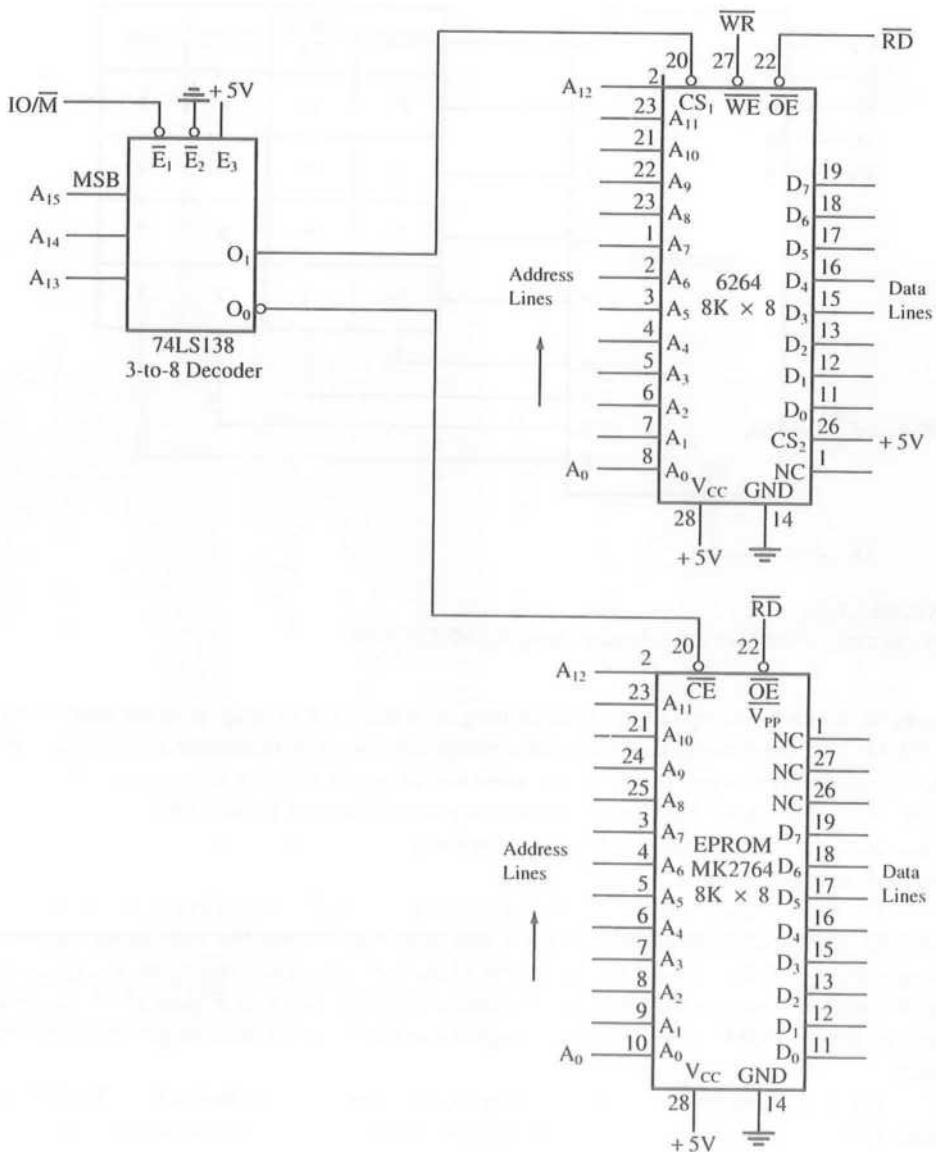
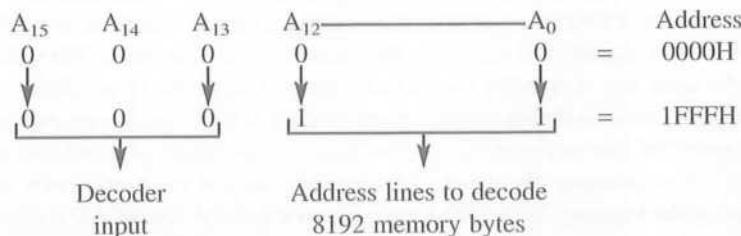


FIGURE 17.11
Schematic: Memory Design

(A_{15} – A_{13}) are decoded by the 74LS138; this provides an 8K decoding resolution for each output line of the decoder. The enable lines \overline{E}_2 and E_3 of the decoder are permanently enabled, and \overline{E}_1 is enabled by the IO/M signal of the 8085. Because \overline{E}_1 is active low, the decoder is enabled only for memory operations and it is disabled for I/O opera-

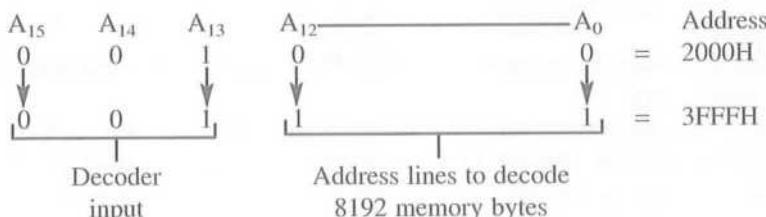
tions. The \overline{RD} signal of the 8085 can be directly connected to the \overline{OE} of the memory chip, thus eliminating the need to generate a separate MEMR signal by combining RD and IO/M.

The memory chip is accessed by connecting O_0 of the decoder to the \overline{CE} of the memory chip. When the address lines $A_{15}-A_{13}$ are at logic 0, and IO/M is active low, the decoder output O_0 is asserted to enable the memory chip. The memory address range for this EPROM is 0000H to 1FFFH, as shown below.



17.3.2 R/W Memory

The R/W memory can be designed with popular CMOS memory chips, such as 6116 (2048×8) and 6264 (8192×8). If we were to use a 2K memory chip, it would require only 11 address lines to decode the 2048 locations. To use the 3-to-8 decoder from Figure 17.11, two address lines will have to be left don't care, thus generating foldback memory space. Figure 17.11 shows the interfacing of R/W memory using the 6264 (8K) CMOS chip. This memory chip requires 13 address lines $A_{12}-A_0$, the same as in EPROM. The Chip Select (CS) logic is generated from the same decoder as for EPROM; therefore, the RD and WR signals are directly connected to the memory chip. When the O_1 line of the decoder goes low, the memory chip is enabled; thus the memory map of this R/W memory ranges from 2000H to 3FFFH, as shown below.



17.3.3 Interfacing Memory with Wait States

In interfacing memory with the microprocessor, the interfacing circuit must satisfy the timing requirements of the microprocessor and the memory chip. In Chapter 4, we assumed that memory response can match the execution speed of the microprocessor, but this assumption is invalid in some situations. Because of cost considerations, memory chips with slow response time are used occasionally in microprocessor-based systems. Therefore, it is necessary to synchronize the execution speed of the microprocessor with

the response time of memory. This can be accomplished by using the READY signal of the 8085 microprocessor.

The READY signal is an active low signal, input to the 8085 as an external request from a slow peripheral, to indicate that the peripheral is not yet ready for data transfer. When this signal is high during the Read or Write cycle, it indicates that I/O or memory is ready to send or receive data. The 8085 samples the READY line during T_2 of each machine cycle, and if the READY line is asserted low, it adds one clock period T^W as a Wait state to its machine cycle. Then it continues to sample the T^W state and adds additional Wait states until the READY signal becomes inactive. During this time, the 8085 extends the time of control signals and preserves the contents of all the buses. Thus, the READY signal can be used to synchronize the response time of any type of peripheral.

Now, to ascertain whether a given memory chip is too slow in comparison with the execution speed of the microprocessor and needs Wait states to synchronize the data transfer, we must examine the timing requirements of the microprocessor and the response time of the memory. When Wait states are not needed, the READY signal must be tied high.

WAIT STATE CALCULATIONS

Figure 17.12 shows the timing diagram of the 2764-45 EPROM with 450 ns access time; this is the time the memory takes to place the data byte on the data bus after receiving an address. The 8085 microprocessor, with a clock of 320 ns, has t_{AD} equal to 575 ns, meaning the 8085 will begin to read data 575 ns after the address is valid (see simplified timing sketch in Figure 17.13). This leaves 125 ns ($575 - 450 = 125$ ns) for delays in the decoding circuit. In Figure 17.11, the delay in the decoder will be 38 ns, and if the address bus has a driver such as 74LS244 (not shown in this figure), it will cause an 18 ns delay.

$$\begin{aligned} t_{AD} \text{ (8085)} &< t_{ACC} \text{ (memory)} + \text{Decoding delays} + \text{Bus driver delays} \\ 575 \text{ ns} &< 450 \text{ ns} + 38 \text{ ns} + 18 \text{ ns} \\ 575 \text{ ns} &< 506 \text{ ns} \end{aligned}$$

In this particular circuit, the memory 2764 with 450 ns access time is adequate, and Wait states are unnecessary.

If the system has a faster version of the 8085 microprocessor, this memory may not be adequate. For example, the 8085A-2 can operate at 5 MHz and has $t_{AD} = 350$ ns. In such a case, we can use either the faster version of the 2764 (2764-25 with 250 ns access time) or the READY signal to extend t_{AD} as shown in Figure 17.14. These decisions are generally dependent on cost consideration: the cost of the faster version of memory vs. that of a flip-flop.

In Figure 17.14, the ALE is connected to the clock of the D flip-flop (F/F_1). When the ALE goes high, the output of F/F_1 goes high. At the next clock, the output (Q) of F/F_1 goes low, which pulls the READY signal low and resets F/F_1 . This makes Q high in the next clock period. Thus, the output (\bar{Q}) is low only for one clock period, adding one Wait state (clock = 200 ns) to every machine cycle, as shown in Figure 17.15. This technique allows enough time for memory to place data on the data bus.

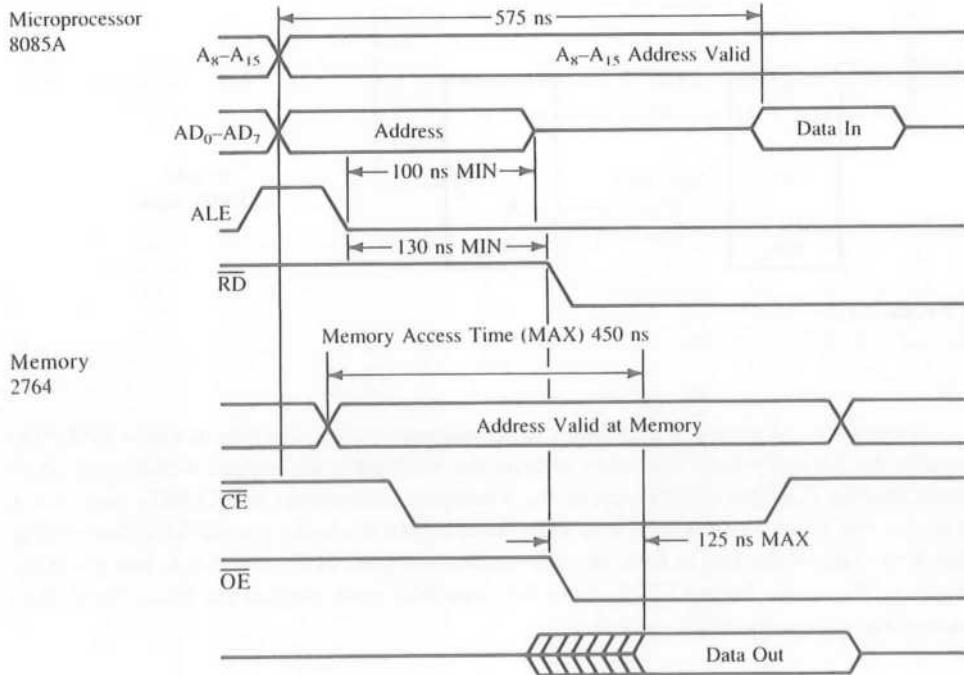
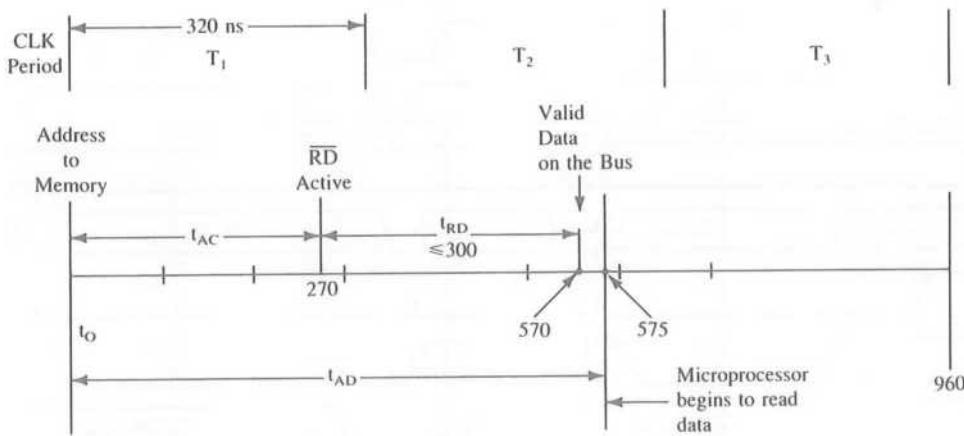


FIGURE 17.12
2764 Memory Chip and Its Read Timing



t_{AC} = Address to \overline{RD}

t_{RD} = \overline{RD} to Valid Data

t_{AD} = Address to Valid Data In

FIGURE 17.13
Typical Timings for an 8085 System with 320 ns Clock Period

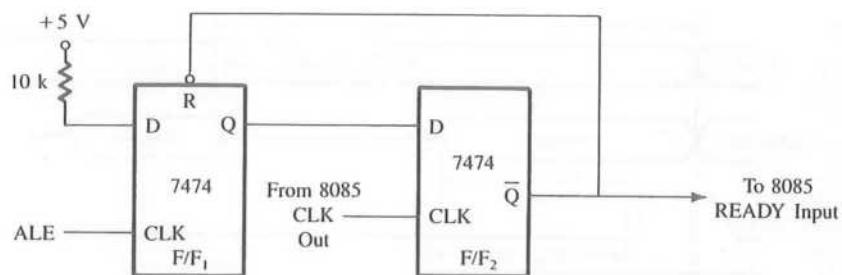


FIGURE 17.14
Circuit to Add One Wait State

Figure 17.15 shows two Memory Read machine cycles. The first machine cycle represents the Memory Read operation without the Wait state; the second includes one Wait state. During T_2 of the machine cycle, the microprocessor checks the READY line; if it is low, the microprocessor inserts one Wait state. Again it checks the READY line during the Wait state. If the line is high, the microprocessor goes to T_3 , and if it is low, it inserts the next Wait state. Figure 17.15 shows how one Wait state extends the Read signal, thus extending t_{AD} by one clock period.

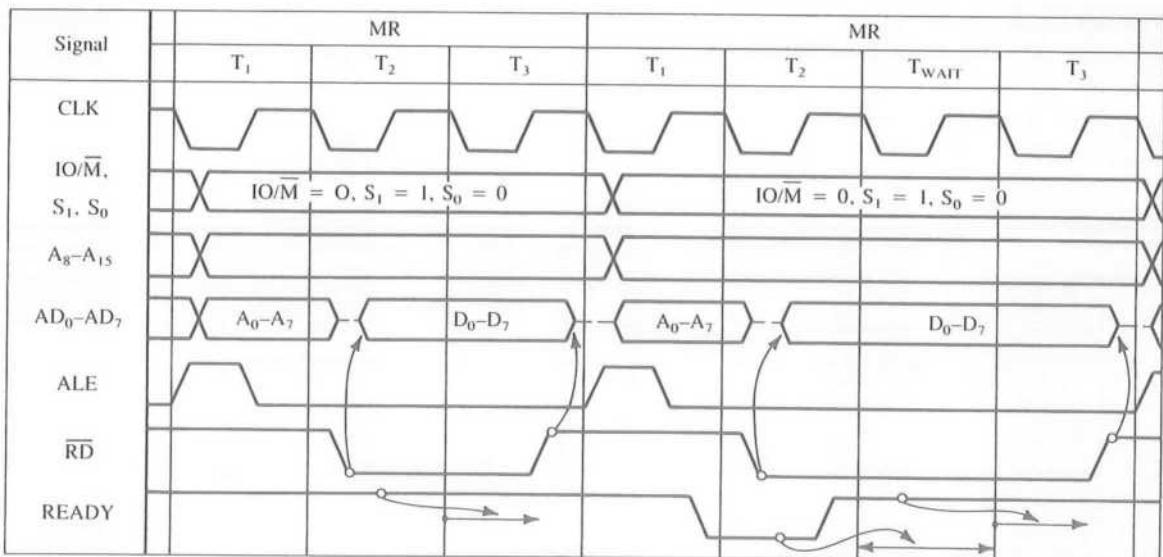


FIGURE 17.15
Memory Read Machine Cycles (with and without WAIT States)

SOURCE: Intel Corporation, *Family User's Manual* (Santa Clara, Calif.: 1979), p. 2-11.

The MPU design is determined primarily by the system requirements, such as load on the system (driving capacity), process of data transfer (interrupt, DMA, etc.), and interfacing devices used in the system. For the time being, we assume that we will design the MPU for a general-purpose single-board microcomputer with appropriate driving capacity for the buses, and we will use general-purpose memory and I/O components.

17.4.1 8085 MPU

The MPU design can be divided into the following segments:

1. Address bus
2. Data bus
3. Control signals
4. Frequency and power requirements
5. Externally triggered signals (Reset, Interrupts, etc.)

ADDRESS BUS

The 8085 has a multiplexed bus; it must be demultiplexed so that general-purpose memory devices can be used. In addition, we must use bus drivers to provide sufficient driving capacity.

Figure 17.16 shows the 74LS244, an octal bus driver used with the high-order address bus to increase its driving capacity. Typically, the 8085 buses can source $400\ \mu A$ ($I_{OH} = -400\ \mu A$) and sink $2\ mA$ ($I_{OL} = 2\ mA$) of current; they can drive one TTL logic load. The 74LS244 driver is capable of sourcing $15\ mA$ and sinking $24\ mA$ of current.

The low-order address bus is demultiplexed by using the ALE signal (Address Latch Enable) and the latch 74LS373. At the beginning of each machine cycle, ALE goes high during T_1 (see Chapter 3, Figure 3.3), and this signal is connected to the Enable line G of the latch. As ALE goes low, the address on bus AD_7-AD_0 is latched, and the eight output lines of the 74LS373 serve as the low-order address bus (A_7-A_0). The address on the output of the 74LS373 remains latched until the next ALE signal. In addition to demultiplexing the address bus, the 74LS373 can serve as a bus driver.

DATA BUS

Figure 17.16 shows the 74LS245 as an 8-bit bidirectional bus driver to increase the driving capacity of the data bus. The 74LS245 can sink $24\ mA$ and source $15\ mA$ of current. The 74LS245 has eight bidirectional data lines; the direction of the data flow is determined by the direction control line (DIR). Figure 17.16 shows that the bus driver is enabled by grounding the enable signal (G). The direction of the data flow is determined by connecting the RD signal from the MPU to the DIR signal. When the MPU is writing to peripherals, the RD is high and data flow from the MPU to peripherals. When it is reading from peripherals, the RD is low and data flow toward the MPU.

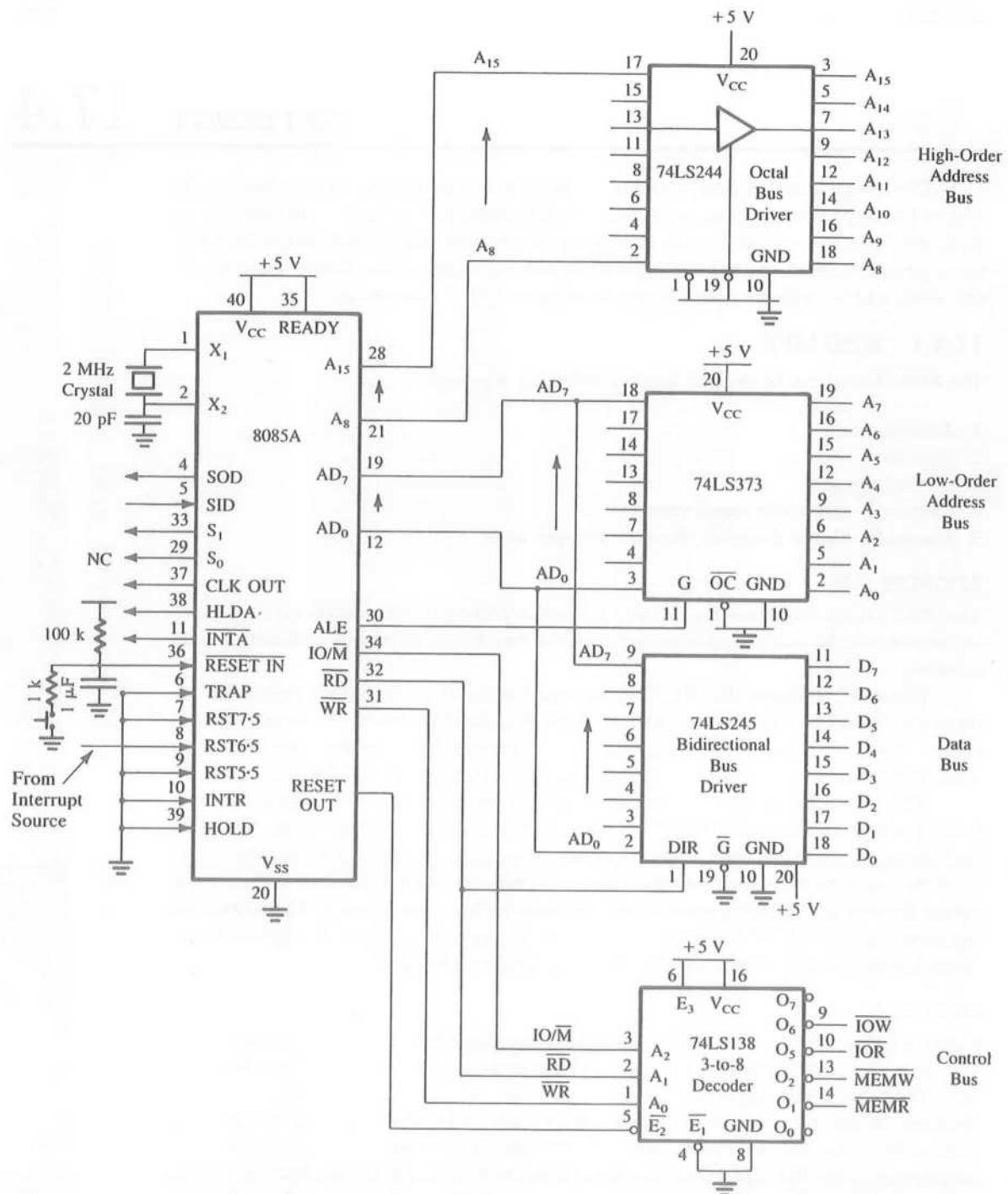


FIGURE 17.16

Schematic of the 8085 MPU with Demultiplexed Address Bus and Control Signals

CONTROL SIGNALS

The 8085 generates three signals: $\overline{IO/M}$ (I/O or memory), \overline{RD} (Read), and \overline{WR} (Write). The $\overline{IO/M}$ signal differentiates between I/O and memory functions. When $\overline{IO/M}$ is high, it is an I/O-related function; when $\overline{IO/M}$ is low, it is a memory-related function. Therefore, by combining $\overline{IO/M}$ with RD and WR signals, appropriate control signals can be generated.

Figure 17.16 shows that three signals— $\overline{IO/M}$, \overline{RD} , and \overline{WR} —are used as inputs to the 74LS138 3-to-8 decoder to generate four control signals— \overline{IOR} , \overline{IOW} , \overline{MEMR} , and \overline{MEMW} . These signals can be used for interfacing with any peripherals.

FREQUENCY AND POWER REQUIREMENTS

The 8085A can operate with a maximum clock frequency of 3 MHz. To obtain 3 MHz operating frequency, the clock logic should be driven by double the desired frequency (6 MHz). The 8085 has two clock inputs: X_1 and X_2 at pins 1 and 2. These inputs can be driven with a crystal, an LC tuned circuit, or an RC network.

Figure 17.17(a) shows a 2 MHz crystal with a 20 pF capacitor to drive the clock inputs. This input frequency is divided in half internally, and the system will run on 1 MHz clock frequency. The capacitor is required to assure oscillator start-up at the correct frequency. Figure 17.17(b) shows an alternative method of providing a clock input using an RC network.

The 8085 and other components used in this system require one power supply with +5 V. The current requirement of the power supply is determined primarily by the display load and the peripherals of the system; the MPU and memory components of the system require less than 400 mA.

EXTERNALLY TRIGGERED SIGNALS

As discussed in Chapter 3, the 8085 has provision for four external input signals: Reset, Interrupt, Ready, and Hold. Of these signals, RESET and one interrupt signal (RST 6.5) are used in this system, and the others are disabled.

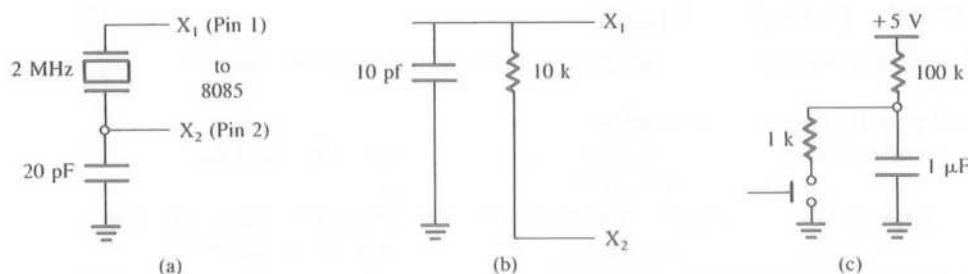


FIGURE 17.17

Clock and Reset Circuits: Clock Circuit with Crystal (a), RC Clock Circuit (b), and Reset Circuit (c)

Reset The RESETIN is an active low signal used to reset the system. When this pin goes low, the program counter is set to 0, the Interrupt Enable and HLDA flip-flops are reset, and all buses are placed in tri-state. The reset circuit shown in Figure 17.17(c) is an RC network with a sufficiently long time constant. When the Reset key is pushed, the RESETIN goes low and slowly rises to +5 V, providing sufficient time for the MPU to reset the system.

Interrupts The 8085 has five interrupt signals, all of them active high. In this system, RST 6.5 will be used; the others need to be grounded as shown in the circuit. Otherwise, the floating interrupt pins can cause the system to malfunction. To allow the use of interrupt signals for further expansion of the system, the pins can be grounded using switches.

HOLD This is an active high signal used in the DMA. This signal is also grounded in this system.

READY When this signal is high, it indicates that the memory or peripheral is ready to send or receive data. When READY goes low, the MPU enters the Wait state until READY goes high; then the MPU completes the Read or Write cycle. This signal is used primarily to synchronize slow peripherals with the MPU. To prevent the MPU from entering the Wait state, this pin is tied high.

A complete single-board microcomputer can be built from this MPU, which will be discussed in the next section.

17.5

DESIGNING A SYSTEM: SINGLE-BOARD MICROCOMPUTER

In the last four sections, we discussed the designing and interfacing of various components of a system, such as a scanned display, a matrix keyboard, and memory. We also examined general requirements of the 8085 MPU. Now we will combine these components in a general-purpose single-board microcomputer system.

17.5.1 Project Statement

Design a single-board microcomputer to meet the following specifications:

- Input: Hex keyboard with minimum of 20 keys
- Output: Six seven-segment LEDs to display memory address and data
 - : Two seven-segment LEDs to display results
- Memory: 8K of EPROM—2764 (8192×8) with 450 ns (or less) access time
 - : 8K of R/W static memory—6264 (8192×8) or equivalent
- Microprocessor: 8085A
- System Frequency: 2 MHz
- Suggested Interfacing Devices: 8255A, bus drivers, 3-to-8 decoders, key encoder, segment and digit drivers, and Hex decoder/drivers.

The system should allow a user to enter and execute programs, and the buses should have enough driving capacity to interface with additional peripherals. While machine codes are being entered, the memory address and data should be displayed by seven-segment LEDs. A two-digit seven-segment LED port should be available as an output port to display the results when programs are executed.

17.5.2 Project Analysis

In analyzing the specifications of a microprocessor-based product, it is essential to consider hardware and software simultaneously. They are interrelated, and each will have an impact on the other.

The functions of the single-board microcomputer according to the specifications (given above) can be classified in three categories as follows:

1. Check the keyboard for data or functions.
2. Display memory address, data, and results.
3. Execute programs.

KEYBOARD

The keyboard in this design is an input port with keys arranged in the matrix format. When a key is pressed, the keyboard routine should provide a binary equivalent of the key. This can be accomplished in various ways: one is the software approach, whereby a key closure is sensed, debounced, and identified and the key code is obtained by using the software. The other is the hardware approach, whereby all these key functions are performed through a programmable keyboard encoder.

The keys are divided into two groups: one group is for Hex digits from 0 to F, and the second is concerned with various functions. Basically, there are two approaches to recognizing Hex keys. One approach is to begin with Hex keys, identify the memory address, and then specify a function such as Examine Memory or Execute. In the second approach, a function is specified first and then Hex keys are accepted.

DISPLAY

This project has two types of display: the system display and the user display. The system display consists of four seven-segment LEDs for memory address and two seven-segment LEDs for data, and the user display consists of two seven-segment LEDs for results. We can explore both hardware and software approaches to design output ports for these displays.

EXECUTE

This is the simplest of the three functions and can be performed with one instruction: PCHL. When the user wants to execute a program, she/he provides the memory address where the program is stored and presses the Execute key. Assuming the memory address is stored in HL registers, the instruction PCHL simply loads the program counter with the specified memory address, and the program control is transferred from the monitor program to the user's program.

17.5.3 System Design

Figure 17.18 shows the block diagram of a single-board microcomputer. We can divide the project design into the following sections:

1. 8085 MPU design
2. Memory design
3. Display design
4. Keyboard interfacing
5. System software

We have designed these system components, except system software, in previous sections. Now our primary focus will be on system software and its implications for design of the components because of the interaction between the components in the system.

SYSTEM BUSES AND THEIR DRIVING CAPACITY

The 8085 has eight address lines A₁₅–A₈ and eight multiplexed lines AD₇–AD₀ with driving current capacity of I_{OH} 400 μ A and sinking capacity of I_{OL} 2 mA. At this point, we do not know the total load the user may have on the buses, but by examining the block diagram, we can make some reasonable estimates of the load. Figure 17.18 shows that the address bus will drive two decode circuits (I/O and memory decoders) and two memory

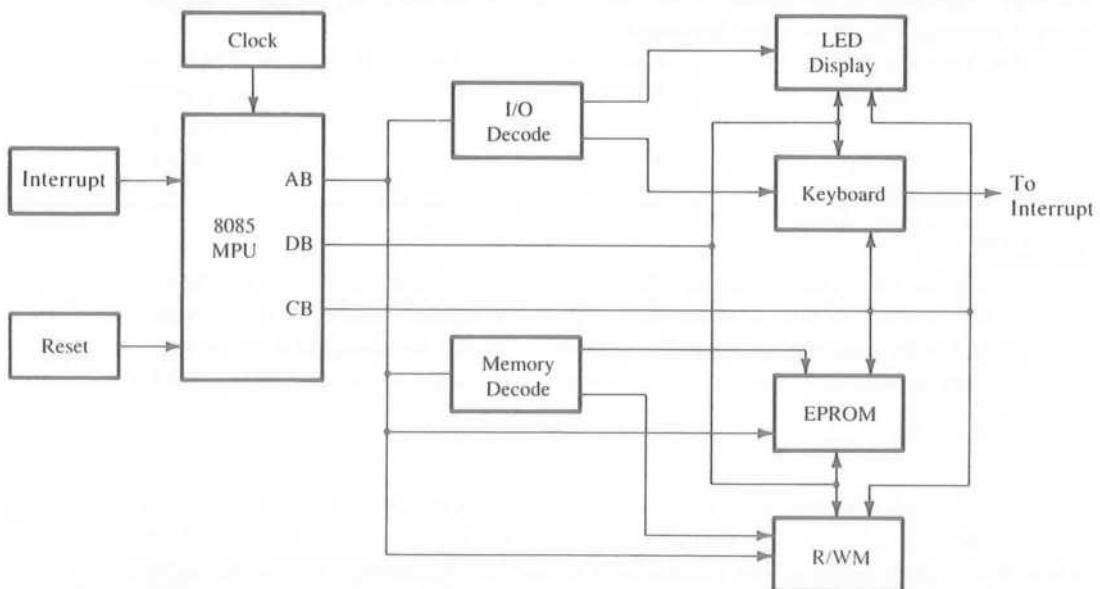


FIGURE 17.18
Block Diagram of a Single-Board Microcomputer

chips (CMOS 6264 and EPROM 2764). We can calculate the bus loading as follows (Figure 17.19):

High-level input current I_{IH}	Low-level input current I_{IL}
Two decoders = $20 \mu A \times 2 = 40 \mu A$	$400 \mu A \times 2 = 800 \mu A$
R/W memory	= $10 \mu A$
2764 EPROM	= $10 \mu A$
	$\underline{60 \mu A}$
	$820 \mu A$

By examining these load currents, we can conclude that the bus drivers are unnecessary for the address bus; we can even add a few decode circuits or gates. However, this single-board microcomputer is expected to be used for general-purpose interfacing; therefore, as a precaution, we will use the 74LS244 as a bus driver to increase the driving capacity. The 74LS244 is an octal buffer/driver, capable of sourcing 15 mA and sinking 24 mA of current. Thus, the 8085 address bus can drive additional devices (decoders, gates, etc.) with sufficient driving capacity.

The 8085 multiplexed bus has eight bidirectional lines with driving capacity similar to that of the address bus. Because the data bus is bidirectional, the loading on the bus varies considerably. When the 8085 is reading from memory, the memory chip that is enabled becomes the driving source and the microprocessor becomes the load, and when the 8085 is writing to an output port, the microprocessor is the source and the latches of the output port constitute the load. An octal latch, such as the 74LS373, requires 400 μA input current at the low-level logic; on the other hand, the 7475 requires 3–6 mA. Therefore, as a precaution, we will use a bidirectional buffer as a data bus driver. We designed the MPU with these considerations in Section 17.4.

KEYBOARD AND DISPLAYS

To interface displays and keyboards in a system, we have various approaches available to us; now we need to make some trade-offs between hardware and software and between

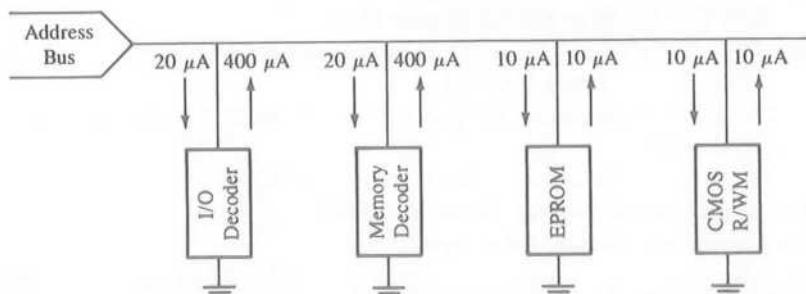


FIGURE 17.19
Loading on the Address Bus

production cost and development cost. The least expensive solution per production unit cost is to use software for refreshing the LED display and checking the keyboard. The other extreme is to use the 8279, the keyboard/display interface, to perform the refreshing of the LED display and checking of the keyboard.

In the software approach, we can minimize the interface devices by combining the matrix keyboard and the scanned display. The digit code driver of the scanned display can be used to connect the rows of the matrix keyboard; thus, we can reduce the number of I/O ports required from four to three. The program scans the display once, turns off the display temporarily by sending logic zero to the LED segments, and then grounds rows of the keyboard and checks the columns for a key closure. This approach requires one 8255A; for example, port A as the segment driver for the display, port B as an input port to read the columns of the keyboard, and port C as a digit code driver for the display as well as to ground the keys. However, we will need an additional port for the user display.

The second approach is to replace refreshing the display and checking the keyboard by a programmable device, such as the Intel 8279. This device removes the burden of scanning the display and checking the keyboard from the MPU. When a key is pressed, the MPU is informed by generating an interrupt. When the MPU reads the keyboard, it places the code in the encoder memory and informs the encoder how many LEDs to display. This simplifies the software necessary for the keyboard monitor.

For instructional purposes, we will take the middle road. We will replace the keyboard software by a keyboard encoder, such as the National MM74C923, and use software for refreshing the display. This approach reduces the software; the program has to continuously refresh the display, and, when a keyboard encoder generates an interrupt, the MPU can read the keyboard, process the key, and go back to refreshing the display. In this approach, we can use two ports of the 8255A for the scanned multiplexed display and one port for the user display. This approach can illustrate important concepts, such as interrupt, refreshing, and key encoding, without being excessively dependent on software.

Now we can summarize the specifications as follows:

1. Memory map: EPROM 0000H to 1FFFH (Figure 17.11)
R/WM 2000H to 3FFFH (Figure 17.11)
2. System display: Scanned display with six LEDs using the 8255A
Port A = 80H, port C = 82H
3. User display: Latched LED port using the port B (81H) of the 8255A and two Hex decoders 9370
4. Keyboard: 20-key matrix keyboard using the encoder MM74C923
Port is interrupt-driven with the address 84H
Four function keys and 16 data keys

On the basis of these specifications, we will illustrate an approach to software design in the next section.

SOFTWARE DESIGN

17.6

The most puzzling aspect of software design is where to begin and how to synthesize all functions in one program. The place to begin is the list of the functions to be performed. In the project analysis section, three functions are listed: check keyboard, display, and execute. The next place to look for clues is hardware. Examination of the hardware design reveals the following clues:

1. The program should begin at location 0000H.
2. Initial memory locations should be reserved for interrupt restarts.
3. Programmable peripherals need initialization instructions.
4. At the system turn-on, a message should be displayed.
5. Four keys are available to identify functions, and 16 keys are used as Hex digits from 0 to F.

By combining the functions to be performed and the clues obtained from hardware design, the task can be divided into the following steps:

1. Initialize programmable peripherals.
2. Display the sign-on message to indicate that the system is ready.
3. The keyboard is interrupt-driven; therefore, continue the display-scanning until an interrupt is generated.
4. When an interrupt is generated, turn off the sign-on message, process the key, and return to the display loop to display the new key or the error message, or blank the display.

The first three steps are fairly simple. The initialization is determined by peripheral devices and their decode logic (discussed in the previous sections). The display involves refreshing the sign-on message with the table look-up technique. The third step is to continue to refresh until an interrupt is generated.

The fourth step—determination of an appropriate key—is critical to the software design, and the appropriateness depends upon how the user is allowed to enter and execute a program. Basically, there are two approaches: one approach is to begin with a memory address and then specify the function to be performed; the second approach is to begin with a function, and then Hex keys are accepted. In addition to the Reset key, at least three keys are required: MEMEX (Memory Examine), NEXT (Next Memory Location), and EXEC (Execute). The MEMEX key allows the user to enter the memory address, examine the data stored in that memory, and enter new data. The NEXT key stores the new data byte and increments to the next memory location. The EXEC key allows the user to execute a program. With this minimum configuration, if an inappropriate sequence of keys is pressed, and the program displays the error message, it can be terminated only by the Reset key. Therefore, a key called MON (Monitor) is added to terminate a program.

In the first approach, in which the user begins with a function, we will use four keys: MON (Monitor), MEMEX (Memory Examine), NEXT (Next address), and EXEC (Execute). If the MON function is selected, the program goes to the beginning and displays a Ready message; if the EXEC function is selected, the program transfers control to the user program. If the MEMEX key is pressed, the program displays the contents of the memory; if the NEXT key is pressed, the program increments to the next memory location. After the MEMEX key, the user is allowed to enter new data or go to any one of the functions except the EXEC function.

In the second approach, the user must begin with a memory address and then specify a function; otherwise, an error message is generated.

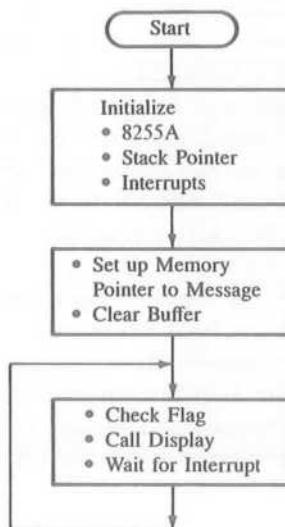
The proposed system includes a scanned multiplexed display, which needs refreshing at a regular interval, and a matrix keyboard that is interrupt-driven. Therefore, the main program revolves primarily around refreshing the display and waiting for an interrupt to occur, as shown in Figure 17.20. The primary task of the interrupt service routine is to read and process the key pressed and perform the designated function of the key. We can divide the software design into the following modules:

1. Initialize.
2. Display the sign-on message and wait for an interrupt.
3. When an interrupt occurs, read the key.
4. Decode the key and jump to the appropriate function.
5. Perform the function and return to refreshing the display.

17.6.1 Initialization

When a system is reset, the 8085 clears the program counter, and the program execution begins at location 0000H. If the system includes several sources of interrupts, the initial

FIGURE 17.20
Flowchart: Main Program



memory locations can be used for interrupts (RST 7 is 0038H). Therefore, the initialization program module can be written starting at 0040H, and the Jump instruction at 0000H can transfer the program to 0040H.

This module must initialize the programmable I/O device—the 8255A—and the stack pointer and enable the interrupts. In Section 17.1, we wrote the initialization instructions for the 8255A. The stack pointer is generally initialized at the top of the R/W memory; however, in this project, we will need six top locations as Display Buffer and one location to save input data. After reserving top locations (for example, from 3FF9H to 3FFFH) for the buffer (explained in the next section), the stack pointer can be initialized. In the 8085, RST 7.5, 6.5, or 5.5 interrupts are the simplest to implement because they do not require any external hardware; we will use RST 6.5 for this project. When an interrupt is generated by the keyboard encoder and accepted by the MPU, the program is automatically transferred to location 0034H. Thus, the keyboard service routine must begin at 0034H or a Jump instruction must be written at 0034H to locate the start of the interrupt service routine.

17.6.2 Display Module

We discussed the scanned multiplexed display routine in Section 17.1. That routine can be used for any fixed message, such as the sign-on message or the error message; the codes for these messages can be stored permanently in the EPROM. However, to display memory address and data that change with key strokes, we need to reserve memory locations: four for memory address and two for data in the R/W memory. These locations are called Display Buffer. For example, in this single-board microcomputer, the R/W memory ranges from 2000H to 3FFFH; we can reserve the last six locations 3FFAH to 3FFFH as the Display Buffer. The display routine must be modified to check the Display Buffer and get the segment code by using the table look-up technique, output the code, and turn on the corresponding digit (Figure 17.3). In addition, the routine must be informed of the number of digits to be refreshed. For example, when a memory address is entered, four digits are displayed; when the MEMEX key is pressed, six digits are displayed.

Now we need to find a way to inform the routine of the number of digits to be displayed and how to differentiate between data keys and address keys. This can be accomplished by using the flag concept. The routine that calls the codes to be displayed sets a flag when four digits are to be displayed. For example, bit D₇ in register B can be used as a flag. When D₇ is 0, the routine refreshes four memory digits; when D₇ is 1, the routine refreshes six digits. Another approach is to use the CY flag, instead of bit D₇ in register B, to perform the same function.

In block 3 of the main program (Figure 17.20), this flag concept is used. Before calling the Display routine, the program checks the flag to determine whether it should display four locations or six locations of the buffer.

17.6.3 Reading the Keyboard and Placing the Byte in the Buffer

When an interrupt request is generated by the keyboard encoder, the MPU should read the keyboard and save the reading in the input buffer (location 3FF9H). If it is a Hex key of

a memory address or of a data byte, the MPU should place the binary value of the new key in the buffer. In addition, a register pair such as DE can be used as the Memory Register to save the memory address, and register C can be used as the Data Register. However, before the key code is placed in the buffer, the previous codes must be shifted by one location; in the process, the MSD (most significant digit) must be discarded (see Programming Assignment 21, in Chapter 10).

Figure 17.21 shows the partial flowchart of the interrupt service routine, which begins with reading a key and saving the data in the input buffer. Then the program checks

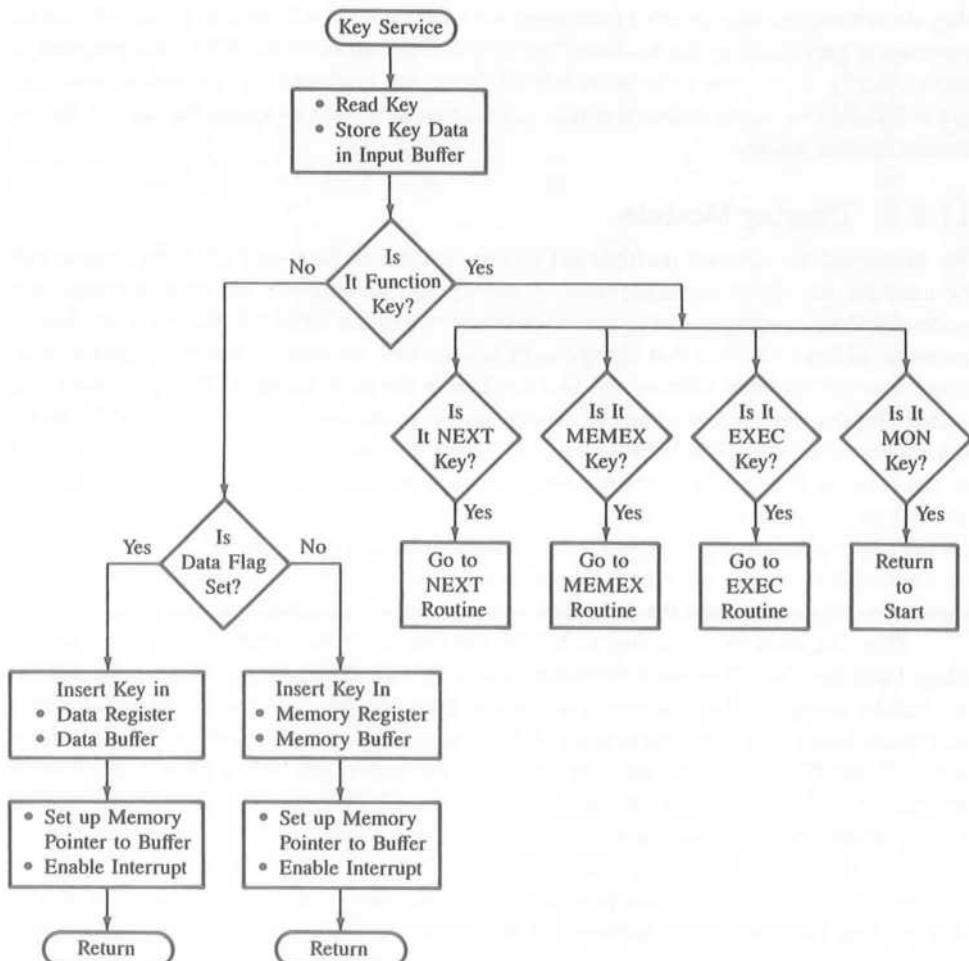


FIGURE 17.21

Flowchart: Key Service Routine

whether it is a function key or a Hex digit key; the keys with binary code 00 to 0FH are Hex digit keys; with binary code higher than 0FH, they are function keys. If the key is a function key, the program determines whether it is the MEMEX, NEXT, MON, or EXEC key and then jumps to appropriate locations. If the key is a Hex digit key, the program checks the flag (bit D₇ in register B) and identifies the key that is a part of a memory address or of a data byte. Then the program inserts the code of the new key as the least significant four bits in the Display Buffer and in the Memory Register or Data Register and returns from the interrupt service routine.

17.6.4 Performing Functions

The user must enter four Hex digits as a memory address. If more than four Hex keys are entered, the binary code of the last four is saved in register DE, and the Display buffer is updated accordingly. After entering an address, if a function key is pressed, the program checks which function key is pressed. If the MEMEX key is pressed, the memory address is already in the DE register. This address can be used as a memory pointer, and data from that memory location can be retrieved and placed in the Display Buffer (Figure 17.22). Similarly, when the NEXT key is pressed, the address in the DE register is incremented, the data byte in that location is obtained, and all six digits are placed in the display buffer.

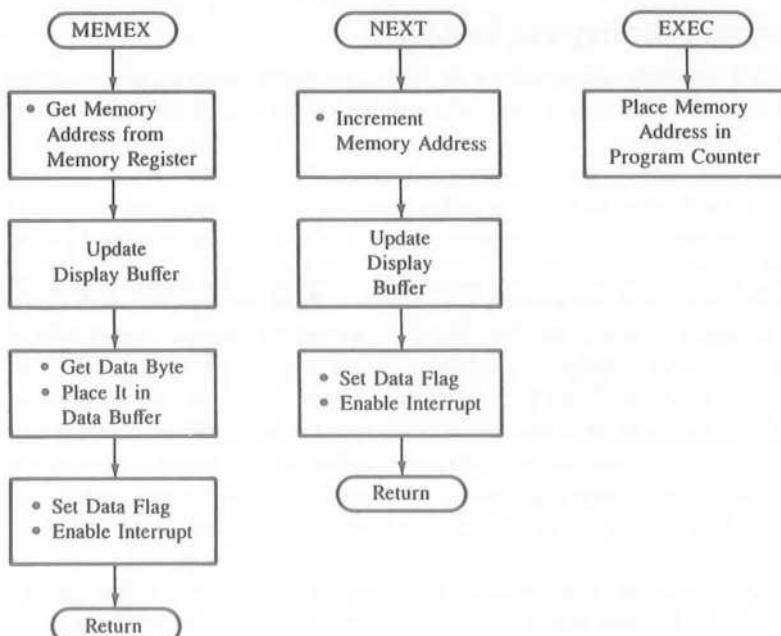


FIGURE 17.22
Function Routines

When a data byte is being entered, we can use register C to save data keys, as mentioned earlier. When a new key is pressed, the most significant nibble can be discarded, and the binary code of the new key can be entered as a least significant nibble (see Problem 3).

If the key is EXEC, the memory address where the execution should begin is already in the DE register. The program places the memory address in the program counter, and the control is transferred to the user program. If the key is MON, the program returns to the beginning and displays the System Ready message.

In writing this monitor program, the critical issue to remember is that the system uses the scanned display and needs continuous refreshing. Therefore, the main program consists primarily of calling the display routine. The next steps are to code this program in 8085 assembly language and to test it on prototype hardware, using such debugging tools as an in-circuit emulator and a logic analyzer (discussed later).

PROGRAM CODING

Assuming that program coding is to be performed by a team, it is necessary to break down the task into small, manageable, and independent modules. It is not always possible to break logic flow into independent subroutine modules. However, it is necessary to agree on symbols or labels that might be used by various members of the team; these are called global symbols.

17.6.5 Prototype Building and Testing

Microprocessor-based products are hardly ever built and tested as complete systems during the initial stages of design. If a system is completely built, it is difficult to troubleshoot. Traditional approaches, such as signal injection and isolating trouble spots, are ineffective for troubleshooting bus-oriented systems. Therefore a system is built and tested in stages. Each subsystem, such as keyboard, displays, and memory, should be built and tested separately as an independent module. Now, the question is: How to test a module without building a system? An answer can be found in such everyday incidents as testing a light bulb or starting a car with a dead battery. The light bulb can be tested by plugging it into a working socket, and the car can be started with a jumper cable. There are two principles involved in these examples: (1) borrowing resources from a working system and (2) substitution. These principles can be used in testing each separate subsystem of a microprocessor-based product. What is needed is a working system that can create an environment similar to the complete prototype system and that is generous enough to share its resources with hardware modules to be built. Such a working system is called an in-circuit emulator, and is described in Section 17.7.

Assuming that such an in-circuit emulator is available, subsystems of the single-board microcomputer can be built and tested one at a time. Similarly, as software modules are being written, they can be tested first on a software development system (discussed in Chapter 11). Finally, hardware and software can be integrated and tested using an in-circuit emulator.

DEVELOPMENT AND TROUBLESHOOTING TOOLS

17.7

In bus-oriented systems, there is constant flow of data, which continuously changes the logic states. This flow of data is controlled by software instructions. Therefore, to examine what is happening inside the system, special instruments, capable of capturing data in relation to instructions, are required. Two such instruments are discussed briefly in the next sections: In-circuit Emulator and Logic State Analyzer.

17.7.1 In-Circuit Emulator

The in-circuit emulation technique has become an essential part of the design process for microprocessor-based products. In-circuit emulation is the execution of a prototype software program in prototype hardware under the control of a software development system. To perform an in-circuit emulation, the microprocessor is removed from the prototype design board, and a 40-pin cable from an in-circuit emulator is plugged into the socket previously occupied by the microprocessor. The in-circuit emulator performs all the functions of the replaced microprocessor; in addition, it allows the prototype hardware to share all its resources, such as software, memory, and I/Os. It provides a window for looking into the dynamic, real-time operation of the prototype hardware. At present, a wide variety of in-circuit emulators are available, ranging from universal emulators with complete software development systems to stand-alone microprocessor units. Figure 17.23 shows a stand-alone in-circuit emulator (EM 188) designed by Applied Micro Systems.

EMULATION PROCESS

To test subsystems (such as I/O and memory) using an in-circuit emulator, the minimum prototype hardware required is a 40-pin microprocessor socket, without the microprocessor, and a power supply. All other resources can be borrowed from the in-circuit emulator. As more and more prototype hardware is built, fewer and fewer resources from the in-circuit emulator will be required. In the final stage, total software and hardware are integrated for testing. A hardware prototype can be viewed as a fetus growing in stages in the womb of an in-circuit emulator; until the fetus is fully developed and functioning independently, the in-circuit emulator provides the necessary environment and resources.

FEATURES OF IN-CIRCUIT EMULATOR

An in-circuit emulator is a software/hardware troubleshooting instrument. It can be a stand-alone unit or part of a software development system. A small program can be entered directly into the emulator, or a program can be transferred into the emulator from a host computer system through an RS-232 serial link. Once a program is loaded, a user can interact with the emulator through its keyboard or a terminal. The emulator has its own



FIGURE 17.23

Stand-Alone In-Circuit Emulator—EM 188

SOURCE: Photograph courtesy of Applied Micro Systems.

software commands to perform various debugging functions. The main capabilities of an in-circuit emulator can be listed as follows:

- **Downloading:** Facilities are provided to transfer programs between a software development system or a host computer and the in-circuit emulator.
- **Resource Sharing:** The in-circuit emulator allows the system being tested to share its memory and I/O ports. The memory and I/O ports of the in-circuit emulator can be assigned any addresses, thus avoiding conflict with memory and I/Os of the prototype; this is called memory and I/O mapping.
- **Debugging Tools:**
 - Breakpoints
 - Mnemonic Display
 - Real-Time Trace
 - In-Line Assembly
 - Disassembly
 - Register Display/Modifications

DEBUGGING TOOLS

The debugging tools listed above are used in troubleshooting programs. Single-stepping and setting breakpoints have already been discussed in Chapter 7. The others are briefly discussed below.

Real-Time Trace The in-circuit emulator has R/W memory used as a buffer to store the last several (such as 128) bus operations, and these can be displayed on the screen. The display is like a snapshot of all the bus operations in real time. The user can specify several requirements, such as a memory address and certain data conditions for recognizing an event, in order to trigger and display a trace. Similarly, a trace can be observed between two breakpoints or at a specified delay after a certain event. The real-time trace is a valuable tool in debugging microprocessor-based products.

In-Line Assembly This allows the user to change data or instructions while the software is in the in-circuit emulator.

Disassembly After instructions are changed in the in-circuit emulator, this facility can write mnemonics in software.

Register Display This displays the register contents after the execution of instructions.

17.7.2 Logic State Analyzer

The logic state analyzer, also known as the logic analyzer, is a multitrace digital oscilloscope especially designed to use with microprocessor-related products. In a multitrace scope, the timing relationships of several signals can be observed with respect to some triggering event or events. For example, a four-trace scope can show the timing relationships of four signals. In a microprocessor-related product, the user is interested in observing digital signals on the address bus, the data bus, the control bus and, possibly, an external instrument relative to a specified triggering event or events. Furthermore, data display should be in a conveniently readable format, such as Hex or binary. The logic analyzer performs these functions.

A typical logic analyzer designed primarily to work with the microprocessor has a 40-pin probe plus an auxiliary probe to gather external information. It includes Read-Only memory (ROM) to store instructions related to the analyzer, R/W buffer memory to store data from a product under test, a microprocessor to monitor data gathering, and a keyboard to specify operations and enter data in Hex or octal format. The analyzer can be triggered to gather information at a specified event related to the microprocessor in the product under test or in relation to an external word. The analyzer in a trace mode takes a snapshot of real-time information at a specific trigger, stores it in its buffer memory, and displays it on its CRT.

The in-circuit emulator is a valuable tool during the initial stages of product development and, in later stages, the logic analyzer can perform some of the troubleshooting functions.

SUMMARY

In this chapter, various techniques of interfacing the scanned display and the matrix keyboard were illustrated, and the trade-offs between hardware and software were dis-

cussed. Then we used the scanned display and keyboard illustrations in designing a single-board microcomputer. In addition, debugging tools such as the in-circuit emulator and the logic analyzer were introduced.

The design of the single-board microcomputer integrates all the concepts of the microprocessor architecture, software, and interfacing discussed throughout this text. In this chapter, we discussed the necessary steps in designing hardware and software. The necessary software modules were illustrated with flowcharts; however, the coding of these modules is given as assignments.

QUESTIONS, PROBLEMS, AND PROGRAMMING ASSIGNMENTS

1. Draw a schematic to interface a 16-key matrix keyboard using port C of the 8255A. Write instructions to initialize the port.
2. Draw a schematic to interface a 30-key matrix board and a six-LED scanned display using the 8255A. Combine the matrix columns and the digit-driver lines, and explain why it is possible.
3. In a key monitor program, register E is used to save 4-bit codes of two data keys. Write a subroutine to insert a new 4-bit key code that is available in the accumulator; the new code must be inserted as a low-order nibble, and the most significant nibble in register E must be discarded.
4. Write instructions to unpack the data keys in Problem 3, and place the codes in two different memory locations of the output buffer.
5. In a monitor program, register BC is used to save a 16-bit memory address. Write instructions to insert a 4-bit code of a new key in the BC register as a least significant nibble.
6. In Problem 5, unpack all the codes, and store them in four memory locations in the output buffer.
7. In Section 17.2, modify the matrix keyboard routine to accommodate 30 keys (six rows and five columns).
8. Modify the program in Section 17.1.1 to display an error message as Err and blanks.
9. Write instructions for the EXEC module, assuming the memory address where execution should begin is in register DE.
10. Write a subroutine to transfer a 16-bit address from register DE and a data byte from register C into the display buffer (3FFAH to 3FFFH); the least significant nibble of the memory address should be placed in location 3FFAH, and the least significant nibble of the data byte in location 3FFEH.
11. Write a Display subroutine that takes the unpacked memory address and the byte from the buffer, looks up the seven-segment code, sends the code to the segment driver, and scans the digit code in a sequence to display the address and the byte.

18

Extending 8-Bit Microprocessor Concepts to Higher-Level Processors and Microcontrollers

The microprocessor has had an impact on industries as diversified as machine tools, chemical processes, medical instrumentation, and sophisticated guidance control. Some applications require simple timing and bit set/reset functions; others require high-speed data processing capability. Therefore, a number of different microprocessor families are being designed to meet these diversified requirements. Microprocessors range from single-chip microcontrollers (microcomputers) to general-purpose 32- and 64-bit microprocessors. In the present state of microprocessor technology, 4-bit microprocessors are used in high-volume products for simple functions.

At the other extreme, 32- and 64-bit microprocessors are commercially available and have begun to invade the territories of traditional mainframe or large computers. This chapter will examine recent trends in this fast-changing technology and their implications for industry.

The microprocessor topics in this book have been discussed in the context of the widely used

8085 microprocessor. However, various other 8-bit microprocessors are available, with varying degrees of capability. In addition to general-purpose 8-bit microprocessors, microprocessor technology is evolving in three different directions. One direction is to design a complete microcomputer on a single chip, called a microcontroller, geared toward dedicated applications. The second is to integrate peripheral devices, such as serial I/O and DMA controllers, with the microprocessor chip. And the third direction is to design 32-bit and 64-bit microprocessors with general-purpose capability similar to that of mainframe computers.

The high-end (32- and 64-bit) general-purpose microprocessors are divided into two distinct design philosophies known as CISC (complex instruction set computing) and RISC (reduced instruction set computing). High-performance workstations are generally designed with RISC processors. The focus of this chapter is to examine similarities and differences between 8-bit processors such as

the 8085 and larger processors such as the Intel 8088 to Pentium-type processors. We will focus on: (1) how concepts learned in the context of the 8085 are applicable to larger processors; (2) limitations of 8-bit processors; and (3) new concepts underlying larger processors.

OBJECTIVES

- List important characteristics of 8-bit microprocessors contemporary to the 8085, such as the Z80, the MC6800, and the Hitachi HD64180.
- Review the concepts of 8-bit microprocessors.
- Describe important features of Intel 8088/86 16-bit microprocessors and explain the concepts of memory segmentation, parallel processing, queuing, coprocessing, and memory and I/O interfacing.
- Compare the features of the Intel 8088/86 and the Motorola MC68000 16-bit microprocessors.
- Explain the design features of 32-bit microprocessors, and describe the characteristics of the Intel 80386, 80486, Pentium, and Pro-Pentium processors.
- List important features of RISC processors.
- List the elements of a single-chip microcontroller, and compare the characteristics of representative microcontrollers such as the Intel MCS[®]-51 and the Motorola MC68HC11.

18.1 8-BIT MICROPROCESSORS CONTEMPORARY TO THE 8085

The Intel 8008, which was later superseded by the Intel 8080A, was the first 8-bit microprocessor. Just about the same time, Motorola brought out the MC6800 as an improvement over the first 8008, but with substantially different architecture. Within a few years, Zilog designed the Z80 and Intel came up with the 8085 as an improvement over the 8080A. Both are upward machine-language compatible with the 8080A. Later, National Semiconductor introduced an 8-bit microprocessor—the NSC800—combining the features of the 8085 and the Z80. In recent years, the trend seems to be toward integrated devices that reduce the chip count; these devices are known as integrated super 8-bit MPUs. One example of such devices is the Hitachi HD64180. The 8085 has been discussed throughout this text; other 8-bit microprocessors mentioned above will be discussed in the next few sections.

18.1.1 The Z80

The Z80 microprocessor is manufactured by Zilog, using N-channel MOS technology. It is upward software-compatible with the 8080A. Its instruction set has 158 basic instructions, which include the 8080A instruction set. However, the Zilog mnemonics are different from the Intel mnemonics, even though the machine codes are identical for the 8080A set. Furthermore, the Z80 instruction set does not include two serial I/O instructions (RIM and SIM) of the 8085. Figure 18.1 shows the Z80 signals and its internal registers. The Z80 is not pin-compatible with the 8080A or the 8085.

The Z80 microprocessor requires one +5 V power supply and the clock frequency of recent versions ranges from 4 MHz to 20 MHz. The Z80 chip has 16 address lines to address 64K memory and eight data lines. No lines are multiplexed.

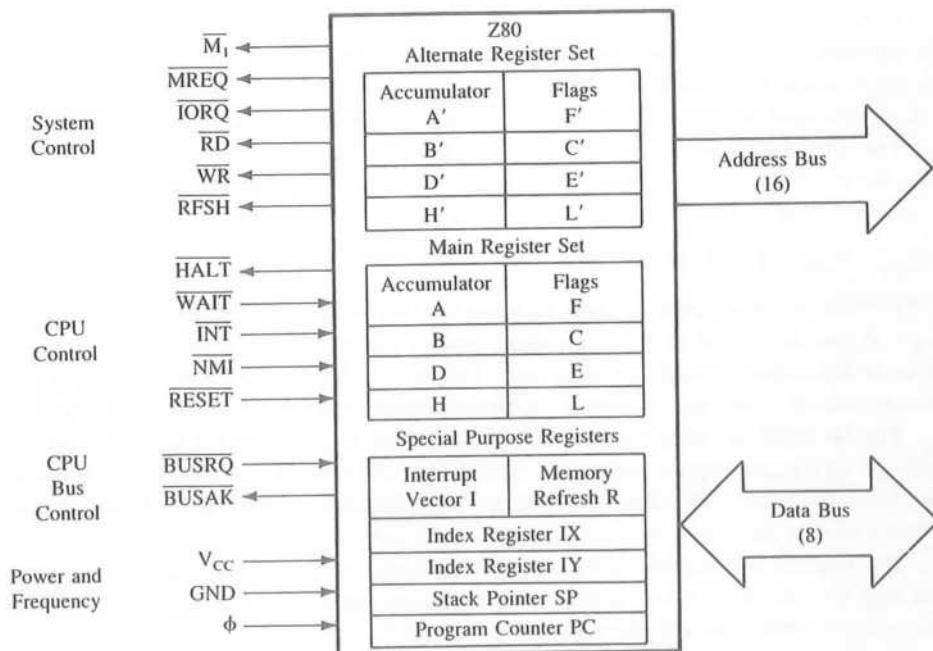


FIGURE 18.1

Z80 Microprocessor: Signals and Internal Registers

SOURCE: Adapted from Zilog, *Intelligent Peripheral Controllers* (Campbell, CA; Author, 1991).

The Z80 has two interrupt lines: one is compatible with the 8080A interrupt line, and the second is a nonmaskable interrupt (NMI). An additional significant feature of the Z80 is its on-board logic (RFSH) to refresh dynamic memories. This allows the user to use dynamic memories in a system without having to build an additional refresh circuit. Dynamic memory chips, in general, are much less expensive than static memory chips.

The internal architecture of the Z80 (Figure 18.1) includes all the 8085 registers: A, B, C, D, E, H, L, the flag register, the program counter, and the stack pointer. In addition, it has the entire set of 8-bit alternate registers, shown as A' through L'. However, access to the alternate set of registers is only through an instruction called Exchange (EXX). This instruction exchanges the contents of registers A to L with the alternate registers A' to L'. In essence, the alternate set of registers is used as temporary storage. Figure 18.1 shows two 16-bit index registers (IX and IY), one 8-bit Interrupt Vector Register (I), and one 7-bit Memory Refresh Register (R). The two index registers in the Z80 allow various types of memory addressing modes—a significant improvement over the 8085, in which memory addressing is restricted primarily to the HL register.

The instruction set of the Z80 is the most powerful set among the 8-bit microprocessors. It includes instructions to transfer data from one block of memory to another (LDIR = Load, Increment, and Repeat) and to search the entire memory for an 8-bit character (CPIR = Compare, Increment, and Repeat). Some of its Jump instructions perform

more than one function, such as Decrement B and Jump if Nonzero (DJNZ). The group of instructions called "Bit Manipulation" can test, set, or reset a bit in any register or a memory location. In addition, the Z80 has an extensive set of I/O instructions that include block input/output instructions and various modes of interrupts.

The Z80 microprocessor is supported by peripheral devices such as the parallel I/O (PIO), the clock timer circuit (CTC), the Direct Memory Access controller (DMA), and the serial I/O (SIO and DART).

18.1.2 The MC6800

The MC6800 microprocessor is manufactured by Motorola using N-channel MOS technology. It was developed at about the same time as the Intel 8080A. Both microprocessors were developed as improvements over Intel's first 8-bit microprocessor, the 8008. However, the MC6800 has a different architecture than its competitor, the Intel 8080A.

The MC6800 has 16 address lines and eight data lines. It requires one +5 V power supply—a significant improvement over the 8080A—and runs on a 1 MHz standard clock signal with two phases: Φ_1 and Φ_2 . However, the chip does not include a clock logic. It has two interrupt lines: a regular interrupt and a nonmaskable interrupt.

The internal architecture of the MC6800 includes two 8-bit accumulators and one index register. The other three registers—the program counter, the stack pointer, and the status register—are similar to the registers in the 8085.

The MC6800 instruction set includes 72 basic instructions and makes extensive use of memory referencing. The set does not include typical direct I/O instructions (IN/OUT); it has only memory-mapped I/O. It has simple timing and control signals; the clock period is the same as the machine cycle. In general, the MC6800 is a much simpler microprocessor than the 8080A. It is currently being replaced by an 8-bit version (68008) of the MC6800 family of microprocessors discussed later.

18.1.3 Hitachi HD64180

This is a Z80 type upward-compatible 8-bit high-integration CMOS microprocessor in a 64-pin package, designed for applications with low power consumption, and it can operate with a 6 MHz clock. It includes a clock generator, an interrupt controller, and a memory management unit (MMU) as support devices for the microprocessor. It has 19 address lines that can address 512K bytes of physical memory, and the MMU translates internal 64K logical addressing into appropriate physical addressing. The interrupt controller is capable of handling four external and eight internal interrupting sources.

The HD64180 includes four I/O related devices: DMA controller (DMAC—two channels), asynchronous serial communication interface (ASCI—two channels), clocked serial I/O port (CSI/O), and programmable reload timer (PRT—two channels). The DMAC has two channels that support high-speed data transfer of 64K bytes per channel anywhere in the physical space of 512K bytes of memory. The ASCI has two separate channels for full duplex communication, and the CSI/O provides a half-duplex communication; it is used primarily for simple high-speed connection between microcomputers. Similarly, the timer has two channels with 16-bit counters, and one of the channels can be used for waveform generation.

The instruction set of the HD64180 is upward-compatible with the Z80 instruction set. The HD64180 has seven additional instructions, including 8-bit Multiply and Sleep. The Sleep instruction reduces the power consumption to 19 mW. One of the powerful features of this device is that the Opcode Fetch cycle of an instruction consists of three T-states versus four T-states in the Z80, resulting in faster program execution.

This device is in many ways similar to Zilog's Z180 and Z280 processors. Examination of this device indicates that the Z80-type microprocessors reasserted their presence in the form of integrated devices in industrial applications.

18.1.4 Review of 8-Bit Microprocessors

The architectures of the Z80 and the 8085 are register-oriented. The Z80 has a larger and more powerful instruction set than the 8085, and it is software-compatible with the 8085, except for serial I/O instructions. In contrast to the register-oriented Z80 and 8085, the 6800 is memory-reference oriented. It includes fewer registers in its architecture than the 8085.

Now the question is: Are 8-bit processors obsolete? To find an answer to this question, we must look at the worldwide sales volume of various processors. The number of 8-bit units (including microcontrollers discussed later) is much larger than that of 32- and 64-bit processors. The 8-bit processor is being used in a variety of applications such as appliances, automobiles, and industrial process and control applications. The Z80 type 8-bit microprocessors have reasserted their presence in the form of integrated devices. Similarly, the 8085 processor is being used in graphic calculators. For many industrial applications, the 8-bit microprocessor is too powerful in terms of its capability, and it is rarely used to its full capacity. On the other hand, 16-bit processors (discussed in Section 18.3) are squeezed out by 32- and 64-bit processors in PC systems; they are almost obsolete. Therefore, it appears that in an educational setting, 8-bit processors are well suited to teach basic microprocessor concepts.

REVIEW OF MICROPROCESSOR CONCEPTS

18.2

In Chapter 1, we defined the microprocessor as a clock-driven, register-based, programmable digital electronic device that reads binary instructions from memory, accepts data from input devices (or reads stored data from memory), processes the data according to the instructions, and displays the results at output devices or stores them in memory.

18.2.1 Microprocessor Architecture

To understand the architecture of a microprocessor, we studied the following concepts in the context of the 8085 microprocessor:

1. address bus, data bus, control and status signals
2. execution of an instruction with reference to a clock signal and contents of various buses at different times

3. interfacing of memory and I/O devices that includes address decoding and timing of various signals
4. processes of accepting external signals (requests) such as interrupt, hold, and ready and responding to those requests

The 8085 microprocessor signals are divided into six categories: (1) address bus, (2) data bus, (3) status and control signals, (4) external requests, (5) response to external requests, and (6) power and clock. These are typical signals any processor must have irrespective of its size. However, the 8085 is housed in a 40-pin package that limits the number of signals; therefore, in this processor, the data bus is multiplexed with the low-order address bus that must be demultiplexed. On the other hand, the Z80 and the MC6800 do not have any multiplexed signals. However, the concept of multiplexing signal lines is important; it is commonly used in 40-pin devices such as microcontrollers.

18.2.2 Programming Registers

The 8085 processor has six general-purpose registers, one accumulator, one flag register, and two memory address registers or pointers (SP and PC). The 8085 lacks memory pointers such as index registers; it relies heavily on the HL register as a memory pointer. The Z80 removed that deficiency by adding two indexed registers (IX and IY). The architectures of the Z80 and the 8085 are register oriented. In contrast to the register-oriented Z80 and 8085, the MC6800 is memory-reference oriented. It has two accumulators and no general-purpose registers; it uses memory registers as its general-purpose registers.

The 8085 instruction set is divided into five groups: (1) data copy (transfer), (2) arithmetic, (3) logic and bit manipulation, (4) program transfer (jump, call), and (5) machine (processor) control. The set does not include instructions such as multiply or divide and instructions that can handle multiple operations such as copying a block of data. However, these operations can be performed by writing a set of instructions.

