9 I/O Interfaces

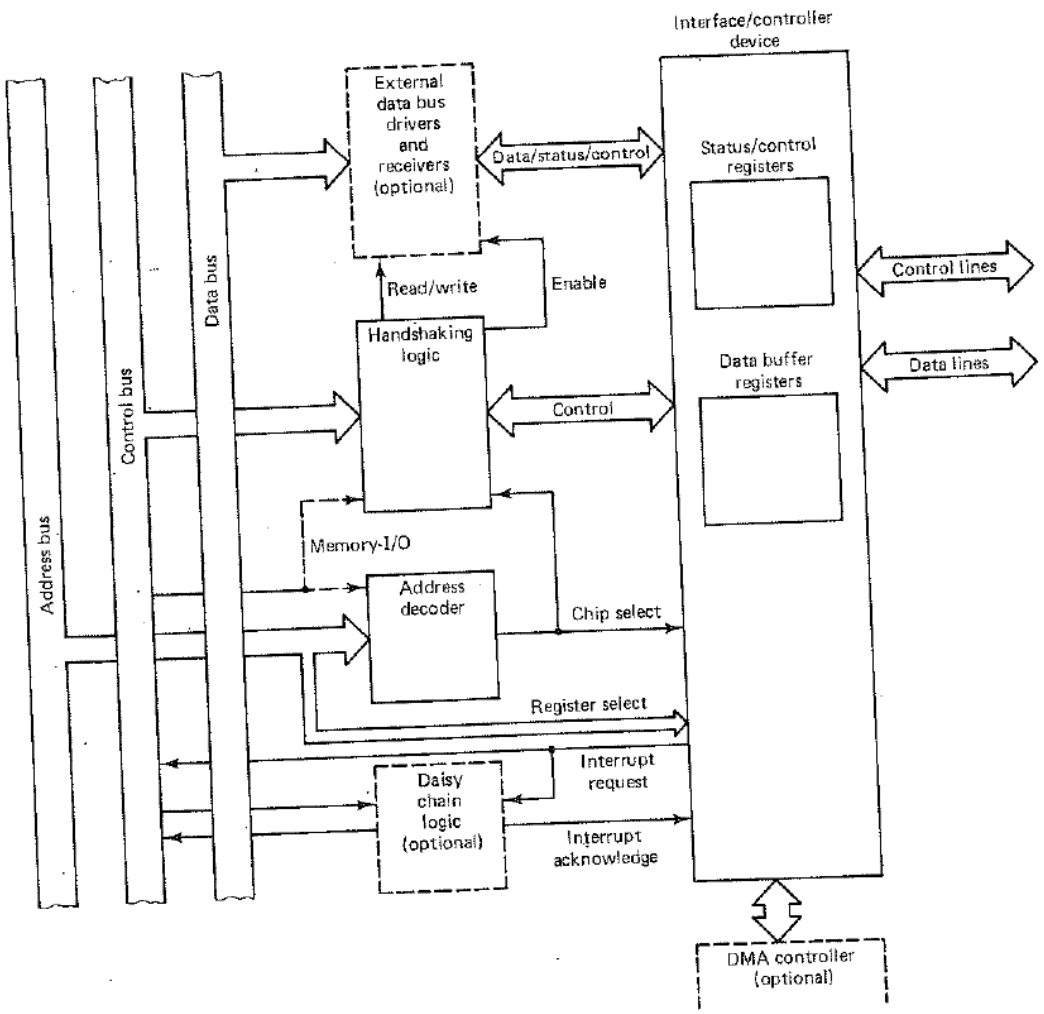
The I/O and memory interfaces are the counterparts to the bus control logic. What goes between the bus control logic and the interfaces is simply the conductors in the bus; therefore, the interfaces must be designed to accept and send signals that are compatible with the bus control logic and its timing. Although there are similarities in memory and I/O interfaces, there are also significant differences. This chapter will concentrate on I/O interfaces and Chap. 10 will study memories and their interfaces.

An I/O interface must be able to:

1. Interpret the address and memory-I/O select signals to determine whether or not it is being referenced and, if so, determine which of its registers is being accessed.
2. Determine whether an input or output is being conducted and accept output data or control information from the bus or place input data or status information on the bus.
3. Input data from or output data to the associated I/O device and convert the data from parallel to the format acceptable to the I/O device, or vice versa.
4. Send a ready signal when data have been accepted from or placed on the data bus, thus informing the processor that a transfer has been completed.
5. Send interrupt requests and, if there is no interrupt priority management in the bus control logic, receive interrupt acknowledgments and send an interrupt type.
6. Receive a reset signal and reinitialize itself and, perhaps, its associated device.

Figure 9-1 contains a block diagram of a typical I/O interface. The function of an I/O interface is essentially to translate the signals between the system bus and the I/O device and provide the buffers needed to satisfy the two sets of timing constraints. Most of the work done by the interface is accomplished by the large block on the right side of the figure. Most often this block is implemented by a single IC device, but its functions could be scattered across several devices. Clearly, its functions vary radically, depending on the I/O device with which it is designed to communicate.

Figure 9-1 Typical block diagram of an I/O interface.



An interface can be divided into two parts, a part that interfaces to the I/O device and a part that interfaces to the system bus. Although little can be said about the I/O device side of the interface without knowing a lot about the device, the bus sides of all interfaces in a given system are very similar because they connect to the same bus. To support the main interface logic there must be data bus drivers and receivers, logic for translating the interface control signals into the proper handshaking signals, and logic for decoding the addresses that appear on the bus. In an 8086/8088 system, 8286 transceivers could drive the data bus, just as they are used to drive the bus at its bus control logic end. However, the main interface devices may have built-in drivers and receivers that are sufficient for small, single-board systems.

The handshaking logic cannot be designed until the control signals needed by the main interface device are known, and these signals may vary from one interface to the next. Typically, this logic must accept read/write signals for determining the data direction and output the $\overline{OE}$ and T signals required by the 8286s. In a maximum mode 8086/8088 system it would receive the $\overline{IOWC}$ (or $\overline{AIOWC}$) and $\overline{IORC}$ signals from the 8288 bus controller and in a minimum mode system it would receive the $\overline{RD}$, $\overline{WR}$, and M/$\overline{IO}$ (or IO/$\overline{M}$) signals. The interrupt request, ready, and reset lines would also pass through this logic. In some cases, the bus control lines may pass through the handshaking logic unaltered (i.e., be connected directly to the main interface device).

The address decoder must receive the address and, perhaps, a bit indicating whether the address is in the I/O address space or the memory address space. In a minimum mode system this bit could be taken from the M/$\overline{IO}$ (or IO/$\overline{M}$) line, but in a maximum mode system the memory-I/O determination is obtained directly from the $\overline{IOWC}$ and $\overline{IORC}$ lines. If the decoder determines that its interface is being referenced, then the decoder must send signals to the main device indicating that it has been selected and which register is being accessed. The bits designating the register may be the low-order address bits, but are often generated inside the interface device from the read/write control signals as well as the address signals. For example, if there are two registers A and B that can be read from and two registers C and D that can be written into, then the read and write signals and bit 0 of the address bus could be used to specify the register as follows:

Write	Read	Address Bit 0	Register Being Accessed
0	1	0	A
0	1	1	B
1	0	0	C
1	0	1	D

If a daisy chain is included in the system instead of an interrupt priority management device, then each interface must contain daisy chain logic, such as that shown in Fig. 6-14b, and must include logic to generate the interrupt type. Also, the interface may be associated with a DMA controller.

Many interfaces are designed to detect at least two kinds of errors. Because the lines connecting an interface to its device are almost always subject to noise,

parity bits are normally appended to the information bytes as they are transmitted. If even parity is used the parity bit is set so that the total number of 1s, including the parity bit, is even. For odd parity the total number of 1s is odd. As these bytes are received the parity is checked and, if it is in error, a certain status bit is set in a status register. Some interfaces are also designed to check error detection redundancy bytes that are placed after blocks of data. The other type of error most interfaces can detect is known as an *overrun error*. As we have seen, when a computer inputs data it brings the data in from a data-in buffer register. If, for some reason, the contents of this register are replaced by new data before they are input by the computer, an overrun error occurs. Such an error also happens when data are put in a data-out buffer before the current contents of the register have been output. As with parity errors, overrun errors cause a certain status bit to be set.

Interfaces can be categorized according to the way in which they communicate with their I/O devices. The first section in this chapter concerns interfaces for those I/O devices which send and receive their information serially, and includes a discussion of long-distance communication. Section 9-2 considers general interfaces that transfer data to and from their devices in a parallel format. The third section considers interfaces that can provide programmable timing of either external or internal events or can count external events, and the fourth section discusses designing an interface into a unit with a keyboard and display. The next two sections examine high-speed communication by studying DMA and diskette controllers.

This chapter contains several examples that are centered on Intel interface devices. These devices have only eight data pins and, because the 8088 has only an 8-bit data bus, it is less confusing to assume that the microprocessor in these examples is an 8088. They can be connected to a 16-bit 8086 bus, but, because an 8086 expects odd-addressed bytes to be transferred over the high-order 8 bits of the bus, certain modifications and/or additions to the interface designs are needed. Therefore, all of the designs in all but the last section assume an 8-bit 8088 bus, and the last section in the chapter discusses the changes needed to connect these devices to a 16-bit 8086 bus. Also, the Intel examples in all but the last section assume a minimum mode system, and the last section considers the modifications required for maximum mode operation.

9-1 SERIAL COMMUNICATION INTERFACES

Many I/O devices transfer information to or from a computer serially, i.e., one bit at a time over a single conductor pair or communication channel, with each bit occupying an interval of time having a specified length. There are several serial interface devices and typically they are constructed as shown in Fig. 9-2. The status register would contain error and other information concerning the state of the current transmission, and the control register is for holding the information that determines the operating mode of the interface. The data-in buffer is paired with a serial input/parallel output shift register. During an input operation, the bits are

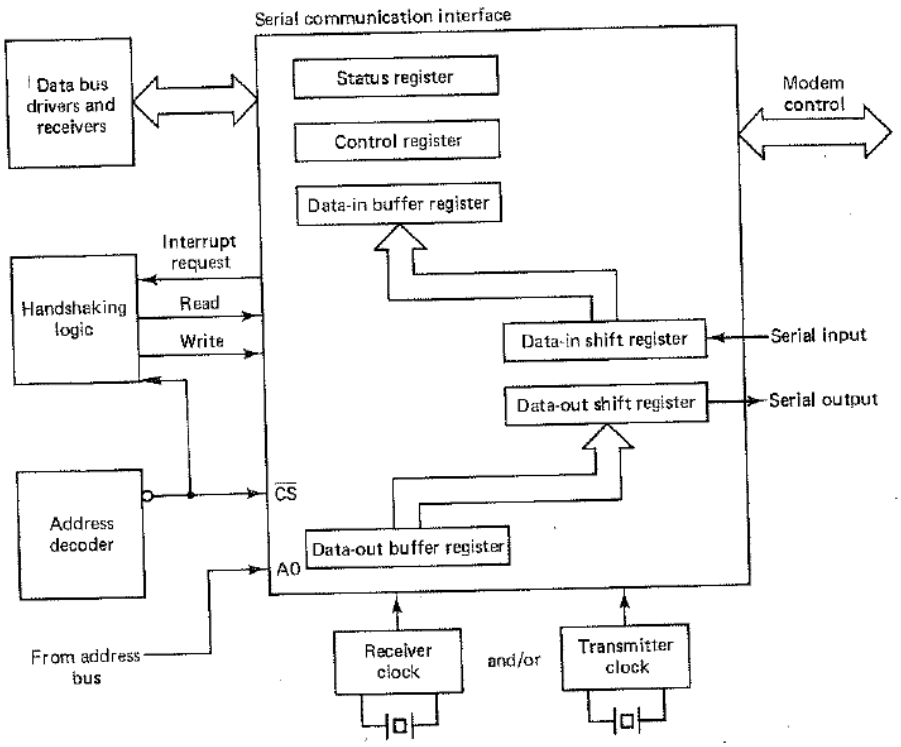


Figure 9-2 Serial interface.

brought into the shift register one at a time and, after a character has been received, the information is transferred to the data-in buffer register, where it waits to be taken by the CPU. (Although a single datum is not necessarily an alphanumerically coded character, it normally is and, therefore, it will be referred to as a character.) Similarly, the data-out buffer is associated with a parallel input/serial output shift register. An output is performed by sending data to the data-out buffer, transferring it to the shift register, and then shifting it onto the serial output line.

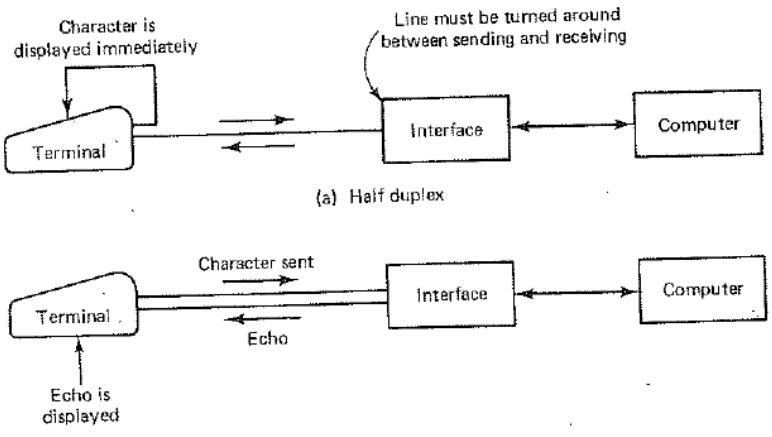
Although there are several ways in which the four port registers could be addressed, it has been assumed that the status register can only be read from and the control register can only be written into. Therefore, an active signal on the read line would indicate either the status or data-in buffer register and A0 could be used to distinguish between these two possibilities. Likewise, the write and A0 signals can be used to designate one of the other two registers.

The interface shown in Fig. 9-2 has separate lines for sending and receiving information. When different lines are used for the two signal directions the communication system is said to be *full duplex*. Such a system can transmit and receive at the same time. Terminals and other serial devices that are connected to a computer through a full duplex system normally send each character to the com-

puter without displaying it. The computer immediately sends the character back, or *echos* the character, and the terminal then displays it. There is no loss of time because the echoed characters can be sent back at the same time new characters are being input by the computer. The advantage of this scheme is that the user gets visual verification not only that he has struck the correct key, but also that the correct character was received by the computer. The alternative, called *half duplex*, is to use the same line for both inputting and outputting. In this case a character typed into a terminal is displayed at the same time it is sent to the computer, and echoing is not used. If the computer has been receiving characters and then wishes to send characters, or vice versa, the communication link must be *turned around*, a process that requires time. Although full duplex avoids turnarounds and provides echoing, it does require an extra line. The two modes of communication are illustrated in Fig. 9-3. Many terminals and interfaces provide both turnaround capability for half duplex transmissions and separate pins for full duplex transmissions. For a device such as a printer that requires only one-way communication, clearly half duplex would be sufficient and turnarounds would not be needed.

There are two basic types of serial communication. There is *asynchronous serial communication* in which special bit patterns separate the characters, and *synchronous serial communication* which allows characters to be sent back to back but must include special "sync" characters at the beginning of each message and special "idle" characters in the data stream to fill up time when no information is being sent. An asynchronous transmission may include dead time of arbitrary lengths between the characters, while in a synchronous transmission the characters must be precisely spaced even though some of the characters may contain no information. Although both types may waste time sending useless bits, the maximum information rate of a synchronous line is higher than that of an asynchronous line with the same bit rate because the asynchronous transmission must include

Figure 9-3 Basic transmission modes.



extra bits with each character. On the other hand, the clocks at the opposing ends of an asynchronous transmission line do not need to have exactly the same frequency (as long as they are within permissible limits) because the special patterns allow for resynchronization at the beginning of each character. For a synchronous transmission the activity must be coordinated by a single clock since it is the clock that determines the position of each bit. This means that the clock timing must be transmitted as well as the data.

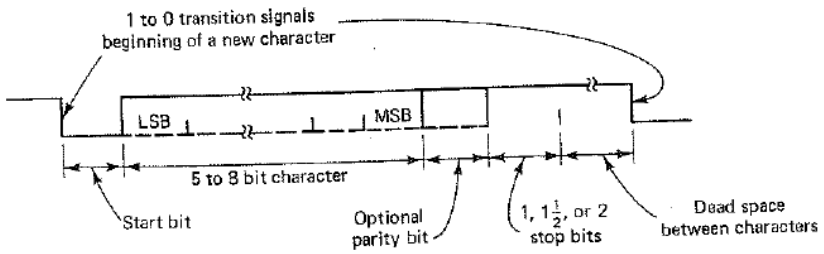
9-1-1 Asynchronous Communication

The format of an asynchronous character is shown in Fig. 9-4. A character contains several extra bits as well as the coded information. Before a character begins the line must be in its "1" state, often called its *mark* state. The transition from the mark state to the "0", or *space*, state marks the beginning of the character. The first bit is always 0 and is called the *start bit*. It is followed by from 5 to 8 information bits, the first of which is the LSB of the character. The information bits may or may not be followed by a parity bit and, if there is one, it may represent either even or odd parity. The last bits are 1s and are called *stop bits*. There may be 1, $1\frac{1}{2}$, or 2 stop bits.

Although the number of information bits, whether or not there is a parity bit, the type of parity, and the number of stop bits may be changed from one transmission (i.e., sequence of characters) to the next, these parameters are constant within a single transmission. On some interfaces these parameters can be programmed using the control registers; on others they are determined by switch settings within the interfaces. If these parameters are programmable, the interface would include a control register such as the one shown in Fig. 9-5. In this example bits 0 and 1 indicate the number of stop bits, bits 2 and 3 indicate the number of information bits, and bit 4 specifies whether or not parity bits are present. If bit 4 is 1, then bit 5 determines the type of parity.

At the transmitter end a clock is needed to determine the period of each bit by regulating the time between the shifts made by shift registers. After shifting out all of the bits the transmitter would continually output a mark state until another character is ready to be sent. At the receiver end a clock is also required for measuring the time between shifts so that the incoming signal is sampled at the correct times. Normally, the frequencies of these clocks are 16, 32, or 64 times the

Figure 9-4 Format of a standard asynchronous transmission.



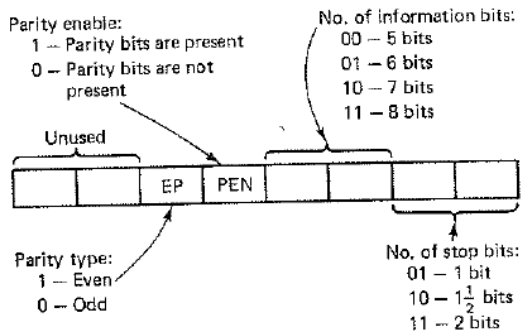


Figure 9-5 Typical control register for specifying the character format.

bit frequency. Assuming a factor of 16, after detecting the 1-to-0 transition at the beginning of a character the receiver would count eight clock pulses and sample the input. If it sees a 0, it assumes the transition did, in fact, initiate a start bit and was not due to noise. The receiver then samples the input at intervals of 16 clock pulses until all of the bits, including the stop bits, have been input, at which time it ceases its sampling until another 1-to-0 transition is detected. It is important to note that the start, stop, and parity bits are not sent to nor received from the CPU. During output, the transmitter inserts these bits in each character and, during input, the receiver deletes these bits from the received data.

This format allows the transmitter to insert intervals of indeterminate length between characters and the receiver to resynchronize itself at the beginning of each character. Without this resynchronization mechanism, the two clocks would eventually drift out of sync and the receiver would sample improperly. With resynchronization the receiver needs to assimilate the bit rate for only one character at a time. Incorrect sampling can occur only if the two clock frequencies are so different that a receiver shift is performed at the wrong time within the first few bits after a start bit. If this happens, the probability that at least one 0 will be detected at a time when a stop bit is supposed to be received becomes high. When a 0 is seen instead of a stop bit a *framing error* is said to occur. Therefore, most asynchronous serial interfaces are designed to detect three kinds of errors: parity errors, overrun errors, and framing errors.

The transmit and receive clocks that regulate the timing in the interface do not have to be the same and need not have the same frequency, but there are obvious advantages to using one clock to provide all of the necessary interface timing pulses. If the transmit and receive bit rates are different, then the electronics at the other end of the communication link must be able to handle two bit rates. The receiving rate for these electronics must match the interface transmit rate, and vice versa.

Devices that are capable of receiving and transmitting data in the format given in Fig. 9-4, converting them to and from parallel form, and detecting parity, overrun, and framing errors are called *Universal Asynchronous Receivers and Transmitters (UARTs)*. Many manufacturers have produced UARTs and their design

has been standardized—see McNamara [1]. The microprocessor manufacturers have tended to expand their designs to complete and versatile interfaces. In some cases the UART is only a small part of the interface, which may satisfy synchronous serial and parallel communication applications as well as asynchronous serial applications.

9-1-2 Synchronous Communications

A synchronously transmitted character also consists of from 5 to 8 bits optionally followed by a parity bit, but contains no start or stop bits. All characters have the same number of bits, say n, and the transmission time is divided into groups of n bits each. The same clock controls the sampling by the receiver as is used to generate the bits, thus guaranteeing the synchronization of the two processes. The transmitter must transmit a character during each n-bit interval. If a character is not available at the beginning of an interval, an *underrun* is said to occur and the transmitter will insert an idle character. Depending on the system, idle characters may be interpreted as errors, called *underrun errors*, by the receiver.

There is no problem in starting up or turning around an asynchronous system because the transmitter can simply put a mark on the line until the transmission is ready to begin. If noise causes an accidental 1-to-0 indication, the false signal is discovered when the first sample does not detect a start bit and the system will wait for another 1-to-0 transition. In a synchronous transmission the problem of starting up after a shutdown or other disruption is more difficult. All transmissions must begin with a series of *sync characters*, which should be selected so that confusion with the other characters is unlikely. Normally, the sync characters are the same as the idle characters, and in the ASCII code the sync character is 0010110.

The receiver, which must know the identity of the sync character, will examine each bit as it arrives, and when a sequence of bits exactly matches the bits in the sync character it will assume a transmission has begun. It may then treat the succeeding characters as information or may attempt also to match one or more of them with the sync character. Because noise may cause the false identification of a sync character most systems require that a transmission begin with a sequence of sync characters. The receiver will then not assume the beginning of a transmission until the proper number of sync characters have been input. It can be the responsibility of either the receiver or an input program to remove the unnecessary sync and idle characters.

As with the number of information bits and the existence and kind of parity, the number and identity of sync characters may be programmable. If this is so, then there would be special control registers for storing the sync character and the other formatting information.

9-1-3 Physical Communication Standards

As computer use grew, so did the need for equipment that would permit human-computer and computer-computer communication over both long and short dis-

tances. This meant the involvement of several parties in certain aspects of the computer industry. So that people in the telephone, communication equipment, and computer industries could converse with each other and build equipment that would function together, several definitions and standards have been established. These definitions and standards primarily affect three areas:

1. Transmission rates.
2. Electrical characteristics.
3. Line definitions and notations.

Transmission rates are measured in *bits per second (bps)* and in *baud*, which means the number of discrete conditions being transmitted per second. If there can be only one of two possible conditions at any point in time, then bps and baud are the same. This is almost always the case at the computer interface because it can normally input or output only 1s or 0s; therefore, this book will treat the two terms as being synonymous. However, it should be noted that many communication links may allow one of four or more conditions at a given time. For example, a phase-modulated signal may take on one of four phases, with each possible phase representing 2 bits, in which case the bps rate would be twice that of the baud rate.

The standard baud rates that are most often used are 110, 300, 600, 1200, 1800, 2400, 4800, 9600, and 19,200. Most CRT terminals are capable of handling any one of these rates up to 9600 baud, while printing terminals are limited by the speed of the print mechanism. Computer-driven typewriters operate at 110 baud and some dot-matrix printers can receive at up to 2400 baud. Most interfaces are such that the transmit and receive baud rates can be set separately, and frequently these rates are programmable.

As an example, consider an asynchronous transmission in which each character contains a start bit, 7 information bits, a parity bit, and 1 stop bit. If the baud rate of the line were 1200, then the maximum number of characters per second that could be transmitted would be $1200/10 = 120$. The maximum rate could be attained only when there is no dead time between characters. As a comparison, a synchronous line operating at 1200 baud and with no parity could transmit four sync characters and a 100-character message in

$$7(100 + 4)/1200 = 0.6067 \text{ second}$$

This would mean that up to $100/0.6067 = 165$ useful characters per second could be transmitted.

There are two organizations that are primarily responsible for establishing standards for electrical characteristics and line definitions. There is the Electrical Industry Association (EIA), which has set most of the standards used in the United States, and the Comité Consultatif International Téléphonique et Télégraphique (CCITT), which is part of the United Nations and is concerned with international standards. The standard that is of the most interest to us is the EIA RS-232-C

standard, which is essentially the same as the CCITT V.24 standard. It is the one that is concerned with sending serial bit streams between interfaces or terminals and communication equipment.

How far a bit stream output by an interface or terminal can be sent before it begins to seriously deteriorate depends on the baud rate and the electrical characteristics of the transmission line. A typical curve showing maximum distance versus baud rate for an unbalanced shielded pair with a 50-pF/ft capacitance is given in Fig. 9-6. It is evident that the maximum distance drops off rapidly for

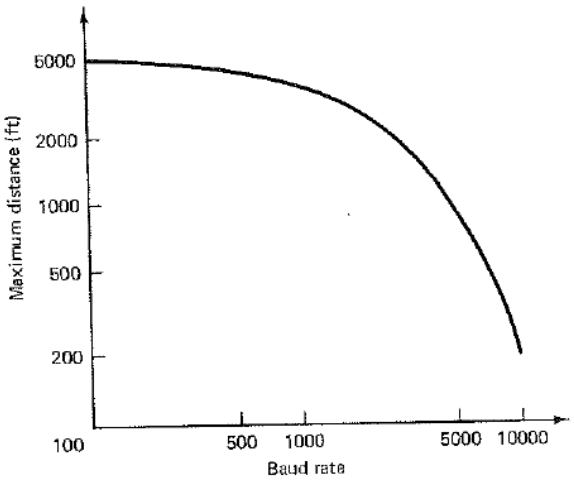
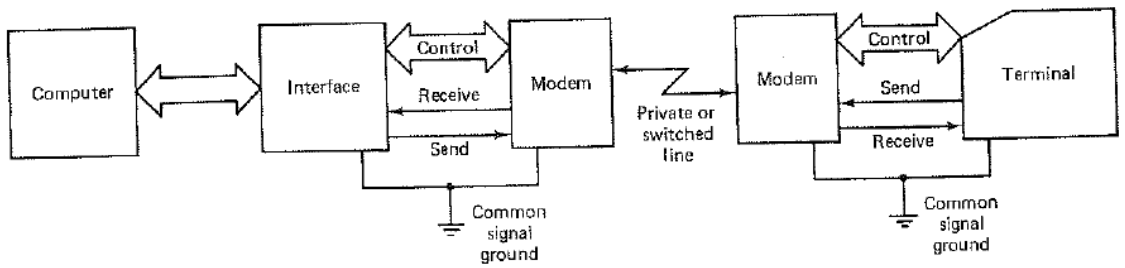


Figure 9-6 Maximum signal distance versus rate for EIA RS-232 voltage levels.

baud rates above 1000, with the maximum distance at 9600 baud being about 250 ft. For distances lying below the curve communication equipment is not required and a full duplex connection can consist of only three wires: a send line, a receive line, and a common signal ground.

For distances above the curve the transmission should be aided by inserting communication equipment as shown in Fig. 9-7. The usual configuration calls for

Figure 9-7 Long-distance serial communication.



identical circuits to be placed at the ends of the communication link for the purposes of modulating the bits going to the communication link and demodulating the bits coming from the link. These modulator/demodulator circuits are called *modems* or *data sets*. Ordinarily, the communication link is a telephone line, which may be either a direct (or private) line or a switched line.

