CHAPTER 9

Digital Interfacing

The major goal of this chapter and the next is to show you the circuitry and software needed to interface a basic microcomputer with a wide variety of digital and analog devices. In each topic we try to show enough detail so that you can build and experiment with these circuits. Perhaps you can use some of them to control appliances around your house or to solve some problems at work.

In this chapter, we concentrate on the devices and techniques used to get digital data into and out of the basic microcomputer. Then, in the next chapter, we concentrate on analog interfacing.

OBJECTIVES

At the conclusion of this chapter, you should be able to:

1. Describe simple input and output, strobed input and output, and handshake input and output.
2. Initialize a programmable parallel-port device such as the 8255A for simple input or output and for handshake input or output.
3. Interpret the timing waveforms for handshake input and output operations.
4. Describe how parallel data is sent to a printer on a handshake basis.
5. Show the hardware connections and the programs that can be used to interface keyboards to a microcomputer.
6. Show the hardware connections and the programs that can be used to interface alphanumeric displays to a microcomputer.
7. Describe how an 8279 can be used to refresh a multiplexed LED display and scan a matrix keyboard.
8. Initialize an 8279 for a given display and keyboard format.
9. Show the circuitry used to interface high-power devices to microcomputer ports.
10. Describe the hardware and software needed to control a stepper motor.
11. Describe how optical encoders are used to determine the position, direction of rotation, and speed of a motor shaft.

PROGRAMMABLE PARALLEL PORTS AND HANDSHAKE INPUT/OUTPUT

Throughout the program examples in the preceding chapters, we have used port devices to input parallel data to the microprocessor and to output parallel data from the microprocessor. Most of the available port devices, such as the 8255A on the SDK-86 board, contain two or three ports which can be programmed to operate in one of several different modes. The different modes allow you to use the devices for many common types of parallel data transfer. First we will discuss some of these common methods of transferring parallel data, and then we will show how the 8255A is initialized and used in a variety of I/O operations.

Methods of Parallel Data Transfer

SIMPLE INPUT AND OUTPUT

When you need to get digital data from a simple switch, such as a thermostat, into a microprocessor, all you have to do is connect the switch to an input port line and read the port. The thermostat data is always present and ready, so you can read it at any time.

Likewise, when you need to output data to a simple display device such as an LED, all you have to do is connect the input of the LED buffer on an output port pin and output the logic level required to turn on the light. The LED is always there and ready, so you can send data to it at any time. The timing waveform in Figure 9-1*a*, p. 246, represents this situation. The crossed lines on the waveform represent the time at which a new data byte becomes valid on the output lines of the port. The absence of other waveforms indicates that this output operation is not directly dependent on any other signals.

SIMPLE STROBE I/O

In many applications, valid data is present on an external device only at a certain time, so it must be read in at that time. An example of this is the ASCII-encoded keyboard discussed in Chapter 4. When a key is pressed,