The architecture of Intel 16- or 32-bit processors, discussed in the following section, is in many ways similar to the architecture of the 8085. They include larger registers and buses. The instruction sets of these processors include additional instructions that make programming tasks easier, such as multiply, divide, and string manipulation. These larger processors are designed to function in a multiuser and multiprocessor environment. Therefore, these processors need additional signals and instructions. However, the concepts of the 8085 microprocessor are applicable and can be easily extended to these larger processors.

18.3

16-BIT MICROPROCESSORS

The 16-bit microprocessor families are used primarily in microcomputers and are oriented toward high-level languages. Their applications sometimes overlap those of the 8-bit microprocessor. They have powerful instruction sets and are capable of addressing megabytes of memory. Typical examples of 16-bit microprocessors include the Intel

8086/8088 and 80186/286, Zilog Z8001/8002, Motorola 68000, and National Semiconductor NS16000.

Apart from design concepts and instruction sets, one of the critical factors that decides the capability of the microprocessor is the number of pins available. In the 1970s, one trend was to stay within the 40-pin package size and take advantage of the existing production and testing facilities. The 40-pin package either limits the size of the memory that can be addressed or necessitates multiplexing of several functions. Intel (8086) and Zilog (Z8002) stayed with the 40-pin package. Another option was to go beyond the 40-pin limit. National Semiconductor (NS16000) and Zilog (Z8001) chose the 48-pin package. Motorola selected the 64-pin package for the MC68000 and Intel chose the 68-pin package for its 80286 microprocessor. The primary objectives of these 16-bit microprocessors can be summarized as follows:

1. Increase memory addressing capacity.
2. Increase execution speed.
3. Provide a powerful instruction set.
4. Facilitate programming in high-level languages.
5. Function in a multiprocessor environment.

These objectives can be met by using various design concepts. To illustrate differences in design philosophies, the following microprocessors are described briefly: the Intel 8086/8088, the Intel 80186 and 80286, and the Motorola MC68000.

18.3.1 Intel 8086/8088

The Intel iAPX 8086/8088 is a 16-bit microprocessor housed in a 40-pin package and capable of addressing one megabyte of memory. Various versions of this chip can operate with clock frequencies from 5 MHz to 10 MHz.

The 8088 is functionally similar to the 8086, except that it has an 8-bit external data bus. Its internal architecture and instruction set are identical with those of the 8086. The only difference is that a 16-bit data word must be transferred in two segments in the 8088. The 8088 can be viewed as an 8-bit microprocessor with the execution power of a 16-bit microprocessor. The next few paragraphs describe the features of the 8086/88 architecture that meet the objectives described above.

8086/8088* ARCHITECTURE

Figure 18.2 shows logical signals of the 8088 and 8086 microprocessors. These processors have identical internal architecture; they differ only in external signals. The 8086 has a 16-bit external data bus, and the 8088 has an 8-bit external data bus. The signals shown in Figure 18.2 are classified in seven categories. Categories 1 through 6—(1) address bus, (2) data bus, (3) control and status signals, (4) external requests, (5) response to external requests, and (6) power and clock—are identical to that of the 8085 processor.

*The terms 8086 and 8088 are used synonymously; the discussion is applicable to both processors unless specifically mentioned otherwise.

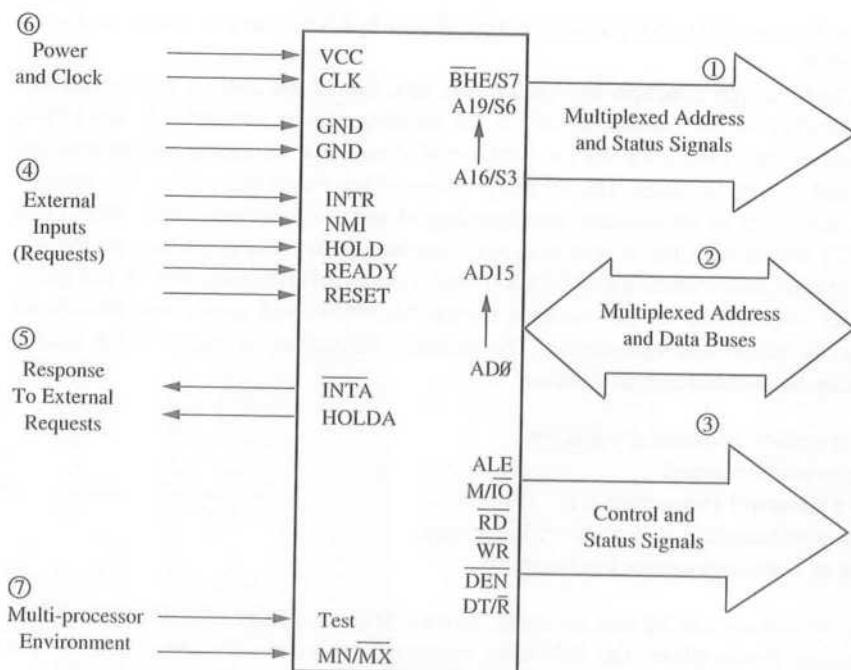


FIGURE 18.2
8086 Microprocessor—Logical Signals in Minimum Mode.

The seventh category, signals for multiprocessor environment, is new. It includes two signals: MN/MX (minimum or maximum mode) and Test. The data bus and status signals are multiplexed with the address bus. In the 8088 processor, eight data lines, AD7–AD0, and in the 8086 processor 16 data lines, AD15–AD0, are multiplexed. Signals that are different from the 8085 signals are explained as follows.

- **MN/MX—Minimum/Maximum Mode:** This signal represents two operation modes of the processor: minimum and maximum. When the signal is high (connected to +5 V), the processor operates in the minimum mode, and when it is low (grounded), the processor operates in the maximum mode. The minimum mode is used for the single processor environment, and the maximum mode is used for the multiprocessor environment such as having a coprocessor in a system. In the maximum mode, eight pins are assigned different functions compared to that of the minimum mode, as shown in Table 18.1, and a bus controller (such as the 8288) is necessary to generate control signals.

TABLE 18.1
8086/8088 Minimum and Maximum Mode Control Signals

Pin	Minimum Mode Function	Maximum Mode Function
24	INTA: Interrupt Acknowledge	QS1: Queue Status Signal
25	ALE: Address Latch Enable	QS0: Queue Status Signal
26	DEN: Data Enable	S0: Input signals to 8288 to generate control signals
27	DT/R: Data Transmit/Receive	S1
28	M/IO (8086): Memory or Input/Output IO/M (8088): Input/Output or Memory	S2
29	WR: Write	LOCK: Lock—to prevent another processor from gaining control
30	HLDA: Hold Acknowledge	RQ/GT1: Request/Grant—enable another processor to gain control
31	Hold: Hold	RQ/GT0

- **TEST**—This signal is used to synchronize operations of multiple processors in a system. When the WAIT instruction is being executed, the processor checks this signal. If it is high, the processor interrupts the execution of the program, and if it is low, it continues the execution.
- **DEN—Data Enable:** This is an active low output signal that is generally connected to a bidirectional buffer (such as 74LS245) to isolate the MPU from the system bus.
- **DT/R—Data Transmit/Receive:** This is also connected to a bidirectional buffer to enable data flow.
- **M/IO and IO/M—Memory and I/O:** This signal indicates whether the processor cycle is an I/O operation or a memory operation. The active levels indicated by the symbols are exactly opposite of each other in the 8086 (M/IO) and 8088 (IO/M) processors.
- **BHE—Bus High Enable:** This is an active low signal used only in the 8086 processor to enable the high-order byte of 16-bit data. In the 8088, it is used as a status line (SSO) to decode the current cycle.

GENERATING MPU* SIGNALS

In the 8088/86 processor, the data bus and the status signals are multiplexed, like the data bus in the 8085. These signals must be demultiplexed to form a complete functional MPU that can be used in a system. Figure 18.3 shows a complete schematic of the demultiplexed bus of the 8086 processor in the minimum mode. In the 8088 processor, the address lines A15–A8 are not multiplexed. Therefore, the 8088 processor needs only two latches.

*The term *MPU* indicates that all signals are demultiplexed and appropriate control signals are available.

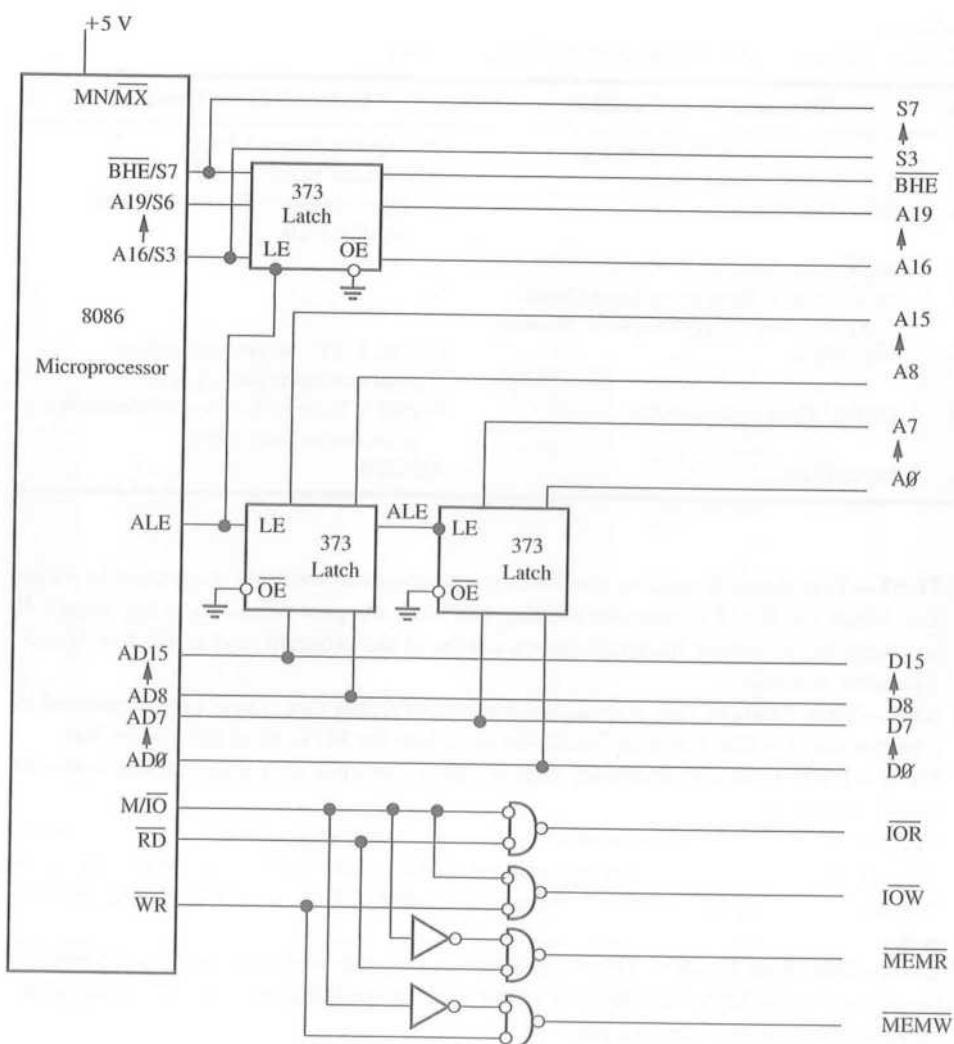


FIGURE 18.3

The 8086 MPU with Demultiplexed Address Bus and Control Signals in the Minimum Mode.

The processor asserts the ALE signal high at the beginning of each machine cycle when it places the address on the address bus. This ALE signal is used to enable three latches (373s) to separate the address lines from the multiplexed bus, as shown in Figure 18.3. The lower two latches in Figure 18.3 generate the address bus A15–A0, and the top latch separates status signals from the address signals and generates the address lines A19–A16 and the BHE signal.

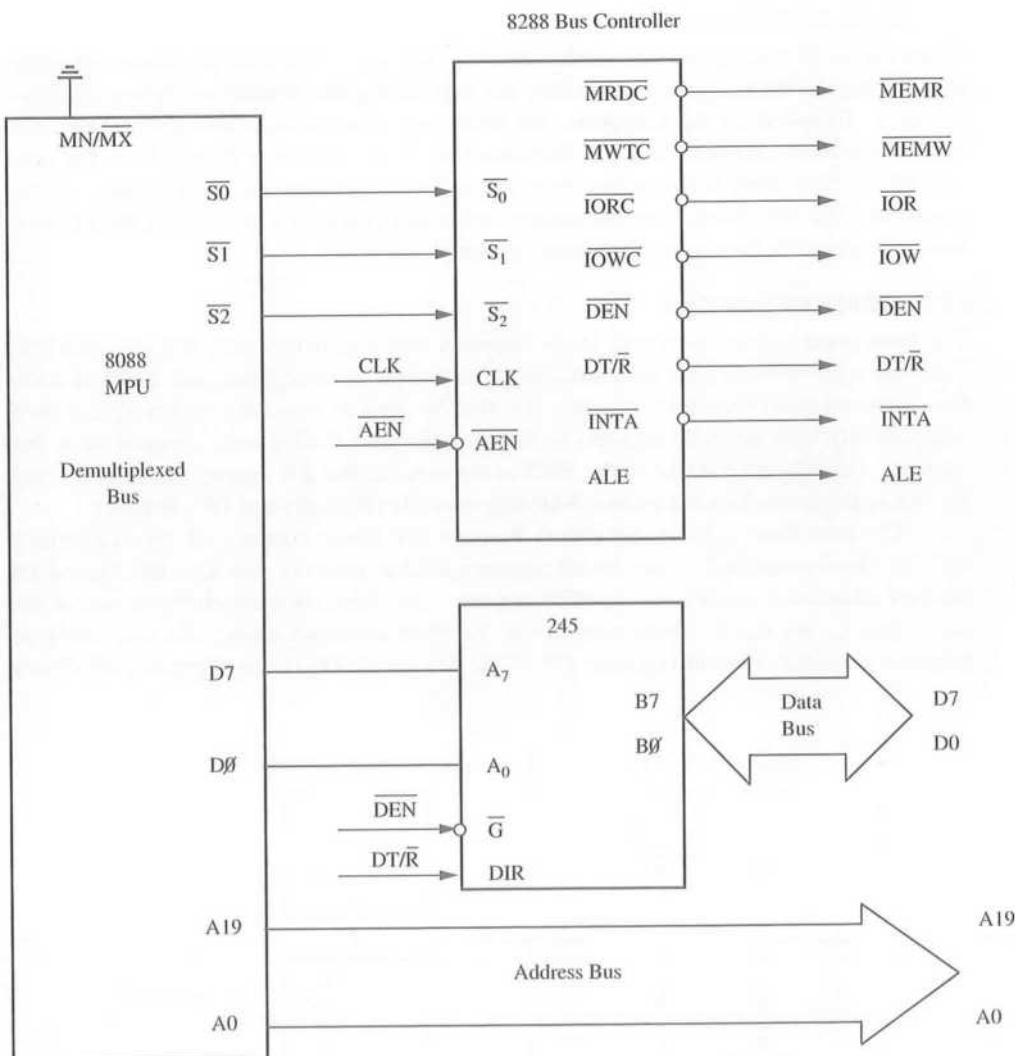


FIGURE 18.4
8088 MPU in the Max Mode and Control Signal Generation.

Figure 18.4 shows the 8088 MPU (with demultiplexed address bus A19–A0) in the maximum mode. In this mode, eight signals are assigned the function of status signals that are used as inputs to the 8288 bus controller to replace the signals that are lost. The 8288 generates the necessary control signals as shown in Figure 18.4. In addition, Figure 18.4 also shows how DT/R and DEN are used to buffer the data bus to avoid excessive loading of the bus.



INTERNAL ARCHITECTURE

When the 8085 microprocessor fetches and executes an instruction, the buses are occupied during the fetch operation, but they are idle during the internal execution of an instruction. To speed up the execution, the 8086 processor includes two processing units called Execution Unit (EU) and Bus Interface Unit (BIU), shown in Figure 18.5. The concept of dividing work between two units and processing it simultaneously speeds up the execution. The BIU fetches the instructions and places them in a queue, and the EU continuously executes them until the queue is empty.

PROGRAMMING MODEL

The 8086 processor includes four 16-bit general purpose registers: AX, BX, CX, and DX. They are equivalent to four accumulators even though some registers are assigned additional special functions. Each register can also be used as two 8-bit registers, and their compatibility with the 8085 registers is shown in the gray shaded area in Figure 18.6. For example, the HL register pair in the 8085 is the same as the BX register in the 8086, and the BX register can be used as two 8-bit registers BH (B-High) and BL (B-Low).

The next four registers, SP (Stack Pointer), BP (Base Pointer), SI (Source Index), and DI (Destination Index), are 16-bit registers used as memory pointers. BP, SI, and DI are new additions compared to the 8085 registers, and these registers eliminate one of the drawbacks of not having index registers in the 8085 microprocessor. The next group of registers is called segment registers: CS (Code Segment), DS (Data Segment), SS (Stack

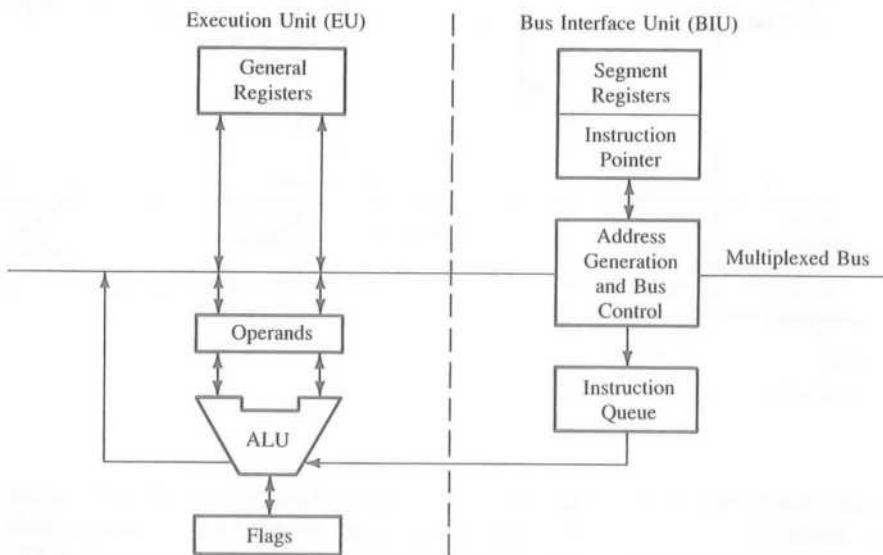


FIGURE 18.5

Execution and Bus Interface Units (EU and BIU) of the 8086

SOURCE: Intel Corporation, *iAPX 86, 88 User's Manual* (Santa Clara, Calif.: Author, 1981), p. 2-5.

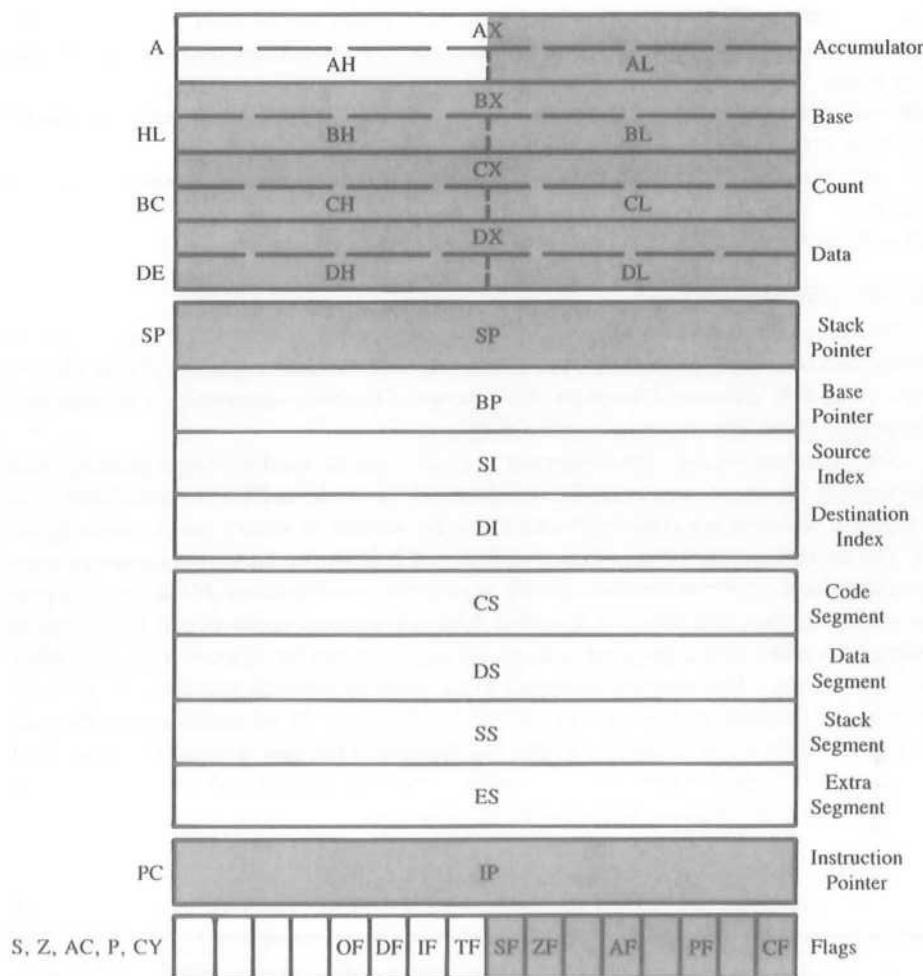


FIGURE 18.6

The 8086 Programming Registers (the Shaded Portions Show Areas Equivalent to the 8085)
 SOURCE: Adapted from Intel Corporation, *Microprocessors Vol. II* (Santa Clara, Calif.: Author, 1993).

Segment), and ES (Extra Segment). The segment registers are combined with memory pointers to generate a memory address as explained in the next section. The last 16-bit register shown in Figure 18.6 is called IP (Instruction Pointer), which is the same as PC (Program Counter) in the 8085.

The flag register includes nine flags, four are new additions. These flags are divided into two groups: six data flags and three control flags. In the data flag category, the OF (Overflow) is new; the remaining are the same as in the 8085 processor. All control flags (DF, IF, and TF) are new additions. The new flags are defined as follows.

- OF—Overflow:** This is used in signed numbers. When the result of a signed number operation is too large, causing the most significant bit to overflow into the sign bit, this flag is set.
- DF—Direction Flag:** This flag is used to control the direction (increment/decrement) of string operations.
- IF—Interrupt Flag:** This flag is used to enable or disable external maskable interrupt requests.
- TF—Trap Flag:** This flag is used for single-stepping instructions.

MEMORY SEGMENTATION

The 8086 processor has a 20-bit address bus; however, its Instruction Pointer is 16-bit wide and can hold a 16-bit address. The processor must be able to place a 20-bit address to access 1M-byte memory. Therefore, the concept of memory segmentation is employed in the design of this processor as explained below.

The processor includes four segment registers that are used to assign memory base addresses: CS for instruction code, DS for data, SS for stack, and ES for additional data. The segment registers are combined with memory pointers to form a 20-bit memory address. The default combination is shown in Table 18.2; however, the instructions can override these default combinations and specify appropriate combinations. If the size of a program (including data and stack) is less than 64K, all segment registers can be defined at the same base address. If a program is large, all segments can be separate from each other or they can overlap. The memory addresses in the 8086 systems are specified in three formats: physical, logical, and offset. The physical address is a 20-bit address, the offset address is the address of an instruction (data) in reference to the base address in the segment register, and the logical address is the combination of a segment and an offset address. These definitions are clarified in the following example.

**Example
18.1**

Assume that all segment registers are initialized at 1200H. The instruction register IP holds the address 0000H, which is the beginning of the instruction code (program). The stack is initialized at FFFFH, and the SI register is initialized at 8000H to indicate where data bytes are stored. Calculate the beginning physical addresses of the instruction code, data, and stack. Explain what the physical address, logic address, and offset address are.

TABLE 18.2
Segment Registers and Associated Offset Registers (Memory Pointers)

Segment Registers	Offset Registers
Code Segments (CS)	Instruction Pointer (IP)
Stack Segment (SS)	Stack Pointer (SP) and Base Pointer (BP)
Data Segment (DS)	Source Index (SI), Destination Index (DI), and BX Register
Extra Segment (ES)	Source Index (SI), Destination Index (DI), and BX Register

The processor generates the beginning address of the instruction code by combining the addresses in CS and IP. However, it is not an addition. The processor shifts the address in the code segment by four bits (one Hex digit) to the left and adds the address in IP. Similarly, it combines DS and SI, and SS and SP.

Solution

Physical Addresses (PA)

Code Address	Stack Address	Data Address
(CS) : 1 2 0 0	(SS): 1 2 0 0	(DS): 1 2 0 0
(IP) : + 0 0 0 0	+ F F F F	+ 8 0 0 0
(PA) : 1 2 0 0 0	2 1 F F F	1 A 0 0 0

The physical address is a 20-bit address that the processor places on the address bus. In the 8086, it ranges from 00000 to FFFFFH. The logical addresses are: Code—1200:0000; Stack—1200:FFFF; and Data—1200:8000. The offset addresses are: Code—0000H; Stack—FFFFH; and Data—8000H.

INSTRUCTION SET

The 8086 has a large instruction set, consisting of 135 basic instructions, which can operate on individual bits, bytes, 16-bit words and 32-bit double words, signed numbers, ASCII characters, and BCD numbers. The instruction set can be divided into six categories: (1) data copy (MOV), (2) arithmetic, (3) logic, (4) program transfer (such as jump and call), (5) string manipulation, and (6) machine or processor control (such as halt and interrupt). We are already familiar with five categories out of these six in the 8085 microprocessor; the only new category is string manipulation. The set includes instructions such as multiply, divide, jumps based on multiple flags, and string manipulation. If one is familiar with the 8085 assembly language and basic programming concepts, the transition to the 8086 assembly language is not difficult and can be self-directed.

In addition to the powerful instruction set, the chip design is oriented toward modular programming, very desirable for high-level languages. The memory-segmentation concept facilitates programming of independent modules that can communicate with each other as well as share common data.

COPROCESSING

In addition to the 8086, Intel has designed a series of special-function devices such as the 8089 (I/O Processor) and the 8087 (Numeric Processor). These processors are compatible with the 8086 in a master-slave relationship. They are designed with additional instructions and can be assigned dedicated functions to increase the overall execution speed of large systems.

MEMORY INTERFACING

The concepts of memory interfacing in the 8088 are identical with the 8085. It requires the address decoding of the high-order address bus. For example, to interface a 16K memory chip, we need 14 address lines for the chip, and the remaining six lines must be

decoded by using a decoder. However, memory interfacing in the 8086 requires a different technique. The 8086 has a 16-bit data bus, but memory has 8 data lines. This raises a question: How do we transfer a 16-bit word over 8 data lines? The answer lies in connecting two memory chips in parallel, called high or odd and low or even bank, and accessing them by using the $\overline{\text{BHE}}$ (Bus High Enable) signal.

Figure 18.7 shows an illustration of interfacing two 32K-byte memory chips (total of 64K bytes) with the 8086. We need 16 address lines (A15–A0) from the processor to address 64K bytes of memory ($2^{16} = 64\text{K}$). The 32K memory chips have 15 address lines, A14–A0. Normally, we connect the processor address lines A14–A0 to the respective address lines of the memory chip. However, Figure 18.7 shows that the processor address lines A15–A1 are connected to lines A14–A0 of the memory chips. A0 of the processor is used to enable the 3-to-8 decoder of the low bank, and the $\overline{\text{BHE}}$ signal is used to enable the high bank. When A0 is at logic 0, the low bank is enabled, and the processor accesses the low-order data byte (D7–D0), and when $\overline{\text{BHE}}$ is low, the high bank is enabled, and the processor accesses the high-order data byte (D15–D8). When both are low, the processor reads (or writes) the entire data bus (D15–D0). The memory addresses in Figure 18.7 range from 40000H to 4FFFFH.

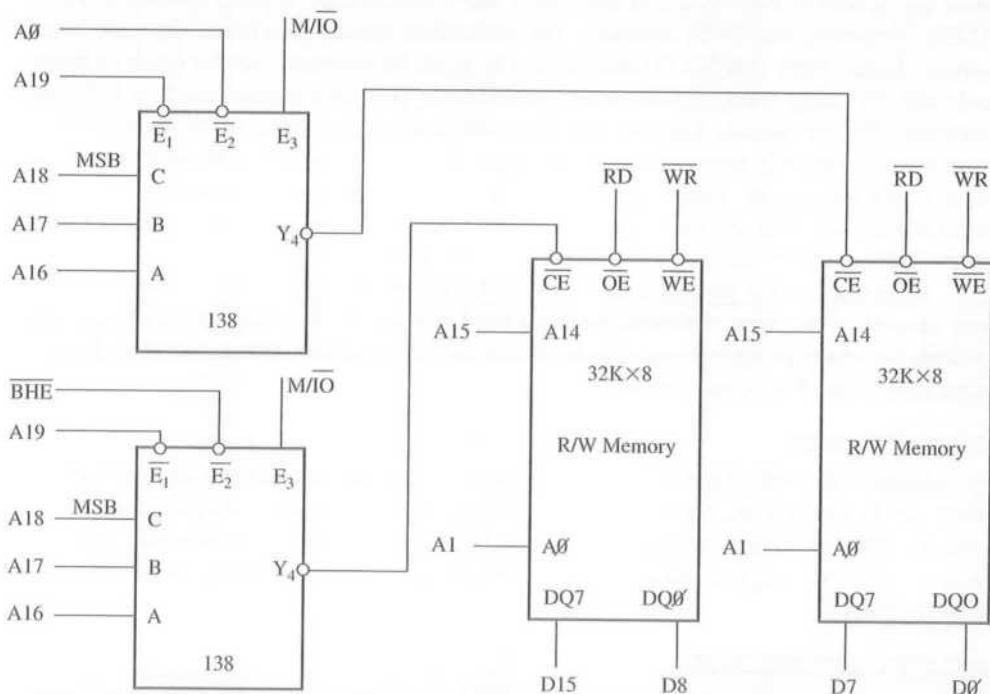


FIGURE 18.7

Interfacing Byte-wide Memory Chips with the 16-bit 8086 Processor.

I/O INTERFACING

The 8085 microprocessor has an 8-bit I/O space, separate from memory; thus the I/O addresses can range from 00 to FFH. The IN and OUT instructions are designed to use this I/O space. On the other hand, the 8088/86 processors can have 8-bit or 16-bit I/O space; thus the I/O addresses can range from 00 to FFH or from 0000 to FFFFH. The IN and OUT instructions are designed to use either 8-bit I/O space or 16-bit I/O space. Figure 18.8 shows the interfacing of the 8255 (Programmable Peripheral Interface) and the 8254 (Timer) in a personal computer (PC); it uses the 16-bit I/O space. However, the address lines A15–A10 and A4–A2 are not decoded, leaving them as don't care lines. Thus the port addresses of the timer range from 0040H to 0043H and of the 8255 from 0060H to 0063H (assuming all don't care lines at logic 0). The absolute decoding practice is generally not used for I/O interfacing in PCs to reduce hardware cost.

18.3.2 Intel 80186 and 80286

The Intel 80186 and 80286 are 16-bit microprocessors, extended versions of the 8086. One of the critical barriers of Intel's earlier microprocessors was the 40-pin package. Once that barrier was broken, it became easier to address large memory. These microprocessors are housed in 68-pin packages and use the concepts of prefetched pipeline structure, parallel processing, and memory management.

The 80186 is an improved version of the 8086, available in two speeds: an 8 MHz and a 10 MHz version. It is an integrated device designed to reduce the chip count rather than to increase the memory-addressing capacity. It has multiplexed address and data buses, and the additional lines of the bigger package are used to include devices such as a clock generator, interrupt controller, timers, DMA controller, and a chip select unit.

The 80286 is also a 16-bit microprocessor, an improved version of the 8086 but with a different architectural philosophy. It eliminates the multiplexing of the buses; it has

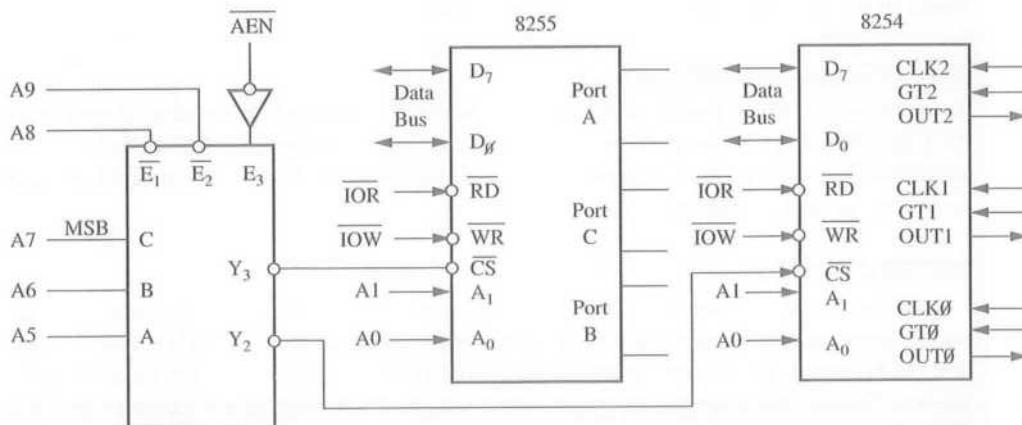


FIGURE 18.8

Interfacing I/O Devices with the 8088 Processor—Schematic from PC.

a linear address bus with 24 address lines that can address 16M bytes of memory directly. It can also support a memory management unit, and through the memory management unit it can address 1G bytes of memory, also known as *virtual memory*. The processor includes various built-in mechanisms that can protect system software from the user programs, protect users' programs, and restrict access to some regions of memory. The 80286 is designed for multiuser systems; its architectural philosophy is closer to Intel's 80386 32-bit microprocessor, which is described in Section 18.4.

18.3.3 Motorola MC68000

The MC68000 is a family of microprocessors that includes 8-, 16-, and 32-bit versions with varying memory addressing capability. The 68000 (also 68010) is a 16-bit microprocessor with a 32-bit internal architecture housed in a 64-pin package. It is capable of addressing 16 megabytes of memory, and the clock frequency ranges from 4 MHz to 10 MHz for different versions of the chip. The internal architecture and registers are almost common to all versions, thus making them software compatible. The 60008 is an 8-bit microprocessor with two versions: one is capable of addressing 1 megabyte of memory and the other 4 megabytes of memory. The 68020, 68030, and 68040 have a 32-bit data bus and a 32-bit address bus with the memory addressing capability of 4 gigabytes.

Figure 18.9 shows the internal architecture of the device. It includes seventeen 32-bit, general-purpose registers, a 32-bit program counter, and a 16-bit status register. The general-purpose registers are divided into three groups: eight data registers, seven address registers, and two stack pointers. The contents of the data registers can be accessed as bytes, 16-bit words, or 32-bit words, and the contents of the address registers can be accessed as 16-bit or 32-bit addresses. The 68000 can operate in two different modes: the user mode and the supervisor mode. The supervisor mode is designed primarily for operating systems; in this mode, some privileged system control instructions can be used. Some of its other features can be described as follows.

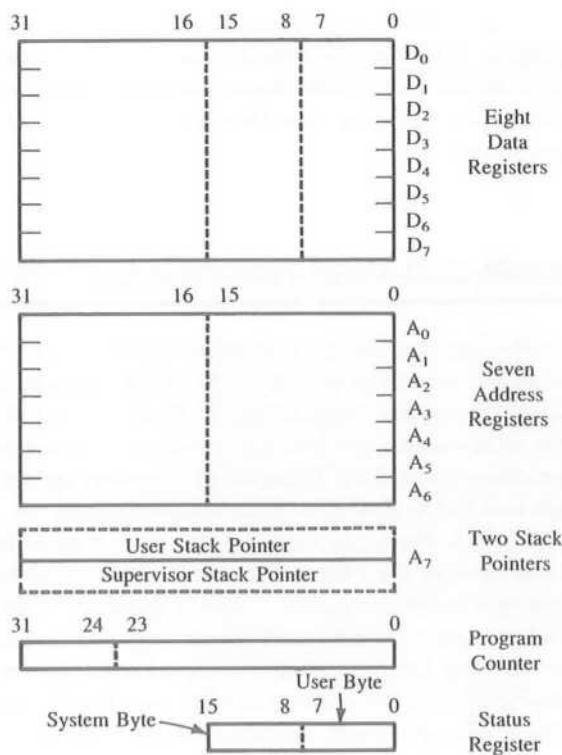
NONSEGMENTED MEMORY

To increase the memory-addressing capacity, Motorola increased the number of pins in its package. The chip is designed with 23 separate lines to address eight megawords (16 megabytes). Similarly, its program counter is 32 bits long; only the low-order 24 bits are necessary to address the entire memory map.

INSTRUCTION SET

The 68000 has one of the most powerful yet simple instruction sets. It includes 56 basic instructions and can operate on five different types of data: bit, byte, BCD, 16-bit word, and 32-bit word. It has only memory-mapped I/O but includes 14 memory-addressing modes. To cite one example of its powerful set, its MOV instruction can transfer data from any source to any destination. It includes instructions such as Multiply and Divide and special instructions to deal with numbers longer than 32 bits. Its orientation toward high-level languages comes primarily from its instruction set.

FIGURE 18.9
Programming Registers of the
68000



ASYNCHRONOUS AND SYNCHRONOUS CONTROL LINES

The 68000 has a special way of handling slow and fast peripherals. It has two sets of control signals, called asynchronous and synchronous signals. Communication with asynchronous peripherals is handled through the control lines called Upper Data Strobe (UDS), Lower Data Strobe (LDS), and Data Acknowledge (DACK). The DACK signal is similar to a handshake line; the bus cycle is not terminated until the signal, DACK, is received. The 68000 family offers some synchronous peripherals, and communication with these peripherals is handled through the control signals called Valid Peripherals Address (VPA), Valid Memory Address (VMA), and Enable (E).

18.3.4 Review of 16-Bit Microprocessors

The 8086 and the 68000 are designed with two distinct philosophies. The 8086 uses only 40 pins and multiplexes most of the functions. It has employed several new architectural concepts such as memory segmentation, parallel processing, queueing, and coprocessing. On the other hand, the 68000 has adopted a 64-pin package and simplified its architecture. However, both are oriented toward high-level languages. The 80186 is an integrated package, whereas the 80286 is essentially a faster version of the 8086 with the memory addressing capacity of 16M bytes without the multiplexed bus.

The 16-bit microprocessors are too powerful to perform the functions of general-purpose 8-bit microprocessors; therefore, they are less likely to replace 8-bit processors. In the early 1980s many microcomputers were designed based on 16-bit microprocessors. However, now they are being replaced by more powerful 32-bit and 64-bit processors.

18.4

HIGH-END-HIGH-PERFORMANCE PROCESSORS

At the high end of the microprocessor range, we have 32- and 64-bit microprocessors available; examples include such microprocessors as the Intel 80386 and 80486, Zilog Z80000, National Semiconductor NS32032, and Motorola MC68020. We are interested not in discussing the details of these microprocessors, but in exploring trends in microprocessor technology. These microprocessors are not merely more of the same except bigger and faster; they offer some unique features not available in the earlier 16-bit microprocessors. The applications and the environments in which they operate are far different from those of the 8-bit microprocessor and the earlier 16-bit microprocessors. It appears that two trends are evolving: one is multiuser, multitasking, time-sharing environments, and the other is distributed processing, interconnected with networks. As soon as we move away from the single-user system, the demands on these microprocessors change drastically; the environment is more like that of minicomputers or mainframe computers. These microprocessors include many features that are normally available in mainframe computers.

In a single-user system, the user has unlimited access to all aspects of the system. The user need not be concerned with sharing the time or the resources of the system, but can schedule various tasks according to his or her convenience. The user has access to the operating system, can tamper with the system to include some personal conveniences, or in the process can lock up the system. However, the multiuser system cannot afford to provide the luxuries of unlimited access to all users. Some of the requirements of the multiuser system are as follows:

1. Higher speed of execution
2. Ability to handle different types of tasks efficiently
3. Large memory space that can be shared by multiple users
4. Appropriate memory allocations and the management of memory access
5. Data security and data access
6. Limited and selected access to part of the system
7. Resource (printer, hard disk, etc.) sharing and management

Some of these requirements must be managed by a multiuser operating system, and some should be facilitated by the architectural design of the microprocessor. The 32-bit microprocessors are designed to work in this type of environment. Some of the important features of the Intel 80386/486 are described in this section as a representative sample of 32-bit microprocessor technology.

18.4.1 The Intel 80386/80486

The 80386 is a 32-bit microprocessor with a nonmultiplexed 32-bit address bus and is housed in a 132-pin grid array package. It has versions that can operate from 20 MHz to 33 MHz. It is capable of addressing 4G bytes of physical memory and, through its memory management unit, can address $64 (2^{46})$ terabytes of virtual memory. The processor can operate in two modes: Real Mode and Protected Mode. In Real Mode, physical address space is 1M bytes (20 address lines), which is extended to 4G bytes in Protected Mode (32 address lines). The primary difference between these modes is the availability of the memory space and the addressing scheme. The 80386 has 32-bit registers and is upward software-compatible with the 8086. The execution of instructions is highly pipelined, and the processor is designed to operate in a multiuser and multitasking environment. It has the protection mechanism necessary for this type of environment.

FUNCTIONAL SIGNAL GROUPS

Figure 18.10 shows the functional groups of the 80386 signals. The functional signal groups are in many ways similar to those of the 8085, except with larger buses: 32-bit address bus and 32-bit data bus. The 80386's arbitration and interrupt signals are similar to the external request signals, and its bus cycle definition and control signals are similar to the control signals and status signals in the 8085. However, the 80386 has a new group of signals that is used for interfacing a coprocessor. In addition, each group has some signals for specialized functions. For example, in the address bus, the signals BE0#–BE3#, called byte enable signals, are used to identify which groups of data lines are active in a transfer of 32-bit data. In the bus control signals, BS16# allows the processor to directly connect to 32-bit and 16-bit data buses.

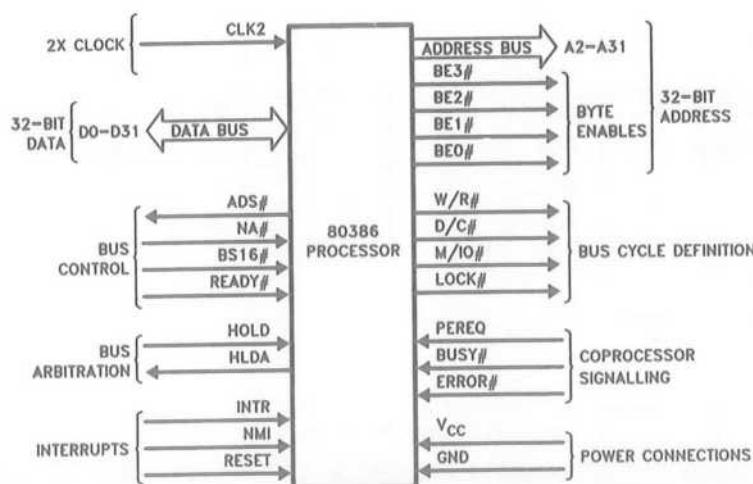


FIGURE 18.10

Functional Groups of the 80386 Signals

SOURCE: Intel Corporation, *Microprocessors Vol. II* (Santa Clara, Calif.: Author, 1993).

PROGRAMMING MODEL

Figure 18.11 shows a programming model of the 80386. It has eight general-purpose registers to hold 32-bit data or addresses; they can be used as 8-, 16-, or 32-bit registers. The 32-bit registers are named with the prefix E (EAX, etc.), and the least 16 bits (0–15) of these registers can be accessed with names such as AX or SI. Similarly, the lower eight bits (0–7) can be accessed with names such as AL and BL and the higher eight bits (8–15) with names such as AH and BH. Figure 18.11 shows six 16-bit registers called segment selector registers; these are used for determining memory addresses. The instruction pointer (EIP), known as a program counter in 8-bit microprocessors, is a 32-bit register to handle 32-bit memory addresses, and the lower 16-bit segment (IP) is used for 16-bit memory addresses.

The flag register is a 32-bit register; however, only 14 bits are being used at present for 13 different tasks; these flags are upward-compatible with those of the 8086 and 80286. The comparison of the available flags in 16-bit and 32-bit microprocessors may provide some clues related to capabilities of these processors. The 8086 has 9 flags, the 80286 has 11 flags, and the 80386 has 13 flags. All of these flag registers include six flags related to data conditions (Sign, Zero, Carry, Auxiliary Carry, Overflow, and Parity) and three flags related to machine operations (Interrupt, Single-Step, and Strings). The 80286 has two additional flags: I/O Privilege and Nested Task. The I/O Privilege uses two bits in Protected Mode to determine which I/O instructions can be used, and the Nested Task is used to show a link between two tasks. The 80386 added two more flags: VM (Virtual Mode) to run the 8086-compatible programs in this mode and RF (Resume Flag) to work with breakpoints.

GENERAL DATA AND ADDRESS REGISTERS								SEGMENT SELECTOR REGISTERS							
31	16	15	8	7	0			15	0						
	AH	A	X	AL		EAX				CS	CODE				
	BH	B	X	BL		EBX				SS	STACK				
	CH	C	X	CL		ECX				DS	DATA				
	DH	D	X	DL		EDX				ES					
				SI		ESI				FS					
				DI		EDI				GS					
				BP		EBP									
				SP		ESP									
31	16	15		0											
								EIP INSTRUCTION POINTER							
								EFLAGS FLAG REGISTER							

FIGURE 18.11

The 80386 Programming Model

SOURCE: Intel Corporation, *Microprocessors Vol. II* (Santa Clara, Calif.: Author, 1993).

The processor also includes Control registers, System Address registers, and Debug and Test registers for system and debugging operations; these registers are shown in Figure 18.11.

INSTRUCTION SET AND ADDRESSING MODES

The instruction set is divided into nine categories of operations and has 11 addressing modes. In addition to commonly available instructions in an 8-bit microprocessor, the set includes operations, such as bit manipulation, string operations, high-level language support, and operating system support. An instruction may have 0 to 3 operands, and the operands can be 8, 16, or 32 bits long. The 80386 handles various types of data, such as a single bit, string of bits, signed and unsigned 8-, 16-, 32-, and 64-bit data, ASCII characters, and BCD numbers.

The high-level language support group includes instructions such as ENTER and LEAVE. The ENTER instruction is used to enter from a high-level language; it assigns memory locations on the stack for the routine being entered and manages the stack. On the other hand, the LEAVE generates a return procedure for a high-level language. The operating system support group includes several instructions such as APRL (Adjust Requested Privilege Level) and VERR/W (Verify Segment for Reading or Writing). The APRL is designed to prevent the operating system from gaining access to routines with a higher priority level, and the instructions VERR/W verify whether the specified memory address can be reached from the current privilege level.

MEMORY MANAGEMENT AND PROTECTION

The memory space available to the user is dependent on the mode of operation. In Real Mode, the maximum memory size is 1M byte; the processor uses 20 address lines. In Protected Mode, the processor has 32 lines available for addressing; thus, the physical memory size is extended to 4G bytes. This physical memory is translated into 64-terabyte virtual address space (logical addresses) by using the Memory Management Unit (MMU). In virtual memory systems, disk storage is used as memory by swapping its information between R/W memory and the disk; thus, the processor's R/W memory becomes a temporary holding area. In the 80386, the memory space can be divided into segments of various lengths or into pages of 4K bytes. The segmentation is suitable for various-sized logical addresses of programs, and the paging simplifies the swapping of information between the physical memory and the disk. In addition to managing the memory space, the MMU provides a four-level protection mechanism suitable for a multiuser system: the operating system can reside at the highest privilege level 0 and application programs at the least privilege level 3. The processor isolates tasks and prevents low-level software from having access to a higher level.

THE INTEL 80486

The 80486 is an upgraded faster version of the 80386. The DX type version is a 32-bit processor housed in a 168-pin grid array package and can operate with the clock frequencies from 25 MHz to 66 MHz. The design is based on 1.2 million transistors compared to 300 thousand transistors of the 386 processor. The important additional features

of the 486 processor in comparison with the 386 processor are as follows. The 486 processor includes:

1. Built-in math coprocessor. In 386 systems, a math coprocessor is an external device. Therefore, the math instructions in 486 systems are executed three times faster than in 386 systems.
2. 8K-byte of code and data cache memory on the chip.
3. Highly pipelined execution unit. Therefore, the execution time for many instructions is one clock period.

In summary, the 486 is a high-speed, high-performance 32-bit microprocessor. It executes many of its instructions in one clock cycle by using highly pipelined execution units. It is designed to facilitate the execution of high-level languages and suited for multiprocessing and multiuser systems. In the early 1990s, 486 was generally used in high-end microcomputers and network environments.

18.4.2 The Intel Pentium™ Processor

The Pentium processor has a 32-bit address bus and a 64-bit data bus and its initial version is designed to operate from 60 MHz to 233 MHz. It is upward software-compatible with the previous line of the Intel microprocessors from the 8088/86 to the 486, and it is also designed for easy upgradability. The chip has more than 3.1 million transistors and is housed in a 273-pin grid array package. It includes many advanced features normally available in mainframe computers. The processor is ideally suited for high-end desktop PCs, workstations, and network file servers where high-speed computation (such as 3D graphics) is needed.

ADVANCED DESIGN FEATURES

The Pentium processor includes enhancement of some of the features available on the 486 processor. For example, it supports either the traditional memory page size 4K byte or the larger size of 4M byte. It has a 64-bit data bus, which increases the processing speed. Similarly, the architecture design makes the processor well suited to multiprocessing applications; it maintains data integrity in multiple processors. The processor also includes many advanced features; some of them are described as follows.

1. **Superscaler architecture.** RISC processors (described later) are generally designed with the superscaler architecture. The term *superscaler* refers to a microprocessor architecture that includes more than one execution unit. The Pentium has two execution units with dual-pipelined architecture; thus, it is able to execute two instructions simultaneously per clock cycle and achieve a high level of performance.
2. **On-chip cache memory for code and data.** The Pentium processor has two 8K byte of cache memory on the chip; one is used for code and the other is used for data. The cache memory is used for temporary storage of commonly used code and data copied from the system's (or the main) memory. For example, if the processor is executing a

loop, the codes within the loop are copied into the cache. This eliminates the need for the processor to go off the chip and access the main memory during the loop execution, thus improving the performance of the processor.

3. **Branch prediction.** This is a mainframe technique implemented in this processor to improve the performance. In this technique, the most likely set of instructions to be executed is predetermined, and the pipelines are kept full accordingly. For example, in a loop, a prediction is based on the assumption that the instructions in the loop are likely to be executed in the next interaction.
4. **High-performance floating-point unit.** This processor has an on-chip floating-point unit that incorporates highly sophisticated seven-stage pipelining and hardwired codes. This processor is capable of executing floating-point instructions five to ten times faster than is the 486 processor.
5. **Performance monitoring.** The processor design enables the user to monitor the performance of the processor and to optimize the performance by identifying potential bottlenecks in code execution. The user can observe and record time for internal events that affect the performance of data read and write, interrupts, cache hits and misses, and bus utilization. And based on these observations, the user can fine-tune programs.

18.4.3 The Intel Pentium Pro-Processor

The Pentium Pro-Processor has more than 4.2 million transistors and is housed in a 387-pin grid array package. It also has a 32-bit address bus and a 64-bit data bus, similar to the Pentium processor; however, there is provision for a 36-bit address bus, enabling the processor to access 64G bytes of physical memory in future. It has many versions based on its operating frequency.

One of the major differences of this processor compared to the Pentium processor is in the area of cache memory. The Pentium processor has two 8K memory cache on the chip known as level 1. The Pentium Pro-Processor has an additional 256K (or 512K) cache memory on the chip known as level 2 cache. This additional cache memory increases processing speed.

18.4.4 RISC Processors

The term *RISC* represents *reduced instruction set computing*. The Intel processors discussed in this text are considered CISC (complex instruction set computing) processors. As the processor technology began to evolve beginning with 4- and 8-bit processors, the trend was to add more and more instruction capability to the processor. As a consequence, the control unit and the instruction decoder became very complex. In some processors, the control unit occupies 50 to 60 percent of the chip area. This results in the reduction in space available to registers. Similarly, the instructions that access memory tend to slow down the code execution. Furthermore, many studies of application programs revealed that even if these processors include a large number of instructions, only a small number of them is used frequently. This fact gave rise to the RISC (reduced instruction set computing) design philosophy that focused on a small set of frequently used instructions, thus

simplifying the hardware design and improving the processor performance. RISC processors do not have a set of prescribed design rules, but they are governed by a set of guidelines evolved over many years. Generally, RISC processors include the following features:

1. The number of instructions is minimized (less than 100).
2. The number of addressing modes is relatively few (less than 3).
3. The number of memory-reference instructions is minimized. Memory is accessed only by Load and Store instructions.
4. The processing is register intensive, meaning it includes many more registers and most of the computing is performed using registers rather than memory. The number of registers ranges from 32 registers to more than 100 registers.
5. The instruction format is simplified and each instruction is executed in one cycle.
6. Design considerations include support for high-level languages through appropriate selection of instructions and optimization for compilers.

At present, many processors designed with RISC philosophy are available. A list includes such processors as Alpha from Digital Equipment Corp., R4400SC from MIPS Technologies, PA7100 from Hewlett-Packard, PowerPC from Apple, IBM, and Motorola, and Super Sparc from Sun Microsystems. They are either 32- or 64-bit processors. The number of general-purpose registers ranges from 32 to 136 registers, and the clock frequency ranges from 50 MHz to 200 MHz; the Alpha processor (Digital Equipment) runs on 200 MHz frequency. They include multiple pipeline stages and support multiprocessing. Some of these processors execute more than one instruction per cycle. These processors are generally used in workstations and file servers. However, some of these processors (such as the PowerPC) are also gearing toward the high end of the personal computer (PC) market. To take advantage of the power and speed of these processors, an operating system that can handle 32- and 64-bit processors in multiuser and multiprocessing environments is needed. At present, operating systems such as UNIX (many variations), OS/2 (IBM), Windows 95 (Microsoft), Mac OS (Apple), and Solaris (Sun Microsystems) are also evolving to meet the demands of the changing microprocessor technology.

18.4.5 Trends in High-Performance Processors

The recent 32-bit and 64-bit processors address large memory space in gigabytes, execute instructions with high speed, and perform floating-point arithmetic operations with high precision. They are high-performance processors oriented toward high-level languages in multiuser and multiprocessing environments.

These processors are classified as CISC or RISC processors. The CISC processors are designed with a large set of instructions and many addressing modes. On the other hand, the RISC processors include a small set of instructions with relatively few addressing modes. These instructions are selected based on how frequently they are used in application programs and how they can optimize compiler designs in high-level languages. Most instructions are executed using registers and the need for memory access is minimized. The RISC processors include multiple execution units and pipeline stages that enable the execution of more than two or more instructions per cycle; this is known as su-

perscaler operation. This operation sets these processors apart from the previous generation processors. At present, the trend is toward the RISC approach because of its relative simplicity and high performance.

SINGLE-CHIP MICROCONTROLLERS

18.5

Single-chip microcomputers, also known as microcontrollers, are used primarily to perform dedicated functions. They are used as independent controllers in machines or as slaves in distributed processing, as described in Chapter 15 (Section 15.3). Generally, they include all the essential elements of a computer on a single chip: MPU, R/W memory, ROM, and I/O lines. Typical examples of the single-chip microcomputers are the Intel MCS-48, 51, and 96 families, the Motorola MC68HC11 family, and the Zilog Z8.

Most of these microcontrollers have an 8-bit word size (except the MCS-96, with a 16-bit word size), at least 64 bytes of R/W memory, and 1K byte of ROM. The range of I/O lines varies considerably, from 16 to 40 lines. However, most of these devices cannot be easily programmed in college laboratories unless they include EPROM on the chip. A variety of single-chip microcontrollers is available in the market to meet diversified industry needs. To illustrate the trend, two different single-chip microcontrollers are described here.

18.5.1 Intel MCS®-51 Single-Chip Family

The Intel MCS®-51 is a widely used 8-bit single-chip microcontroller family. At the high end of the single-chip device spectrum in terms of its capability and versatility, it is designed for use in sophisticated real-time instrumentation and industrial control. It can operate with a 12 MHz clock and has a very powerful instruction set.

Figure 18.12 shows a block diagram of the chip. It includes the following features:

- 4K bytes of ROM or EPROM
- 128 bytes of data memory plus 21 special-function registers (SFR—not included in figure)
- four programmable I/O ports (32 I/O lines)
- two 16-bit timer/event counters
- a serial I/O port with a UART
- five interrupt lines: two for external signals and three for internal operations

The 8051 is known as a “bit and byte processor.” The instruction set includes binary and BCD arithmetic operations, bit set/reset functions, and all logical functions. However, its real power comes from its ability to handle Boolean functions. On any addressable bit, the processor can perform functions such as Set, Clear, Complement, Jump If Set or Not Set, and Jump If Set Then Clear. It can also perform logical functions with two bits and place the result in the Carry flag.

The 8051 can use its 32 I/O lines as 32 individual bits or as four 8-bit parallel ports. It can service five interrupts: two external, two from the counters, and one from

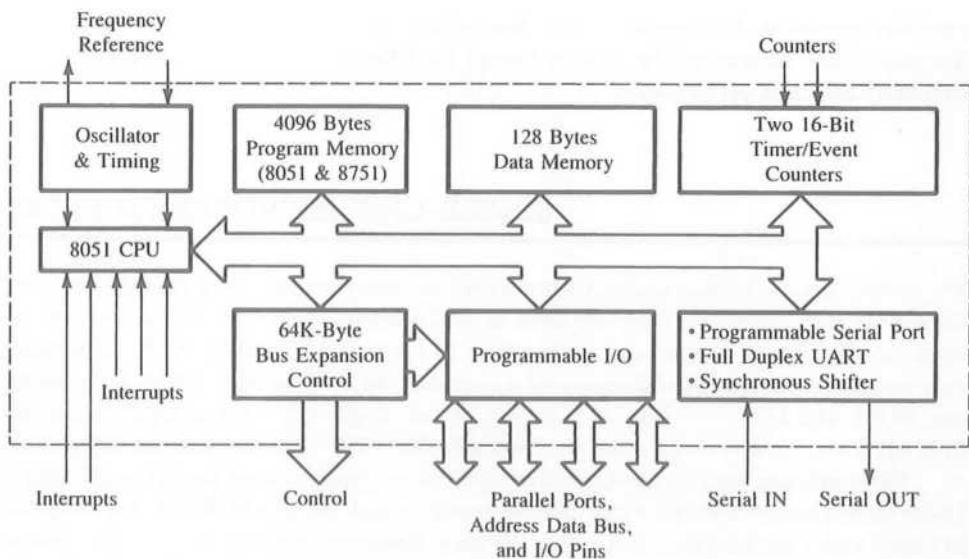


FIGURE 18.12
Block Diagram: The 8051

SOURCE: Adapted from Bob Kochler, "Microcontroller Doubles as Boolean Processors," *Electronic Designs*, vol. 28, no. 11; copyright Hayden Publishing Company, Inc., 1980.

the serial I/O port. The chip includes two 16-bit counters that can operate in three different modes, and a serial I/O port that can operate in the full duplex mode.

18.5.2 Motorola MC68HC11 Microcontroller Family

The MC68HC11 is also a widely used single-chip microcontroller family in industrial applications. It is an advanced 8-bit microcontroller with highly sophisticated on-chip peripherals such as timers, serial I/O, and an eight-channel A/D converter and can operate in four different modes. It has 40 I/O lines, and they serve multiple functions depending on the operating mode. It includes 256 bytes of internal R/W memory that can be expanded to 64K system memory in the expanded mode. It is housed in a 48-pin package and can operate up to 2 MHz clock frequency. Its instruction set is upward-compatible with the older processors 6800 and 6801.

Figure 18.13 shows a block diagram of the MC68HC11A8 version; this version is used as a primary reference. Other versions differ primarily in terms of memory size, availability of on-chip A/D converter, and number of pins in a package. Some of the primary features of this microcontroller are as follows:

- Four operating modes
- 8K bytes of ROM and 512 bytes of EEPROM
- 256 bytes of RAM (R/W memory)
- 40 I/O pins with multiple functions

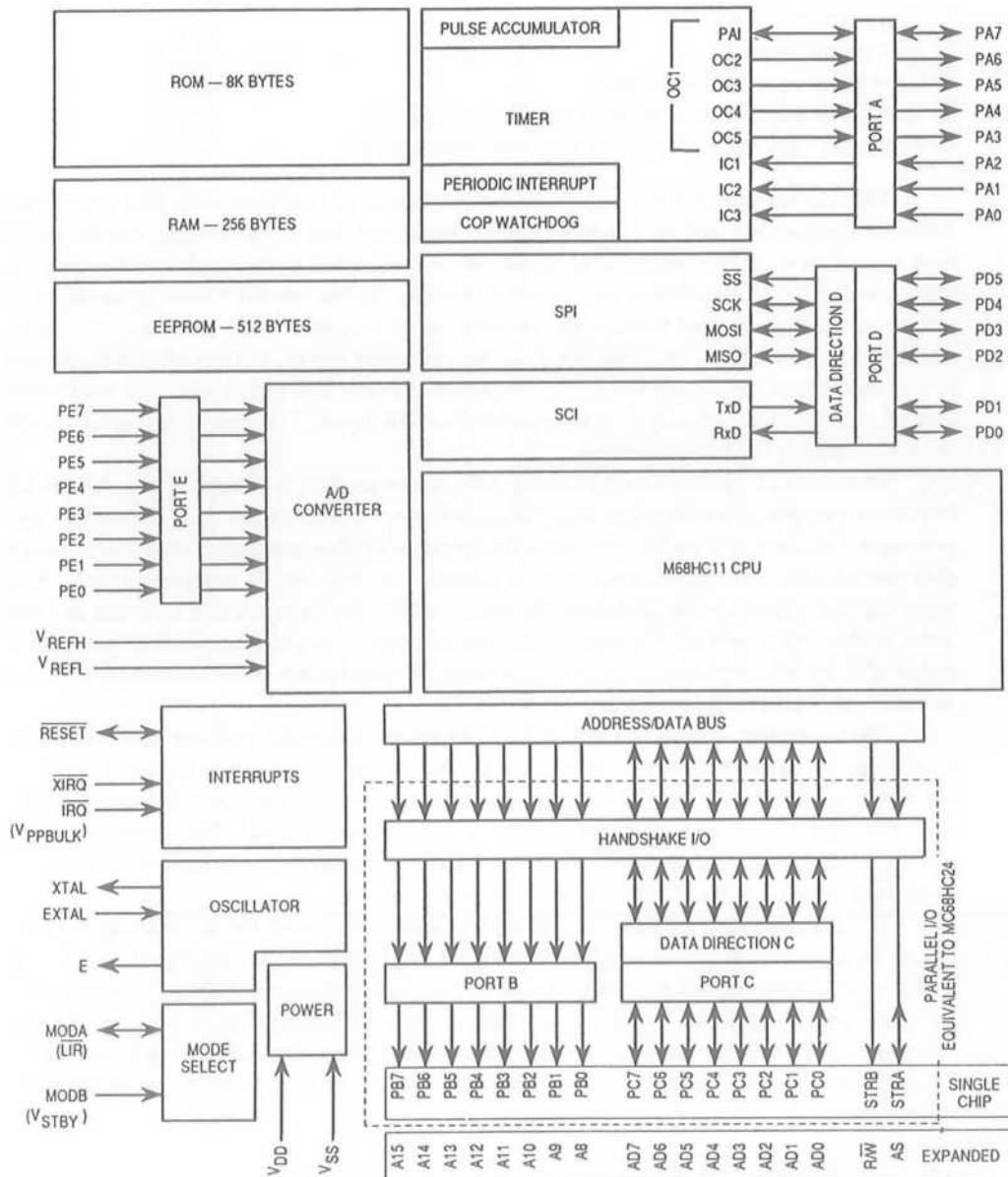


FIGURE 18.13

Block Diagram MC68HC11

SOURCE: Courtesy of Motorola, Inc.

- Eight-channel, 8-bit A/D converter
- 16-bit timer system
- 8-bit pulse accumulator circuit
- Serial communication and serial peripheral interface
- Computer operating properly (COP) watching system

This microcontroller has two fundamental modes of operation and their two variations: (1) single-chip and its variation special bootstrap, and (2) expanded and its variation special test. In the single-chip mode, it is a microcontroller with the features as shown in Figure 18.13 without any external memory. In the special bootstrap mode, programs can be downloaded through the on-chip serial communication interface into internal RAM (R/W memory) for execution. In the expanded mode, 18 lines (I/O ports B and C and two strobe lines) are used as multiplexed address and data buses and read/write control signals; thus memory can be expanded to 64K bytes. The special test mode is intended primarily for factory testing.

The 68HC11 includes an 8-channel A/D converter with 8-bit resolution. This is an important peripheral on the chip that makes the controller ideal for data acquisition applications. It also has a 16-bit programmable timer with four prescalers (that can set four different frequencies), three input capture signals, and five output compare signals. The input capture signals can provide information about the time reference of an event and the pulse width and/or period of a signal. The output compare signals are used to generate a pulse of a specific duration or a specified delay. Similarly, the pulse accumulator is essentially an 8-bit event counter or a timer.

The controller is designed to handle 17 hardware interrupts and one software interrupt. It is also capable of receiving and transmitting serial data through its serial communication interface (SCI) lines. Similarly, the serial peripheral interface (SPI) enables several SPI-type controllers and peripherals to be interconnected. The controller also includes self-monitoring circuitry to protect against system errors; it is known as the computer operating properly (COP) watchdog system.

The programming model is in many ways similar to that of the 6800 discussed earlier. It includes two 8-bit accumulators, A and B, and these can be combined as a 16-bit accumulator. It has an 8-bit condition code register (flag register) and four 16-bit registers: two index registers, X and Y, one stack pointer, and one program counter. On the other hand, the 6800 has only one index register and it cannot combine two accumulators. The instruction set includes 91 new opcodes in addition to the 72 instructions of the 6800 microprocessor.

18.5.3 Review of Single-Chip Microcontrollers

Examination of the examples discussed above shows that the single-chip microcontroller plays a vital role in control applications and is an important segment of microprocessor technology. These devices are designed for special-purpose applications, and the circuitry on the chip varies according to its objectives. Applications range from bit set/reset functions to processing high-speed analog signals.

A

Number Systems

Computers communicate and operate in binary digits 0 and 1; on the other hand, human beings generally use the decimal system with ten digits, from 0 to 9. Other number systems are also used, such as octal with eight digits, from 0 to 7, and hexadecimal (Hex) with digits from 0 to 15. In the hexadecimal system, digits 10 through 15 are designated as A through F, respectively, to avoid confusion with the decimal numbers 10 to 15.

A positional scheme is usually used to represent a number in any of the number systems. This means that each digit will have its value according to its position in a number. The number of digits in a position is also referred to as the base. For example, the binary system has base 2, the decimal system has base 10, and the hexadecimal system has base 16.

NUMBER CONVERSION

A.1

A number in any base system can be represented in a generalized format as follows:

$$N = A_nB^n + A_{n-1}B^{n-1} + \dots + A_1B^1 + A_0B^0$$

N = number, B = base, A = any digit in that base

For example, number 154 can be represented in various number systems as follows:

$$\text{Decimal: } 154 = 1 \times 10^2 + 5 \times 10^1 + 4 \times 10^0 = 154$$

$$\begin{aligned}\text{Octal: } 232 &= 2 \times 8^2 + 3 \times 8^1 + 2 \times 8^0 \\ &= 128 + 24 + 2 = 154\end{aligned}$$

$$\begin{aligned}\text{Hexadecimal: } 9A &= 9 \times 16^1 + A \times 16^0 \\ &= 144 + 10 = 154\end{aligned}$$

Binary: 10011010

$$\begin{aligned}&= 1 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 \\ &= 128 + 0 + 0 + 16 + 8 + 0 + 2 + 0 = 154\end{aligned}$$

The above example also shows how to convert a given number in any system into its decimal equivalent.

CONVERSION TABLE: DECIMAL, BINARY, OCTAL, AND HEXADECIMAL

Decimal	Hex	Binary	Octal
0	0	0000	00
1	1	0001	01
2	2	0010	02
3	3	0011	03
4	4	0100	04
5	5	0101	05
6	6	0110	06
7	7	0111	07
8	8	1000	10
9	9	1001	11
10	A	1010	12
11	B	1011	13
12	C	1100	14
13	D	1101	15
14	E	1110	16
15	F	1111	17

HOW TO CONVERT A NUMBER FROM BINARY INTO HEXADECIMAL AND OCTAL

Example
A.1

Convert the binary number 1 0 0 1 1 0 1 0 into its Hex and octal equivalents.

Hexadecimal

Step 1: Starting from the right (LSB) arrange the binary digits in groups of four.

1 0 0 1 1 0 1 0

Step 2: Convert each group into its equivalent Hex number.

9 A

Octal

Step 1: Starting from the right (LSB) arrange the binary digits in groups of three.

1 0 0 1 1 0 1 0

Step 2: Convert each group into its equivalent octal number.

2 3 2

2'S COMPLEMENT AND ARITHMETIC OPERATIONS

A.2

The 8085 microprocessor performs the subtraction of two binary numbers using the 2's complement method. In digital logic circuits, it is easier to design a circuit to add numbers than to design a circuit to subtract numbers. The 2's complement of a binary number is equivalent to its negative number; thus by adding the complement of the subtrahend (the number to be subtracted) to the minuend, a subtraction can be performed. The method of 2's complement is explained below with the examples from the decimal number system.

DECIMAL SUBTRACTION

Subtract the following two decimal numbers using the borrow method and the 10's complement method: $52 - 23$

Example

A.2

$$\begin{aligned} \text{Minuend: } 52 &= 5 \times 10 + 2 \\ \text{Subtrahend: } 23 &= 2 \times 10 + 3 \end{aligned}$$

Borrow

Method

Step 1: To subtract 3 from 2, 10 must be borrowed from the second place of the minuend.

$$52 = 4 \times 10 + 12$$

Step 2: The subtraction of the digits in the first place and the second place is as follows.

$$\begin{array}{r} 52 = 4 \times 10 + 12 \\ -23 = 2 \times 10 + 3 \\ \hline 2 \times 10 + 9 = 29 \end{array}$$

10's
Complement
Method

Step 1: Find the 9's complement of the subtrahend (23), meaning subtract each digit of the subtrahend from 9.

$$\begin{array}{r} \text{9's complement of 23: } & 9 & 9 \\ & -2 & 3 \\ \hline & 7 & 6 \end{array}$$

Step 2: Add 1 to the 9's complement to find the 10's complement of the subtrahend.

$$\begin{array}{r} \text{10's complement of 23: } & 76 \\ & + 1 \\ \hline & 77 \end{array}$$

The reason to find the 9's complement is to demonstrate a similar procedure to find the 2's complement of a binary number. However, in reality, the 10's complement of 23 is equivalent to subtracting 23 from 100.

Step 3: Add the 10's complement of the subtrahend (77) to the minuend (52) to subtract 23 from 52.

$$\begin{array}{r} \text{10's complement of 23: } & 77 \\ \text{Minuend: } & + 52 \\ \hline & 1 \ 29 = 29 & \text{(by dropping the most significant digit)} \end{array}$$

The elimination of the most significant bit is equivalent to subtracting 100 from the sum. This is necessary to compensate for the 100 that was added to find the 10's complement of 23.

Example
A.3

Perform the subtraction of the following two numbers using the borrow method and the 10's complement method: $23 - 52$.

Borrow
Method

$$\begin{array}{l} \text{Minuend: } 2 \ 3 \\ \text{Subtrahend: } 5 \ 2 \end{array}$$

Step 1: The subtraction of the digits in the first place results in: $3 - 2 = 1$.

Step 2: To subtract the digits in the second place, a borrow is required from the third place. Assuming the borrow is available from the third place, the digit 5 can be subtracted from 2 as follows:

$$\begin{array}{r} 1 \ 2 \\ - 5 \\ \hline 1 \ 7 \end{array} \text{ (the nonexistent borrow is shown with the bar)}$$

$$\begin{array}{r} \text{Result: } 23 \\ - 52 \\ \hline 1 \ 71 \end{array}$$

The same result is obtained with the 10's complement method, as shown below.

Step 1: Find the 9's complement of the subtrahend (52).

$$\begin{array}{r} \text{9's complement of 52:} & 9 & 9 \\ & -5 & 2 \\ \hline & 4 & 7 \end{array}$$

10's
Complement
Method

Step 2: Add 1 to the 9's complement to find 10's complement: $47 + 1 = 48$

Step 3: Add the 10's complement of the subtrahend to the minuend.

$$\text{10's complement of 52: } 48$$

$$\text{Minuend: } \underline{23}$$

$\underline{\quad\quad\quad}$ 71 (this is negative 29, expressed in 10's complement)

By examining these two examples, the following conclusions can be drawn and these conclusions can be used for any number system.