An equivalent circuit of one of the lines between an interface or terminal and a modem is given in Fig. 9-8. The RS-232-C electrical standard is stated in terms

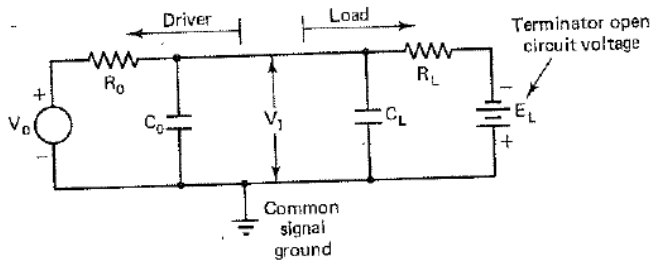


Figure 9-8 Equivalent circuit of a line between an interface or terminal and a modem.

of this circuit and is given in Fig. 9-9. If the interface or terminal is sending information to the modem, it is considered the driver and the modem is the load, and if it is receiving information, it is the load and the modem is the driver. In any case, the various components in the equivalent circuit must meet the specifications stated in Fig. 9-9. Note that the data signals are inverted in that a 1 is represented by the lower voltage. Whether or not the driver circuit for outputting data is part of the main interface device or is a separate circuit depends on the interface and the length of the line to be driven. Figure 9-10 shows a circuit for converting signals between TTL and the RS-232-C standard.

$V_0 < 25$ V

Maximum short circuit current to any wire in cable = 0.5 A

MARK signal at load < -3 V

SPACE signal at load $> +3$ V

MARK signal out of driver < -5 V and > -15 V

SPACE signal out of driver $> +5$ V and $< +15$ V

$R_L < 7000$ ohms when measured with a voltage from 3 to 25 V, but > 3000 ohms

C_L including line capacitance < 2500 pF

When $E_L = 0$, $5 \text{ V} < V_I < 15$ V

$R_0 > 300$ ohms under power off conditions

C_0 is such that the slew rate of the driver's output voltage is < 30 V/microsecond, but the transition between -3 V and $+3$ V does not exceed the smaller of 1 ms or 4% of the bit time

Figure 9-9 Principal RS-232-C electrical standards.

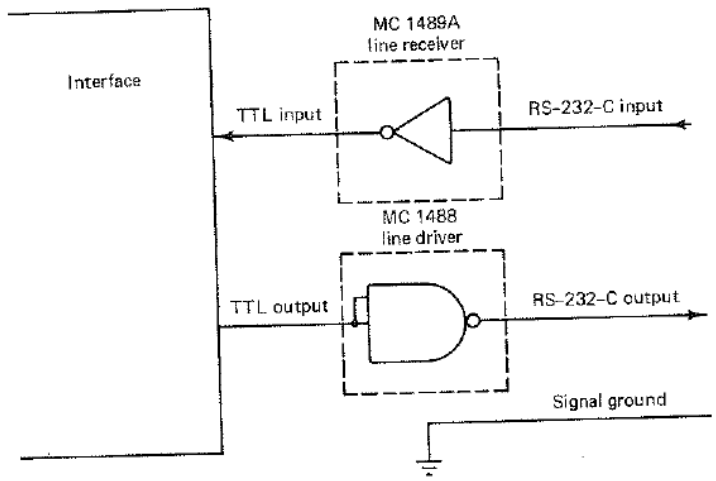


Figure 9-10 Driver circuit for RS-232-C send and receive lines.

Also part of the RS-232-C standard are the definitions and symbolic representations of the control lines that run between an interface or terminal and its modem. These definitions are summarized in Fig. 9-11, with the symbolic representation appearing in the first column, the standard name in the third column, and a brief description in the fourth column. Connections are most often made through 25-pin connectors and the second column gives the pin number of the pin to which each line is normally connected. Beside most symbols in the first column are numbers in parentheses. These numbers are the ones given to the lines by the CCITT standard.

Only the first eight lines in Fig. 9-11 are needed for communicating over a direct telephone line. For transmitting, a Request to Send (CA) signal is sent to the modem and this is acknowledged by a Clear to Send (CB). Then transmission may begin on the Transmitted Data (BA) line. For receiving, the Received Line Signal Detector (CF) line is activated by the modem to indicate that it is receiving a signal from the modem at the other end of the communication link, and the Received Data (BB) line is for sending the received data from the modem to the interface or terminal. The Data Set Ready (CC) indicates that the modem is turned on and is in its data mode. This line must be active when either transmitting or receiving.

If the communication link is part of a switched telephone system, at least two more lines are required. Signals are sent to the modem over the Data Terminal Ready (CD) line to control the switching of the modem and its related equipment so that a communication link is established. The Ring Indicator (CE) line is activated by the modem to indicate to the interface or terminal that the modem is receiving a ringing signal. To answer this signal the interface or terminal would respond by activating the Data Terminal Ready line. This would cause the modem and its switching equipment to "answer the ringing phone."

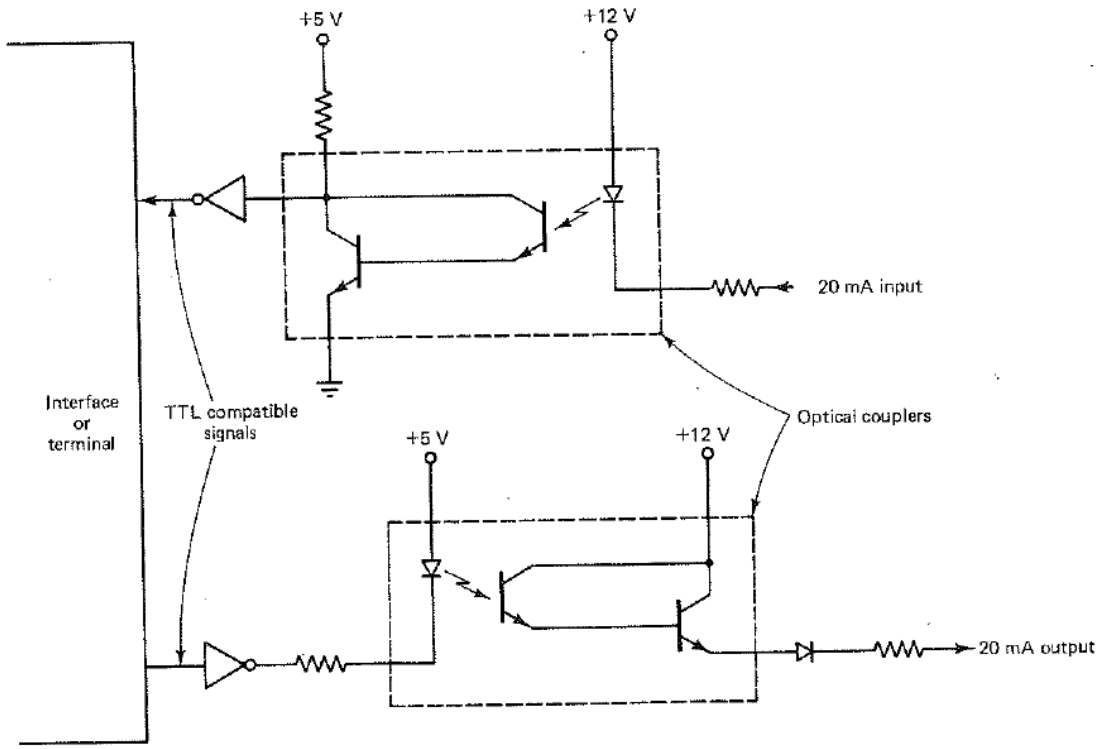
Figure 9-11 Summary of RS-232-C control line definitions.

Symbol EIA (CCITT)	Pin	Name	Description
AA (101)	1	Protective Ground	Used as an equipment ground
AB (102)	7	Signal Ground	Used as a common ground for all signals
BA (103)	2	Transmitted Data	For outputting data to modem
BB (104)	3	Received Data	For inputting data from modem
CA (105)	4	Request to Send	To modem - to turn the modem's transmitter on and off when operating in half duplex
CB (106)	5	Clear to Send	From modem - indicates modem is ready to transmit
CC (107)	6	Data Set Ready	From modem - indicates that modem power is on and the modem is not in test mode
CF (109)	8	Received Line Signal Detector	From modem - indicates the modem is receiving a signal from the modem at the other end of the link
CD (108/2)	20	Data Terminal Ready	To modem - prepares the modem to be connected to the communication link and begin transmitting
CE (125)	22	Ring Indicator	From modem - indicates ringing signal is detected on link
CG (110)	21	Signal Quality Detector	From modem - active when there is a low probability of error in the received data
CH (111)	23	Data Signal Rate Selector (DTE Source)	To modem - indicates to the modem one of two synchronous data rates or ranges of rates
CI (112)	23	Data Signal Rate Selector (DCE Source)	From modem - indicates to interface or terminal one of two synchronous data rates or ranges of rates
DA (113)	24	Transmitter Signal Element Timing (DTE Source)	To modem - provides modem transmitter with signal timing
DB (114)	15	Transmitter Signal Element Timing (DCE Source)	From modem - provides interface or terminal with transmitter signal timing
DD (115)	17	Receiver Signal Element Timing	From modem - provides interface or terminal receiver with signal timing
SBA (118)	14	Secondary Transmitted Data	To modem - for outputting low rate data
SBB (119)	16	Secondary Received Data	From modem - for inputting low rate data
SCA (120)	19	Secondary Request to Send	To modem - to turn on modem's secondary transmitter
SCB (121)	13	Secondary Clear to Send	From modem - indicates secondary transmitter is ready
SCF (122)	12	Secondary Received Line Signal Detector	From modem - indicates signal is detected on secondary link

Synchronous communication requires that timing information be communicated along with the data. There are two lines for providing this information: the Transmitter Signal Element Timing lines. The DTE Source (DA) line is for sending timing information from the interface or terminal to the modem and the DCE Source (DB) line is for transmitting timing information in the other direction. Because modems for synchronous transmissions often accommodate two data rates or two ranges of data rates, the Data Signal Rate Selector (CH and CI) lines are for indicating rate information. CH is for sending a rate selection from the interface or terminal to the modem and CI is used by the modem to indicate to the interface or terminal the data rate it is currently using.

For both asynchronous and synchronous high-speed transmission it has been found to be more efficient to send the information in packets containing identification and error detection information as well as the data. Packets are normally used when computers are communicating with each other. In connection with the transmission of packets a certain amount of handshaking and verification must take place and this can best be done using a secondary low-rate communication link. The last five lines defined in Fig. 9-11 are for these secondary transmissions. Because a full discussion of packets and their associated protocols would take more

Figure 9-12 Circuits for driving and receiving 20-mA loop signals.



space than we have available, these lines are not considered further here. For more information the reader should refer to Reference 1.

It should be mentioned that there is a second electrical standard that is used for transmitting data back and forth between an interface or terminal and a modem. It is called the *20-mA loop standard* and is implemented by having the line consist of a loop that carries a 20-mA current when it is in a mark (1) state and does not carry a current when it is in a space (0) state. In a 20-mA loop configuration one of the commmunicating devices must be the active device and provide the current source, while the other device would be passive. Historically, this standard began with mechanical teleprinters that used the current for driving their solenoids, but even most CRT terminals have the option of communicating over either an RS-232-C line or a 20-mA line. However, as the need for current signals recedes, so does the use of the 20-mA loop standard and RS-232-C links are far more prevalent today. Circuits for driving and receiving 20-mA loop signals are shown in Fig. 9-12.

Many of the details of the above standards were not discussed. There are several books on data communications, some of which are listed in the references at the end of the chapter. The above discussion was directed toward the connection between the computer interface and the communication system, and included primarily information that is needed to understand interface design.

9-1-4 8251A Programmable Communication Interface

As an example of a serial interface device let us consider Intel's 8251A programmable communication interface. The 8251A is diagrammed in Fig. 9-13. It is capable of being programmed for asynchronous or synchronous communication. The data-in buffer and data-out buffer registers share the same port address. For input, the serial bit stream arriving on the RxD pin is shifted into the receiver shift register and then the data bits are transferred to the data-in buffer register, where they can be input by the CPU. Conversely, on output the data bits put in the data-out buffer register by the CPU are transferred to the transmitter shift register and, along with the necessary synchronization bits, are shifted out through the TxD pin. Among other things the contents of the mode register, which are initialized by the executing program, determine whether the 8251A is in asynchronous mode or synchronous mode and the format of the characters being received and transmitted. The control register, which is also set by the program, controls the operation of the interface, and the status register makes certain information available to the executing program. Clearly, the sync character registers are for storing the sync characters needed for synchronous communication.

Even though all seven of the registers on the left side of Fig. 9-13 can be accessed by the processor, the 8251A is associated with only two port addresses. The C/$\overline{D}$ pin is connected to the address line A0 and A0 differentiates the two port

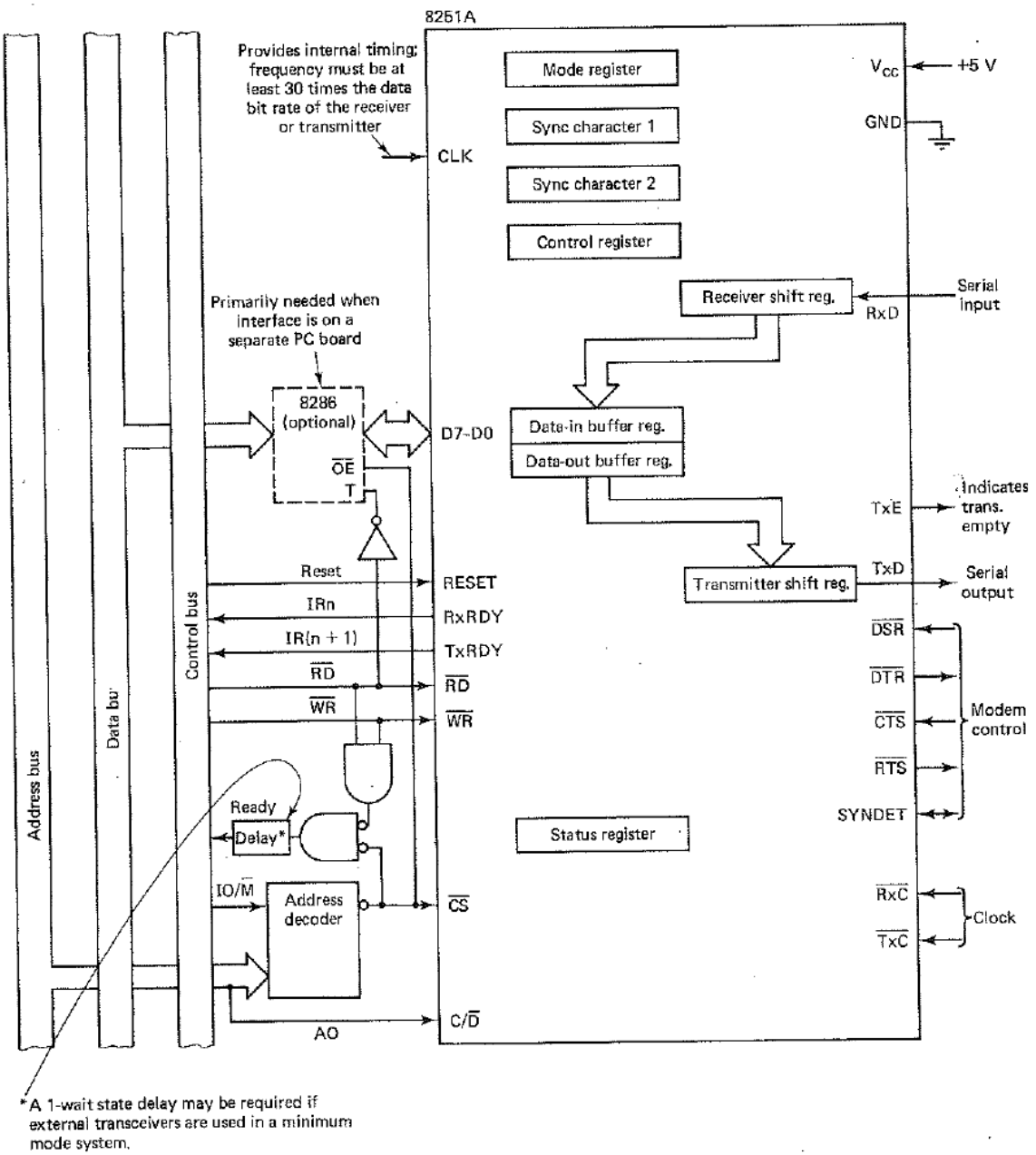


Figure 9-13 8251A serial communication interface.

addresses. The 8251A internally interprets the C/$\overline{D}$, $\overline{RD}$, and $\overline{WR}$ signals as follows:

C/$\overline{D}$ (= A0)	$\overline{RD}$	$\overline{WR}$	
0	0	1	Data input from the data-in buffer
0	1	0	Data output to the data-out buffer
1	0	1	Status register is put on data bus
1	1	0	Data bus is put in mode, control, or sync character register

where 1 means that the pin is high and 0 means that it is low. All other combinations cause the three-state D7-D0 pins to go into their high-impedance state.

Whether the mode, control, or sync character register is selected depends on the accessing sequence. A flowchart of the sequencing is given in Fig. 9-14. After a hardware reset or a command is given with its reset bit set to 1, the next output with A0 = 1 (i.e., with C/$\overline{D}$ = 1, $\overline{RD}$ = 1, and $\overline{WR}$ = 0) is directed to the mode register. The formats of the mode register for both the asynchronous and synchronous cases are defined in Fig. 9-15. If the two LSBs of the mode are zero, then the interface is put in its synchronous mode and the MSB determines the number of sync characters. In the synchronous mode, the next 1 or 2 bytes output with A0 = 1 become the sync characters. If the two LSBs of the mode are not both 0, then the 8251A enters its asynchronous mode. In either case, all subsequent bytes prior to another reset go to the control register if A0 = 1 and the data-out buffer register if A0 = 0.

In the synchronous mode the baud rates of the transmitter and receiver, which are the shift rates of the shift registers, are the same as the frequencies of the signals applied to $\overline{TxC}$ and $\overline{RxC}$, respectively, but in the asynchronous mode the three remaining possible combinations for the two LSBs in the mode register dictate the baud rate factor. The relationship between the frequencies of the $\overline{TxC}$ and $\overline{RxC}$ clock inputs and the baud rates of the transmitter and receiver is:

$$\text{Clock frequency} = \text{Baud rate factor} \times \text{Baud rate}$$

If 10 is in the LSBs of the mode register and the transmitter and receiver baud rates are to be 300 and 1200, respectively, then the frequency applied to $\overline{TxC}$ should be 4800 Hz and the frequency at $\overline{RxC}$ should be 19.2 kHz.

In both the asynchronous and synchronous modes, bits 2 and 3 indicate the number of data bits in each character, bit 4 indicates whether or not there is to be parity bit, and bit 5 specifies the type of parity (odd or even). For the asynchronous mode the two MSBs indicate the number of stop bits, but for the synchronous mode bit 6 determines whether the SYNDET pin is to be used as an input or as an output and, as mentioned above, bit 7 indicates the number of sync characters.

If the SYNDET pin is used as an output it becomes active when a bit-for-bit match has been found between the incoming bit stream and the sync character(s). If the search for sync characters is conducted by an external device, then SYNDET can be used to input a signal, indicating that a match has been found by the external device. The pin also has a meaning during asynchronous operation, but in this case it can only be an output. This output is called the break detect signal and goes high whenever a character consisting of all 0s is received.

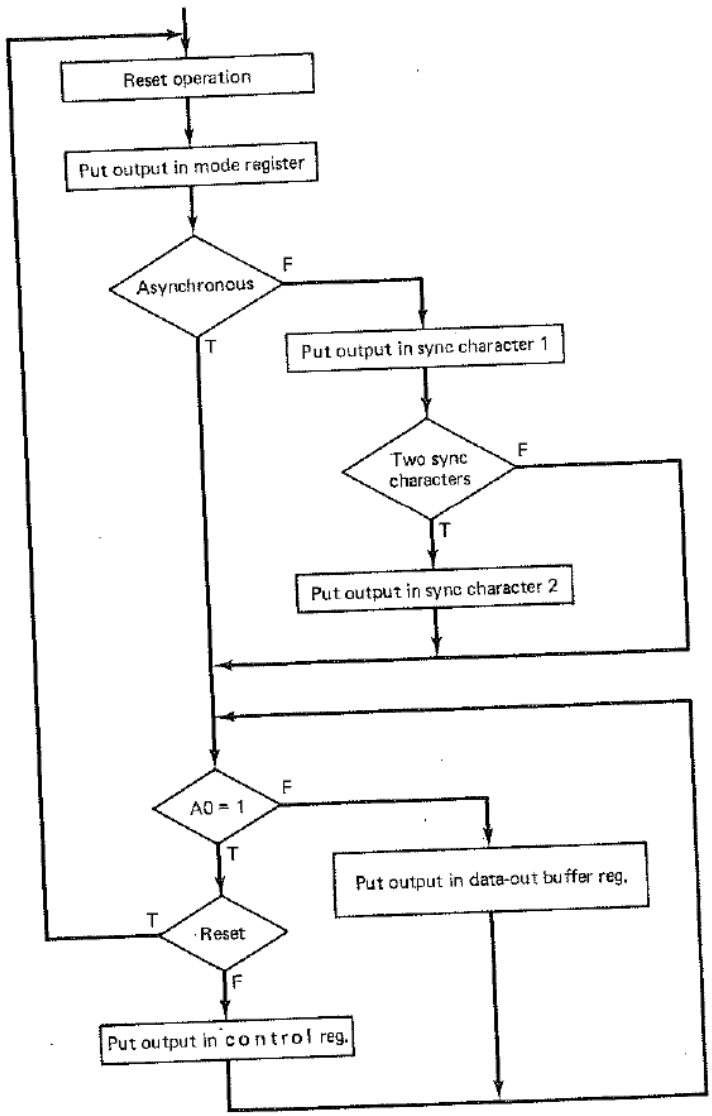
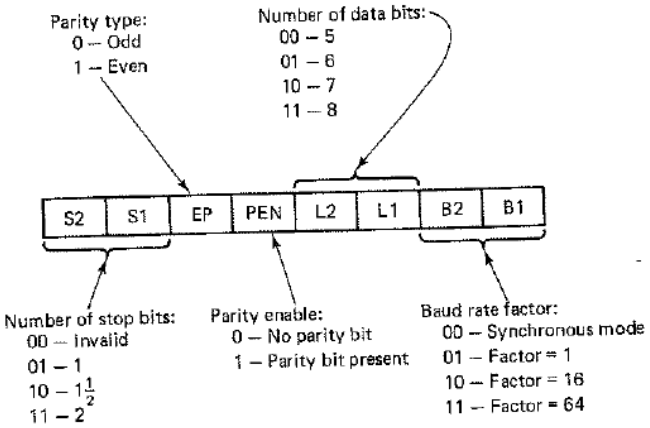
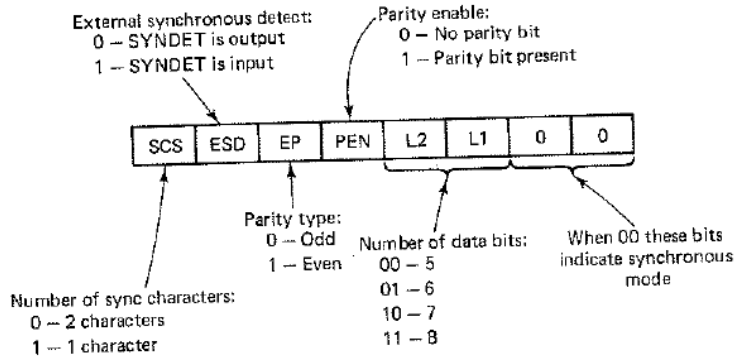


Figure 9-14 Flowchart of the disposition of output.

The format for the control register is given in Fig. 9-16. Bit 0 of this register must be 1 before data can be output and bit 2 must be 1 before data can be received. Programmed answering of a modem is accomplished by setting bit 1 to 1 since this forces the $\overline{\text{DTR}}$ pin to 0 and the complement of $\overline{\text{DTR}}$ is normally connected to the CD line from the modem. Bit 3 equal to 1 forces TxD to 0, thus causing break characters to be transmitted. Setting bit 4 to 1 causes all the error bits in the status register to be cleared (the bits that are set when framing, overrun, and parity errors occur). Bit 5 is used for sending a Request to Send signal to a modem. If the complement of the $\overline{\text{RTS}}$ pin is connected to a modem's CA line, then a 1 put in



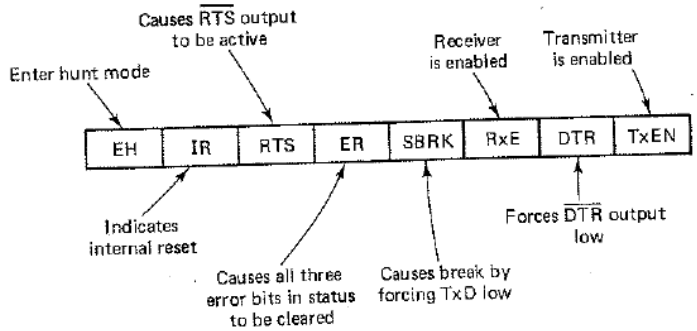
(a) Asynchronous mode



(b) Synchronous mode

Figure 9-15 Format of the mode register.

Figure 9-16 Format of the control register.



Note: In all cases action is taken when the bit is set to 1.

bit 5 will cause the CA line to go high. Setting bit 6 causes the 8251A to be reinitialized and the reset sequence to be reentered (i.e., a return is made to the top of the flowchart shown in Fig. 9-14 and the next output will be to the mode register). Bit 7 is used only with the synchronous mode. When set, it causes the 8251A to begin a bit-by-bit search for a sync character or sync characters.

Typical connections to modems for asynchronous and synchronous transmissions are shown in Fig. 9-17. With regard to the synchronous connections it is assumed that the timing is controlled by the modem and its related communications equipment. Also, if this equipment is used to detect the sync character(s) at the beginning of a received message, then it can inform the 8251A of its success over the SYNDET line. On the other hand, if 8251A searches for the sync character(s), then it can use the SYNDET line as an output to tell the modem that the sync character(s) has been found. To satisfy the RS-232-C standard, drivers and receivers are needed to convert the TTL-compatible signals at TxD and RxD to the proper voltage levels (see Fig. 9-10).

A program sequence which initializes the mode register and gives a command to enable the transmitter and begin an asynchronous transmission of 7-bit characters followed by an even-parity bit and 2 stop bits is:

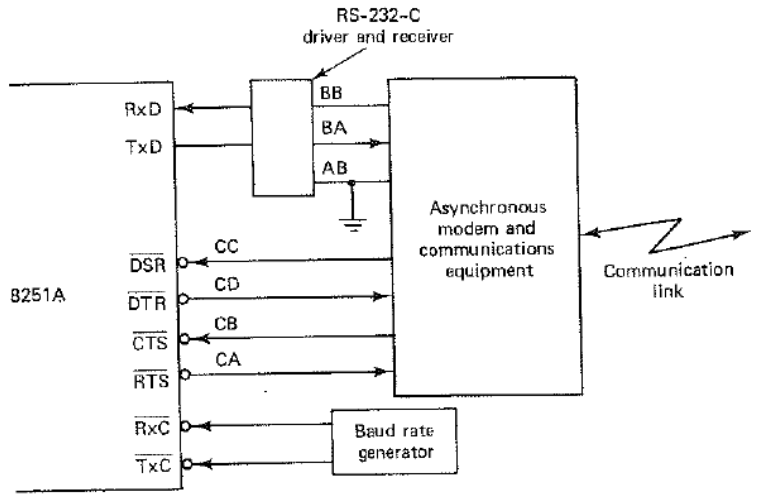
```
MOV   AL,11111010B
OUT   51H,AL
MOV   AL,00110011B
OUT   51H,AL
```

This sequence assumes that the mode and control registers are at address 51H and the clock frequencies are to be 16 times the corresponding baud rates. The sequence:

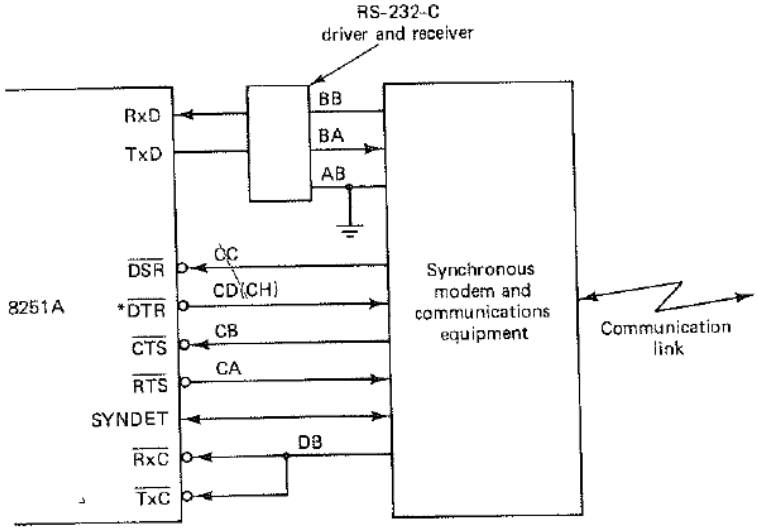
```
MOV   AL,00111000B
OUT   51H,AL
MOV   AL,16H
OUT   51H,AL
OUT   51H,AL
MOV   AL,10010100B
OUT   51H,AL
```

would cause the same 8251A to be put in synchronous mode and to begin searching for two successive ASCII sync characters. As before, the characters are to consist of 7 data bits and an even parity bit, but there will, of course, be no stop bits.