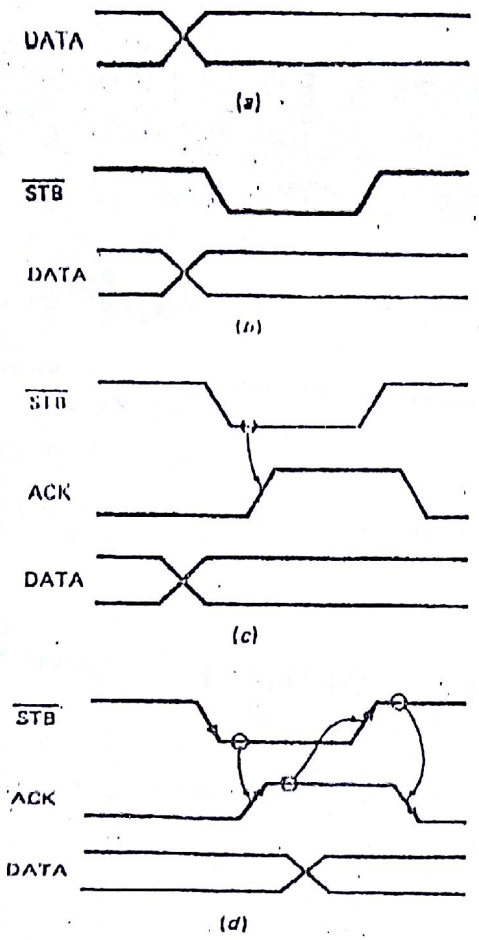

FIGURE 9-1 Parallel data transfer. (*a*) Simple output. (*b*) Simple strobe I/O. (*c*) Single handshake I/O. (*d*) Double handshake I/O.

circuitry on the keyboard sends out the ASCII code for the pressed key on eight parallel data lines, and then sends out a strobe signal on another line to indicate that valid data is present on the eight data lines. As shown in Figure 4-19, you can connect this strobe line to an input port line and poll it to determine when you can input valid data from the keyboard. Another alternative, described in Chapter 8, is to connect the strobe line to an interrupt input on the processor and have an interrupt service procedure read in the data when the processor receives an interrupt. The point here is that this transfer is time dependent. You can read in data only when a strobe pulse tells you that the data is valid.

Figure 9-1*b* shows the timing waveforms which represent this type of operation. The sending device, such as a keyboard, outputs parallel data on the data lines, and then outputs an $\overline{STB}$ signal to let you know that valid data is present.

For low rates of data transfer, such as from a keyboard to a microprocessor, a simple strobe transfer works well. However, for higher-speed data transfer this method does not work, because there is no signal which tells the sending device when it is safe to send the next data byte. In other words, the sending system might send data bytes faster than the receiving system could read them. To prevent this problem, a *handshake* data transfer scheme is used.

SINGLE-HANDSHAKE I/O

Figure 9-2 shows the circuit connections and Figure 9-1*c* shows some example timing waveforms for a *handshake data transfer* from a peripheral device to a microprocessor. The peripheral outputs some parallel data and sends an $\overline{STB}$ signal to the microprocessor. The microprocessor detects the asserted $\overline{STB}$ signal on a polled or interrupt basis and reads in the byte of data. Then the microprocessor sends an Acknowledge signal (ACK) to the peripheral to indicate that the data has been read and that the peripheral can send the next byte of data. From the viewpoint of the microprocessor, this operation is referred to as a *handshake* or *strobed input*.

These same waveforms might represent a handshake output from a microprocessor to a parallel printer. In this case, the microprocessor outputs a character to the printer and asserts an $\overline{STB}$ signal to the printer to tell the printer, "Here is a character for you." When the printer is ready, it answers back with the ACK signal to tell the microprocessor, "I got that one; send me another." We will show you much more about printer interfacing in a later section.

The point of this handshake scheme is that the sending device or system is designed so that it does not send the next data byte until the receiving device or system indicates with an ACK signal that it is ready to receive the next byte.

DOUBLE-HANDSHAKE DATA TRANSFER

For data transfers where even more coordination is required between the sending system and the receiving

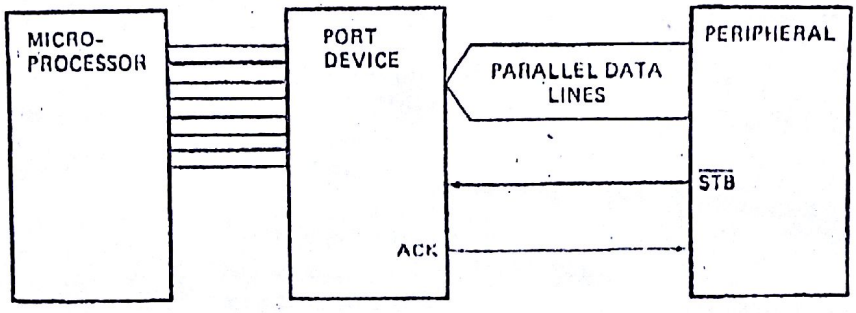

FIGURE 9-2 Signal directions for handshake input data transfer.