1. The complement of a number is its equivalent negative number.
 2. A number can be subtracted by using its complement.
 3. The sum of a number and its complement results in 0 if the most significant digit of the sum is ignored.
 4. When the subtrahend is larger than the minuend, the result of the 10's complement method is negative, and it is expressed in terms of 10's complement. The same result can be obtained by borrowing a digit from the most significant position.
-

PROCEDURE TO FIND THE 2'S COMPLEMENT OF A BINARY NUMBER

Step 1: Find the 1's complement. This amounts to replacing 0 by 1 and 1 by 0.

Step 2: To find the 2's complement, add 1 to the 1's complement. This is similar to the procedure of 10's complement.

Find the 2's complement of the binary number.

Example
A.4

$$0\ 0\ 0\ 1\ \underline{1\ 1\ 0\ 0}\ \text{(1CH or } 28_{10})$$

Step 1: Find the 1's complement, meaning replace 0 with 1 and 1 with 0.

$$\text{1's complement} = \underline{1\ 1\ 1\ 0\ \underline{0\ 0\ 1\ 1}}$$

Step 2: Add 1

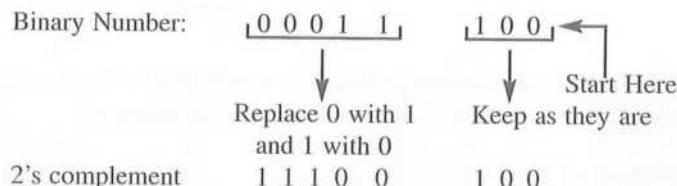
$\underline{1\ 1\ 1\ 0\ \underline{0\ 1\ 0\ 0}}$	$+$	1
--	-----	---

By examining the result of the example, the following rule can be stated to find the 2's complement of a binary number, instead of the above procedure of the 1's complement.

Rule 1: Start at the LSB of a given number, and check all the bits to the left. Keep all the bits as they are up to and including the least significant 1.

Rule 2: After the first 1, replace all 0s with 1s and 1s with 0s.

These rules can be applied to the given binary number (1CH), as illustrated below:



The 2's complement of the number can be verified by adding the complement to the original number as follows; the sum should be 0:

$$\begin{array}{r}
 \text{Binary Number: } 0\ 0\ 0\ 1\ 1\ 1\ 0\ 0 \\
 \text{2's Complement: } \underline{1\ 1\ 1\ 0\ 0\ 1\ 0\ 0} \\
 \hline
 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0 \text{ (ignore the MSB)}
 \end{array}$$

BINARY SUBTRACTION USING 2'S COMPLEMENT

The binary subtraction can be performed by using the 2's complement method; if the result is negative, it is expressed in terms of 2's complement.

**Example
A.5**

Subtract 32H (0011 0010) from 45H (0100 0101).

$$\begin{array}{r}
 \text{Subtrahend: } 32H = 0\ 0\ 1\ 1\ 0\ 0\ 1\ 0 \\
 \text{2's complement of } 32H = 1\ 1\ 0\ 0\ 1\ 1\ 1\ 0 \\
 + \\
 \text{Minuend: } 45H = 0\ 1\ 0\ 0\ 0\ 1\ 0\ 1 \\
 \text{CY} \quad \underline{1\ 1\ 1} \\
 \hline
 0\ 0\ 0\ 1\ 0\ 0\ 1\ 1 = 13H
 \end{array}$$

**Example
A.6**

Subtract 45H (0100 0101) from 32H (0011 0010).

$$\begin{array}{r}
 \text{Subtrahend: } 45H = 0\ 1\ 0\ 0\ 0\ 1\ 0\ 1 \\
 \text{2's complement of } 45H = 1\ 0\ 1\ 1\ 1\ 0\ 1\ 1 \\
 + \\
 \text{Minuend: } 32H = 0\ 0\ 1\ 1\ 0\ 0\ 1\ 0 \\
 \text{CY} = \underline{1\ 1\ 1} \\
 \hline
 1\ 1\ 1\ 0\ 1\ 1\ 0\ 1 = EDH
 \end{array}$$

The result is negative and it is expressed in 2's complement. This can be verified by taking the 2's complement of the result; the 2's complement of the result should be 13H as in Example A.5.

$$\begin{array}{l} \text{Result EDH} = 1\ 1\ 1\ 0\ \ 1\ 1\ 0\ 1 \\ \text{Two's complement of EDH} = 0\ 0\ 0\ 1\ \ 0\ 0\ 1\ 1 = 13H \end{array}$$

SIGNED NUMBERS

To perform the arithmetic operations with signed numbers (positive and negative), the sign must be indicated as well as the magnitude of the number. In 8-bit microprocessors, bit D₇ is used to indicate the sign of a number; 0 in D₇ indicates a positive number and 1 indicates a negative number. Bit D₇ can be used to indicate the sign of a number because

1. the 8085/8080A performs the subtraction of two numbers using 2's complement and, if the result is negative, it saves (shows) the result in the form of 2's complement.
2. the 2's complement of all the 7-bit numbers have 1 in D₇.

When a programmer uses bit D₇ to indicate the sign of a number, the magnitude of the number can be represented by seven bits (D₆–D₀). For example, number 74H is represented with a sign as follows:

D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
+74H	= 0	1	1	1	0	1	0
-74H	= 1	0	0	0	1	1	0

(2's complement of 74H)

However, the microprocessor cannot differentiate between a positive number and a negative number. For example, in the above illustration, -74H can be interpreted as the unsigned positive number 8CH or the bit pattern. It is the responsibility of the programmer to provide the necessary interpretation.

SUBTRACTION PROCESS IN THE 8085 MICROPROCESSOR

The 8085 performs the following operations when it subtracts (SUB or SUI) two binary numbers:

- Step 1:** Finds 1's complement of the subtrahend.
- Step 2:** Finds 2's complement of the subtrahend by adding 1 to the result of Step 1.
- Step 3:** Adds 2's complement of the subtrahend to the minuend.
- Step 4:** Complements the CY flag.

These steps are internal to the microprocessor and invisible to the user; only the result is available to the user.

**Example
A.7**

Show the internal steps performed by the microprocessor to subtract the following unsigned numbers:

- FAH – 62H
- 62H – FAH

a. Minuend: FAH = 1 1 1 1 1 0 1 0
Subtrahend: 62H = 0 1 1 0 0 0 1 0

Step 1: 1's complement of 62H = 1 0 0 1 1 1 0 1

Step 2: Add 1

$$\begin{array}{r} 1 0 0 1 \\ + \quad 1 \\ \hline 1 1 1 0 \end{array}$$

Step 3: Add minuend (FAH)

$$\begin{array}{r} 1 1 1 0 \\ + 1 1 1 1 \\ \hline 1 1 0 0 \end{array}$$

Step 4: Complement CY

$$\begin{array}{r} 1 1 0 0 \\ + 1 0 0 1 \\ \hline 1 0 0 0 \end{array}$$

Result:

$$\begin{array}{r} 0 1 0 0 1 \\ 0 1 0 0 1 \\ \hline 1 0 0 0 \end{array}$$

CY = 0, S = 1, Z = 0, P = 0

b. Minuend: 62H = 0 1 1 0 0 0 1 0
Subtrahend: FAH = 1 1 1 1 1 0 1 0

Step 1: 1's complement of FAH = 0 0 0 0 0 1 0 1

Step 2: Add 1

$$\begin{array}{r} 0 0 0 0 \\ + \quad 1 \\ \hline 0 0 0 1 \end{array}$$

Step 3: Add minuend (62H)

$$\begin{array}{r} 0 0 0 1 \\ + 0 1 1 0 \\ \hline 0 0 1 1 \end{array}$$

Step 4: Complement CY

$$\begin{array}{r} 0 0 1 1 \\ + 1 0 1 1 \\ \hline 1 0 0 0 \end{array}$$

Result:

$$\begin{array}{r} 1 0 1 1 0 \\ 1 0 1 1 0 \\ \hline 1 0 0 0 \end{array}$$

CY = 1, S = 0, Z = 0, P = 0

This result is negative and is expressed in 2's complement of the magnitude.

Results

- FAH – 62H = 98H (positive), CY = 0, S = 1
- 62H – FAH = 68H (negative), CY = 1, S = 0

These results and associated flags appear to be confusing. In Example A.7a, the result is positive but the sign flag indicates that it is negative. On the other hand, in Example A.7b, the result is negative but the sign flag indicates that it is positive. This confusion can be explained as follows:

- This subtraction is concerned with the unsigned numbers; therefore, the sign flag is irrelevant. In signed arithmetic, the number FAH is invalid because it is an 8-bit number.
- The programmer can check whether the result indicates the true magnitude by checking the CY flag. If CY is reset, the result is positive, and if CY is set, the result is expressed in 2's complement.

In Example A.7a, assume that the numbers are signed numbers, and interpret the result.

Example
A.8

Minuend: FAH This is a negative number because $D_7 = 1$; therefore, this must be represented in 2's complement. The magnitude of the number can be found by taking the 2's complement of FAH:

$$\begin{aligned} \text{FAH} &= 1\ 1\ 1\ 1\ 1\ 0\ 1\ 0 \\ \text{2's complement of FAH} &= 0\ 0\ 0\ 0\ 0\ 1\ 1\ 0 \\ &= 06H \text{ (magnitude)} \end{aligned}$$

Subtrahend: 62H This is a positive number because $D_7 = 0$. The problem given in 7a can be represented as follows:

$$\begin{aligned} \text{FAH} - 62H &= (-06H) - (+62H) \\ &= -68H \end{aligned}$$

The final result is $-68H$, which will be in the form of its 2's complement:

$$\begin{aligned} -68H &= -(0\ 1\ 1\ 0\ 1\ 0\ 0\ 0) \\ \text{2's complement of } 68H &= 1\ 0\ 0\ 1\ 1\ 0\ 0\ 0 \\ &= 98H \end{aligned}$$

The final answer is the same as before; however, it will be interpreted as a negative number with the magnitude of $68H$. When signed numbers are used in arithmetic operations, the sign flag will indicate the proper sign of the result.

Add the following two positive numbers and interpret the sign flag: +41H, +54H.

Example
A.9

$$\begin{array}{r} 41H = 0\ 1\ 0\ 0\ 0\ 0\ 0\ 1 \\ + \\ 54H = 0\ 1\ 0\ 1\ 0\ 1\ 0\ 0 \\ \hline 95H = 1\ 0\ 0\ 1\ 0\ 1\ 0\ 1 \end{array} \quad S = 1, CY = 0, Z = 0$$

This is an addition of two positive numbers; therefore, the sign flag indicates that the sum is larger than seven bits. This is also known as overflow. If this had been the sum of two unsigned numbers, the sign flag would have had no significance.

B

Introduction to the EMAC Primer*

B.1 THE PRIMER TRAINER

B.1.1 System Description

The Primer is a low-cost single-board microcomputer from EMAC Inc., designed with the 8085 microprocessor. The system includes all the necessary features for training purposes in college laboratories. It will be referred to as the Primer or the trainer.

Figure B.1 shows the functional block diagram of the basic system. It includes the 8085 microprocessor, EPROM (32K), the 8155 (R/W memory with I/O and timer), 20-key Hex keyboard, and six seven-segment LEDs for display. The keyboard and the display are interfaced with the 8085 through the keyboard/display controller, the 8279. In addition, the board includes an 8-position dip switch, 8 LEDs, D/A and A/D converters, and a speaker port.

*Courtesy of EMAC Inc. This trainer is designed, manufactured, and marketed by EMAC Inc., 11 Emac Way, Carbondale, Illinois 62901 (Tel. 618-529-4525)/www.emacinc.com

The keyboard enables the user to enter and store the 8085 Hex machine code representing the 8085 assembly language programs in R/W memory. The seven-segment LEDs are used to display memory addresses and their contents while entering, modifying, or examining the programs. The 8-position dip switch can be used as an input port and 8 LEDs can be used as an output port. A program can be executed using the function keys on the keyboard, and results can be either displayed at 8 LEDs or stored in memory.

The Primer system can be expanded to include additional 32K EPROM, 32K R/W memory, the 8251 communication port, and I/O ports.

THE EPROM

This is a 32K memory with a memory address range from 0000H to 7FFFH. The system monitor program, called MOS (Monitor Operating System), is stored permanently in this memory. This program continuously monitors the keyboard and displays the Hex keys at seven-segment LEDs. The primary function of the MOS is to enable the user to enter a program (instructions in Hex code) in the trainer's R/W memory, modify it if necessary, execute and debug the program.

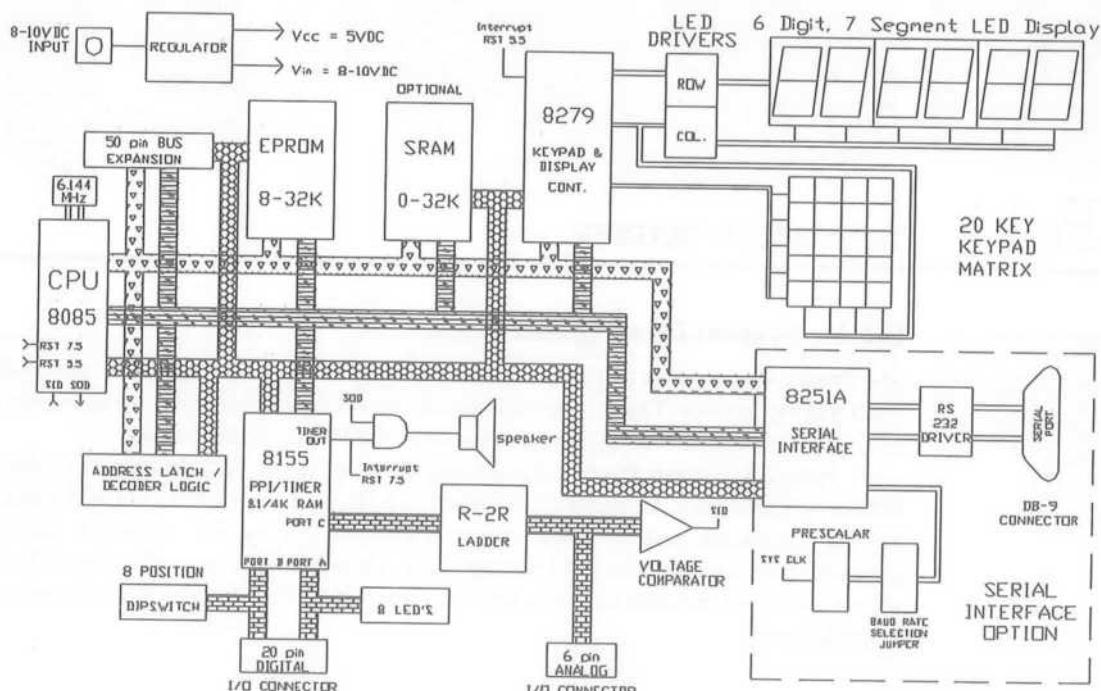


FIGURE B.1

Primer Functional Block Diagram

THE 8155

This chip has 256 bytes of R/W memory, two I/O ports, and one timer. The R/W memory addresses of the 8155 range from FF00H to FFFFH; however, memory locations from FFD0H to FFFFH are reserved for the use of the monitor program. In this memory, the user can enter a program and execute it using the keyboard.

The two I/O ports of the 8155 can be used to interface additional devices, and the timer can be used to provide various timing functions such as time delays.

THE 8279

This is a programmable device used to handle keyboard and display. It displays memory addresses and data from its internal memory at the seven-segment LEDs using the multiplexed technique. When a key is pressed, the 8085 is interrupted, and a new key is stored in the internal memory of the 8279 and displayed.

B.1.2 Keyboard

The keyboard has 20 keys; 16 keys for the Hex digits 0 to F are used for entering data, and the remaining four keys are used to perform various functions. In addition, Hex keys are also assigned to perform other functions. The monitor program operates in two modes: data entry mode and function mode. The functions of these keys are described as follows.

1. 0 to F: Enter Hex digits.
2. Enter: Stores the displayed data in memory, increments the address of the memory location, and displays the address and data of the new location.
3. Dec: Decrements the memory address and displays the new address and its data.
4. Step: Executes one instruction at a time, called single stepping.
5. Func: Selects the second function for Hex keys. The monitor reverts to the data entry mode when the key is pressed twice.

Additional functions of Hex keys:

6. B.P. (Break Point): Displays the current break point address.
7. S.C. (Stack Contents): Displays two bytes from the top of the stack.
8. Run: Executes the program from the displayed memory address.
9. A.F., B.C., D.E., H.L., S.P., and P.C.: Displays contents of the specified registers, high-order byte on the left and low-order byte on the right.
10. Key 1: Invokes the Primer diagnostics.
11. Key 2: Executes the MOS service selected by the value in register C.
12. Key 3: Enables serial I/O for downloading a program from the PC. (Requires upgrade option.)
13. Key 4: Invokes the EPROM programmer menu-driven interface. (Requires EPROM programmer option.)

TABLE B.1
Primer Memory Map

Memory	
0000	Monitor ROM
7FFF	Slot 0 (32K)
4000	Slot 1 (32K)
BFFF	
C000	Foldback or Mirror
EFFF	8155 Memory
FF00	User Memory
FFFF	8155 R/W Memory

B.1.3 Memory Map

The memory map of the system is shown in Table B.1. The user memory ranges from FF00H to FFFFH; however, locations FFD0H to FFFFH are reserved for user interrupts and the monitor stack.

B.1.4 Available Subroutines

The MOS program includes many subroutines called services that are available to the user. The user must supply a service number in register C and call location 1000H. See the Primer manual for details.

B.2

USING THE PRIMER

The Monitor Operating System (MOS) of the Primer can be used to

- enter programs in its R/W memory (F001H–FFCFH).
- examine and modify the contents of memory, registers, and stack.
- execute programs.
- debug programs using single step and breakpoint techniques.

The user can display results of a program at the LED port in binary and/or at the system's seven-segment LEDs in Hex or BCD by calling appropriate display routines from the monitor program.

B.2.1 How to Enter a Program

When the Primer is turned on, it shows the memory address F001H and random data at seven-segment LED display. The memory address is shown by the left four LEDs, called the address field, and the random data byte is shown on the rightmost two LEDs, called

the data field. When the user presses various keys, displays will be as follows (each key press is indicated by a bracket around the key symbol).

1. To examine the contents of R/W memory:

Press	Displays		Data
	Address	Field	
[RESET]	F 0 0 1	XX	;Keys in data entry mode
[ENTER]	F 0 0 2	XX	;XX indicates random data
[ENTER]	F 0 0 3	XX	

2. To change the contents of memory and load a program:

Changing the contents of memory and loading a program are similar functions. When we load a program, we enter Hex codes in memory locations for given instructions. We will accomplish both functions in the following steps. If you are not familiar with the 8085 instruction set, the comments on the right will indicate the function of a given instruction.

The following three instructions load a byte (96H) in the accumulator and display the byte at the user LED port in binary.

[RESET]	F 0 0 1	XX		
[3][E]	F 0 0 1	3 E	MVI A, 96H	;Load 96H in A
[ENTER]	F 0 0 2	XX		
[9][6]	F 0 0 2	96		
[ENTER]	F 0 0 3	XX		
[D][3]	F 0 0 3	D 3	OUT 11	;Display 96H at port #11H
[ENTER]	F 0 0 4	XX		
[1][1]	F 0 0 4	11		
[ENTER]	F 0 0 5	XX		
[F][F]	F 0 0 5	F F	RST 7	;End of program
[ENTER]	F 0 0 6	XX		
[RESET]	F 0 0 1	3E		

The last key [RESET] will take the display at the location F001H.

B.2.2 To Execute a Program

To execute a program, direct the processor to the first instruction of a program being executed by loading the address of the instruction in the program counter. In our example above, the program begins at location FF01H. The Primer is designed to begin at location FF01H. By pushing the RESET key, we direct the processor to the beginning of the program.

To execute a program, the following key sequence should be followed:

Press	Address Field	Data Field
[RESET]	F 0 0 1	3 E
[FUNC]		FUNC
[STEP/RUN]		

To single step:

[RESET]	F 0 0 1	3 E
[STEP/RUN]		

B.2.3 How to Examine and Change Register Contents

To examine:

[RESET]	F 0 0 1	3 E
[FUNC]		FUNC
[A.F.]	9 6 X X	A.F.

This shows 96H in the accumulator and a random byte in the flag register.
To change the contents, press four data keys as follows.

[3][5][0][0]		
[ENTER]	3 5 0 0	A.F.

These keys place 35H in A and clear the flag register. This is a right-entry mode; therefore, if you press two keys (such as 35), followed by the ENTER key, the contents of the flag register will be shifted into A and the flag register will show 35H.

B.2.4 How to Execute a Program Using the Program Counter (PC)

Another approach to execute a program is to enter the address of the first instruction in the program counter. This requires placing MOS in the function mode and entering the address in the PC as follows.

Let us assume that your program begins at memory location FF20H. Enter all the instructions of the program to be executed, and press the following key sequence.

[FUNC]	FUNC		
[P.C.]	X X X X	P.C.	;Address in PC will be where the last instruction is entered
[F][F][2][0]	F F 2 0	P.C.	
[STEP/RUN]			

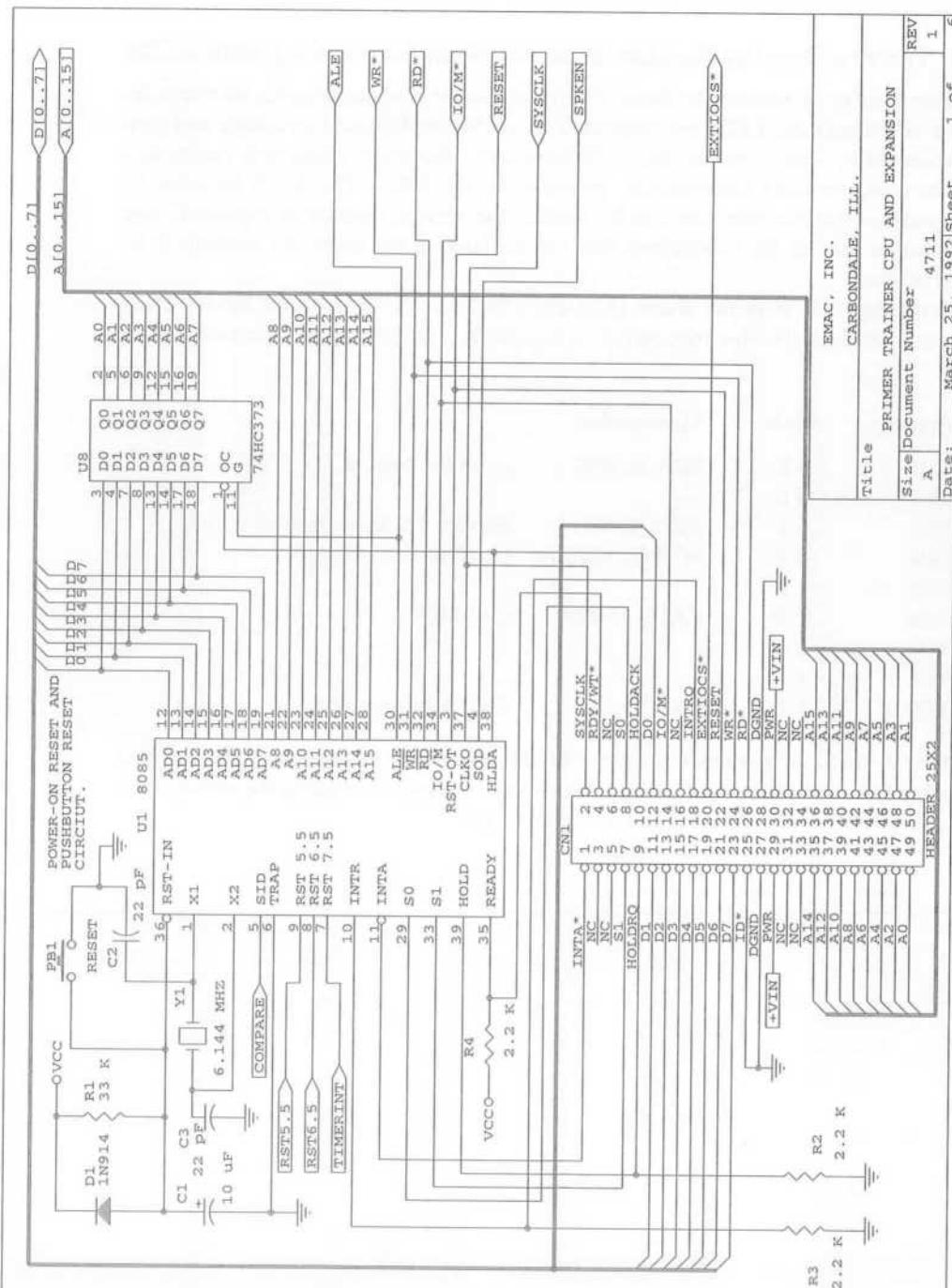
B.2.5 How to Display Results at the System Seven-Segment LEDs

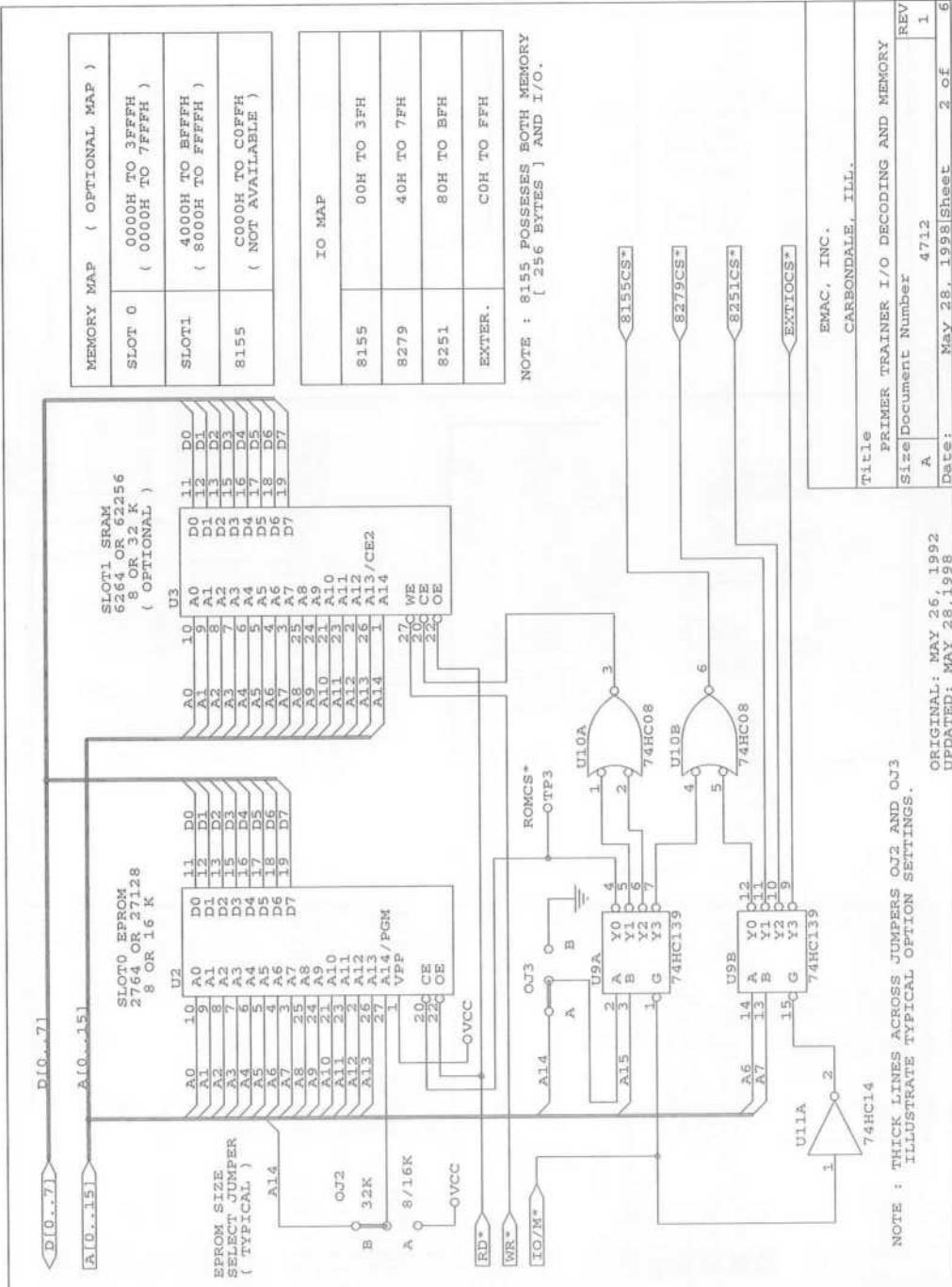
The Primer display is managed by the 8279 programmable keyboard/display interface device. The seven-segment LEDs are connected to use the multiplexed technique and cannot be accessed by simply writing the OUT instruction. However, a data byte can be displayed by using services (subroutines) provided by the MOS. The MOS includes 36 service routines that the user can call by loading the service number in register C (the Primer manual lists all the subroutines that are available to the user). An example is illustrated below.

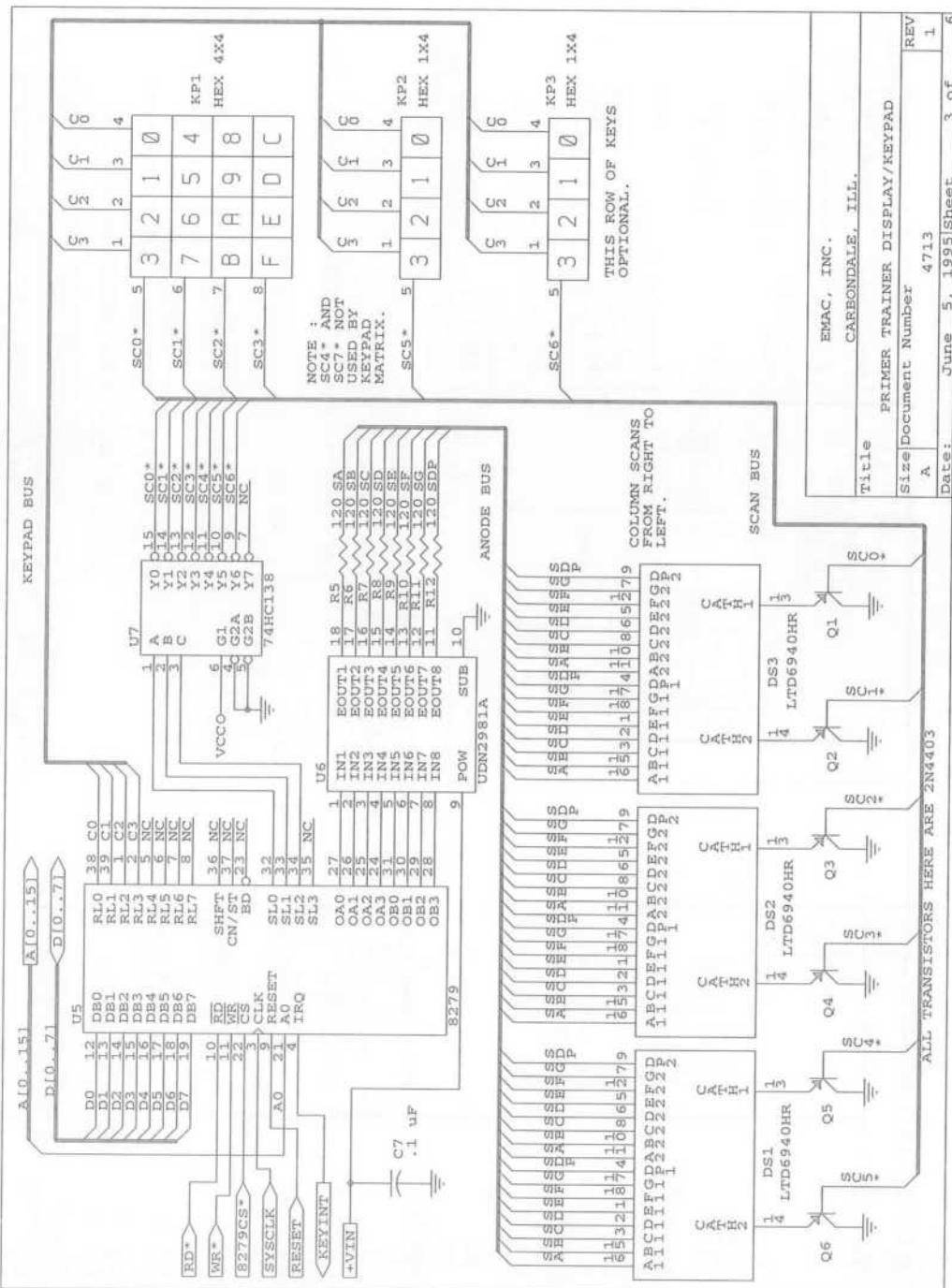
To display 96H from the above illustration (B.2.1), we need to call service 1BH. This service displays the Hex byte stored in register E. The instructions are modified as follows:

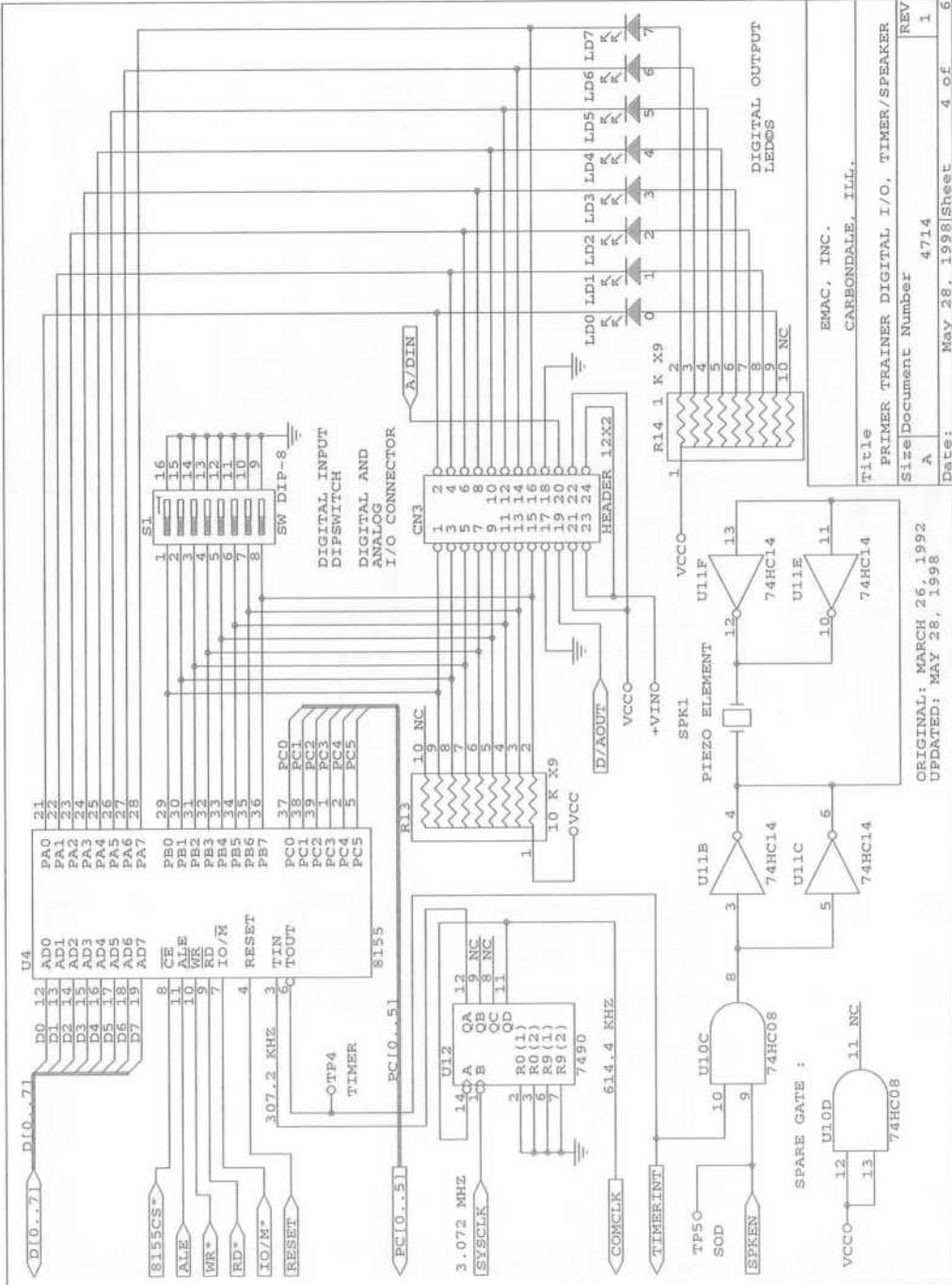
Memory	Code	Mnemonics	
F001	3 E	MVI A, 96H	;Load the byte in A
F002	9 6		
F003	5 F	MOV E,A	;Service 1B displays what is in E
F004	0 E	MVI C, 1BH	;Load service # in C
F005	1 B		
F006	C D	CALL 1000H	;Call MOS
F007	0 0		
F008	1 0		
F009	F F	RST 7	;End of instructions

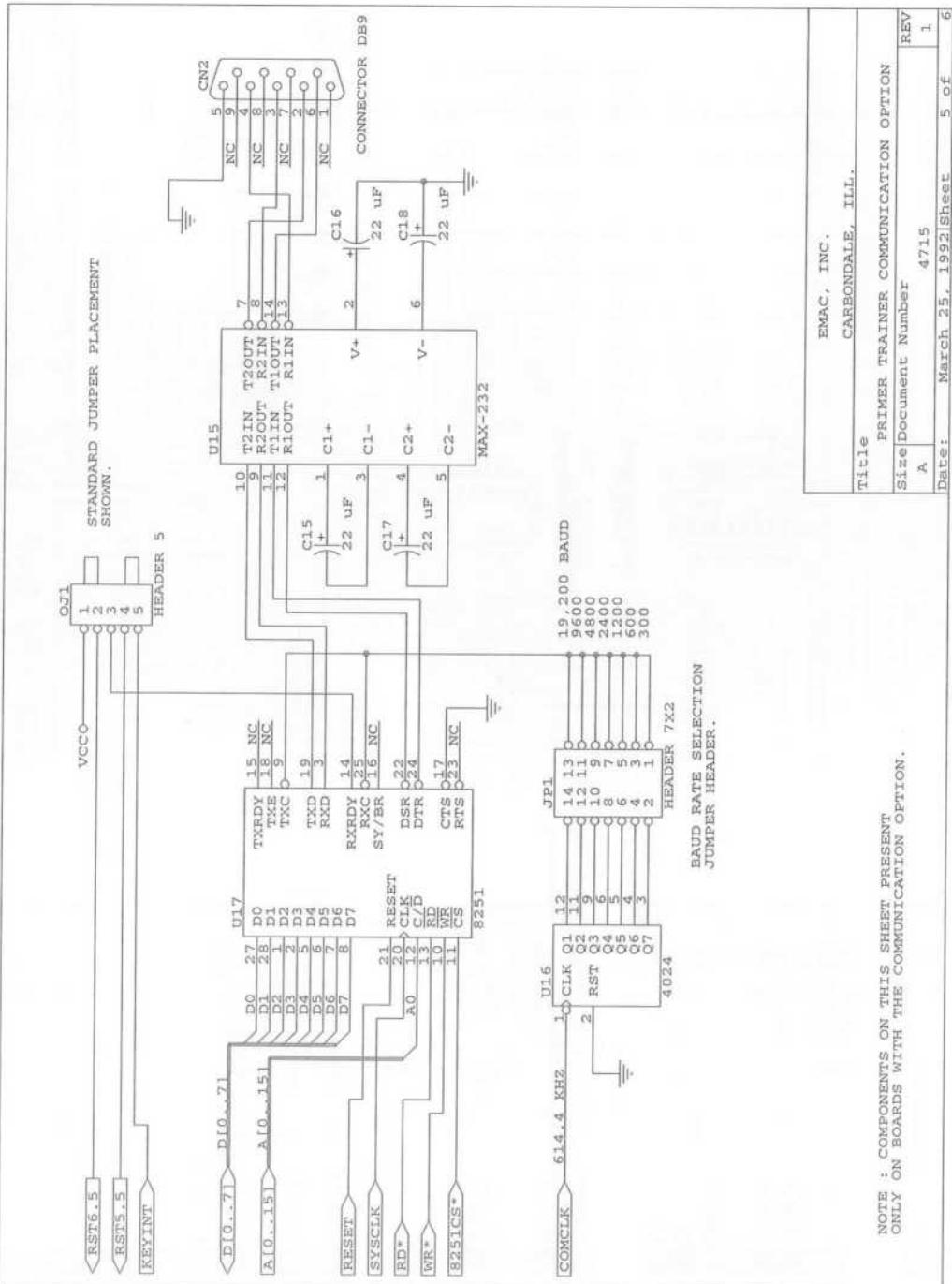
Enter these instructions starting at location F001H and execute the program. The program will display 96H in the data field of the seven-segment display. Similarly, the service 12H displays the contents of register DE in the address field.

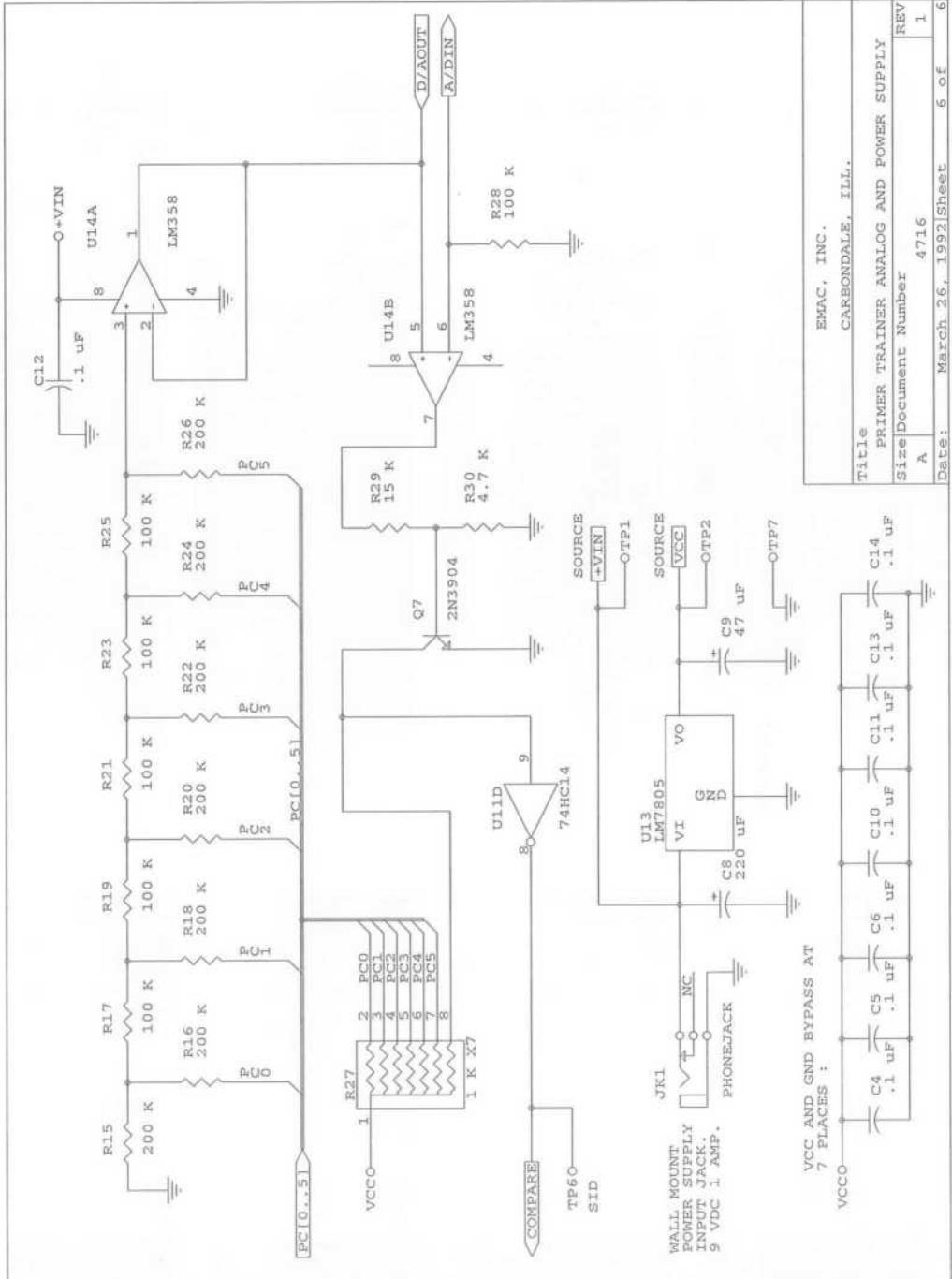










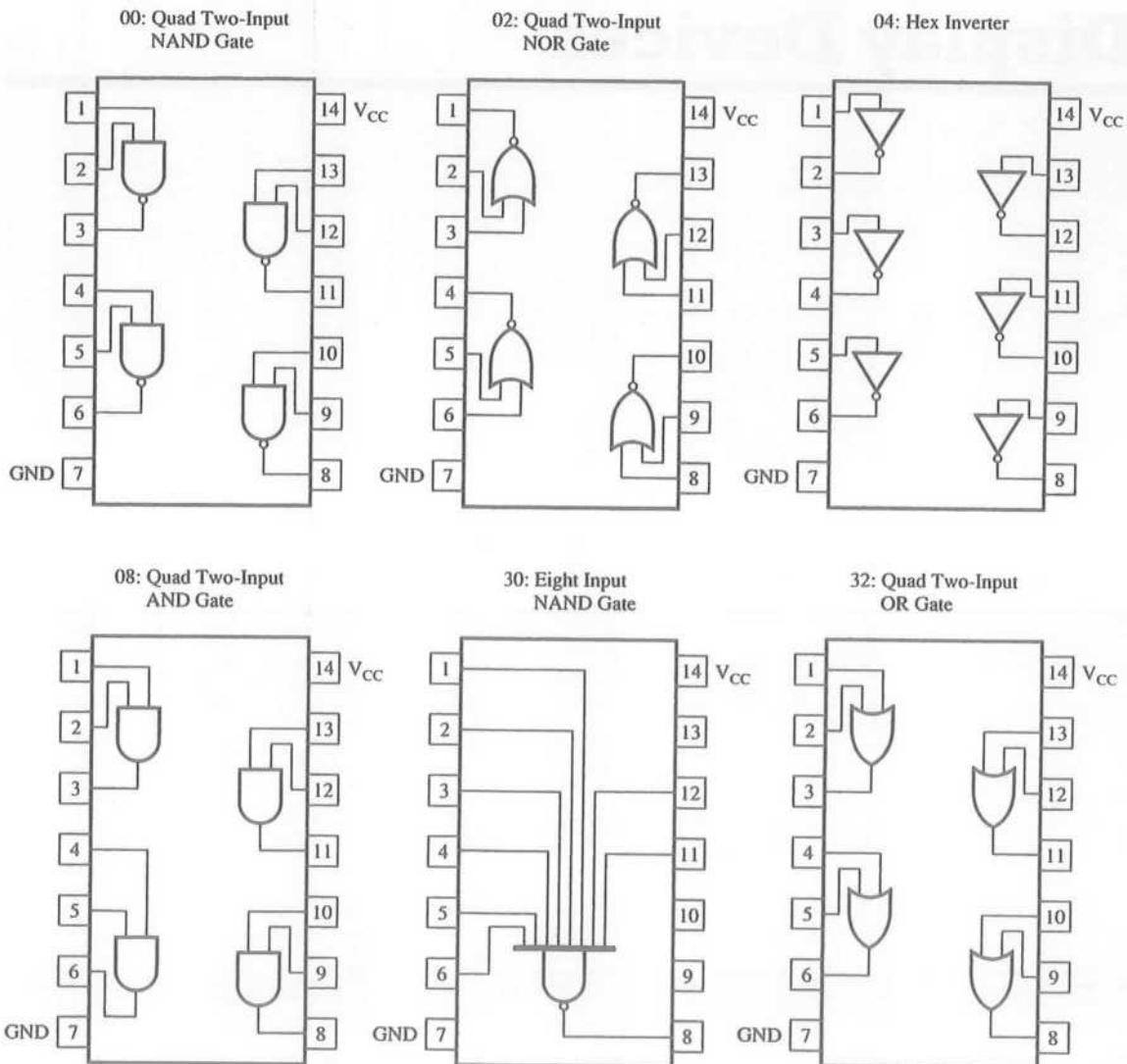


C

Pin Configuration of Selected Logic and Display Devices

C.1

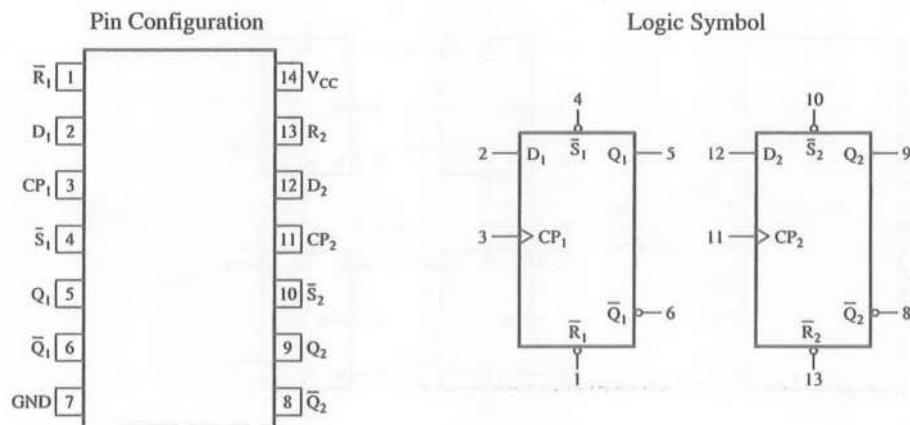
SELECTED LOGIC AND DISPLAY DEVICES: PIN CONFIGURATION AND LOGIC SYMBOLS



74: DUAL D-TYPE POSITIVE EDGE-TRIGGERED FLIP-FLOP

Description This is a positive edge-triggered flip-flop with Set and Reset inputs and complementary outputs (Q and \bar{Q}).

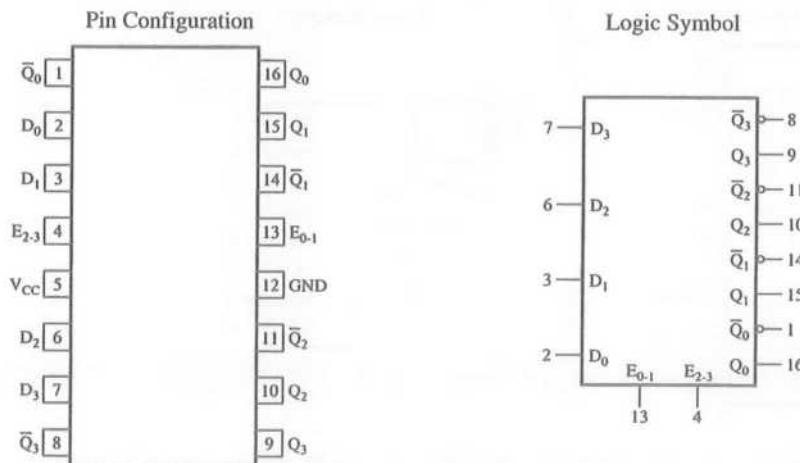
Information at D input is transferred to Q output on the positive edge (Low-to-High) of the clock pulse (CP). Set (S) and Reset (R) are asynchronous active low inputs and operate independently of the clock; S input sets Q to logic 1 and R resets Q to logic 0.



75: 4-BIT BISTABLE LATCH

Description '75 is a level sensitive 4-bit latch; each pair of bits is controlled by High Enable input E. It also has complementary outputs (Q and \bar{Q}).

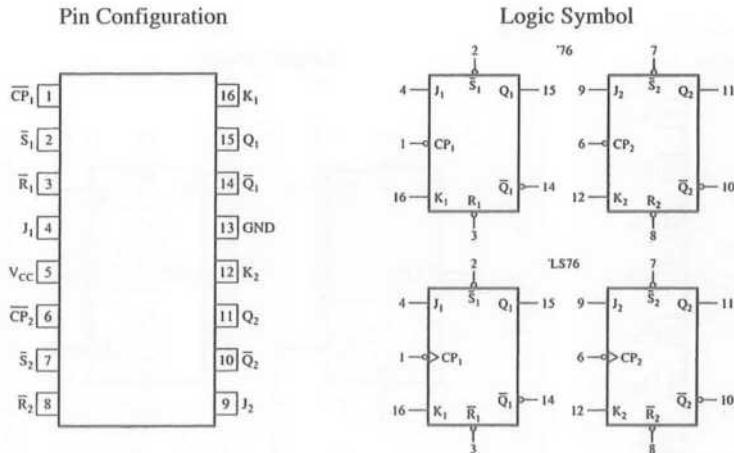
Information at D input is transferred to Q output when E is High, and Q follows D input as long as E is High. When E goes Low, D input is latched at the output and remains latched until E goes High again.



76: DUAL JK FLIP-FLOP

Description '76 is a dual JK flip-flop with J, K, Clock, Set, and Reset inputs for each flip-flop. JK information is loaded into the master when the Clock is high and transferred to the slave on the High-to-Low Clock transition.

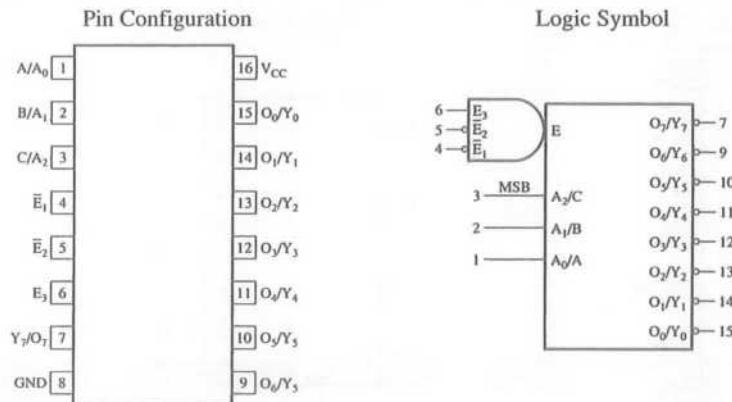
'LS76 is a negative edge-triggered flip-flop.



138: 3-TO-8 DECODER/DEMULTIPLEXER*

Description '138 has three binary weighted inputs (C, B, A or A_2, A_1, A_0) and eight mutually exclusive active Low outputs. It has three enable inputs, two active Low, and one active High; all three must be enabled to obtain an output.

When '138 is enabled, one of the output signals goes active Low corresponding to the decimal equivalent of the input. When it is not enabled, all output signals remain High.

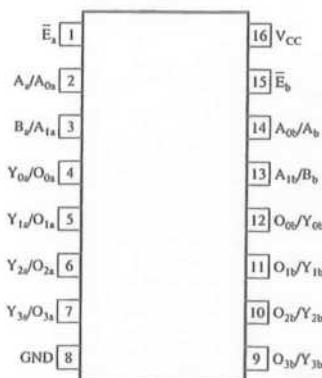


*Note: Pin configuration shows notations commonly used by various manufacturers.

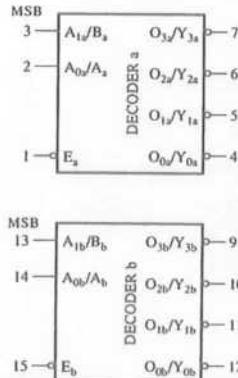
139: 2-TO-4 DECODER/DEMULTIPLEXER

Description '139 is a dual 2-to-4 decoder with two binary weighted inputs (B , A or A_1 , A_0) and one active Low enable input. When it is enabled, one of four output signals, corresponding to the decimal equivalent of the input, goes active Low. When it is not enabled, all output signals remain High.

Pin Configuration



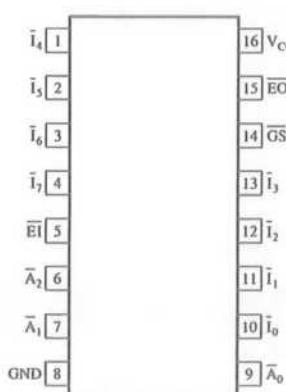
Logic Symbol



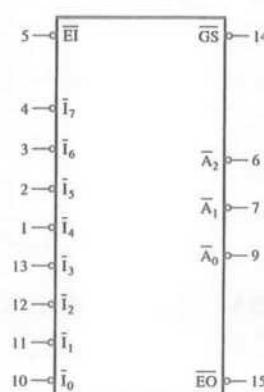
148: 8-INPUT PRIORITY ENCODER

Description '148 has eight active Low inputs (\bar{I}_7 to \bar{I}_0), one enable input, and three active Low outputs. When it is enabled, the output provides the binary equivalent of the active input. If multiple inputs go active simultaneously, the input with the highest priority is encoded on the output; other inputs are ignored. \bar{I}_7 has the highest priority, and \bar{I}_0 has the lowest priority.

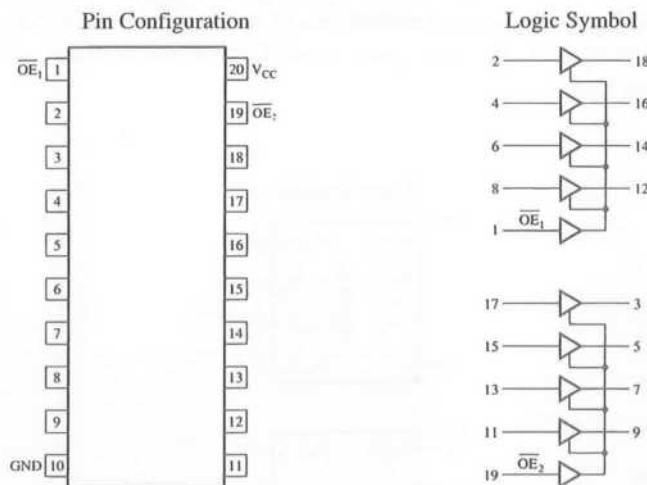
Pin Configuration



Logic Symbol

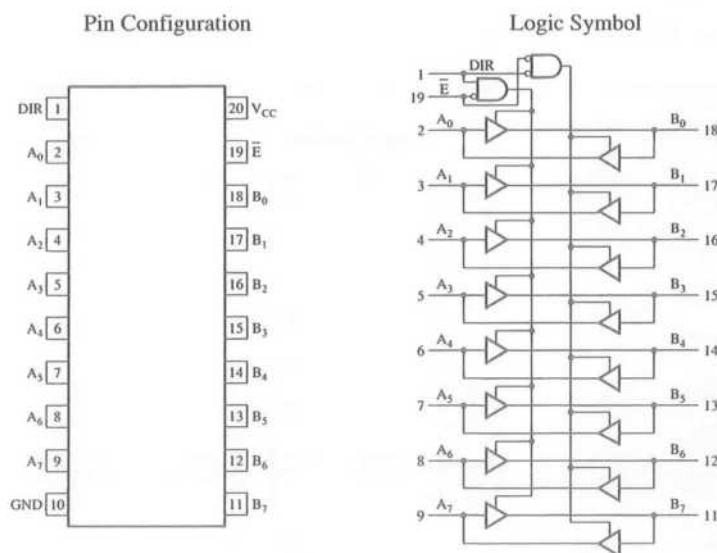


244: OCTAL BUFFER



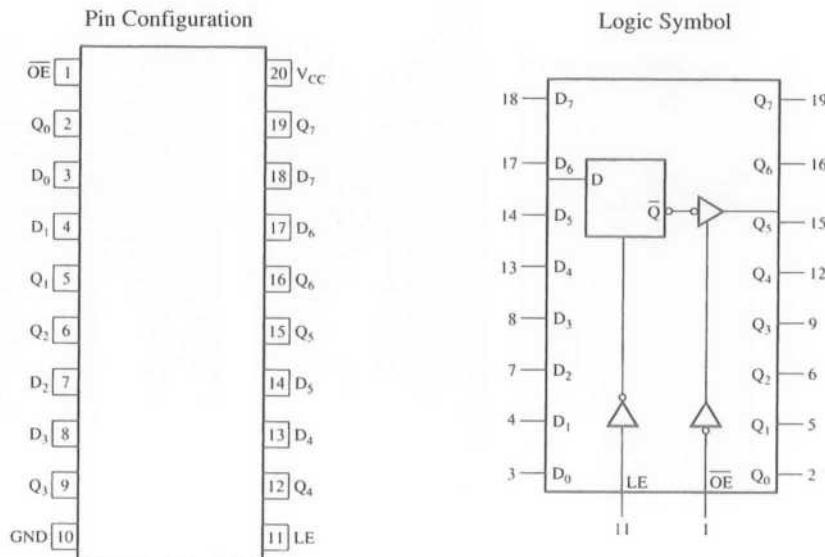
245: OCTAL TRANSCEIVER (BIDIRECTIONAL BUFFER)

Description '245 is a bidirectional buffer with \overline{E} (Enable) and DIR as two control inputs. \overline{E} is active Low, and it enables the buffer. DIR determines the direction of the data flow; if it is High, data flow from A to B and if it is low, data flow from B to A.

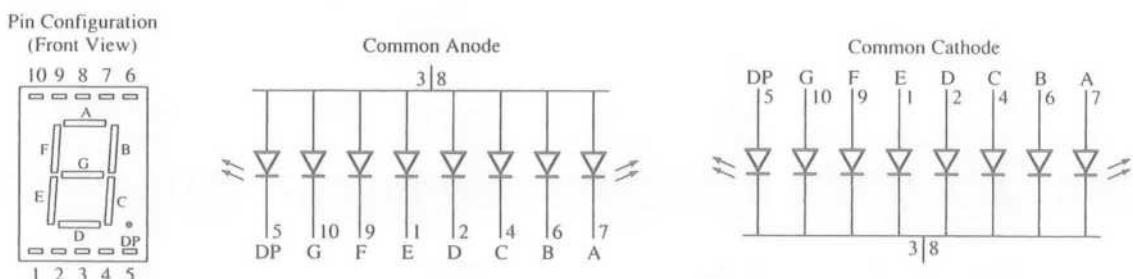


373: OCTAL TRANSPARENT LATCH

Description '373 is an octal transparent latch followed by tri-state output buffers. The data from D inputs are transferred to Q outputs when the latch enable (LE) signal is High and are latched when LE goes from High to Low. When OE goes Low, latched data appear on the outputs; otherwise, the outputs remain in high impedance.



SEVEN-SEGMENT NUMERIC DISPLAY: (10-PIN PACKAGE)



D

Specifications: Data Converters* and Peripheral Devices

Appendix D specifications begin on the next page.

*SOURCES:

Pages 670–680: Courtesy of National Semiconductor Corporation, *Linear Databook 2* (Santa Clara, Calif.: Author, 1988).

Pages 681–723: Reprinted by permission of Intel Corporation, Copyright Intel Corporation, 1995.

Pages 724–733: Courtesy of Hitachi America, Ltd.

DAC0808, DAC0807, DAC0806 8-Bit D/A Converters

General Description

The DAC0808 series is an 8-bit monolithic digital-to-analog converter (DAC) featuring a full scale output current settling time of 150 ns while dissipating only 33 mW with $\pm 5V$ supplies. No reference current (I_{REF}) trimming is required for most applications since the full scale output current is typically ± 1 LSB of $255 I_{REF}/256$. Relative accuracies of better than $\pm 0.19\%$ assure 8-bit monotonicity and linearity while zero level output current of less than $4 \mu A$ provides 8-bit zero accuracy for $I_{REF} \geq 2$ mA. The power supply currents of the DAC0808 series are independent of bit codes, and exhibits essentially constant device characteristics over the entire supply voltage range.

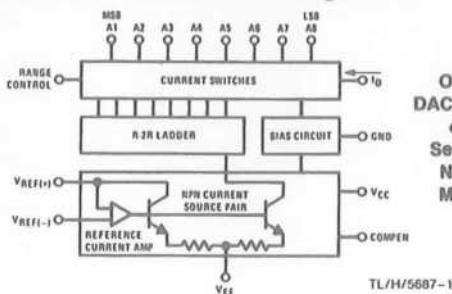
The DAC0808 will interface directly with popular TTL, DTL or CMOS logic levels, and is a direct replacement for the

MC1508/MC1408. For higher speed applications, see DAC0800 data sheet.

Features

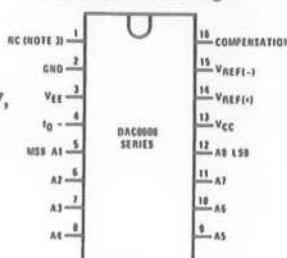
- Relative accuracy: $\pm 0.19\%$ error maximum (DAC0808)
- Full scale current match: ± 1 LSB typ
- 7 and 6-bit accuracy available (DAC0807, DAC0806)
- Fast settling time: 150 ns typ
- Noninverting digital inputs are TTL and CMOS compatible
- High speed multiplying input slew rate: $8 \text{ mA}/\mu\text{s}$
- Power supply voltage range: $\pm 4.5V$ to $\pm 18V$
- Low power consumption: 33 mW @ $\pm 5V$

Block and Connection Diagrams



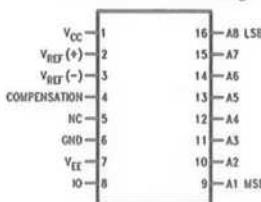
Order Number
DAC0808, DAC0807,
or DAC0806
See NS Package
Number J16A,
M16A or N16A

Dual-In-Line Package



TL/H/5687-2

Small-Outline Package



TL/H/5687-13

Ordering Information

ACCURACY	OPERATING TEMPERATURE RANGE	ORDER NUMBERS		
		J PACKAGE (J16A)*	N PACKAGE (N16A)*	SO PACKAGE (M16A)
8-bit	-55°C ≤ TA ≤ +125°C	DAC0808LJ	MC1508L8	
8-bit	0°C ≤ TA ≤ +75°C	DAC0808LCJ	MC1408L8	DAC0808LCM
7-bit	0°C ≤ TA ≤ +75°C	DAC0807LCJ	MC1408L7	DAC0807LCM
6-bit	0°C ≤ TA ≤ +75°C	DAC0806LCJ	MC1408L6	DAC0806LCM

*Note. Devices may be ordered by using either order number.

DAC0808 is compatible with the 1408.

Absolute Maximum Ratings (Note 1)

If Military/Aerospace specified devices are required, contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Power Supply Voltage

V_{CC}	+ 18 V _{DC}
V_{EE}	- 18 V _{DC}

Digital Input Voltage, V₅–V₁₂

- 10 V_{DC} to + 18 V_{DC}

Applied Output Voltage, V_O

- 11 V_{DC} to + 18 V_{DC}

Reference Current, I₁₄

5 mA

Reference Amplifier Inputs, V₁₄, V₁₅

V_{CC} , V_{EE}

Power Dissipation (Note 3)

1000 mW

ESD Susceptibility (Note 4)

TBD

Storage Temperature Range

- 65°C to + 150°C

Lead Temp. (Soldering, 10 seconds)

260°C

Dual-In-Line Package (Plastic)

300°C

Dual-In-Line Package (Ceramic)

215°C

Surface Mount Package

220°C

Vapor Phase (60 seconds)

Infrared (15 seconds)

Operating Ratings

Temperature Range

DAC0808L

DAC0808LC Series

$T_{MIN} \leq T_A \leq T_{MAX}$

- 55°C $\leq T_A \leq + 125^\circ\text{C}$

0 $\leq T_A \leq + 75^\circ\text{C}$

Electrical Characteristics

($V_{CC} = 5\text{V}$, $V_{EE} = - 15\text{V}_{DC}$, $V_{REF}/R_{14} = 2\text{ mA}$, DAC0808: $T_A = - 55^\circ\text{C}$ to $+ 125^\circ\text{C}$, DAC0808C, DAC0807C, DAC0806C, $T_A = 0^\circ\text{C}$ to $+ 75^\circ\text{C}$, and all digital inputs at high logic level unless otherwise noted.)

Symbol	Parameter	Conditions	Min	Typ	Max	Units
E_r	Relative Accuracy (Error Relative to Full Scale I_O) DAC0808L (LM1508-8), DAC0808LC (LM1408-8) DAC0807LC (LM1408-7), (Note 5) DAC0806LC (LM1408-6), (Note 5) Settling Time to Within $1/2$ LSB (Includes t_{PLH})	(Figure 4) $T_A = 25^\circ\text{C}$ (Note 6), (Figure 5)			± 0.19 ± 0.39 ± 0.78	% % %
t_{PLH}, t_{PHL}	Propagation Delay Time	$T_A = 25^\circ\text{C}$, (Figure 5)		30	100	ns
T_{CIO}	Output Full Scale Current Drift			± 20		ppm/ $^\circ\text{C}$
MSB V_{IH} V_{IL}	Digital Input Logic Levels High Level, Logic "1" Low Level, Logic "0"	(Figure 3)	2		0.8	V _{DC} V _{DC}
MSB	Digital Input Current High Level Low Level	(Figure 3) $V_{IH} = 5\text{V}$ $V_{IL} = 0.8\text{V}$		0 - 0.003	0.040 - 0.8	mA mA
I_{15}	Reference Input Bias Current	(Figure 3)		- 1	- 3	μA
	Output Current Range	(Figure 3) $V_{EE} = - 5\text{V}$ $V_{EE} = - 15\text{V}$, $T_A = 25^\circ\text{C}$	0 0	2.0 2.0	2.1 4.2	mA mA
I_O	Output Current	$V_{REF} = 2.000\text{V}$, $R_{14} = 1000\Omega$, (Figure 3)	1.9	1.99	2.1	mA
	Output Current, All Bits Low	(Figure 3)		0	4	μA
	Output Voltage Compliance (Note 2) $V_{EE} = - 5\text{V}$, $I_{REF} = 1\text{ mA}$ V_{EE} Below - 10V	$E_r \leq 0.19\%$, $T_A = 25^\circ\text{C}$			- 0.55, + 0.4 - 5.0, + 0.4	V _{DC} V _{DC}

DAC0808 is compatible with the 1408.

Electrical Characteristics (Continued)

($V_{CC} = 5V$, $V_{EE} = -15V_{DC}$, $V_{REF}/R14 = 2mA$, DAC0808: $T_A = -55^\circ C$ to $+125^\circ C$, DAC0808C, DAC0807C, DAC0806C, $T_A = 0^\circ C$ to $+75^\circ C$, and all digital inputs at high logic level unless otherwise noted.)

Symbol	Parameter	Conditions	Min	Typ	Max	Units
SRIREF	Reference Current Slew Rate	(Figure 6)	4	8		mA/ μ s
	Output Current Power Supply Sensitivity	$-5V \leq V_{EE} \leq -16.5V$		0.05	2.7	$\mu A/V$
I _{CC} I _{EE}	Power Supply Current (All Bits Low)	(Figure 3)		2.3 -4.3	22 -13	mA mA
V _{CC} V _{EE}	Power Supply Voltage Range	$T_A = 25^\circ C$, (Figure 3)	4.5 -4.5	5.0 -15	5.5 -16.5	V _{DC} V _{DC}
	Power Dissipation All Bits Low	$V_{CC} = 5V$, $V_{EE} = -5V$		33	170	mW
	All Bits High	$V_{CC} = 5V$, $V_{EE} = -15V$		106	305	mW
		$V_{CC} = 15V$, $V_{EE} = -5V$		90		mW
		$V_{CC} = 15V$, $V_{EE} = -15V$		160		mW

Note 1: Absolute Maximum Ratings indicate limits beyond which damage to the device may occur. DC and AC electrical specifications do not apply when operating the device beyond its specified operating conditions.

Note 2: Range control is not required.

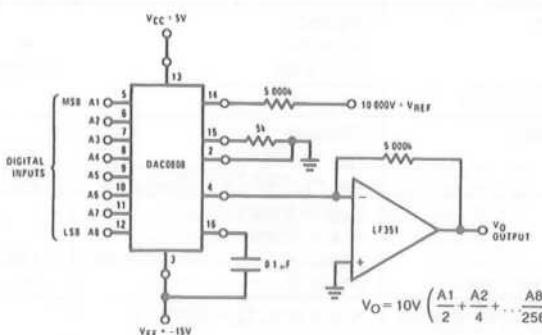
Note 3: The maximum power dissipation must be derated at elevated temperatures and is dictated by T_{JMAX} , θ_{JA} , and the ambient temperature, T_A . The maximum allowable power dissipation at any temperature is $P_D = (T_{JMAX} - T_A)/\theta_{JA}$ or the number given in the Absolute Maximum Ratings, whichever is lower. For this device, $T_{JMAX} = 125^\circ C$, and the typical junction-to-ambient thermal resistance of the dual-in-line J package when the board mounted is $100^\circ C/W$. For the dual-in-line N package, this number increases to $175^\circ C/W$ and for the small outline M package this number is $100^\circ C/W$.

Note 4: Human body model, 100 pF discharged through a 1.5 k Ω resistor.

Note 5: All current switches are tested to guarantee at least 50% of rated current.

Note 6: All bits switched.

Note 7: Pin-out numbers for the DAL080X represent the dual-in-line package. The small outline package pinout differs from the dual-in-line package.