The format of the status register is given in Fig. 9-18. Bits 1, 2, and 6 reflect the signals on the RxRDY, TxE, and SYNDET pins. TxRDY indicates that the data-out buffer is empty. Unlike the TxRDY pin, this bit is not affected by the $\overline{\text{CTS}}$ input pin or the TxEN control bit. RxRDY indicates that a character has been received and is ready to be input to the processor. Either the TxRDY and RxRDY bits can be used for programmed I/O or the signals on the corresponding pins can be connected to interrupt request lines to provide for interrupt I/O. The TxRDY bit is automatically cleared when a character is made available for transmitting and the RxRDY bit is automatically cleared when the character that set it is input by the processor. Bit 2 indicates that the transmitter shift register is waiting



(a) Asynchronous modem connection



*May also be used for rate selection -- modem line CH.

(b) Synchronous modem connection

NOTE: Control line interfacing depends on modem specifications.

Figure 9-17 8251A modem connections.

to be sent a character from the data-out buffer register. During synchronous transmissions, while this bit is set, the transmitter will take its data from the sync character registers until data are put in the data-out buffer register. Bits 3, 4, and 5 indicate parity, overrun, and framing errors, respectively. When an error is detected, the bit having the corresponding error type will be set to 1. If the com-

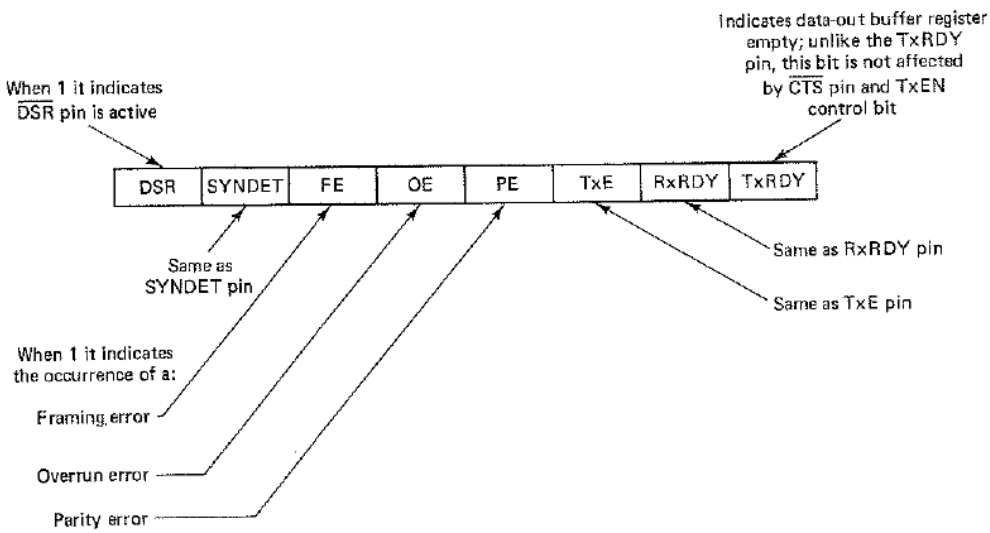


Figure 9-18 Format of the status register.

plement of the $\overline{\text{DSR}}$ pin is connected to the Data Set Ready (CC) line, then bit 7 reflects the state of the modem and is 1 when the modem is turned on and is in its data mode.

Figure 9-19 gives a typical program sequence which uses programmed I/O to input 80 characters from the 8251A, whose data buffer register's address is 0050, and put them in the memory buffer beginning at LINE. The inner loop continually tests the RxRDY bit until it is set by a character being put in the data-in buffer register. Then the newly arrived character is moved to the buffer and the error bits are checked. If the present character arrived before the previous character was input or a parity or framing error occurred during transmission, then the input

Figure 9-19 Inputting a line of characters through an 8251A.

```
        MOV   AL,00110101B        ;ENABLE TRANSMITTER AND RECEIVER
        OUT   51H,AL              ;AND CLEAR ERROR BITS
        MOV   DI,0                ;INITIALIZE INDEX
        MOV   CX,80               ;PUT COUNT IN CX
BEGIN:  IN    AL,51H              ;WAIT FOR INPUT
        TEST  AL,02H
        JZ    BEGIN
        IN    AL,50H              ;INPUT CHARACTER AND
        MOV   LINE[DI],AL         ;PUT IN LINE BUFFER
        INC   DI
        IN    AL,51H              ;CHECK ERROR
        TEST  AL,00111000B        ;BITS AND
        JNZ   ERROR               ;IF NO ERROR IS FOUND
        LOOP  BEGIN               ;CONTINUE INPUTTING
        JMP   SHORT EXIT
ERROR:  CALL  NEAR PTR ERR_ROUT   ;ELSE CALL ERR_ROUT
EXIT:   .
        .
        .
```

ceases and a call is made to an error routine that would presumably examine the individual error bits, print an appropriate message, and clear the error bits.

Because inputting a character automatically resets the RxRDY bit, unless another character is received before the inner loop is reentered, the inner loop must cycle until the RxRDY bit is reset to 1 by the next incoming character. If the incoming characters have fewer than 8 bits, the unused MSBs in the data buffer register are always zeroed. Also, the parity bit is not passed to the processor and checks for parity errors can only be made by examining the parity error bit in the status register. On output, if a character is less than 8 bits long, the unneeded MSBs in the data-out buffer register are ignored.

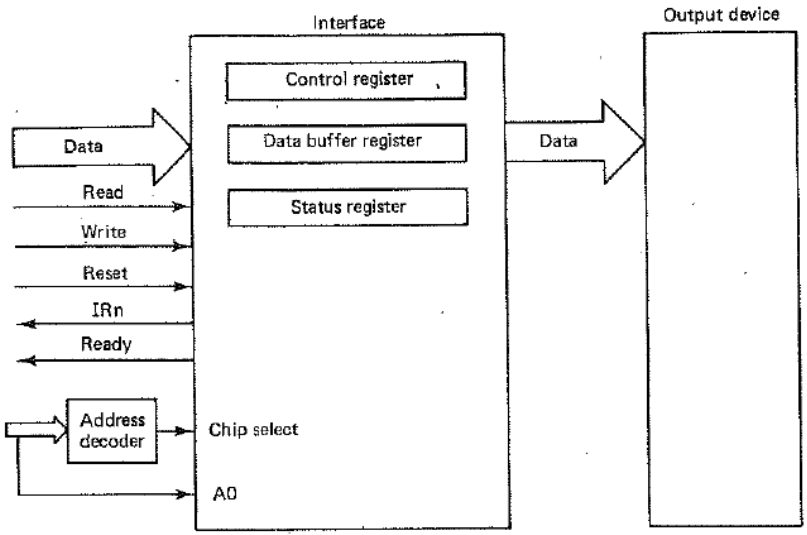
9-2 PARALLEL COMMUNICATION

Parallel communication is accomplished by simultaneously transferring bits over separate lines. Its advantage over serial communication is that for lines with a given maximum bit rate, higher information rates can be attained due to the use of several lines. The disadvantage is, of course, the cost of the extra lines and, because these costs increase with distance, parallel communication is used over long distances only if high data rates are required.

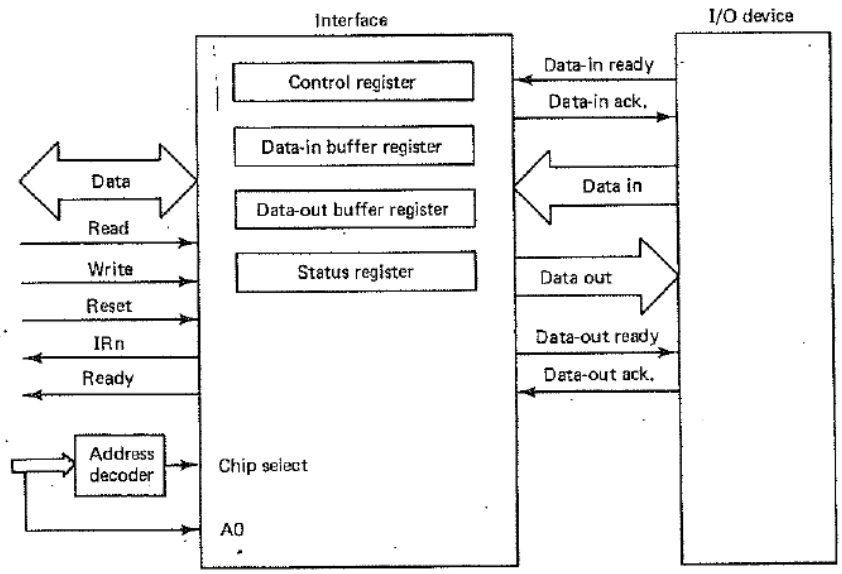
Unlike serial communication, parallel communication has not been well standardized. Parallel transfers may be performed by simply putting data in the interface's data-in buffer register or taking data out of the interface's data-out buffer register, or they may be conducted under the control of handshaking and/or timing signals. Normally, one character (or other piece of information) is transferred at a time and there is no problem in determining the end of a character or in finding the beginning of a transmission. There is no well-defined format for synchronous or asynchronous transmissions, but if a timing signal is to coordinate the activity between the device and the interface, the transmission would be considered synchronous. If only handshaking signals are employed, the transmissions would be asynchronous.

An interface may be designed for only outputting, only inputting, inputting and outputting over separate sets of lines, or performing I/O over a single set of bidirectional lines. If an interface is connected to a line printer, it would only output data, and if it services a card reader, it would only input data. An interface that services both a paper tape reader and a paper tape punch would require one set of input lines and one set of output lines, but an interface for a device that does not input and output data simultaneously could use only one set of bidirectional lines.

A typical parallel output interface which uses no control lines is given in Fig. 9-20(a). In this case the contents of the data buffer register would continually be maintained on the data lines connecting the interface and the output device. If the output device consisted of a latch and a set of relays that are controlled by the bit settings in the latch, then the computer could control the relays by simply outputting data to the data buffer register.



(a) Output only



(b) Input and output

Figure 9-20 Representative parallel communication interfaces.

Figure 9-20(b) shows a representative parallel interface with handshaking control lines and separate input and output connections to an I/O device. In this case, an input would be carried out by first putting the data on the data bus and a 1 on the data-in ready line. The interface would respond by latching the data into the data-in buffer and putting a 1 on the data-in acknowledge line. Upon receiving the acknowledgment the device would drop the data and ready signals. When the data are received by the interface, it would set a "ready" status bit and perhaps send out an interrupt request. After the CPU takes the data the interface would clear the "ready" status bit and put the data lines in their high-impedance state. If an interrupt request were made, it would be handled in the usual manner.

For an output, the interface would set a "ready" status bit and, perhaps, make an interrupt request when the data-out buffer is available. After the CPU outputs the data, the interface would clear the output "ready" status bit, put the data on the data-out bus, and signal the I/O device over the data-out ready line. When the device is ready to accept the data, it would latch the data and then return an acknowledgment. The interface would then drop its data-out ready signal and once again set the output "ready" status bit.

9-2-1 8255A Programmable Peripheral Interface

Intel's 8255A programmable peripheral interface provides a good example of a parallel interface. As shown in Fig. 9-21, the interface contains a control register and three separately addressable ports, denoted A, B, and C. Whether or not an 8255A is being accessed is determined by the signal on the $\overline{CS}$ pin and the direction of the access is according to the $\overline{RD}$ and $\overline{WR}$ signals. Which of the four registers is being addressed is determined by the signals applied to the pins A1 and A0. Therefore, the lowest port address assigned to an 8255A must be divisible by 4. A summary of the 8255A's addressing is:

A1	A0	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	Transfer Description
0	0	0	1	0	Port A to data bus
0	1	0	1	0	Port B to data bus
1	0	0	1	0	Port C to data bus
0	0	1	0	0	Data bus to port A
0	1	1	0	0	Data bus to port B
1	0	1	0	0	Data bus to port C
1	1	1	0	0	Data bus to control register if D7 = 1; if D7 = 0, input from the data bus is treated as a Set/Reset instruction
x	x	x	x	1	D7-D0 go to high-impedance state
1	1	0	1	0	Illegal combination
x	x	1	1	0	D7-D0 go to high-impedance state

where 0 is low and 1 is high.

Because the bits in port C are sometimes used as control bits, the 8255A is designed so that they can be output to individually using a Set/Reset instruction. When the 8255A receives a byte that is directed to its control register, it examines

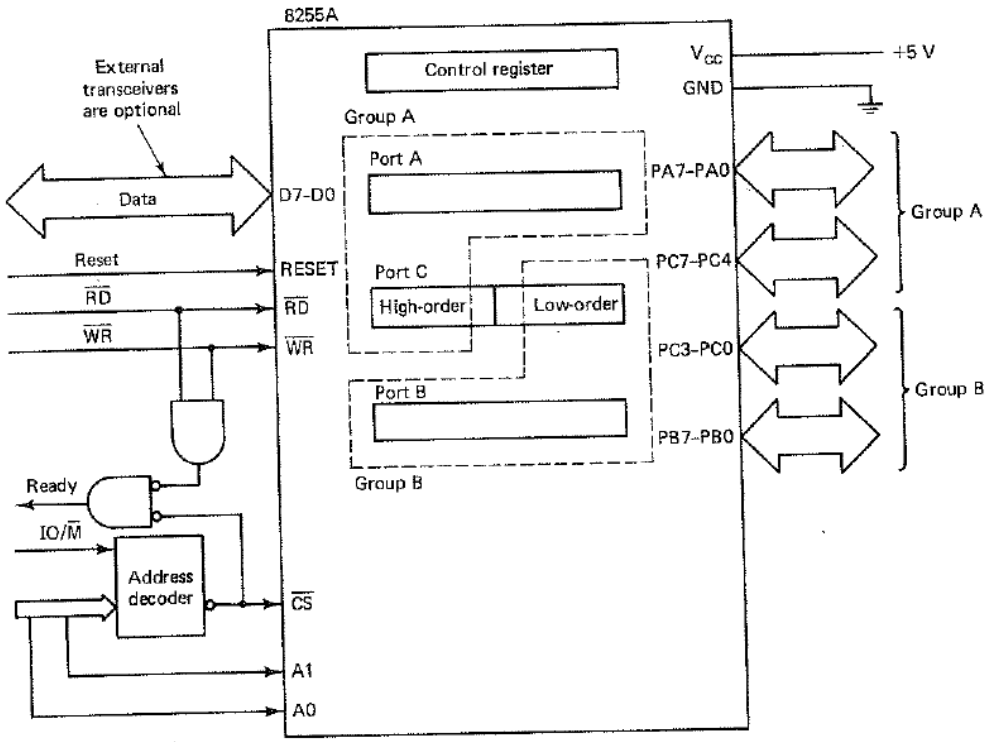


Figure 9-21 Diagram of the 8255A.

the data bit 7. If this bit is 1, the data are transferred to the control register, but if it is 0, the data are treated as a Set/Reset instruction and is used to set or clear the port C bit specified by the instruction. Bits 3-1 give the bit number of the bit to be changed and bit 0 indicates whether it is to be set or cleared. The remaining bits are unused.

The bits in the three ports are attached to pins that may be connected to the I/O device. These bits are divided into groups A and B, with group A consisting of the bits in port A and the four MSBs of port C and group B consisting of port B and the four LSBs of port C. The use of each of the two groups is controlled by the mode associated with it. Group A is associated with one of three modes, mode 0, mode 1, and mode 2, and group B with one of two modes, mode 0 and mode 1. The modes are determined by the contents of the control register, whose format is given in Fig. 9-22. These modes are:

Mode 0—If a group is in mode 0, it is divided into two sets. For group A these sets are port A and the upper 4 bits of port C, and for group B they are port B and the lower 4 bits of port C. Each set may be used for inputting or outputting, but not both. Bits D4, D3, D1, and D0 in the control register

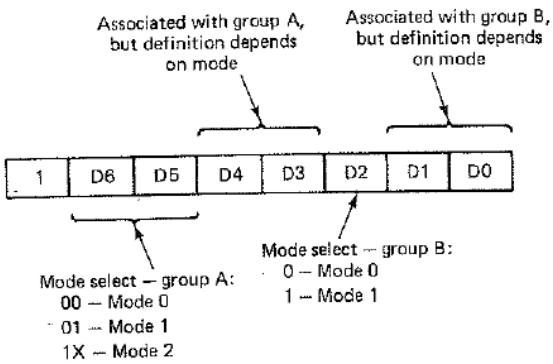


Figure 9-22 Format of the 8255A's control register.

specify which sets are for input and which are for output. These bits are associated with the sets as follows:

D4—Port A.
D3—Upper half of port C.
D1—Port B.
D0—Lower half of port C.

If a bit is 0, then the corresponding set is used for output; if it is 1, the set is for input.

Mode 1—When group A is in this mode port A is used for input or output according to bit D4 (D4 = 1 indicates input), and the upper half of port C is used for handshaking and control signals. For *inputting*, the four MSBs of port C are assigned the following symbols and definitions:

PC4	$\overline{\text{STB}}_A$	A 0 applied to this pin causes PA7-PA0 to be latched, or "strobed," into port A.
PC5	IBF_A	Indicates that the input buffer is full. It is 1 when port A contains data that have not yet been input to the CPU. When a 0 is on this pin the device can input a new byte to the interface.
PC6,PC7		May be used to output control signals to the device or input status from the device. If D3 of the control register is 0, they are for outputting control signals; otherwise, they are for inputting status.

For *outputting*:

PC4,PC5		Serve the same purpose as described above for PC6 and PC7.
PC7	$\overline{\text{OBF}}_A$	Indicates that the output buffer is full. It outputs a 0 to the device when port A is outputting new data to be taken by the device.
PC6	$\overline{\text{ACK}}_A$	Device puts a 0 on this pin when it accepts data from port A.

In mode 1, PC3 is denoted INTR_A and is associated with group A. It is used as an interrupt request line and is tied to one of the IR lines in the system

bus. When inputting to port A, this pin becomes 1 when new data are put in port A (i.e., it is controlled by PC4) and is cleared when the CPU takes the data. For output, this pin is set to 1 when the contents of port A are taken by the device and is cleared when new data are sent from the CPU. If group B is in mode 1, port B is input to or output from according to bit D1 of the control register (D1 = 1 indicates input). For input, PC2 and PC1 are denoted $\overline{STB}_B$ and IBF_B, respectively, and serve the same purposes for group B as $\overline{STB}_A$ and IBF_A do for group A. Similarly, for output PC1 and PC2 are denoted $\overline{OBF}_B$ and $\overline{ACK}_B$. PC0 becomes $INTR_B$ and its use is analogous to that of $INTR_A$. The interrupt enable for group A is controlled by setting or clearing internal flags. Setting or clearing these flags is simulated by setting or clearing PC4, for input, or PC6, for output, using a Set/Reset instruction. Similarly, the interrupt enable for group B is controlled by set/clear of PC2 for both input and output.

Mode 2—This mode applies only to group A, although it also uses PC3 for making interrupt requests. In mode 2, port A is a bidirectional port and the four MSBs of port C are defined as follows:

PC4	$\overline{STB}_A$	A 0 on this line causes the data on PA7-PA0 to be strobed into port A.
PC5	IBF_A	Becomes 1 when port A is filled with new data from lines PA7-PA0 and is cleared when these data are taken by the CPU.
PC6	$\overline{ACK}_A$	Indicates that the device is ready to accept data from PA7-PA0.
PC7	$\overline{OBF}_A$	Becomes 0 when port A is filled with new data from the CPU and is set to 1 when data are taken by the device.

While group A is in mode 2, group B may be in either mode 0 or mode 1. However, if group B is in mode 0, only PC2-PC0 can be used for input or output because group A has borrowed PC3 to use as an interrupt request line. Normally, if group A is in mode 2, PC2-PC0 would be connected to control and status pins on the device attached to the port A lines. Port B may also be used for this purpose.

In all three modes port C reflects the signals on PC7-PC0 and can be read with an IN instruction.

9-2-2 A/D and D/A Example

Figure 9-23 shows how an 8255A could be connected to an A/D and D/A subsystem. Since during an A/D conversion the analog voltage must remain unchanged, a sample-and-hold circuit is needed to keep the analog signal constant while the conversion is being performed. Group A is configured as an input in mode 1. A conversion is initiated by a signal from the 8255A's PC7 pin, which prompts the converter to output a busy signal. The busy line is connected to both a sample-

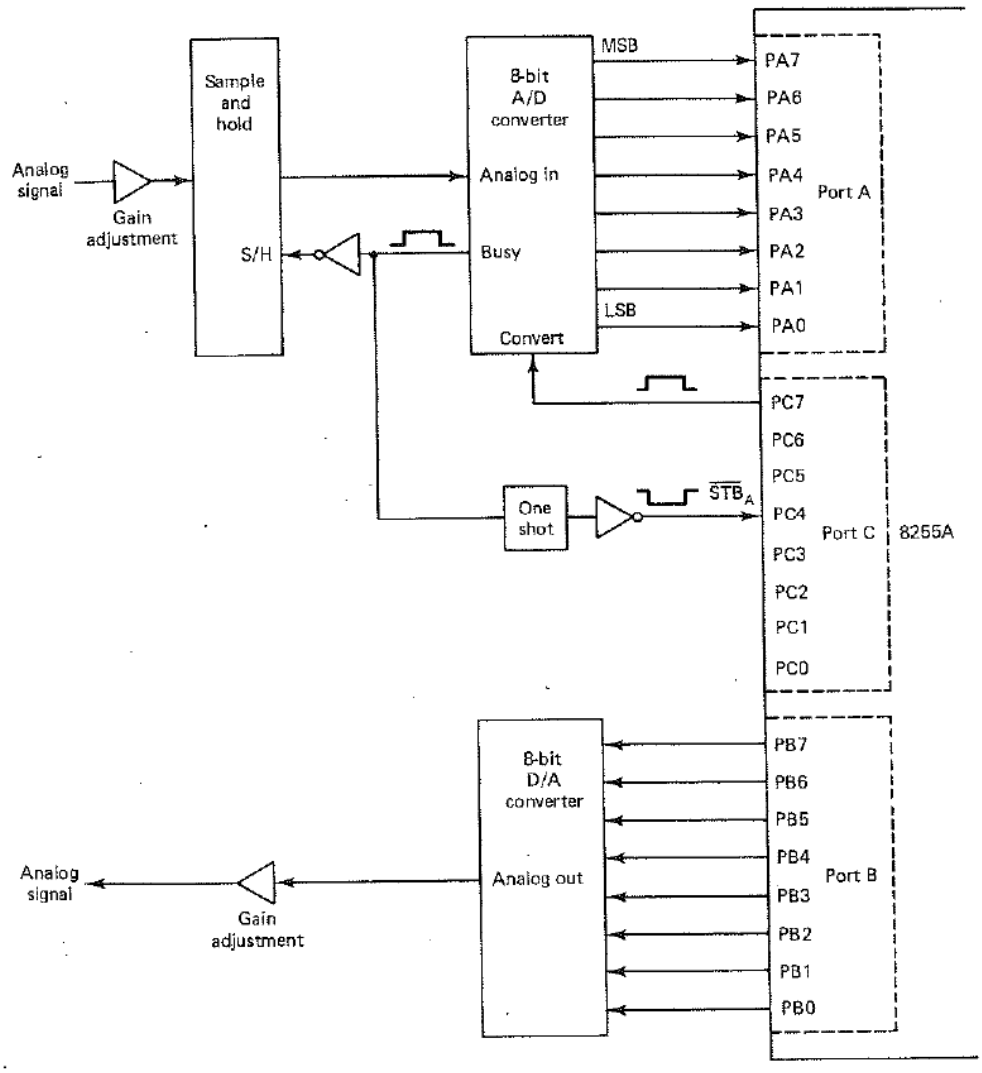


Figure 9-23 Interfacing an A/D and D/A subsystem using an 8255A.

and-hold control pin (S/H) and a negative edge-triggered one-shot. While the busy signal is high the sample-and-hold circuit maintains a constant output, and when the busy signal goes down at the end of the conversion the one-shot is triggered. The output from the one-shot is complemented and applied to the $\overline{STB}_A$ (PC4) input on the 8255A.. This causes the digitized sample to be strobed into port A. For the D/A portion of the subsystem, port B is configured as an output port in mode 0, which is directly connected to the binary input of the D/A converter. No handshaking is used.

Given that port A, port B, port C, and the control register have addresses FFF8, FFF9, FFFA, and FFFB, respectively, the sequence:

```
MOV   DX,0FFFBH
MOV   AL,10110000B
OUT   DX,AL
```

would cause port A to be put in mode 1, port B to be put in mode 0, and PC7 to be an output. The sequence:

```
MOV   DX,0FFFBH
MOV   AL,00001111B
OUT   DX,AL
MOV   AL,00001110B
OUT   DX,AL
```

would output a pulse to the convert pin of the A/D converter. The first instruction of the latter sequence puts the address associated with Set/Reset instruction, which is the same as the address of the control register, in the DX register. The next two instructions cause PC7 to be set and the last two cause it to be cleared. A sequence for providing a programmed I/O input of the converted data is:

```
        MOV   DX,0FFFAH
AGAIN:  IN    AL,DX
        TEST  AL,00100000B
        JZ    AGAIN
        MOV   DX,0FFF8H
        IN    AL,DX
```

For outputting a byte from AL to the D/A converter, only the instructions

```
MOV   DX,0FFF9H
OUT   DX,AL
```

are needed. As soon as the byte arrives at port B its bits are immediately applied to the input pins of the D/A converter, which, in turn, immediately converts it to an analog signal.

In this example it has been assumed that the timing of the conversions is provided by the program and that the gains of the input and output analog amplifiers are manually adjusted. To obtain even spacing between input or output samples from the program the execution times of the instructions must be used. This means that exactly the same instructions be executed between samples and that the total execution time of these instructions be accurately known. Interrupts would need to be disabled because they could randomly insert the execution of different numbers of instructions. The sample time could be adjusted by including a "do nothing" loop, such as:

```
        MOV   CX,N
IDLE:   NOP
        LOOP  IDLE
```

between the inputs or outputs. A flowchart for inputting a block of A/D samples using programmed timing is given in Fig. 9-24.