...m, a *double handshake* is used. The circuit connections are the same as those in Figure 9-2. Figure 9-1*d* shows some example waveforms for a double-handshake input from a peripheral to a microprocessor. Perhaps it will help you to follow these waveforms by thinking of them as a conversation between two people. In these waveforms each signal edge has meaning. The sending device asserts its STB line low to ask, "Are you ready?" The receiving system raises its ACK line high to say, "I'm ready." The peripheral device then sends the byte of data and raises its STB line high to say, "Here is some valid data for you." After it has read in the data, the receiving system drops its ACK line low to say, "I have the data, thank you, and I await your request to send the next byte of data."

For a handshake output of this type, from a microprocessor to a peripheral, the waveforms are the same, but the microprocessor sends the STB signal and the data, and the peripheral sends the ACK signal. In the accompanying laboratory manual we show you how to interface with a speech-synthesizer device using this type of handshake system.

Implementing Handshake Data Transfer

For handshake data transfer, a microprocessor can determine when it is time to send the next data byte on a polled or on an interrupt basis. The interrupt approach is usually used, because it makes better use of the processor's time.

The STB or ACK signals for these handshake transfers can be produced on a port pin by instructions in the program. However, this method usually uses too much processor time, so parallel-port devices such as the 8255A have been designed to automatically manage the handshake operation. The 8255A, for example, can be programmed to automatically receive an STB signal from a peripheral, send an interrupt signal to the processor, and send the ACK signal to the peripheral at the proper times. The following sections show you how to connect, initialize, and use an 8255A for a variety of handshake and nonhandshake applications.

8255A Internal Block Diagram and System Connections

Figure 9-3, p. 248, shows the internal block diagram of the 8255A. Along the right side of the diagram, you can see that the device has 24 input/output lines. Port A can be used as an 8-bit input port or as an 8-bit output port. Likewise, port B can be used as an 8-bit input port or as an 8-bit output port. Port C can be used as an 8-bit input or output port, as two 4-bit ports, or to produce handshake signals for ports A and B. We will discuss the different modes for these lines in detail a little later.

Along the left side of the diagram, you see the signal lines used to connect the device to the system buses. Eight data lines allow you to write data bytes to a port or the control register and to read bytes from a port or the status register under the control of the RD and WR lines. The address inputs, A0 and A1, allow you to selectively access one of the three ports or the control register. The internal addresses for the device are: port A, 00; port B, 01; port C, 10; control, 11. Asserting the CS input of the 8255A enables it for reading or writing. The CS input will be connected to the output of the address decoder circuitry to select the device when it is addressed.

The RESET input of the 8255A is connected to the system reset line so that, when the system is reset, all the port lines are initialized as input lines. This is done to prevent destruction of circuitry connected to port lines. If port lines were initialized as outputs after a power-up or reset, the port might try to output to the output of a device connected to the port. The possible argument between the two outputs might destroy one or both of them. Therefore, all the programmable port devices initialize their port lines as inputs when reset.

We discussed in Chapter 7 how two 8255As can be connected in an 8086 system. Take a look at Figure 7-8 (sheet 5) to refresh your memory of these connections. Note that one of the 8255As is connected to the lower half of the 8086 data bus, and the other 8255A is connected to the upper half of the data bus. This is done so that a byte can be transferred by enabling one device, or a word can be transferred by enabling both devices at the same time. According to the truth table for the I/O port address decoder in Figure 7-16, the A40 8255A on the lower half of the data bus will be enabled for a base address of FFF8H, and the A35 8255A will be enabled for a base address of FFF9H.

Another point to notice in Figure 7-8 is that system address line A1 is connected to the 8255A A0 inputs, and system address line A2 is connected to the 8255A A1 inputs. With these connections, the system addresses for the three ports and the control register in the A40 8255A will be FFF8H, FFFAH, FFFCH, and FFFEH, as shown in Figure 7-16. Likewise, the system addresses for the three ports and the control register of the A35 8255A are FFF9H, FFFBH, FFFDH, and FFFFH.