Typical Application

TL/H/5687-3

FIGURE 1. +10V Output Digital to Analog Converter (Note 7)

DAC0808 is compatible with the 1408.



National
Semiconductor
Corporation

ADC0801, ADC0802, ADC0803, ADC0804, ADC0805 8-Bit μ P Compatible A/D Converters

General Description

The ADC0801, ADC0802, ADC0803, ADC0804 and ADC0805 are CMOS 8-bit successive approximation A/D converters that use a differential potentiometric ladder—similar to the 256R products. These converters are designed to allow operation with the NSC800 and INS8080A derivative control bus with TRI-STATE® output latches directly driving the data bus. These A/Ds appear like memory locations or I/O ports to the microprocessor and no interfacing logic is needed.

Differential analog voltage inputs allow increasing the common-mode rejection and offsetting the analog zero input voltage value. In addition, the voltage reference input can be adjusted to allow encoding any smaller analog voltage span to the full 8 bits of resolution.

Features

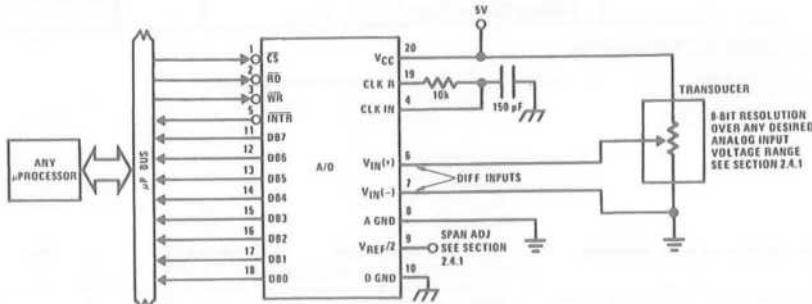
- Compatible with 8080 µP derivatives—no interfacing logic needed - access time - 135 ns
 - Easy interface to all microprocessors, or operates "stand alone"

- Differential analog voltage inputs
- Logic inputs and outputs meet both MOS and TTL voltage level specifications
- Works with 2.5V (LM336) voltage reference
- On-chip clock generator
- 0V to 5V analog input voltage range with single 5V supply
- No zero adjust required
- 0.3" standard width 20-pin DIP package
- 20-pin molded chip carrier or small outline package
- Operates ratiometrically or with 5 V_{DC}, 2.5 V_{DC}, or analog span adjusted voltage reference

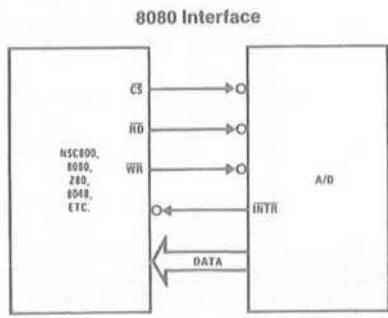
Key Specifications

■ Resolution	8 bits
■ Total error	$\pm \frac{1}{4}$ LSB, $\pm \frac{1}{2}$ LSB and ± 1 LSB
■ Conversion time	100 μ s

Typical Applications



TL/H/5671-1



TH/H/5671-31

Error Specification (Includes Full-Scale, Zero Error, and Non-Linearity)

Zero Error, and Non-Linearity			
Part Number	Full-Scale Adjusted	V _{REF} /2 = 2.500 V _D C (No Adjustments)	V _{REF} /2 = No Connection (No Adjustments)
ADC0801	± 1/4 LSB		
ADC0802		± 1/2 LSB	
ADC0803	± 1/2 LSB		
ADC0804		± 1 LSB	
ADC0805			± 1 LSB

Absolute Maximum Ratings (Notes 1 & 2)

If Military/Aerospace specified devices are required, contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Supply Voltage (V_{CC}) (Note 3)	6.5V
Voltage	
Logic Control Inputs	-0.3V to +18V
All Other Input and Outputs	-0.3V to (V_{CC} + 0.3V)
Lead Temp. (Soldering, 10 seconds)	
Dual-In-Line Package (plastic)	260°C
Dual-In-Line Package (ceramic)	300°C
Surface Mount Package	
Vapor Phase (60 seconds)	215°C
Infrared (15 seconds)	220°C

Storage Temperature Range	-65°C to +150°C
Package Dissipation at $T_A = 25^\circ\text{C}$	875 mW
ESD Susceptibility (Note 10)	800V
Temperature Range	$T_{MIN} \leq T_A \leq T_{MAX}$
ADC0801/02LJ	-55°C $\leq T_A \leq +125^\circ\text{C}$
ADC0801/02/03/04LCJ	-40°C $\leq T_A \leq +85^\circ\text{C}$
ADC0801/02/03/05LCN	-40°C $\leq T_A \leq +85^\circ\text{C}$
ADC0804LCN	0°C $\leq T_A \leq +70^\circ\text{C}$
ADC0802/03/04LCV	0°C $\leq T_A \leq +70^\circ\text{C}$
ADC0802/03/04LCWM	0°C $\leq T_A \leq +70^\circ\text{C}$
Range of V_{CC}	4.5 V _{DC} to 6.3 V _{DC}

Electrical Characteristics

The following specifications apply for $V_{CC} = 5$ V_{DC}, $T_{MIN} \leq T_A \leq T_{MAX}$ and $f_{CLK} = 640$ kHz unless otherwise specified.

Parameter	Conditions	Min	Typ	Max	Units
ADC0801: Total Adjusted Error (Note 8)	With Full-Scale Adj. (See Section 2.5.2)			$\pm \frac{1}{4}$	LSB
ADC0802: Total Unadjusted Error (Note 8)	$V_{REF}/2 = 2.500$ V _{DC}			$\pm \frac{1}{2}$	LSB
ADC0803: Total Adjusted Error (Note 8)	With Full-Scale Adj. (See Section 2.5.2)			$\pm \frac{1}{2}$	LSB
ADC0804: Total Unadjusted Error (Note 8)	$V_{REF}/2 = 2.500$ V _{DC}			± 1	LSB
ADC0805: Total Unadjusted Error (Note 8)	$V_{REF}/2$ -No Connection			± 1	LSB
$V_{REF}/2$ Input Resistance (Pin 9)	ADC0801/02/03/05 ADC0804 (Note 9)	2.5 0.75	8.0 1.1		k Ω k Ω
Analog Input Voltage Range	(Note 4) V(+) or V(-)	Gnd-0.05		$V_{CC} + 0.05$	V _{DC}
DC Common-Mode Error	Over Analog Input Voltage Range		$\pm \frac{1}{16}$	$\pm \frac{1}{8}$	LSB
Power Supply Sensitivity	$V_{CC} = 5$ V _{DC} $\pm 10\%$ Over Allowed $V_{IN}(+)$ and $V_{IN}(-)$ Voltage Range (Note 4)		$\pm \frac{1}{16}$	$\pm \frac{1}{8}$	LSB

AC Electrical Characteristics

The following specifications apply for $V_{CC} = 5$ V_{DC} and $T_A = 25^\circ\text{C}$ unless otherwise specified.

Symbol	Parameter	Conditions	Min	Typ	Max	Units
T_C	Conversion Time	$f_{CLK} = 640$ kHz (Note 6)	103		114	μs
T_C	Conversion Time	(Note 5, 6)	66		73	$1/f_{CLK}$
f_{CLK}	Clock Frequency Clock Duty Cycle	$V_{CC} = 5$ V, (Note 5) (Note 5)	100 40	640	1460 60	kHz %
CR	Conversion Rate in Free-Running Mode	INTR tied to WR with $\bar{CS} = 0$ V _{DC} , $f_{CLK} = 640$ kHz	8770		9708	conv/s
$t_{W(WR)L}$	Width of WR Input (Start Pulse Width)	$\bar{CS} = 0$ V _{DC} (Note 7)	100			ns
t_{ACC}	Access Time (Delay from Falling Edge of RD to Output Data Valid)	$C_L = 100$ pF		135	200	ns
t_{1H}, t_{0H}	TRI-STATE Control (Delay from Rising Edge of RD to Hi-Z State)	$C_L = 10$ pF, $R_L = 10\text{k}$ (See TRI-STATE Test Circuits)		125	200	ns
t_{WI}, t_{RI}	Delay from Falling Edge of WR or RD to Reset of INTR			300	450	ns
C_{IN}	Input Capacitance of Logic Control Inputs			5	7.5	pF
C_{OUT}	TRI-STATE Output Capacitance (Data Buffers)			5	7.5	pF
CONTROL INPUTS [Note: CLK IN (Pin 4) is the input of a Schmitt trigger circuit and is therefore specified separately]						
V_{IN} (1)	Logical "1" Input Voltage (Except Pin 4 CLK IN)	$V_{CC} = 5.25$ V _{DC}	2.0		15	V _{DC}

AC Electrical Characteristics (Continued)

The following specifications apply for $V_{CC} = 5V_{DC}$ and $T_{MIN} \leq T_A \leq T_{MAX}$, unless otherwise specified.

Symbol	Parameter	Conditions	Min	Typ	Max	Units
CONTROL INPUTS [Note: CLK IN (Pin 4) is the input of a Schmitt trigger circuit and is therefore specified separately]						
$V_{IN(0)}$	Logical "0" Input Voltage (Except Pin 4 CLK IN)	$V_{CC} = 4.75 V_{DC}$			0.8	V_{DC}
$I_{IN(1)}$	Logical "1" Input Current (All Inputs)	$V_{IN} = 5 V_{DC}$		0.005	1	μA_{DC}
$I_{IN(0)}$	Logical "0" Input Current (All Inputs)	$V_{IN} = 0 V_{DC}$	-1	-0.005		μA_{DC}
CLOCK IN AND CLOCK R						
V_{T+}	CLK IN (Pin 4) Positive Going Threshold Voltage		2.7	3.1	3.5	V_{DC}
V_{T-}	CLK IN (Pin 4) Negative Going Threshold Voltage		1.5	1.8	2.1	V_{DC}
V_H	CLK IN (Pin 4) Hysteresis $(V_{T+}) - (V_{T-})$		0.6	1.3	2.0	V_{DC}
$V_{OUT(0)}$	Logical "0" CLK R Output Voltage	$I_O = 360 \mu A$ $V_{CC} = 4.75 V_{DC}$			0.4	V_{DC}
$V_{OUT(1)}$	Logical "1" CLK R Output Voltage	$I_O = -360 \mu A$ $V_{CC} = 4.75 V_{DC}$	2.4			V_{DC}
DATA OUTPUTS AND INTR						
$V_{OUT(0)}$	Logical "0" Output Voltage Data Outputs INTR Output	$I_{OUT} = 1.6 \text{ mA}, V_{CC} = 4.75 V_{DC}$ $I_{OUT} = 1.0 \text{ mA}, V_{CC} = 4.75 V_{DC}$			0.4 0.4	V_{DC} V_{DC}
$V_{OUT(1)}$	Logical "1" Output Voltage	$I_O = -360 \mu A, V_{CC} = 4.75 V_{DC}$	2.4			V_{DC}
$V_{OUT(1)}$	Logical "1" Output Voltage	$I_O = -10 \mu A, V_{CC} = 4.75 V_{DC}$	4.5			V_{DC}
I_{OUT}	TRI-STATE Disabled Output Leakage (All Data Buffers)	$V_{OUT} = 0 V_{DC}$ $V_{OUT} = 5 V_{DC}$	-3		3	μA_{DC} μA_{DC}
I_{SOURCE}		V_{OUT} Short to Gnd, $T_A = 25^\circ C$	4.5	6		mA_{DC}
I_{SINK}		V_{OUT} Short to V_{CC} , $T_A = 25^\circ C$	9.0	16		mA_{DC}
POWER SUPPLY						
I_{CC}	Supply Current (Includes Ladder Current)	$f_{CLK} = 640 \text{ kHz}$, $V_{REF}/2 = NC$, $T_A = 25^\circ C$ and $CS = 5V$			1.1 1.9	1.8 2.5 mA
ADC0801/02/03/04LCJ/05 ADC0804LCN/LCV/LCWM						
Note 1: Absolute Maximum Ratings indicate limits beyond which damage to the device may occur. DC and AC electrical specifications do not apply when operating the device beyond its specified operating conditions.						
Note 2: All voltages are measured with respect to Gnd, unless otherwise specified. The separate A Gnd point should always be wired to the D Gnd.						
Note 3: A zener diode exists, internally, from V_{CC} to Gnd and has a typical breakdown voltage of $7 V_{DC}$.						
Note 4: For $V_{IN(-)} \geq V_{IN(+)}$ the digital output code will be 0000 0000. Two on-chip diodes are tied to each analog input (see block diagram) which will forward conduct for analog input voltages one diode drop below ground or one diode drop greater than the V_{CC} supply. Be careful, during testing at low V_{CC} levels (4.5V), as high level analog inputs (5V) can cause this input diode to conduct—especially at elevated temperatures, and cause errors for analog inputs near full-scale. The spec allows 50 mV forward bias of either diode. This means that as long as the analog V_{IN} does not exceed the supply voltage by more than 50 mV, the output code will be correct. To achieve an absolute 0 V_{DC} to 5 V_{DC} input voltage range will therefore require a minimum supply voltage of 4.950 V_{DC} over temperature variations, initial tolerance and loading.						
Note 5: Accuracy is guaranteed at $f_{CLK} = 640 \text{ kHz}$. At higher clock frequencies accuracy can degrade. For lower clock frequencies, the duty cycle limits can be extended so long as the minimum clock high time interval or minimum clock low time interval is no less than 275 ns.						
Note 6: With an asynchronous start pulse, up to 8 clock periods may be required before the internal clock phases are proper to start the conversion process. The start request is internally latched, see Figure 2 and section 2.0.						
Note 7: The CS input is assumed to bracket the WR strobe input and therefore timing is dependent on the WR pulse width. An arbitrarily wide pulse width will hold the converter in a reset mode and the start of conversion is initiated by the low to high transition of the WR pulse (see timing diagrams).						
Note 8: None of these A/Ds requires a zero adjust (see section 2.5.1). To obtain zero code at other analog input voltages see section 2.5 and Figure 5.						
Note 9: The $V_{REF}/2$ pin is the center point of a two resistor divider connected from V_{CC} to ground. Each resistor is 2.2k, except for the ADC0804LCJ where each resistor is 16k. Total ladder input resistance is the sum of the two equal resistors.						
Note 10: Human body model, 100 pF discharged through a 1.5 k Ω resistor.						

LM135/LM235/LM335, LM135A/LM235A/LM335A Precision Temperature Sensors

General Description

The LM135 series are precision, easily-calibrated, integrated circuit temperature sensors. Operating as a 2-terminal zener, the LM135 has a breakdown voltage directly proportional to absolute temperature at $+10 \text{ mV/K}$. With less than 1Ω dynamic impedance the device operates over a current range of $400 \mu\text{A}$ to 5 mA with virtually no change in performance. When calibrated at 25°C the LM135 has typically less than 1°C error over a 100°C temperature range. Unlike other sensors the LM135 has a linear output.

Applications for the LM135 include almost any type of temperature sensing over a -55°C to $+150^\circ\text{C}$ temperature range. The low impedance and linear output make interfacing to readout or control circuitry especially easy.

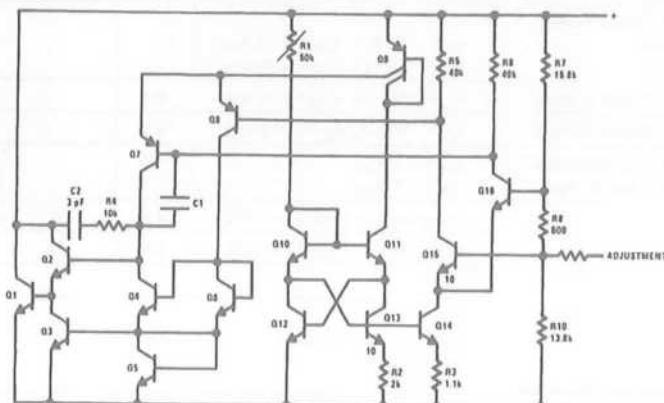
The LM135 operates over a -55°C to $+150^\circ\text{C}$ temperature range while the LM235 operates over a -40°C to $+125^\circ\text{C}$

temperature range. The LM335 operates from -40°C to $+100^\circ\text{C}$. The LM135/LM235/LM335 are available packaged in hermetic TO-46 transistor packages while the LM335 is also available in plastic TO-92 packages.

Features

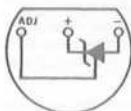
- Directly calibrated in "Kelvin"
- 1°C initial accuracy available
- Operates from $400 \mu\text{A}$ to 5 mA
- Less than 1Ω dynamic impedance
- Easily calibrated
- Wide operating temperature range
- 200°C overrange
- Low cost

Schematic Diagram



TL/H/5698-1

Connection Diagrams

 TO-92
Plastic Package


Bottom View

Order Number LM335Z or LM335AZ
See NS Package Number Z03A

 SO-8
Surface Mount Package


TL/H/5698-25
Order Number LM335M or
LM335AM
See NS Package Number M08A

 TO-46
Metal Can Package*


Bottom View

*Case is connected to negative pin
Order Number LM135H, LM235H,
LM335H, LM135AH,
LM235AH or LM335AH
See NS Package Number H03H

Absolute Maximum Ratings

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications. (Note 4)

		Specified Operating Temp. Range			Intermittent (Note 2)
			Continuous		150°C to 200°C
Reverse Current	15 mA	LM135, LM135A	-55°C to +150°C		125°C to 150°C
Forward Current	10 mA	LM235, LM235A	-40°C to +125°C		100°C to 125°C
Storage Temperature		LM335, LM335A	-40°C to +100°C		
TO-46 Package	-60°C to +180°C		Lead Temp. (Soldering, 10 seconds)		260°C
TO-92 Package	-60°C to +150°C	TO-92 Package:			300°C
SO-8 Package	-65°C to +150°C	TO-46 Package:			300°C
		SO-8 Package:	Vapor Phase (60 seconds)		215°C
			Infrared (15 seconds)		220°C

Temperature Accuracy LM135/LM235, LM135A/LM235A (Note 1)

Parameter	Conditions	LM135A/LM235A			LM135/LM235			Units
		Min	Typ	Max	Min	Typ	Max	
Operating Output Voltage	T _C = 25°C, I _R = 1 mA	2.97	2.98	2.99	2.95	2.98	3.01	V
Uncalibrated Temperature Error	T _C = 25°C, I _R = 1 mA		0.5	1		1	3	°C
Uncalibrated Temperature Error	T _{MIN} ≤ T _C ≤ T _{MAX} , I _R = 1 mA		1.3	2.7		2	5	°C
Temperature Error with 25°C Calibration	T _{MIN} ≤ T _C ≤ T _{MAX} , I _R = 1 mA		0.3	1		0.5	1.5	°C
Calibrated Error at Extended Temperatures	T _C = T _{MAX} (Intermittent)		2			2		°C
Non-Linearity	I _R = 1 mA		0.3	0.5		0.3	1	°C

Temperature Accuracy LM335, LM335A (Note 1)

Parameter	Conditions	LM335A			LM335			Units
		Min	Typ	Max	Min	Typ	Max	
Operating Output Voltage	T _C = 25°C, I _R = 1 mA	2.95	2.98	3.01	2.92	2.98	3.04	V
Uncalibrated Temperature Error	T _C = 25°C, I _R = 1 mA		1	3		2	6	°C
Uncalibrated Temperature Error	T _{MIN} ≤ T _C ≤ T _{MAX} , I _R = 1 mA		2	5		4	9	°C
Temperature Error with 25°C Calibration	T _{MIN} ≤ T _C ≤ T _{MAX} , I _R = 1 mA		0.5	1		1	2	°C
Calibrated Error at Extended Temperatures	T _C = T _{MAX} (Intermittent)		2			2		°C
Non-Linearity	I _R = 1 mA		0.3	1.5		0.3	1.5	°C

Electrical Characteristics (Note 1)

Parameter	Conditions	LM135/LM235 LM135A/LM235A			LM335 LM335A			Units
		Min	Typ	Max	Min	Typ	Max	
Operating Output Voltage Change with Current	400 μA ≤ I _R ≤ 5 mA At Constant Temperature		2.5	10		3	14	mV
Dynamic Impedance	I _R = 1 mA		0.5			0.6		Ω
Output Voltage Temperature Coefficient			+ 10			+ 10		mV/°C
Time Constant	Still Air 100 ft/Min Air Stirred Oil		80			80		sec
			10			10		sec
			1			1		sec
Time Stability	T _C = 125°C		0.2			0.2		°C/khr

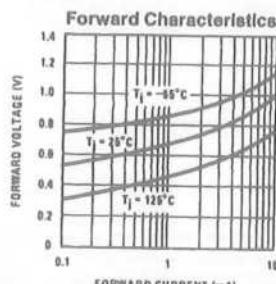
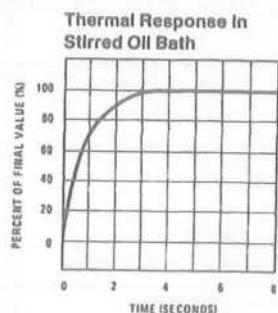
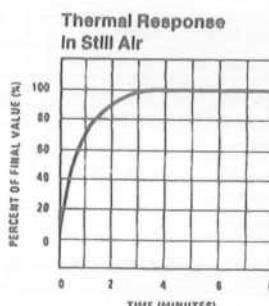
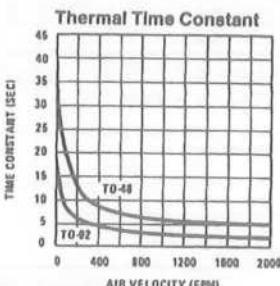
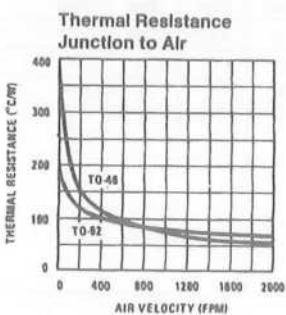
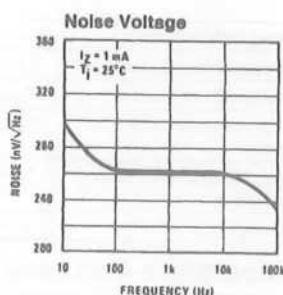
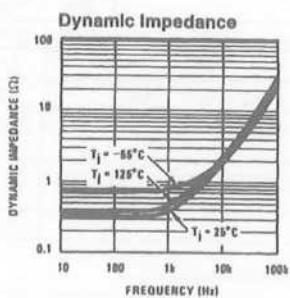
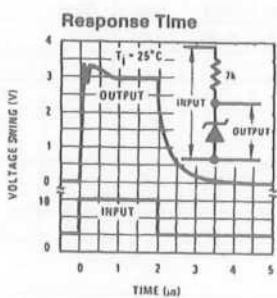
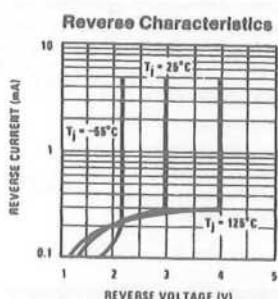
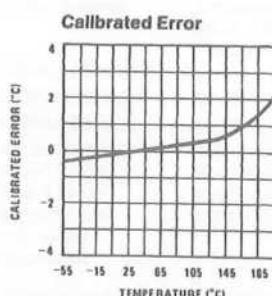
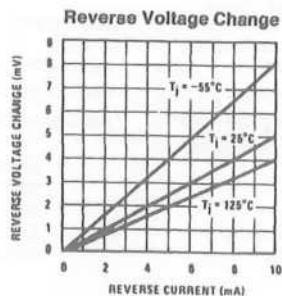
Note 1: Accuracy measurements are made in a well-stirred oil bath. For other conditions, self heating must be considered.

Note 2: Continuous operation at these temperatures for 10,000 hours for H package and 5,000 hours for Z package may decrease life expectancy of the device.

Note 3: Thermal Resistance TO-92 TO-46 SO-8
 θ_{JA} (junction to ambient) 202°C/W 400°C/W 165°C/W
 θ_{JC} (junction to case) 170°C/W N/A N/A

Note 4: Refer to RETS135H for military specifications.

Typical Performance Characteristics



Application Hints

CALIBRATING THE LM135

Included on the LM135 chip is an easy method of calibrating the device for higher accuracies. A pot connected across the LM135 with the arm tied to the adjustment terminal allows a 1-point calibration of the sensor that corrects for inaccuracy over the full temperature range.

This single point calibration works because the output of the LM135 is proportional to absolute temperature with the extrapolated output of sensor going to 0V output at 0K (-273.15°C). Errors in output voltage versus temperature are only slope (or scale factor) errors so a slope calibration at one temperature corrects at all temperatures.

The output of the device (calibrated or uncalibrated) can be expressed as:

$$V_{\text{OUT}_T} = V_{\text{OUT}_{T_0}} \times \frac{T}{T_0}$$

where T is the unknown temperature and T_0 is a reference temperature, both expressed in degrees Kelvin. By calibrating the output to read correctly at one temperature the output at all temperatures is correct. Nominally the output is calibrated at 10 mV/ $^{\circ}\text{K}$.

To insure good sensing accuracy several precautions must be taken. Like any temperature sensing device, self heating can reduce accuracy. The LM135 should be operated at the lowest current suitable for the application. Sufficient current, of course, must be available to drive both the sensor and the calibration pot at the maximum operating temperature as well as any external loads.

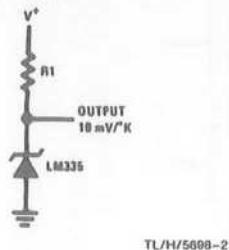
If the sensor is used in an ambient where the thermal resistance is constant, self heating errors can be calibrated out. This is possible if the device is run with a temperature stable current. Heating will then be proportional to zener voltage and therefore temperature. This makes the self heating error proportional to absolute temperature the same as scale factor errors.

WATERPROOFING SENSORS

Meltable inner core heat shrinkable tubing such as manufactured by Raychem can be used to make low-cost waterproof sensors. The LM335 is inserted into the tubing about $\frac{1}{2}$ " from the end and the tubing heated above the melting point of the core. The unfilled $\frac{1}{2}$ " end melts and provides a seal over the device.

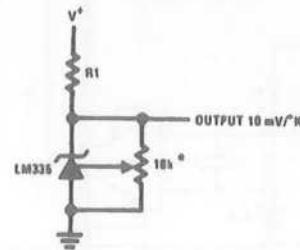
Typical Applications

Basic Temperature Sensor



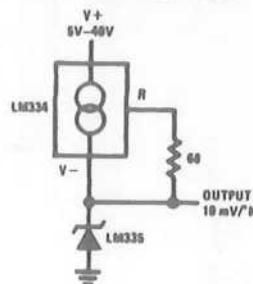
TL/H/5698-2

Calibrated Sensor



TL/H/5698-9

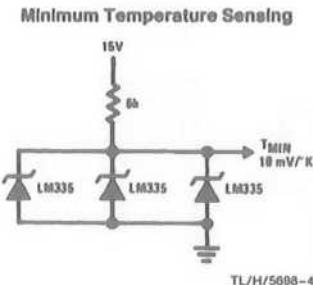
Wide Operating Supply



TL/H/5698-10

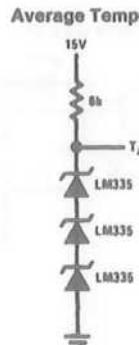
*Calibrate for 2.982V at 25°C

15V
10k
LM335
LM335
LM335



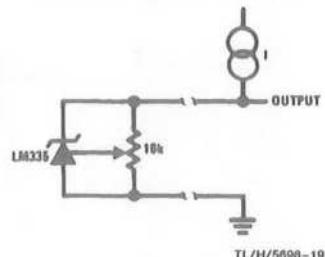
TL/H/5698-4

15V
10k
LM335
LM335
LM335



TL/H/5698-18

15V
10k
LM335
LM335
LM335



TL/H/5698-19

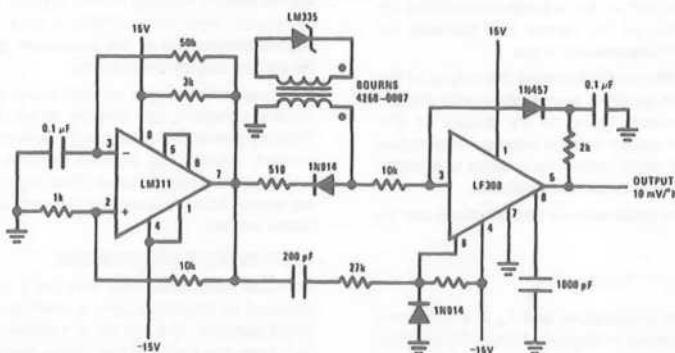
Wire length for 1°C error due to wire drop

AWG	$I_R = 1 \text{ mA}$		$I_R = 0.5 \text{ mA}$	
	FEET	FEET	FEET	FEET
14	4000		8000	
16	2500		5000	
18	1600		3200	
20	1000		2000	
22	625		1250	
24	400		800	

*For $I_R = 0.5 \text{ mA}$, the trim pot must be deleted.

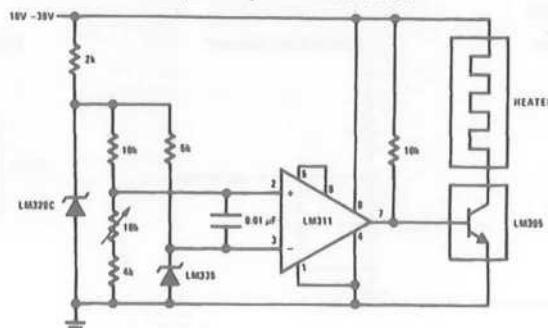
Typical Applications (Continued)

Isolated Temperature Sensor



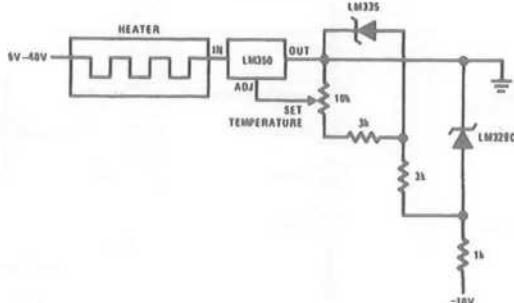
TL/H/5698-20

Simple Temperature Controller



TL/H/5698-5

Simple Temperature Control



TL/H/5698-21



8237A/8237A-4/8237A-5 HIGH PERFORMANCE PROGRAMMABLE DMA CONTROLLER

- Enable/Disable Control of Individual DMA Requests
- Four Independent DMA Channels
- Independent Autoinitialization of all Channels
- Memory-to-Memory Transfers
- Memory Block Initialization
- Address Increment or Decrement
- High performance: Transfers up to 1.6M Bytes/Second with 5 MHz 8237A-5
- Directly Expandable to any Number of Channels
- End of Process Input for Terminating Transfers
- Software DMA Requests
- Independent Polarity Control for DREQ and DACK Signals
- Available In EXPRESS
- Standard Temperature Range

The 8237A Multimode Direct Memory Access (DMA) Controller is a peripheral Interface circuit for microprocessor systems. It is designed to improve system performance by allowing external devices to directly transfer information from the system memory. Memory-to-memory transfer capability is also provided. The 8237A offers a wide variety of programmable control features to enhance data throughput and system optimization and to allow dynamic reconfiguration under program control.

The 8237A is designed to be used in conjunction with an external 8-bit address register such as the 8282. It contains four independent channels and may be expanded to any number of channels by cascading additional controller chips.

The three basic transfer modes allow programmability of the types of DMA service by the user. Each channel can be individually programmed to Autoinitialize to its original condition following an End of Process (EOP).

Each channel has a full 64K address and word count capability.

The 8237A-4 and 8237A-5 are 4 MHz and 5 MHz selected versions of the standard 3 MHz 8237A respectively.

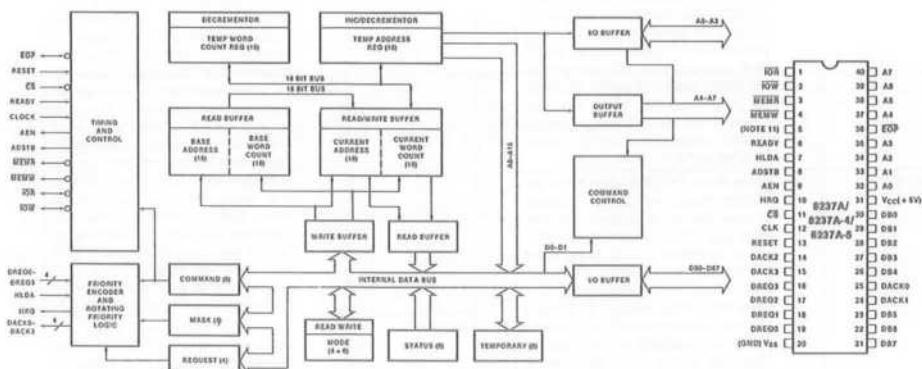


Figure 1. Block Diagram

Figure 2.
Pin Configuration



8237A/8237A-4/8237A-5

Table 1. Pin Description

Symbol	Type	Name and Function	Symbol	Type	Name and Function
V _{CC}		Power: +5 volt supply.			
V _{SS}		Ground: Ground.			
CLK	I	Clock Input: Clock Input controls the Internal operations of the 8237A and its rate of data transfers. The input may be driven at up to 3 MHz for the standard 8237A and up to 5 MHz for the 8237A-5.			Memory-to-Memory operations, data from the memory comes into the 8237A on the data bus during the read-from-memory transfer. In the write-to-memory transfer, the data bus outputs place the data into the new memory location.
CS	I	Chip Select: Chip Select is an active low input used to select the 8237A as an I/O device during the Idle cycle. This allows CPU communication on the data bus.	IOR	I/O	I/O Read: I/O Read is a bidirectional active low three-state line. In the Idle cycle, it is an input control signal used by the CPU to read the control registers. In the Active cycle, it is an output control signal used by the 8237A to access data from a peripheral during a DMA Write transfer.
RESET	I	Reset: Reset is an active high input which clears the Command, Status, Request and Temporary registers. It also clears the first/last flip/flop and sets the Mask register. Following a Reset the device is in the Idle cycle.	IOW	I/O	I/O Write: I/O Write is a bidirectional active low three-state line. In the Idle cycle, it is an input control signal used by the CPU to load information into the 8237A. In the Active cycle, it is an output control signal used by the 8237A to load data to the peripheral during a DMA Read transfer.
READY	I	Ready: Ready is an input used to extend the memory read and write pulses from the 8237A to accommodate slow memories or I/O peripheral devices. Ready must not make transitions during its specified setup/hold time.	EOP	I/O	End of Process: End of Process is an active low bidirectional signal. Information concerning the completion of DMA services is available at the bidirectional EOP pin. The 8237A allows an external signal to terminate an active DMA service. This is accomplished by pulling the EOP input low with an external EOP signal. The 8237A also generates a pulse when the terminal count (TC) for any channel is reached. This generates an EOP signal which is output through the EOP Line. The reception of EOP, either internal or external, will cause the 8237A to terminate the service, reset the request, and, if AutoInitialize is enabled, to write the base registers to the current registers of that channel. The mask bit and TC bit in the status word will be set for the currently active channel by EOP unless the channel is programmed for AutoInitialize. In that case, the mask bit remains clear. During memory-to-memory transfers, EOP will be output when the TC for channel 1 occurs. EOP should be tied high with a pull-up resistor if it is not used to prevent erroneous end of process inputs.
HLDA	I	Hold Acknowledge: The active high Hold Acknowledge from the CPU indicates that it has relinquished control of the system busses.	A0-A3	I/O	Address: The four least significant address lines are bidirectional three-state signals. In the Idle cycle they are inputs and are used by the 8237A to address the control register to be loaded or read. In the Active cycle they are outputs and provide the lower 4 bits of the output address.
DREQ0-DREQ3	I	DMA Request: The DMA Request lines are individual asynchronous channel request inputs used by peripheral circuits to obtain DMA service. In fixed Priority, DREQ0 has the highest priority and DREQ3 has the lowest priority. A request is generated by activating the DREQ line of a channel. DACK will acknowledge the recognition of DREQ signal. Polarity of DREQ is programmable. Reset initializes these lines to active high. DREQ must be maintained until the corresponding DACK goes active.			
DB0-DB7	I/O	Data Bus: The Data Bus lines are bidirectional three-state signals connected to the system data bus. The outputs are enabled in the Program condition during the I/O Read to output the contents of an Address register, a Status register, the Temporary register or a Word Count register to the CPU. The outputs are disabled and the inputs are read during an I/O Write cycle when the CPU is programming the 8237A control registers. During DMA cycles the most significant 8 bits of the address are output onto the data bus to be strobed into an external latch by ADSTB. In mem-			

Table 1. Pin Description (Continued)

Symbol	Type	Name and Function	Symbol	Type	Name and Function
A4-A7	O	Address: The four most significant address lines are three-state outputs and provide 4 bits of address. These lines are enabled only during the DMA service.	AEN	O	Address Enable: Address Enable enables the 8-bit latch containing the upper 8 address bits onto the system address bus. AEN can also be used to disable other system bus drivers during DMA transfers. AEN is active HIGH.
HRQ	O	Hold Request: This is the Hold Request to the CPU and is used to request control of the system bus. If the corresponding mask bit is clear, the presence of any valid DREQ causes 8237A to issue the HRQ. After HRQ goes active at least one clock cycle (TCY) must occur before HLDA goes active.	ADSTB	O	Address Strobe: The active high, Address Strobe is used to strobe the upper address byte into an external latch.
DACK0-DACK3	O	DMA Acknowledge: DMA Acknowledge is used to notify the individual peripherals when one has been granted a DMA cycle. The sense of these lines is programmable. Reset initializes them to active low.	MEMR	O	Memory Read: The Memory Read signal is an active low three-state output used to access data from the selected memory location during a DMA Read or a memory-to-memory transfer.
			MEMW	O	Memory Write: The Memory Write is an active low three-state output used to write data to the selected memory location during a DMA Write or a memory-to-memory transfer.

FUNCTIONAL DESCRIPTION

The 8237A block diagram includes the major logic blocks and all of the internal registers. The data interconnection paths are also shown. Not shown are the various control signals between the blocks. The 8237A contains 344 bits of internal memory in the form of registers. Figure 3 lists these registers by name and shows the size of each. A detailed description of the registers and their functions can be found under Register Description.

Name	Size	Number
Base Address Registers	16 bits	4
Base Word Count Registers	16 bits	4
Current Address Registers	16 bits	4
Current Word Count Registers	16 bits	4
Temporary Address Register	16 bits	1
Temporary Word Count Register	16 bits	1
Status Register	8 bits	1
Command Register	8 bits	1
Temporary Register	8 bits	1
Mode Registers	6 bits	4
Mask Register	4 bits	1
Request Register	4 bits	1

Figure 3. 8237A Internal Registers

The 8237A contains three basic blocks of control logic. The Timing Control block generates internal timing and external control signals for the 8237A. The Program Command Control block decodes the various commands given to the 8237A by the microprocessor prior to servicing a DMA Request. It also decodes the Mode Control word used to select the type of DMA during the servicing. The Priority Encoder block resolves priority contention between DMA channels requesting service simultaneously.

The Timing Control block derives internal timing from the clock input. In 8237A systems this input will usually

be the ϕ_2 TTL clock from an 8224 or CLK from an 8085AH or 8284A. For 8085AH-2 systems above 3.9 MHz, the 8085 CLK(OUT) does not satisfy 8237A-5 clock LOW and HIGH time requirements. In this case, an external clock should be used to drive the 8237A-5.

DMA Operation

The 8237A is designed to operate in two major cycles. These are called Idle and Active cycles. Each device cycle is made up of a number of states. The 8237A can assume seven separate states, each composed of one full clock period. State I (SI) is the inactive state. It is entered when the 8237A has no valid DMA requests pending. While in SI, the DMA controller is inactive but may be in the Program Condition, being programmed by the processor. State S0 (S0) is the first state of a DMA service. The 8237A has requested hold but the processor has not yet returned an acknowledge. The 8237A may still be programmed until it receives HLDA from the CPU. An acknowledge from the CPU will signal that DMA transfers may begin. S1, S2, S3 and S4 are the working states of the DMA service. If more time is needed to complete a transfer than is available with normal timing, wait states (SW) can be inserted between S2 or S3 and S4 by the use of the Ready line on the 8237A. Note that the data is transferred directly from the I/O device to memory (or vice versa) with IOR and MEMW (or MEMR and IOW) being active at the same time. The data is not read into or driven out of the 8237A in I/O-to-memory or memory-to-I/O DMA transfers.

Memory-to-memory transfers require a read-from and a write-to-memory to complete each transfer. The states, which resemble the normal working states, use two digit numbers for identification. Eight states are required for a single transfer. The first four states (S11, S12, S13, S14) are used for the read-from-memory half

and the last four states (S21, S22, S23, S24) for the write-to-memory half of the transfer.

IDLE CYCLE

When no channel is requesting service, the 8237A will enter the Idle cycle and perform "SI" states. In this cycle the 8237A will sample the DREQ lines every clock cycle to determine if any channel is requesting a DMA service. The device will also sample CS, looking for an attempt by the microprocessor to write or read the internal registers of the 8237A. When CS is low and HLDA is low, the 8237A enters the Program Condition. The CPU can now establish, change or inspect the internal definition of the part by reading from or writing to the internal registers. Address lines A0-A3 are inputs to the device and select which registers will be read or written. The IOR and IOW lines are used to select and time reads or writes. Due to the number and size of the internal registers, an internal flip-flop is used to generate an additional bit of address. This bit is used to determine the upper or lower byte of the 16-bit Address and Word Count registers. The flip-flop is reset by Master Clear or Reset. A separate software command can also reset this flip-flop.

Special software commands can be executed by the 8237A in the Program Condition. These commands are decoded as sets of addresses with the CS and IOW. The commands do not make use of the data bus. Instructions include Clear First/Last Flip-Flop and Master Clear.

ACTIVE CYCLE

When the 8237A is in the Idle cycle and a non-masked channel requests a DMA service, the device will output an HRQ to the microprocessor and enter the Active cycle. It is in this cycle that the DMA service will take place, in one of four modes:

Single Transfer Mode — In Single Transfer mode the device is programmed to make one transfer only. The word count will be decremented and the address decremented or incremented following each transfer. When the word count "rolls over" from zero to FFFFH, a Terminal Count (TC) will cause an Autoinitialize if the channel has been programmed to do so.

DREQ must be held active until DACK becomes active in order to be recognized. If DREQ is held active throughout the single transfer, HRQ will go inactive and release the bus to the system. It will again go active and, upon receipt of a new HLDA, another single transfer will be performed. In 8080A, 8085AH, 8088, or 8086 system this will ensure one full machine cycle execution between DMA transfers. Details of timing between the 8237A and other bus control protocols will depend upon the characteristics of the microprocessor involved.

Block Transfer Mode — In Block Transfer mode the device is activated by DREQ to continue making transfers during the service until a TC, caused by word count going to FFFFH, or an external End of Process (EOP) is encountered. DREQ need only be held active until DACK

becomes active. Again, an Autoinitialize will occur at the end of the service if the channel has been programmed for it.

Demand Transfer Mode — In Demand Transfer mode the device is programmed to continue making transfers until a TC or external EOP is encountered or until DREQ goes inactive. Thus transfers may continue until the I/O device has exhausted its data capacity. After the I/O device has had a chance to catch up, the DMA service is re-established by means of a DREQ. During the time between services when the microprocessor is allowed to operate, the intermediate values of address and word count are stored in the 8237A Current Address and Current Word Count registers. Only an EOP can cause an Autoinitialize at the end of the service. EOP is generated either by TC or by an external signal.

Cascade Mode — This mode is used to cascade more than one 8237A together for simple system expansion. The HRQ and HLDA signals from the additional 8237A are connected to the DREQ and DACK signals of a channel of the initial 8237A. This allows the DMA requests of the additional device to propagate through the priority network circuitry of the preceding device. The priority chain is preserved and the new device must wait for its turn to acknowledge requests. Since the cascade channel of the initial 8237A is used only for prioritizing the additional device, it does not output any address or control signals of its own. These could conflict with the outputs of the active channel in the added device. The 8237A will respond to DREQ and DACK but all other outputs except HRQ will be disabled.

Figure 4 shows two additional devices cascaded into an initial device using two of the previous channels. This forms a two level DMA system. More 8237As could be added at the second level by using the remaining channels of the first level. Additional devices can also be added by cascading into the channels of the second level devices, forming a third level.

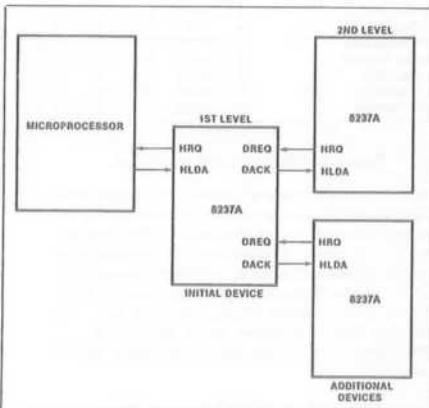


Figure 4. Cascaded 8237As

TRANSFER TYPES

Each of the three active transfer modes can perform three different types of transfers. These are Read, Write and Verify. Write transfers move data from an I/O device to the memory by activating MEMW and IOR. Read transfers move data from memory to an I/O device by activating MEMR and IOW. Verify transfers are pseudo transfers. The 8237A operates as in Read or Write transfers generating addresses, and responding to EOP, etc. However, the memory and I/O control lines all remain inactive. Verify mode is not permitted during memory to memory operation.

Memory-to-Memory — To perform block moves of data from one memory address space to another with a minimum of program effort and time, the 8237A includes a memory-to-memory transfer feature. Programming a bit in the Command register selects channels 0 and 1 to operate as memory-to-memory transfer channels. The transfer is initiated by setting the software DREQ for channel 0. The 8237A requests a DMA service in the normal manner. After HLDA is true, the device, using eight-state transfers in Block Transfer mode, reads data from the memory. The channel 0 Current Address register is the source for the address used and is decremented or incremented in the normal manner. The data byte read from the memory is stored in the 8237A internal Temporary register. Channel 1 then writes the data from the Temporary register to memory using the address in its Current Address register and incrementing or decrementing it in the normal manner. The channel 1 Current Word Count is decremented. When the word count of channel 1 goes to FFFFH, a TC is generated causing an EOP output terminating the service.

Channel 0 may be programmed to retain the same address for all transfers. This allows a single word to be written to a block of memory.

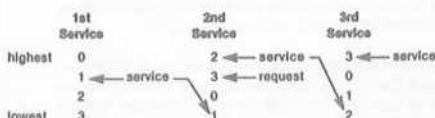
The 8237A will respond to external EOP signals during memory-to-memory transfers. Data comparators in block search schemes may use this input to terminate the service when a match is found. The timing of memory-to-memory transfers is found in Figure 12. Memory-to-memory operations can be detected as an active AEN with no DACK outputs.

AutoInitialize — By programming a bit in the Mode register, a channel may be set up as an AutoInitialize channel. During AutoInitialize initialization, the original values of the Current Address and Current Word Count registers are automatically restored from the Base Address and Base Word count registers of that channel following EOP. The base registers are loaded simultaneously with the current registers by the microprocessor and remain unchanged throughout the DMA service. The mask bit is not set when the channel is in AutoInitialize. Following AutoInitialize the channel is ready to perform another DMA service, without CPU intervention, as soon as a valid DREQ is detected.

Priority — The 8237A has two types of priority encoding available as software selectable options. The first is Fixed Priority which fixes the channels in priority order

based upon the descending value of their number. The channel with the lowest priority is 3 followed by 2, 1 and the highest priority channel, 0. After the recognition of any one channel for service, the other channels are prevented from interfering with that service until it is completed.

The second scheme is Rotating Priority. The last channel to get service becomes the lowest priority channel with the others rotating accordingly.



With Rotating Priority in a single chip DMA system, any device requesting service is guaranteed to be recognized after no more than three higher priority services have occurred. This prevents any one channel from monopolizing the system.

Compressed Timing — In order to achieve even greater throughput where system characteristics permit, the 8237A can compress the transfer time to two clock cycles. From Figure 11 it can be seen that state S3 is used to extend the access time of the read pulse. By removing state S3, the read pulse width is made equal to the write pulse width and a transfer consists only of state S2 to change the address and state S4 to perform the read/write. S1 states will still occur when A8-A15 need updating (see Address Generation). Timing for compressed transfers is found in Figure 14.

Address Generation — In order to reduce pin count, the 8237A multiplexes the eight higher order address bits on the data lines. State S1 is used to output the higher order address bits to an external latch from which they may be placed on the address bus. The falling edge of Address Strobe (ADSTB) is used to load these bits from the data lines to the latch. Address Enable (AEN) is used to enable the bits onto the address bus through a three-state enable. The lower order address bits are output by the 8237A directly. Lines A0-A7 should be connected to the address bus. Figure 11 shows the time relationships between CLK, AEN, ADSTB, DB0-DB7 and A0-A7.

During Block and Demand Transfer mode services, which include multiple transfers, the addresses generated will be sequential. For many transfers the data held in the external address latch will remain the same. This data need only change when a carry or borrow from A7 to A8 takes place in the normal sequence of addresses. To save time and speed transfers, the 8237A executes S1 states only when updating of A8-A15 in the latch is necessary. This means for long services, S1 states and Address Strobes may occur only once every 256 transfers, a savings of 255 clock cycles for each 256 transfers.

REGISTER DESCRIPTION

Current Address Register — Each channel has a 16-bit Current Address register. This register holds the value of the address used during DMA transfers. The address is automatically incremented or decremented after each transfer and the intermediate values of the address are stored in the Current Address register during the transfer. This register is written or read by the microprocessor in successive 8-bit bytes. It may also be reinitialized by an Autoinitialize back to its original value. Autoinitialize takes place only after an EOP.

Current Word Register — Each channel has a 16-bit Current Word Count register. This register determines the number of transfers to be performed. The actual number of transfers will be one more than the number programmed in the Current Word Count register (i.e., programming a count of 100 will result in 101 transfers). The word count is decremented after each transfer. The intermediate value of the word count is stored in the register during the transfer. When the value in the register goes from zero to FFFFH, a TC will be generated. This register is loaded or read in successive 8-bit bytes by the microprocessor in the Program Condition. Following the end of a DMA service it may also be reinitialized by an Autoinitialize back to its original value. Autoinitialize can occur only when an EOP occurs. If it is not Autoinitialized, this register will have a count of FFFFH after TC.

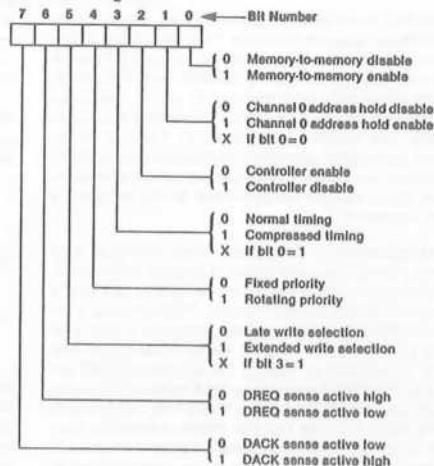
Base Address and Base Word Count Registers — Each channel has a pair of Base Address and Base Word Count registers. These 16-bit registers store the original value of their associated current registers. During Autoinitialize these values are used to restore the current registers to their original values. The base registers are written simultaneously with their corresponding current register in 8-bit bytes in the Program Condition by the microprocessor. These registers cannot be read by the microprocessor.

Command Register — This 8-bit register controls the operation of the 8237A. It is programmed by the microprocessor in the Program Condition and is cleared by Reset or a Master Clear instruction. The following table lists the function of the command bits. See Figure 6 for address coding.

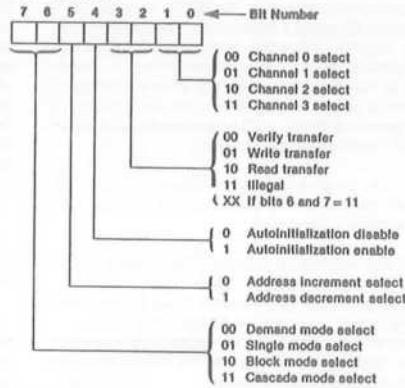
Mode Register — Each channel has a 6-bit Mode register associated with it. When the register is being written to by the microprocessor in the Program Condition, bits 0 and 1 determine which channel Mode register is to be written.

Request Register — The 8237A can respond to requests for DMA service which are initiated by software as well as by a DREQ. Each channel has a request bit associated with it in the 4-bit Request register. These are non-maskable and subject to prioritization by the Priority Encoder network. Each register bit is set or reset sepa-

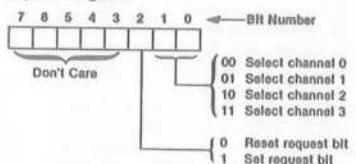
Command Register



Mode Register

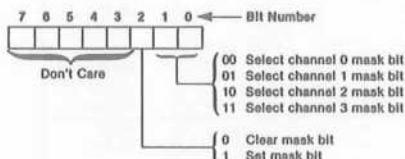


Request Register

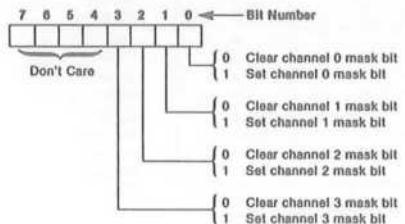


rately under software control or is cleared upon generation of a TC or external EOP. The entire register is cleared by a Reset. To set or reset a bit, the software loads the proper form of the data word. See Figure 5 for register address coding. In order to make a software request, the channel must be in Block Mode.

Mask Register — Each channel has associated with it a mask bit which can be set to disable the incoming DREQ. Each mask bit is set when its associated channel produces an EOP if the channel is not programmed for AutoInitialize. Each bit of the 4-bit Mask register may also be set or cleared separately under software control. The entire register is also set by a Reset. This disables all DMA requests until a clear Mask register instruction allows them to occur. The instruction to separately set or clear the mask bits is similar in form to that used with the Request register. See Figure 5 for instruction addressing.



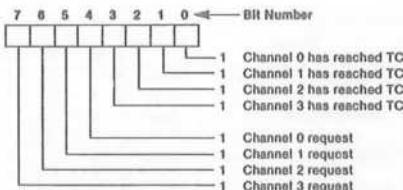
All four bits of the Mask register may also be written with a single command.



Register	Operation	Signals						
		CS	IOR	IOW	A3	A2	A1	A0
Command	Write	0	1	0	1	0	0	0
Mode	Write	0	1	0	1	0	1	1
Request	Write	0	1	0	1	0	0	1
Mask	Set/Reset	0	1	0	1	0	1	0
Mask	Write	0	1	0	1	1	1	1
Temporary	Read	0	0	1	1	1	0	0
Status	Read	0	0	1	1	0	0	0

Figure 5. Definition of Register Codes

Status Register — The Status register is available to be read out of the 8237A by the microprocessor. It contains information about the status of the devices at this point. This information includes which channels have reached a terminal count and which channels have pending DMA requests. Bits 0-3 are set every time a TC is reached by that channel or an external EOP is applied. These bits are cleared upon Reset and on each Status Read. Bits 4-7 are set whenever their corresponding channel is requesting service.



Temporary Register — The Temporary register is used to hold data during memory-to-memory transfers. Following the completion of the transfers, the last word moved can be read by the microprocessor in the Program Condition. The Temporary register always contains the last byte transferred in the previous memory-to-memory operation, unless cleared by a Reset.

Software Commands — These are additional special software commands which can be executed in the Program Condition. They do not depend on any specific bit pattern on the data bus. The two software commands are:

Clear First/Last Flip-Flop: This command is executed prior to writing or reading new address or word count information to the 8237A. This initializes the flip-flop to a known state so that subsequent accesses to register contents by the microprocessor will address upper and lower bytes in the correct sequence.

Master Clear: This software instruction has the same effect as the hardware Reset. The Command, Status, Request, Temporary, and Internal First/Last Flip-Flop registers are cleared and the Mask register is set. The 8237A will enter the Idle cycle.

Clear Mask Register: This command clears the mask bits of all four channels, enabling them to accept DMA requests.

Figure 6 lists the address codes for the software commands:

Signals						Operation
A3	A2	A1	A0	IOR	IOW	
I	0	0	0	0	I	Read Status Register
I	0	0	0	1	0	Write Command Register
I	0	0	1	0	I	Illegal
I	0	0	I	I	0	Write Request Register
I	0	I	0	0	I	Illegal
I	0	I	0	1	0	Write Single Mask Register Bit
I	0	I	I	0	I	Illegal
I	0	I	I	I	0	Write Moda Register
I	1	0	0	0	I	Illegal
I	1	0	0	1	0	Clear Byte Pointer Flip/Flop
I	1	0	I	0	I	Read Temporary Register
I	1	0	I	I	0	Master Clear
I	1	1	0	0	I	Illegal
I	1	1	0	I	0	Clear Mask Register
I	1	1	I	0	I	Illegal
I	1	1	I	I	0	Write All Mask Register Bits

Figure 6. Software Command Codes



Channel	Register	Operation	Signals						Internal Flip-Flop	Data Bus DB0-DB7	
			C5	IOR	IOW	A3	A2	A1	A0		
0	Base and Current Address	Write	0	1	0	0	0	0	0	0	A0-A7 AB-A15
	Current Address	Read	0	0	1	0	0	0	0	0	A0-A7 AB-A15
	Base and Current Word Count	Write	0	1	0	0	0	0	1	0	W0-W7 W8-W15
	Current Word Count	Read	0	0	1	0	0	0	1	0	W0-W7 W8-W15
1	Base and Current Address	Write	0	1	0	0	0	1	0	0	A0-A7 AB-A15
	Current Address	Read	0	0	1	0	0	1	0	0	A0-A7 AB-A15
	Base and Current Word Count	Write	0	1	0	0	0	1	1	0	W0-W7 W8-W15
	Current Word Count	Read	0	0	1	0	0	1	1	0	W0-W7 W8-W15
2	Base and Current Address	Write	0	1	0	0	1	0	0	0	A0-A7 AB-A15
	Current Address	Read	0	1	0	0	1	0	0	1	A0-A7 AB-A15
	Base and Current Word Count	Write	0	1	0	0	1	0	0	0	W0-W7 W8-W15
	Current Word Count	Read	0	0	1	0	1	0	1	0	W0-W7 W8-W15
3	Base and Current Address	Write	0	1	0	0	1	1	0	0	A0-A7 AB-A15
	Current Address	Read	0	1	0	0	1	1	0	1	A0-A7 AB-A15
	Base and Current Word Count	Write	0	1	0	0	1	1	1	0	W0-W7 W8-W15
	Current Word Count	Read	0	0	1	0	1	1	1	0	W0-W7 W8-W15

Figure 7. Word Count and Address Register Command Codes

PROGRAMMING

The 8237A will accept programming from the host processor any time that HLDA is inactive; this is true even if HRQ is active. The responsibility of the host is to assure that programming and HLDA are mutually exclusive. Note that a problem can occur if a DMA request occurs, on an unmasked channel while the 8237A is being programmed. For instance, the CPU may be starting to reprogram the two byte Address register of channel 1 when channel 1 receives a DMA request. If the 8237A is enabled (bit 2 in the command register is 0) and channel 1 is unmasked, a DMA service will occur after only one byte of the Address register has been reprogrammed. This can be avoided by disabling the controller (setting bit 2 in the command register) or masking the channel before programming any other registers. Once the programming is complete, the controller can be enabled/unmasked.

After power-up it is suggested that all internal locations, especially the Mode registers, be loaded with some valid value. This should be done even if some channels are unused.

APPLICATION INFORMATION

Figure 8 shows a convenient method for configuring a DMA system with the 8237A controller and an 8080A/8085AH microprocessor system. The multimode DMA controller issues a HRQ to the processor whenever there is at least one valid DMA request from a peripheral device. When the processor replies with a HLDA signal, the 8237A takes control of the address bus, the data bus and the control bus. The address for the first transfer

operation comes out in two bytes — the least significant 8 bits on the eight address outputs and the most significant 8 bits on the data bus. The contents of the data bus are then latched into the 8282 8-bit latch to complete the full 16 bits of the address bus. The 8282 is a high speed, 8-bit, three-state latch in a 20-pin package. After the initial transfer takes place, the latch is updated only after a carry or borrow is generated in the least significant address byte. Four DMA channels are provided when one 8237A is used.

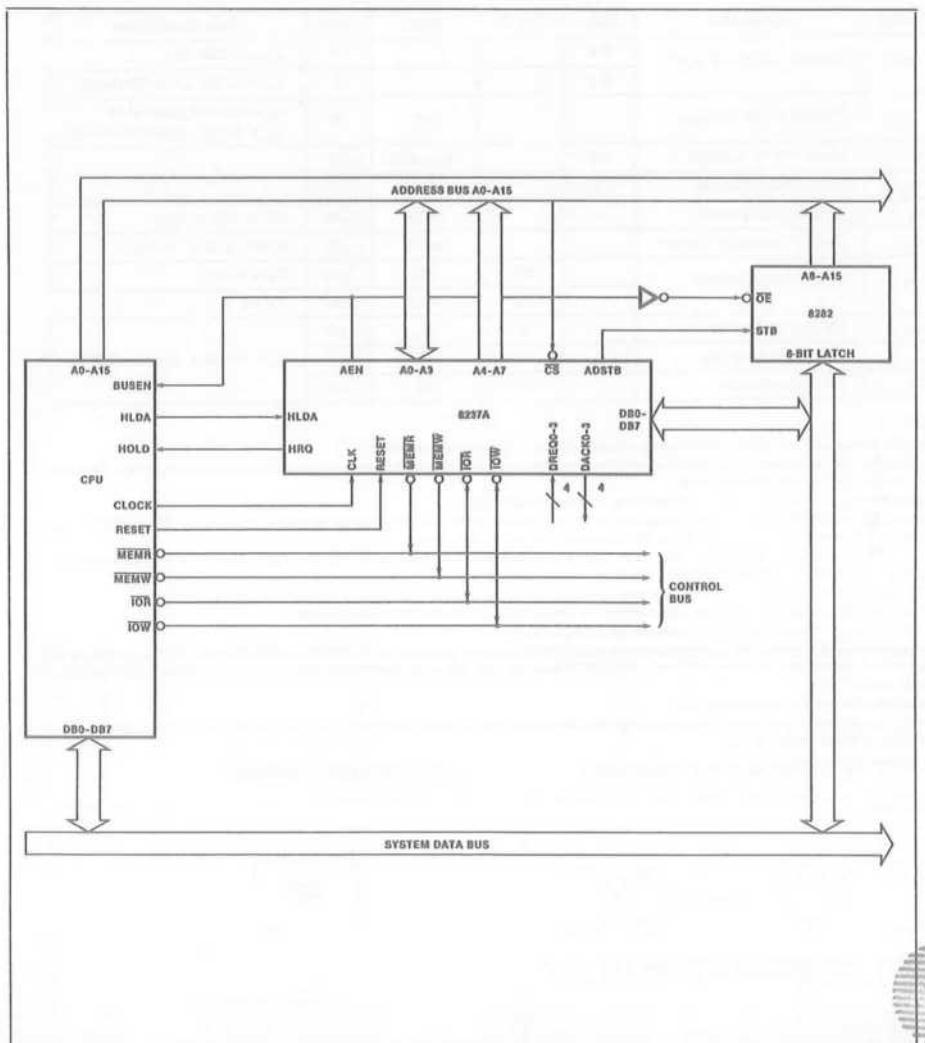


Figure 8. 8237A System Interface

1998

 ATILIM
UNIVERSIT
LIBRARY

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 to 7V
Power Dissipation	1.5 Watt

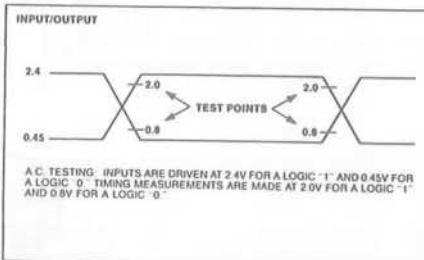
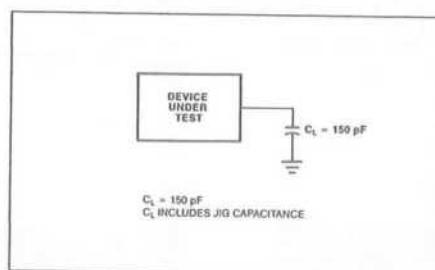
*NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS ($T_A = 0^\circ\text{C}$ to 70°C , $V_{CC} = 5.0\text{ V} \pm 5\%$, GND = 0V)

Symbol	Parameter	Min.	Typ.(1)	Max.	Unit	Test Conditions
V_{OH}	Output HIGH Voltage	2.4			V	$I_{OH} = -200\text{ }\mu\text{A}$
		3.3			V	$I_{OH} = -100\text{ }\mu\text{A}$ (HRQ Only)
V_{OL}	Output LOW Voltage			.45	V	$I_{OL} = 2.0\text{ mA}$ (data bus) $I_{OL} = 3.2\text{ mA}$ (other outputs)
V_{IH}	Input HIGH Voltage	2.0		$V_{CC} + 0.5$	V	
V_{IL}	Input LOW Voltage	-0.5		0.8	V	
I_{LU}	Input Load Current			± 10	μA	$0\text{V} \leq V_{IN} \leq V_{CC}$
I_{LO}	Output Leakage Current			± 10	μA	$0.45\text{V} \leq V_{OUT} \leq V_{CC}$
I_{CC}	V_{CC} Supply Current	65	130	mA		$T_A = +25^\circ\text{C}$
		75	150	mA		$T_A = 0^\circ\text{C}$
C_O	Output Capacitance	4	8	pF		
C_I	Input Capacitance	8	15	pF		$f_C = 1.0\text{ MHz}$, Inputs = 0V
C_{IO}	I/O Capacitance	10	18	pF		

NOTES:

1. Typical values are for $T_A = 25^\circ\text{C}$, nominal supply voltage and nominal processing parameters.
2. Input timing parameters assume transition times of 20 ns or less. Waveform measurement points for both input and output signals are 2.0V for HIGH and 0.8V for LOW, unless otherwise noted.
3. Output loading is 1 TTL gate plus 50 pF capacitance, unless otherwise noted.
4. The net t_{IOR} or t_{MEMW} pulse width for normal write will be $TCY-100$ ns and for extended write will be $2TCY-100$ ns. The net t_{IOR} or t_{MEMR} pulse width for normal read will be $2TCY-50$ ns and for compressed read will be $TCY-50$ ns.
5. TDO is specified for two different output HIGH levels. TDO1 is measured at 2.0V. TDO2 is measured at 3.3V. The value for TDO2 assumes an external 3.3 k Ω pull-up resistor connected from HRO to V_{CC} .
6. DREQ should be held active until DACK is returned.
7. DREQ and DACK signals may be active high or active low. Timing diagrams assume the active high mode.
8. Output loading on the data bus is 1 TTL gate plus 100 pF capacitance.
9. Successive read and/or write operations by the external processor to program or examine the controller must be timed to allow at least 600 ns for the 8237A, at least 500 ns for the 8237A-4 and at least 400 ns for the 8237A-5, as recovery time between active read or write pulses.
10. Parameters are listed in alphabetical order.
11. Pin 5 is an input that should always be at a logic high level. An internal pull-up resistor will establish a logic high when the pin is left floating. Alternatively, pin 5 may be tied to V_{CC} .

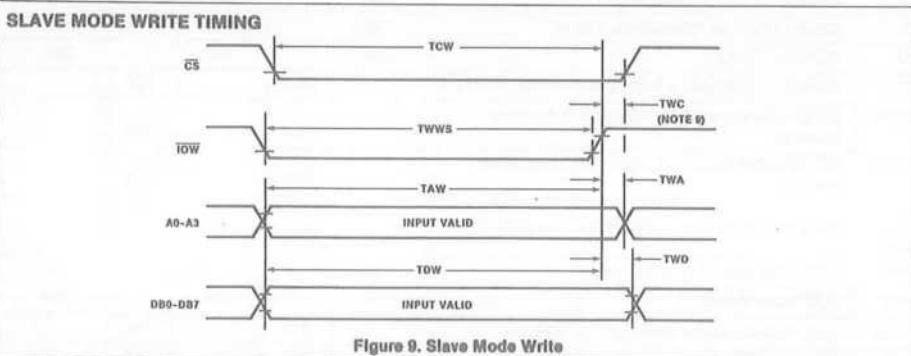
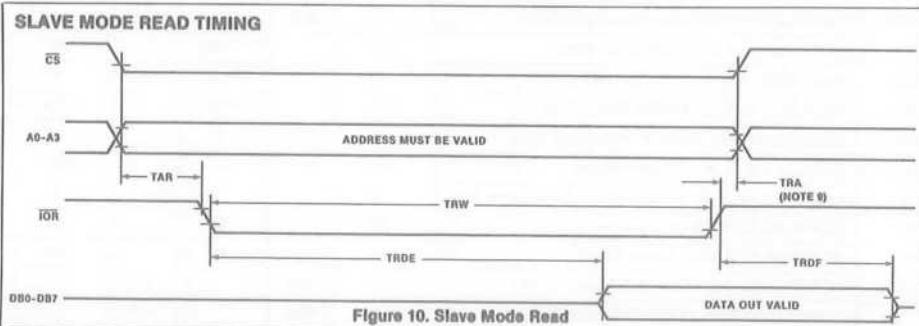
A.C. TESTING INPUT, OUTPUT WAVEFORM**A.C. TESTING LOAD CIRCUIT**

**A.C. CHARACTERISTICS—DMA (MASTER) MODE** ($T_A = 0^\circ\text{C}$ to 70°C ,
 $V_{CC} = +5V \pm 5\%$, $\text{GND} = 0V$)

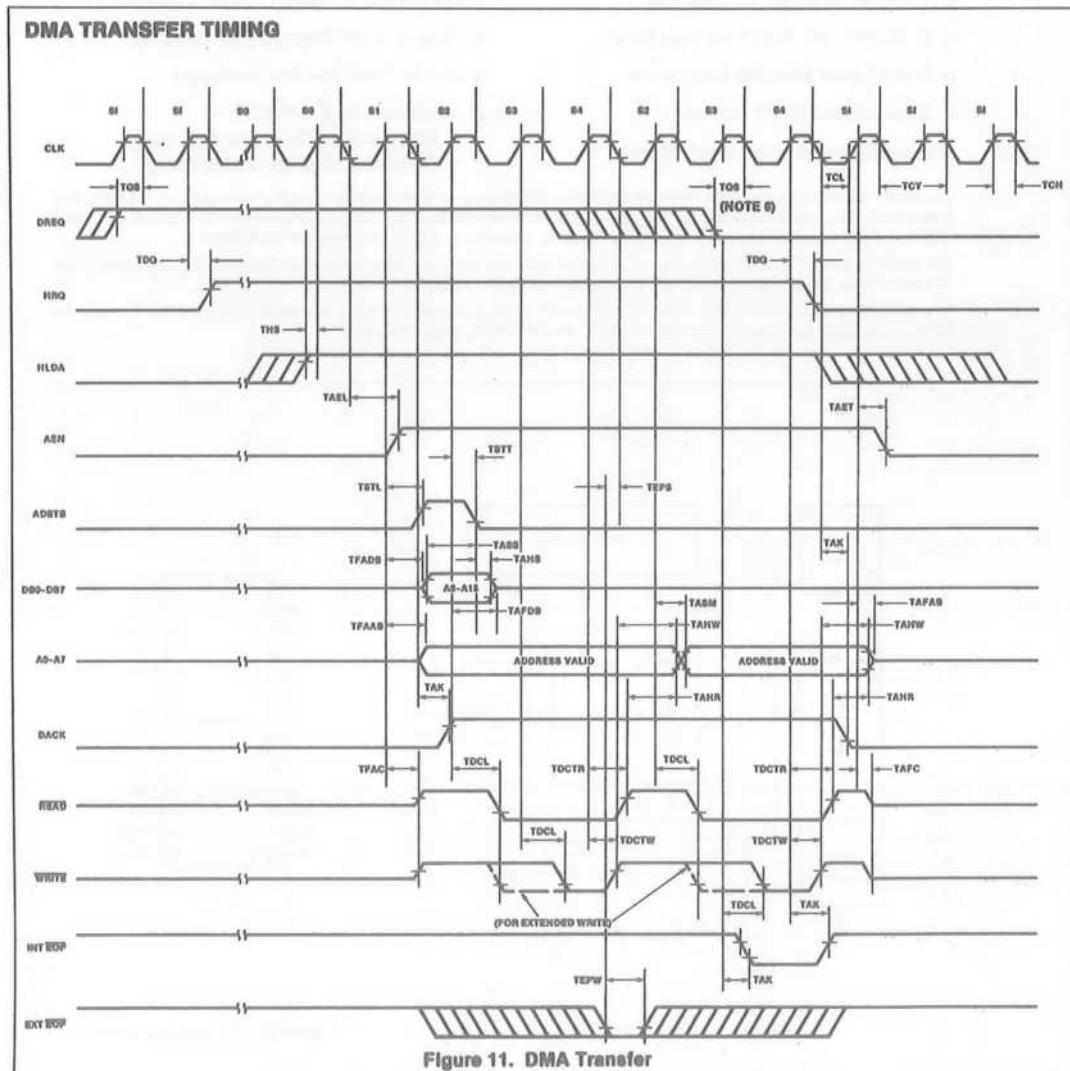
Symbol	Parameter	8237A		8237A-4		8237A-5		Unit
		Min.	Max.	Min.	Max.	Min.	Max.	
TAEL	AEN HIGH from CLK LOW (S1) Delay Time		300		225		200	ns
TAET	AEN LOW from CLK HIGH (S1) Delay Time		200		150		130	ns
TAFAB	ADR Active to Float Delay from CLK HIGH		150		120		90	ns
TAFC	READ or WRITE Float from CLK HIGH		150		120		120	ns
TAFDB	DB Active to Float Delay from CLK HIGH		250		190		170	ns
TAHR	ADR from READ HIGH Hold Time	TCY-100		TCY-100		TCY-100		ns
TAHS	DB from ADSTB LOW Hold Time	50		40		30		ns
TAHW	ADR from WRITE HIGH Hold Time	TCY-50		TCY-50		TCY-50		ns
	DACK Valid from CLK LOW Delay Time (Note 7)		250		220		170	ns
TAK	EOP HIGH from CLK HIGH Delay Time		250		190		170	ns
	EOP LOW to CLK HIGH Delay Time		250		190		100	ns
TASM	ADR Stable from CLK HIGH		250		190		170	ns
TASS	DB to ADSTB LOW Setup Time	100		100		100		ns
TCH	Clock High Time (Transitions ≤ 10 ns)	120		100		80		ns
TCL	Clock Low Time (Transitions ≤ 10 ns)	150		110		68		ns
TCY	CLK Cycle Time	320		250		200		ns
TDCL	CLK HIGH to READ or WRITE LOW Delay (Note 4)		270		200		190	ns
TDCTR	READ HIGH from CLK HIGH (S4) Delay Time (Note 4)		270		210		190	ns
TDCTW	WRITE HIGH from CLK HIGH (S4) Delay Time (Note 4)		200		150		130	ns
TDQ1	HRQ Valid from CLK HIGH Delay Time (Note 5)		160		120		120	ns
			250		190		120	ns
TEPS	EOP LOW from CLK LOW Setup Time	60		45		40		ns
TEPW	EOP Pulse Width	300		225		220		ns
TFAAB	ADR Float to Active Delay from CLK HIGH		250		190		170	ns
TFAC	READ or WRITE Active from CLK HIGH		200		150		150	ns
TFADB	DB Float to Active Delay from CLK HIGH		300		225		200	ns
THS	HLDA Valid to CLK HIGH Setup Time	100		75		75		ns
TIDH	Input Data from MEMR HIGH Hold Time	0		0		0		ns
TIDS	Input Data to MEMR HIGH Setup Time	250		190		170		ns
TODH	Output Data from MEMW HIGH Hold Time	20		20		10		ns
TODV	Output Data Valid to MEMW HIGH	200		125		125		ns
TQS	DREQ to CLK LOW (S1, S4) Setup Time (Note 7)	0		0		0		ns
TRH	CLK to READY LOW Hold Time	20		20		20		ns
TRS	READY to CLK LOW Setup Time	100		60		60		ns
TSTL	ADSTB HIGH from CLK HIGH Delay Time		200		150		130	ns
TSTT	ADSTB LOW from CLK HIGH Delay Time		140		110		90	ns

A.C. CHARACTERISTICS—PERIPHERAL (SLAVE) MODE ($T_A = 0^\circ\text{C}$ to 70°C , $V_{CC} = 5.0\text{V} \pm 5\%$, $\text{GND} = 0\text{V}$)

Symbol	Parameter	8237A		8237A-4		8237A-5		Unit
		Min.	Max.	Min.	Max.	Min.	Max.	
TAR	ADR Valid or CS Low to READ LOW	50		50		50		ns
TAW	ADR Valid to WRITE HIGH Setup Time	200		150		150		ns
TCW	CS Low to WRITE HIGH Setup Time	200		150		150		ns
TDW	Data Valid to WRITE HIGH Setup Time	200		150		150		ns
TRA	ADR or CS Hold from READ HIGH	0		0		0		ns
TRDE	Data Access from READ LOW (Note 8)		200		200		140	ns
TRDF	DB Float Delay from READ HIGH	20	100	20	100	0	70	ns
TRSTD	Power Supply HIGH to RESET LOW Setup Time	500		500		500		ns
TRSTS	RESET to First IOWR		2TCY		2TCY		2TCY	ns
TRSTW	RESET Pulse Width	300		300		300		ns
TRW	READ Width	300		250		200		ns
TWA	ADR from WRITE HIGH Hold Time	20		20		20		ns
TWC	CS HIGH from WRITE HIGH Hold Time	20		20		20		ns
TWD	Data from WRITE HIGH Hold Time	30		30		30		ns
TWWS	Write Width		200		200		160	ns

WAVEFORMS

Figure 9. Slave Mode Write

Figure 10. Slave Mode Read

WAVEFORMS (Continued)



8259A/8259A-2/8259A-8 PROGRAMMABLE INTERRUPT CONTROLLER

- IAPX 86, IAPX 88 Compatible
- MCS-80®, MCS-85® Compatible
- Eight-Level Priority Controller
- Expandable to 64 Levels
- Programmable Interrupt Modes
- Individual Request Mask Capability
- Single +5V Supply (No Clocks)
- 28-Pin Dual-In-Line Package
- Available In EXPRESS
 - Standard Temperature Range
 - Extended Temperature Range

The Intel® 8259A Programmable Interrupt Controller handles up to eight vectored priority interrupts for the CPU. It is cascadable for up to 64 vectored priority interrupts without additional circuitry. It is packaged in a 28-pin DIP, uses NMOS technology and requires a single +5V supply. Circuitry is static, requiring no clock input.