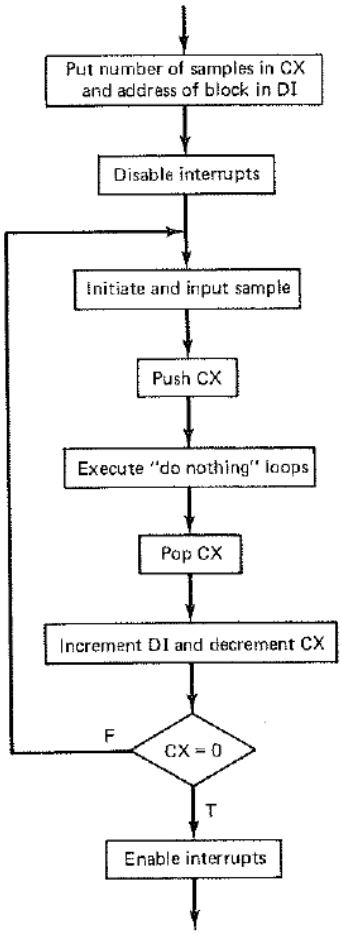


Figure 9-24 Programmed sample timing.

Frequently, programmable clocks and gain control devices are associated with A/D and D/A subsystems so that the timing of the sampling and D/A outputs are more precisely controlled and the gains can be dynamically changed. Also, a DMA controller is often used in conjunction with an A/D and D/A subsystem so that high-speed input and output can be attained.

Only 8-bit A/D and D/A converters are included in the design shown in Fig. 9-23. This limits the resolution to only 1 part in 256. If the voltage range of the input or output were -10 V to $+10$ V, the resolution would be:

$$\frac{20}{256} = 0.078 \text{ V}$$

For higher resolutions, 10-, 12-, or 14-bit converters are required. To accommodate the greater number of bits, a combination of ports A and C or ports B and C could be used (see Exercise 14) or two 8255As could be used in parallel (see Sec. 9-7).

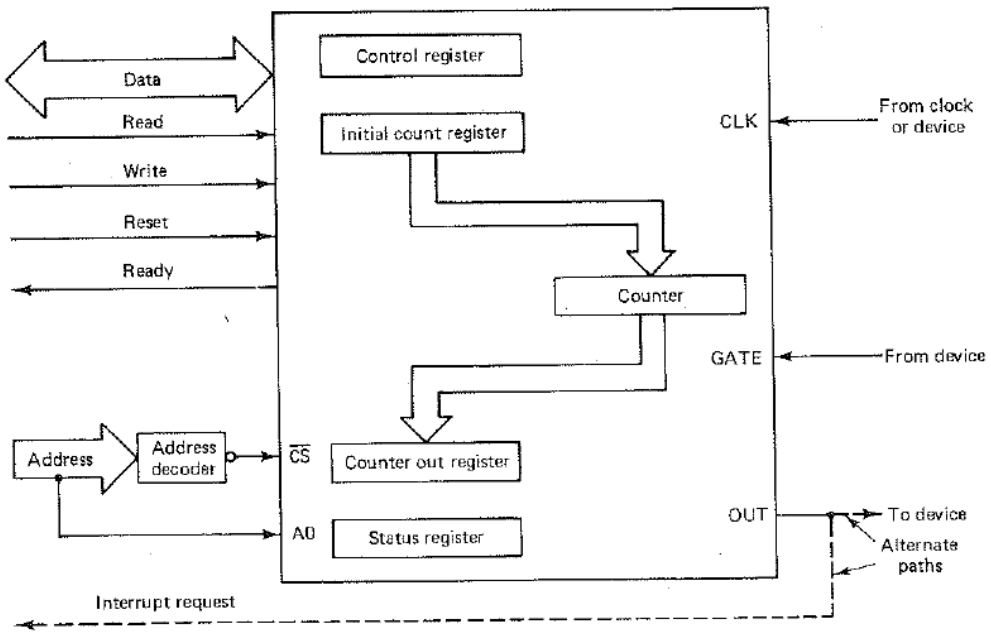
9-3 PROGRAMMABLE TIMERS AND EVENT COUNTERS

Quite often a device is needed to mark intervals of time for both the processor and external devices, count external events and make the count available to the processor, and provide external timing that can be programmed from the processor. Such a device is called a *programmable interval timer/event counter* and some of its uses are to:

1. Interrupt a time-sharing operating system at evenly spaced intervals so that it can switch programs.
2. Output precisely timed signals with programmed periods to an I/O device (e.g., an A/D converter).
3. Serve as a programmable baud rate generator.
4. Measure time delays between external events.
5. Count the number of times an event occurs in an external experiment and provide a means of inputting the count to the computer.
6. Cause the processor to be interrupted after a programmed number of external events have occurred.

A representative design of an interval timer/event counter is given in Fig. 9-25. In this design the four registers on the left are accessible by the computer, with

Figure 9-25 Typical interval timer/event counter.



the upper two being output ports and the lower two input ports. The counter itself is not directly available to the processor, but must be initialized from the initial count register and can be read only by first transferring its contents to the counter out register. The counter operates by starting at an initial value and counting backward to 0. The CLK input determines the count rate, GATE is for enabling and disabling the CLK input and perhaps other purposes, and the OUT output becomes active when the count reaches 0 or, possibly, when a GATE signal is received. OUT may be connected to an interrupt request line in the system bus so that an interrupt will occur when the count reaches 0, or to an I/O device which uses it to initiate specific I/O activity.

The operation is basically to enter a count in the initial count register, transfer the count to the counter, and cause the counter to count backward as pulses are applied to the CLK input. The current contents of the counter can be input at any time without disturbing the count by transferring them to the counter out register and then reading them. By buffering the count through the counter out register it does not have to be input to the processor immediately. The zero count indication would normally be applied to both the OUT pin and one of the bits in the status register. Thus either programmed I/O or interrupt I/O could be used to detect the zero count.

Among other things the control register includes the mode of operation. The mode determines exactly what happens when the count becomes 0 and/or a signal is applied to the gate input. Some possible actions are:

1. The GATE input is used for enabling and disabling the CLK input.
2. The GATE input may cause the counter to be reinitialized.
3. The GATE input may stop the count and force OUT high.
4. The count will give an OUT signal and stop when it reaches 0.
5. The count will give an OUT signal and automatically be reinitialized from the Initial Count Register when the count reaches 0.

The modes could be defined by combinations of these possibilities.

As an example, consider the application of an interval timer to a time-sharing operating system. In this case a clock would be connected to the CLK input and OUT to an interrupt request line, possibly to a nonmaskable line. The GATE input would not be needed. When the system is brought up the initial count register would be filled with

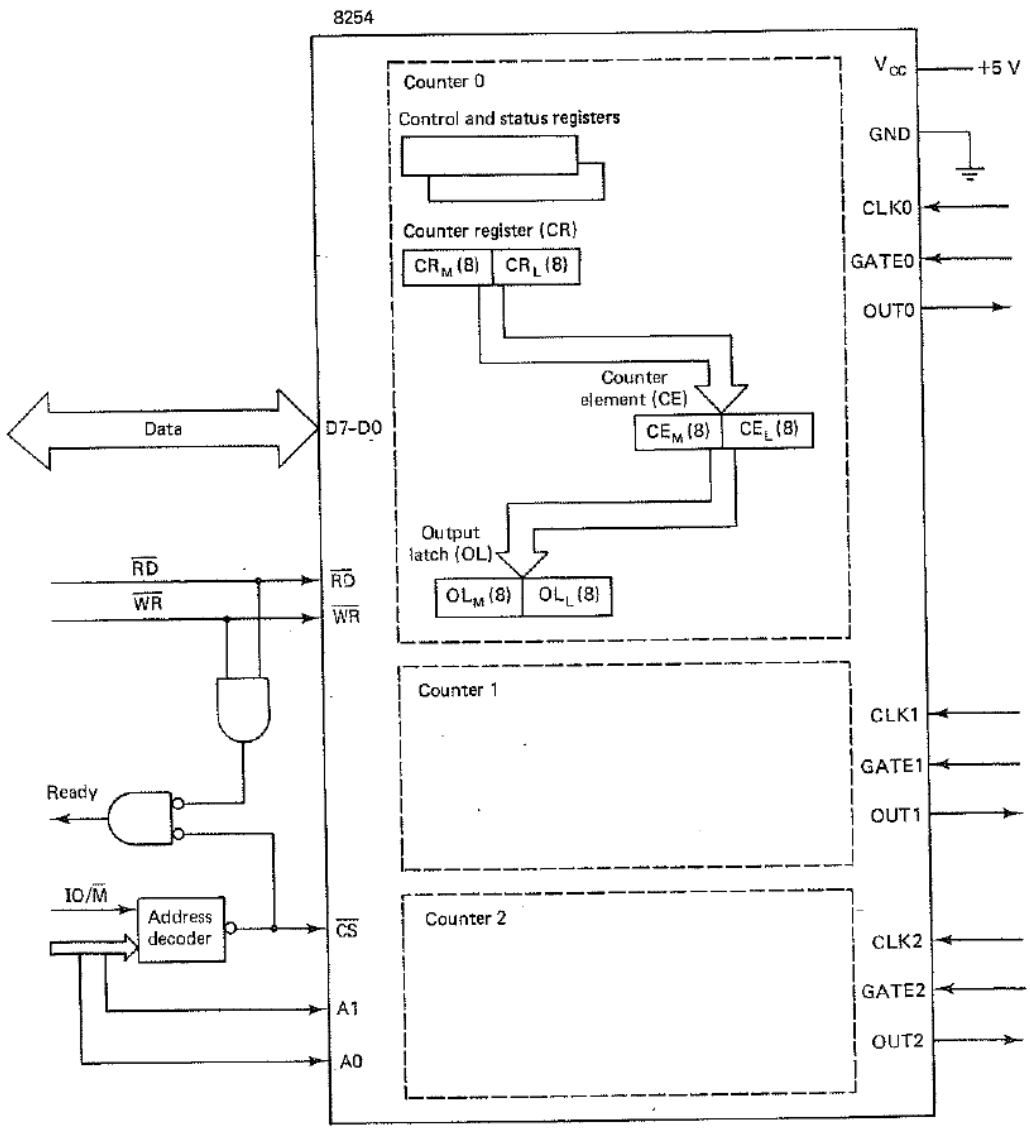
$$\text{Initial count} = \text{Clock frequency} \times T$$

where T is the length of each time slice in seconds, and the mode would be set so that each time the count reaches 0 the contents of the initial count register would be transferred to the counter and OUT would become active. Since OUT is used as an interrupt request, an interrupt routine for switching programs would be entered at the end of each period of T seconds.

9-3-1 Intel's 8254 Programmable Interval Timer

A diagram of Intel's 8254 interval timer/event counter is given in Fig. 9-26. The 8254 consists of three identical counting circuits, each of which has CLK and GATE inputs and an OUT output. Each can be viewed as containing a Control and Status Register pair, a Counter Register (CR) for receiving the initial count, a Counter Element (CE) which performs the counting but is not directly accessible from the

Figure 9-26 Diagram of the 8254.



processor, and an Output Latch (OL) for latching the contents of the CE so that they can be read. The CR, CE, and OL are treated as pairs of 8-bit registers. (Physically, the registers are not exactly as depicted, but to the programmer the figure is conceptually accurate.)

The registers can be accessed according to the following table:

$\overline{CS}$	$\overline{RD}$	$\overline{WR}$	A1	A0	Transfer
0	1	0	0	0	To counter 0 CR
0	1	0	0	1	To counter 1 CR
0	1	0	1	0	To counter 2 CR
0	1	0	1	1	To a control register or indicates a command
0	0	1	0	0	From counter 0 OL or status register
0	0	1	0	1	From counter 1 OL or status register
0	0	1	1	0	From counter 2 OL or status register

where 0 means low and 1 means high. All other combinations result in the data pins being put into their high-impedance state. When A1 = A0 = 1, whether a control register is being written into or a command is being given depends on the MSBs of the byte being output. For the last three combinations, whether an OL or status register is read is determined by a previous command.

There are two types of commands, the counter latch command, which causes the CE in the counter specified by the two MSBs of the command to be latched into the corresponding OL, and the read back command, which may cause a combination of the CEs to be latched or "prepare" a combination of status registers to be read. To prepare a status register means to cause it to be read the next time a read operation inputs from the counter. When the two MSBs are 00, 01, or 10 a counter latch command is indicated, but if they are 11 a read back is to be performed. In a latch command bits 5 and 4 must be 0 and the remaining bits are unused. The read back command has the format:

1	1	$\overline{COUNT}$	$\overline{STAT}$	CNT2	CNT1	CNT0	0

If the $\overline{COUNT}$ bit is 0, then the CEs for all of the counters whose CNT bits are 1 are latched. If CNT0 = CNT2 = 1 but CNT1 = 0, then the CEs in counters 0 and 2 are latched but the CE in counter 1 is not latched. Similarly, $\overline{STAT} = 0$ causes the counters' status registers to be prepared for input. CEs can be latched and status registers can be prepared in the same command.

The formats of the control and status registers are given in Fig. 9-27. If the two MSBs of an output are both 1, they indicate that the output is to be a read back command; otherwise, they specify a counter. If they specify a counter and bits 4 and 5 are both 0, then a latch command is indicated and it is directed to the control register of the counter specified by the top 2 bits, but if they are not both 0, then they indicate the type of the input from OL or output to CR. The combination 01 indicates that the Read/Write operations are from/to the OL_L/CR_L, 10 indicates that they are from/to the OL_M/CR_M, and 11 indicates that these operations are to occur in pairs, with the first byte coming from/going to OL_L/CR_L and the second from/to OL_M/CR_M. A 1-byte write to CR will cause the other byte to be

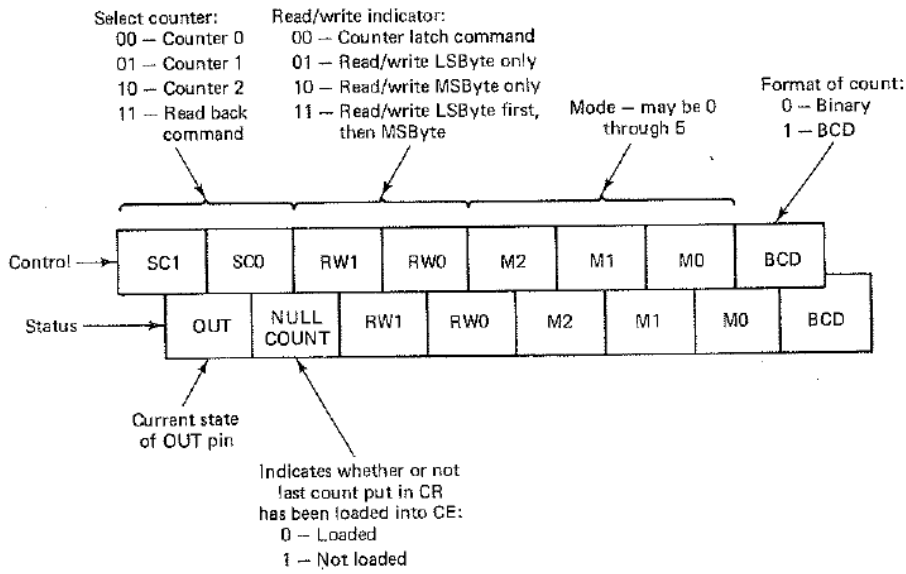


Figure 9-27 Control and status registers for 8254 counters.

zeroed. Bits 1, 2, and 3 determine the mode and bit 0 specifies the format of the count.

Given that N is the initial count, the modes are:

Mode 0 (Interrupt on Terminal Count)—GATE = 1 enables counting and GATE = 0 disables counting, and GATE has no effect on OUT. The contents of CR are transferred to CE on the first CLK pulse after CR is written into by the processor, regardless of the signal on the GATE pin. The pulse that loads CE is not included in the count. OUT goes low when there is an output to the control register and remains low until the count goes to 0. Mode 0 is primarily for event counting.

Mode 1 (Hardware Retriggerable One-Shot)—After CR has been loaded with N, a 0-to-1 transition on GATE will cause CE to be loaded, a 1-to-0 transition at OUT, and the count to begin. When the count reaches 0 OUT will go high, thus producing a negative-going OUT pulse N clock periods long.

Mode 2 (Periodic Interval Timer)—After loading CR with N, a transfer is made from CR to CE on the next clock pulse. OUT goes from 1 to 0 when the count becomes 1 and remains low for one CLK pulse; then it returns to 1 and CE is reloaded from CR, thus giving a negative pulse at OUT after every N clock cycles. GATE = 1 enables the count and GATE = 0 disables the count. A 0-to-1 transition on GATE also causes the count to be reinitialized on the next clock pulse. This mode is used to provide a programmable periodic interval timer.

Mode 3 (Square-Wave Generator)—It is similar to mode 2 except that OUT

goes low when half the initial count is reached and remains low until the count becomes 0. Hence the duty cycle is changed. As before, GATE enables and disables the count and a 0-to-1 transition on GATE reinitializes the count. This mode may be used for baud rate generation.

Mode 4 (Software-Triggered Strobe)—It is similar to mode 0 except that OUT is high while the counting is taking place and produces a one-clock-period negative pulse when the count reaches 0.

Mode 5 (Hardware-Triggered Strobe—Retriggerable)—After CR is loaded, a 0-to-1 transition on GATE will cause a transfer from CR to CE during the next CLK pulse. OUT will be high during the counting but will go low for one CLK period when the count becomes 0. GATE can reinitialize counting at any time.

For all modes, if the initial count is 0, it will be interpreted as 2^{16} or 10^4 depending on the format of the count. The above descriptions were only to provide an overall idea of the operation of the 8254 in the various modes. For a detailed description of the modes, the reader should look in Reference 2.

9-3-2 Interval Timer Application to A/D

Figure 9-28 shows how an 8254 could be used to provide a programmable sample rate generator for an A/D subsystem. By using all three of the 8254's counters, not only can the program set the sample rate, but it can also determine the period over which the samples are taken. Suppose that counter 0 is put in mode 2, counter 1 in mode 1, and counter 2 in mode 3 and that L, M, and N are their initial counts. If F is the frequency of the clock, then the frequency applied to CLK1 will be F/N. This will result in OUT1 producing a pulse having a period of MN/F. Therefore, pulses will occur at OUT0 at a frequency of F/L for a period of MN/F seconds. By applying OUT0 to the Convert input of the A/D converter, F/L samples per second will be taken for MN/F seconds beginning after the three counters have been initialized and the relay or manually operated switch has been closed. After each converted sample has been transmitted to port A of the 8255A, an interrupt request is made through the $INTR_A$ and an interrupt routine is used to input the sample. An initialization sequence for the system is given in Fig. 9-29. The sequence assumes that the addresses associated with the 8254 are 0070 through 0073; LCNT, MCNT, and NCNT contain L, M, and N; and L and N are less than 256. It sets the counters to their modes and puts the initial counts in the CRs. The initial counts L and N are to be in binary and M is to be in BCD.

9-4 KEYBOARD AND DISPLAY

For low-cost small systems, especially single-board microcomputers and microprocessor-based instruments, the front panel (or *console*) is often implemented by using simple keyboard and display units as input and output devices. Through a

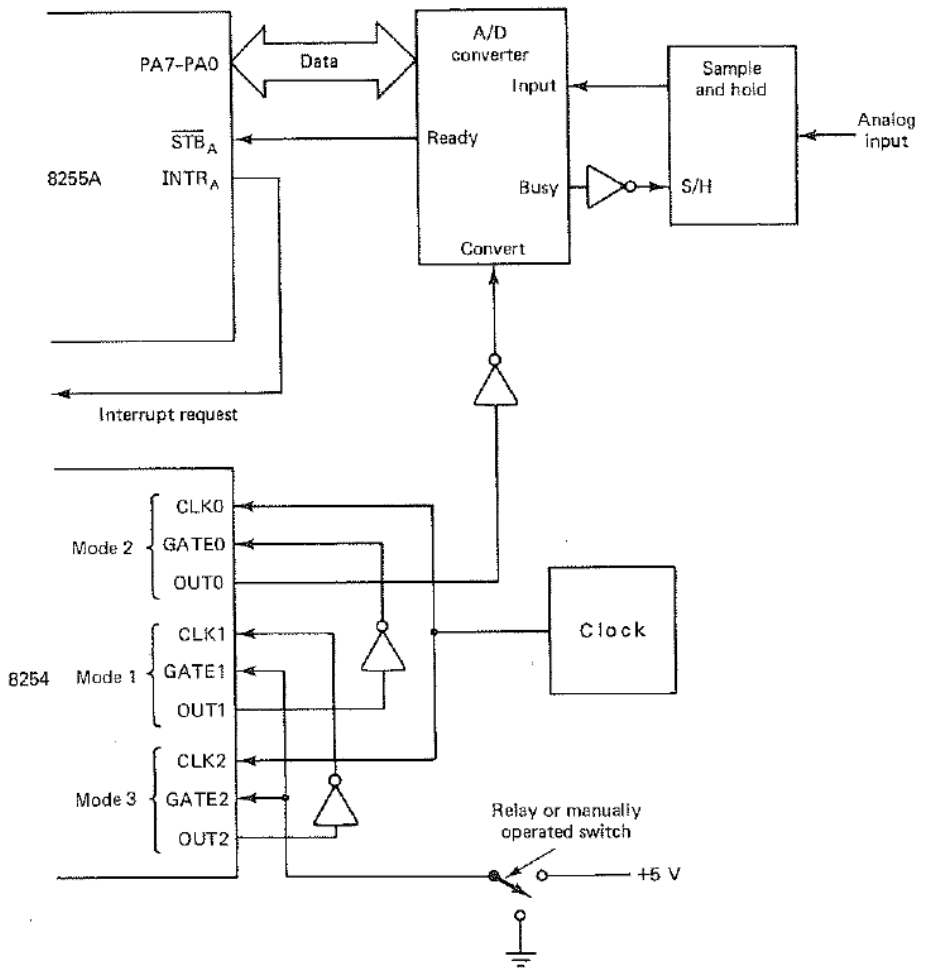


Figure 9-28 Interval timer example.

```
MOV   AL,00010100B      ;OUTPUT COUNTER 0
OUT   73H,AL            ;CONTROL - MODE 2
MOV   AL,LCNT           ;OUTPUT COUNTER 0
OUT   70H,AL            ;INITIAL COUNT - BINARY
MOV   AL,01110011B      ;OUTPUT COUNTER 1
OUT   73H,AL            ;CONTROL - MODE 1
MOV   AX,MCNT           ;OUTPUT COUNTER 1
OUT   71H,AL            ;INITIAL COUNT - BCD
MOV   AL,AH
OUT   71H,AL
MOV   AL,10010110B      ;OUTPUT COUNTER 2
OUT   73H,AL            ;CONTROL - MODE 3
MOV   AL,NCNT           ;OUTPUT COUNTER 2
OUT   72H,AL            ;INITIAL COUNT - BINARY
  .
  .
  .
```

Figure 9-29 Initialization of counters for the A/D example.

keyboard, data, memory addresses, and machine code can be entered in the hexadecimal form. In addition to the numeric keys, a keyboard may include functional keys to enter monitor and control commands. For output, memory addresses and data can be displayed on light-emitting diode (LED) display devices.

9-4-1 Keyboard Design

Unlike a terminal, a mechanical contact keyboard, for which the key switches are organized in a matrix form, does not include any electronics. Figure 9-30 illustrates how a 64-key keyboard can be interfaced to a microcomputer through two parallel I/O ports such as those provided by an 8255A. When a key is depressed, the corresponding row and column are shorted to form a path. By detecting the row and column positions of the contact closure, the code word representing the depressed key is determined. This process is called keyboard scanning and is accomplished as follows. The output port sends a 0 to row 0 and 1s to all 7 of the other rows. The column lines are then read and checked. If a 0 is not found in row 0, then the process is repeated for row 1, then for row 2, and so on. When a 0 is found, a depressed key is detected whose row position is known from the combination that was output and column position is known from the input. By combining the row and column positions of the 0, a unique word which indicates the position of the depressed key can be found.

There are two major problems associated with keyboards; they are contact bounce and striking keys at about the same time. When a key is depressed or released, the contact may bounce between its open and close position several times before it settles to a close or open position. The duration of the bouncing varies and is normally less than 10 ms. Contact closures due to bouncing must be discarded to prevent false key detection and this operation is called *debouncing*. Debouncing can be accomplished through either hardware or software, but the software approach requires too much processor time for most applications. Multiple key closings are most easily handled by scanning the entire array and inputting the closures in the order they are detected.

9-4-2 Display Design

Various types of devices are available for numeric and alphanumeric displays. Seven-segment LED displays such as the one shown in Fig. 9-31 are typically used for hexadecimal digit display. A digit in seven-segment code is fed to the input pins of a through g and DP. The input can be represented by active low or active high signals, depending on whether the display unit is of a common anode or a common cathode type. An individual segment is lit when the corresponding LED is forward biased.

Figure 9-32 shows a multiple-digit display that is configured from eight seven-segment display units. In order to reduce the device count by eliminating external data latches from the display units, they can be connected to two 8-bit parallel

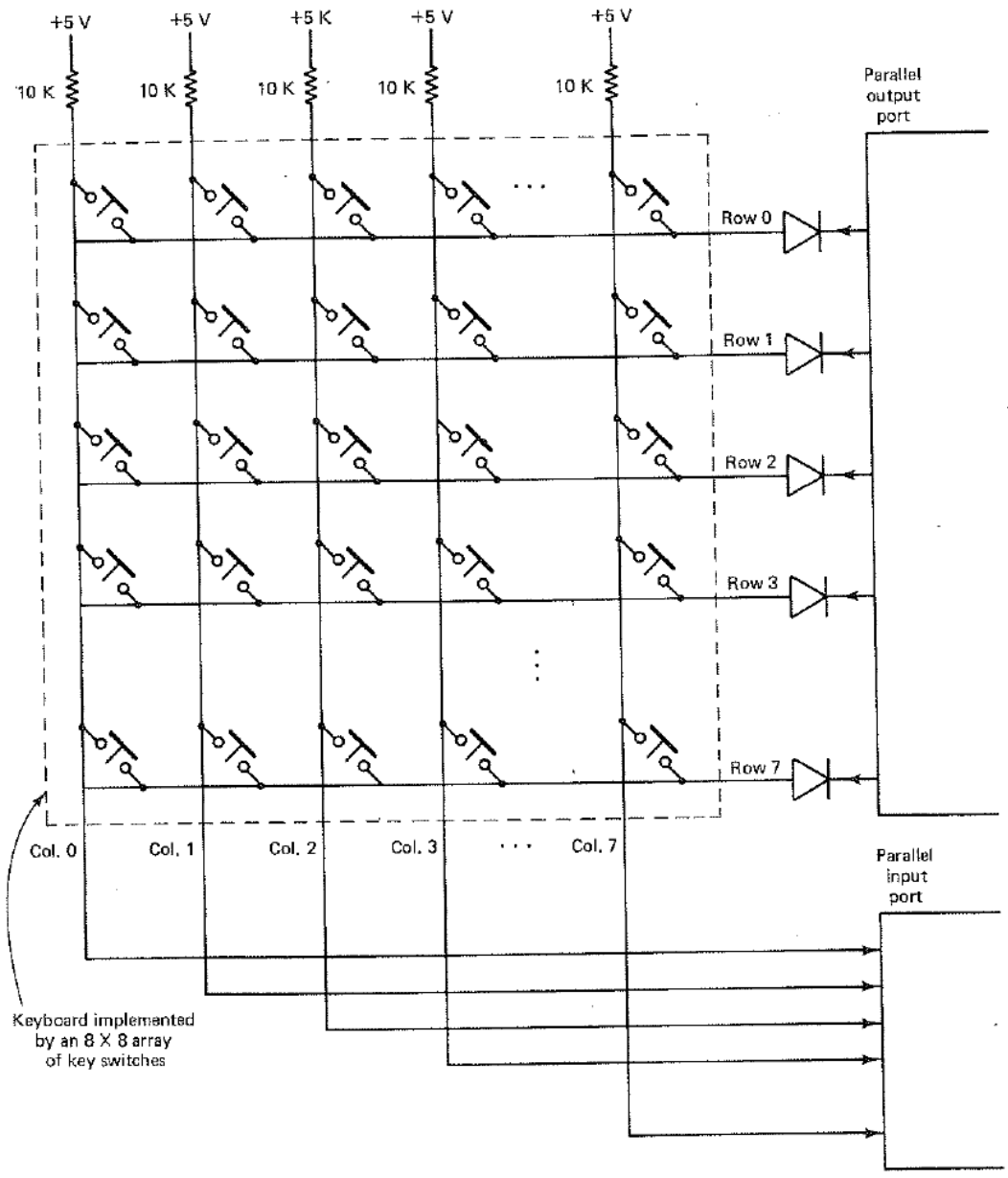
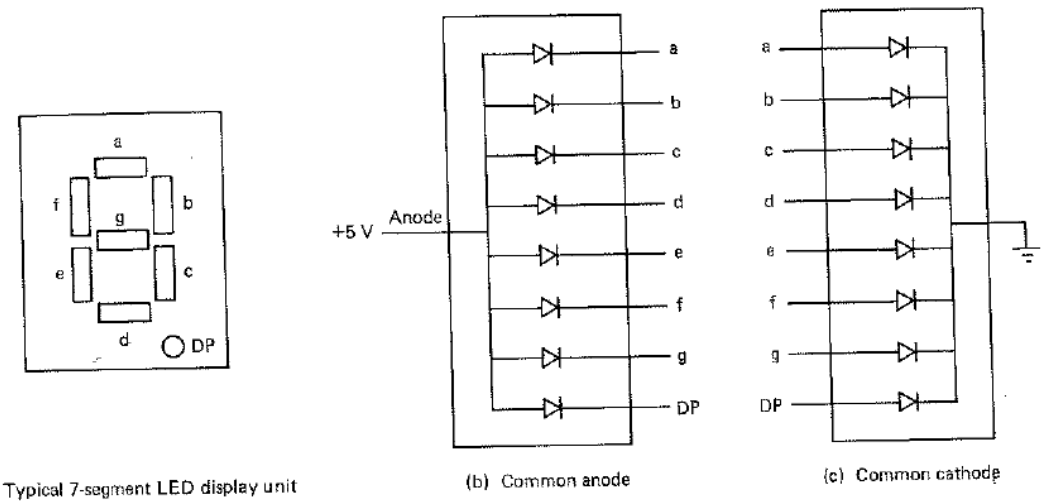


Figure 9-30 Organization of a mechanical keyboard.

output ports and operated in a multiplexed mode. All display units share the common segment lines and only one unit is selected at a time through the digit select lines. Each display is driven for 1 ms and after rotating through all eight units, the first one is returned to and the sequence is repeated. Thus, the displays



(a) Typical 7-segment LED display unit (b) Common anode (c) Common cathode

Figure 9-31 Seven-segment LED display unit.

are repeatedly refreshed to give the illusion of a continuously lit multiple-digit display.

