A microprocessor is great at solving problems, but if it can’t communicate with the outside world, it is of little worth. This chapter outlines some of the basic methods of communications, both serial and parallel, between humans or machines and the microprocessor.

In this chapter, we first introduce the basic I/O interface and discuss decoding for I/O devices. Then, we provide detail on parallel and serial interfacing, both of which have a variety of applications. To study applications, we connect analog-to-digital and digital-to-analog converters, as well as both DC and stepper motors to the microprocessor.

Upon completion of this chapter, you will be able to:

1. Explain the operation of the basic input and output interfaces.
2. Decode an 8-, 16-, and 32-bit I/O device so that they can be used at any I/O port address.
3. Define handshaking and explain how to use it with I/O devices.
4. Interface and program the 82C55 programmable parallel interface.
5. Interface LCD displays, LED displays, keyboards, ADC, DAC, and various other devices to the 82C55.
6. Interface and program the 16550 serial communications interface adapter.
7. Interface and program the 8254 programmable interval timer.
8. Interface an analog-to-digital converter and a digital-to-analog converter to the microprocessor.
9. Interface both DC and stepper motors to the microprocessor.

In this section of the text I/O instructions (IN, INS, OUT, and OUTS) are explained and used in example applications. Also explained here is the concept of isolated (sometimes called direct or I/O mapped I/O) and memory-mapped I/O, the basic input and output interfaces, and handshaking. A working knowledge of these topics makes it easier to understand the connection and
The I/O Instructions

The instruction set contains one type of instruction that transfers information to an I/O device (OUT) and another to read information from an I/O device (IN). Instructions (INS and OUTS, found on all versions except the 8086/8088) are also provided to transfer strings of data between the memory and an I/O device. Table 11–1 lists all versions of each instruction found in the microprocessor’s instruction set.

Instructions that transfer data between an I/O device and the microprocessor’s accumulator (AL, AX, or EAX) are called IN and OUT. The I/O address is stored in register DX as a 16-bit I/O address or in the byte (p8) immediately following the opcode as an 8-bit I/O address. Intel calls the 8-bit form (p8) a fixed address because it is stored with the instruction, usually in a ROM. The 16-bit I/O address in DX is called a variable address because it is stored in a DX, and then used to address the I/O device. Other instructions that use DX to address I/O are the INS and OUTS instructions. I/O ports are 8 bits in width so whenever a 16-bit port is accessed two consecutive 8-bit ports are actually addressed. A 32-bit I/O port is actually four 8-bit ports. For example, port 100H is accessed as a word, then 100H and 101H are actually accessed. Port 100H contains the least significant part of the data and port 101H the most significant part.

Table 11–1 Input/Output instructions.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Data Width</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN AL, p8</td>
<td>8</td>
<td>A byte is input into AL from port p8</td>
</tr>
<tr>
<td>IN AX, p8</td>
<td>16</td>
<td>A word is input into AX from port p8</td>
</tr>
<tr>
<td>IN EAX, p8</td>
<td>32</td>
<td>A doubleword is input into EAX from port p8</td>
</tr>
<tr>
<td>IN AL, DX</td>
<td>8</td>
<td>A byte is input into AL from the port addressed by DX</td>
</tr>
<tr>
<td>IN AX, DX</td>
<td>16</td>
<td>A word is input into AX from the port addressed by DX</td>
</tr>
<tr>
<td>IN EAX, DX</td>
<td>32</td>
<td>A doubleword is input into EAX from the port addressed by DX</td>
</tr>
<tr>
<td>INSB</td>
<td>8</td>
<td>A byte is input from the port addressed by DI and stored into the extra segment memory location addressed by DI, then DI = DI ± 1</td>
</tr>
<tr>
<td>INSW</td>
<td>16</td>
<td>A word is input from the port addressed by DI and stored into the extra segment memory location addressed by DI, then DI = DI ± 2</td>
</tr>
<tr>
<td>INSD</td>
<td>32</td>
<td>A doubleword is input from the port addressed by DI and stored into the extra segment memory location addressed by DI, then DI = DI ± 4</td>
</tr>
<tr>
<td>OUT p8, AL</td>
<td>8</td>
<td>A byte is output from AL into port p8</td>
</tr>
<tr>
<td>OUT p8, AX</td>
<td>16</td>
<td>A word is output from AL into port p8</td>
</tr>
<tr>
<td>OUT p8, EAX</td>
<td>32</td>
<td>A doubleword is output from EAX into port p8</td>
</tr>
<tr>
<td>OUT DX, AL</td>
<td>8</td>
<td>A byte is output from AL into the port addressed by DX</td>
</tr>
<tr>
<td>OUT DX, AX</td>
<td>16</td>
<td>A word is output from AX into the port addressed by DX</td>
</tr>
<tr>
<td>OUT DX, EAX</td>
<td>32</td>
<td>A doubleword is output from EAX into the port addressed by DX</td>
</tr>
<tr>
<td>OUTSB</td>
<td>8</td>
<td>A byte is output from the data segment memory location addressed by SI into the port addressed by DX, then SI = SI ± 1</td>
</tr>
<tr>
<td>OUTSW</td>
<td>16</td>
<td>A word is output from the data segment memory location addressed by SI into the port addressed by DX, then SI = SI ± 2</td>
</tr>
<tr>
<td>OUTSD</td>
<td>32</td>
<td>A doubleword is output from the data segment memory location addressed by SI into the port addressed by DX, then SI = SI ± 4</td>
</tr>
</tbody>
</table>
Whenever data are transferred by using the IN or OUT instructions, the I/O address, often called a **port number** (or simply port), appears on the address bus. The external I/O interface decodes the port number in the same manner that it decodes a memory address. The 8-bit fixed port number (p8) appears on address bus connections \( A_7 - A_0 \) with bits \( A_{15} - A_8 \) equal to 00000000_2. The address connections above \( A_{15} \) are undefined for an I/O instruction. The 16-bit variable port number (DX) appears on address connections \( A_{15} - A_0 \). This means that the first 256 I/O port addresses (00H–FFH) are accessed by both the fixed and variable I/O instructions, but any I/O address from 0100H to FFFFH is only accessed by the variable I/O address. In many dedicated systems, only the rightmost 8 bits of the address are decoded, thus reducing the amount of circuitry required for decoding. In a PC computer, all 16 address bus bits are decoded with locations 0000H–03FFH, which are the I/O addresses used for I/O inside the PC on the ISA (industry standard architecture) bus.

The INS and OUTS instructions address an I/O device by using the DX register, but do not transfer data between the accumulator and the I/O device as do the IN and OUT instructions. Instead, these instructions transfer data between memory and the I/O device. The memory address is located by ES:DI for the INS instruction and by DS:SI for the OUTS instruction. As with other string instructions, the contents of the pointers are incremented or decremented, as dictated by the state of the direction flag (DF). Both INS and OUTS can be prefixed with the REP prefix, allowing more than one byte, word, or doubleword to be transferred between I/O and memory.

The Pentium 4 and Core2 operating in the 64-bit mode have the same I/O instructions. There are no 64-bit I/O instructions in the 64-bit mode. The main reason is that most I/O is still 8 bits and likely will remain so for an indefinite time.

**Isolated and Memory-Mapped I/O**

There are two different methods of interfacing I/O to the microprocessor: **isolated I/O** and **memory-mapped I/O**. In the isolated I/O scheme, the IN, INS, OUT, and OUTS instructions transfer data between the microprocessor’s accumulator or memory and the I/O device. In the memory-mapped I/O scheme, any instruction that references memory can accomplish the transfer. Both isolated and memory-mapped I/O are in use, so both are discussed in this text. The PC does not use memory-mapped I/O.

**Isolated I/O.** The most common I/O transfer technique used in the Intel microprocessor-based system is isolated I/O. The term isolated describes how the I/O locations are isolated from the memory system in a separate I/O address space. (Figure 11–1 illustrates both the isolated and memory-mapped address spaces for any Intel 80X86 or Pentium–Core2 microprocessor.) The addresses for isolated I/O devices, called ports, are separate from the memory. Because the ports are separate, the user can expand the memory to its full size without using any of memory space for I/O devices. A disadvantage of isolated I/O is that the data transferred between I/O and the microprocessor must be accessed by the IN, INS, OUT, and OUTS instructions. Separate control signals for the I/O space are developed (using M/IO and W/R), which indicate an I/O read (IORC) or an I/O write (IOWC) operation. These signals indicate that an I/O port address, which appears on the address bus, is used to select the I/O device. In the personal computer, isolated I/O ports are used for controlling peripheral devices. An 8-bit port address is used to access devices located on the system board, such as the timer and keyboard interface, while a 16-bit port is used to access serial and parallel ports as well as video and disk drive systems.

**Memory-Mapped I/O.** Unlike isolated I/O, memory-mapped I/O does not use the IN, INS, OUT, or OUTS instructions. Instead, it uses any instruction that transfers data between the microprocessor and memory. A memory-mapped I/O device is treated as a memory location in the memory map. The main advantage of memory-mapped I/O is that any memory transfer instruction can be used to access the I/O device. The main disadvantage is that a portion of the memory system is used as the I/O map. This reduces the amount of memory available to applications. Another advantage is that the IORC and IOWC signals have no function in a memory-mapped I/O system and may reduce the amount of circuitry required for decoding.
FIGURE 11–1  The memory and I/O maps for the 8086/8088 microprocessors. (a) Isolated I/O. (b) Memory-mapped I/O.

Personal Computer I/O Map

The personal computer uses part of the I/O map for dedicated functions. Figure 11–2 shows the I/O map for the PC. Note that I/O space between ports 0000H and 03FFH is normally reserved for the computer system and the ISA bus. The I/O ports located at 0400H–FFFFH are generally available for user applications, main-board functions, and the PCI bus. Note that the 80287 arithmetic coprocessor uses I/O address 00F8H–00FFH for communications. For this reason, Intel reserves I/O ports 00F0H–00FFH. The 80386–Core2 use I/O ports 800000F8–800000FFH for communications to their coprocessors. The I/O ports located between 0000H and 00FFH are accessed via the fixed port I/O instructions; the ports located above 00FFH are accessed via the variable I/O port instructions.

Basic Input and Output Interfaces

The basic input device is a set of three-state buffers. The basic output device is a set of data latches. The term IN refers to moving data from the I/O device into the microprocessor and the term OUT refers to moving data out of the microprocessor to the I/O device.

The Basic Input Interface. Three-state buffers are used to construct the 8-bit input port depicted in Figure 11–3. The external TTL data (simple toggle switches in this example) are connected to the
inputs of the buffers. The outputs of the buffers connect to the data bus. The exact data bus connections depend on the version of the microprocessor. For example, the 8088 has data bus connections D7–D0, the 80386/80486 has connections D31–D0, and the Pentium–Core2 have connections D63–D0.

The circuit of Figure 11–3 allows the microprocessor to read the contents of the eight switches that connect to any 8-bit section of the data bus when the select signal \( \text{SEL} \) becomes a logic 0. Thus, whenever the IN instruction executes, the contents of the switches are copied into the AL register.

When the microprocessor executes an IN instruction, the I/O port address is decoded to generate the logic 0 on \( \text{SEL} \). A 0 placed on the output control inputs (\( \text{TG} \) and \( \text{DG} \)) of the 74ALS244 buffer causes the data input connections (A) to be connected to the data output (Y) connections. If a logic 1 is placed on the output control inputs of the 74ALS244 buffer, the device enters the three-state high-impedance mode that effectively disconnects the switches from the data bus.

This basic input circuit is not optional and must appear any time that input data are interfaced to the microprocessor. Sometimes it appears as a discrete part of the circuit, as shown in Figure 11–3; many times it is built into a programmable I/O device.

Sixteen- or 32-bit data can also be interfaced to various versions of the microprocessor, but this is not nearly as common as using 8-bit data. To interface 16 bits of data, the circuit in
The basic input interface illustrating the connection of eight switches. Note that the 74ALS244 is a three-state buffer that controls the application of the switch data to the data bus.

Figure 11–3 is doubled to include two 74ALS244 buffers that connect 16 bits of input data to the 16-bit data bus. To interface 32 bits of data, the circuit is expanded by a factor of 4.

**The Basic Output Interface.** The basic output interface receives data from the microprocessor and usually must hold it for some external device. Its latches or flip-flops, like the buffers found in the input device, are often built into the I/O device.

Figure 11–4 shows how eight simple light-emitting diodes (LEDs) connect to the microprocessor through a set of eight data latches. The latch stores the number output by the microprocessor from the data bus, so that the LEDs can be lit with any 8-bit binary number. Latches are needed to hold the data because when the microprocessor executes an OUT instruction, the data are only present on the data bus for less than 1.0 µs. Without a latch, the viewer would never see the LEDs illuminate.

When the OUT instruction executes, the data from AL, AX, or EAX are transferred to the latch via the data bus. Here, the D inputs of a 74ALS374 octal latch are connected to the data bus to capture the output data, and the Q outputs of the latch are attached to the LEDs. Each time that the OUT instruction executes, the signal to the latch activates, capturing the data output to the latch from any 8-bit section of the data bus. The data are held until the next OUT instruction executes. Thus, whenever the output instruction is executed in this circuit, the data from the AL register appear on the LEDs.

**Handshaking**

Many I/O devices accept or release information at a much slower rate than the microprocessor. Another method of I/O control, called **handshaking** or **polling**, synchronizes the I/O device with the microprocessor. An example of a device that requires handshaking is a parallel printer that prints a few hundred characters per second (CPS). It is obvious that the microprocessor can send more than a few hundred CPS to the printer, so a way to slow the microprocessor down to match speeds with the printer must be developed.

Figure 11–5 illustrates the typical input and output connections found on a printer. Here, data are transferred through a series of data connections (D7–D0). BUSY indicates that the printer is busy. STB is a clock pulse used to send data to the printer for printing.
The ASCII data to be printed by the printer are placed on $D_7–D_0$, and a pulse is then applied to the STB connection. The strobe signal sends or clocks the data into the printer so that they can be printed. As soon as the printer receives the data, it places a logic 1 on the BUSY pin, indicating that the printer is busy printing data. The microprocessor software polls or tests the BUSY pin to decide whether the printer is busy. If the printer is busy, the microprocessor waits; if it is not busy, the microprocessor sends the next ASCII character to the printer. This process of interrogating the printer, or any asynchronous device like a printer, is called handshaking or polling. Example 11–1 illustrates a simple procedure that tests the printer BUSY flag and then sends data to the printer if it is not busy. Here, the PRINT procedure prints the ASCII-coded contents of BL only if the BUSY flag is a logic 0, indicating that the printer is not busy. This procedure is called each time a character is to be printed.

**EXAMPLE 11–1**

;An assembly language procedure that prints the ASCII contents of BL.

```
PRINT PROC NEAR

.REPEAT ;test the busy flag
    IN AL,BUSY ;test the busy flag
    TEST AL,BUSY_BIT
    .UNTIL ZERO
    MOV AL,BL ;position data in AL
    OUT PRINTER,AL ;print data

PRINT ENDP
```

**Notes about Interfacing Circuitry**

A part of interfacing requires some knowledge about electronics. This portion of the introduction to interfacing examines some of the many facets of electronic interfacing. Before a circuit or
device can be interfaced to the microprocessor, the terminal characteristics of the microprocessor and its associated interfacing components must be known. (This subject was introduced at the start of Chapter 9.)

**Input Devices.** Input devices are already TTL and compatible, and therefore can be connected to the microprocessor and its interfacing components, or they are switch-based. Most switch-based devices are either open or connected. These are not TTL levels—TTL levels are a logic 0 (0.0 V–0.8 V) or a logic 1 (2.0 V–5.0 V).

For a switch-based device to be used as a TTL-compatible input device, some conditioning must be applied. Figure 11–6 shows a simple toggle switch that is properly connected to function as an input device. Notice that a pull-up resistor is used to ensure that when the switch is open, the output signal is a logic 1; when the switch is closed, it connects to ground, producing a valid logic 0 level. The value of the pull-up resistor is not critical—it merely assures that the signal is

---

**TABLE 11–4**  The DB25 connector found on computers and the Centronics 36-pin connector found on printers for the Centronics parallel printer interface.

<table>
<thead>
<tr>
<th>DB25 Pin number</th>
<th>CENT36 Pin number</th>
<th>Function</th>
<th>DB25 Pin number</th>
<th>CENT36 Pin number</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Data Strobe</td>
<td>12</td>
<td>12</td>
<td>Paper empty</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>Data 0 (D0)</td>
<td>13</td>
<td>13</td>
<td>Select</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>Data 1 (D1)</td>
<td>14</td>
<td>14</td>
<td>Ack</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>Data 2 (D2)</td>
<td>15</td>
<td>32</td>
<td>Error</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>Data 3 (D3)</td>
<td>16</td>
<td>—</td>
<td>RESET</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>Data 4 (D4)</td>
<td>17</td>
<td>31</td>
<td>Select in</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>Data 5 (D5)</td>
<td>18–25</td>
<td>19–30</td>
<td>Ground</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>Data 6 (D6)</td>
<td>—</td>
<td>17</td>
<td>Frame ground</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>Data 7 (D7)</td>
<td>—</td>
<td>16</td>
<td>Ground</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>Ack</td>
<td>—</td>
<td>33</td>
<td>Ground</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>Busy</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
FIGURE 11-6 A single-pole, single-throw switch interfaced as a TTL device.