The 8259A is designed to minimize the software and real time overhead in handling multi-level priority interrupts. It has several modes, permitting optimization for a variety of system requirements.

The 8259A is fully upward compatible with the Intel® 8259. Software originally written for the 8259 will operate the 8259A in all 8259 equivalent modes (MCS-80/85, Non-Buffered, Edge Triggered).

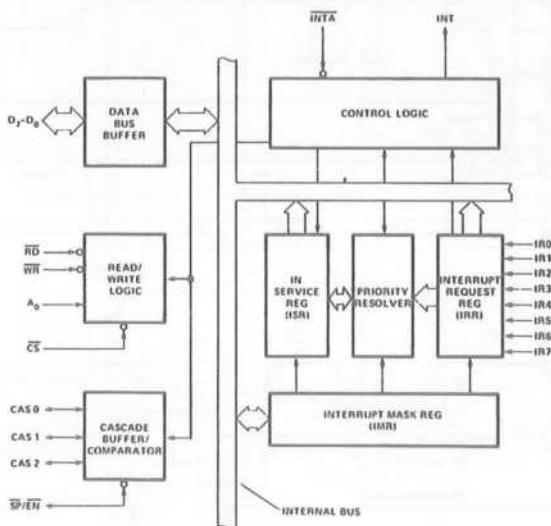


Figure 1. Block Diagram

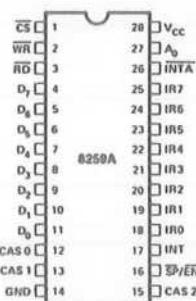


Figure 2. Pin Configuration

Table 1. Pin Description

Symbol	Pin No.	Type	Name and Function
V _{CC}	28	I	Supply: +5V Supply.
GND	14	I	Ground.
CS	1	I	Chip Select: A low on this pin enables RD and WR communication between the CPU and the 8259A. INTA functions are independent of CS.
WR	2	O	Write: A low on this pin when CS is low enables the 8259A to accept command words from the CPU.
RD	3	I	Read: A low on this pin when CS is low enables the 8259A to release status onto the data bus for the CPU.
D ₇ -D ₀	4-11	I/O	Bidirectional Data Bus: Control, status and interrupt-vector information is transferred via this bus.
CAS ₀ -CAS ₂	12, 13, 15	I/O	Cascade Lines: The CAS lines form a private 8259A bus to control a multiple 8259A structure. These pins are outputs for a master 8259A and inputs for a slave 8259A.
SP/EN	16	I/O	Slave Program/Enable Buffer: This is a dual function pin. When in the Buffered Mode it can be used as an output to control buffer transceivers (EN). When not in the buffered mode it is used as an input to designate a master (SP = 1) or slave (SP = 0).
INT	17	O	Interrupt: This pin goes high whenever a valid interrupt request is asserted. It is used to interrupt the CPU, thus it is connected to the CPU's interrupt pin.
IR ₀ -IR ₇	18-25	I	Interrupt Requests: Asynchronous inputs. An interrupt request is executed by raising an IR input (low to high), and holding it high until it is acknowledged (Edge Triggered Mode), or just by a high level on an IR input (Level Triggered Mode).
INTA	26	I	Interrupt Acknowledge: This pin is used to enable 8259A interrupt-vector data onto the data bus by a sequence of interrupt acknowledge pulses issued by the CPU.
A ₀	27	I	AO Address Line: This pin acts in conjunction with the CS, WR, and RD pins. It is used by the 8259A to decipher various Command Words the CPU writes and status the CPU wishes to read. It is typically connected to the CPU A0 address line (A1 for iAPX 86, 88).

FUNCTIONAL DESCRIPTION

Interrupts In Microcomputer Systems

Microcomputer system design requires that I/O devices such as keyboards, displays, sensors and other components receive servicing in an efficient manner so that large amounts of the total system tasks can be assumed by the microcomputer with little or no effect on throughput.

The most common method of servicing such devices is the *Polling* approach. This is where the processor must test each device in sequence and in effect "ask" each one if it needs servicing. It is easy to see that a large portion of the main program is looping through this continuous polling cycle and that such a method would have a serious, detrimental effect on system throughput, thus limiting the tasks that could be assumed by the microcomputer and reducing the cost effectiveness of using such devices.

A more desirable method would be one that would allow the microprocessor to be executing its main program and only stop to service peripheral devices when it is told to do so by the device itself. In effect, the method would provide an external asynchronous input that would inform the processor that it should complete whatever instruction that is currently being executed and fetch a new routine that will service the requesting device. Once this servicing is complete, however, the processor would resume exactly where it left off.

This method is called *Interrupt*. It is easy to see that system throughput would drastically increase, and thus more tasks could be assumed by the microcomputer to further enhance its cost effectiveness.

The Programmable Interrupt Controller (PIC) functions as an overall manager in an Interrupt-Driven system environment. It accepts requests from the peripheral equipment, determines which of the incoming requests is of the highest importance (priority), ascertains whether the incoming request has a higher priority value than the level currently being serviced, and issues an interrupt to the CPU based on this determination.

Each peripheral device or structure usually has a special program or "routine" that is associated with its specific functional or operational requirements; this is referred to as a "service routine". The PIC, after issuing an interrupt to the CPU, must somehow input information into the CPU that can "point" the Program Counter to the service routine associated with the requesting device. This "pointer" is an address in a vectoring table and will often be referred to, in this document, as vectoring data.

The 8259A

The 8259A is a device specifically designed for use in real time, interrupt driven microcomputer systems. It manages eight levels or requests and has built-in features for expandability to other 8259A's (up to 64 levels). It is programmed by the system's software as an I/O peripheral. A selection of priority modes is available to the programmer so that the manner in which the requests are processed by the 8259A can be configured to

match his system requirements. The priority modes can be changed or reconfigured dynamically at any time during the main program. This means that the complete interrupt structure can be defined as required, based on the total system environment.

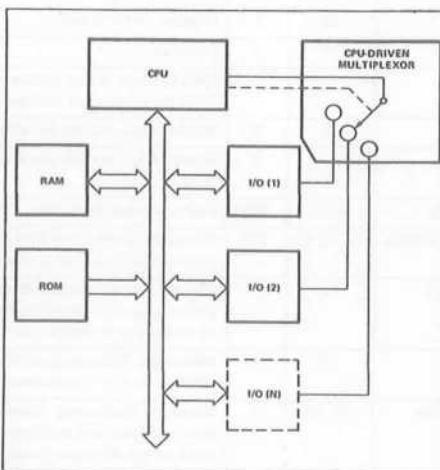


Figure 3a. Polled Method

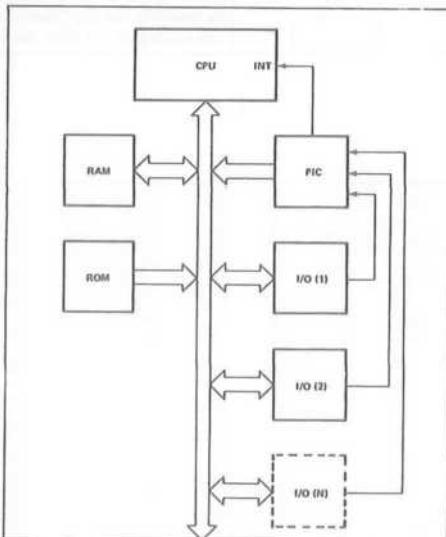


Figure 3b. Interrupt Method

INTERRUPT REQUEST REGISTER (IRR) AND IN-SERVICE REGISTER (ISR)

The interrupts at the IR input lines are handled by two registers in cascade, the Interrupt Request Register (IRR) and the In-Service Register (ISR). The IRR is used to store all the interrupt levels which are requesting service; and the ISR is used to store all the interrupt levels which are being serviced.

PRIORITY RESOLVER

This logic block determines the priorities of the bits set in the IRR. The highest priority is selected and strobed into the corresponding bit of the ISR during INTA pulse.

INTERRUPT MASK REGISTER (IMR)

The IMR stores the bits which mask the interrupt lines to be masked. The IMR operates on the IRR. Masking of a higher priority input will not affect the interrupt request lines of lower priority.

INT (INTERRUPT)

This output goes directly to the CPU interrupt input. The V_{OH} level on this line is designed to be fully compatible with the 8080A, 8085A and 8086 input levels.

INTA (INTERRUPT ACKNOWLEDGE)

INTA pulses will cause the 8259A to release vectoring information onto the data bus. The format of this data depends on the system mode (μ PM) of the 8259A.

DATA BUS BUFFER

This 3-state, bidirectional 8-bit buffer is used to interface the 8259A to the system Data Bus. Control words and status information are transferred through the Data Bus Buffer.

READ/WRITE CONTROL LOGIC

The function of this block is to accept OUTput commands from the CPU. It contains the Initialization Command Word (ICW) registers and Operation Command Word (OCW) registers which store the various control formats for device operation. This function block also allows the status of the 8259A to be transferred onto the Data Bus.

CS (CHIP SELECT)

A LOW on this input enables the 8259A. No reading or writing of the chip will occur unless the device is selected.

WR (WRITE)

A LOW on this input enables the CPU to write control words (ICWs and OCWs) to the 8259A.

RD (READ)

A LOW on this input enables the 8259A to send the status of the Interrupt Request Register (IRR), In Service Register (ISR), the Interrupt Mask Register (IMR), or the interrupt level onto the Data Bus.

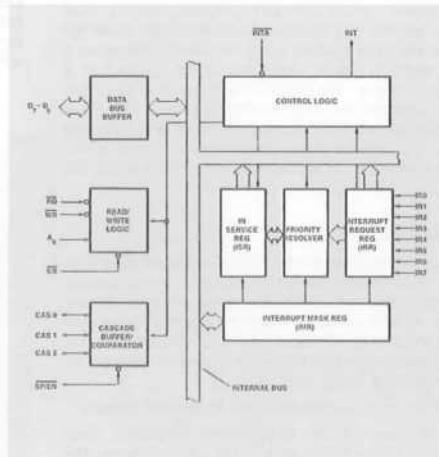


Figure 4a. 8259A Block Diagram

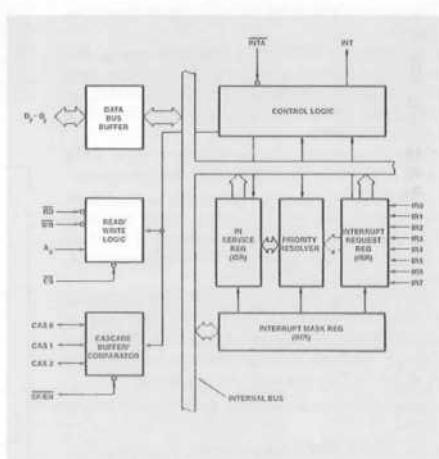


Figure 4b. 8259A Block Diagram

A₀

This input signal is used in conjunction with WR and RD signals to write commands into the various command registers, as well as reading the various status registers of the chip. This line can be tied directly to one of the address lines.

THE CASCADE BUFFER/COMPARATOR

This function block stores and compares the IDs of all 8259A's used in the system. The associated three I/O pins (CAS0-2) are outputs when the 8259A is used as a master and are inputs when the 8259A is used as a slave. As a master, the 8259A sends the ID of the interrupting slave device onto the CAS0-2 lines. The slave thus selected will send its preprogrammed subroutine address onto the Data Bus during the next one or two consecutive INTA pulses. (See section "Cascading the 8259A".)

INTERRUPT SEQUENCE

The powerful features of the 8259A in a microcomputer system are its programmability and the interrupt routine addressing capability. The latter allows direct or indirect jumping to the specific interrupt routine requested without any polling of the interrupting devices. The normal sequence of events during an interrupt depends on the type of CPU being used.

The events occur as follows in an MCS-80/85 system:

1. One or more of the INTERRUPT REQUEST lines (IR7-0) are raised high, setting the corresponding IRR bit(s).
2. The 8259A evaluates these requests, and sends an INT to the CPU, if appropriate.
3. The CPU acknowledges the INT and responds with an INTA pulse.
4. Upon receiving an INTA from the CPU group, the highest priority ISR bit is set, and the corresponding IRR bit is reset. The 8259A will also release a CALL instruction code (11001101) onto the 8-bit Data Bus through its D7-0 pins.
5. This CALL instruction will initiate two more INTA pulses to be sent to the 8259A from the CPU group.
6. These two INTA pulses allow the 8259A to release its preprogrammed subroutine address onto the Data Bus. The lower 8-bit address is released at the first INTA pulse and the higher 8-bit address is released at the second INTA pulse.
7. This completes the 3-byte CALL instruction released by the 8259A. In the AEOI mode the ISR bit is reset at the end of the third INTA pulse. Otherwise, the ISR bit remains set until an appropriate EOI command is issued at the end of the interrupt sequence.

The events occurring in an iAPX 86 system are the same until step 4.

4. Upon receiving an INTA from the CPU group, the highest priority ISR bit is set and the corresponding IRR bit is reset. The 8259A does not drive the Data Bus during this cycle.
5. The iAPX 86/10 will initiate a second INTA pulse. During this pulse, the 8259A releases an 8-bit pointer onto the Data Bus where it is read by the CPU.
6. This completes the interrupt cycle. In the AEOI mode the ISR bit is reset at the end of the second INTA pulse. Otherwise, the ISR bit remains set until an appropriate EOI command is issued at the end of the interrupt subroutine.

If no interrupt request is present at step 4 of either sequence (i.e., the request was too short in duration) the 8259A will issue an interrupt level 7. Both the vectoring bytes and the CAS lines will look like an interrupt level 7 was requested.

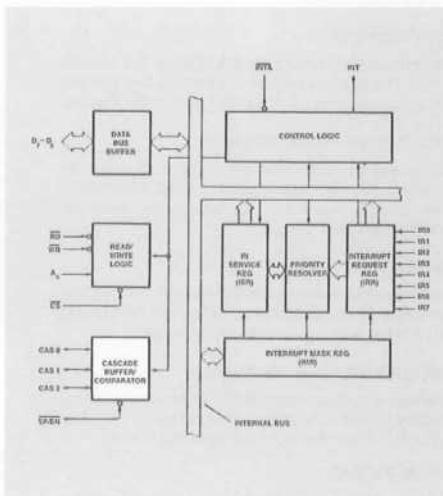


Figure 4c. 8259A Block Diagram

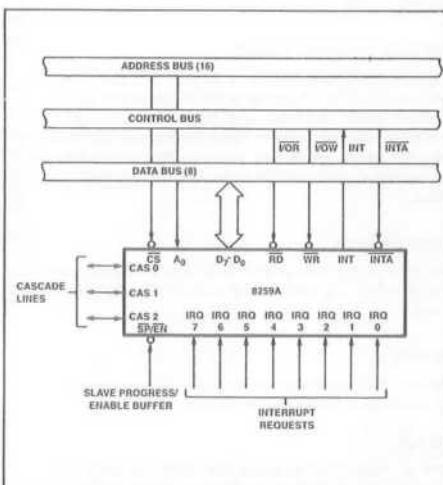


Figure 5. 8259A Interface to Standard System Bus



INTERRUPT SEQUENCE OUTPUTS

MCS-80®, MCS-85®

This sequence is timed by three INTA pulses. During the first INTA pulse the CALL opcode is enabled onto the data bus.

**Content of First Interrupt
Vector Byte**

	D7	D6	D5	D4	D3	D2	D1	D0
CALL CODE	1	1	0	0	1	1	0	1

During the second INTA pulse the lower address of the appropriate service routine is enabled onto the data bus. When Interval = 4 bits A5-A7 are programmed, while A0-A4 are automatically inserted by the 8259A. When Interval = 8 only A6 and A7 are programmed, while A0-A5 are automatically inserted.

**Content of Second Interrupt
Vector Byte**

IR	Interval = 4							
	D7	D6	D5	D4	D3	D2	D1	D0
7	A7	A6	A5	1	1	1	0	0
6	A7	A6	A5	1	1	0	0	0
5	A7	A6	A5	1	0	1	0	0
4	A7	A6	A5	1	0	0	0	0
3	A7	A6	A5	0	1	1	0	0
2	A7	A6	A5	0	1	0	0	0
1	A7	A6	A5	0	0	1	0	0
0	A7	A6	A5	0	0	0	0	0

IR	Interval = 8							
	D7	D6	D5	D4	D3	D2	D1	D0
7	A7	A6	1	1	1	0	0	0
6	A7	A6	1	1	0	0	0	0
5	A7	A6	1	0	1	0	0	0
4	A7	A6	1	0	0	0	0	0
3	A7	A6	0	1	1	0	0	0
2	A7	A6	0	1	0	0	0	0
1	A7	A6	0	0	1	0	0	0
0	A7	A6	0	0	0	0	0	0

During the third INTA pulse the higher address of the appropriate service routine, which was programmed as byte 2 of the initialization sequence (A8-A15), is enabled onto the bus.

**Content of Third Interrupt
Vector Byte**

D7	D6	D5	D4	D3	D2	D1	D0
A15	A14	A13	A12	A11	A10	A9	A8

IAPX 86, IAPX 88

IAPX 86 mode is similar to MCS-80 mode except that only two interrupt acknowledge cycles are issued by the processor and no CALL opcode is sent to the processor. The first interrupt acknowledge cycle is similar to that of MCS-80, 85 systems in that the 8259A uses it to internally freeze the state of the interrupts for priority resolution and as a master it issues the interrupt code on the cascade lines at the end of the INTA pulse. On this first cycle it does

not issue any data to the processor and leaves its data bus buffers disabled. On the second interrupt acknowledge cycle in IAPX 86 mode the master (or slave if so programmed) will send a byte of data to the processor with the acknowledged interrupt code composed as follows (note the state of the AD1 mode control is ignored and A5-A11 are unused in IAPX 86 mode):

**Content of Interrupt Vector Byte
for IAPX 86 System Mode**

	D7	D6	D5	D4	D3	D2	D1	D0
IR7	T7	T6	T5	T4	T3	1	1	1
IR6	T7	T6	T5	T4	T3	1	1	0
IR5	T7	T6	T5	T4	T3	1	0	1
IR4	T7	T6	T5	T4	T3	1	0	0
IR3	T7	T6	T5	T4	T3	0	1	1
IR2	T7	T6	T5	T4	T3	0	1	0
IR1	T7	T6	T5	T4	T3	0	0	1
IR0	T7	T6	T5	T4	T3	0	0	0

PROGRAMMING THE 8259A

The 8259A accepts two types of command words generated by the CPU:

1. **Initialization Command Words (ICWs):** Before normal operation can begin, each 8259A in the system must be brought to a starting point — by a sequence of 2 to 4 bytes timed by WR pulses.

2. **Operation Command Words (OCWs):** These are the command words which command the 8259A to operate in various interrupt modes. These modes are:

- a. Fully nested mode
- b. Rotating priority mode
- c. Special mask mode
- d. Polled mode

The OCWs can be written into the 8259A anytime after initialization.

INITIALIZATION COMMAND WORDS (ICWS)

GENERAL

Whenever a command is issued with A0=0 and D4=1, this is interpreted as Initialization Command Word 1 (ICW1). ICW1 starts the initialization sequence during which the following automatically occur.

- a. The edge sense circuit is reset, which means that following initialization, an interrupt request (IR) input must make a low-to-high transition to generate an interrupt.
- b. The Interrupt Mask Register is cleared.
- c. IR7 input is assigned priority 7.
- d. The slave mode address is set to 7.
- e. Special Mask Mode is cleared and Status Read is set to IRR.
- f. If IC4=0, then all functions selected in ICW4 are set to zero. (Non-Buffered mode*, no Auto-EOI, MCS-80, 85 system).

*Note: Master/Slave in ICW4 is only used in the buffered mode.

INITIALIZATION COMMAND WORDS 1 AND 2 (ICW1, ICW2)

A_5-A_{15} : *Page starting address of service routines.* In an MCS 80/85 system, the 8 request levels will generate CALLs to 8 locations equally spaced in memory. These can be programmed to be spaced at intervals of 4 or 8 memory locations, thus the 8 routines will occupy a page of 32 or 64 bytes, respectively.

The address format is 2 bytes long (A_0-A_{15}). When the routine interval is 4, A_0-A_4 are automatically inserted by the 8259A, while A_5-A_{15} are programmed externally. When the routine interval is 8, A_0-A_5 are automatically inserted by the 8259A, while A_6-A_{15} are programmed externally.

The 8-byte interval will maintain compatibility with current software, while the 4-byte interval is best for a compact jump table.

In an iAPX 86 system $A_{15}-A_{11}$ are inserted in the five most significant bits of the vectoring byte and the 8259A sets the three least significant bits according to the interrupt level. $A_{10}-A_5$ are ignored and ADI (Address Interval) has no effect.

LTIM: If LTIM = 1, then the 8259A will operate in the level interrupt mode. Edge detect logic on the interrupt inputs will be disabled.

ADI: CALL address Interval, ADI = 1 then Interval = 4; ADI = 0 then Interval = 8.

SNGL: Single. Means that this is the only 8259A in the system. If SNGL = 1 no ICW3 will be issued.

IC4: If this bit is set — ICW4 has to be read. If ICW4 is not needed, set IC4 = 0.

INITIALIZATION COMMAND WORD 3 (ICW3)

This word is read only when there is more than one 8259A in the system and cascading is used, in which case SNGL = 0. It will load the 8-bit slave register. The functions of this register are:

- In the master mode (either when SP = 1, or in buffered mode when M/S = 1 in ICW4) a "1" is set for each slave in the system. The master then will release byte 1 of the call sequence (for MCS-80/85 system) and will enable the corresponding slave to release bytes 2 and 3 (for iAPX 86 only byte 2) through the cascade lines.
- In the slave mode (either when $\overline{SP} = 0$, or if $BUF = 1$ and $M/S = 0$ in ICW4) bits 2-0 identify the slave. The slave compares its cascade input with these bits and, if they are equal, bytes 2 and 3 of the call sequence (or just byte 2 for iAPX 86) are released by it on the Data Bus.

INITIALIZATION COMMAND WORD 4 (ICW4)

SFNM: If SFNM = 1 the special fully nested mode is programmed.

BUF: If BUF = 1 the buffered mode is programmed. In buffered mode $\overline{SP/EN}$ becomes an enable output and the master/slave determination is by M/S.

M/S: If buffered mode is selected: M/S = 1 means the 8259A is programmed to be a master, M/S = 0 means the 8259A is programmed to be a slave. If BUF = 0, M/S has no function.

AEOI: If AEOI = 1 the automatic end of interrupt mode is programmed.

μ PM: Microprocessor mode: μ PM = 0 sets the 8259A for MCS-80, 85 system operation, μ PM = 1 sets the 8259A for iAPX 86 system operation.

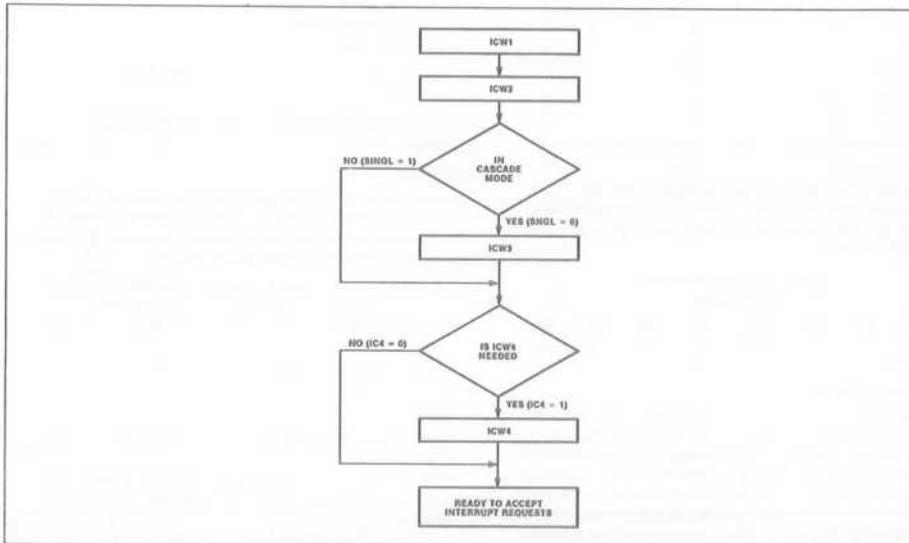


Figure 6. Initialization Sequence

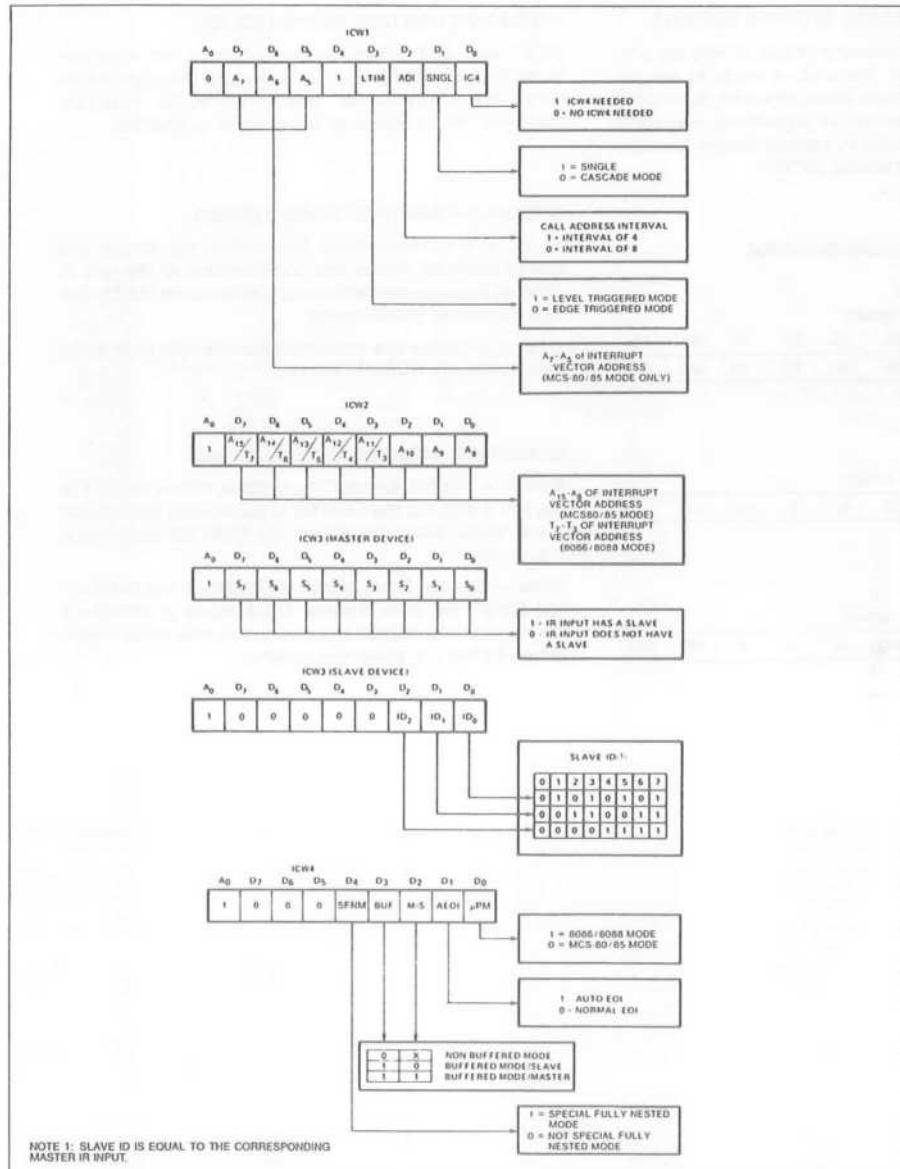


Figure 7. Initialization Command Word Format

OPERATION COMMAND WORDS (OCWs)

After the Initialization Command Words (ICWs) are programmed into the 8259A, the chip is ready to accept Interrupt requests at its input lines. However, during the 8259A operation, a selection of algorithms can command the 8259A to operate in various modes through the Operation Command Words (OCWs).

OPERATION CONTROL WORDS (OCWs)

OCW1								
A0	D7	D6	D5	D4	D3	D2	D1	D0
1	M7	M6	M5	M4	M3	M2	M1	M0

OCW2								
0	R	SL	EOI	0	0	L2	L1	L0

OCW3								
0	ESMM	SMM	0	1	P	RR	RIS	

OPERATION CONTROL WORD 1 (OCW1)

OCW1 sets and clears the mask bits in the interrupt Mask Register (IMR). M₇–M₀ represent the eight mask bits. M = 1 indicates the channel is masked (inhibited), M = 0 indicates the channel is enabled.

OPERATION CONTROL WORD 2 (OCW2)

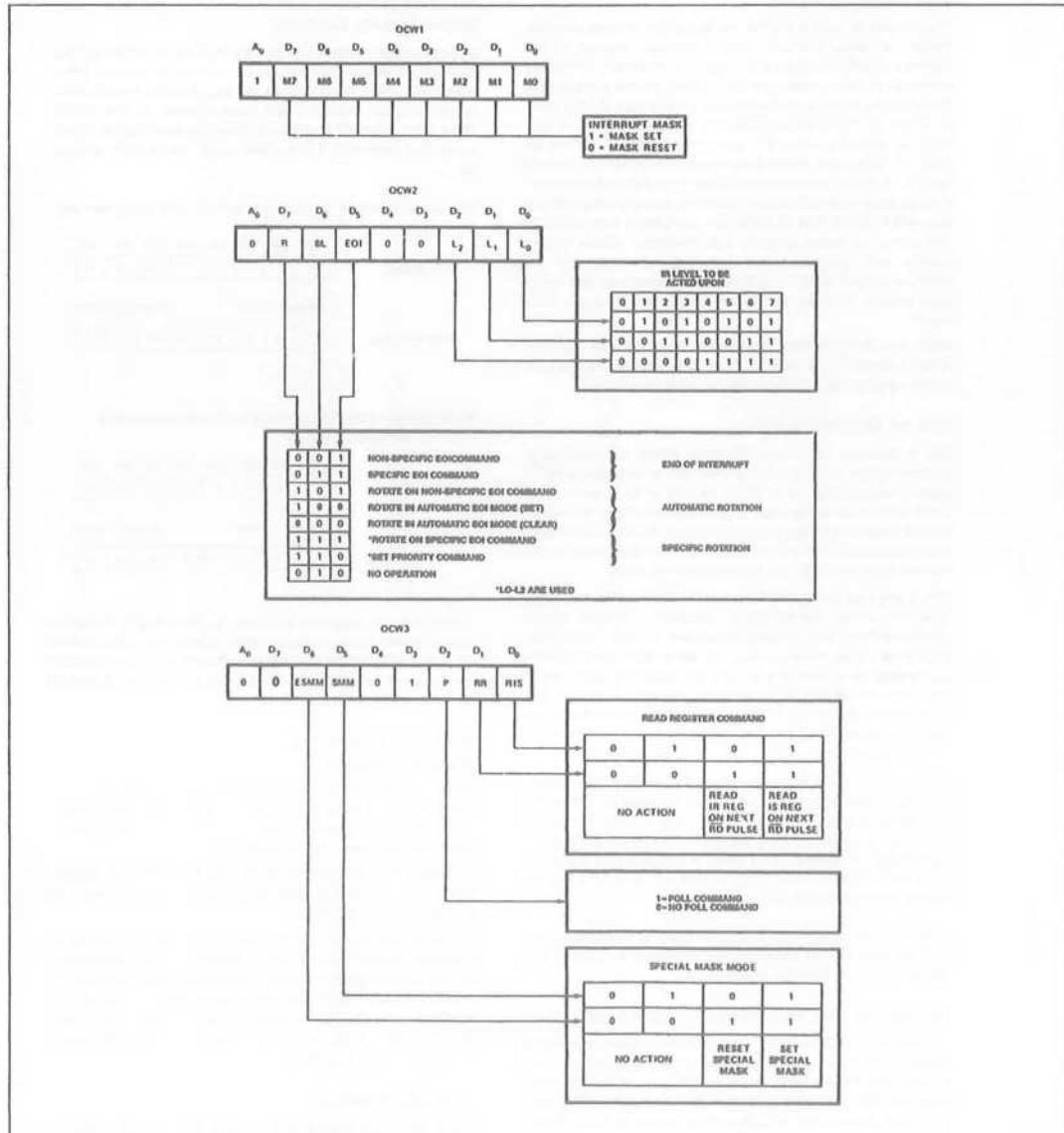
R, SL, EOI — These three bits control the Rotate and End of Interrupt modes and combinations of the two. A chart of these combinations can be found on the Operation Command Word Format.

L₂, L₁, L₀—These bits determine the interrupt level acted upon when the SL bit is active.

OPERATION CONTROL WORD 3 (OCW3)

ESMM — Enable Special Mask Mode. When this bit is set to 1 it enables the SMM bit to set or reset the Special Mask Mode. When ESMM = 0 the SMM bit becomes a "don't care".

SMM — Special Mask Mode. If ESMM = 1 and SMM = 1 the 8259A will enter Special Mask Mode. If ESMM = 1 and SMM = 0 the 8259A will revert to normal mask mode. When ESMM = 0, SMM has no effect.


Figure 8. Operation Command Word Format

FULLY NESTED MODE

This mode is entered after initialization unless another mode is programmed. The interrupt requests are ordered in priority form 0 through 7 (0 highest). When an interrupt is acknowledged the highest priority request is determined and its vector placed on the bus. Additionally, a bit of the Interrupt Service register (ISO-7) is set. This bit remains set until the microprocessor issues an End of Interrupt (EOI) command immediately before returning from the service routine, or if AEOI (Automatic End of Interrupt) bit is set, until the trailing edge of the last INTA. While the IS bit is set, all further interrupts of the same or lower priority are inhibited, while higher levels will generate an interrupt (which will be acknowledged only if the microprocessor internal Interrupt enable flip-flop has been re-enabled through software).

After the initialization sequence, IR0 has the highest priority and IR7 the lowest. Priorities can be changed, as will be explained, in the rotating priority mode.

END OF INTERRUPT (EOI)

The In Service (IS) bit can be reset either automatically following the trailing edge of the last in sequence INTA pulse (when AEOI bit in ICW1 is set) or by a command word that must be issued to the 8259A before returning from a service routine (EOI command). An EOI command must be issued twice if in the Cascade mode, once for the master and once for the corresponding slave.

There are two forms of EOI command: Specific and Non-Specific. When the 8259A is operated in modes which preserve the fully nested structure, it can determine which IS bit to reset on EOI. When a Non-Specific EOI command is issued the 8259A will automatically reset the highest IS bit of those that are set, since in the fully nested mode the highest IS level was necessarily the last level acknowledged and serviced. A non-specific EOI can be issued with OCW2 (EOI = 1, SL = 0, R = 0).

When a mode is used which may disturb the fully nested structure, the 8259A may no longer be able to determine the last level acknowledged. In this case a Specific End of Interrupt must be issued which includes as part of the command the IS level to be reset. A specific EOI can be issued with OCW2 (EOI = 1, SL = 1, R = 0, and LO-L2 is the binary level of the IS bit to be reset).

It should be noted that an IS bit that is masked by an IMR bit will not be cleared by a non-specific EOI if the 8259A is in the Special Mask Mode.

AUTOMATIC END OF INTERRUPT (AEOI) MODE

If AEOI = 1 in ICW4, then the 8259A will operate in AEOI mode continuously until reprogrammed by ICW4. In this mode the 8259A will automatically perform a non-specific EOI operation at the trailing edge of the last interrupt acknowledge pulse (third pulse in MCS-80/85, second in iAPX 86). Note that from a system standpoint, this mode should be used only when a nested multilevel interrupt structure is not required within a single 8259A. The AEOI mode can only be used in a master 8259A and not a slave.

**AUTOMATIC ROTATION
(Equal Priority Devices)**

In some applications there are a number of interrupting devices of equal priority. In this mode a device, after being serviced, receives the lowest priority, so a device requesting an interrupt will have to wait, in the worst case until each of 7 other devices are serviced at most once. For example, if the priority and "in service" status is:

Before Rotate (IR4 the highest priority requiring service)

"IS" Status	IS7	IS6	IS5	IS4	IS3	IS2	IS1	IS0
Priority Status	7	6	5	4	3	2	1	0
Lowest Priority Highest Priority								

After Rotate (IR4 was serviced, all other priorities rotated correspondingly)

"IS" Status	IS7	IS6	IS5	IS4	IS3	IS2	IS1	IS0
Priority Status	2	1	0	7	6	5	4	3
Highest Priority Lowest Priority								

There are two ways to accomplish Automatic Rotation using OCW2, the Rotation on Non-Specific EOI Command ($R = 1, SL = 0, EOI = 1$) and the Rotate in Automatic EOI Mode which is set by ($R = 1, SL = 0, EOI = 0$) and cleared by ($R = 0, SL = 0, EOI = 0$).

**SPECIFIC ROTATION
(Specific Priority)**

The programmer can change priorities by programming the bottom priority and thus fixing all other priorities; i.e., if IR5 is programmed as the bottom priority device, then IR6 will have the highest one.

The Set Priority command is issued in OCW2 where: $R = 1, SL = 1; LO-L2$ is the binary priority level code of the bottom priority device.

Observe that in this mode internal status is updated by software control during OCW2. However, it is independent of the End of Interrupt (EOI) command (also executed by OCW2). Priority changes can be executed during an EOI command by using the Rotate on Specific EOI command in OCW2 ($R = 1, SL = 1, EOI = 1$ and $LO-L2 = IR$ level to receive bottom priority).

INTERRUPT MASKS

Each Interrupt Request input can be masked individually by the Interrupt Mask Register (IMR) programmed through OCW1. Each bit in the IMR masks one interrupt channel if it is set (1). Bit 0 masks IR0, Bit 1 masks IR1 and so forth. Masking an IR channel does not affect the other channels operation.

SPECIAL MASK MODE

Some applications may require an interrupt service routine to dynamically alter the system priority structure during its execution under software control. For example, the routine may wish to inhibit lower priority requests for a portion of its execution but enable some of them for another portion.

The difficulty here is that if an Interrupt Request is acknowledged and an End of Interrupt command did not reset its IS bit (i.e., while executing a service routine), the 8259A would have inhibited all lower priority requests with no easy way for the routine to enable them.

That is where the Special Mask Mode comes in. In the special Mask Mode, when a mask bit is set in OCW1, it inhibits further interrupts at that level and enables interrupts from all other levels (lower as well as higher) that are not masked.

Thus, any interrupts may be selectively enabled by loading the mask register.

The special Mask Mode is set by OCW3 where: SSMM = 1, SMM = 1, and cleared where SSMM = 1, SMM = 0.

POLL COMMAND

In this mode the INT output is not used or the microprocessor internal Interrupt Enable flip-flop is reset, disabling its interrupt input. Service to devices is achieved by software using a Poll command.

The Poll command is issued by setting P = "1" in OCW3. The 8259A treats the next RD pulse to the 8259A (i.e., $\overline{RD} = 0, \overline{CS} = 0$) as an interrupt acknowledge, sets the appropriate IS bit if there is a request, and reads the priority level. Interrupt is frozen from WR to RD.

The word enabled onto the data bus during RD is:

D7	D6	D5	D4	D3	D2	D1	D0
I	-	-	-	-	W2	W1	W0

WO-W2: Binary code of the highest priority level requesting service.

I: Equal to a "1" if there is an interrupt.

This mode is useful if there is a routine command common to several levels so that the INTA sequence is not needed (saves ROM space). Another application is to use the poll mode to expand the number of priority levels to more than 64.

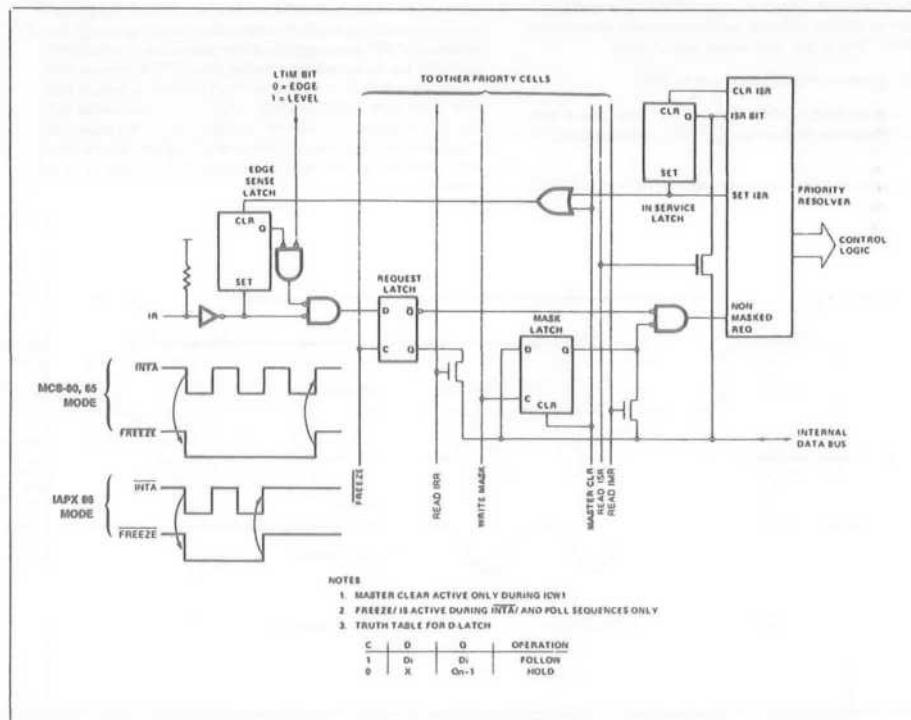


Figure 9. Priority Cell—Simplified Logic Diagram

READING THE 8259A STATUS

The input status of several internal registers can be read to update the user information on the system. The following registers can be read via OCW3 (IRR and ISR or OCW1 [IMR]).

Interrupt Request Register (IRR): 8-bit register which contains the levels requesting an interrupt to be acknowledged. The highest request level is reset from the IRR when an interrupt is acknowledged. (Not affected by IMR.)

In-Service Register (ISR): 8-bit register which contains the priority levels that are being serviced. The ISR is updated when an End of Interrupt Command is issued.

Interrupt Mask Register: 8-bit register which contains the interrupt request lines which are masked.

The IRR can be read when, prior to the RD pulse, a Read Register Command is issued with OCW3 (RR = 1, RIS = 0.)

The ISR can be read when, prior to the RD pulse, a Read Register Command is issued with OCW3 (RR = 1, RIS = 1).

There is no need to write an OCW3 before every status read operation, as long as the status read corresponds with the previous one; i.e., the 8259A "remembers" whether the IRR or ISR has been previously selected by the OCW3. This is not true when poll is used.

After initialization the 8259A is set to IRR.

For reading the IMR, no OCW3 is needed. The output data bus will contain the IMR whenever RD is active and AO = 1 (OCW1).

Polling overrides status read when P = 1, RR = 1 in OCW3.

EDGE AND LEVEL TRIGGERED MODES

This mode is programmed using bit 3 in ICW1.

If LTIM = '0', an interrupt request will be recognized by a low to high transition on an IR input. The IR input can remain high without generating another interrupt.

If LTIM = '1', an interrupt request will be recognized by a 'high' level on IR Input, and there is no need for an edge detection. The interrupt request must be removed before the EOI command is issued or the CPU interrupt is enabled to prevent a second interrupt from occurring.

The priority cell diagram shows a conceptual circuit of the level sensitive and edge sensitive input circuitry of the 8259A. Be sure to note that the request latch is a transparent D type latch.

In both the edge and level triggered modes the IR inputs must remain high until after the falling edge of the first INTA. If the IR input goes low before this time a DEFAULT IR7 will occur when the CPU acknowledges the interrupt. This can be a useful safeguard for detecting interrupts caused by spurious noise glitches on the IR inputs. To implement this feature the IR7 routine is used for "clean up" simply executing a return instruction, thus ignoring the interrupt. If IR7 is needed for other purposes a default IR7 can still be detected by reading the ISR. A normal IR7 interrupt will set the corresponding ISR bit, a default IR7 won't. If a default IR7 routine occurs during a normal IR7 routine, however, the ISR will remain set. In this case it is necessary to keep track of whether or not the IR7 routine was previously entered. If another IR7 occurs it is a default.

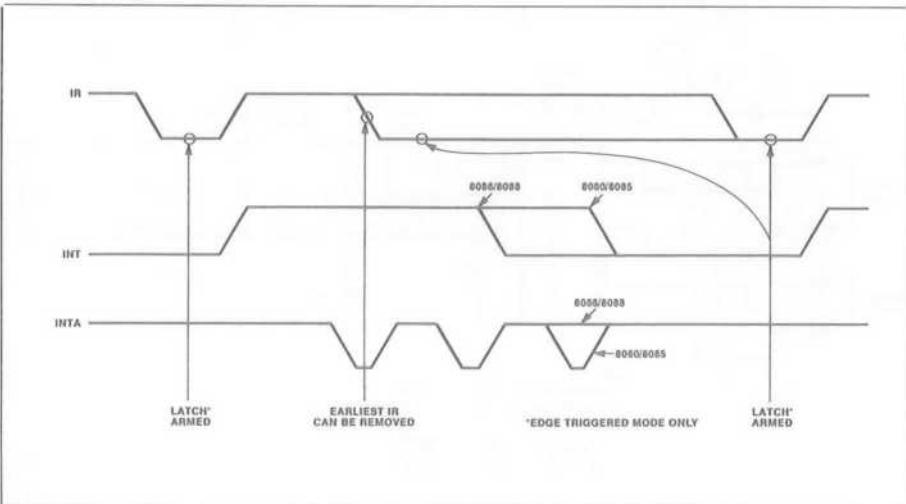


Figure 10. IR Triggering Timing Requirements

THE SPECIAL FULLY NESTED MODE

This mode will be used in the case of a big system where cascading is used, and the priority has to be conserved within each slave. In this case the fully nested mode will be programmed to the master (using ICW4). This mode is similar to the normal nested mode with the following exceptions:

- When an interrupt request from a certain slave is in service this slave is not locked out from the master's priority logic and further interrupt requests from higher priority IR's within the slave will be recognized by the master and will initiate interrupts to the processor. (In the normal nested mode a slave is masked out when its request is in service and no higher requests from the same slave can be serviced.)
- When exiting the Interrupt Service routine the software has to check whether the interrupt serviced was the only one from that slave. This is done by sending a non-specific End of Interrupt (EOI) command to the slave and then reading its In-Service register and checking for zero. If it is empty, a non-specific EOI can be sent to the master too. If not, no EOI should be sent.

BUFFERED MODE

When the 8259A is used in a large system where bus driving buffers are required on the data bus and the cascading mode is used, there exists the problem of enabling buffers.

The buffered mode will structure the 8259A to send an enable signal on SP/EN to enable the buffers. In this

mode, whenever the 8259A's data bus outputs are enabled, the SP/EN output becomes active.

This modification forces the use of software programming to determine whether the 8259A is a master or a slave. Bit 3 in ICW4 programs the buffered mode, and bit 2 in ICW4 determines whether it is a master or a slave.

CASCADE MODE

The 8259A can be easily interconnected in a system of one master with up to eight slaves to handle up to 64 priority levels.

The master controls the slaves through the 3 line cascade bus. The cascade bus acts like chip selects to the slaves during the INTA sequence.

In a cascade configuration, the slave interrupt outputs are connected to the master interrupt request inputs. When a slave request line is activated and afterwards acknowledged, the master will enable the corresponding slave to release the device routine address during bytes 2 and 3 of INTA. (Byte 2 only for 8086/8088).

The cascade bus lines are normally low and will contain the slave address code from the trailing edge of the first INTA pulse to the trailing edge of the third pulse. Each 8259A in the system must follow a separate initialization sequence and can be programmed to work in a different mode. An EOI command must be issued twice: once for the master and once for the corresponding slave. An address decoder is required to activate the Chip Select (CS) input of each 8259A.

The cascade lines of the Master 8259A are activated only for slave inputs, non slave inputs leave the cascade line inactive (low).

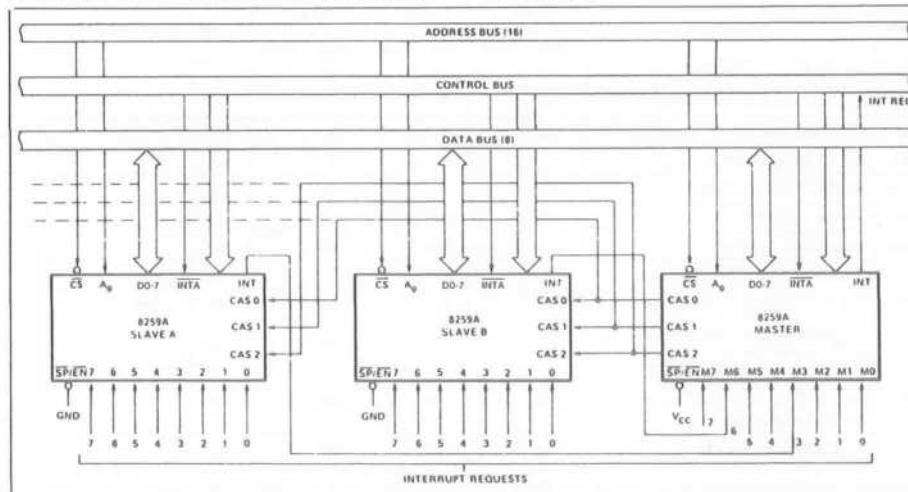


Figure 11. Cascading the 8259A



8259A/8259A-2/8259A-8

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias	0°C to 70°C
Storage Temperature	-65°C to +150°C
Voltage on Any Pin with Respect to Ground	-0.5V to +7V
Power Dissipation	1 Watt

*NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied.

D.C. CHARACTERISTICS [TA = 0°C to 70°C, VCC = 5V ± 5% (8259A-8), VCC = 5V ± 10% (8259A, 8259A-2)]

Symbol	Parameter	Min.	Max.	Units	Test Conditions
VIL	Input Low Voltage	-0.5	0.8	V	
VIH	Input High Voltage	2.0*	VCC + 0.5V	V	
VOL	Output High Voltage		0.45	V	IOL = 2.2mA
VOH	Output High Voltage	2.4		V	IOH = -400μA
VOH(INT)	Interrupt Output High Voltage	3.5		V	IOH = -100μA
		2.4		V	IOH = -400μA
ILI	Input Load Current	-10	+10	μA	0V ≤ VIN ≤ VCC
ILOL	Output Leakage Current	-10	+10	μA	0.45V ≤ VOUT ≤ VCC
ICC	VCC Supply Current		85	mA	
ILIR	IR Input Load Current		-300	μA	VIN = 0
			10	μA	VIN = VCC

*Note: For Extended Temperature EXPRESS V_{IH} = 2.3V.

CAPACITANCE (TA = 25°C; VCC = GND = 0V)

Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Conditions
C _{IN}	Input Capacitance			10	pF	f _c = 1 MHZ
C _{I/O}	I/O Capacitance			20	pF	Unmeasured pins returned to V _{SS}

A.C. CHARACTERISTICS [TA = 0°C to 70°C, VCC = 5V ± 5% (8259A-8), VCC = 5V ± 10% (8259A, 8259A-2)]**TIMING REQUIREMENTS**

Symbol	Parameter	8259A-8		8259A		8259A-2		Units	Test Conditions
		Min.	Max.	Min.	Max.	Min.	Max.		
TAHRL	AO/CS Setup to RD/INTA↓	50	0	0	0	0	0	ns	
TRHAX	AO/CS Hold after RD/INTA↓	5	0	0	0	0	0	ns	
TRLRH	RD Pulse Width	420	235	420	160	420	160	ns	
TAHWL	AO/CS Setup to WR↓	50	0	0	0	0	0	ns	
TWHAX	AO/CS Hold after WR↓	20	0	0	0	0	0	ns	
TWLWH	WR Pulse Width	400	290	400	190	400	190	ns	
TDVWH	Data Setup to WR↓	300	240	300	160	300	160	ns	
TWHDX	Data Hold after WR↓	40	0	40	0	40	0	ns	
TJLJH	Interrupt Request Width (Low)	100	100	100	100	100	100	ns	See Note 1
TCVIAL	Cascade Setup to Second or Third INTA↓ (Slave Only)	55	55	40	40	40	40	ns	
TRHRL	End of RD to next RD End of INTA to next INTA within an INTA sequence only	160	160	160	160	160	160	ns	
TWHWL	End of WR to next WR	190	190	190	190	190	190	ns	

A.C. CHARACTERISTICS (Continued)

Symbol	Parameter	8259A-8		8259A		8259A-2		Units	Test Conditions
		Min.	Max.	Min.	Max.	Min.	Max.		
*TCHCL	End of Command to next Command (Not same command type)	500		500		500		ns	
	End of INTA sequence to next INTA sequence.								

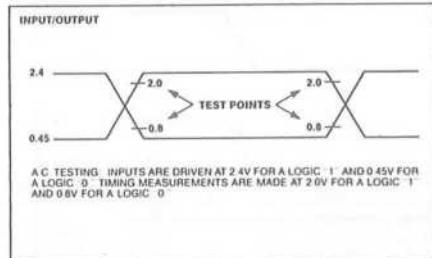
*Worst case timing for TCHCL in an actual microprocessor system is typically much greater than 500 ns (i.e. 8085A = 1.6μs,
8085A-2 = 1μs, 8086 = 1μs, 8086-2 = 625 ns)

NOTE: This is the low time required to clear the input latch in the edge triggered mode.

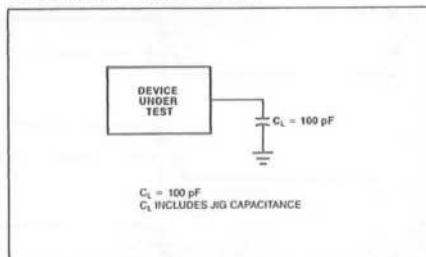
TIMING RESPONSES

Symbol	Parameter	8259A-8		8259A		8259A-2		Units	Test Conditions
		Min.	Max.	Min.	Max.	Min.	Max.		
TRLDV	Data Valid from RD/INTA		300		200		120	ns	C of Data Bus = 100 pF
TRHDZ	Data Float after RD/INTA	10	200	10	100	10	85	ns	C of Data Bus
TJIIH	Interrupt Output Delay		400		350		300	ns	Max test C = 100 pF
TIALCV	Cascade Valid from First INTA (Master Only)		565		565		360	ns	Min. test C = 15 pF
TRLEL	Enable Active from RD or INTA	160		125		100		ns	C _{RI} = 100 pF
TRHEH	Enable Inactive from RD or INTA	325		150		150		ns	C _{CASCADE} = 100 pF
TAHDV	Data Valid from Stable Address	350		200		200		ns	
TCVDV	Cascade Valid to Valid Data		300		300		200	ns	

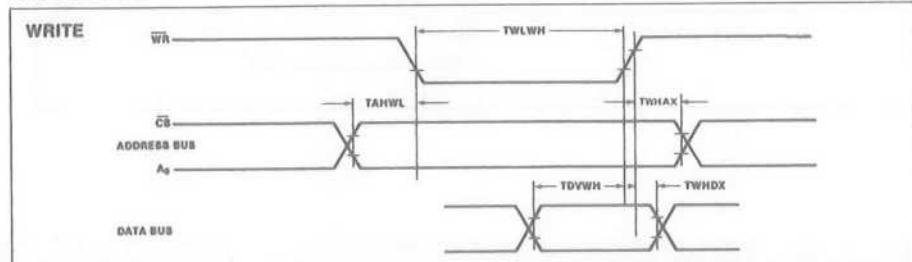
A.C. TESTING INPUT, OUTPUT WAVEFORM

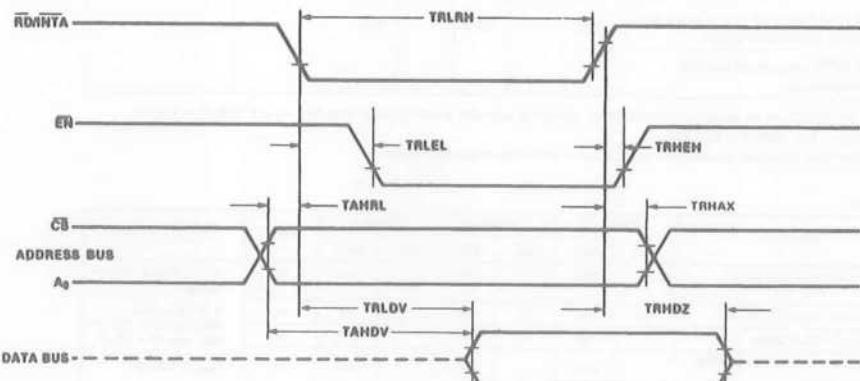
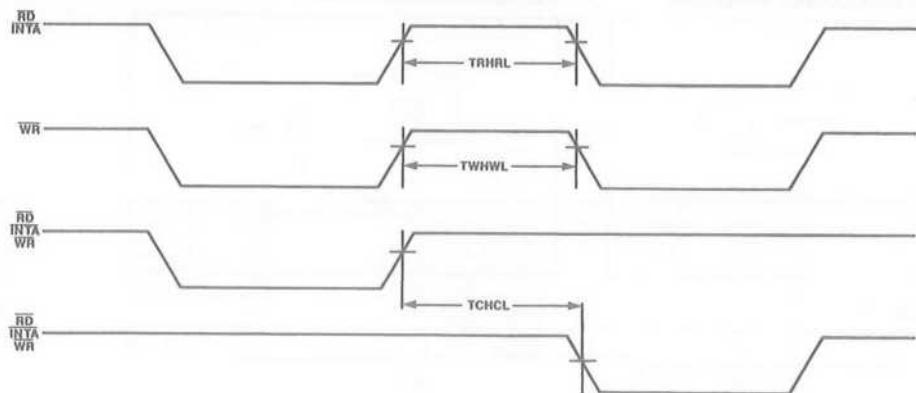


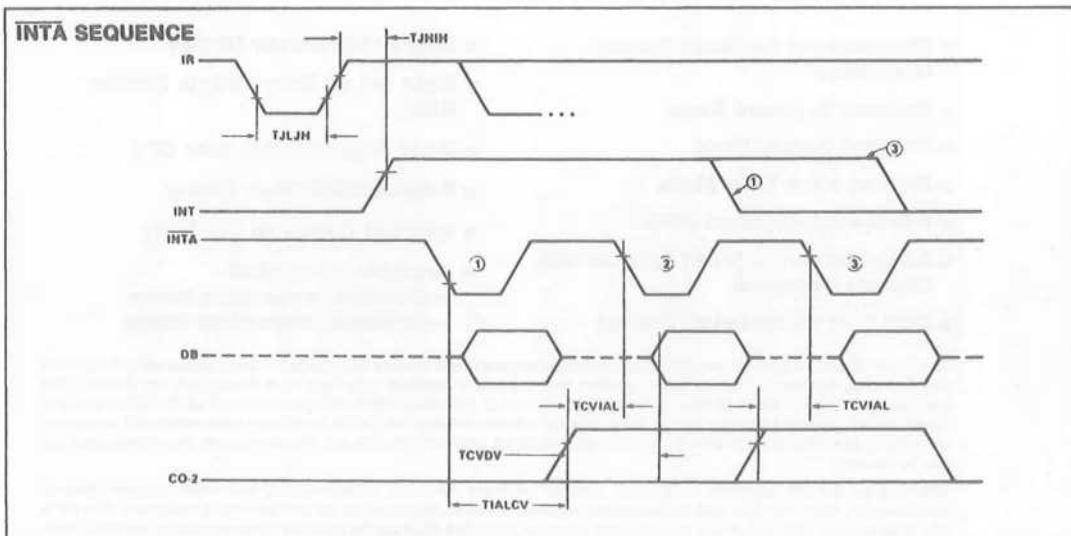
A.C. TESTING LOAD CIRCUIT



WAVEFORMS



WAVEFORMS (Continued)**READ/INTA****OTHER TIMING**

WAVEFORMS (Continued)

NOTES: Interrupt output must remain HIGH at least until leading edge of first INTA.

1. Cycle 1 in iAPX 86, iAPX 88 systems, the Data Bus is not active.



8279/8279-5

PROGRAMMABLE KEYBOARD/DISPLAY INTERFACE

- Simultaneous Keyboard Display Operations
- Scanned Keyboard Mode
- Scanned Sensor Mode
- Strobed Input Entry Mode
- 8-Character Keyboard FIFO
- 2-Key Lockout or N-Key Rollover with Contact Debounce
- Dual 8- or 16-Numerical Display
- Single 16-Character Display
- Right or Left Entry 16-Byte Display RAM
- Mode Programmable from CPU
- Programmable Scan Timing
- Interrupt Output on Key Entry
- Available In EXPRESS
 - Standard Temperature Range
 - Extended Temperature Range

The Intel® 8279 is a general purpose programmable keyboard and display I/O interface device designed for use with Intel® microprocessors. The keyboard portion can provide a scanned interface to a 64-contact key matrix. The keyboard portion will also interface to an array of sensors or a strobed interface keyboard, such as the hall effect and ferrite variety. Key depressions can be 2-key lockout or N-key rollover. Keyboard entries are debounced and strobed in an 8-character FIFO. If more than 8 characters are entered, overrun status is set. Key entries set the interrupt output line to the CPU.

The display portion provides a scanned display interface for LED, incandescent, and other popular display technologies. Both numeric and alphanumeric segment displays may be used as well as simple indicators. The 8279 has 16x8 display RAM which can be organized into dual 16x4. The RAM can be loaded or interrogated by the CPU. Both right entry, calculator and left entry typewriter display formats are possible. Both read and write of the display RAM can be done with auto-increment of the display RAM address.

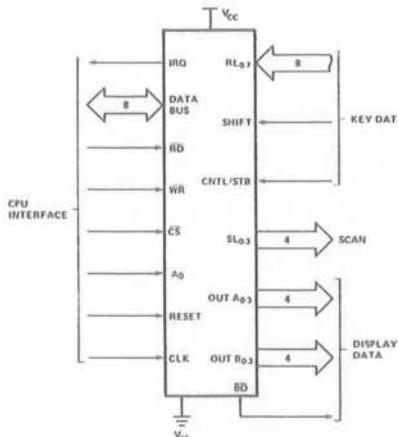


Figure 1. Logic Symbol

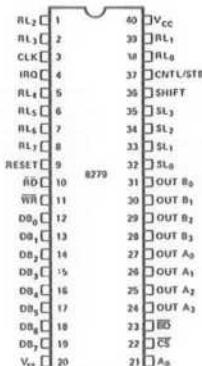


Figure 2. Pin Configuration

HARDWARE DESCRIPTION

The 8279 is packaged in a 40 pin DIP. The following is a functional description of each pin.

Table 1. Pin Descriptions

Symbol	Pin No.	Name and Function	Symbol	Pin No.	Name and Function
DB ₀ -DB ₇	8	Bi-directional data bus: All data and commands between the CPU and the 8279 are transmitted on these lines.	SHIFT	1	Shift: The shift input status is stored along with the key position on key closure in the Scanned Keyboard modes. It has an active internal pullup to keep it high until a switch closure pulls it low.
CLK	1	Clock: Clock from system used to generate internal timing.	CNTL/STB	1	Control/Strobed Input Mode: For keyboard modes this line is used as a control input and stored like status on a key closure. The line is also the strobe line that enters the data into the FIFO in the Strobed Input mode. (Rising Edge). It has an active internal pullup to keep it high until a switch closure pulls it low.
RESET	1	Reset: A high signal on this pin resets the 8279. After being reset the 8279 is placed in the following mode: 1) 16 8-bit character display —left entry. 2) Encoded scan keyboard—2 key lockout. Along with this the program clock prescaler is set to 31.	OUT A ₀ -OUT A ₃ OUT B ₀ -OUT B ₃	4	Outputs: These two ports are the outputs for the 16 x 4 display refresh registers. The data from these outputs is synchronized to the scan lines (SL ₀ -SL ₃) for multiplexed digit displays. The two 4 bit ports may be blanked independently. These two ports may also be considered as one 8-bit port.
CS	1	Chip Select: A low on this pin enables the interface functions to receive or transmit.	BD	1	Blank Display: This output is used to blank the display during digit switching or by a display blanking command.
A ₀	1	Buffer Address: A high on this line indicates the signals in or out are interpreted as a command or status. A low indicates that they are data.			
RD, WR	2	Input/Output Read and Write: These signals enable the data buffers either send data to the external bus or receive it from the external bus.			
IRQ	1	Interrupt Request: In a keyboard mode, the interrupt line is high when there is data in the FIFO/Sensor RAM. The interrupt line goes low with each FIFO/Sensor RAM read and returns high if there is still information in the RAM. In a sensor mode, the interrupt line goes high whenever a change in a sensor is detected.			
V _{SS} , V _{CC}	2	Ground and power supply pins.			
SL ₀ -SL ₃	4	Scan Lines: Scan lines which are used to scan the key switch or sensor matrix and the display digits. These lines can be either encoded (1 of 16) or decoded (1 of 4).			
RL ₀ -RL ₇	8	Return Line: Return line inputs which are connected to the scan lines through the keys or sensor switches. They have active internal pullups to keep them high until a switch closure pulls one low. They also serve as an 8-bit input in the Strobed Input mode.			

FUNCTIONAL DESCRIPTION

Since data input and display are an integral part of many microprocessor designs, the system designer needs an interface that can control these functions without placing a large load on the CPU. The 8279 provides this function for 8-bit microprocessors.

The 8279 has two sections: keyboard and display. The keyboard section can interface to regular typewriter style keyboards or random toggle or thumb switches. The display section drives alphanumeric displays or a bank of indicator lights. Thus the CPU is relieved from scanning the keyboard or refreshing the display.

The 8279 is designed to directly connect to the microprocessor bus. The CPU can program all operating modes for the 8279. These modes include:

Input Modes

- Scanned Keyboard — with encoded (8×8 keyboard) or decoded (4×8 key keyboard) scan lines. A key depression generates a 6-bit encoding of key position. Position and shift and control status are stored in the FIFO. Keys are automatically debounced with 2-key lockout or N-key rollover.
- Scanned Sensor Matrix — with encoded (8×8 matrix switches) or decoded (4×8 matrix switches) scan lines. Key status (open or closed) stored in RAM addressable by CPU.
- Strobed Input — Data on return lines during control line strobe is transferred to FIFO.

Output Modes

- 8 or 16 character multiplexed displays that can be organized as dual 4-bit or single 8-bit ($B_0 = D_0, A_3 = D_7$).
- Right entry or left entry display formats.

Other features of the 8279 include:

- Mode programming from the CPU.
- Clock Prescaler
- Interrupt output to signal CPU when there is keyboard or sensor data available.
- An 8 byte FIFO to store keyboard information.
- 16 byte internal Display RAM for display refresh. This RAM can also be read by the CPU.

PRINCIPLES OF OPERATION

The following is a description of the major elements of the 8279 Programmable Keyboard/Display interface device. Refer to the block diagram in Figure 3.

I/O Control and Data Buffers

The I/O control section uses the \bar{CS} , A_0 , \bar{RD} and \bar{WR} lines to control data flow to and from the various internal registers and buffers. All data flow to and from the 8279 is enabled by \bar{CS} . The character of the information, given or desired by the CPU, is identified by A_0 . A logic one means the information is a command or status. A logic zero means the information is data. \bar{RD} and \bar{WR} determine the direction of data flow through the Data Buffers. The Data Buffers are bi-directional buffers that connect the internal bus to the external bus. When the chip is not selected ($\bar{CS} = 1$), the devices are in a high impedance state. The drivers input during $\bar{WR} = \bar{CS}$ and output during $\bar{RD} = \bar{CS}$.

Control and Timing Registers and Timing Control

These registers store the keyboard and display modes and other operating conditions programmed by the CPU. The modes are programmed by presenting the proper command on the data lines with $A_0 = 1$ and then sending a WR. The command is latched on the rising edge of WR.

