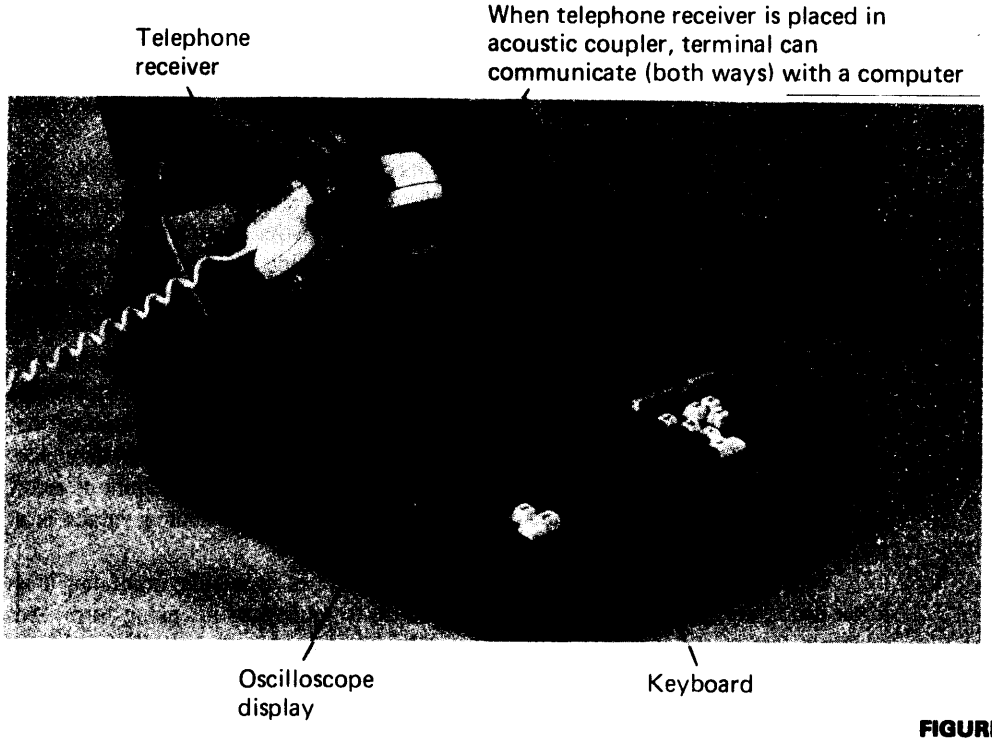


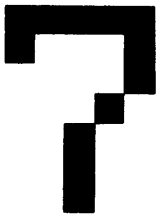
FIGURE 7.22

Oscilloscope display and terminal with acoustic coupler.

Like acoustic couplers, modems generally use a different frequency for a 1 and a 0. As an example, the Bell 103 modem sends data through telephone lines at either 110 or 300 bits/s. The modem at one end of a telephone line uses 1070 Hz (for 1s) and 1270 Hz (for 0s) to send, while the modem at the other end uses 2025 Hz (for 1s) and 2225 Hz (for 0s). The reason for the two sets of frequencies is that transmission can be in either direction, and while one modem is transmitting a character, the other will still be sending its high (mark) frequency. (A single telephone line handles communications in both directions. Both ends can talk at the same time. You can, for instance, interrupt someone who is talking.)

INPUT-OUTPUT DEVICES FOR SYSTEMS WITH ANALOG COMPONENTS

7.14 Not all the inputs to digital machines consist of alphanumeric data. Computers used in data collection systems or in real-time control systems often must measure the physical position of some device or must process electric signals which are analog in nature. Consider a real-time control system in which a computer is used automatically to point a telescope. If, by some system of gears, the position of the telescope along an axis is related to the position of a shaft, the position of this shaft may have to be read into the computer, giving the telescope's pointing angle. This will involve the translation of the shaft position into a binary-coded number which can be read by the computer.



Changing a physical displacement or an analog electric signal to a digital representation is called *analog-to-digital (A-to-D) conversion*. Two major types of A-to-D converters are (1) those that convert mechanical displacements to a digital representation and (2) those that convert an electric analog signal to digital-coded signals.