FIGURE 11-7 Debouncing switch contacts: (a) conventional debouncing and (b) practical debouncing.

at a logic 1 level. A standard range of values for pull-up resistors is usually anywhere between 1K Ω and 10K Ω.

Mechanical switch contacts physically bounce when they are closed, which can create a problem if a switch is used as a clocking signal for a digital circuit. To prevent problems with bounces, one of the two circuits depicted in Figure 11–7 can be constructed. The first circuit (a) is a classical textbook bounce eliminator; the second (b) is a more practical version of the same circuit. Because the first version costs more money to construct, in practice, the second would be used because it requires no pull-up resistors and only two inverters instead of two NAND gates.

You may notice that both circuits in Figure 11–7 are asynchronous flip-flops. The circuit of (b) functions in the following manner: Suppose that the switch is currently at position Q. If it is moved toward Q but does not yet touch Q, the Q output of the circuit is a logic 0. The logic 0 state is remembered by the inverters. The output of inverter B connects to the input of inverter A. Because the output of inverter B is a logic 0, the output of inverter A is a logic 1. The logic 1 output of inverter A maintains the logic 0 output of inverter B. The flip-flop remains in this state until the moving switch-contact first touches the Q connection. As soon as the Q input from the switch becomes a logic 0, it changes the state of the flip-flop. If the contact bounces back away from the Q input, the flip-flop remembers and no change occurs, thus eliminating any bounce.

Output Devices. Output devices are far more diverse than input devices, but many are interfaced in a uniform manner. Before any output device can be interfaced, we must understand what the voltages and currents are from the microprocessor or a TTL interface component. The voltages are TTL-compatible from the microprocessor or a TTL interface component. (Logic 0 = 0.0 V to 0.4 V; logic 1 = 2.4 V to 5.0 V.) The currents for a microprocessor and many microprocessor-interfacing components are less than for standard TTL components. (Logic 0 = 0.0 to 2.0 mA; logic 1 = 0.0 to 400 µA.)

Once the output currents are known, a device can now be interfaced to one of the outputs. Figure 11–8 shows how to interface a simple LED to a microprocessor peripheral pin. Notice that a transistor driver is used in Figure 11–8(a) and a TTL inverter is used in Figure 11–8(b). The TTL inverter (standard version) provides up to 16 mA of current at a logic 0 level, which is more than enough to drive a standard LED. A standard LED requires 10 mA of forward bias current to light. In both circuits, we assume that the voltage drop across the LED is about 2.0 V.
FIGURE 11–8 Interfacing an LED: (a) using a transistor and (b) using an inverter.

![Diagram of LED interfacing](image)

The data sheet for an LED states that the nominal drop is 1.65 V, but it is known from experience that the drop is anywhere between 1.5 V and 2.0 V. This means that the value of the current-limiting resistor is 3.0 V \div 10 \text{ mA} = 300 \Omega. Because 300 \Omega is not a standard resistor value (the lowest cost), a 330 \Omega resistor is chosen for this interface.

In the circuit of Figure 11–8(a), we elected to use a switching transistor in place of the TTL buffer. The 2N2222 is a good low-cost, general-purpose switching transistor that has a minimum gain of 100. In this circuit, the collector current is 10 mA, so the base current will be 1/100 of the collector current of 0.1 mA. To determine the value of the base current–limiting resistor, use the 0.1 mA base current and a voltage drop of 1.7 V across the base current–limiting resistor. The TTL input signal has a minimum value of 2.4 V and the drop across the emitter-base junction is 0.7 V. The difference is 1.7 V, which is the voltage drop across the resistor. The value of the resistor is 1.7 V \div 0.1 \text{ mA} = 17K \Omega. Because 17K \Omega is not a standard value, an 18K \Omega resistor is chosen.

Suppose that we need to interface a 12 V DC motor to the microprocessor and the motor current is 1A. Obviously, we cannot use a TTL inverter for two reasons: The 12 V signal would burn out the inverter and the amount of current far exceeds the 16 mA maximum current from the inverter. We cannot use a 2N2222 transistor either, because the maximum amount of current is 250 mA to 500 mA, depending on the package style chosen. The solution is to use a Darlington-pair, such as a TIP120. The TIP120 costs 25¢ and with the proper heat sink can handle 4A of current.

Figure 11–9 illustrates a motor connected to the Darlington-pair. The Darlington-pair has a minimum current gain of 7000 and a maximum current of 4A. The value of the bias resistor is calculated exactly the same as the one used in the LED driver. The current through the resistor is 1.0 A \div 7000, or about 0.143 mA. The voltage drop across the resistor is 0.9 V because of the two diode drops (base/emitter junctions) instead of one. The value of the bias resistor is 0.9 V \div 0.143 mA = 6.29 K \Omega. The standard value of 6.2 K \Omega is used in the circuit. The Darlington-pair must use a heat sink because of the amount of current going through it. Typically any device that passes more than \frac{1}{2} A of current needs a heat sink. The diode must also be present to prevent the Darlington-pair from being destroyed by the inductive kickback from the motor. This circuit is also used to interface mechanical relays or just about any device that requires a large amount of current or a change in voltage.

FIGURE 11–9 A DC motor interfaced to a system by using a Darlington-pair.

![Diagram of DC motor interfacing](image)
11–2 I/O PORT ADDRESS DECODING

I/O port address decoding is very similar to memory address decoding, especially for memory-mapped I/O devices. In fact, we do not discuss memory-mapped I/O decoding because it is treated the same as memory (except that the \texttt{IORC} and \texttt{IOWC} are not used because there is no \texttt{IN} or \texttt{OUT} instruction). The decision to use memory-mapped I/O is often determined by the size of the memory system and the placement of the I/O devices in the system.

The main difference between memory decoding and isolated I/O decoding is the number of address pins connected to the decoder. We decode $A_{31}–A_0$, $A_{23}–A_0$, or $A_{19}–A_0$ for memory, and $A_{15}–A_0$ for isolated I/O. Sometimes, if the I/O devices use only fixed I/O addressing, we decode only $A_{7}–A_0$. In the personal computer system, we always decode all 16 bits of the I/O port address. Another difference with isolated I/O is that \texttt{IORC} and \texttt{IOWC} activate I/O devices for a read or write operation. On earlier versions of the microprocessor, \texttt{IO/M} = 1 and \texttt{RD} or \texttt{WR} are used to activate I/O devices. On the newest versions of the microprocessor, the $M/I/O = 0$ and \texttt{WR} are combined and used to activate I/O devices.

Decoding 8-Bit I/O Port Addresses

As mentioned, the fixed I/O instruction uses an 8-bit I/O port address that appears on $A_{15}–A_0$ as 0000H–00FFH. If a system will never contain more than 256 I/O devices, we often decode only address connections $A_{7}–A_0$ for an 8-bit I/O port address. Thus, we ignore address connection $A_{15}–A_8$. Embedded systems often use 8-bit port addresses. Please note that the DX register can also address I/O ports 00H–FFH. If the address is decoded as an 8-bit address, we can never include I/O devices that use a 16-bit address. The personal computer never uses or decodes an 8-bit address.

Figure 11–10 illustrates a 74ALS138 decoder that decodes 8-bit I/O ports F0H through F7H. (We assume that this system will only use I/O ports 00H–FFH for this decoder example.) This decoder is identical to a memory address decoder except we only connect address bits $A_{7}–A_0$ to the inputs of the decoder. Figure 11–11 shows the PLD version, using a GAL22V10 (a low-cost device) for this decoder. The PLD is a better decoder circuit because the number of integrated circuits has been reduced to one device. The VHDL program for the PLD appears in Example 11–2.

EXAMPLE 11–2

-- VHDL code for the decoder of Figure 11–11

library ieee;
use ieee.std_logic_1164.all;

entity DECODER_11_11 is

port (  
  A7, A6, A5, A4, A3, A2, A1, A0: in STD_LOGIC;
  D0, D1, D2, D3, D4, D5, D6, D7: out STD_LOGIC
);
end;

architecture V1 of DECODER_11_11 is

begin
  D0 <= not( A7 and A6 and A5 and A4 and not A3 and not A2 and not A1 and not A0 );
  D1 <= not( A7 and A6 and A5 and A4 and not A3 and not A2 and not A1 and A0 );
  D2 <= not( A7 and A6 and A5 and A4 and not A3 and not A2 and A1 and not A0 );

end;
Decoding 16-Bit I/O Port Addresses

Personal computer systems typically use 16-bit I/O addresses. It is relatively rare to find 16-bit port addresses in embedded systems. The main difference between decoding an 8-bit I/O address and a 16-bit I/O address is that eight additional address lines \( A_{15} - A_8 \) must be decoded. Figure 11–12 illustrates a circuit that contains a PLD and a 4-input NAND gate used to decode I/O ports \( \text{EFF8H–EFFFH} \).

The NAND gate decodes part of the address \( A_{15} - A_8 \) because the PLD does not have enough address inputs. The output of the NAND gate connects to the Z input of the PLD and is decoded as a part of the I/O port address. The PLD generates address strobes for I/O ports \( \text{EFF8H–EFFFH} \). The program for the PLD is listed in Example 11–3.

EXAMPLE 11–3

-- VHDL code for the decoder of Figure 11–12

library ieee;
use ieee.std_logic_1164.all;
FIGURE 11–12 A PLD that decodes 16-bit I/O ports EFF8H through EFFFFH.

entity DECODER_11_12 is
  port (  
    Z, A12, A10, A9, A8, A7, A6, A5, A4, A3, A2, A1, A0: in STD_LOGIC;  
    D0, D1, D2, D3, D4, D5, D6, D7: out STD_LOGIC  
  );
end;
architecture V1 of DECODER_11_12 is
begin
  D0 <= not ( not Z and not A12 and A10 and A9 and A8 and A7 and A6 and A5  
    and A4 and A3 and not A2 and not A1 and not A0 );
  D1 <= not ( not Z and not A12 and A10 and A9 and A8 and A7 and A6 and A5  
    and A4 and A3 and not A2 and not A1 and A0 );
  D2 <= not ( not Z and not A12 and A10 and A9 and A8 and A7 and A6 and A5  
    and A4 and A3 and not A2 and A1 and not A0 );
  D3 <= not ( not Z and not A12 and A10 and A9 and A8 and A7 and A6 and A5  
    and A4 and A3 and not A2 and A1 and A0 );
  D4 <= not ( not Z and not A12 and A10 and A9 and A8 and A7 and A6 and A5  
    and A4 and A3 and A2 and not A1 and not A0 );
  D5 <= not ( not Z and not A12 and A10 and A9 and A8 and A7 and A6 and A5  
    and A4 and A3 and A2 and not A1 and A0 );
  D6 <= not ( not Z and not A12 and A10 and A9 and A8 and A7 and A6 and A5  
    and A4 and A3 and A2 and A1 and not A0 );
  D7 <= not ( not Z and not A12 and A10 and A9 and A8 and A7 and A6 and A5  
    and A4 and A3 and A2 and A1 and A0 );
end V1;

8- and 16-Bit Wide I/O Ports

Now that I/O port addresses are understood and we learned that an I/O port address is probably simpler to decode than a memory address (because of the number of bits), interfacing between the microprocessor and 8- or 16-bit-wide I/O devices is explained. Data transferred to an 8-bit I/O device exist in one of the I/O banks in a 16-bit microprocessor such as the 80386SX. There are 64K different 8-bit ports, but only 32K different 8-bit ports because a 16-bit port uses two 8-bit ports. The I/O system on such a microprocessor contains two 8-bit memory banks, just as memory does. This is illustrated in Figure 11–13, which shows the separate I/O banks for a 16-bit system such as the 80386SX.
FIGURE 11–13  The I/O banks found in the 8086, 80186, 80286, and 80386SX.

Because two I/O banks exist, any 8-bit I/O write requires a separate write strobe to function correctly. I/O reads do not require separate read strobes. As with memory, the microprocessor reads only the byte it expects and ignores the other byte. The only time that a read can cause problems is when the I/O device responds incorrectly to a read operation. In the case of an I/O device that responds to a read from the wrong bank, we may need to include separate read signals. This is discussed later in this chapter.

Figure 11–14 illustrates a system that contains two different 8-bit output devices, located at 8-bit I/O address 40H and 41H. Because these are 8-bit devices and because they appear in different I/O banks, separate I/O write signals are generated to clock a pair of latches that capture port data. Note that all I/O ports use 8-bit addresses. Thus, ports 40H and 41H can each be addressed as separate 8-bit ports, or together as one 16-bit port. The program for the PLD decoder used in Figure 11–14 is illustrated in Example 11–4.

EXAMPLE 11–4

-- VHDL code for the decoder of Figure 11–14
library ieee;
use ieee.std_logic_1164.all;
entity DECODER_11_14 is
port (BHE, IOWC, A7, A6, A5, A4, A3, A2, A1, A0: in STD_LOGIC;
      D0, D1: out STD_LOGIC);
end;
architecture V1 of DECODER_11_14 is
begin
  D0 <= BHE or IOWC or A7 or not A6 or A5 or A4 or A3 or A2 or A1 or A0;
  D1 <= BHE or IOWC or A7 or not A6 or A5 or A4 or A3 or A2 or A1 or not A0;
end V1;
When selecting 16-bit-wide I/O devices, the BLE (A₀) and BHE pins have no function because both I/O banks are selected together. Although 16-bit I/O devices are relatively rare, a few do exist for analog-to-digital and digit-to-analog converters, as well as for some video and disk interfaces.

Figure 11–15 illustrates a 16-bit input device connected to function at 8-bit I/O addresses 64H and 65H. Notice that the PLD decoder does not have a connection for address bits BLE (A₀) and BHE because these signals do not apply to 16-bit-wide I/O devices. The program for the PLD, illustrated in Example 11–5, shows how the enable signals are generated for the three-state buffers (74HCT244) used as input devices.

EXAMPLE 11–5

-- VHDL code for the decoder of Figure 11–15

library ieee;
use ieee.std_logic_1164.all;

entity DECODER_11_15 is
port (IORC, A7, A6, A5, A4, A3, A2, A1: in STD_LOGIC;
      D0: out STD_LOGIC
    );
FIGURE 11–15 A 16-bit-wide port decoded at I/O addresses 64H and 65H.

```vhdl
architecture V1 of DECODER_11_15 is
begin
    D0 <= IORC or A7 or not A6 or not A5 or A4 or A3 or not A2 or A1;
end V1;
```

32-Bit-Wide I/O Ports

Although 32-bit-wide I/O ports are not common, they may eventually become commonplace because of newer buses found in computer systems. The once-promising EISA system bus supports 32-bit I/O as well as the VESA local and current PCI bus, but not many I/O devices are 32 bits in width.

The circuit of Figure 11–16 illustrates a 32-bit input port for the 80386DX through the 80486DX microprocessor. As with prior interfaces, this circuit uses a single PLD to decode the I/O ports and four 74HCT244 buffers to connect the I/O data to the data bus. The I/O ports decoded by this interface are the 8-bit ports 70H–73H, as illustrated by the PLD program in Example 11–6. Again, we only decode an 8-bit I/O port address. When writing software to access this port, it is crucial to use the address 70H for the 32-bit input as in the instruction IN EAX, 70H.

EXAMPLE 11–6

```vhdl
-- VHDL code for the decoder of Figure 11–16
library ieee;
use ieee.std_logic_1164.all;
entity DECODER_11_16 is
port (
FIGURE 11–16 A 32-bit-wide port decoded at 70H through 73H for the 80486DX microprocessor.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity DECODER_11_16 is
  port ( IORC, A7, A6, A5, A4, A3, A2: in STD_LOGIC;
         D0: out STD_LOGIC
  );
end DECODER_11_16;

architecture V1 of DECODER_11_16 is
begin
  D0 <= IORC or A7 or not A6 or not A5 or not A4 or A3 or A2;
end V1;
```
CHAPTER 11

**FIGURE 11–17**  A Pentium 4 interfaced to a 16-bit-wide I/O port at port addresses 2000H and 2001H.

With the Pentium–Core2 microprocessors and their 64-bit data buses, I/O ports appear in various banks, as determined by the I/O port address. For example, 8-bit I/O port 0034H appears in Pentium I/O bank 4, while the 16-bit I/O ports 0034H–0035H appear in Pentium banks 4 and 5. A 32-bit I/O access in the Pentium system can appear in any four consecutive I/O banks. For example, 32-bit I/O ports 0100H–0103H appear in banks 0–3. The I/O address range must begin at a location where the rightmost two bits are zeros. Hence, 0100H–0103H is allowable but 0101H–0104H is not.