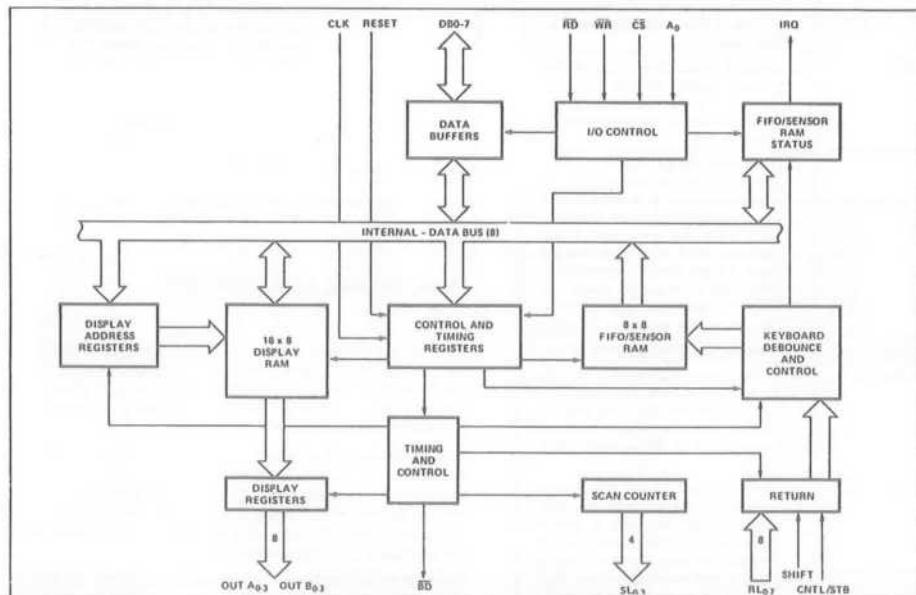


Figure 3. Internal Block Diagram

The command is then decoded and the appropriate function is set. The timing control contains the basic timing counter chain. The first counter is a $\div N$ prescaler that can be programmed to yield an internal frequency of 100 kHz which gives a 5.1 ms keyboard scan time and a 10.3 ms debounce time. The other counters divide down the basic internal frequency to provide the proper key scan, row scan, keyboard matrix scan, and display scan times.

Scan Counter

The scan counter has two modes. In the encoded mode, the counter provides a binary count that must be externally decoded to provide the scan lines for the keyboard and display. In the decoded mode, the scan counter decodes the least significant 2 bits and provides a decoded 1 of 4 scan. Note that when the keyboard is in decoded scan, so is the display. This means that only the first 4 characters in the Display RAM are displayed.

In the encoded mode, the scan lines are active high outputs. In the decoded mode, the scan lines are active low outputs.

Return Buffers and Keyboard Debounce and Control

The 8 return lines are buffered and latched by the Return Buffers. In the keyboard mode, these lines are scanned, looking for key closures in that row. If the debounce circuit detects a closed switch, it waits about 10 msec to check if the switch remains closed. If it does, the address of the switch in the matrix plus the status of SHIFT and CONTROL are transferred to the FIFO. In the scanned Sensor Matrix modes, the contents of the return lines is directly transferred to the corresponding row of the Sensor RAM (FIFO) each key scan time. In Strobed Input mode, the contents of the return lines are transferred to the FIFO on the rising edge of the CNTL/STB line pulse.

FIFO/Sensor RAM and Status

This block is a dual function 8 x 8 RAM. In Keyboard or Strobed Input modes, it is a FIFO. Each new entry is written into successive RAM positions and each is then read in order of entry. FIFO status keeps track of the number of characters in the FIFO and whether it is full or empty. Too many reads or writes will be recognized as an error. The status can be read by an RD with CS low and Ao high. The status logic also provides an IRQ signal when the FIFO is not empty. In Scanned Sensor Matrix mode, the memory is a Sensor RAM. Each row of the Sensor RAM is loaded with the status of the corresponding row of sensor in the sensor matrix. In this mode, IRQ is high if a change in a sensor is detected.

Display Address Registers and Display RAM

The Display Address Registers hold the address of the word currently being written or read by the CPU and the two 4-bit nibbles being displayed. The read/write addresses are programmed by CPU command. They also can be set to auto increment after each read or write. The Display RAM can be directly read by the CPU after the correct mode and address is set. The addresses for the A and B nibbles are automatically updated by the 8279 to match data entry by the CPU. The A and B nibbles can be entered independently or as one word, according to the mode that is set by the CPU. Data entry to the display can be set to either left or right entry. See Interface Considerations for details.

SOFTWARE OPERATION

8279 commands

The following commands program the 8279 operating modes. The commands are sent on the Data Bus with CS low and Ao high and are loaded to the 8279 on the rising edge of WR.

Keyboard/Display Mode Set

MSB		LSB
Code:	0 0 0 D D K K K	

Where DD is the Display Mode and KKK is the Keyboard Mode.

DD

- 0 0 8 8-bit character display — Left entry
- 0 1 16 8-bit character display — Left entry*
- 1 0 8 8-bit character display — Right entry
- 1 1 16 8-bit character display — Right entry

For description of right and left entry, see Interface Considerations. Note that when decoded scan is set in keyboard mode, the display is reduced to 4 characters independent of display mode set.

KKK

- 0 0 0 Encoded Scan Keyboard — 2 Key Lockout*
- 0 0 1 Decoded Scan Keyboard — 2-Key Lockout
- 0 1 0 Encoded Scan Keyboard — N-Key Rollover
- 0 1 1 Decoded Scan Keyboard — N-Key Rollover
- 1 0 0 Encoded Scan Sensor Matrix
- 1 0 1 Decoded Scan Sensor Matrix
- 1 1 0 Strobed Input, Encoded Display Scan
- 1 1 1 Strobed Input, Decoded Display Scan

Program Clock

0 0 1 P P P P

All timing and multiplexing signals for the 8279 are generated by an internal prescaler. This prescaler divides the external clock (pin 3) by a programmable integer. Bits PPPPP determine the value of this integer which ranges from 2 to 31. Choosing a divisor that yields 100 kHz will give the specified scan and debounce times. For instance, if Pin 3 of the 8279 is being clocked by a 2 MHz signal, PPPPP should be set to 10100 to divide the clock by 20 to yield the proper 100 kHz operating frequency.

Read FIFO/Sensor RAM

0 1 0 AI X A A A	X = Don't Care
------------------	----------------

The CPU sets up the 8279 for a read of the FIFO/Sensor RAM by first writing this command. In the Scan Key-

*Default after reset.

board Mode, the Auto-Increment flag (AI) and the RAM address bits (AAA) are irrelevant. The 8279 will automatically drive the data bus for each subsequent read ($A_0 = 0$) in the same sequence in which the data first entered the FIFO. All subsequent reads will be from the FIFO until another command is issued.

In the Sensor Matrix Mode, the RAM address bits AAA select one of the 8 rows of the Sensor RAM. If the AI flag is set ($AI = 1$), each successive read will be from the subsequent row of the sensor RAM.

Read Display RAM

Code:

0	1	1	AI	A	A	A	A
---	---	---	----	---	---	---	---

The CPU sets up the 8279 for a read of the Display RAM by first writing this command. The address bits AAAA select one of the 16 rows of the Display RAM. If the AI flag is set ($AI = 1$), this row address will be incremented after each following read or write to the Display RAM. Since the same counter is used for both reading and writing, this command sets the next read or write address and the sense of the Auto-Increment mode for both operations.

Write Display RAM

Code:

1	0	0	AI	A	A	A	A
---	---	---	----	---	---	---	---

The CPU sets up the 8279 for a write to the Display RAM by first writing this command. After writing the command with $A_0 = 1$, all subsequent writes with $A_0 = 0$ will be to the Display RAM. The addressing and Auto-Increment functions are identical to those for the Read Display RAM. However, this command does not affect the source of subsequent Data Reads; the CPU will read from whichever RAM (Display or FIFO/Sensor) which was last specified. If, indeed, the Display RAM was last specified, the Write Display RAM will, nevertheless, change the next Read location.

Display Write Inhibit/Blanking

Code:

A	B	A	B				
1	0	1	X	IW	IW	BL	BL

The IW Bits can be used to mask nibble A and nibble B in applications requiring separate 4-bit display ports. By setting the IW flag ($IW = 1$) for one of the ports, the port becomes masked so that entries to the Display RAM from the CPU do not affect that port. Thus, if each nibble is input to a BCD decoder, the CPU may write a digit to the Display RAM without affecting the other digit being displayed. It is important to note that bit B_0 corresponds to bit D_0 on the CPU bus, and that bit A_3 corresponds to bit D_7 .

If the user wishes to blank the display, the BL flags are available for each nibble. The last Clear command issued determines the code to be used as a "blank." This code defaults to all zeros after a reset. Note that both BL flags must be set to blank a display formatted with a single 8-bit port.

Clear

Code:

1	1	0	C _D	C _D	C _D	C _F	C _A
---	---	---	----------------	----------------	----------------	----------------	----------------

The C_D bits are available in this command to clear all rows of the Display RAM to a selectable blanking code as follows:

C _D	C _D	C _D					
0	X		All Zeros (X = Don't Care)				
1	0		AB = Hex 20 (0010 0000)				
1	1		All Ones				

Enable clear display when = 1 (or by C_A = 1)

During the time the Display RAM is being cleared ($\sim 160 \mu s$), it may not be written to. The most significant bit of the FIFO status word is set during this time. When the Display RAM becomes available again, it automatically resets.

If the C_F bit is asserted ($C_F = 1$), the FIFO status is cleared and the interrupt output line is reset. Also, the Sensor RAM pointer is set to row 0.

C_A, the Clear All bit, has the combined effect of C_D and C_F; it uses the C_D clearing code on the Display RAM and also clears FIFO status. Furthermore, it resynchronizes the internal timing chain.

End Interrupt/Error Mode Set

Code:

1	1	1	E	X	X	X	X
---	---	---	---	---	---	---	---

 X = Don't care.

For the sensor matrix modes this command lowers the IRQ line and enables further writing into RAM. (The IRQ line would have been raised upon the detection of a change in a sensor value. This would have also inhibited further writing into the RAM until reset.)

For the N-key rollover mode — if the E bit is programmed to "1" the chip will operate in the special Error mode. (For further details, see Interface Considerations Section.)

Status Word

The status word contains the FIFO status, error, and display unavailable signals. This word is read by the CPU when A_0 is high and CS and RD are low. See Interface Considerations for more detail on status word.

Data Read

Data is read when A_0 , CS and RD are all low. The source of the data is specified by the Read FIFO or Read Display commands. The trailing edge of RD will cause the address of the RAM being read to be incremented if the Auto-Increment flag is set. FIFO reads always increment (if no error occurs) independent of AI.

Data Write

Data that is written with A_0 , CS and WR low is always written to the Display RAM. The address is specified by the latest Read Display or Write Display command. Auto-Incrementing on the rising edge of WR occurs if AI set by the latest display command.

INTERFACE CONSIDERATIONS

Scanned Keyboard Mode, 2-Key Lockout

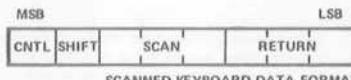
There are three possible combinations of conditions that can occur during debounce scanning. When a key is depressed, the debounce logic is set. Other depressed keys are looked for during the next two scans. If none are encountered, it is a single key depression and the key position is entered into the FIFO along with the status of CNTL and SHIFT lines. If the FIFO was empty, IRO will be set to signal the CPU that there is an entry in the FIFO. If the FIFO was full, the key will not be entered and the error flag will be set. If another closed switch is encountered, no entry to the FIFO can occur. If all other keys are released before this one, then it will be entered to the FIFO. If this key is released before any other, it will be entirely ignored. A key is entered to the FIFO only once per depression, no matter how many keys were pressed along with it or in what order they were released. If two keys are depressed within the debounce cycle, it is a simultaneous depression. Neither key will be recognized until one key remains depressed alone. The last key will be treated as a single key depression.

Increment flag is set to zero, or by the End Interrupt command if the Auto-Increment flag is set to one.

Note: Multiple changes in the matrix Addressed by $(SLO_3 = 0)$ may cause multiple interrupts. ($SLO = 0$ in the Decoded Model). Reset may cause the 8279 to see multiple changes.

Data Format

In the Scanned Keyboard mode, the character entered into the FIFO corresponds to the position of the switch in the keyboard plus the status of the CNTL and SHIFT lines (non-inverted). CNTL is the MSB of the character and SHIFT is the next most significant bit. The next three bits are from the scan counter and indicate the row the key was found in. The last three bits are from the column counter and indicate to which return line the key was connected.



Scanned Keyboard Mode, N-Key Rollover

With N-key Rollover each key depression is treated independently from all others. When a key is depressed, the debounce circuit waits 2 keyboard scans and then checks to see if the key is still down. If it is, the key is entered into the FIFO. Any number of keys can be depressed and another can be recognized and entered into the FIFO. If a simultaneous depression occurs, the keys are recognized and entered according to the order the keyboard scan found them.

In Sensor Matrix mode, the data on the return lines is entered directly in the row of the Sensor RAM that corresponds to the row in the matrix being scanned. Therefore, each switch position maps directly to a Sensor RAM position. The SHIFT and CNTL inputs are ignored in this mode. Note that switches are not necessarily the only thing that can be connected to the return lines in this mode. Any logic that can be triggered by the scan lines can enter data to the return line inputs. Eight multiplexed input ports could be tied to the return lines and scanned by the 8279.



In Strobed Input mode, the data is also entered to the FIFO from the return lines. The data is entered by the rising edge of a CNTL/STB line pulse. Data can come from another encoded keyboard or simple switch matrix. The return lines can also be used as a general purpose strobed input.



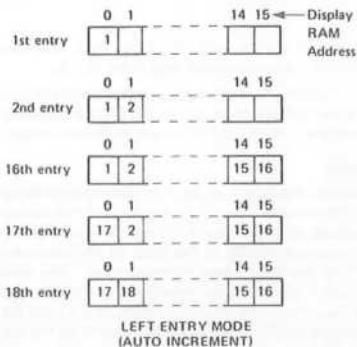
Display

Left Entry

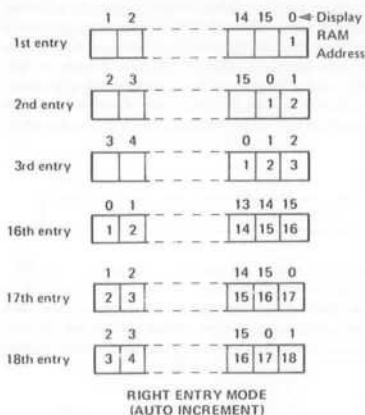
Left Entry mode is the simplest display format in that each display position directly corresponds to a byte (or nibble) in the Display RAM. Address 0 in the RAM is the left-most display character and address 15 (or address 7 in 8 character display) is the right most display character. Entering characters from position zero causes the display to fill from the left. The 17th (9th) character is entered back in the left most position and filling again proceeds from there.

Sensor Matrix Mode

In Sensor Matrix mode, the debounce logic is inhibited. The status of the sensor switch is inputted directly to the Sensor RAM. In this way the Sensor RAM keeps an image of the state of the switches in the sensor matrix. Although debouncing is not provided, this mode has the advantage that the CPU knows how long the sensor was closed and when it was released. A keyboard mode can only indicate a validated closure. To make the software easier, the designer should functionally group the sensors by row since this is the format in which the CPU will read them. The IRQ line goes high if any sensor value change is detected at the end of a sensor matrix scan. The IRQ line is cleared by the first data read operation if the Auto-

**Right Entry**

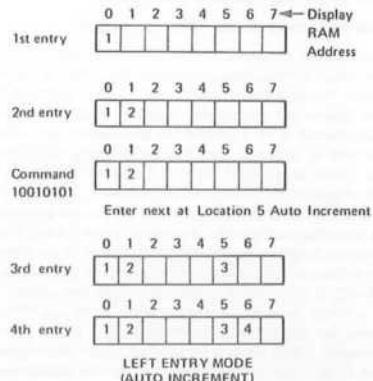
Right entry is the method used by most electronic calculators. The first entry is placed in the right most display character. The next entry is also placed in the right most character after the display is shifted left one character. The left most character is shifted off the end and is lost.



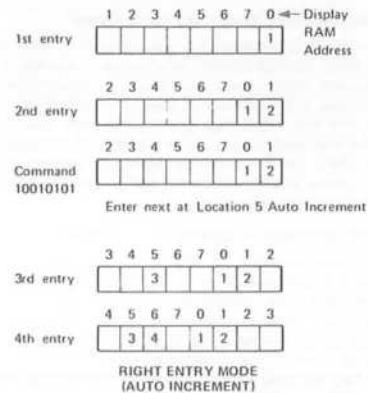
Note that now the display position and register address do not correspond. Consequently, entering a character to an arbitrary position in the Auto Increment mode may have unexpected results. Entry starting at Display RAM address 0 with sequential entry is recommended.

Auto Increment

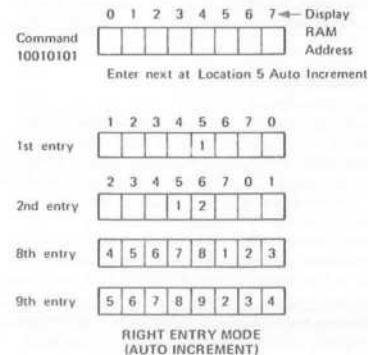
In the Left Entry mode, Auto Incrementing causes the address where the CPU will next write to be incremented by one and the character appears in the next location. With non-Auto Incrementing the entry is both to the same RAM address and display position. Entry to an arbitrary address in the Auto Increment mode has no undesirable side effects and the result is predictable:



In the Right Entry mode, Auto Incrementing and non Incrementing have the same effect as in the Left Entry except if the address sequence is interrupted:



Starting at an arbitrary location operates as shown below:



Entry appears to be from the initial entry point.

8/16 Character Display Formats

If the display mode is set to an 8 character display, the on duty-cycle is double what it would be for a 16 character display (e.g., 5.1 ms scan time for 8 characters vs. 10.3 ms for 16 characters with 100 kHz internal frequency).

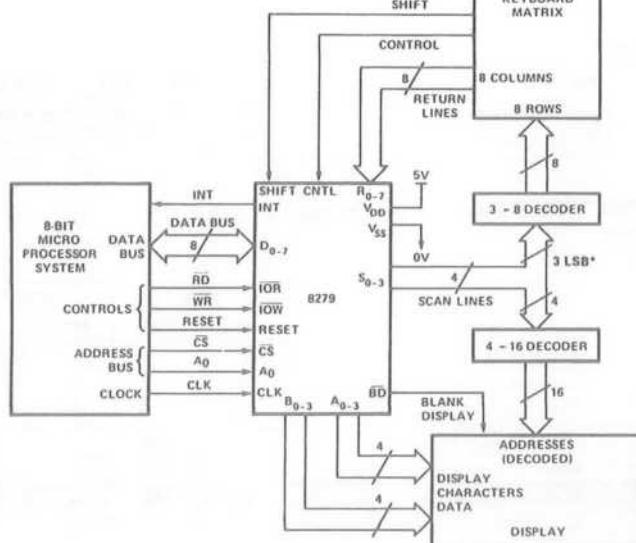
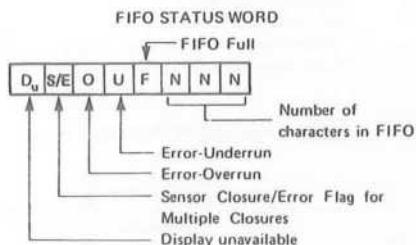
G. FIFO Status

FIFO status is used in the Keyboard and Strobed Input modes to indicate the number of characters in the FIFO and to indicate whether an error has occurred. There are two types of errors possible: overrun and underrun. Overrun occurs when the entry of another character into a full FIFO is attempted. Underrun occurs when the CPU tries to read an empty FIFO.

The FIFO status word also has a bit to indicate that the Display RAM was unavailable because a Clear Display or Clear All command had not completed its clearing operation.

In a Sensor Matrix mode, a bit is set in the FIFO status word to indicate that at least one sensor closure indication is contained in the Sensor RAM.

In Special Error Mode the S/E bit is showing the error flag and serves as an indication to whether a simultaneous multiple closure error has occurred.



*Do not drive the keyboard decoder with the MSB of the scan lines.

Figure 4. System Block Diagram

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature	0°C to 70°C
Storage Temperature	-65°C to 125°C
Voltage on any Pin with Respect to Ground	-0.5V to +7V
Power Dissipation	1 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 [TA = 0°C to 70°C, VSS = VCC = +5V ± 5%, VCC = +5V ± 10% (8279-5)] *

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
VIL1	Input Low Voltage for Return Lines	-0.5	1.4	V	
VIL2	Input Low Voltage for All Others	-0.5	0.8	V	
VIH1	Input High Voltage for Return Lines	2.2		V	
VIH2	Input High Voltage for All Others	2.0		V	
VO _L	Output Low Voltage		0.45	V	Note 1
VOH1	Output High Voltage on Interrupt Line	3.5		V	Note 2
VOH2	Other Outputs	2.4			I _{OH} = -400 μA
IIL1	Input Current on Shift, Control and Return Lines		+10 -100	μA	V _{IN} = V _{CC} V _{IN} = 0V
IIL2	Input Leakage Current on All Others		±10	μA	V _{IN} = V _{CC} to 0V
I _{OFL}	Output Float Leakage		±10	μA	V _{OUT} = V _{CC} to 0.45V
I _{CC}	Power Supply Current		120	mA	

CAPACITANCE

Symbol	Parameter	Typ.	Max.	Unit	Test Conditions
C _{IN}	Input Capacitance	5	10	pF	f _C = 1 MHz Unmeasured pins returned to V _{SS}
C _{OUT}	Output Capacitance	10	20	pF	

A.C. CHARACTERISTICS [TA = 0°C to 70°C, VSS = 0V, (Note 3)] ***Bus Parameters****READ CYCLE**

Symbol	Parameter	8279		8279-5		Unit
		Min.	Max.	Min.	Max.	
t _{AR}	Address Stable Before READ	50		0		ns
t _{RA}	Address Hold Time for READ	5		0		ns
t _{RR}	READ Pulse Width	420		250		ns
t _{RD} ^[4]	Data Delay from READ		300		150	ns
t _{AD} ^[4]	Address to Data Valid		450		250	ns
t _{DF}	READ to Data Floating	10	100	10	100	ns
t _{RCY}	Read Cycle Time	1		1		μs

A.C. CHARACTERISTICS (Continued)**WRITE CYCLE**

Symbol	Parameter	8279		8279-5		Unit
		Min.	Max.	Min.	Max.	
t_{AW}	Address Stable Before WRITE	50		0		ns
t_{WA}	Address Hold Time for WRITE	20		0		ns
t_{WW}	WRITE Pulse Width	400		250		ns
t_{DW}	Data Set Up Time for WRITE	300		150		ns
t_{WD}	Data Hold Time for WRITE	40		0		ns
t_{WCY}	Write Cycle Time	1		1		μ s

OTHER TIMINGS

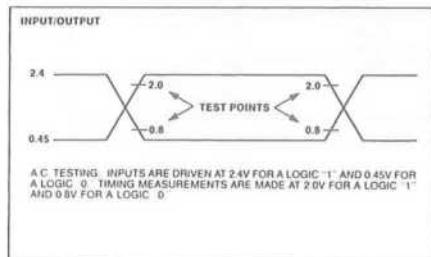
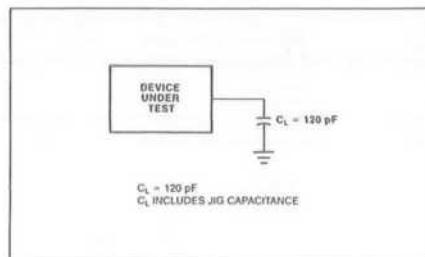
Symbol	Parameter	8279		8279-5		Unit
		Min.	Max.	Min.	Max.	
$t_{\phi W}$	Clock Pulse Width	230		120		nsec
t_{CY}	Clock Period	500		320		nsec

Keyboard Scan Time 5.1 msec
 Keyboard Debounce Time 10.3 msec
 Key Scan Time 80 μ sec
 Display Scan Time 10.3 msec

Digit-on Time 480 μ sec
 Blanking Time 160 μ sec
 Internal Clock Cycle^[5] 10 μ sec

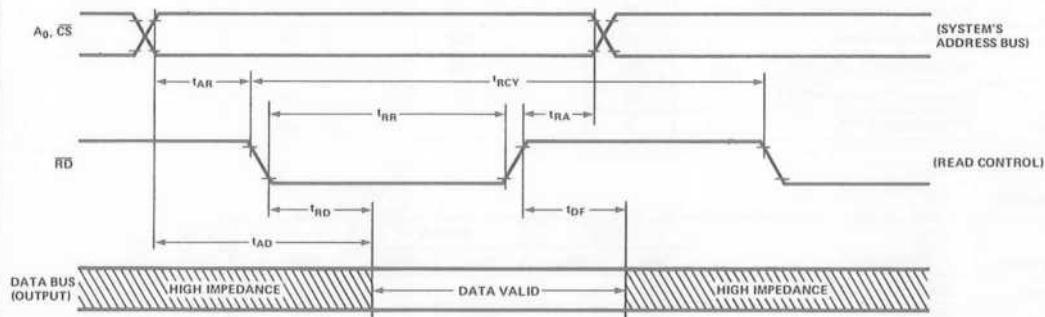
NOTES:

1. 8279, $I_{OL} = 1.6mA$; 8279-5, $I_{OL} = 2.2mA$.
2. $I_{OH} = -100 \mu A$
3. 8279, $V_{CC} = +5V \pm 5\%$; 8279-5, $V_{CC} = +5V \pm 10\%$.
4. 8279, $C_L = 100pF$; 8279-5, $C_L = 150pF$.
5. The Prescaler should be programmed to provide a 10 μ s internal clock cycle.
 * For Extended Temperature EXPRESS, use M8279A electrical parameters.

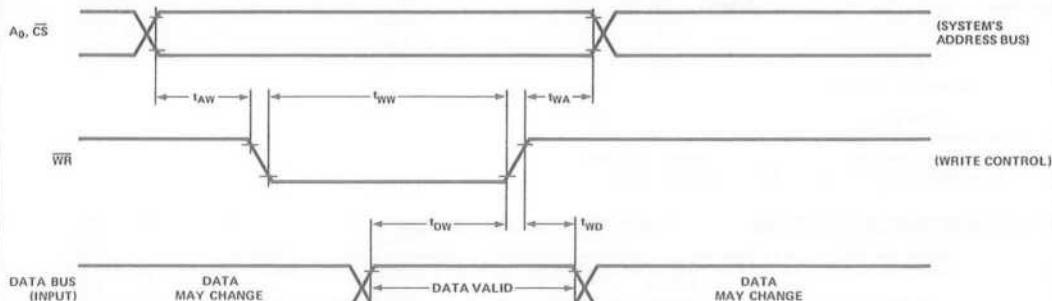
A.C. TESTING INPUT, OUTPUT WAVEFORM**A.C. TESTING LOAD CIRCUIT**

WAVEFORMS

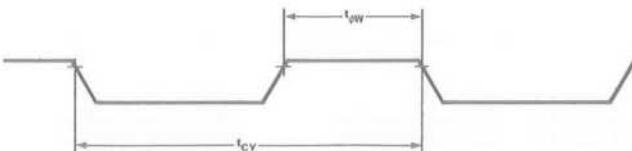
READ OPERATION



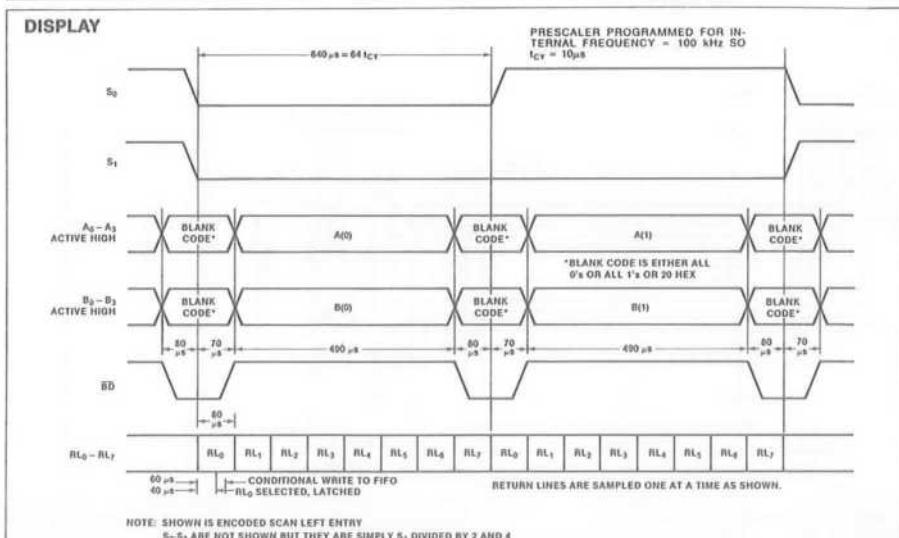
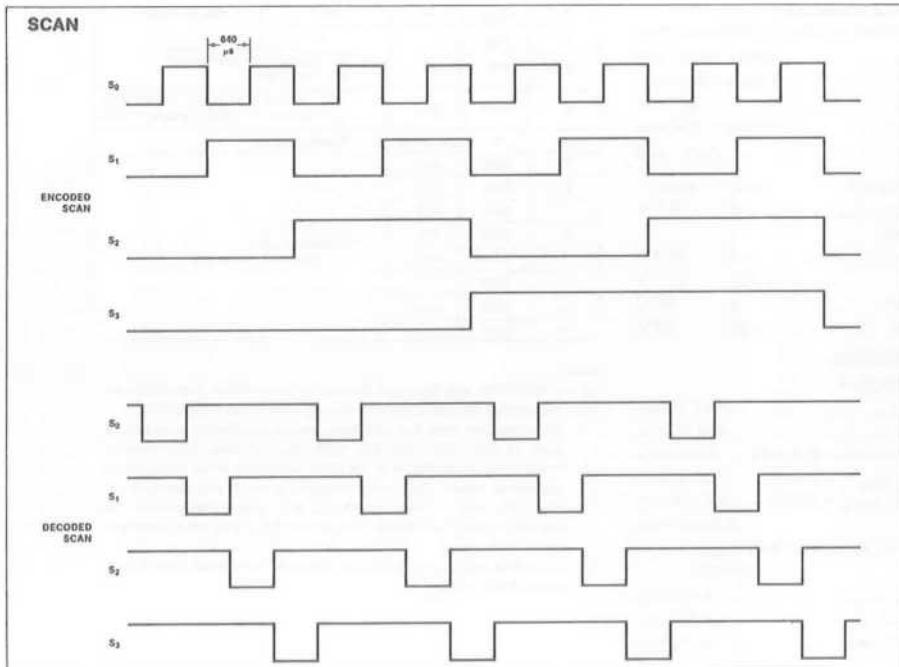
WRITE OPERATION



CLOCK INPUT



WAVEFORMS (Continued)



LM061L

SUMMARY

- 20 Character x 2 lines
- Built-in control LSI HD44780 type (see section 6).
- +5V single power supply

MECHANICAL DATA (Nominal dimensions)

Module size	115W x 39H x 13D (max.) mm
Effective display area	83W x 18.6H mm
Character size (5 x 7 dots)	3.2W x 4.85H mm
Pitch	3.7 mm
Dot size	0.6W x 0.65H mm
Weight	about 50 g

ABSOLUTE MAXIMUM RATINGS	min.	max.
Power supply for logic (V_{DD} – V_{SS})	0	6.5 V
Power supply for LCD drive (V_{DD} – V_O)	0	6.5 V
Input voltage (V_i)	V_{SS}	V_{DD} V
Operating temperature (T_a)	0	50°C
Storage temperature (T_{stg})	-20	70°C

ELECTRICAL CHARACTERISTICS

Ta = 25°C, V_{DD} = 5.0 V ± 0.25 V	
Input "high" voltage (V_{iH})	2.2 V min.
Input "low" voltage (V_{iL})	0.6 V max.
Output high voltage (V_{OH}) ($I_{OH} = 0.2$ mA)	2.4 V min.
Output low voltage (V_{OL}) ($I_{OL} = 1.2$ mA)	0.4 V max.
Power supply current (I_{DD}) ($V_{DD} = 5.0$ V)	1.0 mA typ. 3.0 mA max.
Power supply for LCD drive (Recommended) (V_{DD} – V_O)	
$D_u=1/16$	
at $T_a = 0^\circ\text{C}$	4.6 V typ.
at $T_a = 25^\circ\text{C}$	4.4 V typ.
at $T_a = 50^\circ\text{C}$	4.2 V typ.

OPTICAL DATA See page 15.

INTERNAL PIN CONNECTION

Pin No.	Symbol	Level	Function
1	V_{SS}	—	0V Power supply
2	V_{DD}	—	
3	V_O	—	L: Instruction code input H: Data input
4	RS	H/L	
5	R/W	H/L	H: Data read (LCD module → MPU) L: Data write (LCD module ← MPU)
6	E	H, H → L	
7	DB0	H/L	Data bus line Note (1), Note (2)
8	DB1	H/L	
9	DB2	H/L	
10	DB3	H/L	
11	DB4	H/L	
12	DB5	H/L	
13	DB6	H/L	
14	DB7	H/L	

Note:

In the HD44780, the data can be sent in either 4-bit 2-operation or 8-bit 1-operation so that it can interface to both 4 and 8 bit MPU's.

- (1) When interface data is 4 bits long, data is transferred using only 4 buses of DB_4 ~ DB_7 , and DB_0 ~ DB_3 , are not used. Data transfer between the HD44780 and the MPU completes when 4-bit data is transferred twice. Data of the higher order 4 bits (contents of DB_4 ~ DB_7 , when interface data is 8 bits long) is transferred first and then lower order 4 bits (contents of DB_0 ~ DB_3 , when interface data is 8 bits long).
- (2) When interface data is 8 bits long, data is transferred using 8 data buses of DB_0 ~ DB_7 .

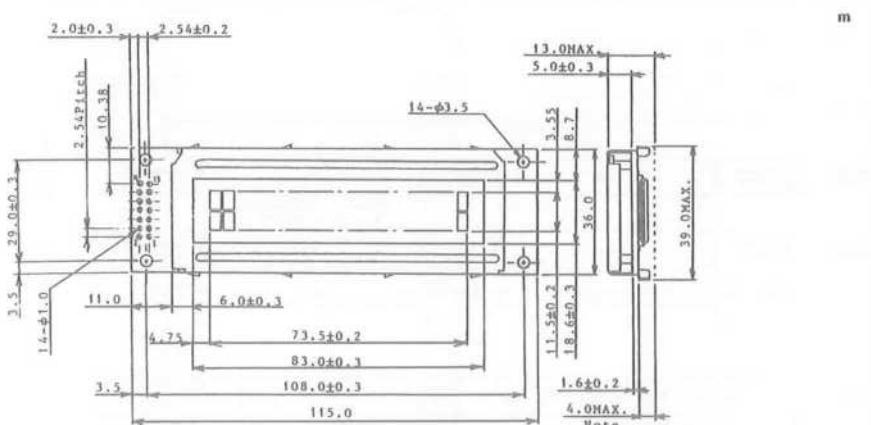


Fig. 2 External dimensions

TIMING CHARACTERISTICS

Item	Symbol	Test condition	Min.	Typ.	Max.	Unit
Enable cycle time	t_{cyc}	Fig. 5, Fig. 6	1.0	—	—	μs
Enable pulse width	P_{wEH}	Fig. 5, Fig. 6	450	—	—	ns
Enable rise/fall time	t_{ER}, t_{EF}	Fig. 5, Fig. 6	—	—	25	ns
RS, R/W set up time	t_{AS}	Fig. 5, Fig. 6	140	—	—	ns
Data delay time	t_{DDR}	Fig. 6	—	—	320	ns
Data set up time	t_{DSW}	Fig. 5	195	—	—	ns
Hold time	t_H	Fig. 5, Fig. 6	20	—	—	ns

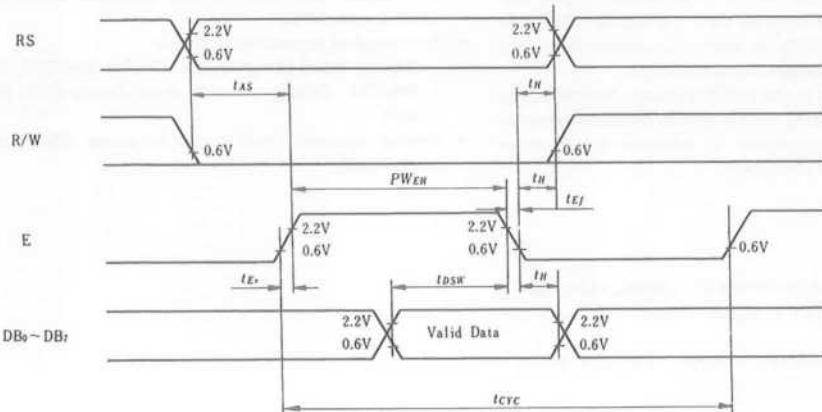


Fig. 5 Interface timing (data write)

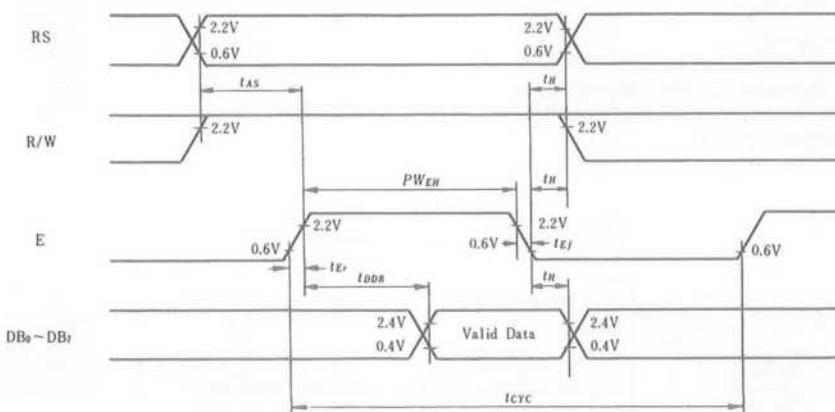


Fig. 6 Interface timing (data read)

HOW TO USE HITACHI'S BUILT-IN CONTROLLER DRIVER

LCD-II (HD44780) DOT MATRIX LCD MODULE

■ INTRODUCTION

The LCD-II (HD44780) is a dot matrix liquid crystal display controller & driver LSI that displays alphanumerics, kana characters and symbols. It drives dot matrix liquid crystal display under 4-bit or 8-bit microcomputer or microprocessor control. All the functions required for dot matrix liquid crystal display drive are internally provided on one chip.

The user can complete dot matrix liquid crystal display systems with less number of chips by using the LCD-II (HD44780).

If a driver LSI HD44100H is externally connected to the HD44780, up to 80 characters can be displayed.

The LCD-II is produced in the CMOS process. Therefore, the combination of the LCD-II with a CMOS microcomputer or microprocessor can accomplish a portable battery-drive device with lower power dissipation.

■ FEATURES

- Capable of interfacing to 4-bit or 8-bit MPU.
- Display data RAM 80 x 8 bits
(80 characters, max.)
- Character generator ROM
 - Character font 5 x 7 dots: 160 characters
 - Character font 5 x 10 dots: 32 characters
- Both display data and character generator RAMs can be read from the MPU.
- Wide range of instruction functions
 - Display clear, Cursor home, Display ON/OFF, Cursor ON/OFF, Display character blink, Cursor shift, Display shift
- Internal automatic reset circuit at power ON. (Internal reset circuit)

1. Applicable type

- (1) 1 line series
 - LM054 • H2570 • LM015 • LM568AF • LM020L • LM070L • LM038 • LM027 • H2571 • LM058
- (2) 2 line series
 - LM052L • LM016L • LM032L • LM060L • LM017L • LM018L • LM075L • LM074L • LM068L • LM061L
- (3) 4 lines series
 - LM041L • LM044L
- (4) Compact version
 - LM104L • LM105L • LM107L

2. Connecting MPU with LCM

2.1 Driver circuit block diagram

Figure 1 shows the driver circuit block diagram of LCM with built-in controller LSI. Controller LSI HD44780 (LCD-II) is built-in this LCM. Also extended LCD driver LSI is built in the LCM that displays more than 16 digits.

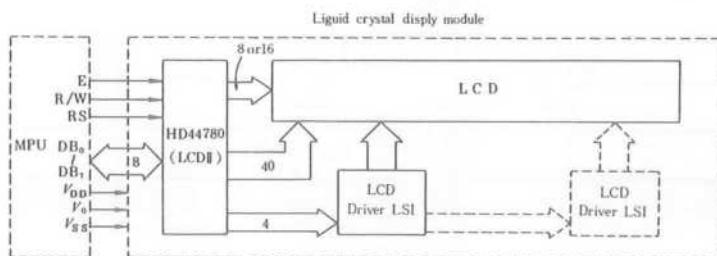


Fig. 1 Driver circuit block diagram

2.2 Interfacing to MPU

In the HD44780, data can be sent in either 4-bit 2-operation or 8-bit 1-operation so it can interface to both 4 and 8 bit MPU's.

- (1) When interface data is 4-bits long, data is transferred using only 4 buses: DB₄ ~ DB₇. DB₀ ~ DB₃ are not used. Data transfer between the HD44780 and the MPU completes when 4-bit data is transferred twice. Data of the higher order 4 bits (contents of DB₄ ~ DB₇ when interface data is 8 bits long) is transferred first, then the

lower order 4 bits (content of DB₀ ~ DB₃ when interface data is 8 bits long) is transferred. Check the busy flag after 4-bit data has been transferred twice (one instruction). A 4-bit 2-operation will then transfer the busy flag and address counter data.

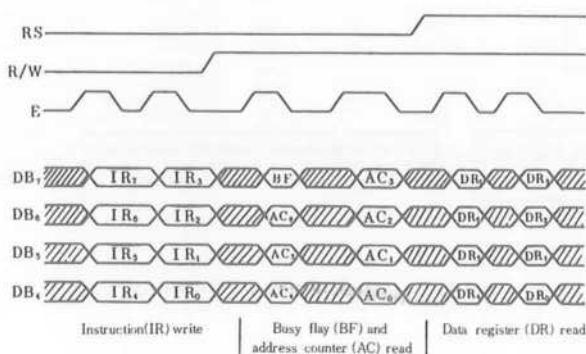


Fig. 2 4-bit data transfer example

- (2) When interface data is 8 bit long, data is transferred using the 8 data buses of DB₀ ~ DB₇.

2.3 Interface to MPU

(1) Interface to 8-bit MPU

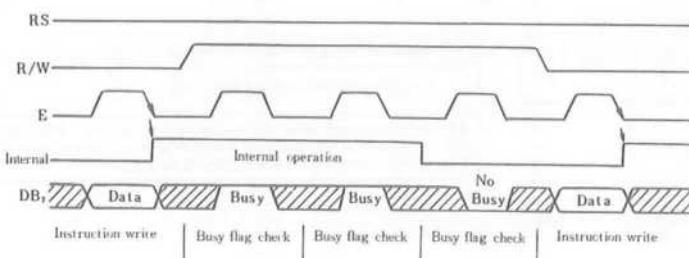


Fig. 3 Example of busy flag check timing sequence

Initializing by instruction

If the power supply conditions for correctly operating the internal reset circuit are not met, initialization by instruction is required.

Use the following procedure for initialization.

(1) When interface is 8 bits long;

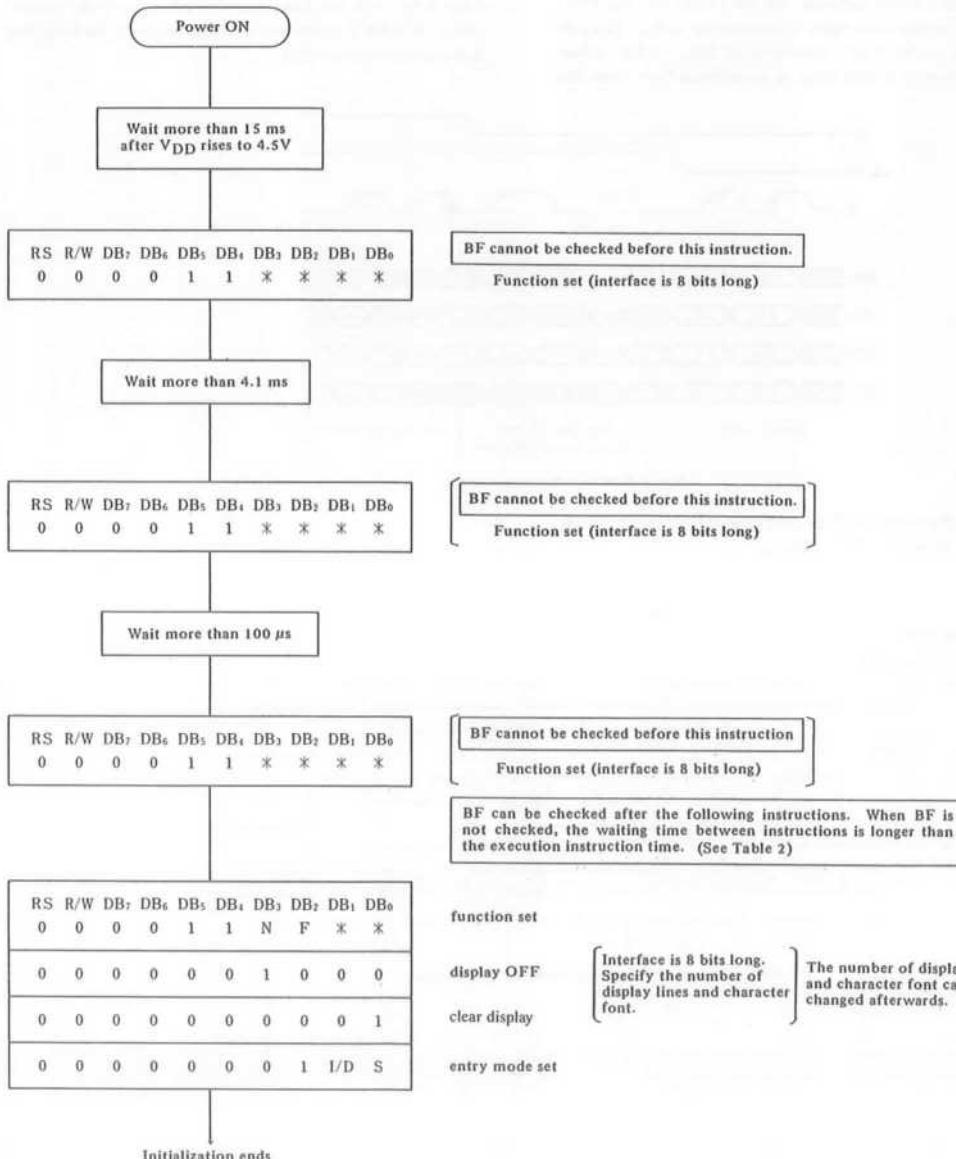


Table 2 Instructions

Instruction	Code										Description	Execution time (when fosc is 250 kHz) Note 1	Execution time (when fosc is 160 kHz) Note 2
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0			
Clear display	0	0	0	0	0	0	0	0	0	1	Clears all display and returns the cursor to the home position (Address 0).	82 µs ~ 1.64 ms	120 µs ~ 4.9 ms
Return home	0	0	0	0	0	0	0	0	1	*	Returns the cursor to the home position (Address 0). Also returns the display being shifted to the original position. DD RAM contents remain unchanged.	40 µs ~ 1.6 ms	120 µs ~ 4.8 ms
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Sets the cursor move direction and specifies or not to shift the display. These operations are performed during data write and read.	40 µs	120 µs
Display ON/OFF control	0	0	0	0	0	0	1	D	C	B	Sets ON/OFF of all display (D), cursor ON/OFF (C), and blink of cursor position character (B).	40 µs	120 µs
Cursor and display shift	0	0	0	0	0	1	S/C	R/L	*	*	Moves the cursor and shifts the display without changing DD RAM contents.	40 µs	120 µs
Function set	0	0	0	0	1	DL	N	F	*	*	Sets interface data length (DL), number of display lines (L) and character font (F).	40 µs	120 µs
Set CG RAM address	0	0	0	1	ACG						Sets the CG RAM address. CG RAM data is sent and received after this setting.	40 µs	120 µs
Set DD RAM address	0	0	1	ADD							Sets the DD RAM address. DD RAM data is sent and received after this setting.	40 µs	120 µs
Read busy flag & address	0	1	BF	AC							Reads Busy flag (BF) indicating internal operation is being performed and reads address counter contents.	1 µs	1 µs
Write data to CG or DD RAM	1	0	Write Data								Writes data into DD RAM or CG RAM.	40 µs	120 µs
Read data to CG or DD RAM	1	1	Read Data								Reads data from DD RAM or CG RAM.	40 µs	120 µs
	I/D = 1: Increment (+1) I/D = 0: Decrement (-1) S = 1: Accompanies display shift. S/C = 1: Display shift S/C = 0: Cursor move R/L = 1: Shift to the right. R/L = 0: Shift to the left. DL = 1: 8 bits DL = 0: 4 bits N = 1: 2 lines N = 0: 1 line F = 1: 5 x 10 dots F = 0: 5 x 7 dots BF = 1: Internally operating BF = 0: Can accept instruction										DD RAM: Display data RAM CG RAM: Character generator RAM ACG: CG RAM address ADD: DD RAM address A: Corresponds to cursor address. AC: Address counter used for both of DD and CG RAM address.	Execution time changes when frequency changes. (Example) When fosc is 270 kHz: $40 \mu s \times \frac{250}{270} = 37 \mu s$	

*No effect

Notes 1. Applied to models driven by 1/8 duty or 1/11 duty.
 2. Applied to models driven by 1/16 duty.

Description of details**(1) Clear display**

Code	RS	R/W	DB ₇	DB ₀
	0	0	0	0 0 0 0 0 0 0 1

Writes space code "20" (hexadecimal) (character pattern for character code "20" must be blank pattern) into all DD RAM addresses. Sets DD RAM address 0 in address counter. Returns display to its original status if it was shifted. In other words, the display disappears and the cursor or blink go to the left edge of the display (the first line if 2 lines are displayed). Set I/D = 1 (Increment Mode) of Entry Mode. S of Entry Mode doesn't change.

(2) Return home

Code	RS	R/W	DB ₇	DB ₀
	0	0	0	0 0 0 0 0 0 0 0 *

* No effect

Sets the DD RAM address 0 in address counter. Returns display to its original status if it was shifted. DD RAM contents do not change. The cursor or blink go to the left edge of the display (the first line if 2 lines are displayed).

(3) Entry mode set

Code	RS	R/W	DB ₇	DB ₀
	0	0	0	0 0 0 0 0 0 0 1/D S

I/D: Increments (I/D = 1) or decrements (I/D = 0) the DD RAM address by 1 when a character code is written into or read from the DD RAM. The cursor or blink moves to the right when incremented by 1 and to the left when decremented by 1. The same applies to writing and reading of CG RAM.

S: Shifts the entire display either to the right or to the left when S is 1; to the left when I/D = 1 and to the right when I/D = 0. Thus it looks as if the cursor stands still and the display moves. The display does not shift when reading from the DD RAM when writing into or reading out from the CG RAM does it shift when S = 0.

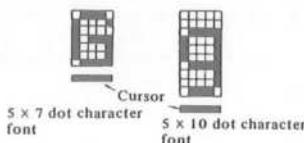
(4) Display ON/OFF control

Code	RS	R/W	DB ₇	DB ₀
	0	0	0	0 0 0 0 0 1 D C B

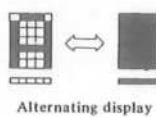
D: The display is ON when D = 1 and OFF when D = 0. When off due to D = 0, display data remains in the DD RAM. It can be displayed immediately by setting D = 1.

C: The cursor displays when C = 1 and does not display when C = 0. Even if the cursor disappears, the function of I/D, etc. does not change during display data write. The cursor is displayed using 5 dots in the 8th line when the 5 × 7 dot character font is selected and 5 dots in the 11th line when the 5 × 10 dot character font is selected.

B: The character indicated by the cursor blinks when B = 1. The blink is displayed by switching between all blank dots and display characters at 409.6 ms interval when f_{CP} or f_{osc} = 250 kHz. The cursor and the blink can be set to display simultaneously. (The blink frequency changes according to the reciprocal of f_{CP} or f_{osc}. $409.6 \times \frac{250}{270} = 379.2$ ms when f_{CP} = 270 kHz.)



(a) Cursor Display Example



(b) Blink Display Example

(5) Cursor or display shift

Code	RS	R/W	DB ₇	DB ₀
	0	0	0	0 1 S/C R/L * *

* No effect

Shifts cursor position or display to the right or left without writing or reading display data. This function is used to correct or search for the display. In a 2-line display, the cursor moves to the 2nd line when it passes the 40th digit of the 1st line. Notice that the 1st and 2nd line displays will shift at the same time. When the displayed data is shifted repeatedly each line only moves horizontally. The 2nd line display does not shift into the 1st line position.

S/C R/L

0	0	Shifts the cursor position to the left. (AC is decremented by one.)
0	1	Shifts the cursor position to the right. (AC is incremented by one.)
1	0	Shifts the entire display to the left. The cursor follows the display shift.
1	1	Shifts the entire display to the right. The cursor follows the display shift.

Address counter (AC) contents do not change if the only action performed is shift display.

(6) Function set

Code	RS	R/W	DB ₇	DB ₆	DB ₅	DB ₄	DB ₃	DB ₂	DB ₁	DB ₀
	0	0	0	0	1	DL	N	F	*	*

* No effect

DL: Sets interface data length. Data is sent or received in 8 bit lengths (DB₇ ~ DB₀) when DL = 1 and in 4 bit lengths (DB₇ ~ DB₄) when DL = 0.
When the 4 bit length is selected, data must be sent or received twice.

N: Sets number of display lines.

F: Sets character font.

(Note) Perform the function at the head of the program before executing all instructions (except "Busy flag/address read"). From this point, the function set instruction cannot be executed unless the interface data length is changed.

N	F	No. of display lines	Character font	Duty factor	Remarks
0	0	1	5 x 7 dots	1/8	
0	1	1	5 x 10 dots	1/11	
1	*	2	5 x 7 dots	1/16	Cannot display 2 lines with 5 x 10 dot character font.

* No effect

(7) Set CG RAM address

Code	RS	R/W	DB ₇	DB ₆	DB ₅	DB ₄	DB ₃	DB ₂	DB ₁	DB ₀
	0	0	0	1	A	A	A	A	A	A

← Higher Order Bits → Lower Order Bits

Sets the CG RAM address into the address counter in binary AAAAAAA. Data is then written or read from the MPU for the CG RAM.

(8) Set DD RAM address

Code	RS	R/W	DB ₇	DB ₆	DB ₅	DB ₄	DB ₃	DB ₂	DB ₁	DB ₀
	0	0	1	A	A	A	A	A	A	A

← Higher Order Bits → Lower Order Bits

Sets the DD RAM address into the address counter in binary AAAAAAA. Data is then written or read from the MPU for the DD RAM.

However, when N = 0 (1-line display), AAAAAAA is "00" ~ "4F" (hexadecimal), when N = 1 (2-line display), AAAAAAA is "00" ~ "27" (hexadecimal) for the first line, and "40" ~ "67" (hexadecimal) for the second line.

(9) Read busy flag & address

Code	RS	R/W	DB ₇	DB ₆	DB ₅	DB ₄	DB ₃	DB ₂	DB ₁	DB ₀
	0	1	BF	A	A	A	A	A	A	A

← Higher Order Bits → Lower Order Bits

Reads the busy flag (BF) that indicates the system is now internally operating by a previously received instruction. BF = 1 indicates that internal operation is in progress. The next instruction will not be accepted until BF is set to "0". Check the BF status before the next wire operation.

At the same time, the value of the address counter expressed in binary AAAAAAA is read out. The address counter is used by both CG and DD RAM addresses, and its value is determined by the previous instruction. Address contents are the same as in Items (7) and (8).

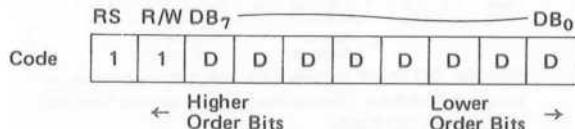
(10) Write data to CG or DD RAM

Code	RS	R/W	DB ₇	DB ₆	DB ₅	DB ₄	DB ₃	DB ₂	DB ₁	DB ₀
	1	0	D	D	D	D	D	D	D	D

← Higher Order Bits → Lower Order Bits

Writes binary 8 bit data DDDDDDD to the CG or the DD RAM. Whether the CG or DD RAM is to be written into is determined by the previous specification of CG RAM or DD RAM address setting. After write, the address is automatically incremented or decremented by 1 according to entry mode. The entry mode also determines display shift.

(11) Read data from CG or DD RAM



Reads binary 8 bit data DDDDDDDD from the CG or DD RAM. The previous designation determines whether the CG or DD RAM is to be read. Before entering the read instruction, you must execute either the CG RAM or DD RAM address set instruction. If you don't, the first read data will be invalidated. When serially executing the "read" instruction, the next address data is normally read from the second read. The "address set" instruction need not be executed just before the "read" instruction when shifting the cursor by cursor shift instruction (when reading out DD RAM). The cursor shift instruction operation is the same as that of the DD RAM's address set instruction.

After a read, the entry mode automatically increases or decreases the address by 1. However, display shift is not executed no matter what the entry mode is.

(Note) The address counter (AC) is automatically incremented or decremented by 1 after "write" instructions to either CG RAM or DD RAM. RAM data selected by the AC cannot than be read out even if "read" instructions are executed. The conditions for correct data read out are: execute either the address set instruction or cursor shift instruction (only with DD RAM), just before reading out execute the "read" instruction from the second time the "read" instruction is serial.

(3) 8-bit operation, 8-digit x 2-line display

For 2-line display, the cursor automatically moves from the first to the second line after the 40th digit of the 1st line has been written. Thus, if there are only 8 characters in the first line, the DD RAM address must again be set after the 8th character is completed. (See the following table) Note that the first and second lines of the display

shift are performed. In the example, the display shift is performed when the cursor is on the second line. However, if shift operation is performed when the cursor is on the first line, both the first and second lines move together. When you repeat the shift, the display of the second display will only move within each line many times.

8 bit operation, 8-digit x 2-line display example (using internal reset)

No.	Instruction	Display	Operation
1	Power supply ON (HD44780 is initialized by the internal reset circuit)	[]	Initialized. No display appears.
2	Function Set RS R/W DB ₇ 0 0 0 0 1 1 1 0 * DB ₀ *	[]	Sets to 8-bit operation and selects 2-line display and 5 x 7 dot character font.
3	Display ON/OFF Control 0 0 0 0 0 1 1 1 0	[]	Turns on display and cursor. All display is in space mode because of initialization.
4	Entry Mode Set 0 0 0 0 0 0 0 1 1 0	[]	Sets mode to increment the address by one and to shift the cursor to the right, at the time of write, to the DD/CG RAM. Display is not shifted.
5	Write Data to CG RAM/DD RAM 1 0 0 1 0 0 1 0 0 0	H []	Write "H". The DD RAM has already been selected by initialization when the power is turned on. The cursor is incremented by one and shifted to the right.
6	[]	[]	
7	Write Data to CG RAM/DD RAM 1 0 0 1 0 0 1 0 0 1	H I T A C H I []	Writes "I".
8	Set DD RAM Address 0 0 1 1 0 0 0 0 0 0	H I T A C H I []	Sets RAM address so that the cursor is positioned at the head of the 2nd line.
9	Write Data to CG RAM/DD RAM 1 0 0 1 0 0 1 1 0 1	H I T A C H I M []	Writes "M".
10	[]	[]	
11	Write Data to CG RAM/DD RAM 1 0 0 1 0 0 1 1 1 1	H I T A C H I M I C R O C O []	Writes "O".
12	Entry Mode Set 0 0 0 0 0 0 0 1 1 1	H I T A C H I M I C R O C O []	Sets mode for display shift at the time of write.
13	Write Data to CG RAM/DD RAM 1 0 0 1 0 0 1 1 0 1	I T A C H I I C R O C O M []	Writes "M". Display is shifted to the right. The first and second lines' shift are operated at the same time.
14	[]	[]	
15	Return Home 0 0 0 0 0 0 0 0 1 0	H I T A C H I M I C R O C O M []	Returns both display and cursor to the original position (Address 0).

E

American Standard Code for Information Interchange: ASCII Codes

Graphic or Control	ASCII (Hexadecimal)	Graphic or Control	ASCII (Hexadecimal)
NUL Null	00	DC4 Device Control 4	14
SOH Start of Heading	01	NAK Negative Acknowledge	15
STX Start of Text	02	SYN Synchronous Idle	16
ETX End of Text	03	ETB End of Transmission Block	17
EOT End of Transmission	04	CAN Cancel	18
ENQ Enquiry	05	EM End of Medium	19
ACK Acknowledge	06	SUB Substitute	1A
BEL Bell	07	ESC Escape	1B
BS Backspace	08	FS File Separator	1C
HT Horizontal Tabulation	09	GS Group Separator	1D
LF Line Feed	0A	RS Record Separator	1E
VT Vertical Tabulation	0B	US Unit Separator	1F
FF Form Feed	0C	SP Space	20
CR Carriage Return	0D	!	21
SO Shift Out	0E	"	22
SI Shift In	0F	#	23
DLE Data Link Escape	10	\$	24
DC1 Device Control 1	11	%	25
DC2 Device Control 2	12	&	26
DC3 Device Control 3	13	'	27

Graphic or Control	ASCII (Hexadecimal)	Graphic or Control	ASCII (Hexadecimal)
(28	T	54
)	29	U	55
*	2A	V	56
+	2B	W	57
,	2C	X	58
-	2D	Y	59
.	2E	Z	5A
/	2F	[5B
0	30	\	5C
1	31]	5D
2	32	^	5E
3	33	-	5F
4	34	a	60
5	35	b	61
6	36	c	62
7	37	d	63
8	38	e	64
9	39	f	65
:	3A	g	66
;	3B	h	67
<	3C	i	68
=	3D	j	69
>	3E	k	6A
?	3F	l	6B
@	40	m	6C
A	41	n	6D
B	42	o	6E
C	43	p	6F
D	44	q	70
E	45	r	71
F	46	s	72
G	47	t	73
H	48	u	74
I	49	v	75
J	4A	w	76
K	4B	x	77
L	4C	y	78
M	4D	z	79
N	4E	{	7A
O	4F		7B
P	50	}	7C
Q	51	~	7D
R	52	DEL Delete	7E
S	53		7F

F

8085 Instruction Set

Appendix F describes each instruction fully in terms of its operation and the operand, including details such as number of bytes, machine cycles, T-states, Hex code, and affected flags. The instructions appear in alphabetical order and are illustrated with examples.

The following abbreviations are used in the description of the instruction set.

Flags	
Reg. = 8080A/8085 Register	S = Sign
Mem. = Memory Location	Z = Zero
R = Register	AC = Auxiliary Carry
Rs = Register Source	P = Parity
Rd = Register Destination	CY = Carry
M = Memory	
() = Contents of	
XX = Random Information	

ACI: Add Immediate to Accumulator with Carry

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
ACI	8-bit data	2	2	7	CE

Description The 8-bit data (operand) and the Carry flag are added to the contents of the accumulator, and the result is stored in the accumulator.

Flags All flags are modified to reflect the result of the addition.

Example Assuming the accumulator contains 26H and the previous operation has set the Carry flag, add byte 57H to the accumulator.

Instruction: ACI 57H Hex Code: CE 57

Addition:

$$\begin{array}{l}
 \text{(A): } 26H = 0\ 0\ 1\ 0\ 0\ 1\ 1\ 0 \\
 \text{(Data): } 57H = 0\ 1\ 0\ 1\ 0\ 1\ 1\ 1 \\
 \text{CY 1 = } \frac{}{1} \\
 \text{7EH = } 0\ 1\ 1\ 1\ 1\ 1\ 1\ 0 \\
 \text{Flags: } S = 0\ Z = 0\ AC = 0 \\
 \quad \quad \quad P = 1\ CY = 0
 \end{array}$$

Comments:

1. After addition the previous Carry flag is cleared.
2. This instruction is commonly used in 16-bit addition. This instruction should not be used to account for a carry generated by 8-bit numbers.

ADC: Add Register to Accumulator with Carry

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Codes	
ADC	Reg.	1	1	4	Reg.	Hex
	Mem.	1	2	7	B	88
					C	89
					D	8A
					E	8B
					H	8C
					L	8D
					M	8E
					A	8F

Description The contents of the operand (register or memory) and the Carry flag are added to the contents of the accumulator and the result is placed in the accumulator. The contents of the operand are not altered; however, the previous Carry flag is reset.

Flags All flags are modified to reflect the result of the addition.

Example Assume register pair BC contains 2498H and register pair DE contains 54A1H. Add these 16-bit numbers and save the result in BC registers.

The steps in adding 16-bit numbers are as follows:

1. Add the contents of registers C and E by placing the contents of one register in the accumulator. This addition generates a Carry. Use instruction ADD (explained on the next page) and save the low-order 8-bits in register C.

98H =	1 0 0 1	1 0 0 0	
AIH =	1 0 1 0	0 0 0 1	
1 39H =	1 0 0 1 1	1 0 0 1	
CY	CY		Store in register C

2. Add the contents of registers B and D by placing the contents of one register in the accumulator. Use instruction ADC.

The result will be as follows.

$$\begin{array}{r}
 24H = 0\ 0\ 1\ 0\ 0\ 1\ 0\ 0 \\
 54H = 0\ 1\ 0\ 1\ 0\ 1\ 0\ 0 \\
 \hline
 79H = 0\ 1\ 1\ 1\ 1\ 0\ 0\ 1
 \end{array}
 \quad \begin{array}{l}
 \text{(Carry from the previous addition)} \\
 \text{Store in register B}
 \end{array}$$

Comments: This instruction is generally used in 16-bit addition. For example, to add the contents of BC registers to the contents of DE registers this instruction is used to account for the carry generated by low-order bytes.

ADD: Add Register to Accumulator

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Codes	
ADD	Reg.	1	1	4	Reg.	Hex
	Mem.	1	2	7	B	80
					C	81
					D	82
					E	83
					H	84
					L	85
					M	86
					A	87

Description The contents of the operand (register or memory) are added to the contents of the accumulator and the result is stored in the accumulator. If the operand is a memory location, that is indicated by the 16-bit address in the HL register.

Flags All flags are modified to reflect the result of the addition.

Example Register B has 51H and the accumulator has 47H. Add the contents of register B to the contents of the accumulator.

Instruction: ADD B Hex Code: 80

Register contents
before instruction

A	47	X	F
B	51	X	C

Addition

$$\begin{array}{r}
 47H = 0\ 1\ 0\ 0 \quad 0\ 1\ 1\ 1 \\
 51H = 0\ 1\ 0\ 1 \quad 0\ 0\ 0\ 1 \\
 \hline
 98H = 1\ 0\ 0\ 1 \quad 1\ 0\ 0\ 0
 \end{array}$$

Register contents
after instruction

	SZ	AC	P	CY	
A	98	1,0,0,0,0			F
B	51		X		C

Flags: S = 1, Z = 0, AC = 0
P = 0, CY = 0

Example Memory location 2050H has data byte A2H and the accumulator has 76H. Add the contents of the memory location to the contents of the accumulator.

Instruction: ADD M Hex Code: 86

Before this instruction is executed, registers HL should be loaded with data 2050H.

Register contents
before instruction

A	76	X
B	X	X
D	X	X
H	20	50

F
C
E
L

2050 [A2]

Addition:

Register contents
after instruction

		S Z AC P CY
(A)	76H = 0 1 1 1 0 1 1 0	A
(2050H) _{Mem}	A2H = 1 0 1 0 0 0 1 0	B
	1/18H = 1/0 0 0 1 1 0 0 0	C
	CY CY	D
		E
		F
		G
		H

Flags: S = 0, Z = 0, AC = 0,
P = 1, CY = 1

ADI: Add Immediate to Accumulator

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Codes
ADI	8-bit data	2	2	7	C6

Description The 8-bit data (operand) are added to the contents of the accumulator, and the result is placed in the accumulator.

Flags All flags are modified to reflect the result of the addition.

Example The accumulator contains 4AH. Add the data byte 59H to the contents of the accumulator.

Instruction: ADI 59H Hex Code: C6 59

Addition:

$$\begin{array}{r}
 (A) : 4AH = 0\ 1\ 0\ 0\ 1\ 0\ 1\ 0 \\
 + \\
 (\text{Data}) : 59H = 0\ 1\ 0\ 1\ 1\ 0\ 0\ 1 \\
 \hline
 A3H = 1\ 0\ 1\ 0\ 0\ 0\ 1\ 1
 \end{array}$$

Flags: S = 1, Z = 0, AC = 1
P = 1, CY = 0

ANA: Logical AND with Accumulator

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Codes	
ANA	Reg.	1	1	4	Reg. B C D E H L M	Hex A0 A1 A2 A3 A4 A5 A6
	Mem.	1	2	7		
					A	A7

Description The contents of the accumulator are logically ANDed with the contents of the operand (register or memory), and the result is placed in the accumulator. If the operand is a memory location, its address is specified by the contents of HL registers.

Flags S, Z, P are modified to reflect the result of the operation. CY is reset. In 8085, AC is set, and in 8080A AC is the result of ORing bits D₃ of the operands.

Example The contents of the accumulator and the register D are 54H and 82H, respectively. Logically AND the contents of register D with the contents of the accumulator. Show the flags and the contents of each register after ANDing.

Instruction: ANA D Hex Code: A2

Register contents before instruction	Logical AND	Register contents after instruction
A 54 X F	$54H = 0\ 1\ 0\ 1$ AND $82H = 1\ 0\ 0\ 0$ $\hline 0\ 0\ 0\ 0$	$0\ 1\ 0\ 0$
B 82 X E	$0\ 0\ 1\ 0$	A 00 [0,1,1,1,0] F D 82 [] E

Flags: S = 0, Z = 1, P = 1
AC = 1, CY = 0
(for 8080A, AC = 0)

ANI: AND Immediate with Accumulator

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
ANI	8-bit data	2	2	7	E6

Description The contents of the accumulator are logically ANDed with the 8-bit data (operand) and the results are placed in the accumulator.

Flags S, Z, P are modified to reflect the results of the operation. CY is reset. In 8085, AC is set.

Example AND data byte 97H with the contents of the accumulator, which contains A3H.

Instruction: ANI 97H Hex Code: E6 97

Logical AND:

(A) :	A3H = 1 0 1 0 0 0 1 1					
		AND				
(Data) :	97H = 1 0 0 1 0 1 1 1				S Z AC P CY	
	1 0 0 0 0 0 1 1			A	83	[1,0,1,1,0,0] F

CALL: Unconditional Subroutine Call

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
CALL	16-bit address	3	5	18	CD

Description The program sequence is transferred to the address specified by the operand. Before the transfer, the address of the next instruction to CALL (the contents of the program counter) is pushed on the stack. The sequence of events is described in the example below.

Flags No flags are affected.

Example Write CALL instruction at memory location 2010H to call a subroutine located at 2050H. Explain the sequence of events when the stack pointer is at location 2099H.

Memory Address	Hex Code	Mnemonics
2010	CD	CALL 2050H
2011	50	
2012	20	

Note: See the difference between writing a 16-bit address as mnemonics and code. In the code, the low-order byte (50) is entered first, then the high-order byte (20) is entered. However, in mnemonics the address is shown in the proper sequence. If an assembler is used to obtain the codes, it will automatically reverse the sequence of the mnemonics.

Execution of CALL: The address in the program counter (2013H) is placed on the stack as follows.

Stack pointer is decremented to 2098H

MSB is stored

Stack pointer is again decremented

LSB is stored

Call address (2050H) is temporarily stored in internal WZ registers and placed on the bus for the fetch cycle

2097	13
2098	20

SP → 2099

Comments: The CALL instruction should be accompanied by one of the return (RET or conditional return) instructions in the subroutine.

Conditional Call to Subroutine

Op Code	Description	Flag Status	Hex
CC	Call on Carry	CY = 1	DC
CNC	Call with No Carry	CY = 0	D4
CP	Call on positive	S = 0	F4
CM	Call on minus	S = 1	FC
CPE	Call on Parity Even	P = 1	EC
CPO	Call on Parity Odd	P = 0	E4
CZ	Call on Zero	Z = 1	CC
CNZ	Call on No Zero	Z = 0	C4

Operand—16-Bit Address

Code	M-Cycles T-States
DC	2/9 (if condition is not true)
D4	5/18 (if condition is true)
F4	<i>Note:</i> If condition is not true it continues the sequence, and thus requires fewer T-states.
EC	If condition is true it calls the subroutine, thus requires more T-states.
E4	
CC	
C4	

Flags No flags are affected.

CMA: Complement Accumulator

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
CMA	None	1	1	4	2F

Description The contents of the accumulator are complemented.

Flags No flags are affected.

Example Complement the accumulator, which has data byte 89H.