Suppose that an analog device has as its output a voltage which is to be used by a digital machine. Let us assume that the voltage varies within the limits of 0 to 63 V dc. We can then represent the voltage values with a set of 6-bit numbers ranging from 000000 to 111111. For each integer value the input voltage may assume we assign a corresponding value of the 6-bit number. If the input voltage is 20 V, the corresponding digital value will be 010100, and if the input voltage is 5 V, the corresponding number will be 000101. If, however, the input signal is 20.249 V dc, the 6-bit binary number will not completely describe the input voltage, but will only approximate the input value. The process of approximating the input value is called *quantizing*. The number of bits in the binary number which represents the analog signal is the *precision* of the coder, and the amount of error which exists between the digital output values and the input analog values is a measure of the *accuracy* of the coder.

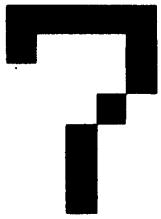
Not only are the inputs to a computer sometimes in analog form, but it is often necessary for the outputs of a computer to be expressed in analog form. An example lies in the use of a cathode-ray tube as an output device. If the output from the computer is to be displayed as a position on an oscilloscope tube, then in some systems the binary-coded output signals from the computer must be converted to voltages or currents, which may be used to position the electron beam and which are proportional to the magnitude of the output binary number represented by the computer's output signals. This involves D-to-A conversion, and a device that performs this conversion is called a *D-to-A converter*. When digital computers are used in control systems, it is generally necessary to convert the digital outputs from the machine to analog-type signals, which are then used to control the physical system.

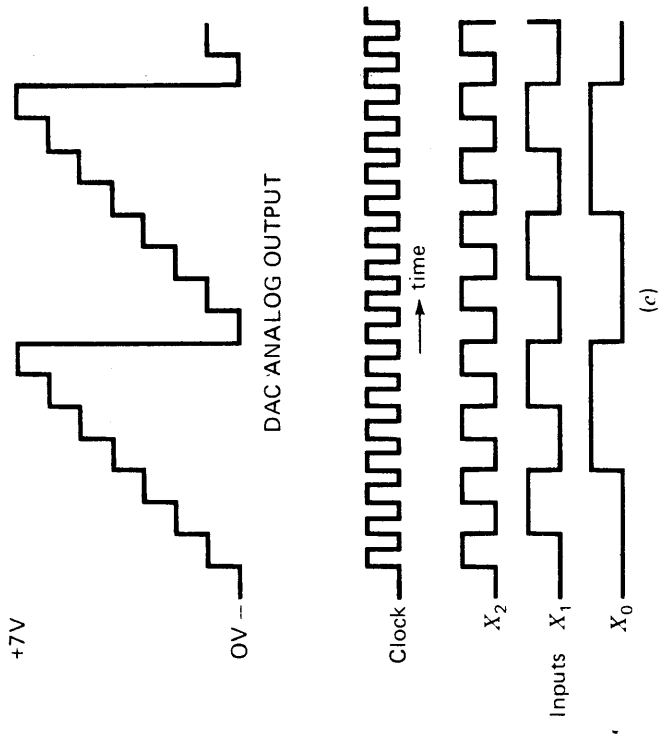
DIGITAL-TO-ANALOG CONVERTERS

7.15 The most used digital-to-analog converters (DACs) convert a binary unsigned number to either an electric voltage or an electric current. DACs which convert from binary inputs to a voltage are discussed first.

A block diagram for a DAC is shown in Fig. 7.23(*a*). There are three input lines X_0, X_1, and X_2, each of which will carry a binary 0 or 1. The number of binary inputs is called the *resolution* of the DAC.[4] The output from this DAC ranges from 0 to 7 V. A list of input-output relations for this DAC is shown in Fig. 7.23(*b*). For each input value there is a corresponding analog output voltage which in this case is equal to the value of the input as a binary integer. Examination of this input-output relation will show that the output value can be calculated by giving the input value X_0 a *weight* of 1 V, X_1 the weight of 2 V, and X_2 the weight of 4 V. Then the output value is equal to the sum of the weights for which the X_i

[4]Appendix G shows circuits for IC realization of DACs.