8255A Operational Modes and Initialization

Figure 9-4, p. 249, summarizes the different modes in which the ports of the 8255A can be initialized.

MODE 0

When you want to use a port for simple input or output without handshaking, you initialize that port in mode 0. If both port A and port B are initialized in mode 0, then the two halves of port C can be used together as an additional 8-bit port, or they can be used individually as two 4-bit ports. When used as outputs, the port C lines can be individually set or reset by sending a special control word to the control register address. Later we will show you how to do this. The two halves of port C are independent, so one half can be initialized as input, and the other half initialized as output.

MODE 1

When you want to use port A or port B for a handshake (strobed) input or output operation such as we discussed in previous sections, you initialize that port in mode 1.

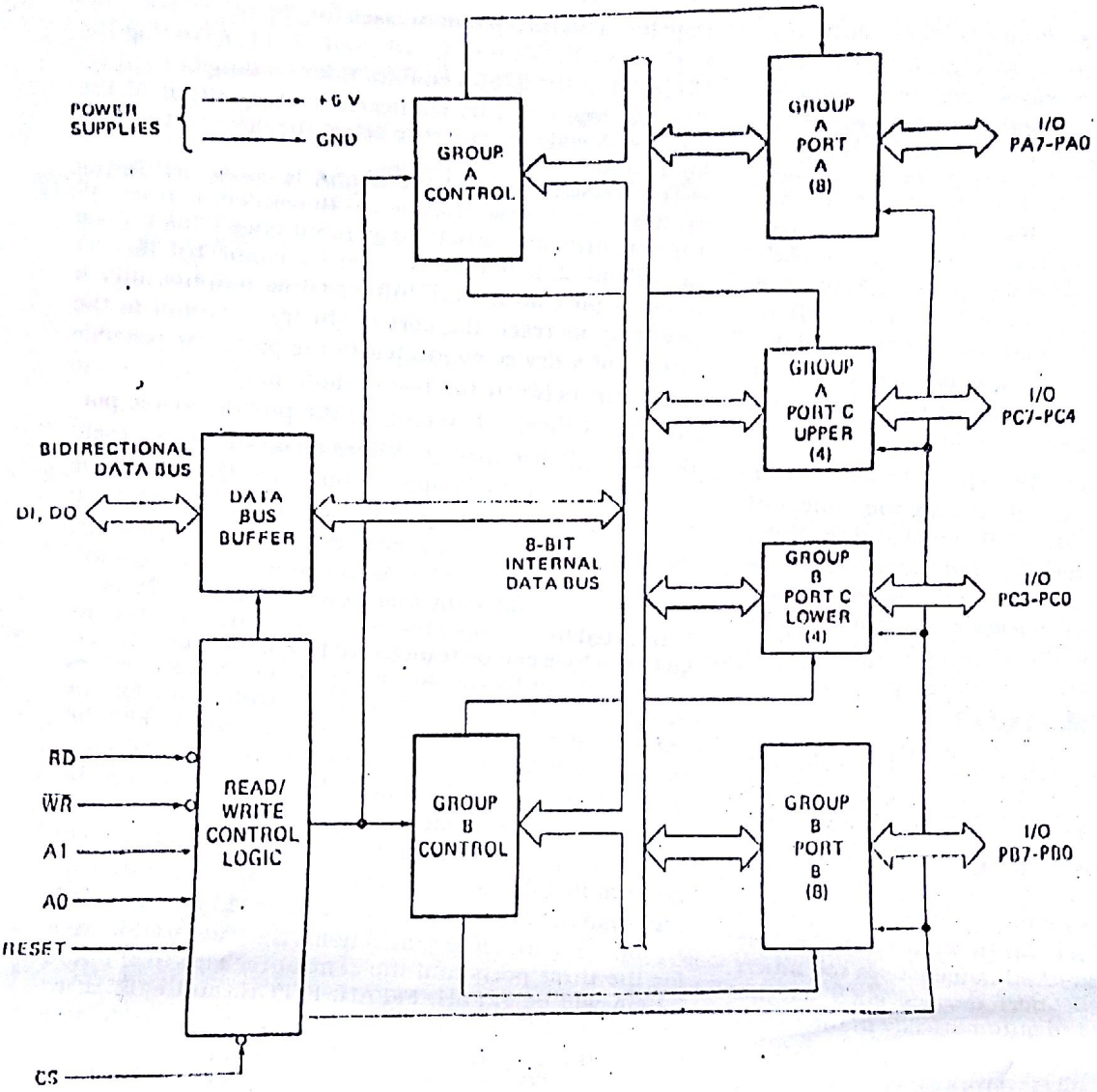

FIGURE 9-3 Internal block diagram of 8255A programmable parallel port device. (*Intel Corporation*)

In this mode, some of the pins of port C function as handshake lines. Pins PC0, PC1, and PC2 function as handshake lines for port B if it is initialized in mode 1. If port A is initialized as a handshake (mode 1) input port, then pins PC3, PC4, and PC5 function as handshake signals. Pins PC6 and PC7 are available for use as input lines or output lines. If port A is initialized as a handshake output port, then port C pins PC3, PC6, and PC7 function as handshake signals. Port C pins PC4 and PC5 are available for use as input or output lines. Since the 8255A is often used in mode 1, we show several examples in the following sections.

MODE 2

Only port A can be initialized in mode 2. In mode 2, port A can be used for *bidirectional handshake* data transfer. This means that data can be output or input on the same eight lines. The 8255A might be used in this mode to extend the system bus to a slave microprocessor or to transfer data bytes to and from a floppy disk controller board. If port A is initialized in mode 2, then pins PC3 through PC7 are used as handshake lines for port A. The other three pins, PC0 through PC2, can be used for I/O if port B is in mode 0. The three pins will be used for port B handshake lines if port B is initialized in mode 1. After you work your way through the mode 1 examples in the following sections, you should have little difficulty understanding the discussion of mode 2 in the Intel data sheet if you encounter it in a system.

Constructing and Sending 8255A Control Words

Figure 9-5 shows the formats for the two 8255A control words. Note that the MSB of the control word tells the 8255A which control word you are sending it. You use the *mode definition control word* format in Figure 9-5*a* to tell the device what modes you want the ports to operate in. You use the *bit set/reset control word* format in Figure 9-5*b* when you want to set or reset the output on a pin of port C or when you want to enable the