Instruction: CMA Hex Code: 2F

Before instruction		After instruction	
A [1 0 0 0 1 0 0 1]	= 89H	A [0 1 1 1 0 1 1 0]	= 76H

CMC: Complement Carry

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
CMC	None	1	1	4	3F

Description The Carry flag is complemented.

Flags The Carry flag is modified, no other flags are affected.

CMP: Compare with Accumulator

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Codes
CMP	Reg.	1	1	4	Reg. Hex
	Mem.	1	2	7	B B8
					C B9
					D BA
					E BB
					H BC
					L BD
					M BE
					A BF

Description The contents of the operand (register or memory) are compared with the contents of the accumulator. Both contents are preserved and the comparison is shown by setting the flags as follows:

- If $(A) < (\text{Reg/Mem})$: Carry flag is set and Zero flag is reset.
- If $(A) = (\text{Reg/Mem})$: Zero flag is set and Carry flag is reset.
- If $(A) > (\text{Reg/Mem})$: Carry and Zero flags are reset.

The comparison of two bytes is performed by subtracting the contents of the operand from the contents of the accumulator; however, neither contents are modified.

Flags S, P, AC are also modified in addition to Z and CY to reflect the results of the operation.

Example Register B contains data byte 62H and the accumulator contains data byte 57H. Compare the contents of register B with those of the accumulator.

Instruction: CMP B Hex Code: B8

Before instruction After instruction

A	57	XX	F
B	62	XX	C

A	57	1	F
B	62	XX	C

Flags: S = 1, Z = 0, AC = 1
P = 1, CY = 1

Results after executing the instruction:

- No contents are changed.
- Carry flag is set because $(A) < (B)$.
- S, Z, P, AC flags will also be modified as listed above.

CPI: Compare Immediate with Accumulator

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
CPI	8-bit	2	2	7	FE

Description The second byte (8-bit data) is compared with the contents of the accumulator. The values being compared remain unchanged and the results of the comparison are indicated by setting the flags as follows.

- If $(A) < \text{Data}$: Carry flag is set and Zero flag is reset.
- If $(A) = \text{Data}$: Zero flag is set and Carry flag is reset.
- If $(A) > \text{Data}$: Carry and Zero flags are reset.

The comparison of two bytes is performed by subtracting the data byte from the contents of the accumulator; however, neither contents are modified.

Flags S, P, AC are also modified in addition to Z and CY to reflect the result of the operation.

Example Assume the accumulator contains data byte C2H. Compare 98H with the accumulator contents.

Instruction: CPI 98H Hex Code: FE 98

Results after executing the instruction:

- The accumulator contents remain unchanged.
- Z and CY flags are reset because $(A) > \text{Data}$.
- Other flags: S = 0, AC = 0, P = 0.

Example Compare data byte C2H with the contents of the accumulator in the above example.

Instruction: CPI C2H Hex Code: FE C2

Results after executing the instruction:

- The accumulator contents remain unchanged.
- Zero flag is set because $(A) = \text{Data}$.
- Other flags: S = 0, AC = 1, P = 1, CY = 0.

DAA: Decimal-Adjust Accumulator

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
DAA	None	1	1	4	27

Description The contents of the accumulator are changed from a binary value to two 4-bit binary-coded decimal (BCD) digits. This is the only instruction that uses the auxiliary flag (internally) to perform the binary-to-BCD conversion; the conversion procedure is described below.

Flags S, Z, AC, P, CY flags are altered to reflect the results of the operation. Instruction DAA converts the binary contents of the accumulator as follows:

1. If the value of the low-order four bits (D_3-D_0) in the accumulator is greater than 9 or if AC flag is set, the instruction adds 6 (06) to the low-order four bits.
 2. If the value of the high-order four bits (D_7-D_4) in the accumulator is greater than 9 or if the Carry flag is set, the instruction adds 6 (60) to the high-order four bits.

Example Add decimal 12_{BCD} to the accumulator, which contains 39_{BCD}.

The binary sum is 4BH. The value of the low-order four bits is larger than 9. Add 06 to the low-order four bits.

$$\begin{array}{r}
 4B = 0\ 1\ 0\ 0\ 1\ 0\ 1\ 1 \\
 + 06 = 0\ 0\ 0\ 0\ 0\ 1\ 1\ 0 \\
 \hline
 51 = 0\ 1\ 0\ 1\ 0\ 0\ 0\ 1
 \end{array}$$

Example Add decimal 68_{BCD} to the accumulator, which contains 85_{BCD}.

$$(A) = \begin{array}{r} 85_{BCD} = 1\ 0\ 0\ 0\ 0\ 1\ 0\ 1 \\ + 68_{BCD} = 0\ 1\ 1\ 0\ 1\ 0\ 0\ 0 \\ \hline 153_{BCD} = 1\ 1\ 1\ 0\ 1\ 1\ 0\ 1 \end{array}$$

The binary sum is EDH. The values of both, low-order and high-order, four bits are higher than 9. Add 6 to both.

$$\begin{array}{r}
 = \quad ED = \quad 1\ 1\ 1\ 0 \quad 1\ 1\ 0\ 1 \\
 + 66 = \quad 0\ 1\ 1\ 0 \quad 0\ 1\ 1\ 0 \\
 \hline
 & 1\ 1 & 1 & 1 \\
 \boxed{1} \ 53 = \boxed{1} & 0\ 1\ 0\ 1 & 0\ 0\ 1\ 1 \\
 CY & CY
 \end{array}$$

The accumulator contains 53 and the Carry flag is set to indicate that the sum is larger than eight bits (153). The program should keep track of the Carry; otherwise it may be altered by the subsequent instructions.

DAD: Add Register Pair to H and L Registers

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Codes
DAD	Reg. pair	1	3	10	
					Reg. Pair
					B
					09
					D
					19
					H
					29
					SP
					39

Description The 16-bit contents of the specified register pair are added to the contents of the HL register and the sum is saved in the HL register. The contents of the source register pair are not altered.

Flags If the result is larger than 16 bits the CY flag is set. No other flags are affected.

Example Assume register pair HL contains 0242H. Multiply the contents by 2.

Instruction: DAD H Hex Code: 29

Before instruction	DAD operation	After instruction
H [02 42] L	0242 $\begin{array}{r} +0242 \\ \hline 0484 \end{array}$	H [04 84] L

Example Assume register pair HL is cleared. Transfer the stack pointer (register) that points to memory location 2099H to the HL register pair.

Instruction: DAD SP Hex Code: 39

Before instruction	DAD operation	After instruction
H [00 00] L SP [2099]	0000 $\begin{array}{r} +2099 \\ \hline 2099 \end{array}$	H [20 99] L SP [2099]

Note: After the execution of the instruction, the contents of the stack pointer register are not altered.

DCR: Decrement Source by 1

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Codes
DCR	Reg.	1	1	4	Reg. B 05
	Mem.	1	3	10	C 0D D 15 E 1D H 25 L 2D M 35 A 3D

Description The contents of the designated register/memory is decremented by 1 and the results are stored in the same place. If the operand is a memory location, it is specified by the contents of the HL register pair.

Flags S, Z, P, AC are modified to reflect the result of the operation. CY is not modified.

Example Decrement register B, which is cleared, and specify its contents after the decrement.

Instruction: DCR B Hex Code: 05

Before instruction		Decrement operation	
A	XX	F	(B) = 0 0 0 0 0 0 0 0
B	00	C	-01 = 0 0 0 0 0 0 0 1

Subtraction is performed in 2's complement:

$$\begin{array}{r}
 (B) = 0 0 0 0 0 0 0 0 \\
 + \\
 \text{2's complement of } 1 = \overline{1} 1 1 1 1 1 1 1 \\
 (B) = \overline{1} 1 1 1 1 1 1 1
 \end{array}$$

After the execution of the DCR instruction register B will contain FFH; however, this instruction does not modify the CY flag.

Example Decrement the contents of memory location 2085, which presently holds A0H.

Assume the HL register contains 2085H.

Instruction: DCR M Hex Code: 35

Before instruction			Memory	
H	20	85	L	2084 2085 2086
After instruction			Memory	
H	20	85	L	2084 2085 2086

DCX: Decrement Register Pair by 1

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Codes
DCX	Reg. pair	1	1	6	Reg. Pair
					Hex
					B 0B
					D 1B
					H 2B
					SP 3B

Description The contents of the specified register pair are decremented by 1. This instruction views the contents of the two registers as a 16-bit number.

Flags No flags are affected.

Example Register pair DE contains 2000H. Specify the contents of the entire register if it is decremented by 1.

Instruction: DCX D Hex Code: 1B

After subtracting 1 from the DE register pair the answer is

D 1F FF E

Example Write instructions to set the Zero flag when a register pair (such as BC) is used as a down-counter.

To decrement the register pair, instruction DCX is necessary; instruction DCR is used for one register. However, instruction DCX does not set the Zero flag when the register pair goes to 0 and it continues counting indefinitely. The Zero flag can be set by using the following instructions.

For BC pair:

→ DCX B	;Decrement register pair BC
MOV A,C	;Load accumulator with the contents of register C
ORA B	;Set Zero flag if B and C are both 0
→ JNZ	;If Zero flag is not set, go back and decrement the contents of BC ;pair

DI: Disable Interrupts

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
DI	None	1	1	4	F3

Description The Interrupt Enable flip-flop is reset and all the interrupts except the TRAP (8085) are disabled.

Flags No flags are affected.

Comments: This instruction is commonly used when the execution of a code sequence cannot be interrupted. For example, in critical time delays, this instruction is used at the beginning of the code and the interrupts are enabled at the end of the code. The 8085 TRAP cannot be disabled.

EI: Enable Interrupts

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
EI	None	1	1	4	FB

Description The Interrupt Enable flip-flop is set and all interrupts are enabled.

Flags No flags are affected.

Comments: After a system reset or the acknowledgment of an interrupt, the Interrupt Enable flip-flop is reset, thus disabling the interrupts. This instruction is necessary to reenable the interrupts (except TRAP).

HLT: Halt and Enter Wait State

Opcode	Operand	Bytes	M-Cycle	T-States	Hex Code
HLT	None	1	2 or more	5 or more	76

Description The MPU finishes executing the current instruction and halts any further execution. The MPU enters the Halt Acknowledge machine cycle and Wait states are inserted in every clock period. The address and the data bus are placed in the high imped-

ance state. The contents of the registers are unaffected during the HLT state. An interrupt or reset is necessary to exit from the Halt state.

Flags No flags are affected.

IN: Input Data to Accumulator from a Port with 8-bit Address

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
IN	8-bit port address	2	3	10	DB

Description The contents of the input port designated in the operand are read and loaded into the accumulator.

Flags No flags are affected.

Comments: The operand is an 8-bit address; therefore, port addresses can range from 00H to FFH. While executing the instruction, a port address is duplicated on low-order (A_7-A_0) and high-order ($A_{15}-A_8$) address buses. Any one of the sets of address lines can be decoded to enable the input port.

INR: Increment Contents of Register/Memory by 1

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Codes
INR	Reg.	1	1	4	B 04
	Mem.	1	3	10	C 0C
					D 14
					E 1C
					H 24
					L 2C
					M 34
					A 3C

Description The contents of the designated register/memory are incremented by 1 and the results are stored in the same place. If the operand is a memory location, it is specified by the contents of HL register pair.

Flags S, Z, P, AC are modified to reflect the result of the operation. CY is not modified.

Example Register D contains FF. Specify the contents of the register after the increment.

Instruction: INR D Hex Code: 14

$$\begin{array}{r}
 (D) = \quad 1 \ 1 \ 1 \ 1 \quad 1 \ 1 \ 1 \ 1 \\
 + 1 = \quad 0 \ 0 \ 0 \ 0 \quad 0 \ 0 \ 0 \ 1 \\
 \hline
 \begin{array}{c} 1 \ 1 \ 1 \ 1 \quad 1 \ 1 \ 1 \\ \hline 0 \ 0 \ 0 \ 0 \quad 0 \ 0 \ 0 \ 0 \end{array} \text{ Carry} \\
 \text{CY}
 \end{array}$$

After the execution of the INR instruction, register D will contain 00H; however, no Carry flag is set.

Example Increment the contents of memory location 2075H, which presently holds 7FH. Assume the HL register contains 2075H.

Instruction: INR M Hex Code: 34

Before instruction	Memory
H [20 75] L	2074
	2075 7F
	2076
After instruction	
H [20 75] L	2074
	2075 80
	2076

INX: Increment Register Pair by 1

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Codes										
INX	Reg. pair	1	1	6	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <th>Reg. Pair</th> <th>Hex</th> </tr> <tr> <td>B</td> <td>03</td> </tr> <tr> <td>D</td> <td>13</td> </tr> <tr> <td>H</td> <td>23</td> </tr> <tr> <td>SP</td> <td>33</td> </tr> </table>	Reg. Pair	Hex	B	03	D	13	H	23	SP	33
Reg. Pair	Hex														
B	03														
D	13														
H	23														
SP	33														

Description The contents of the specified register pair are incremented by 1. The instruction views the contents of the two registers as a 16-bit number.

Flags No flags are affected.

Example Register pair HL contains 9FFFH. Specify the contents of the entire register if it is incremented by 1.

Instruction: INX H Hex Code: 23

After adding 1 to the contents of the HL pair the answer is

H	A0	00	L
---	----	----	---

JMP: Jump Unconditionally

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
JMP	16-bit	3	3	10	C3

Description The program sequence is transferred to the memory location specified by the 16-bit address. This is a 3-byte instruction; the second byte specifies the low-order byte and the third byte specifies the high-order byte.

Example Write the instruction at location 2000H to transfer the program sequence to memory location 2050H.

Instruction:

Memory Address	Code	Mnemonics
2000	C3	JMP 2050H
2001	50	
2002	20	

Comments: The 16-bit address of the operand is entered in memory in reverse order, the low-order byte first, followed by the high-order byte.

Jump Conditionally

Operand: 16-bit address

Op Code	Description	Flag Status	Hex Code	M-Cycles/T-States
JC	Jump on Carry	CY = 1	DA	2M/7T (if condition
JNC	Jump on No Carry	CY = 0	D2	is not true)
JP	Jump on positive	S = 0	F2	3M/10T (if condition
JM	Jump on minus	S = 1	FA	is true)
JPE	Jump on Parity Even	P = 1	EA	
JPO	Jump on Parity Odd	P = 0	E2	
JZ	Jump on Zero	Z = 1	CA	
JNZ	Jump on No Zero	Z = 0	C2	

Flags No flags are affected.

Comments: The 8085 requires only seven T-states when condition is not true. For example, instruction JZ 2050H will transfer the program sequence to location 2050H when the Zero flag is set ($Z = 1$) and the execution requires ten T-states. When the Zero flag is reset ($Z = 0$), the execution sequence will not be changed and this requires seven T-states.

LDA: Load Accumulator Direct

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
LDA	16-bit address	3	4	13	3A

Description The contents of a memory location, specified by a 16-bit address in the operand, are copied to the accumulator. The contents of the source are not altered. This is a 3-byte instruction; the second byte specifies the low-order address and the third byte specifies the high-order address.

Flags No flags are affected.

Example Assume memory location 2050H contains byte F8H. Load the accumulator with the contents of location 2050H.

Instruction: LDA 2050H Hex Code: 3A 50 20 (note the reverse order)

A	F8	X	F	2050	[F8]
---	----	---	---	------	------

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
LDAX	B/D reg. pair	1	2	7	Reg. BC DE Hex 0A 1A

LDAX: Load Accumulator Indirect

Description The contents of the designated register pair point to a memory location. This instruction copies the contents of that memory location into the accumulator. The contents of either the register pair or the memory location are not altered.

Flags No flags are affected.

Example Assume the contents of register B = 20H, C = 50H, and memory location 2050H = 9FH. Transfer the contents of the memory location 2050H to the accumulator.

Instruction: LDAX B Hex Code: 0A

Register contents before instruction		Memory contents		Register contents after instruction	
A	XX XX	F		A	9F XX
B	20 50	C	2050 9F	B	20 50

LHLD: Load H and L Registers Direct

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
LHLD	16-bit address	3	5	16	2A

Description The instruction copies the contents of the memory location pointed out by the 16-bit address in register L and copies the contents of the next memory location in register H. The contents of source memory locations are not altered.

Flags No flags are affected.

Example Assume memory location 2050H contains 90H and 2051H contains 01H. Transfer memory contents to registers HL.

Instruction: LHLD 2050H Hex Code: 2A 50 20

Memory contents before instruction		Register contents after instruction	
2050	90	H	01 90 L
2051	01		

LXI: Load Register Pair Immediate

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
LXI	Reg. pair, 16-bit data	3	3	10	Reg. Pair Hex B 01 D 11 H 21 SP 31

Description The instruction loads 16-bit data in the register pair designated in the operand. This is a 3-byte instruction; the second byte specifies the low-order byte and the third byte specifies the high-order byte.

Flags No flags are affected.

Example Load the 16-bit data 2050H in register pair BC.

Instruction: LXI B,2050H Hex Code: 01 50 20

This instruction loads 50H in register C and 20H in register B.

Comments: Note the reverse order in entering the code of 16-bit data. This is the only instruction that can directly load a 16-bit address in the stack pointer register.

MOV: Move—Copy from Source to Destination

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
MOV	Rd,Rs	1	1	4	See table below
MOV	M,Rs		2	7	
MOV	Rd,M				

Description This instruction copies the contents of the source register into the destination register; the contents of the source register are not altered. If one of the operands is a memory location, it is specified by the contents of HL registers.

Flags No flags are affected.

Hex Code

		Source Location								
		B	C	D	E	H	L	M	A	
Destination Location	B	40	41	42	43	44	45	46	47	
	C	48	49	4A	4B	4C	4D	4E	4F	
	D	50	51	52	53	54	55	56	57	
	E	58	59	5A	5B	5C	5D	5E	5F	
	H	60	61	62	63	64	65	66	67	
	L	68	69	6A	6B	6C	6D	6E	6F	
	M	70	71	72	73	74	75		77	
	A	78	79	7A	7B	7C	7D	7E	7F	

Example Assume register B contains 72H and register C contains 9FH. Transfer the contents of register C to register B.

Instruction: MOV B,C Hex Code: 41

Note the first operand B specifies the destination and the second operand C specifies the source.

Register contents before instruction		Register contents after instruction	
B	72	9F	C

Example Assume the contents of registers HL are 20H and 50H, respectively. Memory location 2050H contains 9FH. Transfer the contents of the memory location to register B.

Instruction: MOV B,M Hex Code: 46

Register contents before instruction		Memory contents	Register contents after instruction	
B	XX XX	C	B	9F XX
D	XX XX	E → 2050 [9F]	D	XX XX
H	20 50	L	H	20 50

MVI: Move Immediate 8-Bit

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
MVI	Reg., Data	2	2	7	Reg. Hex
	Mem., Data	2	3	10	B 06
					C 0E
					D 16
					E 1E
					H 26
					L 2E
					M 36
					A 3E

Description The 8-bit data are stored in the destination register or memory. If the operand is a memory location, it is specified by the contents of HL registers.

Flags No flags are affected.

Example Load 92H in register B.

Instruction: MVI B,92H Hex Code: 06 92
This instruction loads 92H in register B.

Example Assume registers H and L contain 20H and 50H, respectively. Load 3AH in memory location 2050H.

Instruction: MVI M,3AH Hex Code: 36

Contents before instruction		Contents after instruction
H H	20 50	L → 2050 [3A] H 20 50 L

NOP: No Operation

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
NOP	None	1	1	4	00

Description No operation is performed. The instruction is fetched and decoded; however, no operation is executed.

Flags No flags are affected.

Comments: The instruction is used to fill in time delays or to delete and insert instructions while troubleshooting.

ORA: Logically OR with Accumulator

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
ORA	Reg.	1	1	4	Reg. Hex
	Mem.	1	2	7	B B0 C B1 D B2 E B3 H B4 L B5 M B6 A B7

Description The contents of the accumulator are logically ORed with the contents of the operand (register or memory), and the results are placed in the accumulator. If the operand is a memory location, its address is specified by the contents of HL registers.

Flags Z, S, P are modified to reflect the results of the operation. AC and CY are reset.

Example Assume the accumulator has data byte 03H and register C holds byte 81H. Combine the bits of register C with the accumulator bits.

Instruction: ORA C Hex Code: B1

Register contents before instruction		Logical OR	Register contents after instruction											
A	F		S	Z	AC	P								
A <table border="1" style="display: inline-table;"><tr><td>03</td><td>XX</td></tr><tr><td>XX</td><td>81</td></tr></table> F	03	XX	XX	81		03H = 0 0 0 0 0 0 1 1 81H = 1 0 0 0 0 0 0 1 83H = 1 0 0 0 0 0 1 1 S = 1, Z = 0, P = 0	A <table border="1" style="display: inline-table;"><tr><td>83</td><td>1,0,0,0,0</td></tr><tr><td>XX</td><td>81</td></tr></table> F	83	1,0,0,0,0	XX	81			
03	XX													
XX	81													
83	1,0,0,0,0													
XX	81													
B	C		B											

Flags: CY = 0, AC = 0

Comments: The instruction is commonly used to

- reset the CY flag by ORing the contents of the accumulator with itself.
 - set the Zero flag when 0 is loaded into the accumulator by ORing the contents of the accumulator with itself.
 - combine bits from different registers.
-

ORI: Logically OR Immediate

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
ORI	8-bit data	2	2	7	F6

Description The contents of the accumulator are logically ORed with the 8-bit data in the operand and the results are placed in the accumulator.

Flags S, Z, P are modified to reflect the results of the operation. CY and AC are reset.

OUT: Output Data from Accumulator to a Port with 8-Bit Address

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
OUT	8-bit port address	2	3	10	D3

Description The contents of the accumulator are copied into the output port specified by the operand.

Flags No flags are affected.

Comments: The operand is an 8-bit address; therefore, port addresses can range from 00H to FFH. While executing the instruction, a port address is placed on the low-order address bus (A_7-A_0) as well as the high-order address bus ($A_{15}-A_8$). Any of the sets of address lines can be decoded to enable the output port.

PCHL: Load Program Counter with HL Contents

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
PCHL	None	1	1	6	E9

Description The contents of registers H and L are copied into the program counter. The contents of H are placed as a high-order byte and of L as a low-order byte.

Flags No flags are affected.

Comments: This instruction is equivalent to a 1-byte unconditional Jump instruction. A program sequence can be changed to any location by simply loading the H and L registers with the appropriate address and by using this instruction.

POP: Pop off Stack to Register Pair

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code	
POP	Reg. pair	1	3	10	Reg.	Hex
					B	C1
					D	D1
					H	E1
					PSW	F1

Description The contents of the memory location pointed out by the stack pointer register are copied to the low-order register (such as C, E, L, and flags) of the operand. The stack pointer is incremented by 1 and the contents of that memory location are copied to the high-order register (B, D, H, A) of the operand. The stack pointer register is again incremented by 1.

Flags No flags are modified.

Example Assume the stack pointer register contains 2090H, data byte F5 is stored in memory location 2090H, data byte 01H is stored in location 2091H. Transfer the contents of the stack to register pair H and L.

Instruction: POP H Hex Code: E1

Register contents before instruction	Stack contents	Register contents after instruction													
H <table border="1"><tr><td>XX</td><td>XX</td></tr><tr><td>SP</td><td>2090</td></tr></table> L	XX	XX	SP	2090	2090 <table border="1"><tr><td>F5</td></tr><tr><td>2091</td><td>01</td></tr><tr><td>2092</td><td></td></tr></table>	F5	2091	01	2092		H <table border="1"><tr><td>01</td><td>F5</td></tr><tr><td>SP</td><td>2092</td></tr></table> L	01	F5	SP	2092
XX	XX														
SP	2090														
F5															
2091	01														
2092															
01	F5														
SP	2092														

Comments: Operand PSW (Program Status Word) represents the contents of the accumulator and the flag register; the accumulator is the high-order register and the flags are the low-order register.

Note that the contents of the source, stack locations, are not altered after the POP instruction.

PUSH: Push Register Pair onto Stack

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code	
PUSH	Reg. pair	1	3	12	Reg.	Hex
					B	C5
					D	D5
					H	E5
					PSW	F5

Description The contents of the register pair designated in the operand are copied into the stack in the following sequence. The stack pointer register is decremented and the contents of the high-order register (B, D, H, A) are copied into that location. The stack pointer register is decremented again and the contents of the low-order register (C, E, L, flags) are copied to that location.

Flags No flags are modified.

Example Assume the stack pointer register contains 2099H, register B contains 32H and register C contains 57H. Save the contents of the BC register pair on the stack.

Instruction: PUSH B Hex Code: C5

Register contents before instruction	Stack contents after instruction	Register contents after instruction											
B <table border="1"><tr><td>32</td><td>57</td></tr><tr><td>C</td><td></td></tr></table>	32	57	C		2097 <table border="1"><tr><td>57</td></tr><tr><td>32</td></tr><tr><td>XX</td></tr></table>	57	32	XX	B <table border="1"><tr><td>32</td><td>57</td></tr><tr><td>C</td><td></td></tr></table>	32	57	C	
32	57												
C													
57													
32													
XX													
32	57												
C													
SP <table border="1"><tr><td>2099</td></tr></table>	2099		SP <table border="1"><tr><td>2097</td></tr></table>	2097									
2099													
2097													

Comments: Operand PSW (Program Status Word) represents the contents of the accumulator and the flag register; the accumulator is the high-order register and the flags are the low-order register.

Note that the contents of the source registers are not altered after the PUSH instruction.

RAL: Rotate Accumulator Left through Carry

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
RAL	None	1	1	4	17

Description Each binary bit of the accumulator is rotated left by one position through the Carry flag. Bit D₇ is placed in the bit in the Carry flag and the Carry flag is placed in the least significant position D₀.

Flags CY is modified according to bit D₇. S, Z, AC, P are not affected.

Example Rotate the contents of the accumulator through Carry, assuming the accumulator has A7H and the Carry flag is reset.

Instruction: RAL Hex Code: 17

		CY	
		[0]	
Accumulator content before instruction	D ₇ D ₆ D ₅ D ₄ D ₃ D ₂ D ₁ D ₀		
	[1] [0] [1] [0] [0] [1] [1] [1]		
		CY	
		[1]	
Accumulator contents after instruction	[0] [1] [0] [0] [1] [1] [1] [0]		

Comment: This instruction effectively provides a 9-bit accumulator. The original contents of the accumulator can be restored by using instruction RAR (Rotate Accumulator Right through Carry). However; the contents will be modified if the instruction RRC (Rotate Accumulator Right) is used to restore the contents.

RAR: Rotate Accumulator Right through Carry

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
RAR	None	1	1	4	1F

Description Each binary bit of the accumulator is rotated right by one position through the Carry flag. Bit D₀ is placed in the Carry flag and the bit in the Carry flag is placed in the most significant position, D₇.

Flags CY is modified according to bit D₀. S, Z, P, AC are not affected.

Example Rotate the contents of the accumulator assuming it contains A7H and the Carry flag is reset to 0.

Instruction: RAR Hex Code: 1F

		CY	
		[0]	
Accumulator contents before instruction	D ₇ D ₆ D ₅ D ₄ D ₃ D ₂ D ₁ D ₀		
	[1] [0] [1] [0] [0] [1] [1] [1]		
		CY	
		[1]	
Accumulator contents after instruction	[0] [1] [0] [1] [0] [0] [1] [1]		

RLC: Rotate Accumulator Left

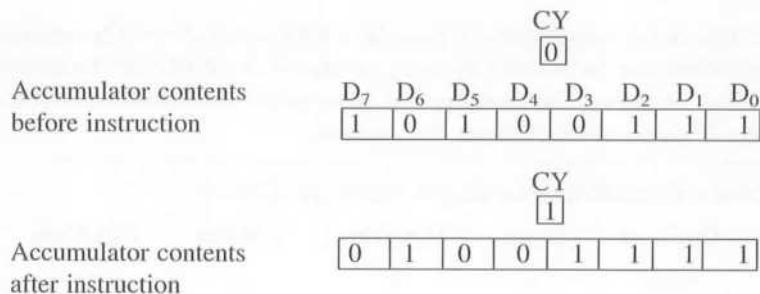
Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
RLC	None	1	1	4	07

Description Each binary bit of the accumulator is rotated left by one position. Bit D₇ is placed in the position of D₀ as well as in the Carry flag.

Flags CY is modified according to bit D₇. S, Z, P, AC are not affected.

Example Rotate the contents of the accumulator left, assuming it contains A7H and the Carry flag is reset to 0.

Instruction: RLC Hex Code: 07



Comments: The contents of bit D₇ are placed in bit D₀, and the Carry flag is modified accordingly. However, the contents of the Carry are not placed in bit D₀ as in instruction RAL.

RRC: Rotate Accumulator Right

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
RRC	None	1	1	4	0F

Description Each binary bit of the accumulator is rotated right by one position. Bit D₀ is placed in the position of D₇ as well as in the Carry flag.

Flags CY is modified according to bit D₀. S, Z, P, AC are not affected.

Example Rotate the contents of the accumulator right, if it contains A7H and the Carry flag is reset to 0.

Instruction: RRC Hex Code: 0F

		CY	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>0</td></tr></table>	0								
0												
Accumulator contents before instruction	D ₇ D ₆ D ₅ D ₄ D ₃ D ₂ D ₁ D ₀	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td></tr></table>	1	0	1	0	0	1	1	1		
1	0	1	0	0	1	1	1					
Accumulator contents after instruction		<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr></table>	1	1	0	1	0	0	1	1	CY <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>1</td></tr></table>	1
1	1	0	1	0	0	1	1					
1												

Comments: The contents of bit D₀ are placed in bit D₇, and the Carry flag is modified accordingly. However, the contents of the Carry are not placed in bit D₇, as in the instruction RAR.

RET: Return from Subroutine Unconditionally

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
RET	None	1	3	10	C9

Description The program sequence is transferred from the subroutine to the calling program. The two bytes from the top of the stack are copied into the program counter and the program execution begins at the new address. The instruction is equivalent to POP Program Counter.

Flags No flags are affected.

Example Assume the stack pointer is pointing to location 2095H. Explain the effect of the RET instruction if the contents of the stack locations are as follows:

2095	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>50</td></tr></table>	50
50		
2096	<table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>20</td></tr></table>	20
20		

After instruction RET, the program execution is transferred to location 2050H and the stack pointer is shifted to location 2097H.

Comments: This instruction is used in conjunction with CALL or conditional call instructions.

Return Conditionally

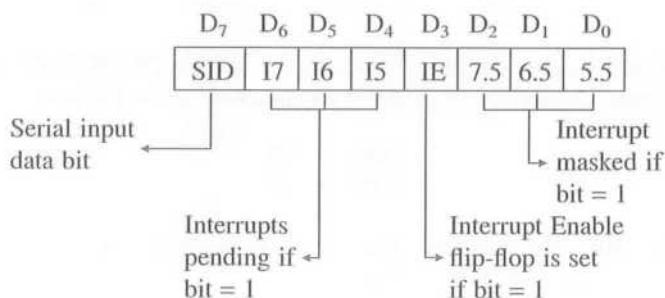
Op Code	Description	Flag Status	Hex Code	M-Cycles/T-States
RC	Return on Carry	CY = 1	D8	
RNC	Return with No Carry	CY = 0	D0	1/6 (if condition is not true)
RP	Return on positive	S = 0	F0	3/12 (if condition is true)
RM	Return on minus	S = 1	F8	<i>Note:</i> If condition is not true, it continues the sequence and thus requires fewer T-states.
RPE	Return on Parity Even	P = 1	E8	
RPO	Return on Parity Odd	P = 0	E0	If condition is true, it returns to the calling program and thus requires more T-states.
RZ	Return on Zero	Z = 1	C8	
RNZ	Return on No Zero	Z = 0	C0	

Flags No flags are affected.

RIM: Read Interrupt Mask

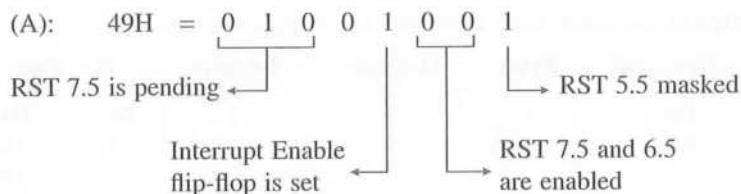
Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
RIM	None	1	1	4	20

Description This is a multipurpose instruction used to read the status of interrupts 7.5, 6.5, 5.5 and to read serial data input bit. The instruction loads eight bits in the accumulator with the following interpretations:



Flags No flags are affected.

Example After the execution of instruction RIM, the accumulator contained 49H. Explain the accumulator contents.



RST: Restart

Bytes	M-Cycles	T-States	Restart	
Opcode/Operand	Binary Code	Hex Code	Address (H)	
RST 0	1 1 0 0 0 1 1 1	C7	0000	
RST 1	1 1 0 0 1 1 1 1	CF	0008	
RST 2	1 1 0 1 0 1 1 1	D7	0010	
RST 3	1 1 0 1 1 1 1 1	DF	0018	
RST 4	1 1 1 0 0 1 1 1	E7	0020	
RST 5	1 1 1 0 1 1 1 1	EF	0028	
RST 6	1 1 1 1 0 1 1 1	F7	0030	
RST 7	1 1 1 1 1 1 1 1	FF	0038	

Description The RST instructions are equivalent to 1-byte call instructions to one of the eight memory locations on page 0. The instructions are generally used in conjunction with interrupts and inserted using external hardware. However, these can be used as software instructions in a program to transfer program execution to one of the eight locations.

Flags No flags are affected.

Additional 8085 Interrupts The 8085 has four additional interrupts and these interrupts generate RST instructions internally and thus do not require any external hardware. These instructions and their Restart addresses are as follows:

Interrupts	Restart Address
TRAP	24H
RST 5.5	2CH
RST 6.5	34H
RST 7.5	3CH

SBB: Subtract Source and Borrow from Accumulator

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
SBB	Reg.	1	1	4	Reg.
	Mem.	1	2	7	
					B 98
					C 99
					D 9A
					E 9B
					H 9C
					L 9D
					M 9E
					A 9F

Description The contents of the operand (register or memory) and the Borrow flag are subtracted from the contents of the accumulator and the results are placed in the accumulator. The contents of the operand are not altered; however, the previous Borrow flag is reset.

Flags All flags are altered to reflect the result of the subtraction.

Example Assume the accumulator contains 37H, register B contains 3FH, and the Borrow flag is already set by the previous operation. Subtract the contents of B with the borrow from the accumulator.

Instruction: SBB B Hex Code: 98

The subtraction is performed in 2's complement; however, the borrow needs to be added first to the subtrahend:

$$\begin{array}{r}
 \text{(B):} \quad \begin{array}{c} 3F \\ + 1 \\ \hline 40H = 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \end{array} \\
 \text{Borrow: } \begin{array}{c} 40H = 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \\ - 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \\ \hline 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 1 \ 1 \end{array} \\
 \text{Subtrahend: } \begin{array}{c} 40H = 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \\ - 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \\ \hline 0 \ 1 \ 1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 1 \end{array} \\
 \text{2's complement of } \begin{array}{c} 40H = 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \\ - 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \\ \hline 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 1 \ 1 \end{array} \\
 \text{(A)} \quad \begin{array}{c} 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 1 \ 1 \\ - 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \\ \hline 1 \ 1 \ 1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 1 \end{array} \\
 \text{Complement Carry: } \begin{array}{c} 1 \ 1 \ 1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 1 \end{array} = F7H
 \end{array}$$

The Borrow flag is set to indicate the result is in 2's complement. The previous Borrow flag is reset during the subtraction.

SBI: Subtract Immediate with Borrow

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
SBI	8-bit data	2	2	7	DE

Description The 8-bit data (operand) and the borrow are subtracted from the contents of the accumulator, and the results are placed in the accumulator.

Flags All flags are altered to reflect the result of the operation.

Example Assume the accumulator contains 37H and the Borrow flag is set. Subtract 25H with borrow from the accumulator.

Instruction: SBI 25H Hex Code: DE 25

$$\begin{array}{r}
 \text{(Data): } 25\text{H} \\
 + \text{(Borrow): } 1\text{H} \\
 \hline
 \text{Subtrahend: } 26\text{H} = 0\ 0\ 1\ 0\ 0\ 1\ 1\ 0 \\
 \text{2's complement of } 26\text{H} = 1\ 1\ 0\ 1\ 1\ 0\ 1\ 0 \\
 \text{(A) } 37\text{H} = 0\ 0\ 1\ 1\ 0\ 1\ 1\ 1 \\
 \hline
 1/0\ 0\ 0\ 1\ 0\ 0\ 0\ 1 = 11\text{H} \\
 \text{Complement Carry: } 0/0\ 0\ 0\ 1\ 0\ 0\ 0\ 1 = 11\text{H} \\
 \text{Flags: S = 0, Z = 0, AC = 1} \\
 \text{P = 1, CY = 0}
 \end{array}$$

SHLD: Store H and L Registers Direct

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
SHLD	16-bit address	3	5	16	22

Description The contents of register L are stored in the memory location specified by the 16-bit address in the operand, and the contents of H register are stored in the next memory location by incrementing the operand. The contents of registers HL are not altered. This is a 3-byte instruction; the second byte specifies the low-order address and the third byte specifies the high-order address.

Flags No flags are affected.

Example Assume the H and L registers contain 01H and FFH, respectively. Store the contents at memory locations 2050H and 2051H.

Instruction: SHLD 2050H Hex Code: 22 50 20

Register contents
before instruction

H	01	FF	L
---	----	----	---

Memory and register contents
after instruction

H	01	FF	L
---	----	----	---

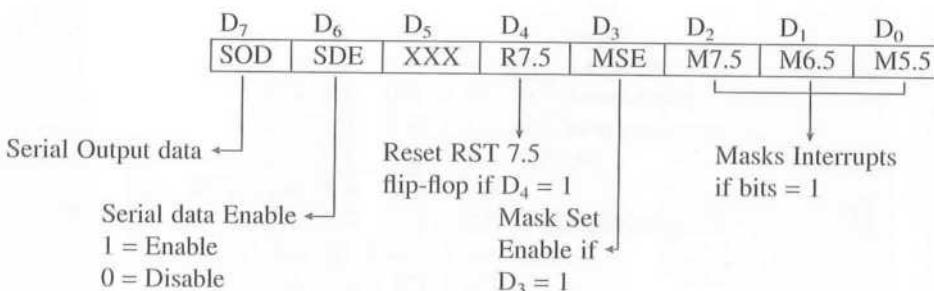
2050	FF
2051	01

SIM: Set Interrupt Mask

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
SIM	None	1	1	4	30

Description This is a multipurpose instruction and used to implement the 8085 interrupts (RST 7.5, 6.5, and 5.5) and serial data output.

The instruction interprets the accumulator contents as follows:



- **SOD**—Serial Output Data: Bit D₇ of the accumulator is latched into the SOD output line and made available to a serial peripheral if bit D₆ = 1.
- **SDE**—Serial Data Enable: If this bit = 1, it enables the serial output. To implement serial output, this bit needs to be enabled.
- **XXX**—Don't Care
- **R7.5**—Reset RST 7.5: If this bit = 1, RST 7.5 flip-flop is reset. This is an additional control to reset RST 7.5.
- **MSE**—Mask Set Enable: If this bit is high, it enables the functions of bits D₂, D₁, D₀. This is a master control over all the interrupt masking bits. If this bit is low, bits D₂, D₁, and D₀ do not have any effect on the masks.
- **M7.5**—D₂ = 0, RST 7.5 is enabled
 - = 1, RST 7.5 is masked or disabled
- **M6.5**—D₁ = 0, RST 6.5 is enabled
 - = 1, RST 6.5 is masked or disabled
- **M5.5**—D₀ = 0, RST 5.5 is enabled
 - = 1, RST 5.5 is masked or disabled

Example Write instructions to enable interrupt RST 5.5 and mask other interrupts.

Instruction: MVI A, 0EH ;Bits D₃ = 1 and D₀ = 0
SIM ;Enable RST 5.5

Example A TTY receiver line is connected to the SOD pin of the 8085. Disable all interrupts and send START bit (logic 0) to TTY without affecting interrupt masks.

Instructions: MVI A,40H ;D₇ = 0, START bit at logic 0
;D₆ = 1, Enables serial output bit D₇
;D₃ = 0, Does not affect masks
SIM ;Send START bit

Comments: This instruction does not affect TRAP interrupt.

SPHL: Copy H and L Registers to the Stack Pointer

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
SPHL	None	1	1	6 (8085) 5 (8080)	F9

Description The instruction loads the contents of the H and L registers into the stack pointer register; the contents of the H register provide the high-order address, and the contents of the L register provide the low-order address. The contents of the H and L registers are not altered.

Flags No flags are affected.

STA: Store Accumulator Direct

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
STA	16-bit	3	4	13	32

Description The contents of the accumulator are copied to a memory location specified by the operand. This is a 3-byte instruction; the second byte specifies the low-order address and the third byte specifies the high-order address.

Flags No flags are affected.

Example Assume the accumulator contains 9FH. Load the accumulator contents into memory location 2050H.

Instruction: STA 2050H Hex Code: 32 50 20

Register contents
before instruction

A [9F] [XX] F

Memory contents
after instruction

2050 [9F]

STAX: Store Accumulator Indirect

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code	
STAX	B/D reg. pair	1	2	7	Reg.	Hex

B	02
D	12

Description The contents of the accumulator are copied into the memory location specified by the contents of the operand (register pair). The contents of the accumulator are not altered.

Flags No flags are affected.

Example Assume the contents of the accumulator are F9H and the contents of registers B and C are 20H and 50H, respectively. Store the accumulator contents in memory location 2050H.

Instruction: STAX B Hex Code: 02

Register contents before instruction			Register and memory contents after instruction		
A	F9	XX	F	2050	F9
B	20	50	C		A F9 XX F B 20 50 C

Comments: This instruction performs the same function as MOV A,M except this instruction uses the contents of BC or DE as memory pointers.

STC: Set Carry

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
STC	None	1	1	4	37

Description The Carry flag is set to 1.

Flags No other flags are affected.

SUB: Subtract Register or Memory from Accumulator

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
SUB	Reg.	1	1	4	Reg.
	Mem.	1	2	7	Hex
					B 90
					C 91
					D 92
					E 93
					H 94
					L 95
					M 96
					A 97

Description The contents of the register or the memory location specified by the operand are subtracted from the contents of the accumulator, and the results are placed in the accumulator. The contents of the source are not altered.

Flags All flags are affected to reflect the result of the subtraction.

Example Assume the contents of the accumulator are 37H and the contents of register C are 40H. Subtract the contents of register C from the accumulator.

Instruction: SUB C Hex Code: 91

$$\begin{array}{l}
 (\text{C}): 40\text{H} = 0100\ 0000 \\
 \text{2's complement (C)}: \quad = 1100\ 0000 \\
 (\text{A}): 37\text{H} = \underline{0011\ 0111} \\
 \qquad \qquad \qquad 0/1111\ 0111 = F7\text{H} \\
 \text{Complement Carry:} \quad 1/1111\ 0111 \\
 \text{Flags: S = 1, Z = 0, AC = 0} \\
 \qquad \qquad \qquad P = 0, CY = 1
 \end{array}$$

The result, as a negative number, will be in 2's complement and thus the Carry (Borrow) flag is set.

SUI: Subtract Immediate from Accumulator

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
SUI	8-bit data	2	2	7	D6

Description The 8-bit data (the operand) are subtracted from the contents of the accumulator, and the results are placed in the accumulator.

Flags All flags are modified to reflect the results of the subtraction.

Example Assume the accumulator contains 40H. Subtract 37H from the accumulator.

Instruction: SUI 37H Hex Code: D6 37

$$\begin{array}{r}
 \text{Subtrahend: } 37H = \begin{array}{rr} 0 & 0 \\ 1 & 1 \end{array} \quad \begin{array}{rr} 0 & 1 \\ 1 & 1 \end{array} \\
 \text{2's complement of } 37H = \begin{array}{rr} 1 & 1 \\ 0 & 0 \end{array} \quad \begin{array}{rr} 1 & 0 \\ 0 & 1 \end{array} \\
 + \\
 (\text{A}): 40H = \begin{array}{rr} 0 & 1 \\ 1 & 0 \end{array} \quad \begin{array}{rr} 0 & 0 \\ 0 & 0 \end{array} \\
 \hline
 \begin{array}{rr} 1 & 0 \\ 0 & 0 \end{array} \quad \begin{array}{rr} 1 & 0 \\ 0 & 1 \end{array}
 \end{array}$$

Complement Carry: 0/ $\begin{array}{rr} 0 & 0 \\ 0 & 0 \end{array}$ $\begin{array}{rr} 1 & 0 \\ 0 & 1 \end{array} = 09H$

Flags: S = 0, Z = 0, AC = 0
P = 1, CY = 0

XCHG: Exchange H and L with D and E

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
XCHG	None	1	1	4	EB

Description The contents of register H are exchanged with the contents of register D, and the contents of register L are exchanged with the contents of register E.

Flags No flags are affected.

XRA: Exclusive OR with Accumulator

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
XRA	Reg.	1	1	4	Reg. Hex
	Mem.	1	2	7	B A8 C A9 D AA E AB H AC L AD M AE A AF

Description The contents of the operand (register or memory) are Exclusive ORed with the contents of the accumulator, and the results are placed in the accumulator. The contents of the operand are not altered.

Flags Z, S, P are altered to reflect the results of the operation. CY and AC are reset.

Example Assume the contents of the accumulator are 77H and of register D are 56H. Exclusive OR the contents of the register D with the accumulator.

Instruction: XRA D Hex Code: AA

$$\begin{array}{l}
 \text{(A): } 77H = 0\ 1\ 1\ 1 \quad 0\ 1\ 1\ 1 \\
 \text{(D): } 56H = 0\ 1\ 0\ 1 \quad 0\ 1\ 1\ 0 \\
 \text{Exclusive OR:} \qquad \qquad \qquad \underline{0\ 0\ 1\ 0 \quad 0\ 0\ 0\ 1} \\
 \text{Flags: } S = 0, Z = 0, P = 1, \\
 \text{CY} = 0, AC = 0
 \end{array}$$

XRI: Exclusive OR Immediate with Accumulator

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
XRI	8-bit data	2	2	7	EE

Description The 8-bit data (operand) are Exclusive ORed with the contents of the accumulator, and the results are placed in the accumulator.

Flags Z, S, P are altered to reflect the results of the operation. CY and AC are reset.

Example Assume the contents of the accumulator are 8FH. Exclusive OR the contents of the accumulator with A2H.

Instruction: XRI A2H Hex Code: EE A2

$$\begin{array}{l}
 \text{(A): } 8FH = 1\ 0\ 0\ 0 \quad 1\ 1\ 1\ 1 \\
 \text{(Data): } A2H = 1\ 0\ 1\ 0 \quad 0\ 0\ 1\ 0 \\
 \text{Exclusive OR:} \qquad \qquad \qquad \underline{0\ 0\ 1\ 0 \quad 1\ 1\ 0\ 1} \\
 \text{Flags: } S = 0, Z = 0, P = 1 \\
 \text{CY} = 0, AC = 0
 \end{array}$$

XTHL: Exchange H and L with Top of Stack

Opcode	Operand	Bytes	M-Cycles	T-States	Hex Code
XTHL	None	1	5	16	E3

Description The contents of the L register are exchanged with the stack location pointed out by the contents of the stack pointer register. The contents of the H register are exchanged with the next stack location ($SP + 1$); however, the contents of the stack pointer register are not altered.

Flags No flags are affected.

Example The contents of various registers and stack locations are as shown:

Registers			Stacks	
H	A2	57	L	2095 38
SP	2095			2096 67

Illustrate the contents of these registers after instruction XTHL.

Register contents

after XTHL

Registers			Stacks	
H	67	38	L	2095 57
SP	2095			2096 A2

8085

Instruction Summary: Hexadecimal Order

Hex	Mnemonic	Hex	Mnemonic	Hex	Mnemonic	Hex	Mnemonic
00	NOP	11	LXI D	21	LXI H	31	LXI SP
01	LXI B	12	STAX D	22	SHLD	32	STA
02	STAX B	13	INX D	23	INX H	33	INX SP
03	INX B	14	INR D	24	INR H	34	INR M
04	INR B	15	DCR D	25	DCR H	35	DCR M
05	DCR B	16	MVI D	26	MVI H	36	MVI M
06	MVI B	17	RAL	27	DAA	37	STC
07	RLC	19	DAD D	29	DAD H	39	DAD SP
09	DAD B	1A	LDAX D	2A	LHLD	3A	LDA
0A	LDAX B	1B	DCX D	2B	DCX H	3B	DCX SP
0B	DCX B	1C	INR E	2C	INR L	3C	INR A
0C	INR C	1D	DCR E	2D	DCR L	3D	DCR A
0D	DCR C	1E	MVI E	2E	MVI L	3E	MVI A
0E	MVI C	1F	RAR	2F	CMA	3F	CMC
0F	RRC	20	RIM	30	SIM	40	MOV B,B

Hex	Mnemonic	Hex	Mnemonic	Hex	Mnemonic	Hex
Mnemonic						
41	MOV B,C	70	MOV M,B	9F	SBB A	CF
42	MOV B,D	71	MOV M,C	A0	ANA B	D0
43	MOV B,E	72	MOV M,D	A1	ANA C	D1
44	MOV B,H	73	MOV M,E	A2	ANA D	D2
45	MOV B,L	74	MOV M,H	A3	ANA E	D3
46	MOV B,M	75	MOV M,L	A4	ANA H	D4
47	MOV B,A	76	HLT	A5	ANA L	D5
48	MOV C,B	77	MOV M,A	A6	ANA M	D6
49	MOV C,C	78	MOV A,B	A7	ANA A	D7
4A	MOV C,D	79	MOV A,C	A8	XRA B	D8
4B	MOV C,E	7A	MOV A,D	A9	XRA C	DA
4C	MOV C,H	7B	MOV A,E	AA	XRA D	DB
4D	MOV C,L	7C	MOV A,H	AB	XRA E	DC
4E	MOV C,M	7D	MOV A,L	AC	XRA H	DE
4F	MOV C,A	7E	MOV A,M	AD	XRA L	DF
50	MOV D,B	7F	MOV A,A	AE	XRA M	E0
51	MOV D,C	80	ADD B	AF	XRA A	E1
52	MOV D,D	81	ADD C	B0	ORA B	E2
53	MOV D,E	82	ADD D	B1	ORA C	E3
54	MOV D,H	83	ADD E	B2	ORA D	E4
55	MOV D,L	84	ADD H	B3	ORA E	E5
56	MOV D,M	85	ADD L	B4	ORA H	E6
57	MOV D,A	86	ADD M	B5	ORA L	E7
58	MOV E,B	87	ADD A	B6	ORA M	E8
59	MOV E,C	88	ADC B	B7	ORA A	E9
5A	MOV E,D	89	ADC C	B8	CMP B	EA
5B	MOV E,E	8A	ADC D	B9	CMP C	EB
5C	MOV E,H	8B	ADC E	BA	CMP D	EC
5D	MOV E,L	8C	ADC H	BB	CMP E	EE
5E	MOV EM	8D	ADC L	BC	CMP H	EF
5F	MOV EA	8E	ADC M	BD	CMP L	F0
60	MOV H,B	8F	ADC A	BE	CMP M	F1
61	MOV H,C	90	SUB B	BF	CMP A	F2
62	MOV H,D	91	SUB C	C0	RNZ	F3
63	MOV H,E	92	SUB D	C1	POP B	F4
64	MOV H,H	93	SUB E	C2	JNZ	F5
65	MOV H,L	94	SUB H	C3	JMP	F6
66	MOV H,M	95	SUB L	C4	CNZ	F7
67	MOV H,A	96	SUB M	C5	PUSH B	F8
68	MOV L,B	97	SUB A	C6	ADI	F9
69	MOV L,C	98	SBB B	C7	RST 0	FA
6A	MOV L,D	99	SBB C	C8	RZ	FB
6B	MOV L,E	9A	SBB D	C9	RET	FC
6C	MOV L,H	9B	SBB E	CA	JZ	FE
6D	MOV L,L	9C	SBB H	CC	CZ	FF
6E	MOV L,M	9D	SBB L	CD	CALL	RST
6F	MOV L,A	9E	SBB M	CE	ACI	7

8085 Instruction Summary by Functional Groups

DATA TRANSFER GROUP

	Move	Move (cont)	Move Immediate	Add*	Increment**	Logical*
MOV	A,A 7F	[E,A 5F	A, byte 3E	A 87	A 3C	A A7
	A,B 78	E,B 58	B, byte 06	B 80	B 04	B A0
	A,C 79	E,C 59	C, byte 0E	C 81	C 0C	C A1
	A,D 7A	MVI-	D, byte 16	D 82	D 14	D A2
	A,E 7B	E,D 5A	E, byte 1E	E 83	E 1C	E A3
	A,H 7C	E,E 5B	H, byte 26	H 84	H 24	H A4
	A,L 7D	E,H 5C	L, byte 2E	L 85	L 2C	L A5
	A,M 7E	E,L 5D	M, byte 36	M 86	M 34	M A6
	B,A 47	H,A 67		A 8F	B 03	A AF
	B,B 40	H,B 60		B 88	D 13	B A8
MOV	B,C 41	H,C 61	Load Immediate	C 89	H 23	C A9
	B,D 42	H,D 62		D 8A	SP 33	D AA
	B,E 43	H,E 63		E 8B		E AB
	B,H 44	H,H 64		H 8C		H AC
	B,L 45	H,L 65		L 8D		L AD
	B,M 46	H,M 66		M 8E		M AE
	C,A 4F	L,A 6F		A 3D		A B7
	C,B 48	L,B 68	Load/Store	B 05		B B0
	C,C 49	L,C 69		C 0D		C B1
	C,D 4A	MVI-	LDAX B 0A	D 15		D B2
MOV	C,E 4B	L,D 6A	LDAX D 1A	E 1D		E B3
	C,H 4C	L,E 6B	LHLD adr 2A	H 25		H B4
	C,L 4D	L,H 6C	LDA adr 3A	L 2D		L B5
	C,M 4E	L,L 6D	STAX B 02	M 35		M B6
	D,A 57	L,M 6E	STAX D 12	H 94	B 0B	A BF
	D,B 50	M,A 77	SHLD adr 22	L 95	D 1B	B B8
	D,C 51	M,B 70	STA adr 32	M 96	H 2B	C B9
	D,D 52	M,C 71		A 9F	SP 3B	D BA
	D,E 53	M,D 72		B 98		E BB
	D,H 54	M,E 73		C 99		H BC
MOV	D,L 55	M,H 74		D 9A		L BD
	D,M 56	M,L 75		E 9B	DAA* 27	M BE
	XCHG	EB		H 9C	CMA 2F	Arith & Logical
				L 9D	STC† 37	Immediate
				M 9E	CMC† 3F	
						ADI byte C6
						ACI byte CE
						SUI byte D6
						SBI byte DE
						ANI byte E6

byte = constant, or logical/arithmetic expression that evaluates to an 8-bit data quantity. (Second byte of 2-byte instructions).

dble = constant, or logical/arithmetic expression that evaluates to a 16-bit data quantity. (Second and Third bytes of 3-byte instructions).

adr = 16-bit address (Second and Third bytes of 3-byte instructions).

* = all flags (C, Z, S, P, AC) affected.

** = all flags except CARRY affected; (exception: INX and DCX affect no flags).

† = only CARRY affected.

All mnemonics copyright ©Intel Corporation 1976.

ARITHMETIC AND LOGICAL GROUP

	Double Add †	Rotate †	
DAD	B 09	RLC 07	ADI byte C6
	D 19	RRC 0F	ACI byte CE
	H 29	RAL 17	SUI byte D6
	SP 39	RAR 1F	SBI byte DE
			ANI byte E6
			XRI byte EE
			ORI byte F6
			CPI byte FE

BRANCH CONTROL GROUP		I/O AND MACHINE CONTROL		ASSEMBLER REFERENCE (Cont.)
Jump		Stack Ops		Pseudo Instruction
JMP adr	C3	PUSH	B C5 D D5 H E5 PSW F5	General: ORG END EQU
JNZ adr	C2			
JZ adr	CA			
JNC adr	D2			
JC adr	DA	POP	B C1 D D1 H E1 PSW F1	SET DS DB DW
JPO adr	E2			
JPE adr	EA			
JP adr	F2			
JM adr	FA			
PCHL	E9	XTHL SPHL	E3 F9	Macros: MACRO ENDM LOCAL REPT
Call				
CALL adr	CD			IRP
CNZ adr	C4			IRPC
CZ adr	CC			EXITM
CNC adr	D4	OUT byte	D3	
CC adr	DC	IN byte	DB	
CPO adr	E4			
CPE adr	EC			
CP adr	F4			
CM adr	FC	Control		Relocation:
		DI EI	F3 FB	ASEG NAME DSEG STKLN CSEG STACK
Return		NOP	00	PUBLIC MEMORY
RET	C9	HLT	76	EXTRN
RNZ	C0			
RZ	C8			
RNC	D0	New Instructions (8085 Only)		Conditional Assembly:
RC	D8			IF
RPO	E0	RIM	20	ELSE
RPE	E8	SIM	30	ENDIF
RP	F0			
RM	F8			

Restart

RESTART TABLE

		Name	Code	Restart Address
RST	0 C7	RST 0	C7	000016
	1 CF	RST 1	CF	000816
	2 D7	RST 2	D7	001016
	3 DF	RST 3	DF	001816
	4 E7	RST 4	E7	002016
	5 EF	TRAP	Hardware*	002416
	6 F7	RST 5	Function	
	7 FF	RST 5 5	EF	002816
			Hardware*	002C16
		RST 6	Function	
		RST 6 5	F7	003016
		RST 7	Hardware*	003416
		RST 7 5	Function	
			FF	003816
			Hardware*	003C16
			Function	

*NOTE: The hardware functions refer to the on-chip interrupt feature of the 8085 only.

Instruction Set with Machine Cycles and Flag Status (see notes at end of table)

Instruction	Code ¹	B/M/T ²	Machine ³ Cycles	S D ₇	Z D ₆	AC D ₄	P D ₂	CY D ₀	Flags ⁴
AACI DATA	CE data	2/2/7	F R	✓	✓	✓	✓	✓	✓
ADC REG	1000 1SSS	1/1/4	F	✓	✓	✓	✓	✓	✓
ADC M	8E	1/2/7	F R	✓	✓	✓	✓	✓	✓
ADD REG	1000 OSSS	1/1/4	F	✓	✓	✓	✓	✓	✓
ADD M	86	1/2/7	F R	✓	✓	✓	✓	✓	✓
ADI DATA	C6 DATA	2/2/7	F R	✓	✓	✓	✓	✓	✓
ANI REG	1010 OSSS	1/1/4	F	✓	✓	✓	✓	✓	✓
ANA M	A6	1/2/7	F R	✓	✓	✓	✓	✓	✓
ANI DATA	E6 data	2/2/7	F R	✓	✓	✓	✓	✓	✓
CALL ADDR	CD addr	3/5/18	S R R W W	✓	✓	✓	✓	✓	✓
ADDR	DC addr	3/5/18	S R R W W	✓	✓	✓	✓	✓	✓
CDM ADDR	FC addr	3/5/18	S R R W W	✓	✓	✓	✓	✓	✓
CMC REG	2F	3/5/18	S R R W W	✓	✓	✓	✓	✓	✓
CMC M	3F	1/1/4	F	✓	✓	✓	✓	✓	✓
CMC REG	1011 1SSS	1/1/4	F	✓	✓	✓	✓	✓	✓
CMC M	BE	1/2/7	F R	✓	✓	✓	✓	✓	✓
CNC ADDR	D4 addr	3/5/18	S R R W W	✓	✓	✓	✓	✓	✓
CNZ ADDR	C4 addr	3/5/18	S R R W W	✓	✓	✓	✓	✓	✓
CP ADDR	F4 addr	3/5/18	S R R W W	✓	✓	✓	✓	✓	✓
CPE ADDR	EC addr	3/5/18	S R R W W	✓	✓	✓	✓	✓	✓
CPI DATA	FE data	2/2/7	F R	✓	✓	✓	✓	✓	✓
CPO ADDR	E4 addr	3/5/18	S R R W W	✓	✓	✓	✓	✓	✓
CZ ADDR	CC addr	3/5/18	S R R W W	✓	✓	✓	✓	✓	✓

DAA		: Decimal-Adjust A																		
DAD	Rp	: Add Reg. Pair to HL	27	00Rp 1001	1/1/4	F														
DCR	REG	: Decrement Reg.		00SS S101	1/3/10	F B B														
DCR	M	: Decrement Mem. Contents	35		1/1/4	F														
DCR	Rp	: Decrement Reg. Pair		00Rp 1011	1/3/10	F R W														
DCX	Rp	: Disable Interrupt		F3	1/1/6	S														
DI		: Enable Interrupt			1/1/4	F														
EI		: Halt		FB	1/1/4	F														
HLT		: Input from 8-bit Port	76		1/2/5	F B														
IN	PORT	: Increment Reg.		DB data	2/3/10	F R I														
INR	REG	: Increment Mem. Contents	34	00SS S100	1/1/4	F														
INR	M	: Increment Reg. Pair		00Rp 0011	1/3/10	F R W														
INX	Rp	: Increment Reg. Pair		DA addr	1/1/6	S														
JC	ADDR	: Jump On Carry		FA addr	3/3/7-10	F R R														
JM	ADDR	: Jump On Minus		C3 addr	3/3/7-10	F R R														
JMP	ADDR	: Unconditional Jump		D2 addr	3/3/7-10	F R R														
JNC	ADDR	: Jump On No Carry		C2 addr	3/3/7-10	F R R														
JNZ	ADDR	: Jump On No Zero		F2 addr	3/3/7-10	F R R														
JP	ADDR	: Jump On Positive		EA addr	3/3/7-10	F R R														
JPE	ADDR	: Jump On Parity Even		E2 addr	3/3/7-10	F R R														
JPO	ADDR	: Jump On Parity Odd		CA addr	3/3/7-10	F R R														
JZ	ADDR	: Jump On Zero		3A addr	3/4/13	F R R														
LDA	ADDR	: Load A Direct		000X 1010	1/2/7	F R														
LDAX	Rp	: Load A from M; memory address is in BC/DE																		
Codes ¹																				
Machine Cycles ³																				
B/M/T ²																				
B = Bytes																				
M = Machine cycles																				
T = T-states																				
S = Sign																				
F = Fetch with 4 T-states																				
S = Fetch with 6 T-states																				
R = Memory Read																				
I = I/O Read																				
E = I/O Write																				
W = Memory Write																				
O = I/O Write																				
B = Bus Idle																				
Flags ⁴																				
P = Flag is modified																				
according to result																				
0 = Flag is cleared																				
1 = Flag is set																				
Blank = No change in flag.																				
B = Bus Idle																				

DDD = Binary digits identifying a destination register
 SSS = Binary digits identifying a source register
 B = 000, C = 001, D = 010, Memory = 110
 E = 001, H = 100, L = 101, A = 111
 J = 100, BC = 00, HL = 10
 JP = Register Pair DE = 01, SP = 11

AC = Auxiliary Carry
 P = Parity
 CY = Carry
 S = Sign
 Z = Zero
 R = Memory Read
 I = I/O Read
 W = Memory Write
 O = I/O Write

Flags⁴

P = Flag is modified according to result
 0 = Flag is cleared
 1 = Flag is set
 Blank = No change in flag.
 remains in previous state

Instruction Set with Machine Cycles and Flag Status (see notes at end of table)

		Instruction	Code ¹	B/M/T ²	Machine ³ Cycles	S D ₇	Z D ₆	Flags ⁴ AC D ₄	P D ₂	CY D ₀
LHLD	ADDR	: Load HL Direct	2A addr	3/5/16	F R R R					
LXI	Rp, 16-bit	: Load 16-bit in Reg. Pair	00Rp 0001 16-bit	3/3/10	F R R					
MOV	Rd,Rs	: Move from Reg. R _s to Reg. R _d	01DD DSSS	1/1/4	F					
MOV	M,R	: Move from Reg. to Mem.	0111 OSSS	1/2/7	F W					
MOV	R,M	: Move from Mem. to Reg.	01DD D110	1/2/7	F R					
MVI	R,DATA	: Load 8-bit in Reg.	00DD D110 data	2/2/7	F R					
MVI	M,DATA	: Load 8-bit in Mem.	36 data	2/3/10	F R W					
NOP		: No Operation	00	1/1/4	F					
ORA	R	: OR Reg. with A	1011 OSSS	1/1/4	F					0
ORA	M	: OR Mem. Contents with A	B6	1/2/7	F R					0
ORI	DATA	: OR 8-bit with A	F6 data	2/2/7	F R					0
OUT	PORT	: Output to 8-bit Port	D3 data	2/3/10	F R O					
PCHL		: Move HL to Program Counter	E9	1/1/6	S					
POP	Rp	: Pop Reg. Pair	11Rp 0001	1/3/10	F R R					
PUSH	Rp	: Push Reg. Pair	11Rp 0101	1/3/12	S W W					
RAL		: Rotate A Left through CY	17	1/1/4	F					
RAR		: Rotate A Right through CY	IF	1/1/4	F					
RC		: Return On Carry	D8	1/3/6-12	S R R					
RET		: Return	C9	1/3/10	F R R					
RIM		: Read Interrupt Mask	20	1/1/4	F					
RLC		: Rotate A Left	07	1/1/4	F					
RM		: Return On Minus	F8	1/3/6-12	S R R					
RNC		: Return On No Carry	D0	1/3/6-12	S R R					
RNZ		: Return On No Zero	C0	1/3/6-12	S R R					
RP		: Return On Positive	F0	1/3/6-12	S R R					

RPE		: Return On Parity Even	E8	1/3/6-12	S R R			
RPO		: Return On Parity Odd	E0	1/3/6-12	S R R			
RRC		: Rotate A to Right	0F	1/1/4	F			
RST	N	: Restart	11XX X111	1/3/12	S W W			
RZ		: Return On Zero	C8	1/3/6-12	S R R			
SBB	R	: Subtract Reg. from A with Borrow	1001 1SSS	1/1/4	F			
SBB	M	: Subtract Mem. Contents from A with Borrow	9E	1/2/7	F R			
SBI	DATA	: Subtract 8-bit from A	DE data	2/2/7	F R			
SHLD	ADDR	: Store HL Direct	22 addr	3/5/16	F R R W W			
SIM		: Set Interrupt Mask	30	1/1/4	F			
SPHL		: Move HL to Stack Pointer	F9	1/1/6	S			
STA	ADDR	: Store A Direct	32 addr	3/4/13	F R R W			
STAX	Rp	: Store A in M, memory address is in BC/DE	000X 0010	1/2/7	F W			
STC		: Set Carry	37	1/1/4	F			1
SUB	R	: Subtract Reg. from A	1001 0SSS	1/1/4	F			
SUB	M	: Subtract Mem. from A	96	1/2/7	F R			
SUI	DATA	: Subtract 8-bit from A	D6 data	2/2/7	F R			
XCHG		: Exchange DE with HL	EB	1/1/4	F			
XRA	R	: Exclusive OR Reg. with A	1010 1SSS	1/1/4	F		0	
XRA	M	: Exclusive OR Mem. with A	AE	1/2/7	F R		0	
XRI	DATA	: Exclusive OR 8-bit with A	EE data	2/2/7	F R		0	
XTHL		: Exchange Stack with HL	E3	1/4/16	F R R W W		0	
Codes ¹								
B/M/T ²								
B = Bytes M = Machine cycles T = T-states								
S = Sign Z = Zero AC = Auxiliary Carry P = Parity CY = Carry								
F = Fetch with 4 T-states S = Fetch with 6 T-states R = Memory Read I = I/O Read W = Memory Write O = I/O Write B = Bus Idle								
Machine Cycles ³								
DDD = Binary digits identifying a destination register SSS = Binary digits identifying a source register B = 000, C = 001, D = 010, Memory = 110 E = 001, H = 100, L = 101, A = 111 Rp = Register Pair, BC = 00, HL = 10 DE = 01, SP = 11								
Flags ⁴								
↖ = Flag is modified according to result								
0 = Flag is cleared								
1 = Flag is set								
Blank = No change in flag.								
Remains in previous state								

G

Solutions to Selected Questions, Problems, and Programming Assignments

CHAPTER 1

1. Components of a computer: ALU and Control Unit (CPU), Memory, Input and Output.
3. A microprocessor functions as the CPU of a microcomputer, and includes the ALU, register arrays, and the control unit on one chip; it is manufactured using the LSI technology. On the other hand, the CPU is designed with various discrete boards. Functionally, both are similar; however, technology and processes used for designing are different.
4. A microprocessor is one component of a microcomputer, and the microcomputer is a complete computer consisting of a microprocessor, memory, input and output.
7. Four bytes.
8. The machine language of the 8085 is the commands to the microprocessor given in binary. These are the binary instructions the processor can understand and execute. The assembly language is comprised of mnemonics (group of letters to represent commands) assigned by the manufacturer for the convenience of the users.
12. The assembly language mnemonics represent instructions to the microprocessor; therefore, when they are translated into machine language, there is one-to-one correspondence between the mnemonics and the machine code. The assembly language programs are compact, require less memory space, and are efficient. The high-level languages are written in English-like statements, and when these statements are translated in machine language, the object code tends to be large, and requires large memory. The execution of the programs written in high-level languages is less efficient than that of assembly language programs.

CHAPTER 2

1. Memory Read, Memory Write, I/O Read, and I/O Write.
4. A microprocessor with 14 address lines is capable of addressing 16K (2^{14}) memory locations.
5. 21 address lines.
7. IOR (I/O Read), IOW (I/O Write), MEMR (Memory Read), and MEMW (Memory Write).
8. In memory write operation, the control signal required is MEMW, and the direction of the data flow is from the MPU to memory.
10. A flag is the output of a given flip-flop to indicate certain data conditions.
11. The program counter and the stack pointer are used as memory pointers. In this microprocessor the size of the memory address is 16 bits; thus these registers are required to store 16-bit addresses.
12. The program counter always points to the next memory location; therefore, the content of the program counter will be 2058H.
13. 128 registers and $128 \times 4 = 512$ memory cells.
15. 8-bit word size.
18. 32 chips.
20. 11 address lines.
23. The starting address is: E000H.
26. The memory map ranges from 2000H to 23FFH.
28. 8 address lines are required for a peripheral I/O port, and 16 address lines are required for a memory-mapped I/O port.
31. From B to A.
32. None. The decoder is not enabled; all output lines will be high.
33. The line 6 (O_6).
35. A transparent latch is a flip-flop; its output changes according to input when the clock signal is high, and it latches the input on the trailing edge of the clock (high to low). The latch is necessary for output devices to retain the result; otherwise, the result will disappear.
38. The memory occupies the memory space from F000H to FFFFH. The don't care line A_{11} generates additional address range from F800 to FFFFH when it is assumed to be at logic 1. This is a 2K memory chip that occupies 4K of memory space in the map, thus wasting 2K of memory space.

CHAPTER 3

1. The ALE signal goes high at the beginning of each machine cycle indicating the availability of an address on the address bus, and the signal is used to latch the low-order address bus. The IO/M signal is a status signal indicating whether the machine cycle is I/O or memory operation. The IO/M signal is combined with the RD and WR control signals to generate IOR, IOW, MEMR, and MEMW control signals.
3. In Fig. 3.22, the input signals RD and WR cannot be low at the same time. Therefore, the valid combinations of the input signals are:

<u>IO/M</u>	<u>RD</u>	<u>WR</u>	Decoder Output
1	0	1	<u>O₅ – IOR</u>
1	1	0	<u>O₆ – IOW</u>
0	0	1	<u>O₁ – MEMR</u>
0	1	0	<u>O₂ – MEMW</u>
0	0	0	– Invalid
1	0	0	– Invalid

The remaining two output signals O_3 and O_7 do not represent any operation.

5. In Fig. 3.23, the 74LS139 is enabled when IO/M is low. Therefore, the following memory control signals can be generated.

<u>RD</u>	<u>WR</u>	Decoder Output
0	0	<u>O₀ – Invalid</u>
0	1	<u>O₁ – MEMR</u>
1	0	<u>O₂ – MEMW</u>
1	1	<u>O₃ – No operation</u>

6. The output of the latch will be 05H; however, it will not be latched until the ALE goes low.
10. The sum of $87H + 79H = 100H$. Therefore, the accumulator will have 00H, and the flags will be S = 0, CY = 1, Z = 1.
12. $18T \times .2 \text{ micro-sec} = 3.6 \text{ micro-sec}$.
13. $(A_{15}-A_8) = 20H$, $(AD_7-AD_0) = 47H$.
15. The second machine cycle is Memory Read; the processor reads the contents of memory in register B, and the control signal is RD.
17. $(A_{15}-A_0) = 2050H$.
 (AD_7-AD_0) as data bus = Contents of location 2050H.
19. Memory map: 6000H to 6FFFH.
23. Memory map: 28000H to 2FFFFH.
24. Total range = 16K. Map = 8000H to BFFFFH.
26. Memory map: 0800H–08FFH, and the foldback memory ranges from 0900 to OFFFH.
29. ROM1: 0000H – 1FFFH, ROM2: E000H – FFFFH, R/WM1: 8000H – 83FFH.
32. Address range: 4000H to 7FFFH.
36. When $A_{14} = 1$, Y_1 is active. Address range: 4000H to 7FFFH. When $A_{14} = 0$, Y_2 is active. Address range: 8000H to BFFFH.
38. The last MEMR is the third byte of the STA Instruction. It reads FFH.