Since a typical average current through each turned-on segment could be as high as 20 mA and more than one segment may be lit per digit, the resulting current exceeds the driving capability of a regular TTL gate. Therefore, digit drivers and segment drivers are needed. A low on the base of a digit driver transistor turns the transistor on and effectively connects that display unit to +5 V. A low on the base of a segment driver transistor turns on the transistor, thus letting current flow throught the segment. In addition, since the voltage drop across a forward-biased segment diode is constant, a resistor in series with each segment is required to limit the current, thus protecting the display and drivers. The design resistance of these resistors depends on the desired display brightness.

Another type of hexadecimal digit display is the Texas Instruments (TI) TIL311 shown in Fig. 9-33, which uses a matrix-dot array of 20 LEDs. It inputs a 4-bit binary number and internally decodes the digit input to light the LEDs corresponding to the equivalent hexadecimal digit. Since it has built-in latches and constant current drivers, the input is TTL compatible, thus eliminating a need for external drivers and current-limiting resistors. Also, the input latch holds data for constant display and refresh is no longer required.

9-4-3 Keyboard/Display Controller

Although interfacing a keyboard and multiple-digit display to parallel I/O ports is simple from a hardware standpoint, while inputting and outputting the processor would be tied up by keyboard scanning and display refreshing routines. The Intel 8279 keyboard/display controller is an LSI device designed to release the processor from performing the time-consuming scan and refresh operations.

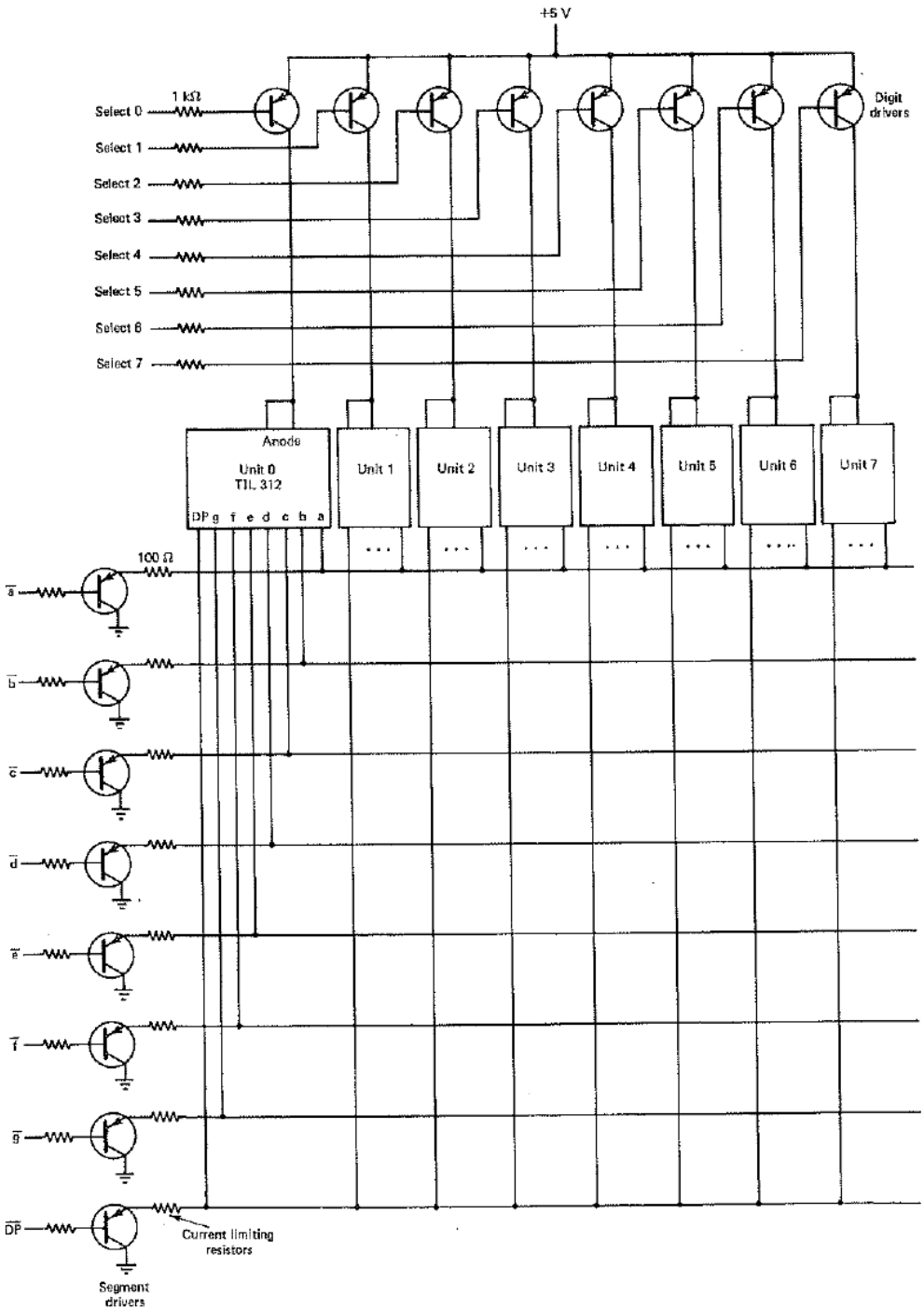


Figure 9-32 Eight-digit display.

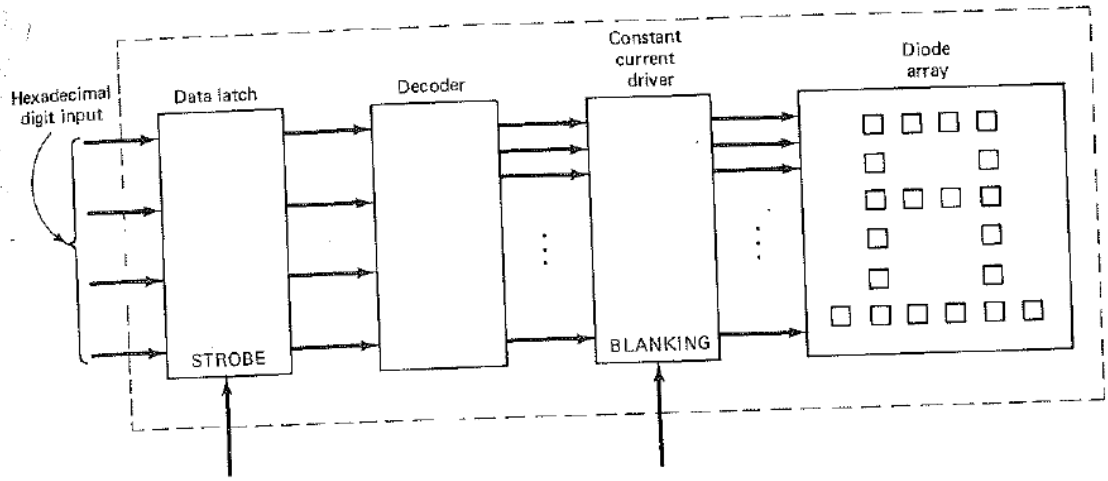


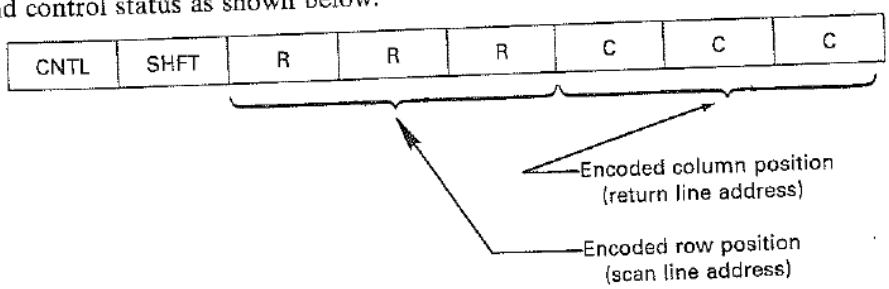
Figure 9-33 TI TIL311 matrix-dot hexadecimal display.

Figure 9-34 shows the general structure of the 8279 and its interface to the bus. The control and status registers share the odd address and the data buffer register uses the even address. The addressing is according to the following table:

$\overline{CS}$	$\overline{RD}$	$\overline{WR}$	A0	Transfer Description
0	1	0	0	Data bus to data buffer register
0	1	0	1	Data bus to control register
0	0	1	0	Data buffer register to data bus
0	0	1	1	Status register to data bus

where 1 means high and 0 means low.

For keyboard control, the 8279 constantly scans each row of the keyboard by sending out row addresses on SL2-SL0 and inputting signals on the return lines RL7-RL0, which represent the column addresses. Note that the SL3-SL0 lines are used for both keyboard scanning and display refreshing and will accommodate up to 16 display units. When a depressed key is detected, the key is automatically debounced by waiting 10 ms to check if the same key remains depressed. If a depressed key is detected an 8-bit code word corresponding to the key position is assembled by combining the encoded column position, row position, shift status, and control status as shown below.



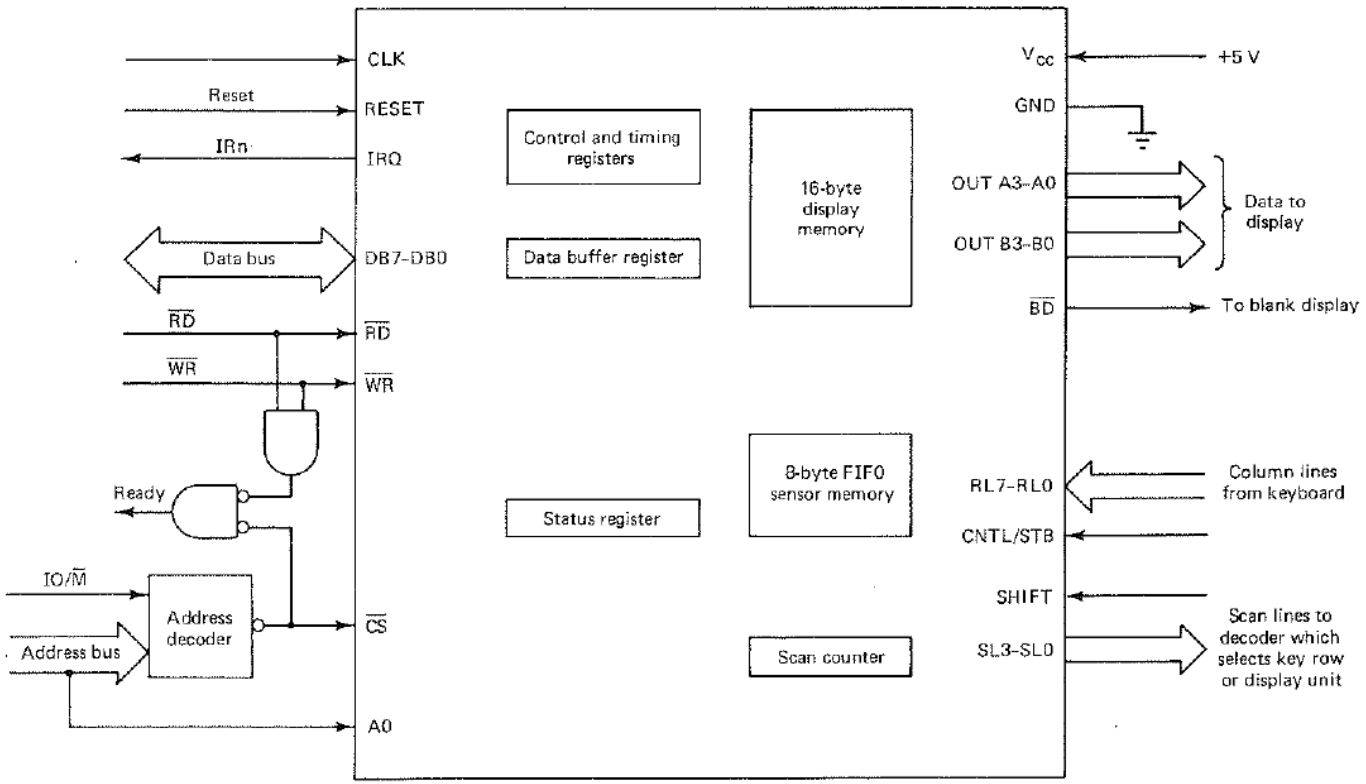
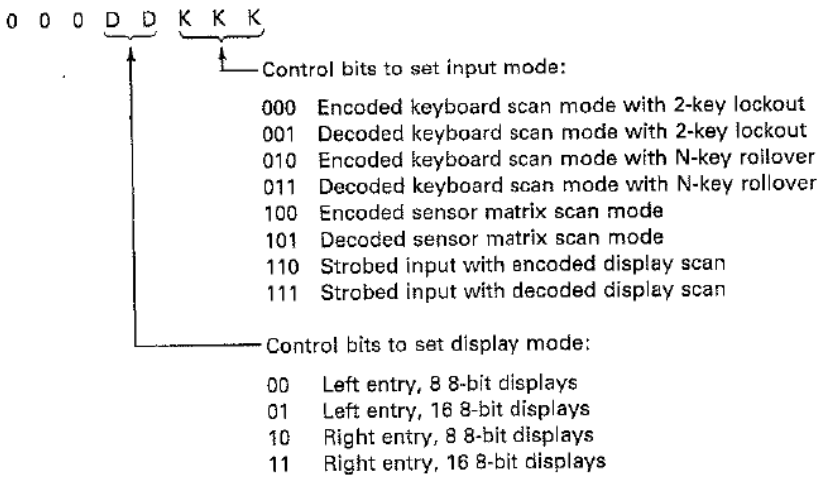


Figure 9-34 Structure of the 8279.

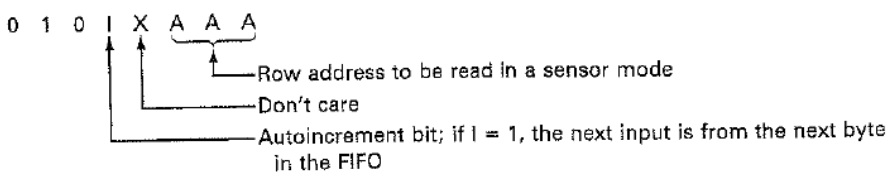
The SHFT and CNTL pins are used primarily to support typewriter-like keyboards which have shift and control keys. The key position is then entered into the 8 × 8 first-in/first-out (FIFO) sensor memory, and the IRQ (interrupt request) line is activated if the sensor memory was previously empty.

The control and timing registers are physically a collection of flags and registers that are accessed by commands which are sent to the 8279's odd address. The three MSBs of a command determine its type and the meaning of the remaining 5 bits depends on the type. Although there are eight types, only three of them are considered here. (For the other types as well as other detailed information about the 8279, see Reference 2.) The formats of the three commands of interest to us are:

Keyboard Display Mode Set—Specifies the input and display modes and is used to initialize the 8279. Its format is:



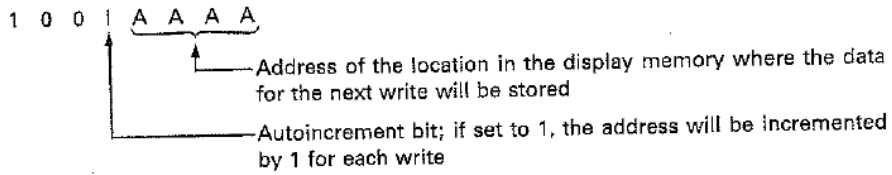
Read FIFO Sensor Memory—Specifies that a read from the data buffer register will input a byte from the FIFO memory and, if the 8279 is in the sensor mode, it indicates which row is to be read. This command is required before inputting data from the FIFO memory. Its format is:



Note that if the input mode is a keyboard scan mode, a read is always from the byte which first entered the FIFO, hence the I and AAA bits are ignored.

Write to Display Memory—Indicates that a write to the data buffer register will put data in the display memory. This command must be given before the

CPU can send the characters to be displayed to the 8279. Its format is:



The 8279 provides two options for handling the situation in which more than one key is depressed at about the same time. With the two-key lockout option, if another key is depressed while the first key is being debounced, the key which is released last will be entered into the FIFO. If the second key is depressed within two scan cycles after the first is debounced and the first key remains depressed after the second one is released, then the first depressed key is recognized. Otherwise, both are ignored. The N-key rollover option treats each key depression independently. If more than one is depressed, after they are depressed they are all entered in the order they were sensed.

In addition, the 8279 has a sensor matrix mode under which signals on the return lines are stored into the FIFO at the row corresponding to the scan address. Unlike a keyboard scan mode, the SHFT and CNTL status and the scan address are not entered. This mode keeps an image of the status of the sensor matrix and is useful when information from several devices is input by polling each device through the scan lines.

The status of the FIFO is kept in the status register. Bits 0, 1, and 2 of this register give the number of data bytes currently in FIFO memory and bit 3 = 1 indicates that this memory is full. Bits 4 and 5 indicate underflows, which occur when an attempt is made to read from an empty FIFO memory, and overflows, which occur when an input to a full FIFO memory is attempted. In both cases a 1 signifies the presence of the error. A 1 in bit 6 reflects a closure when the 8279 is in its sensor matrix mode and a multiple closure when it is in special error mode. Bit 7 shows whether or not the display is available.

For display control, the 8279 provides a 16-byte display memory and refresh logic. Each address in the display memory corresponds to a display unit, with address 0 representing the leftmost display unit. Once the CPU loads the characters to be displayed into this memory, the 8279 needs no further instructions and the processor is released from refreshing the display units. The output is accomplished by the 8279 repeatedly sending out characters over the lines OUT A3–A0 and OUT B3–B0 and unit select addresses over SL3–SL0. Although the display memory can be directly addressed, the processor may sequentially enter data to the display memory either from the left or from the right. For the autoincrement left entry, after each write to the display the address is incremented by 1, so that the next character appears in the next display unit to the right. The autoincrement right entry allows characters to be displayed in the form used by many electronic calculators. It causes the display to be shifted left one character and stores the next character from the right.

Figure 9-35 illustrates one way in which a 64-key keyboard and an eight-digit

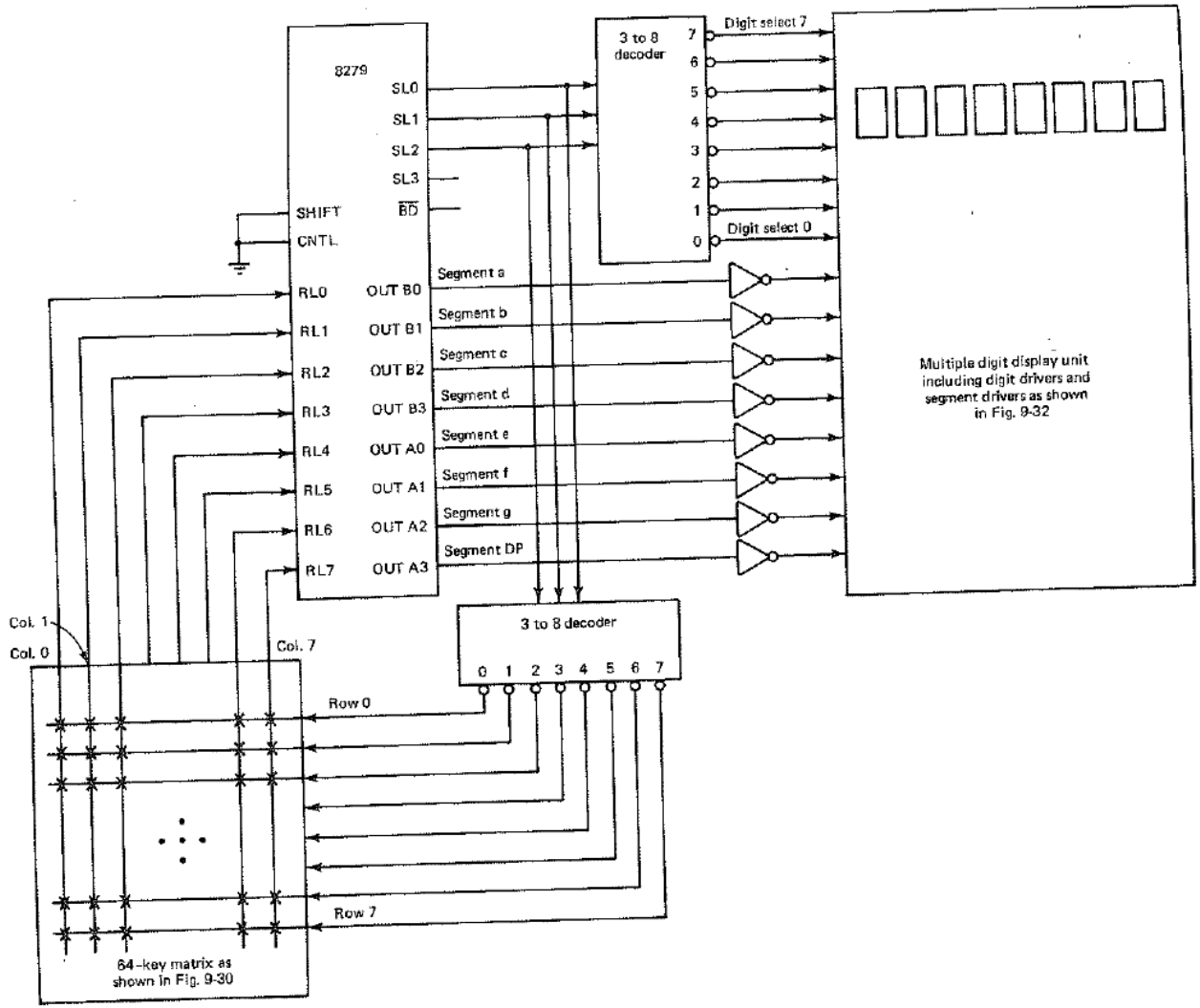


Figure 9-35 Use of an 8279 to interface a keyboard and a multiple-digit display.

seven-segment display can be connected to an 8279. Both the keyboard and display are scanned and refreshed under the control of the select signals SL2–SL0. The SL3 pin is not connected because there are only eight display units. Both 3-to-8 decoders have active low outputs. One decoder selects one row of keys while the other decoder enables one of the eight digit drivers.

To demonstrate how to program an 8279, let us assume that the device is connected to a keyboard and multiple-digit display as shown in Fig. 9-35, the 8279's addresses are FFE8 and FFE9, and the interrupt request pin IRQ is not used. First, the device must be initialized by sending a mode set command to the control register. The following instructions set the keyboard/display controller to its encoded keyboard scan mode, with two-key lockout, and its left entry eight 8-bit displays mode:

```
MOV   DX,0FFE9H
MOV   AL,0
OUT   DX,AL
```

Then, characters generated by the depressed keys can be read through the FIFO memory. A program segment that uses programmed I/O to input eight keywords and store them in an 8-byte array KEYS with the first byte at the highest address is:

```
      MOV   SI,8
      MOV   DX,0FFE9H
      MOV   AL,01000000B
      OUT   DX,AL
NEXT: MOV   DX,0FFE9H
IDLE: IN    AL,DX
      TEST  AL,0FH
      JZ    IDLE
      MOV   DX,0FFE8H
      IN    AL,DX
      MOV   KEYS[SI-1],AL
      DEC   SI
      JNZ   NEXT
```

The first instruction puts the count in SI, the next three instructions cause the input from the even address to come from the FIFO, the three instructions beginning at IDLE force the processor to idle until an input is ready, and the following three instructions transfer the input data to KEYS. The last two instructions cause the sequence to repeat until eight characters have been read.

To display characters, the CPU must first give a write display memory command and then output to the display memory. The following instruction sequence displays eight seven-segment digits which are stored beginning at DIGITS with the least significant digit being stored at the low address:

```
MOV   SI,8
MOV   DX,0FFE9H
MOV   AL,10010000B
OUT   DX,AL
MOV   DX,0FFE8H
```

```
AGAIN:  MOV   AL,DIGITS[SI-1]
        OUT   DX,AL
        DEC   SI
        JNZ   AGAIN
```

The first instruction puts the digit count in SI, the following three instructions output the write display memory command, and the next instruction puts the address of the data buffer register in DX. The loop outputs the digits to the display memory.

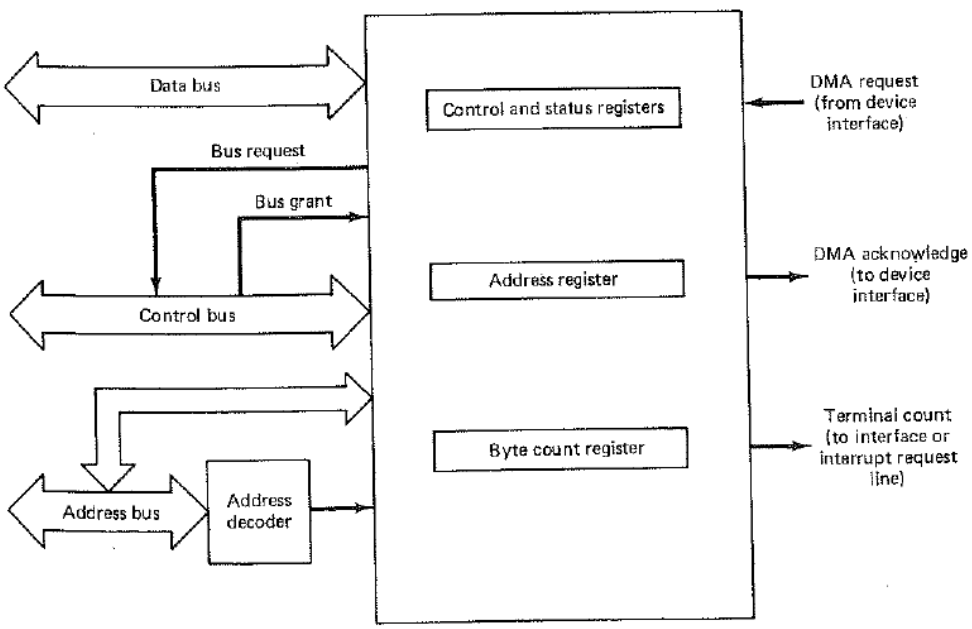
9-5 DMA CONTROLLERS

As discussed in Chap. 6, a DMA controller is capable of becoming the bus master and supervising a transfer between an I/O or mass storage interface and memory. While making a transfer, it must be able to place memory addresses on the bus and send and receive handshaking signals in a manner similar to that of the bus control logic. The purpose of a DMA controller is to perform a sequence of transfers (i.e., a block transfer) by stealing bus cycles.