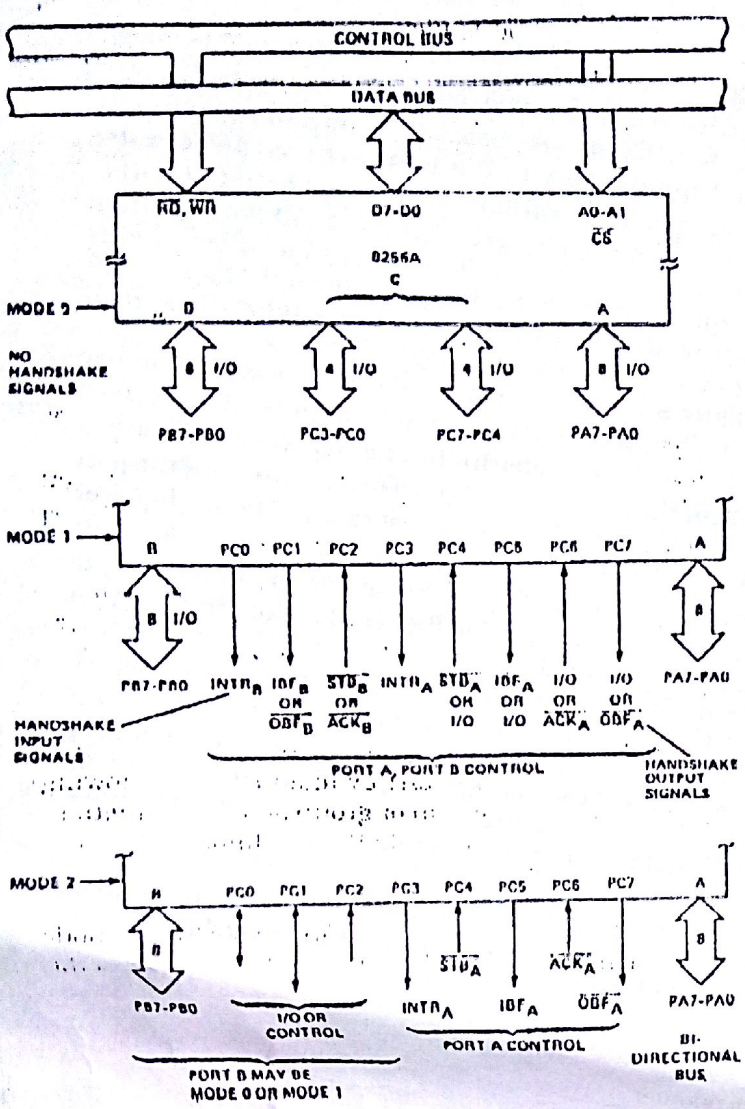

FIGURE 9-4 Summary of 8255A operating modes. *(Intel Corporation)*

interrupt output signals for handshake data transfers. Both control words are sent to the control register address of the 8255A.

As usual, initializing a device such as this consists of working your way through the steps we described in the last chapter. As an example for this device, suppose that you want to initialize the 8255A (A40) in Figure 7-8 as follows:

Port B as mode 1 input

Port A as mode 0 output

Port C upper as inputs

Port C bit 3 as output

As we said previously, the base address for the A40 8255A is FFF8H, and the control register address is FFFEH. The next step is to make up the control word by figuring out what to put in each of the little boxes, one bit at a time. Figure 9-6a, p. 250, shows the control word which will program the 8255A as desired for this example. The figure also shows how you should document