CHAPTER 4

1. The number of output ports in the peripheral I/O is restricted to 256 ports because the operand of the OUT instruction is 8 bits; it can have only 256 combinations.
3. The 8085 differentiates between the input and the output ports of the same address by the control signal. The input port requires the RD and the output port requires the WR control signals.

6. Trailing edge.
9. A latch is necessary to hold the output data for display; however, the input data byte is obtained by enabling a tri-state buffer and placed in the accumulator.
10. RD, WR, and IO/M (low).
12. 78H.
15. 8000H.
17. If $A_7 = 0$, port address = 75H, and if $A_7 = 1$, address = F5H.
18. If IO/M is connected to E1 (active low), it will be a memory-mapped I/O. The port address = 00F5H.
Replace OUT F5H by STA 00F5H.
19. Output code either 40H or C0H.
21. If $A_7 = 0$, the addresses are: 04H, 0CH, 14H, and 1CH.
If $A_7 = 1$, the addresses are: 84H, 8CH, 94H, and 9CH (as shown in Section 4.3.4).
23. Port A = Memory-mapped Output Port (0085H).
Port B = Memory-mapped Input Port (0085H).
25. In Figure 4.10, the output O_5 is enabled by the address, which is active for three T-states. On the other hand, the IOW signal requires WR signal, which is active for approximately one and one-half T-states.
27. a. LDA FFF9H → OF, MR, MR, and MR
STA FFF8H → OF, MR, MR, and MW
MOV B,A → OF
JMP START → OF, MR, and MR
b. FFF9H.
c. RD = 11 times and WR = 1 time.
d. $40T \times 0.5 \text{ micro-sec} = 20 \text{ micro-sec}$.
28. Assuming $A_4 = 0$: Input Port = 2FH and Output Port = 8FH.
Assuming $A_4 = 1$: Input Port = 3FH and Output Port = 9FH.
29. START: IN 2FH ;Read input port
ANI 00000011B ;Mask all bits except D1 and D0
JNZ START ;If a switch is open, read again
MVI A, 00 ;Unnecessary instruction, used for clarity
OUT 8FH ;Turn on all LEDs
HLT
32. MVI A, 98H ;Code for '9' to upper LED
OUT F5H
MVI A, F8H ;Code for '7' to lower LED
OUT F4H
HLT

CHAPTER 5

2. Opcode: MOV and Operand: H, L.
5. (a) Hex Code = 32 50 20, Opcode = STA Operands = 2050H.
(b) Hex Code = C2 70 20, Opcode = JNZ Operands = 2070H.

6. Mnemonics Bytes Hex Code

MVI	B,4FH	(2)	064F
MVI	C,78H	(2)	0E78
MOV	A,C	(1)	79
ADD	B	(1)	80
OUT	07H	(2)	D307
HLT		(1)	76

11. MVI B, A2H ;Load bytes

MVI C, 18H
 MOV A,B
 ADD C ;The result is in A
 HLT

13. The processor assumes the code of the next instruction (HLT:76H) as the port address, outputs the contents of A to the address 76H, and continues the execution; the result is unpredictable.

CHAPTER 6

3. MVI C,65H
 MVI A,92H
 OUT PORT1 ; DISPLAY 92H
 MOV A,C ; Copy C into A for display
 OUT PORT0 ; Display 65H
 HLT

5. 82H.

7. Both will be 80H.

9. The instruction ADD A will add the content of the accumulator to itself; this is equivalent to multiplying by 2.

10. The instruction SUB A will clear the accumulator. The flag status will be:

CY = 0, Z = 1.

14. MVI A, 00H (A = 0 0 0 0 0 0 0
 DCR A — 0 0 0 0 0 0 1
 OUT PORT#

 HLT 1 1 1 1 1 1 1 = FFH

The instruction DCR does not set the CY flag.

15. A = 8FH, S = 1 and CY = 0.

The S flag has no meaning when subtracting unsigned numbers.

18. SUB A ; Clear accumulator
 ADI 47H
 SUI 92H ; A = B5H, CY = 1 (Borrow Flag)
 OUT PORT0 ; Display B5H
 ADI 64H ; A = 19H
 OUT PORT1
 HLT

20. The instruction XRA A will clear the accumulator, and the flag status will be: CY = 0, Z = 1.
23. The instruction ORA A will set the flag without affecting the content of the accumulator.
26. MVI C, A8H
 MOV A,C
 ANI 0FH ; Masking byte to mask D₇-D₄
 OUT PORT0
 HLT
28. MVI B, 91H
 MVI C, 87H
 MOV A,B
 ANI 01H ; Mask all bits of 91H except D₀
 MOV B,A ; Save D₀ from first byte
 MOV A,C
 ANI 01H ; Mask all bits of 87H except D₀
 ANA B ; AND bits D₀ of 91H and 87H
 OUT PORT1 ; Turn on/off light connected to D₀
 HLT
29. IN 07H
 CMA ; Complement data from port 07H
 ORA A ; Set Z flag if all switches are open
 | ; Continue
32. In this problem, the range of bytes that will be displayed at PORT2 is 50H to 7FH.
34. 00.
35. This routine displays the absolute value (magnitude) of BYTE1.
38. XRA A ; Clear CY
 MVI B, FFH
 INR B
 MOV A, B
 JNC DISPLAY
 MVI A, 01H
 DISPLAY: OUT PORT# ; Output = 00H because INR does not
 HLT ; set CY flag.

To clear the CY flag, the instructions such as ANA A, SUB A, ORA A can be used instead of the instruction XRA A.

40. MVI B, BYTE1
 MVI C, BYTE2
 MOV A, B
 SUB C
 JNC DISPLAY ; Jump if result is positive
 CMA ; Take one's complement
 ADI 01H ; Find two's complement
- DISPLAY: OUT PORT1
 HLT

CHAPTER 7

The following programs assume the system's R/W memory begins at location 2000H.

The symbols XX in the assignments are assumed as memory page 20H.

5. Location 2075H will contain F7H.
7. A = FFH and (2070H) = FFH.
9. A = 00H, D = 00H, HL = 209FH.

```
11.      LXI    B,2090H
          SUB    A
          MVI    D, 0FH
LOOP:   STAX   B
          INX    B
          DCR    D
          JNZ    LOOP
          HLT
```

13. 7 times.

```
15. START: LXI    H, 2055H ; Index for data source
          LXI    D, 2085H ; Index for data destination, starting
                           ; at last location
          MVI    B, 06H ; Byte counter
NEXT:   MOV    A, M ; Get data byte
          STAX   D ; Store data byte
          INX    H ; Next location
          DCX    D
          DCR    B ; Next count
          JNZ    NEXT ; If counter is not 0, go back
                           ; to transfer next byte
          HLT
```

```
18. START: MVI    B, 6 ; Byte count
          LXI    H, 2050H ; Source
          LXI    D, 2050H ; Destination
LOOP:   MOV    A, M ; Get byte
          ORA    A ; Set flag in zero
          JNZ    SKIP
          STAX   D ; Not zero, so store it
          INX    D
SKIP:   INX    H ; Go on to next
          DCR    B
          JNZ    LOOP
          HLT
```

22. Locations 2070H to 2074H contain 01, 02, 03, 04, and 05 respectively.

25. S = 0, Z = 1, CY is unchanged.

34. (a) (b)

A	CY
MVI A, C5H	C5 NA
ORA A	C5 0

A	CY
MVI A, A7H	A7 NA
ORA A	A7 0

RAL	8A	I	RAR	4E	I
RRC	45	0	RAL	A7	0

35. These instructions will move the MSD of a BCD number (7 in this case) to the unit's position. A = 07H.

37. Multiply by 10.

41.

	A	S	Z	CY
MVI A, 7FH	7F	NA	NA	NA
ORA A	7F	0	0	0
CPI A2H	7F	1	0	1

43. 00, 00, 7A, 87, 00, 00.

49. START: LXI H, 2050H ; Set index to point to data location
 MVI C, 08H ; Set up counter
 MVI B, 00H ; Clear (B) to save the highest reading
 NXTBYTE: MOV A, M ; Get data byte
 CMP B ; Is (B) > (A)?
 JNC NEXT ; If yes, replace (B) with (A)
 MOV B, A ; Save the larger number
 NEXT: INX H ; Point to next data byte
 DCR C ; Next count
 JNZ NXTBYTE ; Jump to get next byte
 MOV A, B ; Load the largest byte
 OUT PORT1 ; Display the largest byte in the string
 HLT

53. START: LXI H, 2070H ; Source pointer
 LXI D, 2090H ; Save pointer
 LOOP: MOV A, M
 CPI 0DH ; Check for end of string
 JZ ENDS ; If = 0DH, then end of string
 CPI 30H
 JC REJECT ; Reject if < 30H
 CPI 3AH ; Note: if subtract 39H, would reject 39H
 JNC REJECT ; Reject if < 39H
 STAX D ; OK, so save it
 INX D

REJECT: INX H ; Loop for next
 JMP LOOP

ENDS: HLT

55. START: LXI H,XXXX ; Set up HL as a pointer to the data set
 SUB A ; Clear A

 MOV C, A ; Clear C to set up as a counter

AGAIN: CMP M ; Is this end of data (00)?

 JZ DSPLAY ; If yes, this is end of counting

 MVI A, A3H ; Load television code

 CMP M ; Is this television code?

```

        JNZ NEXT    ; If not, go to the next data byte
        INR C      ; Update the count
NEXT:   INX H      ; Next data
        SUB A      ; Clear A
        JMP AGAIN  ; Go back to check next byte
DISPLAY: MOV A, C ; Get count
        OUT PORT1
        HLT

```

57. (a) 256 bytes (b) 2090H–2091H (c) A must be cleared before addition and INX H is missing in the loop.

CHAPTER 8

2. 255.59 mSec.
3. 511.58 mSec.
5. 234.129 mSec.
7. (a) infinite (b) infinite (c) 1
9. If the system frequency is 3.072 MHz, the clock period will be 325 ns. This will reduce the delay to .325 s.
11. BC = 34965₁₀. Insignificant difference when the delay is calculated with JNZ = 7 T-states in the last iteration.

16.	START: MVI D, 00H	;	Load bit pattern		
ROTATE:	MOV A, D			4	
	CMA		;	Complement bit pattern	4
	MOV D, A			4	
	ANI 01H		;	Mask D ₇ –D ₁	7
	OUT PORT1			10	
	MVI B, COUNT			7	
LOOP:	DCR B			4	
	JNZ LOOP			10/7	
	JMP ROTATE			10	

Total Delay TD = To + TL (System Frequency = 3.072 MHz)

200s = 46 × 325.5 ns + 4 × 325.5 ns × COUNT

COUNT = 42.5 ≈ 42

18.	START: MVI L, 10101010B	;	Alternating light pattern	
LIGHTS:	MOV A, L			
	RRC			
	OUT PORT			
	MOV L, A			
	MVI B, 50		;	20 × 50 mSec DELAY = 1Sec
OUTER:	LXI D, 2559		;	20 mSec DELAY
INNER:	DCX D			
	MOV A, D			
	ORA E			
	JNZ INNER			

```

DCR    B
JNZ    OUTER
JMP    LIGHTS

```

CHAPTER 9

1. (a) 3 (b) 1 (c) 4 (d) 3 (e) 2 (f) 2
3. (a) The address in the program counter will be stored on the stack: (20CCH) = 20H, (20CBH) = 0BH.
The program counter will have the same address as the stack.
The contents of the stack pointer will be incremented to 20CBH.
(b) (20CAH) = 00H (H), (20C9H) = 08H (L)
(20C8H) = 0FH (B), (20C7H) = XX (C)
(c) (stack pointer) = 20C7H.
(d) (stack pointer) = 20CDH.
5. (a) All flags are cleared, and A = 00H.
(b) 20FEH = 0EH, 20FFH = 20H.
(c) SP = 20FEH, PC = 2064H.
(d) 200BH.
(e) Endless loop.

10. This is a 20 ms delay subroutine. It clears Z flag before returning to the calling program. However, the contents of the accumulator are destroyed.

```

DELAY: PUSH  B           ; Save register contents from
       PUSH  H           ; main or calling program
       LXI   B, COUNT    ; Load delay count for 20 ms
LOOP:  DCX   B
       MOV   A, C
       ORA   B           ; Set Z flag if (B) and (C) = 0
       JNZ   LOOP
       PUSH  PSW          ; Save flag status on stack
       POP   H           ; Copy flag status in register L
       MOV   A, L
       ANI   BFH          ; Set bit D6 (Z flag) = 0
       MOV   L, A
       PUSH  H           ; Save flag status with Z flag reset
       POP   PSW          ; Clear Z flag
       POP   H
       POP   B
       RET               ; Return to calling program

```

CHAPTER 10

2. This program converts a set of BCD numbers into binary equivalents and stores them in the Output Buffer.

```
; Main program—Uses subroutine BCDBIN (9-1)
;
START: LXI SP, STACK    ; Initialize stack pointer
        LXI H, INBUF    ; HL pair points to input buffer
        LXI B, OUTBUF   ; BC pair points to output buffer
        MVI D, COUNT    ; How many numbers
LOOP:  CALL BCDBIN
        INX H           ; Bump pointers
        INX B
        DCR D           ; Decrement count
        JNZ LOOP         ; Loop until done
        HLT
```

4. BCD to binary conversion.

```
; Input: 2 Digit BCD in accumulator
; Output: Binary in accumulator
; No registers are changed
;
```

BCDBIN:

```
PUSH D      ; Save contents of D & E registers
MOV E, A    ; Save accumulator
ANI 0F0H    ; Mask MSD
RRC
MOV D, A    ; D = 8 * MSD
RRC
RRC
ADD D
MOV D, A    ; D = 10 * MSD
MOV A, E    ; Restore accumulator
ANI 0FH     ; Mask LSD
ADD D       ; Add in 10 * MSD
POP D       ; Restore DE
RET
```

6. This program supplies the powers of ten in registers B and C.

```
STACK EQU 0FFFFH
BINBYT EQU 1000H
OUTBUF EQU 2000H
PWRTEN EQU 640AH
ORG 100H
START: LXI SP, STACK
        LXI H, BINBYT
        MOV A, M
        LXI H, OUTBUF
        LXI B, PWRTEN
        CALL BINBCD
```

```

MOV  B,C
CALL BINBCD
MOV  M, A
HLT

```

BINBCD: No changes in this routine.

12. This is a modified version of the LEDCOD subroutine (Section 10.3.1) that can be used irrespective of code locations.

LEDCOD:

```

PUSH H
LXI H, CODE
ADD L
JNC NOCY
INR H
NOCY: MOV A, L
      MOV A, M
      STAX B
      POP H
      RET

```

16. This program converts the unpacked BCD digits (from the Illustrative Program—Section 10.5.1) into ASCII characters.

Main Program: Same as in Section 10.5.1

Subroutine BCDADD: Same as in Section 10.5.1

```

UNPAK:  MOV D, A
        CALL UNPAKI
        MOV A, D
        RRC
        RRC
        RRC
        RRC
UNPAK1: ANI 0FH
        ORI '0'      ;This is ASCII 0 = 30H
        MOV M, A
        DCX H
        RET

```

18. This program subtracts two-digit BCD numbers stored in two consecutive memory locations. It uses a programming trick to perform these BCD subtractions rather than following the hints given in the assignment.

```

START: LXI H, NUMBERS
       MOV A, M
       INX H
       SUB M
       MOV B, A
       MVI A, 0    ; Do not change flags of subtraction
       DAA          ; If CY was = 1, A = 60 and AC = 1, A = 06
       MOV C, A    ; C = 06 or 60

```

```

MOV A, B
SUB C
OUT PORT
HLT

```

19. This program adds 16-bit numbers; each number is stored in two consecutive memory locations. The total numbers to be added are specified by the contents of register B, and the maximum sum can be 24 bits.

```

START: LXI SP, 2050H
        MVI B, NUMCNT
        XRA A
        LXI H, 0          ; Clear HL
LOOP:  XCHG           ; Save (HL) in DE
        POP H
        DAD D
        ACI 0            ; if there is a 17th bit, and 1 to A.
        DCR B
        JNZ LOOP
        SHLD OUTBUF
        STA OUTBUF + 2   ; Store the MSB
        HLT
END

```

CHAPTER 12

2. (a) RST 6 (b) 0030H (c) JMP 2075H
3. (a) 03FF = 01 Memory address of the instruction.
03FE = 23 following the CALL instruction.
- (b) 03FD = (B) Contents of registers B and C.
03FC = (C).
- (c) 03FB = 01 Memory address of the instruction.
03FC = 54 following the LXI B instruction.
5. (a) 11 110 111 = F7H (RST 6).
(b) The priority encoder accepts the request from I₅ and rejects all other low priority requests. The instruction that will be placed on the data bus is 11 101 111 = EFH (RST 5).

CHAPTER 13

1. $I_O = 2 \text{ mA} (1/2 + 1/128) = 1.0156 \text{ mA}$.
3. $V_O = (2 \text{ mA} (1/4 + 1/64 + 1/256) - 1 \text{ mA}) \times 5\text{k} = -2.305 \text{ V}$.
5. LXI SP, STACK


```

START: MVI A, 00H
        OUT FFH      ; Output 0 V
        CALL DELAY   ; Wait 0.5 ms
        MVI A, 80H   ; Load A for 5 V output
        OUT FFH      ; Output 5 V
      
```

```

        CALL  DELAY ; Wait 0.5 ms
        JMP   START

7. If input = 01H, Output = 10 V/4096 = 2.44 mV
   If input = 82H, Output = 10/(1/32 + 1/2048) = 317.38 mV

9. START: OUT 80H
TEST:   IN 80H    ; Read Data Ready status
RAL     ; Rotate D7 into CY
JC TEST ; If D0 = 1, wait
IN 81H
RET

```

CHAPTER 14

3. The handshake signals are used to verify the readiness (status) of peripherals.
5. See Section 14.2.5.
8. Control/Status Port = 38H, Port A = 39H, Port B = 3AH
Port C = 3BH, Timer (LSB) = 3CH, Timer (MSB) = 3DH
11. The control word = FFF8H and the instructions are as follows:


```

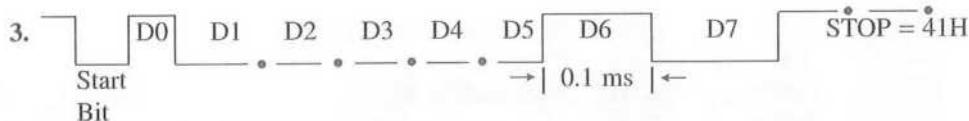
MVI  A, F8H
OUT LO      ; Load the LSB timer register
MVI  A, FFH
OUT HI      ; Load the MSB timer in mode 3
      
```
13. To obtain the square wave of 500 μ s with 3.072 MHz clock frequency, the count should be $1536 = 0600H$, and the timer should be set up in Mode 1. The control word = 4600H.

CHAPTER 15

3. BSR mode D7 = 0, and I/O mode D7 = 1.
5. Yes. If the port A is set up as input port and INTR (bit D3) of the status word is high, IBF signal (bit D5) should be 1.
If port A is set up as output port and INTR (bit D3) of the status word is high, INTE (bit D6) should be set to 1.
9. To start the timer, appropriate control word and count should be loaded, and gate should be high.
11. The most significant two bits (D_{15} and D_{14}) of the count register (Fig. 15.34) are used to specify DMA function and the remaining fourteen bits are used to specify the number of bytes to be transferred. The maximum number of bytes that can be specified with a 14-bit count is 16K.
13. Assuming the don't care lines are at logic 0, the port addresses are as follows:
Port A = 8000H, Port B = 8100H, Port C = 8200H, and Control Register = 8300H

CHAPTER 16

1. (a) F (b) F (c) F (d) T (e) T (f) F (g) T
(h) F (i) T (j) F (k) F (l) T (m) F
2. BITTIME delay = $1/1200 = .833$ mS and Half BITTIME = $.416$ mS.



8. Control Word = 7FH, Command Word = 15H or 05H.

9. ASCBIN:

```

SUI    30H      ; ASCII Zero
JM     ERROR    ; Characters NUL to \ are errors
CP     10        ; If 0 to 9, OK, Done conversion
JP     ASCBN1
SUI    7
JM     ERROR    ; Characters up to & are errors
CP     16
JP     ERROR    ; Characters G to DEL are errors

```

ASCBNI:

```

PUSH   PSW
LXI    H, CKSUM ; Correct ASCII Hex,
              ; so update checksum
ADD    M         ; Checksum is stored in
              ; memory CKSUM
MOV    M, A
POP    PSW
RET

```

CHAPTER 17

3. SHIFT: PUSH PSW ; Save new key info
 MOV A, E ; Get old data
 ADD A ; Shift old key out
 ADD A
 ADD A
 ADD A
 MOV E, A ; Save in E
 POP PSW ; Retrieve new key
 ANI 0FH ; Get rid of garbage
 OR E ; Combine old with new
 MOV E, A ; Store result in E
 RET

5. INSERT: PUSH H ; Save HL
 MOV H, B ; Move BC to HL
 MOV L, C
 DAD H ; Shift left 4 bits
 DAD H
 DAD H
 DAD H

	ANI	0FH	; Remove high order garbage
	OR	L	; Combine
	MOV	B, H	; Store back in BC
	MOV	C, A	
	POP	H	; Restore HL
	RET		
7. Change	ANI	00001111B	to ANI 00011111B (3 places)
	CPI	OFH	to CPI 1FH (2 places)
	MVI	B, 05H	to MVI B, 06H
	MVI	C, 04H	to MVI C, 05H
9. EXEC:	PUSH D		
	RET		

H

Introduction to 8085 Assemblers and Simulators*

This textbook includes a CD with two programs: an 8085 simulator program designed, developed, and written by Abhijit Bhattacharjee and the ENVI85 program by Professors Stefan Fedyschyn and Edwin Kay. Both programs are bundled in a package, and both will be available for the user once the user completes the installation procedure.

The 8085 simulator includes a simulated keypad that enables the user to write mnemonics without errors. The ENVI85 is an integrated program that provides an editor, an assembler, and a simulator. The simulator in ENVI85 is not as extensive as the 8085 simulator. An editor enables the user to write mnemonics (text), an assembler looks up the codes of the mnemonics written by the editor and assigns memory addresses, and a simulator enables the user to observe the internal operations of the 8085 microprocessor at the instruction level on a

PC. We will describe the use of the 8085 simulator first in Section H.1, followed by the ENVI85 in Section H.2.

The 8085 Simulator is a software program that can be executed on IBM-compatible PCs under the Windows operating system. The execution of an 8085 program using the Simulator is an alternative to the execution of a program on a trainer such as SDK-85 or EMAC Primer. This Simulator can execute a program stored in an appropriate format (discussed later) or provide a simulated keypad to write and execute a program. The programs can be executed in two modes: Step and Run. In the Step mode (also known as Single Step), one instruction is executed at a time and changes in registers, flags, and output ports can be observed. In the Run mode, the entire program is executed, and the final results can be observed.

*Questions related to the simulator, write to: gaonkarr@sunyocc.edu or penram@bom3.vsnl.net.in or visit the website: <http://www.insoluz.com>

H.1

HOW TO USE THE 8085 SIMULATOR

The CD includes ten files, many of them titled Setup, which are necessary to install the 8085 Simulator. The Simulator and its support files use more than 1M of disk space leaving around 300K bytes for the user programs. Therefore, the Simulator programs must be installed on your hard disk as explained in the next section.

H.1.1 How to Install the 8085 Simulator

Turn on your computer and wait until the operating system shows the icons on your screen. Insert the CD that accompanied this textbook in the CD drive and access the drive. You can access the CD drive two ways: using the My Computer icon on your screen or using the program Windows Explorer.

Accessing the CD Drive Using the My Computer Icon Double-click on the icon and the screen opens up showing icons for various drives and programs such as Control Panel. Double-click on CD icon and the computer will access the CD drive and show all the files necessary to install the 8085 Simulator.

Accessing the CD Drive Using Windows Explorer After you turn on your computer, click on → Start button → Programs → Windows Explorer. You will see → Desk Top, My Computer, 3 ½ Floppy [A:], [C:], and various folders on C:drive. You may have to scroll down to see the CD icon. Click on → CD icon and you will see all the files necessary to install the 8085 Simulator.

Installing the 8085 Simulator Double-click on the Setup.exe file (it can be identified by a PC monitor icon or .exe extension). The Setup.exe program will guide you through the entire installation process. It will show you various informational screens about the simulator program and a shareware license agreement. Click on the necessary buttons. One of the screens will show you the Destination Directory that will identify the complete path where Setup.exe will install the simulator program.

C:\Program Files\Infotech Solutions\Microprocessor Simulator 8085

The last step in the installation process is to click on Finish. As part of the installation process, Setup.exe will also create an icon and list the path that can be accessed through the Start button as follows:

Click on Start → Program → Infosolutions → Microprocessor Simulator 8085

How to Save the Simulator Under a Different Folder If you want to store the program under a different folder and also list it in the Start menu, use the Browse button when the screen shows Destination Directory. Let us say you want to save the program under the title 8085.

Click on → Browse →
Under Path delete what is in the space

Type C:\8085 in that space and click → OK → Under Program Folders → type 8085 → click OK

Setup.exe will install the 8085 Simulator program and other support files on the C: drive under the folder 8085. It will also create an icon and list the program under the Start menu.

Simulator Screen When you double-click on the Simulator icon, the following display appears on the screen: a page of a **List File** (Figure H.1 or as in Section 11.4.3) showing, on the left, columns for memory address, mnemonics, Hex (or machine) code, and comments; **Code Key Pad** on the right, and title and menu bars on the top.

The menu bar is similar to that of any other application program. The menu bar displays five tasks: File, Edit, View, Options, and Help. If you click on any of these words, you will see various functions you can perform:

File	Enables you to open an existing file and save a file.
Edit	This function is somewhat different from the Edit function in a word processor. It enables you to add and delete a row, and change an active row.
View	Used primarily to display three types of screen: List File memory page, Code Key Pad, and Simulator.
Options	Provides six different options: coding, execution, prompt, file, I/O port, and simulation.
Help	This is a help file accessed through a Web browser.

H.1.2 How to Execute an Existing Program Stored on a Disk in Code or Hex Format

Step 1: Open the 8085 Simulator program either by using the icon or accessing the Micro3 program by one of the following paths:

1. Click → Start → Programs → Infotech Solutions → Microprocessors Simulator 8085
2. Click → Start → Programs → Select Windows Explorer, and click to Select → Program Files → Infotech Solutions → Microprocessor Simulator 8085 → Micro3 and double-click. This path is necessary if you want to use the assembler included in ENVI85 to assemble a program (see Section H.2.2).

The program opens up with the split screen displaying a blank page of a List File on the left and Code Key Pad on the right. The blank page of the List File includes Code Grid and Data Grid. The Code Grid has four columns: memory address (Addr), instructions (mnemonic), Hex code (MC), and comments. The Data Grid also shows two columns: Addr and MC (Figure H.1).

Step 2: Here we set up options. In most situations, the default values are appropriate. However, if the OUT instruction is used in a program, the output port must be set up in the Option category.

Click on Options →. Go to Port Options and enter the 8-bit address 01 and click → OK. Now the simulator is set up to load a program that uses an output port with the address 01.

Step 3: Click → File → Open Code File.

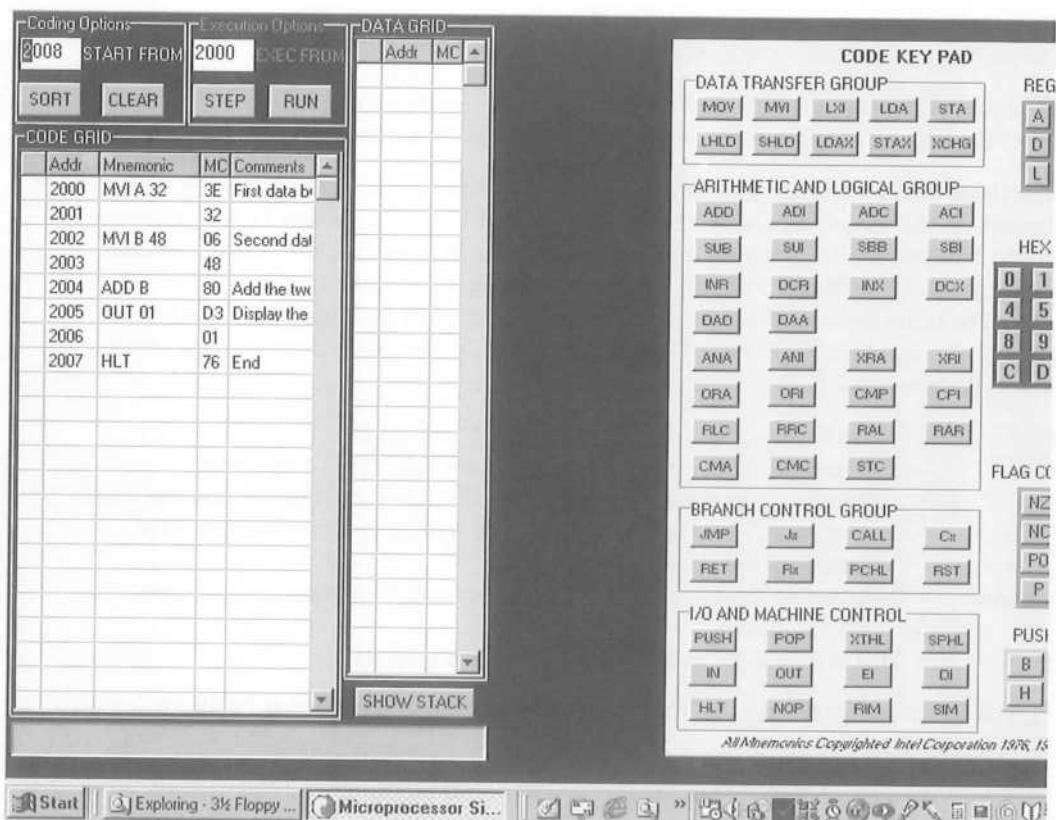


Figure H.1 Display of a List file and Code Key Pad

You get a list of Code Files. Select Add2 and click on Open. The code file is entered in the print file page with mnemonics on the screen (Figure H.1).

Step 4: Click → Step.

The Simulator screen appears as shown in Figure H.2. The Simulator places 32H in A after the execution of the first instruction, MVI A, 32H. The change in the contents of A is also indicated by the violet color; the other registers and flags remain white. The color white indicates that there is no change; the flags are not affected. The Program Counter (PC) is incremented to 2002H, pointing to the next instruction to be executed.

Continue to Click Step. The next instruction places 48H in B with no change in the flags. When the Simulator executes the ADD instruction, it places 7A (Hex) in A. The result 7A in Hex resets all the flags indicated by the flag register in violet. Even if the flags remain zero, they are reset, which is different from no change. When the instruction OUT 01 is executed, the result 7A is displayed at the simulated output port 01.

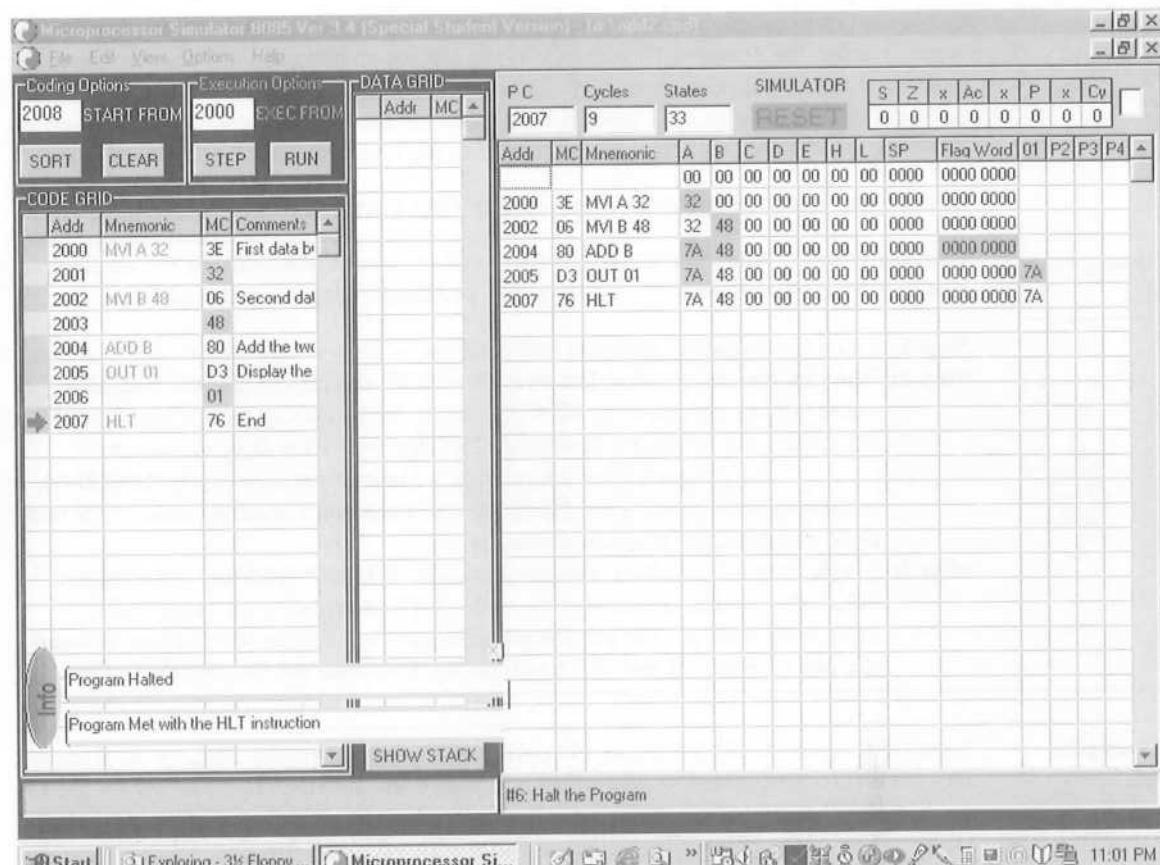


Figure H.2 Display of a Simulator Program

Step 5: Click → Reset on the Simulator screen. This clears the simulation screen.
Click → Run. The Simulator executes all the instruction and shows the final result.

To execute additional programs, go back to Step 3, load another code file, repeat Steps 4 and 5, and observe the results.

Hex Files If you open a hex file, the program loads Hex codes in the specified memory locations without mnemonics. To see mnemonics, you can use either Step or Run mode.

H.1.3 How to Modify, Add, and Delete Instructions from an Existing Program

The Simulator is not an editor; therefore, the edit functions are very limited. The Edit from the menu bar provides the capability of deleting and inserting one row at a time. You can select any row by clicking on the row. By clicking on Edit and Delete, a row can be deleted;

by clicking on Insert, a row can be added. However, the memory addresses are not renumbered to reflect the changes. Therefore, you can replace an instruction of fewer or the same number of bytes. If a new instruction is inserted, all remaining memory addresses need to be modified manually using a keyboard. You need to take the following steps to insert or delete a row. The following illustrations are from the same example discussed previously.

REPLACING INSTRUCTION HEX CODES AND DATA BYTES

Example H.1	<p>Load the program Add2 and define I/O port as 01 by using Options category. Replace the Hex data byte 48 in B with the data byte F3, and Single Step the instructions.</p> <p>Step 1: Open the Simulator program and click on → File → Open Code File.</p> <p>Step 2: You can replace the Hex data byte 48 with F3 directly in the MC (machine code) column by selecting 48. Click inside the box using the mouse. Enter F3 using the keys on the PC keyboard. The Hex byte F3 will be displayed.</p> <p>The program does not modify the instruction mnemonics MVI B 48 but displays the term “User Mode” in the mnemonics column for F3. This term does not affect the program execution.</p> <p>Step 3: Single Step the program. The result will be 25H in A with the carry flag set. When the Simulator executes the code 06 F3, it replaces the mnemonics MVI B 48 with MVI B F3 (the program does not display the comma between the two operands B and 48). Click on Reset to clear the simulation screen.</p>
-------------	--

Example H.2	<p>Load the program Add2 and define I/O port as 01 by using Options category. Replace the Hex code (06 48) of the instruction MVI B, 48H with the Hex code (0E F3) of the instruction MVI C, F3H, and replace the Hex code (80) of the instruction ADD B with the Hex code (81) of the instruction ADD C, and run the program. Assume that the Simulator is already open.</p> <p>Step 1: Click on → File → Open Code File. The Add2 program will appear as a List file, and the screen on the right side will be dark.</p> <p>Step 2: You can replace the Hex code and data byte 06 48 by 0E F3 directly in the MC (machine code) column. Select 06 as in Example H.1 and enter 0E. The instruction mnemonic MVI B is replaced by the term “User Mode.”</p> <p>Step 3: To enter the code F3, select 48 using the mouse and replace it with F3 using the keyboard.</p> <p>Step 4: To enter the code 81, select 80 and replace it with 81.</p> <p>Step 5: Run the program. The program will replace MVI B with MVI C, and the result will be shown as before: 25 in A and the CY flag set.</p>
-------------	---

MODIFYING MNEMONICS

Modifying mnemonics is an editing process. This 8085 Simulator provides a limited editing procedure through the **Edit** menu on the bar. Replacing mnemonic by another equivalent byte-size mnemonic is a four-step process. Three steps are accomplished using the

Edit menu that provides three commands: **Delete Row**, **Insert Row**, and **Last Clicked Row as Current Row**. These commands can insert or delete only one row at a time. The fourth step involves specifying the memory address where a mnemonic should be inserted. This address is specified by entering the 16-bit address in the box labeled **Start From** in the Coding Option Grid. These steps are illustrated in Example H.3.

In program Add2, replace the instruction MVI B, 48H with the instruction MVI C, F3H and the instruction ADD B with ADD C, and run the program.

Example
H.3

Step 1: Click on → File → Open Code File. Select Add2 program. The Add2 program will appear as a List file, and the right side of the screen will be dark. Click on Options and insert 01 for an I/O port.

Click → View → Code Key Pad. Code Key Pad is displayed on the right side of the screen.

Step 2: The instruction MVI B, 48H is a two-byte instruction stored in locations 2002 and 2003. Therefore, the rows with the addresses 2002 and 2003 should be deleted.

Click on → 2002 → Edit → Delete Row (2002) → Edit → Delete Row (2003) → Edit → Insert Row (Blank) → Edit → Insert Row (Blank). The program will insert two blank rows. The memory addresses of the remaining instructions will remain the same as before.

Step 3: In preparation for writing the instruction MVI C, F3H in location 2002H:

Click on the first blank row. Enter the address 2002 in the box Start From in the Coding Option. Click → Edit → Last Row as Current Row.

Step 4: To write the instruction MVI C, F3H in location 2002H:

Go to Code Key Pad. Click on → MVI in Data Transfer Group → C in Registers → F and 3 in Hex Digits. The memory addresses, mnemonics, and Hex codes will be written in appropriate columns.

To write the instruction ADD C, click → ADD → C

Step 5: To execute the program:

Enter the address 2000 in the box Execute From in Execution Options. Click on Run. Observe the output and the flags. Click → Reset to clear the Simulator screen.

REPLACING AN INSTRUCTION WITH A SMALLER SIZE INSTRUCTION

If a two-byte instruction is replaced with a one-byte instruction, a memory location is left without an instruction. The Simulator assumes the NOP instruction in that location if you use Step mode. It can also prompt you that it is executing the NOP instruction if this option is enabled in the Prompt option. In the Run mode, the simulator stops the execution that can be continued in the Step mode. However, the program can be properly executed in either mode if the NOP instruction is inserted in the empty memory location.

Example H.4 Load program Add2 and define the I/O port at 01. Replace the instruction MVI B, 48H with the instruction MOV B, A and execute the program.

Step 1: Click on → File → Open Code File and select the Add2 program. The Add2 program appears as a List file, and the right side of the screen is dark.

Click → View → Code Key Pad. Code Key Pad is displayed on the right side of the screen.

Step 2: The instruction MVI B, 48H is a two-byte instruction stored in locations 2002 and 2003. Therefore, the rows with the addresses 2002 and 2003 should be deleted.

Click on → 2002 → Edit → Delete Row (2002) → Edit → Delete Row (2003) → Edit → Insert Row (Blank) → Edit → Insert Row (Blank). The program will insert two blank rows. The memory addresses of the remaining instructions will remain the same as before.

Step 3: In preparation for writing the instruction MOV B, A in location 2002H:

Click on the first blank row. Enter the address 2002 in the box Start From in the Coding Option.

Click → Edit → Last. Click Row as Current Row.

Step 4: To write the instruction MOV B, A in location 2002H:

Go to the Code Key pad. Click on → MOV in Data Transfer Group → B followed by A in Registers. The memory addresses, mnemonics, and Hex codes will be written in appropriate columns.

Click → NOP in I/O and Machine Control Group.

Step 5: To execute the program:

Enter the address 2000 in the box Execute From in Execution Options.

Click on Run. Observe the output and the flags. The output should be 64H with no flags set.

REPLACING AN INSTRUCTION WITH A LARGER SIZE INSTRUCTION AND ADDING AN INSTRUCTION

When a larger size instruction or a new instruction is added, the program needs more memory locations than already available in the existing program. These programs can be added by providing an appropriate number of insert commands. However, the memory addresses of the remaining instructions remain the same as before. Therefore, these instructions must be reentered.

Example H.5 Load the program Add2 and define the I/O port at 01. Add the instruction STA 2050H (32 50 20) before the OUT instruction and execute the program.

Step 1: Click on → File → Open Code File and select the program Add2. The Add2 program appears as a List file, and the right side of the screen is dark.

Click → View → Code Key Pad. Code Key Pad is displayed on the right side of the screen.

Step 2: The instruction STA 2050H is a three-byte instruction that should be stored in Hex locations 2005-2006-2007. However, these memory locations already hold

the codes for OUT and HLT instructions. Therefore, three rows must be inserted starting from location 2005H.

Click on → 2005 → Edit → Insert Row (Blank) → Edit → Insert Row (Blank) → Edit → Insert Row (Blank). The program will insert three blank rows, pushing down the instructions OUT and HLT without changing the addresses of these instructions. Therefore, the memory address of the OUT instruction still will be 2005H.

Step 3: In preparation for writing the instruction STA 2050H in location 2005H:

Click on the first blank row.

Enter the address 2005 in the box Start From in the Coding Option.

Click → Edit → Last Row as Current Row.

Step 4: To write the instruction STA 2050H starting from location 2005H:

Click on → STA in Data Transfer Group → 2, 0, 5, and 3 in Hex Digits.

The memory addresses, mnemonics, and Hex codes will be written in appropriate columns.

Step 5: To change the memory addresses of the instructions OUT and HLT, select the memory location of the OUT instruction and enter the address 2008. Continue to enter in the subsequent memory locations by incrementing the addresses in a sequence.

Step 6: To execute the program:

Enter the address 2000 in the box Execute From in Execution Options.

Click on Run. Observe the output and the flags. The Data Grid will show the memory address 2050 and the sum 7A.

H.1.4 Writing a New Program and Generating a List File

Write instructions to subtract two bytes already stored in memory addresses 2051H and 2052H. The location 2051H holds the byte 9FH and 2052H holds the byte 25H. Subtract the first byte, 9FH, from the second byte, 25H, and store the answer in memory location 2053H. Write instructions beginning at memory location 2030H.

Example
H.6

This is the same problem as in Section 2.6 except the data bytes are different. To write this program, data bytes from memory locations 2051H and 2052H must be copied in processor registers and the second data byte subtracted from the first byte. The mnemonics are as follows:

Instructions

LDA 2051H	;Get the first byte in A
MOV B, A	;Save the byte in B
LDA 2052H	;Get the second byte
SUB B	;Subtract the first byte
STA 2053H	;Save the answer in 2053H
HLT	

GENERATING A NEW FILE USING THE SIMULATOR

Step 1: Open the simulator. Enter 2030 in the box Start From. (If you do not see Code Key Pad, click → View → Code Key Pad.)

Step 2: Go to Code Key Pad. Click → LDA → 2, 0, 5, 1. Enter the remaining five instructions (Step 3).

Step 3: To enter data, go to Data Grid.

Click under the Addr column and enter 2051. Click under MC column and enter the data byte 9F. Press the Enter key of the PC keyboard.

Enter 25 in the next location. Press the Enter key. This completes generating a new file.

Step 4: To execute the program, click → Run.

Step 5: Observe the result. If we subtract 9FH from 25H, the answer is 86H. We are subtracting a larger number from a smaller number; therefore, the answer is negative 86H in 2's complement. The simulator will set the Sign and Carry flags.

H.2

HOW TO USE ENVI85

As mentioned earlier, this is an integrated environment of three programs included as an integrated package. The primary advantage of an integrated package is that it enables the user to write, assemble, and simulate the execution of a program without having to call or open three different programs. The following overview is copied from the manual after editing for additional clarification, which is included on the CD packaged with the text as a Word file ENVI85.

H.2.1 ENVI85 Overview

GETTING STARTED

The command ENVI85 (now replaced by Envi85a) with a given file runs the program. If no filename is specified, then ENVI85 asks for the name of a file. If you enter a filename without a suffix, then ENVI85 uses the suffix <.ASM>. If you do not enter a filename, ENVI85 begins with NONAME.ASM>. You can edit any ASCII file by entering the appropriate filename.

Once you enter the filename, you are in the Editor. If the file named already exists, the contents of the file appear in the Editor; if not, the Editor creates a new empty file. While in the Editor, you can perform various operations. You can edit a file, create a program, compile a program, or run a program. For ease of use, various Editor options are listed across the bottom of the screen.

At the bottom of each screen is the Options Line. This list gives you the available function key operations. This list changes depending on where you are in the environment. It also displays information while you are debugging your program.

AN OVERVIEW

Many options are available to you when you use ENVI85. In case you are one who never reads all the directions, this section gives the information to get you started. Detailed de-

scriptions of the integrated environment and the commands are in the CD (Word file ENVI85).

The Editor Window When you first start ENVI85 and have loaded a file, <filename.ext>, you are in the Editor Window. The full range of Editor commands and a number of function key operations are available. The Editor function key operations are:

- <f2> Save the file and continue editing
- <f3> Load a new file to edit
- <f4> See a directory of files
- <f5> Compile the program
- <f6> Run the simulator, recompiling if necessary
- <f7> Mark the beginning of an editor block
- <f8> Mark the end of an editor block
- <f10> Quit and return to DOS

The Compiler Window When you press the <f5> function key, the Compiler Window appears. ENVI85 compiles the program, showing you what line it has reached. You have the option of stopping the compilation by pressing the spacebar and generating a user interrupt. After you press a key, ENVI85 returns to the Editor at the line where you stopped the compilation.

If ENVI85 finds an error, it displays that error's message in an Error Window and waits for you to press a key. ENVI85 then returns to the Editor and places the cursor at a possible location for the error. ENVI85 continues to display the error message above the function key help line.

Upon finding no errors, ENVI85 asks whether the user would like to save either a file with the entire compilation data or just the symbol table. If the user wants either of these files, ENVI85 saves it under the name <filename.prn>. ENVI85 also creates a file named <filename.hex>. This file has the hexadecimal code of the program ready to run. After compiling the program and saving the desired files, ENVI85 returns to the Editor.

The Simulator Window When you press the <f6> function key, a Starting Address Window replaces the Editor. This window asks for the starting address of the program. the default starting address is 0100H. This sets the starting contents for the program counter (PC).

In the Simulator window, the following function key operations are available:

- <f1> Run the program
- <f2> Restart the program
- <f3> Single-step the program
- <f4> Set a breakpoint
- <f5> View the internal registers and memory
- <f7> Undo the previous instruction
- <f10> Quit the simulator and return to the Editor

1. Pressing <f1> starts the program running at the address in the program counter. While it is running, you can press <f9> to stop the program. This works unless the Options

Line message shows: "BIOS is waiting for an ASCII input to be typed." At this time, the simulator BIOS is waiting for an ASCII key to be pressed.

When you stop the program, you return to the Simulator Window. You return to the Editor by pressing the <f10> function key. If the program stops itself, ENVI85 returns to the Editor.

2. If you press <f2>, the Starting Address Window will reopen. This resets the contents of the program counter (PC).
3. If you press <f3>, you execute the instruction at the program counter.
4. If you press <f4>, a Set Breakpoint Window appears. You enter the address where you want the program to stop. The address can be entered by value or label; we recommend using a label. The default breakpoint is 0FFFFH.
5. If you press <f5>, a Register Window appears and you view the status of your program. The internal registers, register pairs, flag registers, stack counter and part of the stack, program counter with a portion of code, and some user-definable memory locations, in hexadecimal and ASCII, are shown. This window disappears each time the simulator BIOS is called or another window is opened.
6. If you press <f7>, you "undo" the program instruction that has just been executed.

The Word file ENVI85 includes the complete manual that explains the integrated environment in greater detail.

H.2.2 Writing (Editing) and Assembling a Program Using Envi85a

Envi85a is included with the Simulator 8085, but this integrated program cannot be accessed by clicking on the Simulator icon. You need to access the Envi85a program by specifying the complete path, and you can specify the path by one of the methods suggested here. If you write the complete path, you can access either of the programs. See Section H.1.1 for additional information. The complete path of these programs can be specified as one of the following ways:

1. Click on Start → Select Programs → Select Windows Explorer, and click Select Program Files → double-click on Infotech Solutions → Microprocessor 8085 → Envi85a. This will open the Editor.
2. Double-click on all the following icons: My Computer → C: Drive → Program Files → Infotech Solutions → Microprocessor Simulator 8085 → Envi85a or Micro3 for the 8085 Simulator.
3. This long path can be shortened if you install the program under the folder 8085 as suggested in Section H.1.1. The path will be:

Double-click → My Computer → C: Drive → 8085 → Envi85a for assembly or Micro3 for the 8085 Simulator

After you open the Editor in Envi85a, a program can be written and assembled as follows. Now you will be operating under DOS mode with the PC keyboard (and not a mouse).

the first byte, 9FH, from the second byte, 25H, and store the answer in memory location 2053H. Write instructions beginning at memory location 2030H. This is the same program as in Example H.6.

Step 1: Open the Envi85a program as suggested previously. The screen will open as New File and ask for a filename.

Step 2: Assign a filename of up to eight characters with the extension.asm of three characters; otherwise the program assumes the extension to be .asm. Let us give a name: Subtract.asm. Press Return.

Step 3: The editing screen will open showing the name of the file, Line 1, Column 1, a tool bar at the top of the screen, and various functions that can be performed with various keys from f2 to f10 (except f9). Begin typing your program with the ORG statement in column 10 and the program as shown. The last command in the program must be the End statement.

ASM (Source) File

```

ORG 2030H      ;Begin assembly at 2030H
START: LDA 2051H ;Get the first byte in A
          MOV B, A   ;Save the byte in B
          LDA 2052H ;Get the second byte in A
          SUB B      ;Subtract first byte from
                      ;second byte
          STA 2053H ;Save the result in 2053H
          HLT        ;End of program
          ORG 2051H ;Store data beginning 2051H
DATA:  DB 9FH, 25H ;Data bytes
          END        ;End of assembly

```

Step 4: Press <f5> to compile (assemble) the program. If it finds an error, it will give an error message and ask you to press any key to go back to the editing (writing) process. If you press a key, it will take you back to the line where the error occurred. Correct the error and press <f5> to compile again. If it finds another error, it will give another error message. The process will continue until all syntax errors are corrected.

Step 5: After all errors are corrected, it will compile and show a message indicating no errors and the name of the Hex file and ask whether you want a prn (print) file. It can also generate a symbol table. At the end of the assembly (compilation) process, the assembler will have three files: (1) ASM: Assembly source file (shown in Step 3 above), (2) HEX: Hex code, and (3) PRN: Print file as follows.

Hex File

```
:0C2030003A5120473A522090325320765B
:022051009F25C9
:0000000000
```

PRN (Print) File

ENVI85 1.14 by E. J. Kay & S. A. Fedyschyn Copyright, 1991. Page 1

```

0000          ORG 2030H ;Begin assembly at 2030H
2030 3A 51 20   START: LDA 2051H ;Get the first byte in A
2033 47          MOV B, A ;Save the byte in B
2034 3A 52 20          LDA 2052H ;Get the second byte in A
2037 90          SUB B ;Subtract first byte from
                   ;second byte
2038 32 53 20          STA 2053H ;Save the result in 2053H
203B 76          HLT ;End of program
203C          ORG 2051H ;Store data beginning 2051H
2051 9F 25   DATA: DB 9FH,25H ;Data bytes
2053          END ;End of assembly

```

ENVI85 1.14 by E. J. Kay & S. A. Fedyschyn Copyright, 1991. Page 2

Symbol	Address/Value	Type
<hr/>		
DATA	2051	Address (data)
START	2030	Address

Step 6: You can run the program by pressing <f6> and also observe various simulation options as listed in the Simulator Window described in Section H.2.1 (ENVI85 Overview). Or skip this step and go to Step 7 to observe the simulation performed by Simulator 85.

Step 7: Open the Micro3—Simulator 8085 program. Review the path as explained in Section H.2.2. Now follow the steps from Step 3 in Section H.1.2—How to Execute an Existing Program Stored on a Disk in Code or Hex Format. These steps are repeated here for easy reference:

Click → File → Open Hex File

All Hex files will be shown. Select the Subtract Hex file. The simulator program will load the Hex file without the mnemonics. To see the mnemonics, execute the Run command, and all mnemonics will appear on the List file screen and the executed program on the Simulator screen. Click on Reset to clear the Simulator screen. Click on Step, continue the execution until the end of the program, and observe the contents of the registers and the flags.

Index

- Accumulator, 14
A/D Converters. *See* Data converter, analog-to-digital
Alphanumeric Code, 523
ALU, 6, 10–11, 13
Arithmetic-logic unit. *See* ALU
Arithmetic operations
 memory-related, 241–47
 for 16 bits or register pairs, 236–38
ASCII, 12, 16, 524
 codes, 735–36
 converting to binary code, 332–34
Assembler, 16, 357, 359–63
 advantages of, 362–63
 directives for, 361–62
 8085 assemblers, 801, 810–814
 use in writing programs, 363–68
Assembly language, 3, 14, 43
 for 8085 MPU, 15–17
 format for, 362
Assembly language programs
 for 8085 MPU, 31–53
 tools for developing, 356–57
 assembler, 357
 debugger, 357
 editor, 356–57
 loader, 357
 writing, 16–17, 211–13
 documentation, 212–13
- Baud, 524
BCD
 addition, 334–37
 packed, 324
 subtraction, 337–38
BCD-to-binary conversion, 324–27
 multiplication in, 342–44
BCD-to-seven-segment-LED code conversion, 329–32
Bidirectional buffer, 84
Binary digit, 5
Binary-to-ASCII code conversion, 332–34
Binary-to-BCD conversion, 327–29
Bit. *See* Binary digit
Branch instructions, 36, 49, 204–10
Buffers, 83–84
 bidirectional, 84
 tri-state, 83–84
Buses
 bidirectional, 33
 for 8085 MPU, 33
 address, 33, 98, 587
 control, 33
 data, 33, 587
 demultiplexing AD7-ADO, 102–04
 timing, 100–102, 111
 system, 12
- modular design approach, 211
unidirectional, 33
- Byte, 13
- CD-ROM, 353
Central processing unit, 6
Chip select signal, 66, 67, 68
Compiler, 17
Computer languages
 assembly language, 14
 for 8085 MPU, 15–16
 format for, 361
 high-level, 14, 17–18
 machine language, 14–15
 writing and executing, 16–17
Computers
 applications of, 20–23
 digital, 4–13
CPU, 6
 input/output devices for, 6, 80–81
 large, 20
 medium-sized, 20–21
 memory, 5–6, 11
 microcomputers, 21–23
 programming for, 5
Conditional call instructions, 315–16
Conditional jump, 36
Conditional loop, 228–230
Conditional return instructions, 315, 316

- Continuous loop, 228–29
 Control unit, 6, 10
 Conversions
 ASCII-to-binary code, 332–34
 BCD-to-binary, 324–27
 BCD-to-seven-segment-LED code, 329–32
 binary-to-ASCII code, 332–34
 binary-to-BCD, 327–29
 Counter, 276
 hexadecimal, 282–85
 zero-to-nine, 285–88
 Counter design, with time delay, 281–82
 Counter programs, debugging, 290–92
 Counting, 229–30
 CPU. *See* Central processing unit
 Cross assembler, 361–63
 advantages of, 362–63
 format of, 362
 Cross-assembler program, and MS-DOS operating system, 358
- D/A converters. *See* Data converters, digital-to-analog
 Data converters
 analog-to-digital (A/D), 414–22
 interfacing 8-bit A/D converter, 416–22
 successive approximation A/D, 414–16
 digital-to-analog (DA), 403, 404–14
 circuits for, 405–07
 interfacing 10-bit D/A converter, 411–14
 microprocessor-compatible, 411
 interfacing, 403–22
 specifications for, 664–65
 Data transfer (copy) from memory to microprocessor, 233–35
 from microprocessor to memory or directly into memory, 235–36
 to register pairs (LXI), 232–33
 16-bit, 338–39
 and 16-bit arithmetic instructions, 232–41
 Debugging programs, 18, 215, 261–63, 357
- counter and time delay, 290–92
 debugger, 357
 dynamic, 215, 261–63
 machine code, 215
 static, 215
 Decoders, 85–86
 absolute vs. partial 146–47
 examples of, 86
 use to interface input/outputs, 147–49
 Demultiplexers. *See* Decoders
 Device address pulse, 144
 Direct memory access (DMA), 397–98, 514–20
 DMA. *See* Direct memory access
 Drivers. *See* Serial I/O
 Duplex transmission, 526
 DVD, 354
- EE-PROM, 79, 80
 Eight-bit addresses, input/outputs with, 80–81
 Eight-bit MPUs, 607–11
 HD64180, 608, 610–11
 MC6800, 608, 610
 Z80, 8, 11, 608–10
 8008 MPU, 8
 8080 MPU, 8, 9
 8085 MPU, 8, 9, 96–109
 assembly language for, 14–16
 assembly language programming for, 31–53
 bus organization, 59–60
 address bus, 59–60, 96, 587, 589
 control bus, 60
 data bus, 60, 587, 589
 demultiplexing bus AD7-ADO, 102–104
 timing, 100–02, 111
 control signals, 98, 588, 590, 591
 generating, 104–05
 table, 99
 decoding and executing instructions, 108
 design, 589–97
 control signals, 591
 externally-triggered signals, 591–92
- frequency and power requirements, 591
 externally-initiated signals, 99–100
 table, 99
 hardware model, 33
 instruction format for, 37–42
 opcode format, 39–40
 word size, 37–39
 instruction set for, 31–53, 737–83
 arithmetic operations, 35, 48
 branching operations, 36, 49, 204–10
 bytes, recognizing number in instructions, 45–46
 data transmission (copy) operations, 35, 47, 176–80
 logical operations, 36, 48–49, 196–204
 machine control operations, 36, 50
 overview of, 46–50
 table, 776–77, 778–83
 internal architecture, 105–08
 internal data operations, 61–62
 accumulator, 33
 flags, 33–34
 program counter, 34
 registers, 33
 stack pointer, 34
 interrupt process, 376–93
 additional, 385–93
 multiple interrupts and priorities, 384–85
 restart instructions, 378–79
 table, 99
 machine cycles for, 111
 memory read, 112–14
 opcode fetch, 111–12
 recognizing, 114–15
 status of (table), 99
 machine language for, 14–15
 memory interfacing with, 116–129
 microprocessor communication and bus timings, 100–02
 pinout and signals for, 96–100
 power supply and clock signals, 98
 programming model for, 32–33, 62
 accumulator, 33

- flags, 33–34
program counter, 34
stack pointer, 34
serial input/output lines for, 534–37
ports, 100
status signals, 98, 99
vectored interrupts, 99–100
writing, assembling, and executing
 simple program for, 42–46
- 8085-based microcomputer example
of, 109–16
- 8086/8088 iAPX MPU, 613–23
- 8155 multipurpose programmable
devices, 432–49
- 8205 decoder, 86
- 8251A programmable communication
interface, 540–58
- 8254 (8253) interval timer, 494–505
- 8255A programmable peripheral
interface, 460–93
- interfacing keyboard and seven
segment display with, 479–87
- transferring bidirectional data
between two microcomputers,
488–93
- 8237 DMA controller, 514–20
- 8259A interrupt controller, 396–97,
505–13
- 8279 programmable keyboard/
display interface, 450–56
- 80186 MPU, 623–24
- 80286 MPU, 623–24
- 80386 MPU, 627–29
- 80486, 629–30
- Encoders, priority, 88
- EPROM, 79, 80
 memory, 581–85
- Errors. *See* Programs, debugging
- File, 354
- Flag register, 34
- Flags, 33–34, 107
- Flash memory, 79–80
- Flip-flops
 D, 88–89
 clock, 88
 latch, 88–89
as storage elements, 64–67
- Floppy disk, 21–22, 353, 354
- Foldback memory address, 125
- 4004 MPU, 8
- Framing, 525
- Gates, 77
- Global symbols, 602
- Hand assembly, 16
- Handshake signals, 430–31
- Hard disk, 353
- Hardware, 4
- HD64180 MPU, 610–11
- ICs, 7, 404
- In-circuit emulator, 603–05
- Indexing, 230
- Input devices, interfacing, 155–56
 hardware, 155–56
 interfacing circuit, 156
 multiport addresses, 156
- Input interfacing, 147
- I/O (input/output) memory-mapped,
12, 81, 157–62
- I/O address pulse. *See* Device
 address pulse
- I/O devices, 6, 12, 80–81
 with 8-bit addresses, 80–81
 serial, and data communications,
 523–58
 with 16-bit addresses, 81
- I/O device selection and data transfer,
144–46
- I/O execution, 141–44
 in instruction, 143–44
 out instruction, 141–43
- I/O instructions, peripheral, 140–41
- I/O interfacing circuits, testing and
troubleshooting, 163–64
- Input unit, 12
- Instruction, 14–15, 34–37
- Instruction cycle, 104
- Instruction fetch, 74–75
- Instruction set, 14, 35
- Instruction word size
 one-byte, 37, 46
 two-byte, 38, 46
 three-byte, 38, 46
- Integers
 signed, 41
 unsigned, 41
- Integrated circuits. *See* ICs
- Integrated RAM, 76, 80
- Interface devices, programmable. *See*
 SDK-85 programmable
 interface devices
- Interfacing, logic devices for, 83–90
 buffer, 83–84
 decoder, 85–86
 D flip-flops, 88–90
 encoders, 86–88
 tri-state, 83–84
- Interfacing devices, 83
- Interpreter, 17–18
- Interrupt controller, programmable
(8259A), 396–97
- Interrupt process for 8085 MPU,
376–93
 restarting with software
 instructions, 393–95
- Interrupts, 375–98
 8085 interrupt, 376–93
 non-maskable. *See* Trap
 vectored, 385–93
 RST 5.5, 385, 386–90
 RST 6.5, 385, 386–90
 RST 7.5, 385, 386–90
 TRAP, 385–386
- Jumps
 conditional, 206–209
 unconditional, 204–206
- Key debounce technique, 484–85
- Language. *See* Computer languages
- Large-scale integration. *See* LSI
- Latch storage elements. *See* Flip-
flops, as storage elements
- LCD, 26, 564–73, 724–33
 controller driver, 567, 726–33
 interfacing, 569
- LED, 6, 12
- Light-emitting diode. *See* LED
- Line driver, 543
- Line receiver, 543

- Liquid Crystal Display. *See LCD*
- Logic devices, for interfacing, 564–73
- Logic operations
 compare, 254–61
 rotate, 247–54
- Logic state analyzer, 605
- Looping, 228–30
 conditional loop, 229–30
 continuous loop, 228–29
 and time delay using loop within a loop, 279–80
- LSI, 7
- Machine code, debugging, 215
- Machine cycles, 104
 for 8085 MPU, 111–16
- Machine language, 14
 for 8085 MPU, 14–15
- Mainframe computers. *See Computers*, large
- Manual assembly. *See Hand assembly*
- Masked ROM, 79
- Matrix keyboard
 combining with scanned display, 580
 hardware approaches to, 580–81
 interfacing, 573–81
 program for, 576–80
- MC6800 MPU, 610
- MC68000 MPU, 624–25
- MCS-51 microcomputer, 633–34
- MCTS. *See Microprocessor-controlled temperature system*
- Medium-scale integration. *See MSI*
- Memory, 5–6, 11, 63–80
 address lines and calculations, 73–74
 arithmetic operations related to, 241–47
 instructions, 242–47
 classifications of, 76–80
 prime, 76, 77
 storage, 76, 80
- DRAM, 77–78
 dynamic, 77–78
- flash memory, 79
- flip-flop storage elements, 64–69
- and instruction fetch, 74
- interfacing with wait states, 585–88
- SRAM, 77
- static, 77
- structure and requirements of, 116
- technological advances in, 80
- word, 63, 74
- Memory addressing, 119–123
 foldback address, 125
 memory map, 69
 mirror address, 125
- Memory chip
 requirements of, 68–69
- Memory design, 581–88
 EPROM memory, 581, 583–85
 RAM, 45, 46
- Memory interfacing, 116–32
 address decoding, 119–23
 circuit for, 120–23
 foldback memory address, 125
 and memory map, 121–23
 mirror memory address, 125
 troubleshooting, 129–30
- Memory map, 69–72, 121–23
 of 1K memory chip, 72–73
- Memory-mapped input/output, 81, 157–62
- Memory pointer, 34
- Memory system, for SDK-85
 microcomputer, 123–26
- Microcomputer, 21
 business, 21
 personal, 21–22
- single-board, 22–23, 129–32
 designing, 592–97
 8085-based, 132–33
- single-chip, 23, 633–36
 MCS-51, 633–34
 MC68HC11, 634–36
- Microcontrollers, 6, 23, 633–36
 block diagram, 7
- MCU, 6
- Microprocessor, 10
 applications of
 matrix keyboard, 573–81
 scanned display, 563, 564–69
- communication and bus timings, 100–102
- operation of, 58–63
- parts of, 10
 arithmetic/logic unit, 10
 control unit, 10
 register unit, 10
- various types
 4008, 8
 8080, 8–9
 8085, 8–9, 14, 96–109
 8088, 613–23
 80186, 613
 80286, 613
 80386, 9, 627–29
 80486, 627–30
 HD64180, 610–11
 iAPX 8086/8088, 613–23
 MC6800, 610
 MC68000, 624–25
 Z80, 8, 608–10
- Microprocessor-based system
 components of, 9–13
 input unit, 12
 memory, 11
 microprocessor, 10
 output unit, 12
 peripherals, 10, 12
 system bus, 12
 example of, 81–82
- Microprocessor-controlled
 temperature system, 1, 23, 25
 designing memory for, 126–129
 interfacing I/O devices, 467–469
 system hardware, 23
 system application, 90
- Microprogramming, 11
- Minicomputers. *See Computers*, medium-sized
- Modulo ten counter. *See Counter*, zero-to-nine
- Monitor program, 11
- Motorola, 610, 624
- MS-DOS operating system, 355
 and cross-assembler program, 358–60
- MSI, 7
- Multiplexing, 96

- National Semiconductor, 83
Nibble, 13
Non-volatile RAM, 78
Numbers, converting to other bases, 637–39
Number systems, 637–45
- Object code, 17
Octal bus transceiver. *See*
 Bidirectional buffer, 54, 654
Operating systems, 18, 354–56
 CP/M, 355
 GUI, 18
 Linux, 18, 356
 ME, 18, 356
 MS-DOS, 355
 NT, 18
 OS/2, 18, 355–56, 632
 UNIX, 18, 356, 632
 Windows 95 (98), 18, 356, 632
Operation code, 37
Output displays, interfacing, 150–55
Output unit, 6, 12
- Parameter passing, 314–15
Parity
 parity check, 527–28
 parity flags. *See* Flags
Pentium microprocessor, 628–29
Pentium Pro-Processor, 631
Peripheral devices, programmable.
 See Programmable peripheral devices
Peripheral-mapped input/output, 80–81
Peripheral operations, 62–63
 hold, 62–63
 interrupt, 62–63
 ready, 62–63
 reset, 62–63
Peripherals, 10
Prime memory, 63, 76–80. *See also*
 RAM, ROM
Primer trainer, 23, 645–57
Program, 4, 175
Program counter, 34
- Programmable interface devices. *See*
 SDK-85 programmable
 interface devices
Programmable peripheral devices
 8254 (8253) interval timer, 494–505
 8255A interface, 460–93
 8237 DMA controller, 514–18
 8259A interrupt controller, 505–13
 general purpose, 459–520
Programming, and digital computers, 5
Programs
 debugging, 18, 215, 261–63,
 290–92, 359
 writing with assembler, 363–68
PROM, 78, 79
Prototype building and testing, 602
- RAM, 11, 77, 80
 design of, 581
 DRAM, 77–78
 integrated, 80
 non-volatile, 80
 SRAM, 77
 zero-power, 80
Random Access Memory. *See* ROM
Read-Only Memory. *See* ROM
Read signal, 64
Registers, programmable, 62
Register unit, 10
Restart
 instructions, 378–79, 767
 table, 378
RISC processors, 631–32
ROM, 11, 77, 78–79
 EE-PROM, 78, 79
 EPROM, 78, 79
 masked, 79
 permanent, 78
 PROM, 79
 semipermanent, 78
R/WM (Read/Write memory). *See*
 RAM
- Scanned multiplexed display
 interfacing circuit, 564–67
hardware approaches to, 580–81
program for, 568–73
with matrix keyboard, 580
- SDK-85 microcomputer
 absolute vs. partial decoding, 126
 interfacing 8155 memory section, 123–26
memory system for, 123–26
 8155, 432–49
 8279 programmable
 keyboard/display, 450–56
 with handshake signals, 430–31
74LS245 transceiver, 427–28
 with status register, 429–30
Semiconductor technology, advances
 in, 7–8
Serial I/O
 alphanumeric codes, 524, 525
 asynchronous software-controlled, 534–37
 basic concepts, 523–34
 data communication over telephone
 lines, 528–29
 driver/receiver MAX 232, 549
 for 8085 MPU, 537–40
 error checks in data
 communication, 527–28
 checksum, 528, 554, 555
 CRC, 528
 parity check, 527–528
hardware-controlled using
 programmable chips, 540–48
interfacing an RS-232 terminal, 546, 548
interfacing requirements, 524–25
line driver MC 1488, 549
line receiver MC 1489, 549
RS-232 signals, 531
standards in, 529–32
 RS-232C, 529, 530, 531–32
 RS-422A, 530, 532
 RS-423A, 530, 532
 transmission format, 525–26
74LS245 transceiver, 427–28
SID (Serial Input Data), 523, 537, 538–40
Simplex transmission, 526
Simulator, 801–810
Single-board microcomputers, 22–23
 designing, 592–96
 8085-based, 132–33

Single-chip microcomputers, 23, 633–36
 MCS-51, 633–34
 MC68HC11, 634–36
 Sixteen-bit data arithmetic
 group for, 339–41
 program counter, 341–42
 stack pointer, 341–42
 subtraction with carry, 344–46
 transferring (copying), 338–39
 Sixteen-bit MPUs, 612–26
 80186, 623–24
 80286, 623–24
 iAPX 8086/8088, 613–23
 MC68000, 624–25
 SLSI, 7–8
 Small-scale integration. *See* SSI
 SOD (Serial Output Data), 537–40
 Software, 4
 design, 597–602
 display module, 599
 functions, 601–02
 initialization, 598–99
 reading keyboard and placing
 byte in buffer, 599–601
 development systems
 microprocessor-based, 352–54

storage memory, 352–54
 system hardware, 352–54
 upward compatible, 8
 Sorting program, 259–61
 Source code, 17
 SSI, 7–8
 Stack, 295, 296–304
 Stack pointer, 34
 Storage memory, 76–77
 backup, 77
 secondary, 77
 Subroutine documentation and
 parameter passing, 314–15
 Subroutines, 295, 296–315
 advanced, 316–17
 multiple-ending, 317
 nesting, 317–18
 Super-large-scale integration. *See* SLSI
 Thirty-two bit MPUs, 626–30
 80386, 626, 627–29
 80486, 626, 627–29
 Pentium, 630–31
 Time delay, 276
 counter design with, 281–82
 techniques for (additional), 280–81
 using loop within a loop, 279–80
 using one register, 276–78
 using register pair, 278–79

Time-delay programs, debugging, 290–92
 Transmission rate. *See* Baud
 Transparent latch. *See* Flip-flops, D
 Tri-state logic devices, 83–84
 Troubleshooting tools, 603–05
 in-circuit emulator, 603–05
 logic state analyzer, 605
 T -state, 104
 2's complement, and arithmetic
 operations, 639–45
 UNIX, 356
 Unconditional jump, 204–06
 User memory. *See* RAM
 Very-large-scale integration (VLSI), 7–8
 Winchester disk. *See* Hard disk
 Word, 13
 Workstation, 22
 Write signal, 64
 Z80 MPU, 8, 608–10
 Zero-Power RAM, 80
 Zilog, 608
 Zip disk, 353