How is a 64-bit I/O device interfaced? The widest I/O transfers are 32 bits, and currently there are no 64-bit I/O instructions to support 64-bit transfers. This event is true for the Pentium 4 or Core2 operated in the 64-bit mode.

Suppose that we need to interface a simple 16-bit-wide output port at I/O port address 2000H and 2001H. The rightmost three bits of the lowest port address are 000 for port 2000H. This means that port 2000H is in memory bank 0. Likewise the rightmost three binary bits of I/O port 2001H are 001, which means that port 2001H is in bank 1. An interface is illustrated in Figure 11–17 and the PLD program is listed in Example 11–7.

The control signal M/IO and W/R must be combined to generate an I/O write signal for the latches and both BE0 and BE1 bank enable signals must be used to steer the write signal to the correct latch clock for address 2000H (bank 0) and 2001H (bank 1). The only problem that can arise in interfacing is when the I/O port spans across a 64-bit boundary, for example, a 16-bit-wide port located at 2007H and 2008H. In this case, port 2007H uses bank 7 and 2008H uses bank 0, but the address that is decoded is different for each location. A 0010 0000 0000 0XXX is decoded for 2007H and 0010 0000 0000 1XXX is decoded for 2008H. It is probably best to avoid situations such as this.
EXAMPLE 11−7

-- VHDL code for the decoder of Figure 11−17

library ieee;
use ieee.std_logic_1164.all;

entity DECODER_11_17 is
port (  
    MIO, BE0, BE1, WR, A15, A14, A13, A12, A11, A10, A9, A8, A7, A6, A5, A4,  
    A3: in STD_LOGIC;  
    D0, D1: out STD_LOGIC  
);  
end;

architecture V1 of DECODER_11_17 is
begin
    D0 <= MIO or BE0 or not WR or A15 or A14 or not A13 or A12 or A11 or A10  
        or A9 or A8 or A7 or A6 or A5 or A4 or A3;  
    D1 <= MIO or BE1 or not WR or A15 or A14 or not A13 or A12 or A11 or A10  
        or A9 or A8 or A7 or A6 or A5 or A4 or not A3;  
end V1;

11−3 THE PROGRAMMABLE PERIPHERAL INTERFACE

The 82C55 programmable peripheral interface (PPI) is a very popular, low-cost interfacing component found in many applications. This is true even with all the programmable devices available for simple applications. The PPI, which has 24 pins for I/O that are programmable in groups of 12 pins, has groups that operate in three distinct modes of operation. The 82C55 can interface any TTL-compatible I/O device to the microprocessor. The 82C55 (CMOS version) requires the insertion of wait states if operated with a microprocessor using higher than an 8 MHz clock. It also provides at least 2.5 mA of sink (logic 0) current at each output, with a maximum of 4.0 mA. Because I/O devices are inherently slow, wait states used during I/O transfers do not impact significantly upon the speed of the system. The 82C55 still finds application (compatible for programming, although it may not appear in the system as a discrete 82C55), even in the latest Core2-based computer system. The modern computer uses a few 82C55s located inside the chip set for various features on the personal computer. The 82C55 is used for interface to the keyboard and the parallel printer port in many personal computers, but it is found as a function within a interfacing chip set. The chip set also controls the timer and reads data from the keyboard interface.

A low-cost experimentation board is available that plugs into the parallel port of a PC that allows access to an 8255 located on the board. The 8255 is programmed in either assembly language or Visual C++ through drivers available with the board. Visit the following Internet link for pricing and additional information: http://www.microdigitaled.com/hardware/mdelpc/MDELPPT.htm.

Basic Description of the 82C55

Figure 11−18 illustrates the pin-out diagram of the 82C55 in both the DIP format and the surface mount (flat pack). Its three I/O ports (labeled A, B, and C) are programmed as groups. Group A connections consist of port A (PA7−PA0) and the upper half of port C (PC7−PC4), and group B consists of port B (PB7−PB0) and the lower half of port C (PC3−PC0). The 82C55 is selected by its CS pin for programming and for reading or writing to a port. Register selection is accomplished through the A1 and A0 input pins that select an internal register for programming.
or operation. Table 11–2 shows the I/O port assignments used for programming and access to the I/O ports. In the personal computer a pair of 82C55s, or their equivalents, are decoded at I/O ports 60H–63H and also at ports 378H–37BH.

The 82C55 is a fairly simple device to interface to the microprocessor and program. For the 82C55 to be read or written, the input must be a logic 0 and the correct I/O address must be applied to the A<sub>1</sub> and A<sub>0</sub> pins. The remaining port address pins are don’t car es as far as the 82C55 is concerned, and are externally decoded to select the 82C55.

Figure 11–19 shows an 82C55 connected to the 80386SX so that it functions at 8-bit I/O port addresses C0H (port A), C2H (port B), C4H (port C), and C6H (command register). This interface uses the low bank of the 80386SX I/O map. Notice from this interface that all the 82C55 pins are direct connections to the 80386SX, except for the CS pin. The CS pin is decoded and selected by a 74ALS138 decoder.

The RESET input to the 82C55 initializes the device whenever the microprocessor is reset. A RESET input to the 82C55 causes all ports to be set up as simple input ports using mode 0 operation. Because the port pins are internally programmed as input pins after a RESET, damage is prevented when the power is first applied to the system. After a RESET, no other commands are needed to program the 82C55, as long as it is used as an input device for all three ports. Note

<table>
<thead>
<tr>
<th>TABLE 11–2</th>
<th>I/O port assignments for the 82C55.</th>
</tr>
</thead>
<tbody>
<tr>
<td>A&lt;sub&gt;1&lt;/sub&gt;</td>
<td>A&lt;sub&gt;0&lt;/sub&gt;</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
that an 82C55 is interfaced to the personal computer at port addresses 60H–63H for keyboard control, and also for controlling the speaker, timer, and other internal devices such as memory expansion. It is also used for the parallel printer port at I/O ports 378H–37BH.

**Programming the 82C55**

The 82C55 is programmed through the two internal command registers that are illustrated in Figure 11–20. Notice that bit position 7 selects either command byte A or command byte B. Command byte A programs the function of group A and B, whereas command byte B sets (1) or resets (0) bits of port C only if the 82C55 is programmed in mode 1 or 2.

Group B pins (port B and the lower part of port C) are programmed as either input or output pins. Group B operates in either mode 0 or mode 1. Mode 0 is the basic input/output mode that allows the pins of group B to be programmed as simple input and latched output connections. Mode 1 operation is the strobed operation for group B connections, where data are transferred through port B and handshaking signals are provided by port C.

Group A pins (port A and the upper part of port C) are programmed as either input or output pins. The difference is that group A can operate in modes 0, 1, and 2. Mode 2 operation is a bidirectional mode of operation for port A.

If a 0 is placed in bit position 7 of the command byte, command byte B is selected. This command allows any bit of port C to be set (1) or reset (0), if the 82C55 is operated in either mode 1 or 2. Otherwise, this command byte is not used for programming. The bit set/reset feature is often used in a control system to set or clear a control bit at port C. The bit set/reset function is glitch-free, which means that the other port C pins will not change during the bit set/reset command.
FIGURE 11–20  The command byte of the command register in the 82C55. (a) Programs ports A, B, and C. (b) Sets or resets the bit indicated in the select a bit field.

Mode 0 Operation

Mode 0 operation causes the 82C55 to function either as a buffered input device or as a latched output device. These are the same as the basic input and output circuits discussed in the first section of this chapter.
Figure 11–21 shows the 82C55 connected to a set of eight seven-segment LED displays. These are standard LEDs, but the interface can be modified with a change in resistor values for an organic LED (OLED) display or high-brightness LEDs. In this circuit, both ports A and B are programmed as (mode 0) simple latched output ports. Port A provides the segment data inputs to the display and port B provides a means of selecting one display position at a time for multiplexing the displays. The 82C55 is interfaced to an 8088 microprocessor through a PLD so that it functions at I/O port numbers 0700H–0703H. The program for the PLD is listed in Example 11–8. The PLD decodes the I/O address and develops the write strobe for the WR pin of the 82C55.

**EXAMPLE 11–8**

-- VHDL code for the decoder of Figure 11-21

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity DECODER_11_21 is
  port (
    D0: out STD_LOGIC
  );
end;

architecture V1 of DECODER_11_17 is
begin
  D0 <= not IOM or A15 or A14 or A13 or A12 or A11 or not A10 or not A9 or not A8 or A7 or A6 or A5 or A4 or A3 or A2;
end V1;
```

The resistor values are chosen in Figure 11–21 so that the segment current is 80 mA. This current is required to produce an average current of 10 mA per segment as the displays are multiplexed. A six-digit display uses a segment current of 60 mA for an average of 10 mA per segment. In this type of display system, only one of the eight display positions is on at any given instant. The peak anode current in an eight-digit display is 560 mA (seven segments × 80 mA), but the average anode current is 80 mA. In a six-digit display, the peak current would be 420 mA (seven segments × 60 mA). Whenever displays are multiplexed, we increase the segment current from 10 mA (for a display that uses 10 mA per segment as the nominal current) to a value equal to the number of display positions times 10 mA. This means that a four-digit display uses 40 mA per segment, a five-digit display uses 50 mA, and so on.

In this display, the segment load resistor passes 80 mA of current and has a voltage of approximately 3.0 V across it. The LED (1.65 V nominally) and a few tenths are dropped across the anode switch and the segment switch, hence a voltage of 3.0 V appears across the segment load resistor. The value of the resistor is 3.0 V ÷ 80 mA = 37.5 Ω. The closest standard resistor value of 39 Ω is used in Figure 11–21 for the segment load.

The resistor in series with the base of the segment switch assumes that the minimum gain of the transistor is 100. The base current is therefore 80 mA ÷ 100 = 0.8 mA. The voltage across the base resistor is approximately 3.0 V (the minimum logic 1 voltage level of the 82C55), minus the drop across the emitter-base junction (0.7 V), or 2.3 V. The value of the base resistor is therefore 2.3 V ÷ 0.8mA = 2.875 KΩ. The closest standard resistor value is 2.7 KΩ, but 2.2 KΩ is chosen for this circuit.

The anode switch has a single resistor on its base. The current through the resistor is 560 mA ÷ 100 = 5.6 mA because the minimum gain of the transistor is 100. This exceeds the
FIGURE 11–21 An 8-digit LED display interfaced to the 8088 microprocessor through an 82C55 PIA.
maximum current of 4.0 mA from the 82C55, but this is close enough so that it will work without problems. The maximum current assumes that you are using the port pin as a TIL input to another circuit. If the amount of current were over 8.0–10.0 mA, then appropriate circuitry (in the form of either a Darlington-pair or another transistor switch) would be required. Here, the voltage across the base resistor is 5.0 V, minus the drop across the emitter-base junction (0.7 V), minus the voltage at the port pin (0.4 V), for a logic 0 level. The value of the resistor is $3.9 \, \text{V} \div 5.66 \, \text{mA} = 68.9 \, \Omega$. The closest standard resistor value is 69 $\Omega$, which is chosen for this example.

Before software to operate the display is examined, we must first program the 82C55. This is accomplished with the short sequence of instructions listed in Example 11–9. Here, ports A and B are both programmed as outputs.

**EXAMPLE 11–9**

;programming the 82C55 PIA

```assembly
MOV AL,10000000B ;command
MOV DX,703H ;address port 703H
OUT DX,AL ;send command to port 703H
```

The procedure to multiplex the displays is listed in Example 11–10 in both assembly language and C++ with assembly language. For the display system to function correctly, we must call this procedure often. Notice that the procedure calls another procedure (DELAY) that causes a 1.0 ms time delay. The time delay is not illustrated in this example, but it is used to allow time for each display position to turn on. Manufacturers of LED displays recommend that the display flashes between 100 and 1500 times per second. Using a 1.0 ms time delay, each digit is displayed for 1.0 ms for a total display flash rate of 1000 Hz ÷ 8 or a flash rate of 125 Hz for all eight digits.

**EXAMPLE 11–10**

;An assembly language procedure that multiplexes the 8-digit display.

;This procedure must be called often for the display to appear correctly.

```assembly
DISP PROC NEAR USES AX BX DX SI

PUSHF
MOV BX,8 ;load counter
MOV AH,7FH ;load selection pattern
MOV SI,OFFSET MEM-1 ;address display data
MOV DX,701H ;address Port B

;display all 8 digits
.REPEAT
  MOV AL,AH ;send selection pattern to Port B
  OUT DX,AL
  DEC DX
  MOV AL,[BX+SI] ;send data to Port A
  OUT DX,AL
  CALL DELAY ;wait 1.0 ms
  ROR AH,1 ;adjust selection pattern
  INC DX
  DEC BX ;decrement counter
UNTIL BX == 0