A DMA controller is designed to service one or more I/O or mass storage interfaces, and each interface is connected to the controller by a set of conductors. A portion of a DMA controller for servicing a single interface is called a *channel*.

The general organization of a one-channel DMA controller and its principal connections is shown in Fig. 9-36. In addition to the usual control and status registers, each channel must contain an address register and a byte (or word) count

Figure 9-36 General organization of a DMA controller.



register. Initializing the controller consists of filling these registers with the beginning (or ending) address of the memory array that is to be used as a buffer and the number of bytes (words) to be transferred. For an input to memory, each time the interface has data to transfer it makes a DMA request. The controller then makes a bus request and, when it receives a bus grant, it puts the contents of the address register on the address bus, sends an acknowledgment back to the interface, and issues I/O read and memory write signals. The interface then puts the data on the data bus and drops its request. When the memory accepts the data it returns a ready signal to the controller, which then increments (or decrements) the address register, decrements the byte (word) count, and drops its bus request. Upon the count reaching zero, the process stops and a signal is sent to the processor as an interrupt request or to the interface to notify it that the transfers have terminated. An output is similarly executed, except that the controller issues I/O write and memory read signals and the data are transferred in the other direction.

As an example, let us consider the Intel 8237 DMA controller, whose overall organization along with that of the associated logic needed in an 8088 system is given in Fig. 9-37. A minimum mode 8088 configuration containing an 8237 is shown in Fig. 9-38.

When data are being put in or taken out of the 8237's registers, the 8237 is a slave just like the system's I/O interfaces. As a slave it receives 16-bit addresses with the 12 MSBs of these addresses determining whether or not the chip is selected and the 4 LSBs being used for internal addressing. When both HRQ and $\overline{CS}$ are low, the 8237 becomes a slave with the $\overline{IOR}$ and $\overline{IOW}$ being the input control pins. The CPU can read from or write to the internal registers of the controller by activating $\overline{IOR}$ or $\overline{IOW}$. The AEN, which is active when the controller is a master and is outputting an address, is 0 while the system is communicating with the controller's registers.

If the controller is the master, then it must supply the bus address. When it is master it puts the low-order byte of the address on the pins A7–A0 and the high-order byte on DB7–DB0, and sets AEN to 1. With AEN = 1, the outputs of the external address latch are enabled, thereby allowing the upper address byte to be put on the A15–A8 lines. The AEN line is also used to disable the 8282 address latches connected to the CPU's A19–A8 and AD7–AD0 pins. Shortly after the AEN signal is activated a pulse is sent out over the Address Strobe (ADSTB) pin. This pulse is used to strobe bits 8 through 15 of the address into the 8282 address latch connected to A15–A8. The upper 4 bits of the address, A19–A16, cannot be obtained from the 8237, but must be programmed separately before the block transfer begins. A 4-bit I/O port, which can be output to just like any other I/O port, is needed to hold the high-order 4 bits of the address. During a block transfer, the contents of this port should not be changed; therefore, any single transfer is limited to 2^{16} bytes.

While it is master the controller must also output the necessary read/write commands. These commands are $\overline{IOR}$, $\overline{MEMR}$, $\overline{IOW}$, and $\overline{MEMW}$ and indicate an I/O read, a memory read, an I/O write, and a memory write, respectively. Because these signals do not match the $\overline{RD}$, $\overline{WR}$, and IO/$\overline{M}$ signals output by a

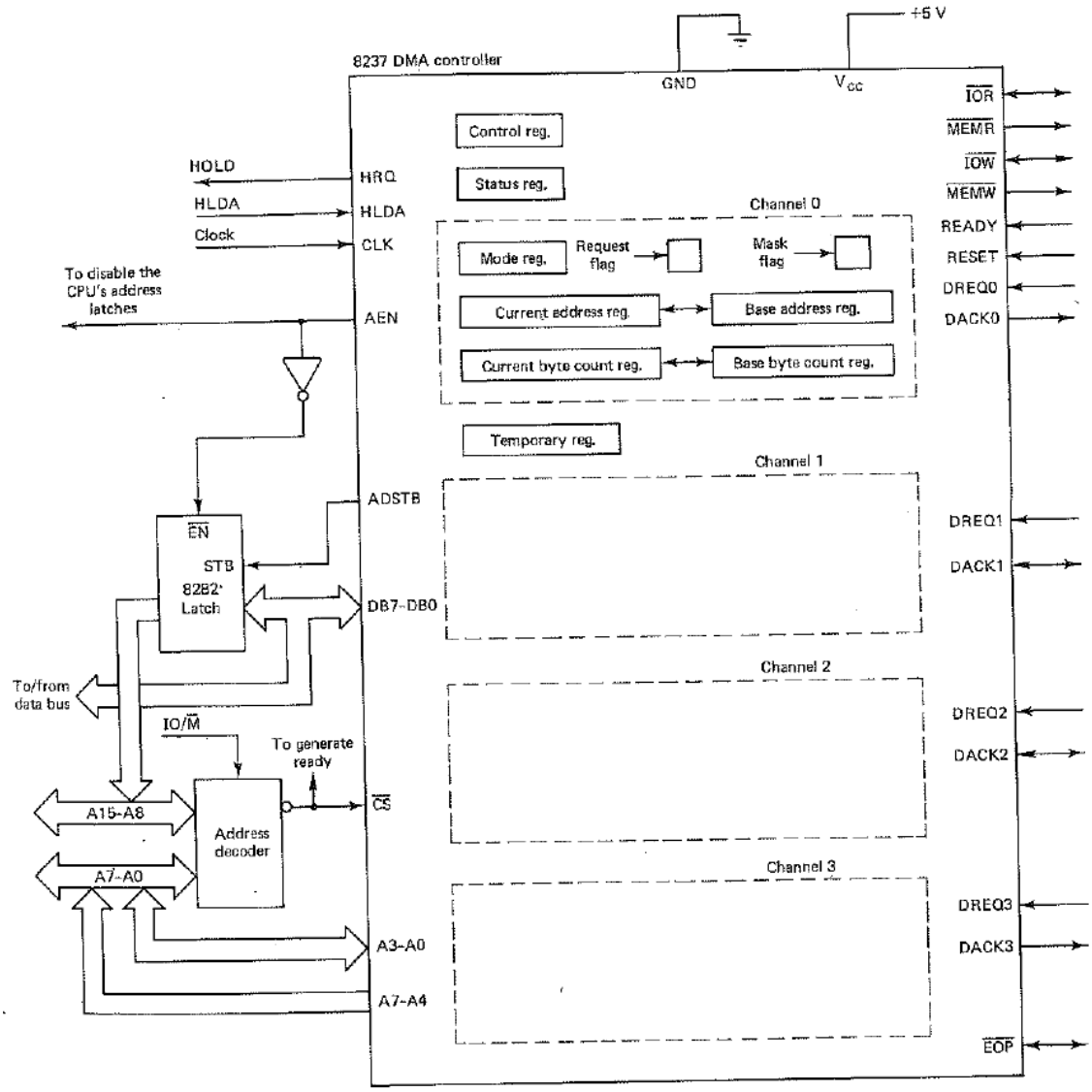


Figure 9-37 Organization of an 8237 and its associated logic.

minimum mode-8088, a read/write encoding circuit such as the one shown in Fig. 9-39 is needed to perform the translation between the two sets of signals. During a DMA transfer, the controller disables the outputs of the read/write encoder from the command lines by activating the AEN signal. Of course, memory and I/O interfaces in the system must be designed so that they respond to $\overline{MEMR}$, $\overline{MEMW}$, $\overline{IOR}$, and $\overline{IOW}$ instead of $\overline{RD}$, $\overline{WR}$, and IO/$\overline{M}$. As with the processor timing, the READY signal is used to extend the bus cycles by inserting wait states when servicing slow devices. A RESET signal clears the control, status, and temporary registers and the request flags, and sets the mask flags. The $\overline{EOP}$ pin is bidirectional. The count going to zero causes a negative pulse to be output through $\overline{EOP}$, which

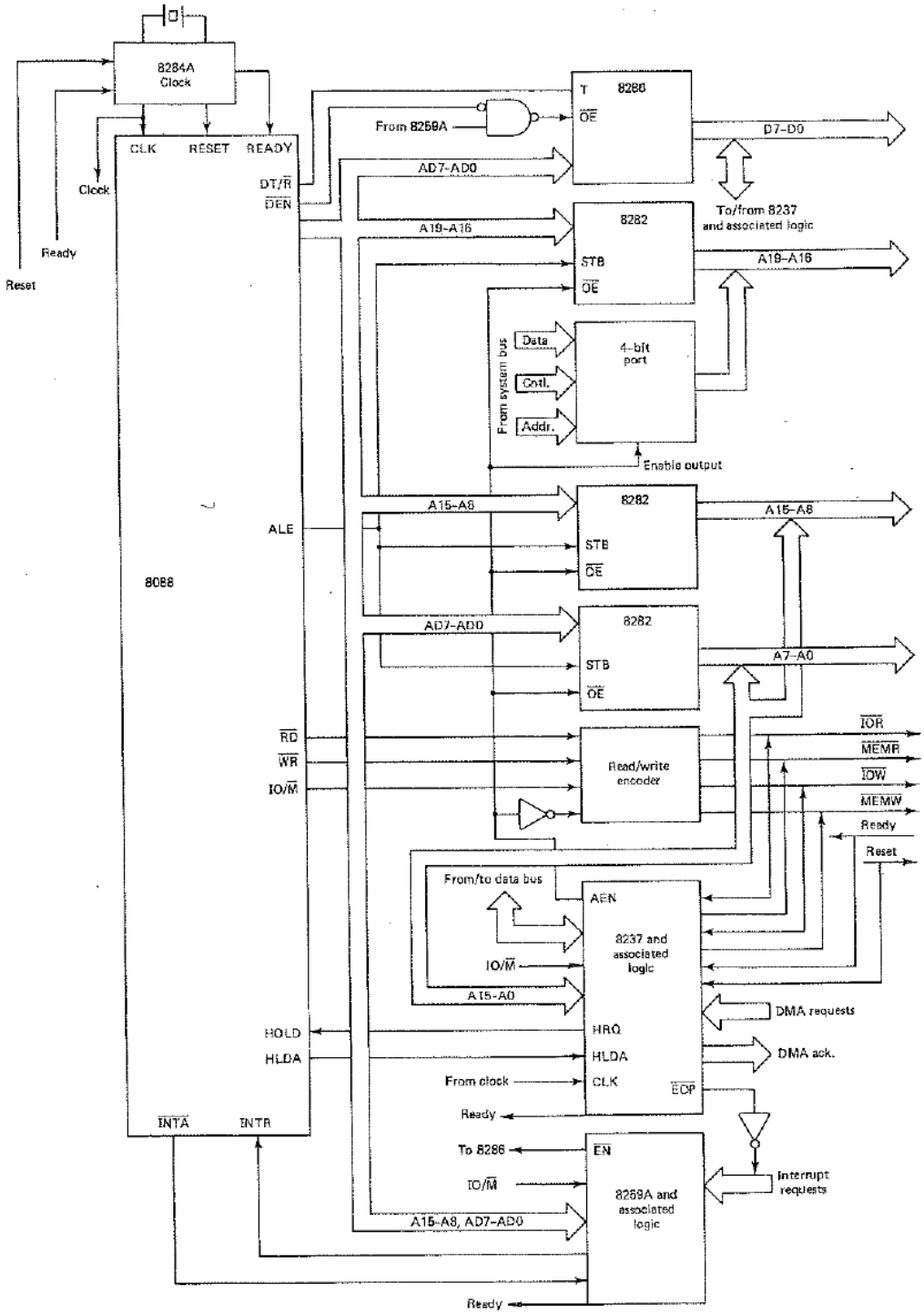
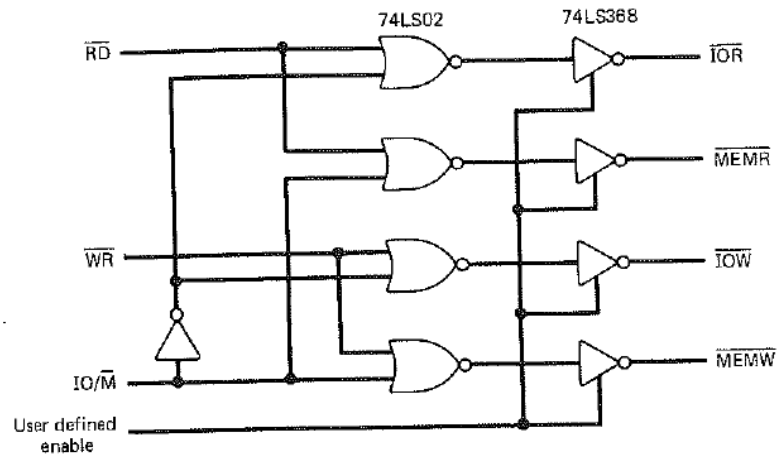


Figure 9-38 Minimum mode 8088 configuration that includes an 8237.



Note: If it is not necessary to three-state the command lines, a decoder (8205 or 74S138) could be used. The 74LS257 is not recommended since the outputs may experience voltage spikes when entering or leaving three state.

Figure 9-39 Encode logic for $\overline{RD}$, $\overline{WR}$, and IO/$\overline{M}$ signals. (Reprinted by permission of Intel Corporation. Copyright 1979.)

can be used either as an interrupt request or by the device interface. On the other hand, the interface may send an active low signal to the 8237 through this pin. In either case, a low on the $\overline{EOP}$ will result in the block transfer being suspended. After the suspension the block transfer may resume according to the mode of the 8237.

An 8237 includes control, status, and temporary registers, and four channels, each containing a mode register, current address register, base address register, current byte count register, base byte count register, request flag, and mask flag. Each channel may be put in one of four modes, with its current mode being determined by bits 7 and 6 of the channel's mode register. The four possible modes are:

Single Transfer Mode (01)—After each transfer the controller will release the bus to the processor for at least one bus cycle, but will immediately begin testing for DREQ inputs and proceed to steal another cycle as soon as a DREQ line becomes active.

Block Transfer Mode (10)—DREQ need only be active until DACK becomes active, after which the bus is not released until the entire block of data has been transferred.

Demand Transfer Mode (00)—This mode is similar to the block mode except that DREQ is tested after each transfer. If DREQ is inactive, transfers are suspended until DREQ once again becomes active, at which time the block transfer continues from the point at which it was suspended. This allows the interface to stop the transfer in the event that its device cannot keep up.

Cascade Mode (11)—In this mode 8237s may be cascaded so that more than four channels can be included in the DMA subsystem. In cascading the controllers, those in the second level are connected to those in the first level by joining HRQ to DREQ and HLDA to DACK. To conserve space, this mode will not be considered further.

In all cases the count going to zero will cause $\overline{\text{EOP}}$ to become active and the transfer process to cease.

Bit 5 of the mode register specifies whether the contents of the address register are to be incremented (0) or decremented (1) after each transfer, thus determining the order in which the data are stored in memory. If bit 4 is 1, then autoinitialization is enabled. When the current address and current byte count registers are initially loaded their contents are also put in the base address and base byte count registers. If autoinitialization is enabled, then the current registers are automatically reloaded from the base registers whenever the count goes to zero. Bits 2 and 3 indicate the type of transfer to be made. There are three types: verify (00), write (01), and read (10). The verify type is for verifying information concerning the previous input or output operation and is not actually associated with a current transfer. It is of little interest to us and will not be discussed further. The two LSBs of an output to a mode register direct the output to the indicated channel, i.e., select the mode register that is to receive the output.

In addition to block transfers between I/O or mass storage devices and memory, the 8237 can supervise memory-to-memory transfers. Such transfers are conducted by bringing bytes from the source memory area into the 8-bit temporary register in the 8237 and then outputting them to the destination memory area. Therefore, two bus cycles are required for each memory-to-memory transfer. The channel 0 current address register is used for source addressing. The channel 1 current address and current byte count registers provide the destination addressing and counting. The destination address must increment or decrement as usual, but by setting the proper bit in the control register the source address may be held constant. This allows the same data byte to be transferred into the entire destination array.

With regards to the control register, a memory-to-memory transfer is enabled by setting bit 0 to 1, in which case bit 1 = 1 indicates that the source address is to be held constant. Bit 2 is used for enabling (0) or disabling (1) the controller and bit 3 specifies the type of timing. If the speed characteristics of the system permit, then bit 3 can be set to 1 to indicate compressed timing. With compressed timing only two clock cycles are needed to perform most transfers. (Timing is discussed below.) Bit 4 determines whether the priority is fixed or rotating. Normally, channel 0 has the highest priority and channel 3 the lowest; however, if bit 4 is 1, the priority will rotate after each transfer, e.g., if the priority before a transfer is 2–3–0–1, then after the transfer it will be 3–0–1–2. By rotating the priority the controller can prevent one channel from dominating the bus. The 8237 permits the program to specify the length of the $\overline{\text{IOW}}$ and $\overline{\text{MEMW}}$ signals when normal

timing is being used. A 1 in bit 5 indicates that these signals are to be extended over two clock cycles. The program can also specify whether the DREQ and DACK pins are to be active high or active low by setting or clearing bits 6 (DREQ) and 7 (DACK), respectively. Bit 6 = 1 indicates that DREQ is active low and bit 7 = 1 indicates that DACK is active high. How these bits should be set depends on the characteristics of the associated interfaces.

The format of the status register is such that the lower 4 bits indicate the states of the terminal counts of the four channels and the upper 4 bits show the current presence or absence of DMA requests. For the lower 4 bits a 1 in bit n indicates that the terminal count for channel n is active, and for the upper four bits a 1 in bit $n + 4$ signals the presence of a request on channel n.

Each channel also has associated with it a request flag and a mask flag. A DMA request can be programmed as well as input through the DREQ pin. Setting the request flag for a channel has the same effect as the DREQ pin becoming active, and it is cleared when $\overline{\text{EOP}}$ goes active. When set to 1, the mask flag disables the channel so that DMA requests (either hardware or software) are not recognized. If a channel is not programmed for auto-initialization, this flag is automatically set by $\overline{\text{EOP}}$ going active. The request and mask flags are programmed using commands in which bit 2 determines the setting of the flag and bits 1 and 0 give the channel number of the flag. The remaining bits are unused. There is also a command for adjusting all four of the mask flags at once. For this command, bit n being set or cleared causes the mask bit for channel n to be set or cleared.

Besides the commands for setting the flags, there are a master clear command and a clear first/last flip-flop command. A master clear command has the same effect as a RESET signal. The first/last flip-flop is for loading the 16-bit address and count registers. Because an 8237 can be sent only one byte at a time, these registers must be loaded using two outputs. Assuming that the first/last flip-flop is initially 0, the first byte output to one of these registers is put in the low-order byte and the flip-flop is set to 1. When the second byte is sent, the flip-flop being 1 directs it to the high-order byte of the register. Then the flip-flop is reset to 0. The purpose of the clear first/last flip-flop command is to initialize this flip-flop before outputting to the address and count registers. Neither of the two commands discussed in this paragraph involves the transfer of data over the data bus. They are automatically executed by the controller when a write is made to the appropriate address.

The addressing of the various registers and commands associated with the controller is done via the $\overline{\text{CS}}$, $\overline{\text{IOR}}$, $\overline{\text{IOW}}$, and A3–A0 lines. $\overline{\text{CS}}$ = 0, of course, indicates that the controller is being accessed. A3 equals 0 when an address or count register is addressed and is 1 when the control or status register is accessed or a command is being given. When addressing an address or count register A2–A1 gives the channel number and A0 = 0 specifies the current address register and A0 = 1 the current count register. For a read, $\overline{\text{IOR}}$ is low and $\overline{\text{IOW}}$ is high, and for a write, $\overline{\text{IOR}}$ is high and $\overline{\text{IOW}}$ is low. A write to either the current address or current count register will also write to the associated base register. Addressing

the control and status registers and giving commands are summarized as follows:

A3	A2	A1	A0	$\overline{IOR}$	$\overline{IOW}$	Transfer or Command
1	0	0	0	0	1	Read from status register
1	0	0	0	1	0	Write to control register
1	0	0	1	1	0	Write to a request flag
1	0	1	0	1	0	Write to a mask flag
1	0	1	1	1	0	Write to mode register
1	1	0	0	1	0	Clear first/last flip-flop
1	1	0	1	0	1	Read temporary register
1	1	0	1	1	0	Master clear
1	1	1	0	1	0	Clear mask flags
1	1	1	1	1	0	Write to all mask flags

where 0 means low and 1 means high. (Note that the lowest port address assigned to an 8237 must be divisible by 16_{10}.) The temporary register can be read only after the completion of the entire memory-to-memory block transfer. All combinations not shown are illegal.

The 8237 timing can be divided into the SI, S0, S1, S2, S3, S4, and SW states. A flowchart of the timing as seen by the 8237 is given in Fig. 9-40. Between transfers the controller is in a succession of idle, SI, states. During each SI state the DREQ lines are tested to determine if a channel is requesting a DMA transfer. If $\overline{CS} = 0$ and all of the DREQs are inactive, the controller becomes a slave and the CPU can communicate with it. If there is an active DREQ input, HRQ is activated and an S0 state is entered. S0 states are repeated until a HLDA signal is returned, and then a sequence of the S1 through S4 states occurs. S1 may be skipped if the high-order byte of the transfer address does not need to be changed. When a transfer is made to or from a device that cannot respond quickly enough, wait (SW) states are inserted. If normal timing is used, S3 states are needed, but if the bus and its interfaces can handle compressed timing, the S3 states are deleted. During S4 the transfer mode is tested and, except for incomplete block transfers in the block and demand modes, a return is made to SI. A timing diagram for interface to memory transfers is shown in Fig. 9-41.

9-6 DISKETTE CONTROLLERS

The primary means of storing large quantities of data are magnetic tape and disk units, although magnetic bubble memory (MBM) is currently having some impact in this area. Magnetic tapes and disks have two major advantages over MBMs: portability between systems and capacity. Magnetic tapes tend to cost less per byte of stored information and be the most durable, but disks have much lower access times. Tapes and disks come in many sizes and shapes and the designs of the equipment for reading from and writing onto them vary correspondingly. It is not possible to give a reasonable presentation of the numerous types of tape and disk units in the space available here; therefore, we will concentrate on what is by far the most popular mass storage units in microcomputer systems, the *diskette*, or *floppy disk*, units.

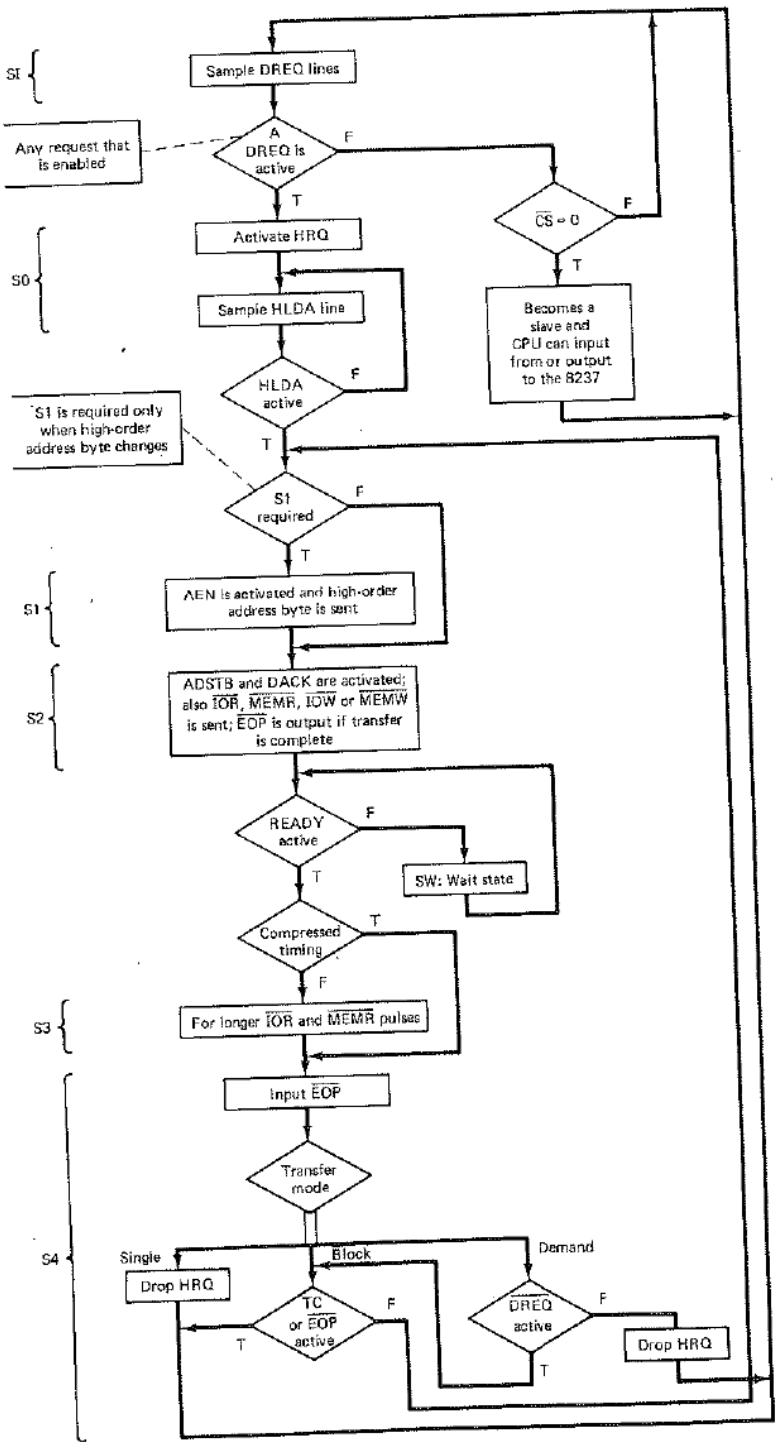


Figure 9-40 Flowchart of 8237 activity by states.

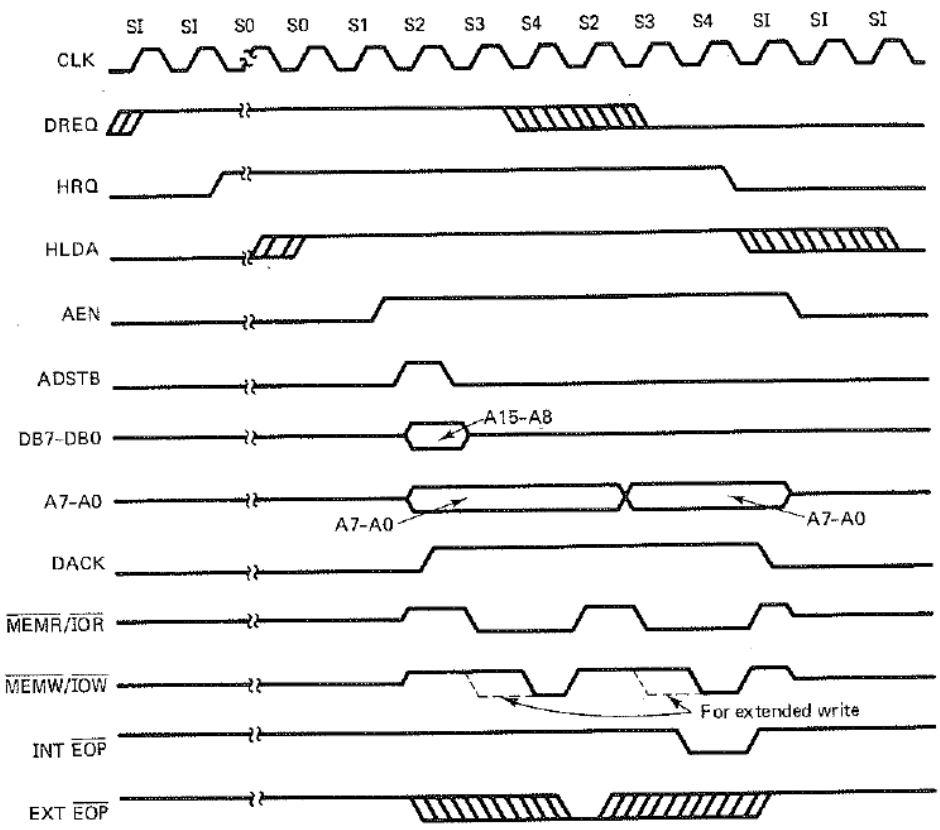


Figure 9-41 Typical timing diagram for an 8237.