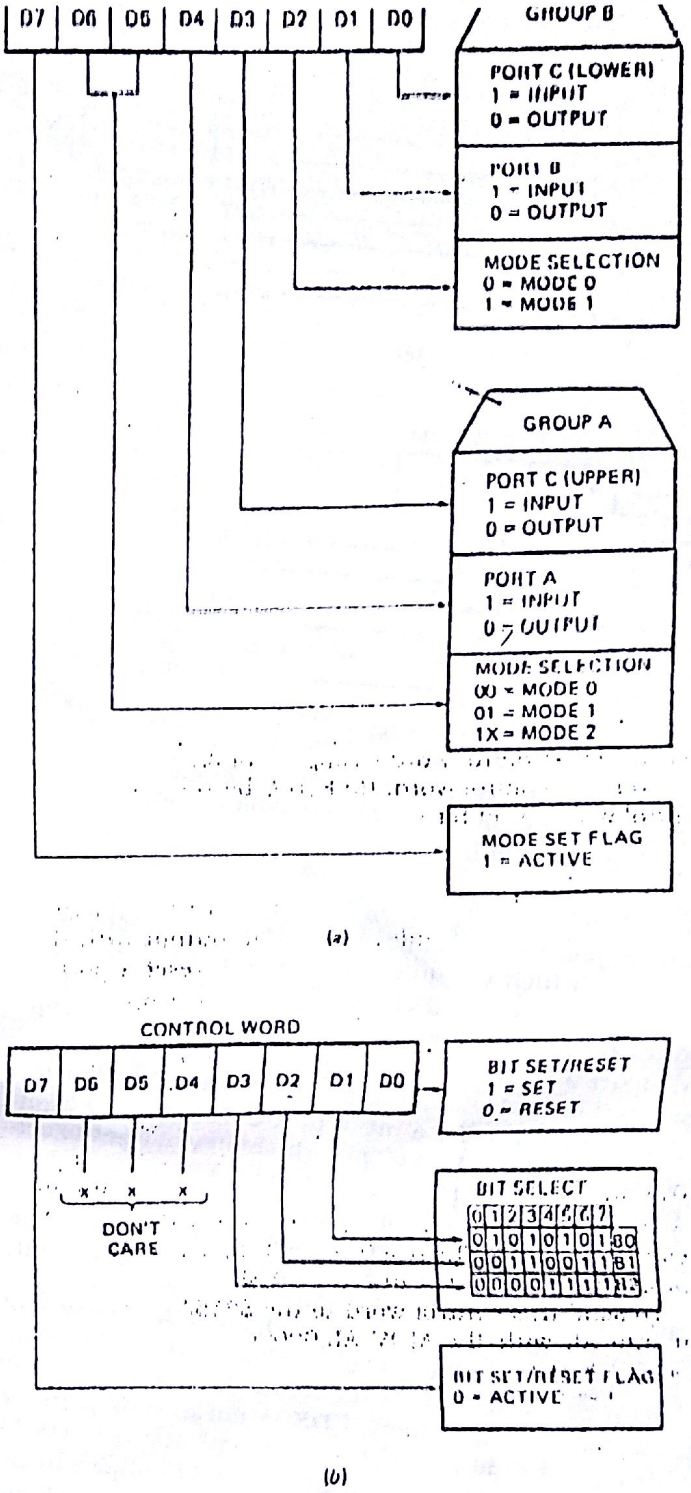

FIGURE 9-5 8255A control word formats. (a) Mode-set control word. (b) Port C bit set/reset control word.

any control words you make up for use in your programs. Using Figure 9-5a, work your way through this word to make sure you see why each bit has the value it does.

To send the control word, you load the control word in AL with a MOV AL,10001110B instruction, point DX at the port address with the MOV DX,0FFFEH instruction, and send the control word to the 8255A control register with the OUT DX,AL instruction.

As an example of how to use the bit set/reset control

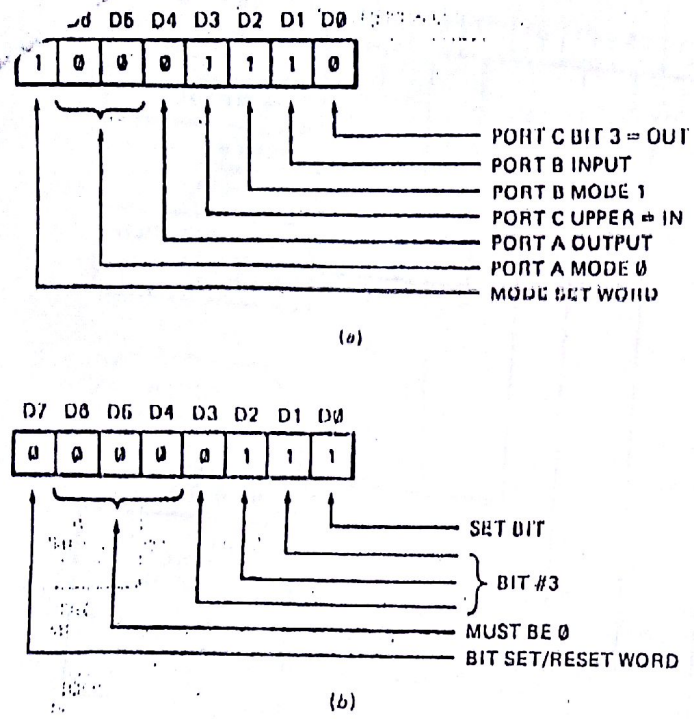

FIGURE 9-6 Control word examples for 8255A. (*a*) Mode-set control word. (*b*) Port C bit set/reset control word to set bit 3.

word, suppose that you want to output a 1 to (set) bit 3 of port C, which was initialized as an output with the mode set control word above. To set or reset a port C output pin, you use the bit set/reset control word shown in Figure 9-5*b*. Make bit D7 a 0 to identify this as a bit set/reset control word, and put a 1 in bit D0 to specify that you want to set a bit of port C. Bits D3, D2, and D1 are used to tell the 8255A which bit you want to act on. For this example you want to set bit 3, so you put 011 in these 3 bits. For simplicity and compatibility with future products, make the other 3 bits of the control word 0's. The result, 00000111B, is shown with proper documentation in Figure 9-6*b*.