POPF
RET
```

// A C/C++ function that multiplexes the 8-digit displays
// uses char sized array MEM
void Disp()
{
    unsigned int *Mem = &MEM[0]; //point to array element 0
    for ( int a = 0; a < 8; a++ )
    {
        unsigned char b = 0xff ^ ( 1 << a ); //form select pattern
        asm
        {
            mov al,b
            mov dx,701H
            out dx,al ;send select pattern to Port B
            mov al,Mem[a]
            dec dx
            out dx,al ;send data to Port A
        }
        Sleep(1); ;wait 1.0 ms
    }
}

The display procedure (DISP) addresses an area of memory where the data, in seven-segment code, are stored for the eight display digits called MEM. The AH register is loaded with a code (7FH) that initially addresses the most significant display position. Once this position is selected, the contents of memory location MEM +7 is addressed and sent to the most significant digit. The selection code is then adjusted to select the next display digit. This process repeats eight times to display the contents of location MEM through MEM +7 on the eight display digits.

The time delay of 1.0 ms can be obtained by writing a procedure that uses the system clock frequency to determine how long each instruction requires to execute. The procedure listed in Example 11–11 causes a time delay of a duration determined by the number of times that the LOOP instruction executes. Here XXXX was used and will be filled in with a value after a few facts are discussed. The LOOP instruction requires a certain number of clocks to execute—how many can be located in Appendix B. Suppose that the interface is using the 80486 microprocessor running with a 20 MHz clock. Appendix B represents that the LOOP instruction requires 7/6 clocks. The first number is the number of clocks required when a jump to D1 occurs and the second number is when the jump does not occur. With a 20 MHz clock, one clock requires 1 ÷ 20 MHz = 50 ns. The LOOP instruction, in this case, requires 350 ns to execute in all but the very last iteration. To determine the count (XXXX) needed to accomplish a 1.0 ms time delay, divide 1.0 ms by 350 ns. In this case XXXX = 2,857 to accomplish a 1.0 ms time delay. If a larger count occurs, a LOOPD instruction can be used with the ECX register. The time required to execute the MOV CX, XXXX, and RET instructions can usually be ignored.

Suppose a Core2 with a 2.0 GHz clock is used for the delay. Here one clock is 0.5 ns and LOOP requires five clocks per iteration. This requires a count of 400,000, so LOOPD would be used with ECX.

EXAMPLE 11–11

; equation for the delay
;
; Delay Time
; XXXX = -------------------
; time for LOOP
;
DELAY PROC NEAR USES CX
    MOV CX, XXXX
D1:
    LOOP D1
    RET
DELAY ENDP
If the program is written for the Windows environment, such as for use in an embedded system using embedded Windows, time delays can use a timer. The timer can operate with a precision of milliseconds, and in the embedded version of Windows, the delays are guaranteed.

An LCD Display Interfaced to the 82C55

LCDs (liquid crystal displays) have replaced LED displays in many applications. The only disadvantage of the LED display is that it is difficult to see in low-light situations in which the LED is still in limited use. An example is medical equipment for older people with poor eyesight. If the price of the OLED becomes low enough, LCD displays will disappear. A German company currently manufactures an OLED display panel that sells for under $10.

Figure 11–22 illustrates the connection of the Optrex DMC-20481 LCD display interfaced to an 82C55. The DMC-20481 is a 4-line by 20-characters-per-line display that accepts ASCII code as input data. It also accepts commands that initialize it and control its application. As you can see in Figure 11–22, the LCD display has few connections. The data connections, which are attached to the 82C55 port A, are used to input display data and to read information from the display. This illustrates an 8-bit interface. If a 4-bit interface is desired, D₄–D₇ pins are used for the data where the data must be formatted with the high nibble first, followed by the low nibble. A few newer OLED devices also contain a serial interface that uses a single pin for the data.

There are four control pins on the display. The V_{EE} connection is used to adjust the contrast of the LED display and is normally connected to a 10 KΩ potentiometer, as illustrated. The RS (register select) input selects data (RS = 1) or instructions (RS = 0). The E (enable) input must be a logic 1 for the DMC-20481 to read or write information and functions as a clock. Finally, the R/W pin selects a read or a write operation. Normally, the RS pin is placed at a 1 or 0, the R/W pin is set or cleared, data are placed on the data input pins, and then the E pin is pulsed to access the DMC-20481. This display also has two inputs (LEDA [anode] and LEDK [cathode]) for back-lighting LED diodes, which are not shown in the illustration.

In order to program the DMC-20481 we must first initialize it. This applies to any display that uses the HD44780 (Hitachi) display driver integrated circuit. The entire line of small display panels from Optrex and most other manufacturers is programmed in the same manner. Initialization is accomplished via the following steps:

1. Wait at least 15 ms after V_{CC} rises to 5.0 V.
2. Output the function set command (30H), and wait at least 4.1 ms.
3. Output the function set command (30H) a second time, and wait at least 100 μs.
4. Output the function set command (30H) a third time, and wait at least 40 μs.
5. Output the function set command (38H) a fourth time, and wait at least 40 μs.

*Current max is 480 mA, nominal 260 mA.
6. Output 08H to disable the display, and wait at least 40 µs.
7. Output a 01H to home the cursor and clear the display, and wait at least 1.64 ms.
8. Output the enable display cursor off (0CH), and wait at least 40 µs.
9. Output 06H to select auto-increment, shift the cursor, and wait at least 40 µs.

The software to accomplish the initialization of the LCD display is listed in Example 11–12. It is long, but the display controller requires the long initialization dialog. Note that the software for the three time delays is not included in the listing. If you are interfacing to a PC, you can use the RDTSC instruction as discussed in the Pentium chapter for the time delay. If you are developing the interface for another application, then you must write separate time delays, which must provide the delay times indicated in the initialization dialog. The time delays can also be obtained by using a timer in C++.

**EXAMPLE 11–12**

PORTA_ADDRESS EQU 700H ;set port addresses
PORTB_ADDRESS EQU 701H
COMMAND_ADDRESS EQU 703H

;macro to send a command or data to the LCD display
SEND MACRO PORTA_DATA, PORTB_DATA, DELAY
MOV AL,PORTA_DATA ;PORTA_DATA to Port A
MOV DX,PORTA_ADDRESS
OUT DX,AL
MOV AL,PORTB_DATA ;PORTB_DATA to Port B
MOV DX,PORTB_ADDRESS
OUT DX,AL
OR AL,00000100B ;Set E bit
OUT DX,AL ;send to Port B
AND AL,11111011B ;Clear E bit
NOP ;a small delay
NOP
NOP
OUT DX,AL ;send to Port B
MOV BL,DELAY ;BL = delay count
CALL MS_DELAY ;ms Time Delay
ENDM

;Program to initialize the LCD display
START:
MOV AL,80H ;Program the 82C55
MOV DX,COMMAND_ADDRESS
OUT DX,AL
MOV AL,0
MOV DX,PORTB_ADDRESS ;Clear Port B
SEND 30H, 2, 16 ;send 30H for 16 ms
SEND 30H, 2, 5 ;send 30H for 5 ms
SEND 30H, 2, 1 ;send 30H for 1 ms
SEND 38H, 2, 1 ;send 38H for 1 ms
SEND 8, 2, 1 ;send 8 for 1 ms
SEND 1, 2, 2 ;send 1 for 2 ms
SEND 0CH, 2, 1 ;send 0CH for 1 ms
SEND 6, 2, 1 ;send 6 for 1 ms

The NOP instructions are added in the SEND macro to ensure that the E bit remains a logic 1 long enough to activate the LCD display. This process should work in most systems at most clock frequencies, but additional NOP instructions may be needed to lengthen this time in some cases. Also notice that equate statements are used to equate the port addresses to labels. This is done so that the software can be changed easily if the port numbers differ from those used in the program.

Before programming the display, the commands used in the initialization dialog must be explained. See Table 11–3 for a complete listing of the commands or instructions for the LCD display. Compare the commands sent to the LCD display in the initialization program to Table 11–3.
Once the LCD display is initialized, a few procedures are needed to display information and control the display. After initialization, time delays are no longer needed when sending data or many commands to the display. The clear display command still needs a time delay because the busy flag is not used with that command. Instead of a time delay, the busy flag is tested to see whether the display has completed an operation. A procedure to test the busy flag appears in Example 11–13. The BUSY procedure tests the LCD display and only returns when the display has completed a prior instruction.

**EXAMPLE 11–13**

```
PORTA_ADDRESS EQU 700H ;set port addresses
PORTB_ADDRESS EQU 701H
COMMAND_ADDRESS EQU 703H

BUSY PROC NEAR USES DX AX

  PUSHF
  MOV DX,COMMAND_ADDRESS
  MOV AL,90H ;program Port A as IN
  OUT DX,AL

  .REPEAT
      MOV AL,5 ;select read from LCD
      MOV DX,PORTB_ADDRESS
      OUT DX,AL ;and pulse E
      NOP
      NOP
      MOV AL,1
      OUT DX,AL
```

**TABLE 11–3** Instructions for most LCD displays.

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Code</th>
<th>Description</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear display</td>
<td>0000 0001</td>
<td>Clears the display and homes the cursor</td>
<td>1.64 ms</td>
</tr>
<tr>
<td>Cursor home</td>
<td>0000 0010</td>
<td>Homes the cursor</td>
<td>1.64 ms</td>
</tr>
<tr>
<td>Entry mode set</td>
<td>0000 00AS</td>
<td>Sets cursor movement direction</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(A = 1, increment) and shift</td>
<td>40 µs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(S = 1, shift)</td>
<td></td>
</tr>
<tr>
<td>Display on/off</td>
<td>0000 1DCB</td>
<td>Sets display on/off (D = 1, on)</td>
<td>40 µs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(C = 1, cursor on)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(B = 1, cursor blink)</td>
<td></td>
</tr>
<tr>
<td>Cursor/display shift</td>
<td>0001 SR00</td>
<td>Sets cursor movement and display shift</td>
<td>40 µs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(S = 1, shift display)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(R = 1, right)</td>
<td></td>
</tr>
<tr>
<td>Function set</td>
<td>001L NF00</td>
<td>Programs LCD circuit (L = 1,</td>
<td>40 µs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8-bit interface) (N = 1, 2 lines)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(F = 1, 5 × 10 characters)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(F = 0, 5 × 7 characters)</td>
<td></td>
</tr>
<tr>
<td>Set CGRAM address</td>
<td>01XX XXXX</td>
<td>Sets character generator RAM address</td>
<td>40 µs</td>
</tr>
<tr>
<td>Set DRAM address</td>
<td>10XX XXXX</td>
<td>Sets display RAM address</td>
<td>40 µs</td>
</tr>
<tr>
<td>Read busy flag</td>
<td>B000 0000</td>
<td>Reads busy flag (B = 1, busy)</td>
<td>0</td>
</tr>
<tr>
<td>Write data</td>
<td>Data</td>
<td>Writes data to the display RAM</td>
<td>40 µs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>or the character generator RAM</td>
<td></td>
</tr>
<tr>
<td>Read data</td>
<td>Data</td>
<td>Reads data from the display RAM</td>
<td>40 µs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>or character generator RAM</td>
<td></td>
</tr>
</tbody>
</table>
MOV DX, PORTA_ADDRESS
MOV AL, DX ;read busy command
SHL AL, 1
JNE BUSY ;until not busy
MOV DX, COMMAND_ADDRESS
MOV AL, 80H
OUT DX, AL ;program Port A as OUT
POPF
RET

BUSY ENDP

Once the BUSY procedure is available, data can be sent to the display by writing another procedure called WRITE. The WRITE procedure uses BUSY to test before trying to write new data to the display. Example 11–14 shows the WRITE procedure, which transfers the ASCII character from the BL register to the current cursor position of the display. Note that the initialization dialog has sent the cursor for auto-increment, so if WRITE is called more than once, the characters written to the display will appear one next to the other, as they would on a video display.

**EXAMPLE 11–14**

WRITE PROC NEAR
MOV AL, BL ;BL to Port A
MOV DX, PORTA_ADDRESS
OUT DX, AL
MOV AL, 0 ;write ASCII
MOV DX, PORTB_ADDRESS
OUT DX, AL
OR AL, 00000100B ;Set E bit
OUT DX, AL ;send to Port B
AND AL, 11111011B ;Clear E bit
NOP ;a small delay
NOP
OUT DX, AL ;send to Port B
CALL BUSY ;wait for completion
RET
WRITE ENDP

The only other procedure that is needed for a basic display is the clear and home cursor procedure, called CLS, shown in Example 11–15. This procedure uses the SEND macro from the initialization software to send the clear command to the display. With CLS and the procedures presented thus far, you can display any message on the display, clear it, display another message, and basically operate the display. As mentioned earlier, the clear command requires a time delay (at least 1.64 ms) instead of a call to BUSY for proper operation.

**EXAMPLE 11–15**

CLS PROC NEAR
SEND 1, 2, 2
RET
CLS ENDP

Additional procedures that could be developed might select a display RAM position. The display RAM address starts at 0 and progresses across the display until the last character address on the first line is location 19, location 20 is the first display position of the second line, and so forth. Once you can move the display address, you can change individual characters on the display and even read data from the display. These procedures are for you to develop if they are needed.

A word about the display RAM inside of the LCD display. The LCD contains 128 bytes of memory, addressed from 00H to 7FH. Not all of this memory is always used. For example, the
A Stepper Motor Interlaced to the 82C55. Another device often interfaced to a computer system is the **stepper motor**. A stepper motor is a digital motor because it is moved in discrete steps as it traverses through 360°. A common stepper motor is geared to move perhaps 15° per step in an inexpensive stepper motor, to 1° per step in a more costly, high-precision stepper motor. In all cases, these steps are gained through many magnetic poles and/or gearing. Notice that two coils are energized in Figure 11–23. If less power is required, one coil may be energized at a time, causing the motor to step at 45°, 135°, 225°, and 315°.

Figure 11–23 shows a four-coil stepper motor that uses an armature with a single pole. Notice that the stepper motor is shown four times with the armature (permanent magnetic) rotated to four discrete places. This is accomplished by energizing the coils, as shown. This is an illustration of full stepping. The stepper motor is driven by using NPN Darlington amplifier pairs to provide a large current to each coil.

A circuit that can drive this stepper motor is illustrated in Figure 11–24, with the four coils shown in place. This circuit uses the 82C55 to provide it with the drive signals that are used to rotate the armature of the motor in either the right-hand or left-hand direction.

**Figure 11–23** The stepper motor showing full-step operation: (a) 45° (b) 135° (c) 225° (d) 315°.
A simple procedure that drives the motor (assuming that port A is programmed in mode 0 as an output device) is listed in Example 11–16 in both assembly language and as a function in C++. This subroutine is called, with CX holding the number of steps and direction of the rotation. If CX is greater than 8000H, the motor spins in the right-hand direction; if CX is less than 8000H, it spins in the left-hand direction. For example, if the number of steps is 0003H, the motor moves in the left-hand direction three steps and if the number of steps is 8003H, it moves three steps in the right-hand direction. The leftmost bit of CX is removed and the remaining 15 bits contain the number of steps. Notice that the procedure uses a time delay (not illustrated) that causes a 1 ms time delay. This time delay is required to allow the stepper-motor armature time to move to its next position.

**EXAMPLE 11–16**

PORT EQU 40H

; An assembly language procedure that controls the stepper motor
STEP PROC NEAR USES CX AX

    MOV AL, POS ; get position
    OR CX, CX ; set flag bits
    IF !ZERO?
        .IF !SIGN?
            .REPEAT
                ROL AL, 1 ; rotate step left
                OUT PORT, AL
                CALL DELAY ; wait 1 ms
            .UNTIL CXZ
        .ELSE
            AND CX, 7FFFH ; make CX positive
            .REPEAT
                ROR AL, 1 ; rotate step right
                OUT PORT, AL
                CALL DELAY ; wait 1 ms
            .UNTIL CXZ
        .ENDIF
    .ENDIF

    MOV POS, AL
    RET

STEP ENDP
// A C++ function that controls the stepper motor

char Step(char Pos, short Step)
{
    char Direction = 0;
    if (Step < 0)
    {
        Direction = 1;
        Step &= 0x8000;
    }
    while (Step)
    {
        if (Direction)
        {
            if ((Pos & 1) == 1)
            {
                Pos = (Pos >> 1) | 0x80;
            }
            else
            {
                Pos >>= 1;
            }
        }
        else
        {
            if ((Pos & 0x80) == 0x80)
            {
                Pos = (Pos << 1) | 1;
            }
            else
            {
                Pos <<= 1;
            }
        }
    }
    _asm
    {
        mov al,Pos
        out 40h, al
    }
    return Pos;
}

The current position is stored in memory location POS, which must be initialized with 33H, 66H, 0EEH, or 99H. This allows a simple ROR (step right) or ROL (step left) instruction to rotate the binary bit pattern for the next step.

The C++ version has two parameters: Pos is the current position of the stepper motor and Step is the number of steps as described earlier. The new Pos is returned in the C++ version instead of being stored in a variable.

Stepper motors can also be operated in the half-step mode, which allows eight steps per sequence. This is accomplished by using the full-step sequence described with a half step obtained by energizing one coil interspersed between the full steps. Half-stepping allows the armature to be positioned at 0°, 90°, 180°, and 270°. The half-step position codes are 11H, 22H, 44H, and 88H. A complete sequence of eight steps would be as follows: 11H, 33H, 22H, 66H, 44H, 0CCH, 88H, and 99H. This sequence could be either output from a lookup table or generated with software.

**Key Matrix Interface.** Keyboards come in a vast variety of sizes, from the standard 101-key QWERTY keyboards interfaced to the microprocessor to small specialized keyboards that may contain only four to 16 keys. This section of the text concentrates on the smaller keyboards that may be purchased preassembled or may be constructed from individual key switches.

Figure 11-25 illustrates a small key-matrix that contains 16 switches interfaced to ports A and B of an 82C55. In this example, the switches are formed into a 4 x 4 matrix, but any matrix could be used, such as a 2 x 8. Notice how the keys are organized into four rows (ROW0–ROW3)
FIGURE 11–25  A 4 × 4 keyboard matrix connected to an 8088 microprocessor through the 82C55 PIA.
and four columns (COL₀–COL₃). Each row is connected to 5.0 V through a 10 KΩ pull-up resistor to ensure that the row is pulled high when no push-button switch is closed.

The 82C55 is decoded (the PLD program is not shown) at I/O ports 50H–53H for an 8088 microprocessor. Port A is programmed as an input port to read the rows and port B is programmed as an output port to select a column. For example, if 1110 is output to port B pins PB₃–PB₀, column 0 has a logic 1, so the four keys in column 0 are selected. Notice that with a logic 0 on PB₀, the only switches that can place a logic 0 onto port A are switches 0–3. If switches 4–F are closed, the corresponding port A pins remain a logic 1. Likewise, if 1101 is output to port B, switches 4–7 are selected, and so forth.

A flowchart of the software required to read a key from the keyboard matrix and debounce the key is illustrated in Figure 11–26. Keys must be debounced, which is normally accomplished with a short time delay of 10–20 ms. The flowchart contains three main sections. The first waits for the release of a key. This seems awkward, but software executes very quickly in a microprocessor and there is a possibility that the program will return to the top of this program before

**FIGURE 11–26** The flowchart of a keyboard-scanning procedure.
the key is released, so we must wait for a release first. Next, the flowchart shows that we wait for a keystroke. Once the keystroke is detected, the position of the key is calculated in the final part of the flowchart.

The software uses a procedure called SCAN to scan the keys and another called DELAY10 (not shown in this example) to waste 10 ms of time for debouncing. The main keyboard procedure is called KEY and it appears with the others in Example 11–17. Example 11–17 also lists a C++ function to accomplish a key read operation. Note that the KEY procedure is generic, so it can handle any keyboard configuration from a $1 \times 1$ matrix to an $8 \times 8$ matrix. Changing the two equates at the start of the program (ROWS and COLS) will change the configuration of the software for any size keyboard. Also note that the steps required to initialize the 82C55 so that port A is an input port and port B is an output port are not shown.

With certain keyboards that do not follow the way keys are scanned, a lookup table may be needed to convert the raw key codes returned by KEY into key codes that match the keys on the keyboard. The lookup software is placed just before returning from KEY. It is merely a MOV BX,OFFSET TABLE followed by the XLAT instruction.

**EXAMPLE 11–17(a)**

```
;assembly language version;

;KEY scans the keyboard and returns the key code in AL.