The physical construction and overall data storage format of a diskette is shown in Fig. 9-42. The diskette is a Mylar disk that is coated with a magnetic substance. It has a spindle hole in the center and at least one index hole that is slightly offset from the center. The purpose of the index hole(s) is to give the diskette drive a reference point(s) while the drive is storing or retrieving data. The diskette is sealed in a square jacket that also has a spindle hole and index hole. In addition, the jacket has a slot so that the drive's read/write head can contact the diskette's surface, and a write protect notch.

The data are bit serially stored (i.e., as a succession of bits) in concentric circles called *tracks* and are grouped into arcs known as *sectors*. Some diskette drives have only one read/write head and can only store and retrieve data from one surface of the diskette, while others have two read/write heads and can utilize both surfaces. If both surfaces can be accessed, then the pairs of tracks that are the same distance from the center of the diskette are referred to as *cylinders*.

There are two standard sizes for diskettes, ones with 8-in. jackets and ones with 5.25-in. jackets (called *mini-diskettes*). Some have only one index hole and are said to be *soft sectored*, while others have an index hole for each sector and

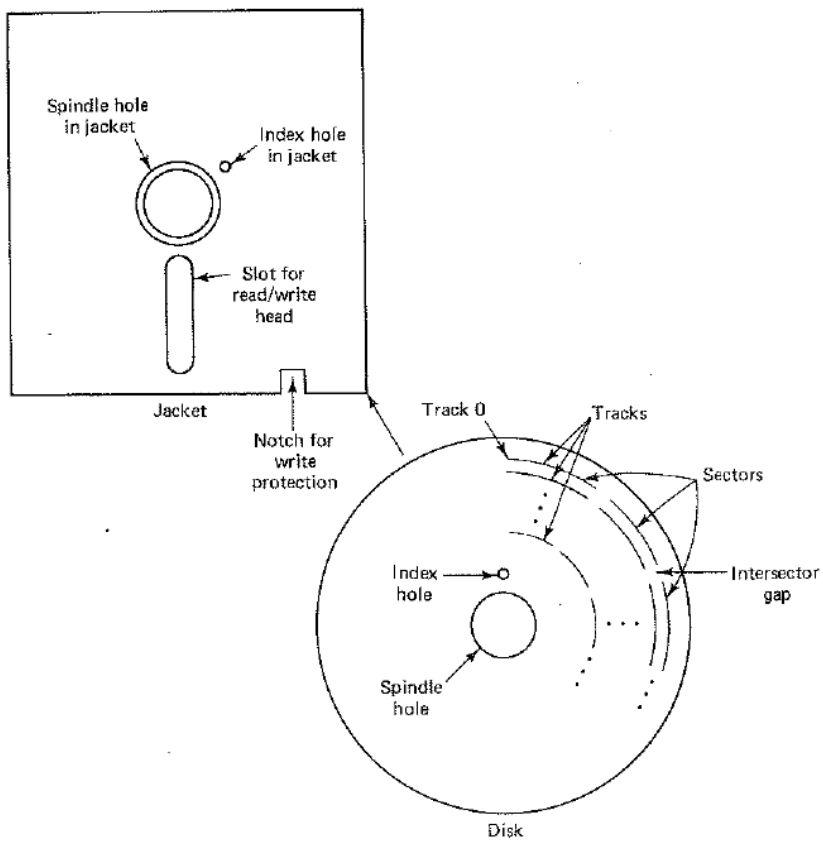


Figure 9-42 Construction of a diskette.

are said to be *hard sectored*. Each sector on a soft-sectored diskette must start with formatting information that marks the beginning of the sector and only the first sector of a track is marked by an index hole. For a hard-sectored diskette the beginning of all sectors are marked by index holes.

The tracks (and cylinders) are numbered, with the outermost track being given the number 0. The sectors are also numbered and on a soft-sectored diskette, the first sector after the index hole is assigned the number 1. When a diskette is inserted in a drive, the spindle passes through the spindle hole and the diskette is clamped to the spindle and is continuously rotated within its jacket at 360 rpm. The read/write head moves in and out so that it can be positioned over any desired track. Once positioned, the rotational motion causes the sectors within the selected track to pass under the head, thus permitting the individual sectors to be accessed. When a read or write is to be performed, the head is lowered, or *loaded*, onto the surface of the diskette through the slot in the jacket, and is then positioned over the desired track. For a soft-sectored diskette, the drive waits until the index hole is encountered and then begins to read the sector formatting information. Upon

finding the needed sector the drive may begin its read or write operation. In the case of a hard-sectored disk the sector is found by means of the index holes.

The time needed to access a sector is subdivided into:

Load Time—For bringing the head in contact with the diskette.

Position Time—For positioning the head over the track.

Rotational Time—For rotating the diskette until it is over the desired sector.

Typical average load, position, and rotational times are 16, 225, and 80 ms, respectively. Once a sector is found the average information transfer rate in bytes per second is approximately

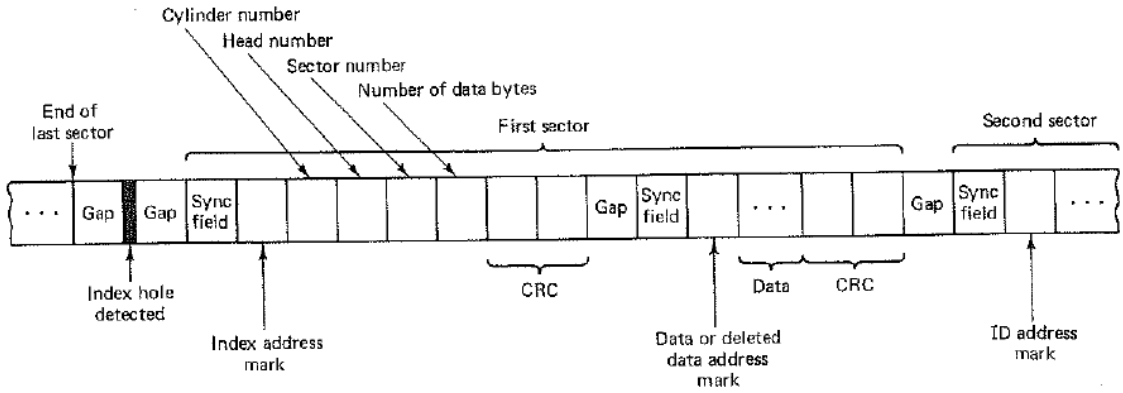
$$\text{Bytes per sector} \times \text{Sectors per track} \times \text{Speed in rpm}/60$$

(This includes the time wasted in traversing gaps in the data.)

In order to make our discussion more specific, let us now limit it to a single type of diskette, the IBM 3740-compatible, soft-sectored, 8-in. diskette. These diskettes contain 77 tracks and either 15 sectors (single density) or 26 sectors (double density), although our examples will permit from 8 to 26 sectors. Normally, only 75 of the tracks are used, thus allowing for two bad tracks. The good tracks are numbered 0 through 74.

The track and sector format of these diskettes is illustrated in Fig. 9-43. There is a gap between each of the sectors and the index hole indicator must become active during the gap between the last sector and first sector of each track. Each sector contains an identification field and a data field, both of which begin with a sync field. There is a gap between the two fields which gives the head time to be turned on or off, depending on whether the previous or following data field was or is to be accessed. The identification field contains an address mark followed by the cylinder (or track) number, head number, sector number, number of data bytes

Figure 9-43 Track and sector format for a soft-sectored diskette.



in the sector, and two error detection bytes. The data field consists of an address mark followed by the data bytes and two error detection bytes.

The address mark in the identification field of the first sector is usually different from those in the other sectors and the address marks in the identification fields are different from those in the data fields. Also, the address marks for the data field differ depending on whether the field contains useful information or is occupied by filler bytes.

There are two principal methods for generating the error detection bytes. One is to obtain a *checksum* by summing all of the bytes in the field that occur before the checksum (and ignoring any overflow). The other is to consider the bits in the field as coefficients of one big binary polynomial and let the error detection bytes be the remainder, called the *cyclic redundancy check (CRC)*, that results from dividing this polynomial by a fixed polynomial of degree 16. It is the latter method that will be assumed here.

The bit pattern of the serially stored information is shown in Fig. 9-44. The information is grouped into cells, each of which is divided into four intervals, with the first interval containing a clock pulse. The third interval is for indicating a data bit and will contain a pulse if the data bit is 1. A representative value for the cell period for a 360-rpm drive is 4 μs.

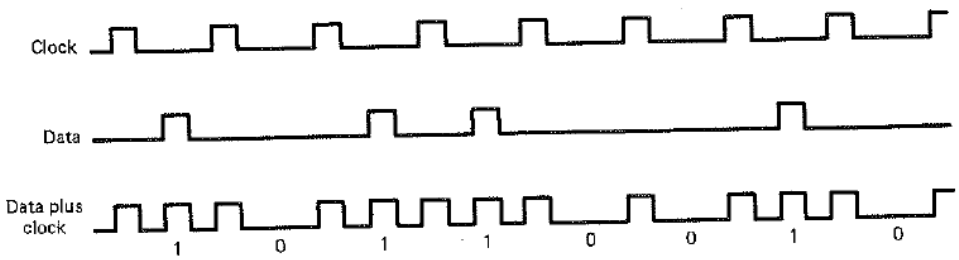


Figure 9-44 Bit pattern of stored data.

Figure 9-45 gives the organization of a typical diskette controller and Fig. 9-46 shows how the controller would appear in a diskette subsystem. Because the information is serially stored on the diskette the inputs from and outputs to the diskette drive are bit streams. Therefore, the input bytes are received by a shift register before they are transferred to the data buffer register, and the output bytes are put in a shift register so that they can be shifted out in serial form. The output electronics must be such that the pulses from the write clock input are interspersed with the data bits to produce the format given in Fig. 9-44.

Because most disk controllers are designed to service more than one drive, some of the pins on the controller must be reserved for drive selection lines. Other control outputs to the drive should include a step pulse line for stepping through the tracks, a step direction signal, and a head load signal. There should be control inputs for inputting signals indicating when the head is positioned at track 0, a fault or error has occurred, or the diskette is write protected, and a pulse each time the index hole is sensed.

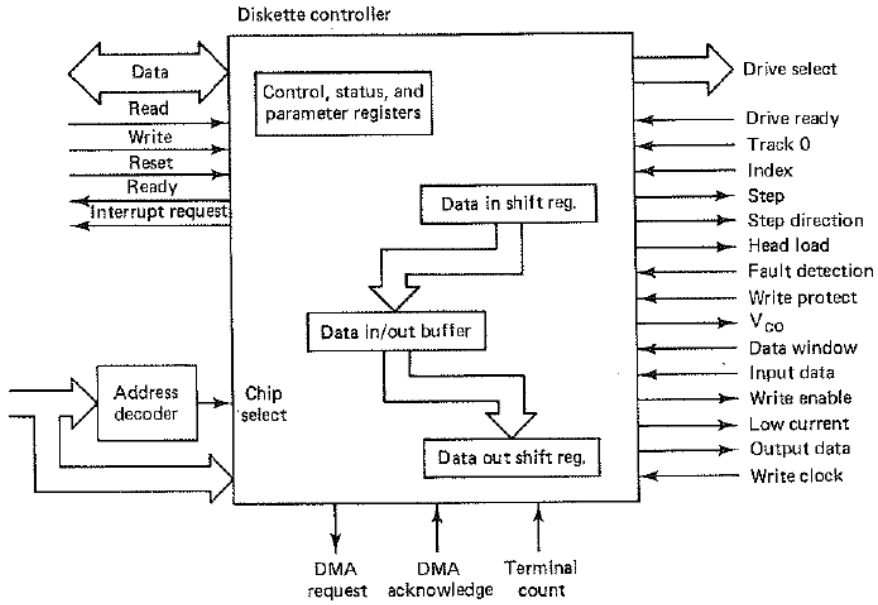


Figure 9-45 Organization of a diskette controller.

For reading data from the disk the data must be separated from the clock pulses. This is done by using a phase-locked-loop circuit to provide a data window signal. The controller must supply a V_{CO} sync signal to the phase-locked loop. For writing data there should be a write enable and low-current lines in addition to the output data line. The low-current line is needed because less current is used when writing to the inner tracks, those tracks with numbers greater than 43.

If a diskette having 26 sectors and 128 data bytes per sector is rotated at 360 rpm, the average transfer rate is approximately

$$26 \times 128 \times 360/60 = 19{,}968 \text{ bytes/second}$$

which gives an average period of about 50 μs. Taking into account the gaps, the actual period for transferring a byte is normally 32 μs (or 4 μs per bit). Although it would be possible to perform transfers at this rate without a DMA controller, it is much more efficient to use one. Consequently, disk controllers normally have DMA request, DMA acknowledge, and terminal count pins. Another reason for using a DMA controller with a disk controller is that disk transfers always involve blocks of data, not single bytes.

As an example of a diskette controller let us consider the Intel 8272, whose organization is diagrammed in Fig. 9-47. It has two registers that can be addressed by the processor, a status register that is read when A0 = 0 and a data in/out register that is accessed when A0 = 1. The format of the status register is given in Fig. 9-48. There are several other control, status, and parameter registers and flags that can be addressed via the data in/out register. Whether these registers

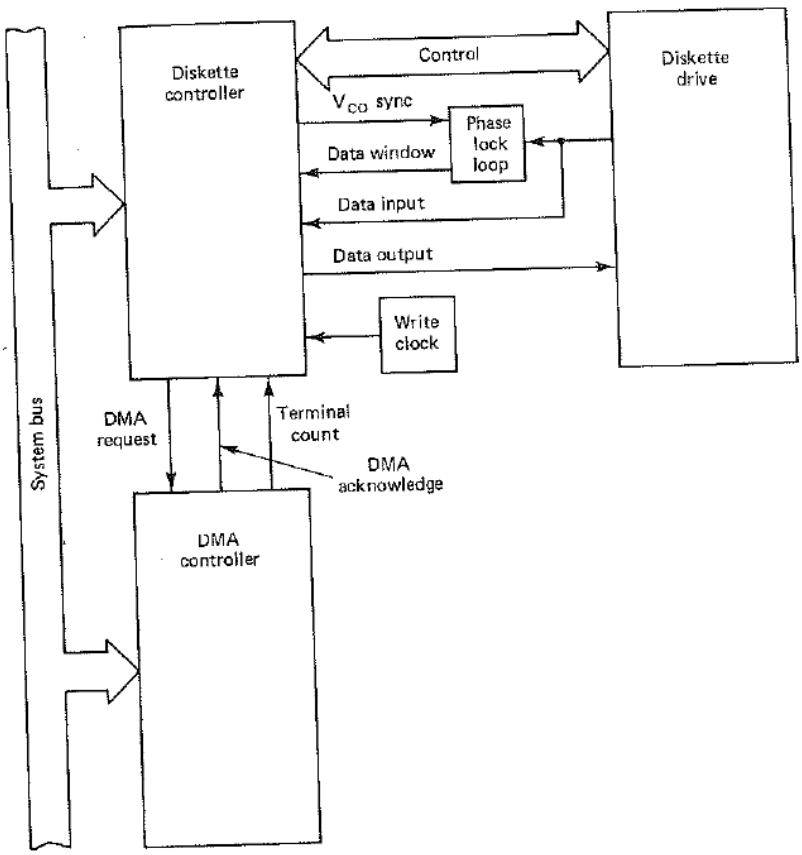


Figure 9-46 Diskette subsystem.

and flags are being accessed or data I/O is being performed with the diskette drive depends on the sequencing.

Operations executed by the controller are divided into a command phase, an execution phase, and a result phase. During the command phase bytes are sent to the control registers and flags via the data in/out register. Then the requested operation is performed during the execution phase and, upon completion of the operation, the result phase is entered and status information is read by the processor. The outputs during the command phase and inputs during the result phase are performed using single-byte transfers, even though any data transfer that takes place in the execution phase is normally supervised by a DMA controller.

The possible 8272 commands are:

Read Data—Reads data from a data field on a diskette.

Write Data—Writes data to a data field on a diskette.

Read Deleted Data—Reads data from a field marked as being deleted.

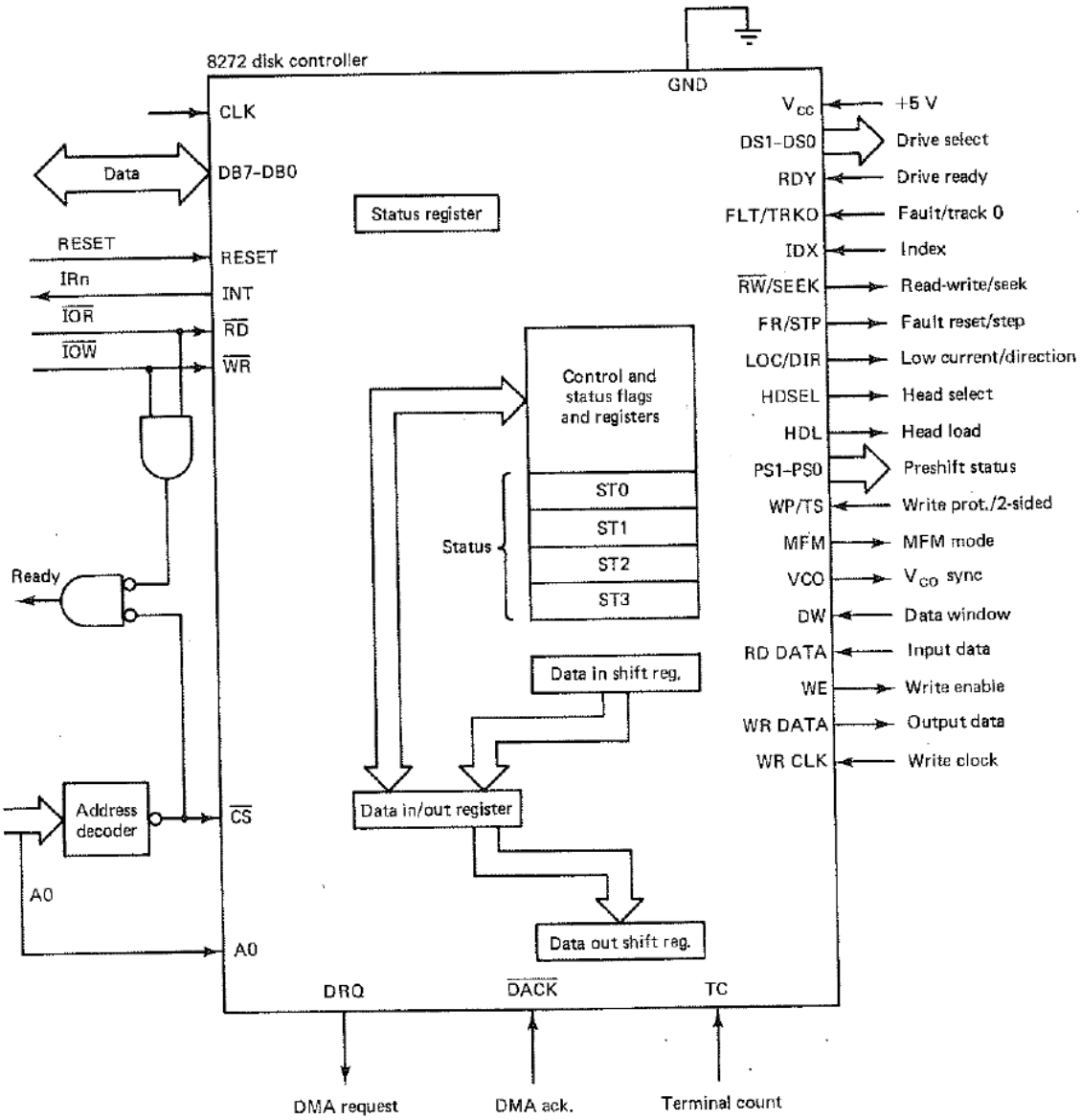
Write Deleted Data—Writes a deleted data address mark and puts filler characters in the data field.

Read A Track—Reads an entire track of data fields.

Read ID—Reads the identification field.

Format A Track—Writes the formatting information to a track using program supplied formatting parameters.

Figure 9-47 Diagram of an 8272 diskette controller.



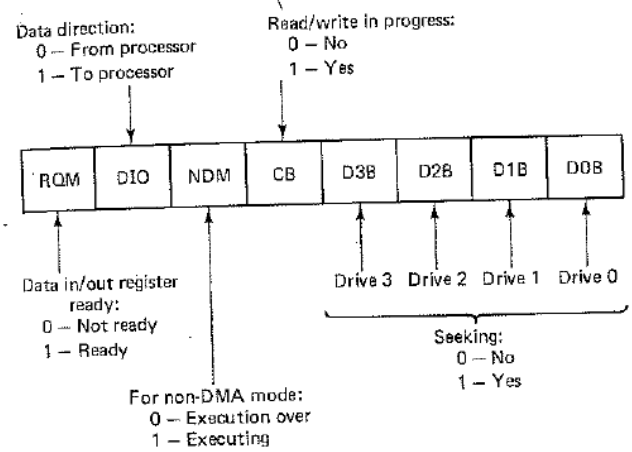


Figure 9-48 Format of the 8272's status register.

Scan Equal
Scan Low or Equal
Scan High or Equal
} Scans the data for specified condition and makes an interrupt request when the condition is satisfied.

Recalibrate—Causes the head to be retracted to track 0.

Sense Interrupt Status—Reads status information from ST0 after interrupts caused by ready line changes and seek operations.

Specify—Sets the step rate, head unload time, head load time, and DMA mode.

Sense Drive Status—Inputs the drive status from ST3.

Seek—Positions head over a specified track.

The sequence needed to complete four of these commands is given in Fig. 9-49 as an example. The read data command includes all three phases, the seek command includes only the command and execution phases, the sense drive status command includes only the command and result phases, and the specify command consists only of the command phase. In all cases, the commands must be performed in their entirety (including the result phase, if applicable) or they will be considered incomplete. If an invalid command is given, the 8272 will return the status byte ST0 in response to the next input from the data in/out register.

Several of the commands require that some of the status bytes ST0, ST1, ST2, and ST3 be read during their result phases. These registers contain bits for indicating such things as CRC errors, overrun errors, drive selected, end of seek, whether or not the diskette is two-sided, and so on.

As an example, consider an 8272 whose even address is 002A. The 8272 could be initialized to a step rate of 6 ms per track, a head unload time of 48 ms, a head load time of 16 ms, and the DMA mode by the specify command sequence

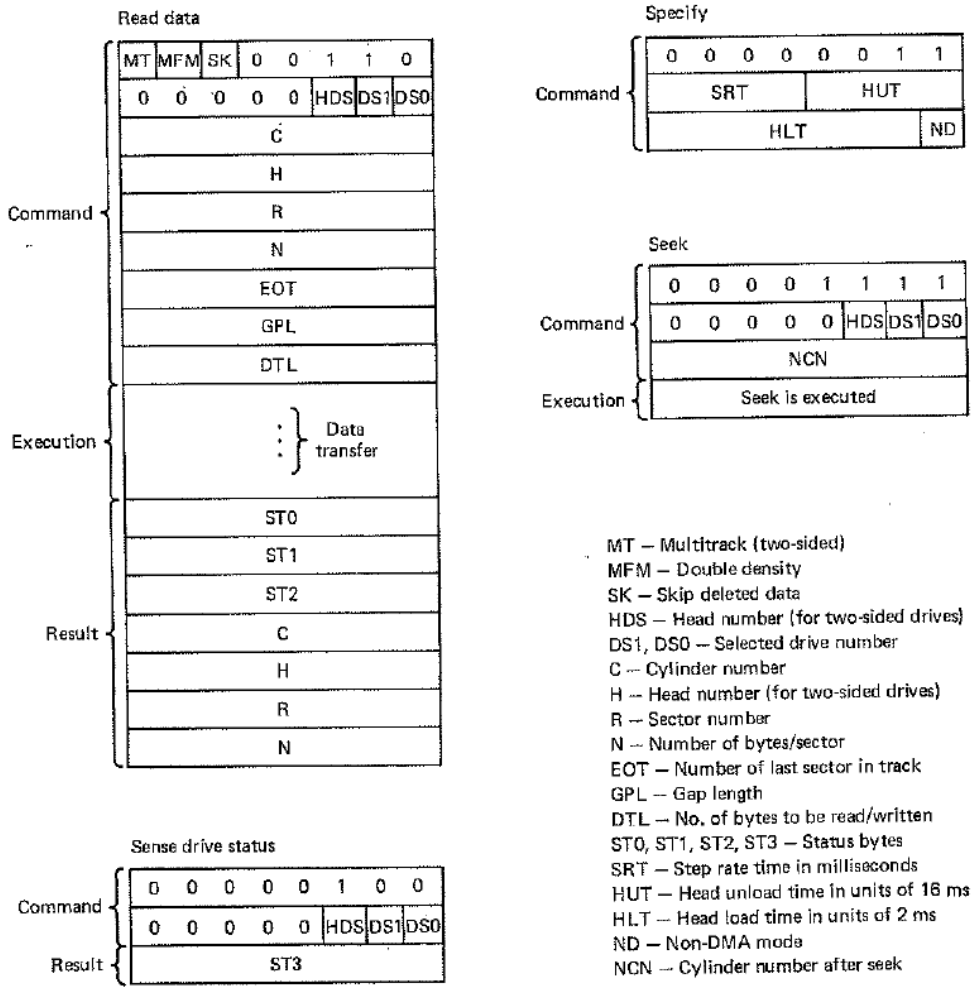
```
%CHECK  (2AH,80H)
        MOV     AL,03H
        OUT     2BH,AL
```

```
%CHECK  (2AH,80H)
        MOV     AL,63H
        OUT     2BH,AL
%CHECK  (2AH,80H)
        MOV     AL,10H
        OUT     2BH,AL
```

where the macro CHECK is defined as follows:

```
%*DEFINE  (CHECK (PORT,STAT))  LOCAL AGAIN
(%AGAIN:  IN     AL,%PORT
          AND    AL,0C0H
          XOR    AL,%STAT
          JNE    %AGAIN
)
```

Figure 9-49 Typical 8272 commands.



MT — Multitrack (two-sided)
MFM — Double density
SK — Skip deleted data
HDS — Head number (for two-sided drives)
DS1, DS0 — Selected drive number
C — Cylinder number
H — Head number (for two-sided drives)
R — Sector number
N — Number of bytes/sector
EOT — Number of last sector in track
GPL — Gap length
DTL — No. of bytes to be read/written
ST0, ST1, ST2, ST3 — Status bytes
SRT — Step rate time in milliseconds
HUT — Head unload time in units of 16 ms
HLT — Head load time in units of 2 ms
ND — Non-DMA mode
NCN — Cylinder number after seek

The reason the %CHECK macro is needed before each output is that the two MSBs of the status register must be 10 before each command byte is written into the 8272. Also during the result phase, these 2 bits must be 11 before each byte is read from the 8272's data register.