To send this control word to the 8255A, simply load it into AL with the MOV AL,00000111B instruction, point DX at the control register address with the MOV DX,0FFFEH instruction if DX is not already pointing there, and send the control word with the OUT DX,AL instruction. As part of the application examples in the following sections, we will show you how you know which bit in port C to set to enable the interrupt output signal for handshake data transfer.

8255A Handshake Application Examples

INTERFACING TO A MICROCOMPUTER-CONTROLLED LATHE

All the machines in the machine shop of our computer-controlled electronics factory operate under microcomputer control. One example of these machines is a lathe which makes bolts from long rods of stainless steel. The cutting instructions for each type of bolt that we need to make are stored on a 3/4-in.-wide teletype-like metal tape. Each instruction is represented by a series of holes in the tape. A tape reader pulls the tape through an optical or mechanical sensor to detect the hole patterns and converts these to an 8-bit parallel code. The microcomputer reads the instruction codes from the tape reader on a handshake basis and sends the appropriate control instructions to the lathe. The microcomputer must also monitor various conditions around the lathe. It must, for example, make sure the lathe has cutting lubricant oil, is not out of material to work on, and is not jammed up in some way. Machines that operate in this way are often referred to as *computer numerical control*, or *CNC, machines*.

Figure 9-7 shows in diagram form how you might use an 8255A to interface a microcomputer to the tape reader and lathe. Later in the chapter, we will show you some of the actual circuitry needed to interface the port pins of the 8255A to the sensors and the high-power motors of the lathe. For now, we want to talk about initializing the 8255A for this application and analyze the timing waveforms for the handshake input of data from the tape reader.

Your first task is to make up the control word which will initialize the 8255A in the correct modes for this application. To do this, start by making a list showing how you want each port pin or group of pins to function. Then put in the control word bits that implement those pin functions. For our example here,

Port A needs to be initialized for handshake input (mode 1) because instruction codes have to be read in from the tape reader on a handshake basis.

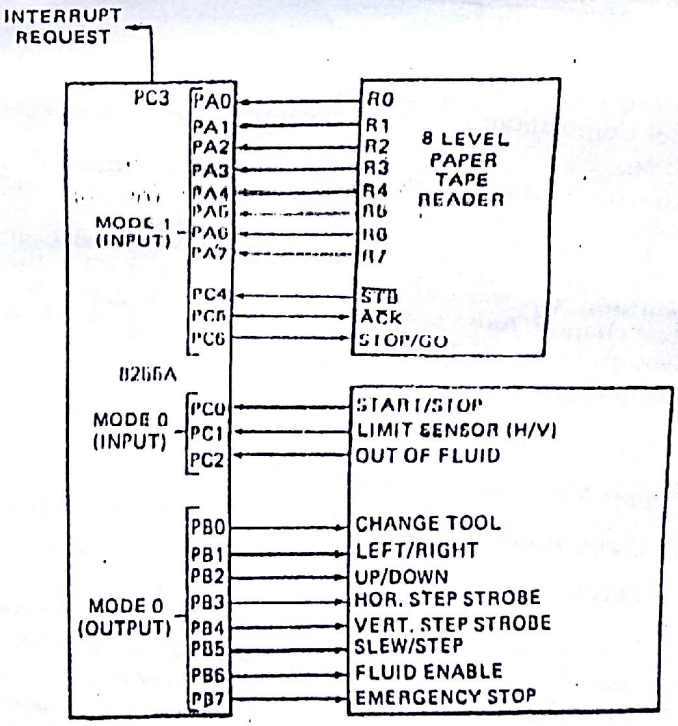

FIGURE 9-7 Interfacing a microprocessor to a tape reader and lathe.