COLS EQU 4
ROWS EQU 4
PORTA EQU 50H
PORTB EQU 51H

KEY PROC NEAR USES CX BX
  MOV  BL,FFH ;compute row mask
  SHL  BL,ROWS
  MOV  AL,0
  OUT  PORTB,AL ;place zeros on Port B

.REPEAT ;wait for release
  CALL SCAN
  UNTIL ZERO?
  CALL DELAY10
  CALL SCAN
  UNTIL ZERO?
.REPEAT ;wait for key
  CALL SCAN
  UNTIL !ZERO?
  CALL DELAY10
  CALL SCAN
  UNTIL !ZERO?

  MOV CX,00FEH
  .WHILE 1 ;find column
    MOV  AL,CL
    OUT  PORTB,AL
    CALL SHORTDELAY ;see text
    CALL SCAN
    .BREAK !ZERO?
    ADD CH,COLS
    ROL CL,1
  .ENDW
  .WHILE 1 ;find row
    SHR  AL,1
    .BREAK .IF !CARRY?
    INC  CH
  .ENDW
```

BASIC I/O INTERFACE

MOV AL,CH ;get key code
RET

KEY ENDP

SCAN PROC NEAR

IN AL,PORTA ;read rows
OR AL,BL
CMP AL,0FFH ;test for no keys
RET

SCAN ENDP

EXAMPLE 11–17(b)

// C++ language version of keyboard scanning software

#define ROWS 4
#define COLS 4
#define PORTA 50h
#define PORTB 51h

cchar Key()
{
    char mask = 0xff << ROWS;
    _asm
    {
        mov al,0 ;select all columns
        out PORTB,al
    }
    do
    {
        //wait for release
        while (Scan(mask));
        Delay();
    }
    while (Scan(mask));
    do
    {
        //wait for key press
        while (!Scan(mask));
        Delay();
    }
    while (!Scan(mask));
    unsigned char select = 0xfe;
    char key = 0;
    _asm
    {
        mov al,select
        out PortB,al
    }
    ShortDelay();
    while(!Scan(mask))
    {
        //calculate key code
        _asm
        {
            mov al,select
            rol al,1
            mov select,al
            out PortB,al
        }
        ShortDelay();
        key += COLS;
    }
    _asm
    {
        in al,PortA
        mov select,al
    }
}
while ((Select & 1) != 0)
{
    Select <<= 1;
    key ++;
}
return key;

bool Scan(mask)
{
    bool flag;
    _asm
    {
        in   al,PORTA
        mov  flag,al
    }
    return (flag | mask);
}

The ShortDelay procedure is needed because the computer changes port B at a very high rate of speed. The short time delay allows time for the data sent to port B to settle to their final state. In most cases, this is not needed if the scan rate (time between output instructions) of this part of the software does not exceed 30 KHz. If the scanning frequency is higher, the device generates radio interference. If it does, the Federal Communications Commission (FCC) will not approve its application in any accepted system. Without FCC Type A or Type B certification the system cannot be sold.

**Mode 1 Strobed Input**

Mode 1 operation causes port A and/or port B to function as latching input devices. This allows external data to be stored into the port until the microprocessor is ready to retrieve it. Port C is also used in mode 1 operation—not for data, but for control or handshaking signals that help operate either or both port A and port B as strobed input ports. Figure 11–27 shows how both ports are structured for mode 1 strobed input operation and the timing diagram.

The strobed input port captures data from the port pins when the strobe $\text{STB}$ is activated. Note that the strobe captures the port data on the 0-to-1 transition. The $\text{STB}$ signal causes data to be captured in the port, and it activates the $\text{IBF}$ ($\text{input buffer full}$) and $\text{INTR}$ ($\text{interrupt request}$) signals. Once the microprocessor, through software ($\text{IBF}$) or hardware ($\text{INTR}$), notices that data are strobed into the port, it executes an $\text{IN}$ instruction to read the port $\text{RD}$. The act of reading the port restores both $\text{IBF}$ and $\text{INTR}$ to their inactive states until the next datum is strobed into the port.

**Signal Definitions for Mode 1 Strobed Input**

- **$\text{STB}$** The strobe input loads data into the port latch, which holds the information until it is input to the microprocessor via the IN instruction.
- **$\text{IBF}$** Input buffer full is an output indicating that the input latch contains information.
- **$\text{INTR}$** Interrupt request is an output that requests an interrupt. The INTR pin becomes a logic 1 when the $\text{STB}$ input returns to a logic 1, and is cleared when the data are input from the port by the microprocessor.
- **$\text{INTE}$** The interrupt enable signal is neither an input nor an output; it is an internal bit programmed via the port $\text{PC}_4$ (port A) or $\text{PC}_2$ (port B) bit position.
- **$\text{PC}_7$, $\text{PC}_6$** The port C pins 7 and 6 are general-purpose I/O pins that are available for any purpose.

**Strobed Input Example.** An excellent example of a strobed input device is a keyboard. The keyboard encoder debounces the key switches and provides a strobe signal whenever a key is
FIGURE 11–27  Strobed input operation (mode 1) of the 82C55. (a) Internal structure and (b) timing diagram.

FIGURE 11–28  Using the 82C55 for strobed input operation of a keyboard.
DEPRESSED and the data output contain the ASCII-coded key code. Figure 11–28 illustrates a key-
board connected to strobed input port A. Here DAV (data available) is activated for 1.0 µs each
time that a key is typed on the keyboard. This causes data to be strobed into port A because DAV
is connected to the STB input of port A. Each time a key is typed, therefore, it is stored into port A
of the 82C55. The STB input also activates the IBF signal, indicating that data are in port A.

Example 11–17 shows a procedure that reads data from the keyboard each time a key is
typed. This procedure reads the key from port A and returns with the ASCII code in AL. To
detect a key, port C is read and the IBF bit (bit position PC5) is tested to see whether the buffer is
full. If the buffer is empty (IBF = 0), then the procedure keeps testing this bit, waiting for a char-
acter to be typed on the keyboard.

EXAMPLE 11–18

;A procedure that reads the keyboard encoder and
;returns the ASCII key code in AL

BIT5 EQU 20H
PORTC EQU 22H
PORTA EQU 20H

READ PROC NEAR

.REPEAT ;poll IBF bit
    IN   AL,PORTC
    TEST AL,BIT5
    .UNTIL !ZERO?
    IN   AL,PORTA ;get ASCII data
    RET

READ ENDP

Mode 1 Strobed Output

Figure 11–29 illustrates the internal configuration and timing diagram of the 82C55 when it is
operated as a strobed output device under mode 1. Strobed output operation is similar to mode 0
output operation, except that control signals are included to provide handshaking.

Whenever data are written to a port programmed as a strobed output port, the OBF (output
buffer full) signal becomes a logic 0 to indicate that data are present in the port latch. This sig-
nal indicates that data are available to an external I/O device that removes the data by strobing
the ACK (acknowledge) input to the port. The ACK signal returns the OBF signal to a logic 1,
indicating that the buffer is not full.

Signal Definitions for Mode 1 Strobed Output

OBF Output buffer full is an output that goes low whenever data are output
(OUT) to the port A or port B latch. This signal is set to a logic 1 whenever
the ACK pulse returns from the external device.

ACK The acknowledge signal causes the OBF pin to return to a logic 1 level.
The ACK signal is a response from an external device, indicating that it has
received the data from the 82C55 port.

INTR Interrupt request is a signal that often interrupts the microprocessor
when the external device receives the data via the ACK signal. This pin is
qualified by the internal INTE (interrupt enable) bit.

INTE Interrupt enable is neither an input nor an output; it is an internal bit
programmed to enable or disable the INTR pin. The INTE A bit is pro-
grammed using the PC6 bit and INTE B is programmed using the PC2 bit.
PC₄, PC₅

Port C pins PC₄ and PC₅ are general-purpose I/O pins. The bit set and reset command is used to set or reset these two pins.

Strobed Output Example. The printer interface discussed in Section 11–1 is used here to demonstrate how to achieve strobed output synchronization between the printer and the 82C55. Figure 11–30 illustrates port B connected to a parallel printer, with eight data inputs for receiving ASCII-coded data, a DS (data strobe) input to strobe data into the printer, and an ACK output to acknowledge the receipt of the ASCII character.

In this circuit, there is no signal to generate the DS signal to the printer, so PC₄ is used with software that generates the DS signal. The ACK signal that is returned from the printer acknowledges the receipt of the data and is connected to the ACK input of the 82C55.

Example 11–19 lists the software that sends the ASCII-coded character in AH to the printer. The procedure first tests OBF to decide whether the printer has removed the data from port B. If not, the procedure waits for the ACK signal to return from the printer. If OBF = 1, then the procedure sends the contents of AH to the printer through port B and also sends the DS signal.
EXAMPLE 11–19

; A procedure that transfers an ASCII character from AH to the printer connected to port B

BIT1 EQU 2
PORTC EQU 63H
PORTB EQU 61H
CMD EQU 63H

PRINT PROC NEAR

.REPEAT ; wait for printer ready
    IN AL, PORTC
    TEST AL, BIT1
    .UNTIL !ZERO?

    MOV AL, AH ; send ASCII
    OUT PORTB, AL

    MOV AL, 8 ; pulse data strobe
    OUT CMD, AL
    MOV AL, 9
    OUT CMD, AL

RET

PRINT ENDP

Mode 2 Bidirectional Operation

In mode 2, which is allowed with group A only, port A becomes bidirectional, allowing data to be transmitted and received over the same eight wires. Bidirectional bused data are useful when interfacing two computers. It is also used for the IEEE-488 parallel high-speed GPIB (general-purpose instrumentation bus) interface standard. Figure 11–31 shows the internal structure and timing diagram for mode 2 bidirectional operation.

Signal Definitions for Bidirectional Mode 2

INTR Interrupt request is an output used to interrupt the microprocessor for both input and output conditions.

OBF Output buffer full is an output indicating that the output buffer contains data for the bidirectional bus.

ACK Acknowledge is an input that enables the three-state buffers so that data can appear on port A. If ACK is a logic 1, the output buffers of port A are at their high-impedance state.
The **strobe** input loads the port A input latch with external data from the bidirectional port A bus.

**IBF** *Input buffer full* is an output used to signal that the input buffer contains data for the external bidirectional bus.

**INTE** *Interrupt enable* are internal bits (INTE1 and INTE2) that enable the INTR pin. The state of the INTR pin is controlled through port C bits PC₆ (INTE1) and PC₄ (INTE2).

**PC₀, PC₁, and PC₂** These pins are general-purpose I/O pins in mode 2 controlled by the bit set and reset command.

**The Bidirectional Bus.** The bidirectional bus is used by referencing port A with the IN and OUT instructions. To transmit data through the bidirectional bus, the program first tests the **OBF** signal.
to determine whether the output buffer is empty. If it is, then data are sent to the output buffer via the OUT instruction. The external circuitry also monitors the OBF signal to decide whether the microprocessor has sent data to the bus. As soon as the output circuitry sees a logic 0 on OBF, it sends back the ACK signal to remove it from the output buffer. The ACK signal sets the OBF bit and enables the three-state output buffers so that data may be read. Example 11–20 lists a procedure that transmits the contents of the AH register through bidirectional port A.

**EXAMPLE 11–20**

; A procedure transmits AH through the bidirectional bus

BIT7 EQU 80H
PORTC EQU 62H
PORTA EQU 60H
TRANS PROC NEAR
    .REPEAT ; test OBF
        IN AL, PORTC
        TEST AL, BIT7
    .UNTIL !ZERO?
    MOV AL, AH ; send data
    OUT PORTA, AL
RET
TRANS ENDP

To receive data through the bidirectional port A bus, the IBF bit is tested with software to decide whether data have been strobed into the port. If IBF = 1, then data is input using the IN instruction. The external interface sends data into the port by using the STB signal. When STB is activated, the IBF signal becomes a logic 1 and the data at port A are held inside the port in a latch. When the IN instruction executes, the IBF bit is cleared and the data in the port are moved into AL. Example 11–21 lists a procedure that reads data from the port.

**EXAMPLE 11–21**

; A procedure that reads data from the bidirectional bus into AL

BIT5 EQU 20H
PORTC EQU 62H
PORTA EQU 60H
READ PROC NEAR
    .REPEAT ; test IBF
        IN AL, PORTC
        TEST AL, BIT5
    .UNTIL !ZERO?
    IN AL, PORTA
RET
READ ENDP

The INTR (interrupt request) pin can be activated from both directions of data flow through the bus. If INTR is enabled by both INTE bits, then the output and input buffers both cause interrupt requests. This occurs when data are strobed into the buffer using STB or when data are written using OUT.

**82C55 Mode Summary**

Figure 11–32 shows a graphical summary of the three modes of operation for the 82C55. Mode 0 provides simple I/O, mode 1 provides strobed I/O, and mode 2 provides bidirectional I/O. As mentioned, these modes are selected through the command register of the 82C55.
The Serial EEPROM Interface

In Chapter 10, Figure 10–23, a serial EEPROM is illustrated, but at that point in the text, no I/O existed for an interface. Suppose that port C of an 82C55 is used for connection to this interface and software is needed to drive the interface. It is assumed that pin PC4 connects to the SCL input and pin PC0 connects to the serial connection (SDA). PC4 is programmed as an output pin to provide a clock signal. The PC0 pin is programmed as an output to send data and as an input to receive data from the EEPROM.

Refer to Figure 10–24; the data format for the software for reading and writing data to the EEPROM is also illustrated in Example 11–22. This software is written in C with some assembly language, but it can also be developed in assembly language. The I/O port addresses are 0x1203 for the command register and 0x1202 for the port C register. The time delay should be $1.25 \mu s$ for a data rate of 400 KHz. Note that the time delay software is not illustrated here and the while loop is used to wait for an ACK signal after a write.

**EXAMPLE 11–22**

```c
unsigned char void PC0in(unsigned char bit)
{
    _asm
    {
        mov dx,1203h
        mov al,81h
        out dx,al
        dec dx
        mov al,bit
        out dx,al
    }
    Delay();
    _asm
    {
        mov dx,1202h
        in al,dx ;al is returned
    }
}

void PC0out(unsigned char bit)
{
    _asm
    {
        mov dx,1203h
        mov al,80h
        out dx,al
        dec dx
    }
}
```
CHAPTER 11