The seek command sequence

```
%CHECK  (2AH,80H)
        MOV     AL,0FH
        OUT     2BH,AL
%CHECK  (2AH,80H)
        MOV     AL,02H
        OUT     2BH,AL
%CHECK  (2AH,80H)
        MOV     AL,30
        OUT     2BH,AL
```

would cause the head on drive 2 to be moved to cylinder 30 and head 0 to be selected. Figure 9-50 defines two macros for executing the read data command. It is assumed that a sequence such as

```
IDLE:   IN      AL,2AH
        TEST    AL,12H
        JNZ     IDLE
```

would appear just prior to a READ_COM macro call to make certain that the desired drive (which in this example is drive 1) is available. Also, the associated DMA must be initialized before the READ_COM macro call is made so that DMA transfers may begin as soon as the read operation is performed. It is assumed that a call to the READ_STAT macro would be made only after the execution phase, and the read is known to be complete (e.g., after an interrupt on completion interrupt has occurred).

9-7 MAXIMUM MODE AND 16-BIT BUS INTERFACE DESIGNS

As noted in the chapter's introduction, all of the Intel examples in the first six sections of this chapter are based on a minimum mode 8088 processor. To convert the designs to a maximum mode system two primary changes are necessary. First of all, an 8288 bus controller must be connected into the system as shown in Figs. 8-9 and 8-10. For a system that contains an 8237 DMA controller, the 8288 would replace the circuit for encoding the $\overline{RD}$, $\overline{WR}$, and IO/$\overline{M}$ signals shown in Fig. 9-38 and detailed in Fig. 9-39. In any event, with the inclusion of an 8288 the $\overline{RD}$ and $\overline{WR}$ pins on the interfaces would be attached to the $\overline{IORC}$ and $\overline{IOWC}$ outputs from the 8288 and the IO/$\overline{M}$ lines shown entering the address decoders would no longer be required.

The other major change concerns the HRQ and HLDA signals used to make bus requests and grants. The 8237 is designed to output a continuous request HRQ signal until it is ready to relinquish the bus, at which time it drops the signal. Also, the processor outputs a continuous HLDA signal. This is compatible with an 8086/

```
%*DEFINE (READ_COM (ADDR,HDSDS,C,H,R,N,DTL))
(
        PUSH    AX              ;SAVE REGISTERS
    %CHECK (2AH,80H)
        MOV     AL,46H          ;OUTPUT COMMAND
        OUT     %ADDR,AL        ;CODE
    %CHECK (2AH,80H)
        MOV     AL,%HDSDS       ;OUTPUT HEAD AND
        OUT     %ADDR,AL        ;DRIVE NOS.
    %CHECK (2AH,80H)
        MOV     AL,%C           ;OUTPUT CHANNEL NO.,
        OUT     %ADDR,AL
    %CHECK (2AH,80H)
        MOV     AL,%H           ;HEAD NO.,
        OUT     %ADDR,AL
    %CHECK (2AH,80H)
        MOV     AL,%R           ;SECTOR NO.,
        OUT     %ADDR,AL
    %CHECK (2AH,80H)
        MOV     AL,%N           ;NO. OF DATA BYTES,
        OUT     %ADDR,AL
    %CHECK (2AH,80H)
        MOV     AL,26           ;NO. OF SECTORS,
        OUT     %ADDR,AL
    %CHECK (2AH,80H)
        MOV     AL,17           ;GAP LENGTH, AND
        OUT     %ADDR,AL
    %CHECK (2AH,80H)
        MOV     AL,%DTL         ;DATA LENGTH
        OUT     %ADDR,AL
        POP     AX
)
%*DEFINE (READ_STAT (ADDR,RDSTAT)) LOCAL LP
(       PUSH    AX              ;SAVE REGISTERS
        PUSH    CX
        PUSH    SI
        MOV     SI,0            ;INPUT STATUS TO
        MOV     CX,7            ;ARRAY RDSTAT
    %LP:NOP
    %CHECK (2AH,0C0H)
        IN      AL,%ADDR
        MOV     %RDSTAT[SI],AL
        INC     SI
        LOOP    %LP
        POP     SI              ;RESTORE REGISTERS
        POP     CX
        POP     AX
)
```

Figure 9-50 Macros for executing read data commands.

8088 processor in minimum mode, but in maximum mode the processor uses a single $\overline{RQ}/\overline{GT}$ line to both receive requests and issue grants and expects to see only a pulse at the time the request is made. The request is acknowledged by outputting a pulse and a second pulse must be sent to the processor from the DMA controller at the conclusion of the DMA activity. A circuit for converting between the two-line continuous signals associated with the 8237 and the one-line pulses of a maximum mode processor is given in Fig. 9-51.

The problems associated with connecting the 8-bit interface devices to a 16-bit bus of an 8086 are related to the need to transfer even-addressed bytes over the lower half of the data bus and odd-addressed bytes over the upper half. For interfaces that communicate only with the processor (i.e., do not utilize DMA), the problem can be solved quite simply. Instead of connecting the address lines for selecting individual registers internal to the interface, say An–A0, to the interface's pins An–A0, attach lines A(n+1) – A1 to those pins as shown in Fig. 9-52.

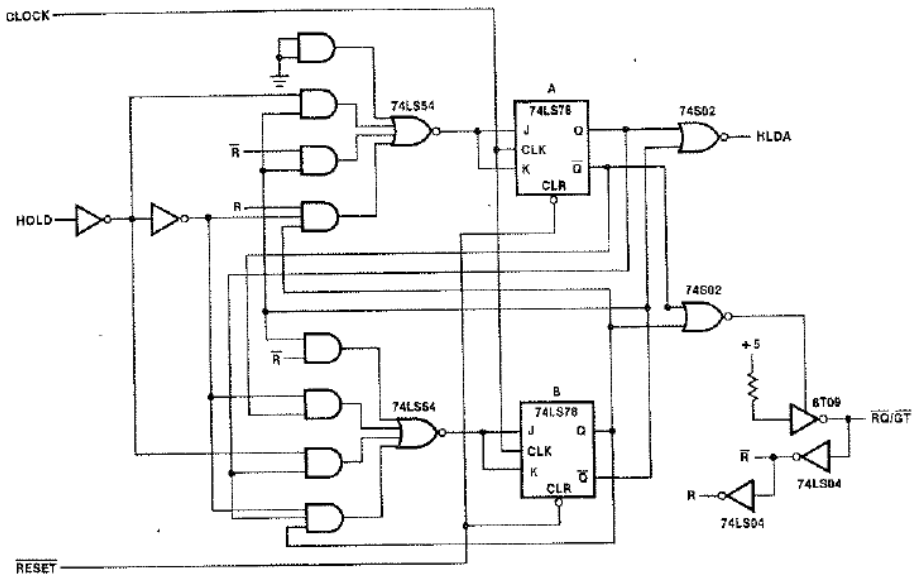


Figure 9-51 Bus request and grant conversions needed when connecting an 8237 DMA controller to a maximum mode 8086/8088. (Reprinted by permission of Intel Corporation. Copyright 1979.)

This means that the interface will be assigned only even addresses in the I/O address space beginning with an address divisible by 2^{n+1} and the interspersed odd addresses would not be used. For example, if the A1 and A0 pins on the 8255A were connected to the A2 and A1 address lines and the beginning address of the 8255A ports were 08F8, then all transfers to and from the 8255A would be made over the low-order byte of the bus. The ports A, B, and C would have the addresses 08F8, 08FA and 08FC, respectively, and the control register would be assigned the address 08FE. Likewise, consecutive odd addresses can be assigned to an interface if the interface is connected to the high-order byte of the bus.

To use successive addresses and both halves of a 16-bit data bus, the bus could be connected as shown in Fig. 9-53. To access a register the 8086 uses its $\overline{BHE}$ and A0 as follows:

$\overline{BHE}$	A0	Transfer
0	0	Not useful
0	1	Odd-addressed byte on upper half of bus
1	0	Even-addressed byte on lower half of bus
1	1	Not possible

where 0 is low and 1 is high. By using these signals and two 8286 transceivers data can be transferred between the interface and the data bus lines D7–D0 when $\overline{BHE} = 1$ and A0 = 0, and the bus lines D15–D8 when $\overline{BHE} = 0$ and A0 = 1. The $\overline{RD}$ signal is used to determine the direction of the data flow. The ready logic is not shown.

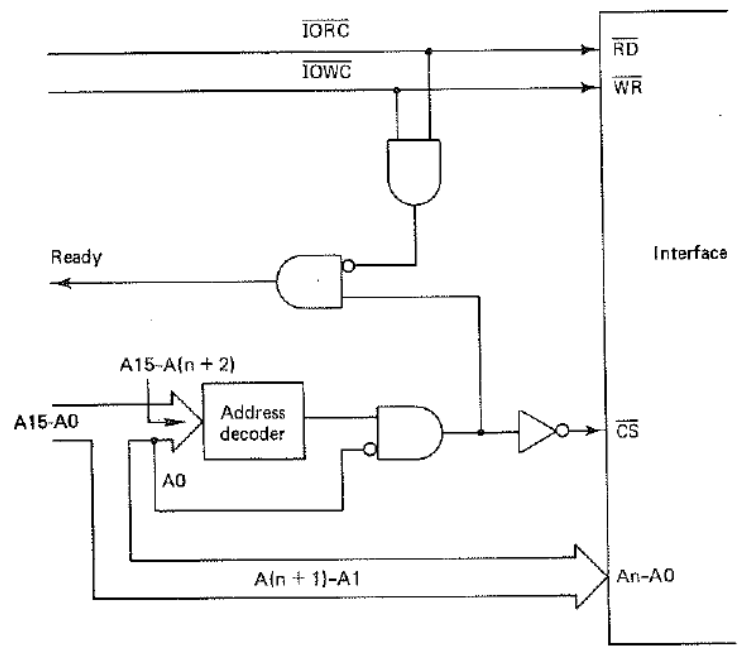


Figure 9-52 Accessing only even-addressed I/O ports.

For an 8237-based DMA system, the bus control logic shown in Fig. 9-38 must be altered as indicated in Fig. 9-54. The $\overline{BHE}$ signal coming out of the 8086 would be latched by the same 8282 as is used by the A19–A16 lines. The A0 line would be connected to the $\overline{BHE}$ line through a tristate inverter which is controlled by the 8237's AEN signal. When AEN is active and A0 = 0, $\overline{BHE}$ is high, indicating that the transfer is to be made over the lower byte of the bus. If AEN is active and A0 = 1, $\overline{BHE}$ is low and the transfer is made over the upper byte. In addition, both the interface and the 8237 must be connected to the bus through extra logic similar to that given in Fig. 9-53.

Although the above paragraphs have been concerned with the communication between an 8-bit interface and a 16-bit data bus, some attention should be given to the design of a 16-bit interface. Such an interface would transfer entire words to and from the data bus and would tend to double the utilization of the available bus cycles. A 16-bit interface design based on two 8255As is given in Fig. 9-55. The A2 and A1 lines in the address bus are connected to the A1 and A0 pins of both 8255As; thus the 16-bit ports are formed from the pairs of ports A, B, C, and the control/status registers. The lower 8255A occupies 4 consecutive even addresses and the upper 8255A occupies 4 consecutive odd addresses. If bits A15–A3 match the address designed into the address decoder, then the decoder will emit a 0 chip select signal. For the lower 8255A, if both the chip select and A0 signals are 0, then a 0 is applied to $\overline{CS}$. For the upper 8255A, both the chip select and $\overline{BHE}$ signals must be 0 in order for a 0 to be sent to the $\overline{CS}$. (Therefore, it is

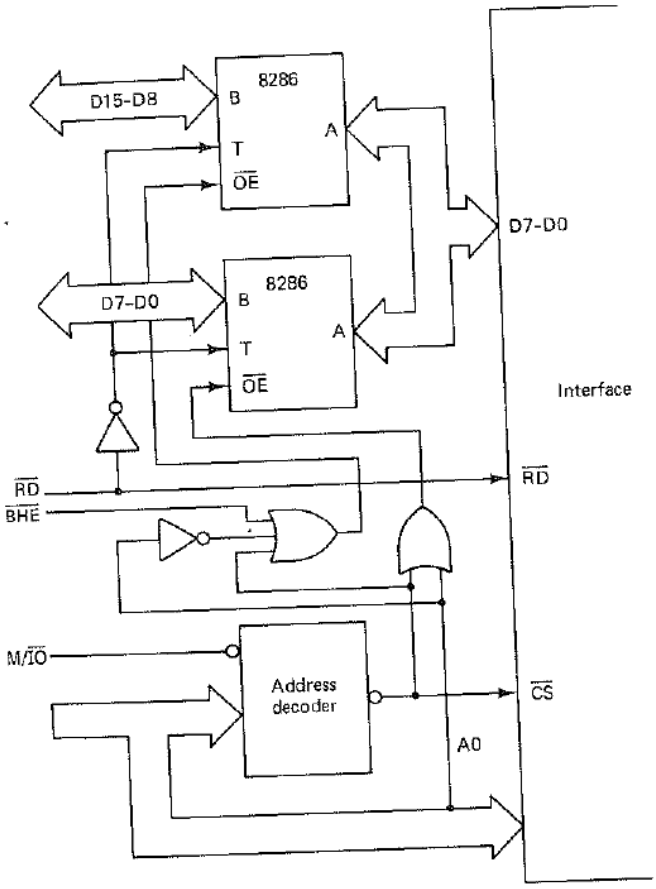


Figure 9-53 Connection of an 8-bit interface to a 16-bit data bus.

possible to address the 8255As individually.) The read, write, and reset control lines are connected to the $\overline{RD}$, $\overline{WR}$, and RESET pins of both of the 8255As and a ready signal is returned if either $\overline{CS}$ signal is active.

One other alternative in interface design is to treat registers of an I/O device as memory locations. These locations are then referred to as *memory-mapped I/O*. This scheme requires the interface to respond to memory read and memory write commands. The chief advantage of memory-mapped I/O is that the device registers can be accessed and manipulated with any instruction or addressing mode that references memory operands, thus providing programming flexibility and reducing coding. However, if the control and status registers share the same address, as they do for several of the Intel interface devices, one must not use an instruction such as

```
OR CONTROL,00000001B
```

which would input from the status register and output to the control register. Because the same address may not be assigned to both memory and an I/O device,

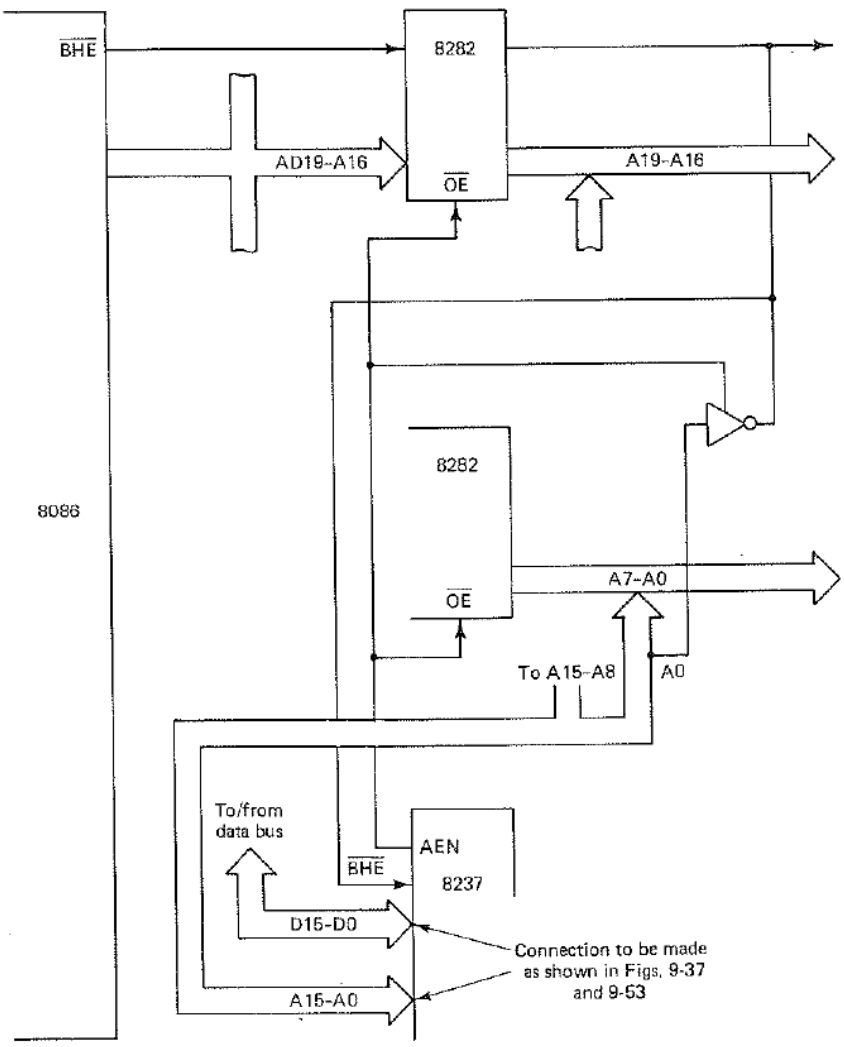


Figure 9-54 Using an 8237 DMA controller with a 16-bit data bus.

memory-mapped I/O causes the maximum number of available memory locations to be reduced.

BIBLIOGRAPHY

1. McNamara, John E., *Technical Aspects of Data Communications* (Bedford, Mass.: Digital Equipment Corporation, 1978).
2. *1981 Data Component Catalog* (Santa Clara, Calif.: Intel Corporation, 1981).

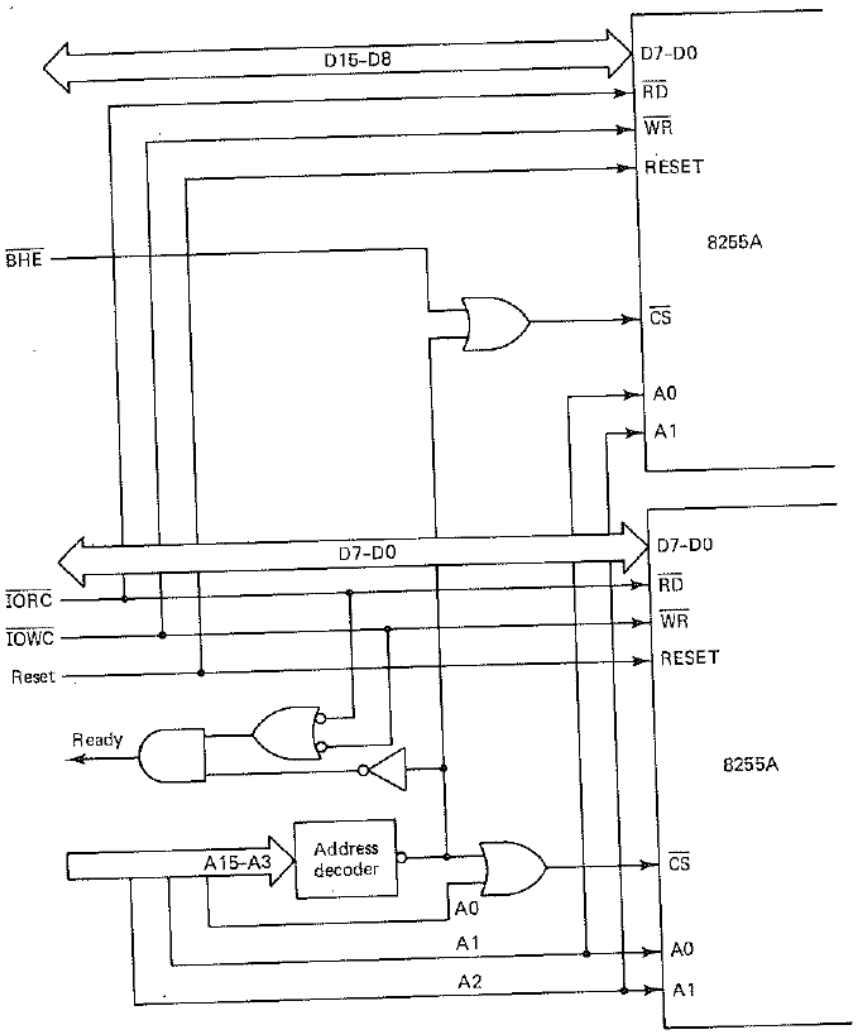


Figure 9-55 Example of a 16-bit parallel I/O interface.

3. Peatman, John B., *Microcomputer-Based Design* (New York: McGraw-Hill Book Company, 1977).
4. Hall, Douglas V., *Microprocessors and Digital Systems* (New York: McGraw-Hill Book Company, 1980).

EXERCISES

1. Assume the 7-bit ASCII code, even parity, and 2 stop bits and sketch the timing diagram of the letter C as it is transmitted over an asynchronous serial communication link.

2. If there is no dead time between characters, characters contain 7 information bits, there is a parity bit and 2 stop bits with each character, and the bit rate is 600 bits per second, what is the information transfer rate in characters per second? Compare this figure with the information transfer rate of a synchronous transmission having the same bit rate, a message format consisting of two sync characters followed by 200 seven-bit information characters, and no imbedded idle characters.
3. According to the discussion in Sec. 9-1-3, what is the approximate maximum distance a 4800-baud signal could be transmitted without the aid of communications equipment?
4. If the equivalent circuit shown in Fig. 9-8 is such that

$$R_0 = 500\ \Omega \qquad C_0 = 2\ \mu F$$

C_L is due to 100 ft of a 20-pF/ft line

$$R_L = 4000\ \Omega \qquad E_L = 0\ V$$

what would V_I be when $V_0 = 7$ V? Determine the maximum of the derivative of V_I as V_0 instantaneously changes from a steady state -7 V to $+7$ V. How long would it take for V_I to go from -3 V to $+3$ V if V_0 switches from -7 V to $+7$ V? Would all of the RS-232-C electrical specifications be satisfied if the line is operated at 1200 baud?

5. If the mode register of an 8251A contains 01111011, what will be the format of a transmitted character? In order for the receiver and transmitter baud rates to be 300 and 1200, respectively, what would the frequencies applied to RxC and TxC need to be?
6. Assuming that the contents of the mode register of an 8251A are 00010100, determine the character and message formats of the 8251A's serial transmission.
7. Given that the sequence shown below is output with A0 = 1 following a reset, describe the action taken as each output is received by the 8251A.

```
00010100
00010110
00010110
00110011
   .
   .      } Output with A0 = 0
   .
01000000
```

(See the flowchart in Fig. 9-14.)

8. Write a program sequence that will initialize an 8251A so that it will transmit asynchronous characters containing 7 information bits, 1 stop bit, and no parity bit. The baud rate factor is to be 64. The even address associated with the 8251A is to be 008A. Extend the sequence so that it outputs a command that activates Request to Send and Data Terminal Ready signals and enables the transmitter.
9. Extend the program sequence in Exercise 8 so that it will use programmed I/O to output the message

AN OVERRUN ERROR HAS OCCURRED

and then clear the overrun error bit.

10. Write an interrupt routine that will output the 20 bytes beginning at MESSAGE to the

8251A whose even address is 25C0. Each time the routine is entered it is to output 1 byte, and when it has output 20 bytes it is to put a 1 in FLAG. CNT is to be used to store the byte count.

11. Suppose that the beginning address of an 8255A is 0500 and write a program sequence that will:
- **(a)** Put both groups A and B in mode 0 with ports A and C being input ports and port B being an output port.
- **(b)** Put group A in mode 2 and group B in mode 1 with port B being an output.
- **(c)** Put group A in mode 1 with port A being an input and PC6 and PC7 being outputs, and group B in mode 1 with port B being an input.

12. Assume an 8255A, whose beginning address is 0030, is such that group A is in mode 2 and group B is in mode 0 with PB7–PB0 as inputs and PC2–PC0 as outputs. Write a program that will output 100 bytes from an array beginning at OUT_ARRAY to the device connected to port A. Before outputting each byte the program must loop until PB1, PB2, and PB3 are all active and after outputting the byte the program must set PC1 to 1.

13. Using an 8255A, an 8282 latch, an 8286, and other logic, design a circuit for setting and reading a bank of eight relays. Assume that the 8286 drivers are sufficient to drive the output circuits which supply the current to the relay coils. Specify the 8255A modes and all of the necessary control signals. The relays and their associated circuitry need not be included in the design.

14. By also using port B as a mode 1 input, modify the design in Fig. 9-23 so that a 12-bit A/D converter can be serviced. Give a program sequence for inputting a sample and storing it in the word whose location is represented by SAMPLE.

15. Describe how an 8254 could be used to count the pulses input to it for a period of time that is controlled by a second input. At the end of the overall counting time an interrupt request is to be made on an IR2 line, and after every 20 of the pulses being counted an interrupt request is to be made on an IR3 line. Also write a program sequence for initializing the 8254.

16. Suppose that an 8254 connected to a 1-kHz clock is to be used to keep the time of day in BCD format to the nearest second. It is to start its timekeeping immediately after the bytes HOURS, MINUTES, SECONDS, and AMPM have been loaded with the current time. Write a program sequence that will initialize the 8254 and an interrupt routine that will update the time at the end of each second.

17. Assume that in Fig. 9-30, ports A and B of an 8255A are connected to the keyboard. Draw a flowchart for the keyboard scanning routine. (Do not consider multiple key closures.)

18. Suppose that an 8255A is used to drive the multiple-digit display in Fig. 9-32 in the multiplexed mode.
- **(a)** Show the connections between the 8255A and the display.
- **(b)** Draw a flowchart for a display routine.

19. For the design in Exercise 18, draw a block diagram to show how the refresh operations can be eliminated by using external data latches.

20. Suppose that eight devices are to be monitored and that each device has an 8-bit register for storing its status. Draw a logic diagram to illustrate how this can be accomplished using an 8279.

21. Write a program sequence that will cause an 8237 DMA controller to perform a memory-to-memory transfer of the 1000 bytes beginning at SOURCE to an array beginning at DST. The port addresses of the 8237 are to be 0600 through 060F.
22. Explain exactly how the 8286s in Fig. 9-53 are controlled during an:
    (a) Output to the control register.
    (b) Input from the status register.
    (c) I/O interface to memory transfer.
    (d) Memory-to-memory transfer.
    given that the interface is an 8237.
23. Write a program sequence that will cause an 8237 to perform a demand mode block transfer of 500 bytes to an array beginning at BUFFER. The bytes are to be stored starting with the highest address (i.e., BUFFER + 499). The port addresses of the 8237 are to be 1020 through 102F.
24. Give a set of instructions that will:
    (a) Mask channels 0 and 3 of an 8237.
    (b) Cause a program-initiated DMA transfer on channel 2 of an 8237 according to the current contents of its control, address, and byte count registers.
25. Draw a timing diagram for an 8237 block mode transfer of 4 bytes from an I/O device to memory.
26. Given that the head load time is 32 ms, seek time is 8 ms per track and there are 40 tracks, and the rotational speed is 360 rpm, determine the average access time of a soft-sectored diskette. Next determine the average time needed to first recalibrate the head (i.e., move the head to track 0) and then access an arbitrary sector.
27. Write a program sequence that will use the sense drive status command of an 8272 to put a 1 in location FLAG2 if drive 2 is write protected (bit 6 of result is 1).
28. Give a set of instructions that will loop until one of the drives in an 8272-based diskette subsystem is not busy and will then branch to ROUTn, where n is the number of the drive that is first found to be idle.
29. Draw a flowchart of the complete sequence needed to output a block of data to an 8272-based diskette subsystem that includes an 8237 DMA controller. The flowchart should take into account the initialization steps for both the diskette and DMA controllers as well as the steps to execute the phases of write data command.
30. Describe in detail the bus activity that takes place while two consecutive bytes are transferred from memory to an I/O device over a 16-bit 8086 bus by an 8237 DMA controller.
31. Consider the 16-bit parallel I/O device shown in Fig. 9-55. Give a program sequence that will put both groups in both 8255As in mode 0 with ports A and C as inputs and port B as an output. Then write a pair of instructions that will output 1720 through port B.
32. Modify Fig. 9-47 so that the data buffer register and control/status registers are assigned to consecutive even addresses.