```c
mov al,bit
out dx,al
}
Delay();
}

unsigned char void SendByte(unsigned char data)
{
    for (int a = 7; a >= 0; a--)
    {
        PC0out((data >> a) & 0xef);
        PC0out((data >> a) | 0x10);
    }
    PC0in(0xef); //ack bit
    return PC0in(0x10);
}

unsigned char GetByte()
{
    unsigned char temp = 0;
    for (int a = 7; a >= 0; a--)
    {
        PC0in(0xef);
        temp |= PC0in(0x10) << a;
    }
    PC0in(0xef); //ack bit
    PC0in(0x10);
    return temp;
}

void SendStart()
{
    // start is one
    PC0out(0xef);
    PC0out(0x10);
}

void SendStop()
{
    // stop is zero
    PC0out(0xee);
    PC0out(0x10);
}

void SendData(char device, short address, unsigned char data)
{
    Char c = 0;
    SendStart();
    SendByte(0xa0 | device << 1);
    SendByte(address >> 8);
    SendByte(address);
    SendByte(data);
    while (c == 0)
    {
        // wait for ACK = 1;
        c = SendByte(0xa0 | device << 1);
    }
    SendStop();
}

unsigned char ReadData(char device, short address)
{
    SendStart();
    SendByte(0xa0 | device << 1);
    SendByte(address >> 8);
    SendByte(address);
    SendByte(0xa1 | device << 1);
    unsigned char temp = GetByte();
    SendStop();
    return temp;
}
```
The 8254 programmable interval timer consists of three independent 16-bit programmable counters (timers). Each counter is capable of counting in binary or binary-coded decimal (BCD). The maximum allowable input frequency to any counter is 10 MHz. This device is useful wherever the microprocessor must control real-time events. Some examples of usage include real-time clock and an events counter, and for motor speed and direction control.

This timer also appears in the personal computer decoded at ports 40H–43H to do the following:

1. Generate a basic timer interrupt that occurs at approximately 18.2 Hz.
2. Cause the DRAM memory system to be refreshed.
3. Provide a timing source to the internal speaker and other devices. The timer in the personal computer is an 8253 instead of an 8254.

8254 Functional Description

Figure 11–33 shows the pin-out of the 8254, which is a higher-speed version of the 8253, and a diagram of one of the three counters. Each timer contains a CLK input, a gate input, and an output (OUT) connection. The CLK input provides the basic operating frequency to the timer, the gate pin controls the timer in some modes, and the OUT pin is where we obtain the output of the timer.

The signals that connect to the microprocessor are the data bus pins (D_7–D_0), RD, WR, CS, and address inputs A_1 and A_0. The address inputs are present to select any of the four internal registers used for programming, reading, or writing to a counter. The personal computer contains an 8253 timer or its equivalent, decoded at I/O ports 40H–43H. Timer zero is programmed to generate an 18.2 Hz signal that interrupts the microprocessor at interrupt vector 8 for a clock tick. The tick is often used to time programs and events in DOS. Timer 1 is programmed for 15 µs, which is used on the personal computer to request a DMA action used to refresh the dynamic RAM. Timer 2 is programmed to generate a tone on the personal computer speaker.
TABLE 11–4  Address selection inputs to the 8254.

<table>
<thead>
<tr>
<th>( A_1 )</th>
<th>( A_0 )</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Counter 0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Counter 1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Counter 2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Control word</td>
</tr>
</tbody>
</table>

Pin Definitions

\( A_0 \), \( A_1 \)  
The address inputs select one of four internal registers within the 8254. See Table 11–4 for the function of the \( A_1 \) and \( A_0 \) address bits.

CLK  
The clock input is the timing source for each of the internal counters. This input is often connected to the PCLK signal from the microprocessor system bus controller.

CS  
Chip select enables the 8254 for programming and reading or writing a counter.

G  
The gate input controls the operation of the counter in some modes of operation.

GND  
Ground connects to the system ground bus.

OUT  
A counter output is where the waveform generated by the timer is available.

RD  
Read causes data to be read from the 8254 and often connects to the IORC signal.

Vcc  
Power connects to the +5.0 V power supply.

WR  
Write causes data to be written to the 8254 and often connects to the write strobe (IOWC).

Programming the 8254

Each counter is individually programmed by writing a control word, followed by the initial count. Figure 11–34 lists the program control word structure of the 8254. The control word allows the programmer to select the counter, mode of operation, and type of operation (read/write). The control word also selects either a binary or BCD count. Each counter may be programmed with a count of 1 to FFFFH. A count of 0 is equal to FFFFH+1 (65,536) or 10,000.

FIGURE 11–34  The control word for the 8254-2 timer.
in BCD. The minimum count of 1 applies to all modes of operation except modes 2 and 3, which have a minimum count of 2. Timer 0 is used in the personal computer with a divide-by count of 64K (FFFFH) to generate the 18.2 Hz (18.196 Hz) interrupt clock tick. Timer 0 has a clock input frequency of 4.77 MHz + 4 or 1.1925 MHz.

The control word uses the BCD bit to select a BCD count (BCD = 1) or a binary count (BCD = 0). The M₂, M₁, and M₀ bits select one of the six different modes of operation (000–101) for the counter. The RW₁ and RW₀ bits determine how the data are read from or written to the counter. The SC₁ and SC₀ bits select a counter or the special read-back mode of operation, discussed later in this section.

Each counter has a program control word used to select the way the counter operates. If two bytes are programmed into a counter, then the first byte (LSB) will stop the count, and the second byte (MSB) will start the counter with the new count. The order of programming is important for each counter, but programming of different counters may be interleaved for better control. For example, the control word may be sent to each counter before the counts for individual programming. Example 11–23 shows a few ways to program counters 1 and 2. The first method programs both control words, then the LSB of the count for each counter, which stops them from counting. Finally, the MSB portion of the count is programmed, starting both counters with the new count. The second example shows one counter programmed before the other.

**EXAMPLE 11–23**

```plaintext
PROGRAM CONTROL WORD 1  PROGRAM CONTROL WORD 2  PROGRAM LSB 1
PROGRAM LSB 2
PROGRAM MSB 1
PROGRAM MSB 2

;setup counter 1
;setup counter 2
;stop counter 1 and program LSB
;stop counter 2 and program LSB ;program MSB of counter 1 and start it
;program MSB of counter 2 and start it

or

PROGRAM CONTROL WORD 1  PROGRAM LSB 1
PROGRAM MSB 1
PROGRAM CONTROL WORD 2  PROGRAM LSB 2
PROGRAM MSB 2

;setup counter 1
;stop counter 1 and program LSB ;program MSB of counter 1 and start it
;setup counter 2
;stop counter 2 and program LSB ;program MSB of counter 2 and start it
```

**Modes of Operation.** Six modes (mode 0–mode 5) of operation are available to each of the 8254 counters. Figure 11–35 shows how each of these modes functions with the CLK input, the gate (G) control signal, and OUT signal. A description of each mode follows:

**MODE 0**

Allows the 8254 counter to be used as an events counter. In this mode, the output becomes a logic 0 when the control word is written and remains there until N plus the number of programmed counts. For example, if a count of 5 is programmed, the output will remain a logic 0 for 6 counts beginning with N. Note that the gate (G) input must be a logic 1 to allow the counter to count. If G becomes a logic 0 in the middle of the count, the counter will stop until G again becomes a logic 1.

**MODE 1**

Causes the counter to function as a retriggerable, monostable multivibrator (one-shot). In this mode the G input triggers the counter so that it develops a pulse at the OUT connection that becomes a logic 0 for the duration of the
CHAPTER 11

If the count is 10, then the OUT connection goes low for 10 clocking periods when triggered. If the G input occurs within the duration of the output pulse, the counter is again reloaded with the count and the OUT connection continues for the total length of the count.

MODE 2

Allows the counter to generate a series of continuous pulses that are one clock pulse wide. The separation between pulses is determined by the count. For example, for a count of 10, the output is a logic 1 for nine clock periods and low for one clock period. This cycle is repeated until the counter is programmed with a new count or until the G pin is placed at a logic 0 level. The G input must be a logic 1 for this mode to generate a continuous series of pulses.
MODE 3
Generates a continuous square wave at the OUT connection, provided that the G pin is a logic 1. If the count is even, the output is high for one half of the count and low for one half of the count. If the count is odd, the output is high for one clocking period longer than it is low. For example, if the counter is programmed for a count of 5, the output is high for three clocks and low for two clocks.

MODE 4
Allows the counter to produce a single pulse at the output. If the count is programmed as a 10, the output is high for 10 clocking periods and low for one clocking period. The cycle does not begin until the counter is loaded with its complete count. This mode operates as a software triggered one-shot. As with modes 2 and 3, this mode also uses the G input to enable the counter. The G input must be a logic 1 for the counter to operate for these three modes.

MODE 5
A hardware triggered one-shot that functions as mode 4, except that it is started by a trigger pulse on the G pin instead of by software. This mode is also similar to mode 1 because it is retriggerable.

Generating a Waveform with the 8254. Figure 11–36 shows an 8254 connected to function at I/O ports 0700H, 0702H, 0704H, and 0706H of an 80386SX microprocessor. The addresses are decoded by using a PLD that also generates a write strobe signal for the 8254, which is connected to the low-order data bus connections. The PLD also generates a wait signal for the microprocessor that causes two wait states when the 8254 is accessed. The wait state generator connected to the microprocessor actually controls the number of wait states inserted into the timing. The program for the PLD is not illustrated here because it is the same as many of the prior examples.

FIGURE 11–36 The 8254 interfaced to an 8 MHz 8086 so that it generates a 100 KHz square wave at OUT0 and a 200 KHz continuous pulse at OUT1.
Example 11–24 lists the program that generates a 100 KHz square-wave at OUT0 and a 200 KHz continuous pulse at OUT1. Counter 0 uses mode 3 and counter 1 uses mode 2. The count programmed into counter 0 is 80 and the count for counter 1 is 40. These counts generate the desired output frequencies with an 8 MHz input clock.

EXAMPLE 11–24

;A procedure that programs the 8254 timer to function as illustrated in Figure 11–36

TIME PROC NEAR USES AX DX
  MOV DX,706H ;program counter 0 for mode 3
  MOV AL,00110110B
  OUT DX,AL
  MOV DX,0701H ;program counter 1 for mode 2
  MOV AL,01110100B
  OUT DX,AL
  MOV DX,700H ;program counter 0 with 80
  MOV AL,80
  OUT DX,AL
  MOV AL,0
  OUT DX,AL
  MOV DX,702H ;program counter 1 with 40
  MOV AL,40
  OUT DX,AL
  MOV AL,0
  OUT DX,AL
RET

TIME ENDP

Reading a Counter. Each counter has an internal latch that is read with the read counter port operation. These latches will normally follow the count. If the contents of the counter are needed, then the latch can remember the count by programming the counter latch control word (see Figure 11–37), which causes the contents of the counter to be held in a latch until they is read. Whenever a read from the latch or the counter is programmed, the latch tracks the contents of the counter.

When it is necessary for the contents of more than one counter to be read at the same time, we use the read-back control word, illustrated in Figure 11–38. With the read-back control word, the CNT bit is a logic 0 to cause the counters selected by CNT0, CNT1, and CNT2 to be latched. If the status register is to be latched, then the ST bit is placed at a logic 0. Figure 11–39 shows the status register, which shows the state of the output pin, whether the counter is at its null state (0), and how the counter is programmed.

![Figure 11–37](image-url) The 8254-2 counter latch control word.

<table>
<thead>
<tr>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC1</td>
<td>SC0</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Select counter
00 = counter 0
01 = counter 1
10 = counter 2
11 = read-back command
DC Motor Speed and Direction Control

One application of the 8254 timer is as a motor speed controller for a DC motor. Figure 11–40 shows the schematic diagram of the motor and its associated driver circuitry. It also illustrates the interconnection of the 8254, a flip-flop, and the motor and its driver.

The operation of the motor driver circuitry is straightforward. If the Q output of the 74ALS112 is a logic 1, the base Q₂ is pulled up to +12 V through the base pull-up resistor, and the base of Q₂ is open circuited. This means that Q₁ is off and Q₂ is on, with ground applied to the positive lead of the motor. The bases of both Q₁ and Q₂ are pulled low to ground through the inverters. This causes Q₂ to conduct or turn on and Q₁ to turn off, applying ground to the negative lead of the motor. The logic 1 at the Q output of the flip-flop therefore connects +12 V to the positive lead of the motor and ground to the negative lead. This connection causes the motor to spin in its forward direction. If the state of the Q output of the flip-flop becomes a logic 0, then the conditions of the transistors are reversed and +12 V is attached to the negative lead of the motor, with ground attached to the positive lead. This causes the motor to spin in the reverse direction.

If the output of the flip-flop is alternated between a logic 1 and 0, the motor spins in either direction at various speeds. If the duty cycle of the Q output is 50%, the motor will not spin at all and exhibits some holding torque because current flows through it. Figure 11–41 shows some timing diagrams and their effects on the speed and direction of the motor. Notice how each counter generates pulses at different positions to vary the duty cycle at the Q output of the flip-flop. This output is also called pulse width modulation.

To generate these wave forms, counters 0 and 1 are both programmed to divide the input clock (PCLK) by 30,720. We change the duty cycle of Q by changing the point at which
FIGURE 11–40  Motor speed and direction control using the 8254 timer.
counter 1 is started in relationship to counter 0. This changes the direction and speed of the motor. But why divide the 8 MHz clock by 30,720? The divide rate of 30,720 is divisible by 256, so we can develop a short program that allows 256 different speeds. This also produces a basic operating frequency for the motor of about 260 Hz, which is low enough in frequency to power the motor. It is important to keep this operating frequency below 1000 Hz, but above 60 Hz.

Example 11–25 lists a procedure that controls the speed and direction of the motor. The speed is controlled by the value of AH when this procedure is called. Because we have an 8-bit number to represent speed, a 50% duty cycle, for a stopped motor, is a count of 128. By changing the value in AH when the procedure is called, we can adjust the motor speed. The speed of the motor will increase in either direction by changing the number in AH when this procedure is called. As the value in AH approaches 00H, the motor begins to increase its speed in the reverse

FIGURE 11–41  Timing for the motor speed and direction control circuit of Figure 11–40. (a) No rotation, (b) high-speed rotation in the reverse direction, and (c) high-speed rotation in the forward direction.
direction. As the value of AH approaches FFH, the motor increases its speed in the forward
direction.

**EXAMPLE 11–25**

; A procedure that controls the speed and direction of the motor
; in Figure 11–40.
; AH determines the speed and direction of the motor where
; AH is between 00H and FFH.

CNTR EQU 706H
CNT0 EQU 700H
CNT1 EQU 702H
COUNT EQU 30720

SPEED PROC NEAR USES BX DX AX
  MOV  BL,AH           ;calculate count
  MOV  AX,120
  MUL  BL
  MOV  BX,AX
  MOV  AX,COUNT
  SUB  AX,BX
  MOV  BX,AX

  MOV  DX,CNTR
  MOV  AL,00110100B ;program control words
  OUT  DX,AL
  MOV  AL,01110100B
  OUT  DX,AL

  MOV  DX,CNT1        ;program counter 1
  MOV  AX,COUNT      ;to generate a clear
  OUT  DX,AL
  MOV  AL,AH
  OUT  DX,AL

  .REPEAT              ;wait for counter 1
  IN   AL,DX
  XCHG  AL,AH
  IN   AL,DX
  XCHG  AL,AH
  .UNTIL  BX == AX

  MOV  DX,CNT0        ;program counter 0
  MOV  AX,COUNT      ;to generate a set
  OUT  DX,AL
  MOV  AL,AH
  OUT  DX,AL

  RET

SPEED ENDP

The procedure adjusts the wave form at Q by first calculating the count at which counter 0 is
to start in relationship to counter 1. This is accomplished by multiplying AH by 120 and then
subtracting it from 30,720. This is required because the counters are down-counters that count
from the programmed count to 0 before restarting. Next, counter 1 is programmed with a count
of 30,720 and started so it generates the clear-wave form for the flip-flop. After counter 1 is
started, it is read and compared with the calculated count. Once it reaches this count, counter 0 is
started with a count of 30,720. From this point forward, both counters continue generating the
clear and set wave forms until the procedure is again called to adjust the speed and direction of
the motor.
16550 PROGRAMMABLE COMMUNICATIONS INTERFACE

The National Semiconductor Corporation’s PC16550D is a programmable communications interface designed to connect to virtually any type of serial interface. The 16550 is a universal asynchronous receiver/transmitter (UART) that is fully compatible with the Intel microprocessors. The 16550 is capable of operating at 0–1.5 M baud. Baud rate is the number of bits transferred per second (bps), including start, stop, data, and parity (Bps is bytes per second and bps are bits per second). The 16550 also includes a programmable baud rate generator and separate FIFOs for input and output data to ease the load on the microprocessor. Each FIFO contains 16 bytes of storage. This is the most common communications interface found in modem microprocessor-based equipment, including the personal computer and many modems.

Asynchronous Serial Data

Asynchronous serial data are transmitted and received without a clock or timing signal. Figure 11–42 illustrates two frames of asynchronous serial data. Each frame contains a start bit, seven data bits, parity, and one stop bit. The figure shows a frame that contains one ASCII character and 10 bits. Most dial-up communications systems of the past, such as CompuServe, Prodigy, and America Online, used 10 bits for asynchronous serial data with even parity. Most Internet and bulletin board services also use 10 bits, but they normally do not use parity. Instead, eight data bits are transferred, replacing parity with a data bit. This makes byte transfers of non-ASCII data much easier to accomplish.

16550 Functional Description

Figure 11–43 illustrates the pin-out of the 16550 UART. This device is available as a 40-pin DIP (dual in-line package) or as a 44-pin PLCC (plastic leadless chip carrier). Two completely separate sections are responsible for data communications: the receiver and the transmitter. Because each of these sections is independent, the 16550 is able to function in simplex, half-duplex, or full-duplex modes. One of the main features of the 16550 is its internal receiver and transmitter FIFO (first-in, first-out) memories. Because each is 16 bytes deep, the UART requires attention only from the microprocessor after receiving 16 bytes of data. It also holds 16 bytes before the microprocessor must wait for the transmitter. The FIFO makes this UART ideal when interfacing to high-speed systems because less time is required to service it.

An example simplex system is one in which the transmitter or receiver is used by itself such as in an FM (frequency modulation) radio station. An example half-duplex system is a CB (citizens band) radio, on which we transmit and receive, but not both at the same time. The full-duplex system allows transmission and reception in both directions simultaneously. An example of a full-duplex system is the telephone.

The 16550 can control a modem (modulator/demodulator), which is a device that converts TTL levels of serial data into audio tones that can pass through the telephone system. Six pins on

---

**FIGURE 11–42** Asynchronous serial data.
the 16650 are devoted to modem control: \( \text{DSR} \) (data set ready), \( \text{DTR} \) (data terminal ready), \( \text{CTS} \) (clear-to-send), \( \text{RTS} \) (request-to-send), \( \text{RI} \) (ring indicator), and \( \text{DCD} \) (data carrier detect). The modem is referred to as the data set and the 16650 is referred to as the data terminal.

### 16550 Pin Functions

- **A0, A1, A2**: The **address inputs** are used to select an internal register for programming and also data transfer. See Table 11–5 for a list of each combination of the address inputs and the registers selected.

- **ADS**: The **address strobe** input is used to latch the address lines and chip select lines. If not needed (as in the Intel system), connect this pin to ground. The ADS pin is designed for use with Motorola microprocessors.

- **BAUDOUT**: The **baud out** pin is where the clock signal generated by the baud rate generator from the transmitter section is made available. It is most often connected to the RCLK input to generate a receiver clock that is equal to the transmitter clock.

- **CS0, CS1, CS2**: The **chip select** inputs must all be active to enable the 16550 UART.

- **CTS**: The **clear-to-send** (if low) indicates that the modem or data set is ready to exchange information. This pin is often used in a half-duplex system to turn the line around.

### Table 11–5

<table>
<thead>
<tr>
<th>A2</th>
<th>A1</th>
<th>A0</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Receiver buffer (read) and transmitter holding (write)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Interrupt enable</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Interrupt identification (read) and FIFO control (write)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Line control</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Modern control</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Line status</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Modern status</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Scratch</td>
</tr>
</tbody>
</table>
### Basic I/O Interface

<table>
<thead>
<tr>
<th>PIN</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D0–D7</td>
<td>The data bus pins are connected to the microprocessor data bus.</td>
</tr>
<tr>
<td>DCD</td>
<td>The data carrier detect input is used by the modem to signal the 16550 that a carrier is present.</td>
</tr>
<tr>
<td>DDIS</td>
<td>The disable driver output becomes a logic 0 to indicate that the microprocessor is reading data from the UART. DDIS can be used to change the direction of data flow through a buffer.</td>
</tr>
<tr>
<td>DSR</td>
<td>Data set ready is an input to the 16550, indicating that the modem or data set is ready to operate.</td>
</tr>
<tr>
<td>DTR</td>
<td>Data terminal ready is an output that indicates that the data terminal (16550) is ready to function.</td>
</tr>
<tr>
<td>INTR</td>
<td>Interrupt request is an output to the microprocessor used to request an interrupt (INTR = 1) whenever the 16550 has a receiver error, it has received data, and the transmitter is empty.</td>
</tr>
<tr>
<td>MR</td>
<td>Master reset initializes the 16550 and should be connected to the system RESET signal.</td>
</tr>
<tr>
<td>OUT1, OUT2</td>
<td>User-defined output pins that can provide signals to a modem or any other device as needed in a system.</td>
</tr>
<tr>
<td>RCLK</td>
<td>Receiver clock is the clock input to the receiver section of the UART. This input is always 16 times the desired receiver baud rate.</td>
</tr>
<tr>
<td>RD, RD</td>
<td>Read inputs (either may be used) cause data to be read from the register specified by the address inputs to the UART.</td>
</tr>
<tr>
<td>RI</td>
<td>The ring indicator input is placed at the logic 0 level by the modem to indicate that the telephone is ringing.</td>
</tr>
<tr>
<td>RTS</td>
<td>Request-to-send is a signal to the modem indicating that the UART wishes to send data.</td>
</tr>
<tr>
<td>SIN, SOUT</td>
<td>These are the serial data pins. SIN accepts serial data and SOUT transmits serial data.</td>
</tr>
<tr>
<td>RXRDY</td>
<td>Receiver ready is a signal used to transfer received data via DMA techniques (see text).</td>
</tr>
<tr>
<td>TXRDY</td>
<td>Transmitter ready is a signal used to transfer transmitter data via DMA techniques (see text).</td>
</tr>
<tr>
<td>WR, WR</td>
<td>Write (either may be used) connects to the microprocessor write signal to transfer commands and data to the 16550.</td>
</tr>
<tr>
<td>XIN, XOUT</td>
<td>These are the main clock connections. A crystal is connected across these pins to form a crystal oscillator, or XIN is connected to an external timing source.</td>
</tr>
</tbody>
</table>

### Programming the 16550

Programming the 16550 is simple, although it may be slightly more involved when compared to some of the other programmable interfaces described in this chapter. Programming is a two-part process that includes the initialization dialog and operational dialog.

In the personal computer, which uses the 16550 or its programming equivalent, the I/O port addresses are decoded at 3F8H through 3FFH for COM port 0 and 2F8H through 2FFH for COM port 2. Although the examples in this section of the chapter are not written specifically for the personal computer, they can be adapted by changing the port numbers to control the COM ports on the PC.
CHAPTER 11

**FIGURE 11–44** The contents of the 16550 line control register.

- **Initializing the 16550.** Initialization dialog, which occurs after a hardware or software reset, consists of two parts: programming the line control register and the baud rate generator. The line control register selects the number of data bits, number of stop bits, and parity (whether it’s even or odd, or if parity is sent as a 1 or a 0). The baud rate generator is programmed with a divisor that determines the baud rate of the transmitter section.

Figure 11–44 illustrates the line control register. The line control register is programmed by outputting information to I/O port 011 (A₂, A₁, A₀). The rightmost two bits of the line control register select the number of transmitted data bits (5, 6, 7, or 8). The number of stop bits is selected by S in the line control register. If S = 0, one stop bit is used; if S = 1, 1.5 or 2 stop bits are used for five data bits, and two stop bits are used with six, seven, or eight data bits.

The next three bits are used together to send even or odd parity, to send no parity, or to send a 1 or a 0 in the parity bit position. To send even or odd parity, the ST (stick) bit must be placed at a logic 0 level, and parity enable must be a logic 1. The value of the parity bit then determines even or odd parity. To send no parity (common in Internet connections), ST = 0 as well as the parity enable bit. This sends and receives data without parity. Finally, if a 1 or a 0 must be sent and received in the parity bit position for all data, ST = 1 with a 1 in parity enable. To send a 1 in the parity bit position, place a 0 in the parity bit; to send a 0, place a 1 in the parity bit. (See Table 11–6 for the operation of the parity and stick bits.)

**TABLE 11–6** The operation of the ST and parity bits.

<table>
<thead>
<tr>
<th>ST</th>
<th>P</th>
<th>PE</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>No parity</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Odd parity</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>No parity</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Even parity</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Undefined</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Send/receive 1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Undefined</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Send/receive 0</td>
</tr>
</tbody>
</table>
The remaining bits in the line control register are used to send a break and to select programming for the baud rate divisor. If bit position 6 of the line control register is a logic 1, a break is transmitted. As long as this bit is a 1, the break is sent from the SOUT pin. A break, by definition, is at least two frames of logic 0 data. The software in the system is responsible for timing the transmission of the break. To end the break, bit position 6 or the line control register is returned to a logic 0 level. The baud rate divisor is only programmable when bit position 7 of the line control register is a logic 1.

**Programming the Baud Rate.** The baud rate generator is programmed at I/O addresses 000 and 001 (A₂, A₁, A₀). Port 000 is used to hold the least significant part of the 16-bit divisor and port 001 is used to hold the most significant part. The value used for the divisor depends on the external clock or crystal frequency. Table 11–7 illustrates common baud rates obtainable if an 18.432 MHz crystal is used as a timing source. It also shows the divisor values programmed into the baud rate generator to obtain these baud rates. The actual number programmed into the baud rate generator causes it to produce a clock that is 16 times the desired baud rate. For example, if 240 is programmed into the baud rate divisor, the baud rate is (18.432 MHz ÷ 16) × 240 = 4800 baud.

**Sample Initialization.** Suppose that an asynchronous system requires seven data bits, odd parity, a baud rate of 9600, and one stop bit. Example 11–24 lists a procedure that initializes the 16550 to function in this manner. Example 11–26 describes the function of the FIFO control register in the next few paragraphs.

**EXAMPLE 11–26**

;Initialization dialog for Figure 11–45
;Baud rate 9600, 7 data, odd parity, 1 stop

<table>
<thead>
<tr>
<th>Baud Rate</th>
<th>Divisor Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>110</td>
<td>10,473</td>
</tr>
<tr>
<td>300</td>
<td>3840</td>
</tr>
<tr>
<td>1200</td>
<td>920</td>
</tr>
<tr>
<td>2400</td>
<td>480</td>
</tr>
<tr>
<td>4800</td>
<td>240</td>
</tr>
<tr>
<td>9600</td>
<td>120</td>
</tr>
<tr>
<td>19,200</td>
<td>60</td>
</tr>
<tr>
<td>38,400</td>
<td>30</td>
</tr>
<tr>
<td>57,600</td>
<td>20</td>
</tr>
<tr>
<td>115,200</td>
<td>10</td>
</tr>
</tbody>
</table>

TABLE 11–7 The divisor used with the baud rate generator for an 18.432 MHz crystal illustrating common baud rates.
After the line control register and baud rate divisor are programmed into the 16550, it is still not ready to function. After programming the line control register and baud rate, we still must program the FIFO control register, which is at port F2H in the circuit of Figure 11–45.

Figure 11–46 illustrates the FIFO control register for the 16550. This register enables the transmitter and receiver (bit 0 = 1), and clears the transmitter and receiver FIFOs. It also provides control for the 16550 interrupts, which are discussed in Chapter 12. Notice that the last section of Example 11–26 places a 7 into the FIFO control register. This enables the transmitter and receiver, and clears both FIFOs. The 16550 is now ready to operate, but without interrupts. Interrupts are automatically disabled when the MR (master reset) input is placed at a logic 1 by the system RESET signal.

**Sending Serial Data.** Before serial data can be sent or received through the 16550, we need to know the function of the line status register (see Figure 11–47). The line status register contains information about error conditions and the state of the transmitter and receiver. This register is tested before a byte is transmitted or can be received.

Suppose that a procedure (see Example 11–27) is written to transmit the contents of AH to the 16550 and out through its serial data pin (SOUT). The TH bit is polled by software to determine whether the transmitter is ready to receive data. This procedure uses the circuit of Figure 11–45.
### FIGURE 11–46 The FIFO control register of the 16550 UART.

<table>
<thead>
<tr>
<th>RT1</th>
<th>RT0</th>
<th>0</th>
<th>0</th>
<th>DMA</th>
<th>RXMT</th>
<th>RXVC</th>
<th>RST</th>
<th>RST</th>
<th>EN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>FIFO Enable</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0 = disable the FIFO</td>
<td>1 = enable the FIFO</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Receiver Reset</td>
<td>0 = no effect</td>
<td>1 = reset receiver FIFO</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Transmitter Reset</td>
<td>0 = no effect</td>
<td>1 = reset transmitter FIFO</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DMA mode control</td>
<td>0 = set to function as 16450 UART</td>
<td>1 = FIFO mode</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Receiver Trigger level</td>
<td>00 = 1 byte in FIFO</td>
<td>01 = 4 bytes in FIFO</td>
<td>10 = 8 bytes in FIFO</td>
<td>11 = 14 bytes in FIFO</td>
</tr>
</tbody>
</table>

### FIGURE 11–47 The contents of the line status register of the 16550 UART.

<table>
<thead>
<tr>
<th>ER</th>
<th>TE</th>
<th>TH</th>
<th>BI</th>
<th>FE</th>
<th>PE</th>
<th>OE</th>
<th>DR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Data Ready</td>
<td>0 = no data to read</td>
<td>1 = data in FIFO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Overrun Error</td>
<td>0 = no overrun error</td>
<td>1 = overrun error</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Parity Error</td>
<td>0 = no parity error</td>
<td>1 = parity error</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Framing Error</td>
<td>0 = no framing error</td>
<td>1 = framing error</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Break Indicator</td>
<td>0 = no break</td>
<td>1 = break being received</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Transmitter Holding Register</td>
<td>0 = wait for transmitter</td>
<td>1 = transmitter ready for data</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Transmitter Empty</td>
<td>0 = transmitter not empty</td>
<td>1 = transmitter empty</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Error</td>
<td>0 = no error</td>
<td>1 = at least one error in FIFO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### EXAMPLE 11–27

; A procedure that transmits AH via the 16650 UART

```
LSTAT EQU 0F5H
DATA  EQU 0F0H

SEND  PROC  NEAR  USES  AX
  .REPEAT
    IN  AL, LSTAT
    TEST AL, 20H
  .UNTIL  IZERO?
  MOV  AL, AH
  OUT  DATA, AL
  RET

SEND  ENDP
```
Receiving Serial Data. To read received information from the 16550, test the DR bit of the line status register. Example 11–28 lists a procedure that tests the DR bit to decide whether the 16550 has received any data. Upon the reception of data, the procedure tests for errors. If an error is detected, the procedure returns with AL equal to an ASCII ‘?’. If no error has occurred, then the procedure returns with AL equal to the received character.

EXAMPLE 11–28

; A procedure that receives data from the 16550 UART and returns it in AL.

LSTAT EQU 0F5H
DATA EQU 0F0H

REVC PROC NEAR

.REPEAT
   IN   AL,LSTAT ; test DR bit
   TEST AL,1
   .UNTIL !ZERO?
   TEST AL,0EH ; test for any error
   .IF ZERO? ; no error
      IN   AL,DATA
   .ELSE ; any error
      MOV  AL,‘?’
   .ENDIF
   RET

RECV ENDP

UART Errors. The types of errors detected by the 16550 are parity error, framing error, and overrun error. A parity error indicates that the received data contain the wrong parity. A framing error indicates that the start and stop bits are not in their proper places. An overrun error indicates that data have overrun the internal receiver FIFO buffer. These errors should not occur during normal operation. If a parity error occurs, it indicates that noise was encountered during reception. A framing error occurs if the receiver is receiving data at an incorrect baud rate. An overrun error occurs only if the software fails to read the data from the UART before the receiver FIFO is full. This example does not test the BI (break indicator bit) for a break condition. Note that a break is two consecutive frames of logic 0s on the SIN pin of the UART. The remaining registers, which are used for interrupt control and modem control, are developed in Chapter 12.

11–6 ANALOG-TO-DIGITAL (ADC) AND DIGITAL-TO-ANALOG (DAC) CONVERTERS

Analog-to-digital (ADC) and digital-to-analog (DAC) converters are used to interface the microprocessor to the analog world. Many events that are monitored and controlled by the microprocessor are analog events. These can range from monitoring all forms of events, even speech, to controlling motors and like devices. In order to interface the microprocessor to these events, we must have an understanding of the interface and control of the ADC and DAC, which convert between analog and digital data.

The DAC0830 Digital-to-Analog Converter

A fairly common and low-cost digital-to-analog converter is the DAC0830 (a product of National Semiconductor Corporation). This device is an 8-bit converter that transforms an 8-bit binary number into an analog voltage. Other converters are available that convert from 10-, 12-,...
or 16-bit binary numbers into analog voltages. The number of voltage steps generated by the converter is equal to the number of binary input combinations. Therefore, an 8-bit converter generates 256 different voltage levels, a 10-bit converter generates 1024 levels, and so forth. The DAC0830 is a medium-speed converter that transforms a digital input to an analog output in approximately 1.0 µs.

Figure 11–48 illustrates the pin-out of the DAC0830. This device has a set of eight data bus connections for the application of the digital input code, and a pair of analog outputs labeled IOUT1 and IOUT2 that are designed as inputs to an external operational amplifier. Because this is an 8-bit converter, its output step voltage is defined as $-\frac{V_{REF}}{255}$. For example, if the reference voltage is $-5.0$ V, its output step voltage is $+.0196$ V. Note that the output voltage is the opposite polarity of the reference voltage. If an input of $1001 \ 0010_2$ is applied to the device, the output voltage will be the step voltage times $1001 \ 0010_2$, or, in this case, $+2.862$ V. By changing the reference voltage to $-5.1$ V, the step voltage becomes $+.02$ V. The step voltage is also often called the resolution of the converter.

**Internal Structure of the DAC0830.** Figure 11–49 illustrates the internal structure of the DAC0830. Notice that this device contains two internal registers. The first is a holding register,
and the second connects to the R–2R internal ladder converter. The two latches allow one byte to be held while another is converted. In many cases, we disable the first latch and only use the second for entering data into the converter. This is accomplished by connecting a logic 1 to ILE and a logic 0 to CS (chip select).

Both latches within the DAC0830 are transparent latches. That is, when the G input to the latch is a logic 1, data pass through the latch, but when the G input becomes a logic 0, data are latched or held. The converter has a reference input pin (V_{REF}) that establishes the full-scale output voltage. If $-10\ V$ is placed on V_{REF}, the full-scale ($11111111_2$) output voltage is $+10\ V$. The output of the R–2R ladder within the converter appears at IOUT1 and IOUT2. These outputs are designed to be applied to an operational amplifier such as a 741 or similar device.

**Connecting the DAC0830 to the Microprocessor.** The DAC0830 is connected to the microprocessor as illustrated in Figure 11–50. Here, a PLD is used to decode the DAC0830 at 8-bit I/O port address 20H. Whenever an OUT 20H,AL instruction is executed, the contents of data bus connections AD_0–AD_7 are passed to the converter within the DAC0830. The 741 operational amplifier, along with the $-12\ V$ zener reference voltage, causes the full-scale output voltage to equal $+12\ V$. The output of the operational amplifier feeds a driver that powers a 12 V DC motor. This driver is a Darlington amplifier for large motors. This example shows the converter driving a motor, but other devices could be used as outputs.

**The ADC080X Analog-to-Digital Converter**

A common, low-cost ADC is the ADC080X, which belongs to a family of converters that are all identical, except for accuracy. This device is compatible with a wide range of microprocessors such as the Intel family. Although there are faster ADCs available and some have more resolution than 8 bits, this device is ideal for many applications that do not require a high degree of accuracy. The ADC080X requires up to 100\ \mu s to convert an analog input voltage into a digital output code.

Figure 11–51 shows the pin-out of the ADC0804 converter (a product of National Semiconductor Corporation). To operate the converter, the WR pin is pulsed with CS grounded to start the conversion process. Because this converter requires a considerable amount of time for the conversion, a pin labeled INTR signals the end of the conversion. Refer to Figure 11–52 for
a timing diagram that shows the interaction of the control signals. As can be seen, we start the converter with the WR pulse, we wait for INTR to return to a logic 0 level, and then we read the data from the converter. If a time delay is used that allows at least 100 µs of time, then we don’t need to test the INTR pin. Another option is to connect the INTR pin to an interrupt input, so that when the conversion is complete, an interrupt occurs.

**The Analog Input Signal.** Before the ADC0804 can be connected to the microprocessor, its analog inputs must be understood. There are two analog inputs to the ADC0804: VIN(+) and VIN(−). These inputs are connected to an internal operational amplifier and are differential inputs, as shown in Figure 11–53. The differential inputs are summed by the operational amplifier to

![Diagram](image-url)
produce a signal for the internal analog-to-digital converter. Figure 11–53 shows a few ways to use these differential inputs. The first way (see Figure 11–53a) uses a single input that can vary between 0 V and +5.0 V. The second way (see Figure 11–53b) shows a variable voltage applied to the VIN(−) pin, so the zero reference for VIN(+) can be adjusted.

Generating the Clock Signal. The ADC0804 requires a clock source for operation. The clock can be an external clock applied to the CLK IN pin or it can be generated with an RC circuit. The permissible range of clock frequencies is between 100 KHz and 1460 KHz. It is desirable to use a frequency that is as close as possible to 1460 KHz, so conversion time is kept to a minimum.

If the clock is generated with an RC circuit, we use the CLK IN and CLK R pins connected to an RC circuit, as illustrated in Figure 11–54. When this connection is in use, the clock frequency is calculated by the following equation:

\[ F_{\text{clk}} = \frac{1}{1.1RC} \]

Connecting the ADC0804 to the Microprocessor. The ADC0804 is interfaced to the 8086 microprocessor, as illustrated in Figure 11–55. Note that the V_REF signal is not attached to anything, which is normal. Suppose that the ADC0804 is decoded at 8-bit I/O port address 40H for the data and port address 42H for the INTR signal, and a procedure is required to start and read the data from the ADC. This procedure is listed in Example 11–29. Notice that the INTR bit is polled and if it becomes a logic 0, the procedure ends with AL, containing the converted digital code.
EXAMPLE 11–29
ADC PROC NEAR
    OUT 40H,AL
    .REPEAT
    IN  AL,42H
    TEST AL,80H
    .UNTIL ZERO?
    IN  AL,40H
    RET
ADC ENDP

Using the ADC0804 and the DAC0830

This section of the text illustrates an example that uses both the ADC0804 and the DAC0830 to capture and replay audio signals or speech. In the past, we often used a speech synthesizer to generate speech, but the quality of the speech was poor. For human quality speech, we can use the ADC0804 to capture an audio signal and store it in memory for later playback through the DAC0830.

Figure 11–56 illustrates the circuitry required to connect the ADC0804 at I/O ports 0700H and 0702H. The DAC0830 is interfaced at I/O port 704H. These I/O ports are in the low bank of

FIGURE 11–56 A circuit that stores speech and plays it back through the speaker.
a 16-bit microprocessor such as the 8086 or 80386SX. The software used to run these converters appears in Example 11–30. This software reads a 1-second burst of speech and then plays it back 10 times. One procedure reads the speech called READS and the other, called PLAYS, plays it back. The speech is sampled and stored in a section of memory called WORDS. The sample rate is chosen at 2048 samples per second, which renders acceptable-sounding speech.

**EXAMPLE 11–30**

;Software that records a second of speech and plays it back 10 times.

;Assumes the clock frequency is 20 MHz on an 80386EX microprocessor

```assembly
READS PROC NEAR USES ECX DX
    MOV ECX,2048 ;count = 2048
    MOV DX,700H ;address port 700H
    .REPEAT
        OUT DX,AL ;start conversion
        ADD DX,2 ;address status port
    .REPEAT ;wait for converter
        IN AL,DX
        TEST AL,80H
        .UNTIL ZERO?
        SUB DX,2 ;address data port
        IN AL,DX ;get data
        MOV WORDS[ECX-1] ;address data port
        CALL DELAY ;wait for 1/2048 sec
    .UNTIL ECXZ
    RET

READS ENDP

PLAYS PROC NEAR USES DX ECX
    MOV ECX,2048 ;count = 2048
    MOV DX,704H ;address DAC
    .REPEAT
        MOV AL,WORDS[EAX-1] ;send byte to DAC
        CALL DELAY ;wait for 1/2048 sec
    .UNTIL ECXZ
    RET

PLAYS ENDP

DELAY PROC NEAR USES CX
    MOV CX,888
    .REPEAT
    .UNTIL ECXZ
    RET

DELAY ENDP
```

11–7 **SUMMARY**

1. The 8086–Core2 microprocessors have two basic types of I/O instructions: IN and OUT. The IN instruction inputs data from an external I/O device into either the AL (8-bit) or AX (16-bit) register. The IN instruction is available as a fixed port instruction, a variable port instruction, or a string instruction (80286–Pentium 4) INSB or INSW. The OUT instruction
outputs data from AL or AX to an external I/O device and is available as a fixed, variable, or string instruction OUTSB or OUTSW. The fixed port instruction uses an 8-bit I/O port address, while the variable and string I/O instructions use a 16-bit port number found in the DX register.

2. Isolated I/O, sometimes called direct I/O, uses a separate map for the I/O space, freeing the entire memory for use by the program. Isolated I/O uses the IN and OUT instructions to transfer data between the I/O device and the microprocessor. The control structure of the isolated I/O map uses IORC (I/O read control) and IOWC (I/O write control), plus the bank selection signals BHE and BLE (A0 on the 8086 and 80286), to effect the I/O transfer. The early 8086/8088 used the M/IO (I/O/M) signal with RD and WR to generate the I/O control signals.

3. Memory-mapped I/O uses a portion of the memory space for I/O transfers. This reduces the amount of memory available, but it negates the need to use the IORC and IOWC signals for I/O transfers. In addition, any instruction that addresses a memory location using any addressing mode can be used to transfer data between the microprocessor and the I/O device using memory-mapped I/O.

4. All input devices are buffered so that the I/O data are connected only to the data bus during the execution of the IN instruction. The buffer is either built into a programmable peripheral or located separately.

5. All output devices use a latch to capture output data during the execution of the OUT instruction. This is necessary because data appear on the data bus for less than 100 ns for an OUT instruction, and most output devices require the data for a longer time. In many cases, the latch is built into the peripheral.

6. Handshaking or polling is the act of two independent devices synchronizing with a few control lines. For example, the computer asks a printer if it is busy by inputting the BUSY signal from the printer. If it isn’t busy, the computer outputs data to the printer and informs the printer that data are available with a data strobe (DS) signal. This communication between the computer and the printer is a handshake or a poll.

7. Interfaces are required for most switch-based input devices and for most output devices that are not TTL-compatible.

8. The I/O port number appears on address bus connections A7–A0 for a fixed port I/O instruction and on A15–A0 for a variable port I/O instruction (note that A15–A8 contains zeros for an 8-bit port). In both cases, address bits above A15 are undefined.

9. Because the 8086/80286/80386SX microprocessors contain a 16-bit data bus and the I/O addresses reference byte-sized I/O locations, the I/O space is also organized in banks, as is the memory system. In order to interface an 8-bit I/O device to the 16-bit data bus, we often require separate write strobes (an upper and a lower) for I/O write operations. Likewise, the 80486 and Pentium–Core2 also have I/O arranged in banks.

10. The I/O port decoder is much like the memory address decoder, except instead of decoding the entire address, the I/O port decoder decodes only a 16-bit address for variable port instructions and often an 8-bit port number for fixed I/O instructions.

11. The 82C55 is a programmable peripheral interface (PIA) that has 24 I/O pins that are programmable in two groups of 12 pins each (group A and group B). The 82C55 operates in three modes: simple I/O (mode 0), strobed I/O (mode 1), and bidirectional I/O (mode 2). When the 82C55 is interfaced to the 8086 operating at 8 MHz, we insert two wait states because the speed of the microprocessor is faster than the 82C55 can handle.

12. The LCD display device requires a fair amount of software, but it displays ASCII-coded information.

13. The 8254 is a programmable interval timer that contains three 16-bit counters that count in binary or binary-coded decimal (BCD). Each counter is independent and operates in six different modes: (1) events counter, (2) retriggerable, monostable multivibrator, (3) pulse...
generator, (4) square-wave generator, (5) software-triggered pulse generator, and (6) hardware-triggered pulse generator.

14. The 16550 is a programmable communications interface, capable of receiving and transmitting asynchronous serial data.

15. The DAC0830 is an 8-bit digital-to-analog converter that converts a digital signal to an analog voltage within 1.0 µs.

16. The ADC0804 is an 8-bit analog-to-digital converter that converts an analog signal into a digital signal within 100 µs.

11–8 QUESTIONS AND PROBLEMS

1. Explain which way the data flow for an IN and an OUT instruction.

2. Where is the I/O port number stored for a fixed I/O instruction?

3. Where is the I/O port number stored for a variable I/O instruction?

4. Where is the I/O port number stored for a string I/O instruction?

5. To which register are data input by the 16-bit IN instruction?

6. Describe the operation of the OUTSB instruction.

7. Describe the operation of the INSW instruction.

8. Contrast a memory-mapped I/O system with an isolated I/O system.

9. What is the basic input interface?

10. What is the basic output interface?

11. Explain the term handshaking as it applies to computer I/O systems.

12. An even-number I/O port address is found in the ___________ I/O bank in the 8086 microprocessor.

13. In the Pentium 4, what bank contains I/O port number 000AH?

14. How many I/O banks are found in the Pentium 4 or Core2 microprocessor?

15. Show the circuitry that generates the upper and lower I/O write strobes.

16. What is the purpose of a contact bounce eliminator?

17. Develop an interface to correctly drive a relay. The relay is 12 V and requires a coil current of 150 mA.

18. Develop a relay coil driver that can control a 5.0 V relay that requires 60 mA of coil current.

19. Develop an I/O port decoder, using a 74ALS138, that generates low-bank I/O strobes, for a 16-bit microprocessor, for the following 8-bit I/O port addresses: 10H, 12H, 14H, 16H, 18H, 1AH, 1CH, and 1EH.

20. Develop an I/O port decoder, using a 74ALS138, that generates high-bank I/O strobes, for a 16-bit microprocessor, for the following 8-bit I/O port addresses: 11H, 13H, 15H, 17H, 19H, 1BH, 1DH, and 1FH.

21. Develop an I/O port decoder, using a PLD, that generates 16-bit I/O strobes for the following 16-bit I/O port addresses: 1000H–1001H, 1002H–103H, 1004H–1005H, 1006H–1007H, 1008H–1009H, 100AH–100BH, 100CH–100DH, and 100EH–100FH.

22. Develop an I/O port decoder, using a PLD, that generates the following low-bank I/O strobes: 00A8H, 00B6H, and 00EEH.

23. Develop an I/O port decoder, using a PLD, that generates the following high-bank I/O strobes: 300DH, 300BH, 1005H, and 1007H.

24. Why are both BHE and BLE (A0) ignored in a 16-bit port address decoder?

25. An 8-bit I/O device, located at I/O port address 0010H, is connected to which data bus connections in a Pentium 4?

26. An 8-bit I/O device, located at I/O port address 100DH, is connected to which data bus connections in a Core2 microprocessor?
27. The 82C55 has how many programmable I/O pin connections?
28. List the pins that belong to group A and to group B in the 82C55.
29. Which two 82C55 pins accomplish internal I/O port address selection?
30. The RD connection on the 82C55 is attached to which 8086 system control bus connection?
31. Using a PLD, interface an 82C55 to the 8086 microprocessor so that it functions at I/O locations 0380H, 0382H, 0384H, and 0386H.
32. When the 82C55 is reset, its I/O ports are all initialized as __________.
33. What three modes of operation are available to the 82C55?
34. What is the purpose of the STB signal in strobed input operation of the 82C55?
35. Develop a time delay procedure for the 2.0 GHz Pentium 4 that waits for 80 µs.
36. Develop a time delay procedure for the 3.0 GHz Pentium 4 that waits for 12 ms.
37. Explain the operation of a simple four-coil stepper motor.
38. What sets the IBF pin in strobed input operation of the 82C55?
39. Write the software required to place a logic 1 on the PC7 pin of the 82C55 during strobed input operation.
40. How is the interrupt request pin (INTR) enabled in the strobed input mode of operation of the 82C55?
41. In strobed output operation of the 82C55, what is the purpose of the ACK signal?
42. What clears the OFB signal in strobed output operation of the 82C55?
43. Write the software required to decide whether PC4 is a logic 1 when the 82C55 is operated in the strobed output mode.
44. Which group of pins is used during bidirectional operation of the 82C55?
45. Which pins are general-purpose I/O pins during mode 2 operation of the 82C55?
46. Describe how the display is cleared in the LCD display.
47. How is a display position selected in the LCD display?
48. Write a short procedure that places an ASCII null string in display position 6 on the LCD display.
49. How is the busy flag tested in the LCD display?
50. What changes must be made to Figure 11–25 so that it functions with a keyboard matrix that contains three rows and five columns?
51. What time is usually used to debounce a keyboard?
52. Develop the interface to a three- by four-key telephone-style keypad. You will need to use a lookup table to convert to the proper key code.
53. The 8254 interval timer functions from DC to __________ Hz.
54. Each counter in the 8254 functions in how many different modes?
55. Interface an 8254 to function at I/O port addresses XX10H, XX12H, XX14H, and XX16H.
56. Write the software that programs counter 2 to generate an 80 KHz square wave if the CLK input to counter 2 is 8 MHz.
57. What number is programmed in an 8254 counter to count 300 events?
58. If a 16-bit count is programmed into the 8254, which byte of the count is programmed first?
59. Explain how the read-back control word functions in the 8254.
60. Program counter 1 of the 8254 so that it generates a continuous series of pulses that have a high time of 100 µs and a low time of 1 µs. Make sure to indicate the CLK frequency required for this task.
61. Why does a 50% duty cycle cause the motor to stand still in the motor speed and direction control circuit presented in this chapter?
62. What is asynchronous serial data?
63. What is baud rate?
64. Program the 16550 for operation using six data bits, even parity, one stop bit, and a baud rate of 19,200 using a 18.432 MHz clock. (Assume that the I/O ports are numbered 20H and 22H.)
65. If the 16550 is to generate a serial signal at a baud rate of 2400 baud and the baud rate divisor is programmed for 16, what is the frequency of the signal?
66. Describe the following terms: simplex, half-duplex, and full-duplex.
67. How is the 16550 reset?
68. Write a procedure for the 16550 that transmits 16 bytes from a small buffer in the data segment address (DS is loaded externally) by SI (SI is loaded externally).
69. The DAC0830 converts an 8-bit digital input to an analog output in approximately ___________.
70. What is the step voltage at the output of the DAC0830 if the reference voltage is −2.55 V?
71. Interface a DAC0830 to the 8086 so that it operates at I/O port 400H.
72. Develop a program for the interface of question 71 so the DAC0830 generates a triangular voltage wave-form. The frequency of this wave-form must be approximately 100 Hz.
73. The ADC080X requires approximately ____________ to convert an analog voltage into a digital code.
74. What is the purpose of the INTR pin on the ADC080X?
75. The WR pin on the ADC080X is used for what purpose?
76. Interface an ADC080X at I/O port 0260H for data and 0270H to test the INTR pin.
77. Develop a program for the ADC080X in question 76 so that it reads an input voltage once per 100 ms and stores the results in a memory array that is 100H bytes long.
78. Rewrite Example 11–29 using C++ with inline assembly code.