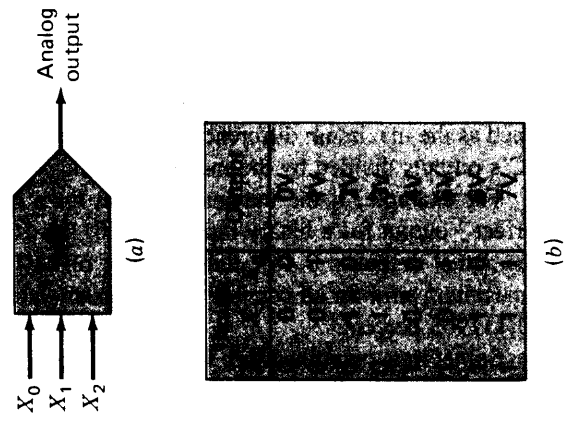
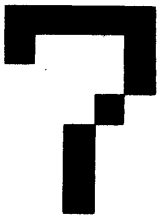


FIGURE 7.23

DAC operation. (*a*) Block diagram. (*b*) List of input-output values. (*c*) Staircase DAC output.



are equal to 1. This is a general principle for DACs: Each input has a *weight*, and the output voltage is the sum of the weights for which the binary inputs are 1s.

In this DAC, X_2 is the *most significant bit* (MSB) and X_0 the *least significant bit* (LSB).

If a three-flip-flop counter with positive-edge triggering were connected to the DAC's three inputs with X_0 connected to the counter's LSB and the counter were clocked, the input-output relation would be as in Fig. 7.23(*c*). The analog waveform shown here is called a *staircase*.

The minimum and maximum values which the analog output of a DAC can take vary for different DACs. Some manufacturer's DACs have only a single built-in minimum and maximum while others allow users to introduce reference signals which will control the minimum and maximum output voltages. In most cases the minimum output voltage for a DAC will be 0 V.

If the maximum output voltage of a DAC is V volts[5] and if the resolution is R bits, then the weight of the least significant bit will be $V/(2^R - 1)$. [For our 3-bit resolution 7-V maximum output, this gives $7/(2^3 - 1) = 7/7 = 1$ V.] The weight of the second least significant bit will be $2V/(2^R - 1)$, the next least significant bit has weight $4V/(2^R - 1)$, and so on up to the most significant bit which has weight $(2^{R-1})V/(2^R - 1)$. [For our example this is $(2^{3-1})7/(2^3 - 1) = 4$.]

As a further example, if a DAC had a maximum output voltage of 10 V and a resolution of 8 bits, the least significant bit would have weight.

$$\frac{10}{2^8 - 1} = \frac{10}{255} \simeq 0.0392157$$

This means that if an 8-bit counter is connected to the DAC's input, the staircase at the output will have steps of $\frac{10}{255}$ V from 0 to +10 V, and there will be 256 steps (counting the 0 step).

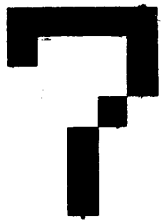
Since actual DACs are made of physical devices, they are imperfect and will have analog outputs which will not exactly be at the output for a "perfect" DAC. To give the user some idea of the size of the DAC's errors, the manufacturers of DACs generally specify the accuracy of the converter. The *absolute accuracy* is defined as the maximum difference between the actual DAC's outputs and a perfect DAC's outputs divided by the maximum analog output value.

For example, if the maximum output for a 6-bit DAC is +10 V, then the "perfect" output for a binary input of 000011 is $3 \times 10/(2^6 - 1) \simeq 0.4761905$. If the actual output is 0.465, then an error of 0.0111905 will exist; and if this is the maximum error for all possible inputs, the absolute accuracy will be 0.0111905/10 = 0.111905 percent.

Sometimes manufacturers will simply specify the absolute value in general terms such as "less than $\frac{1}{2}$ LSB," meaning that the maximum error between perfect and actual values will never exceed half the weight of the least significant bit.

If the maximum error in a DAC is less than half the weight of the least significant bit, then the output will be *monotonic*, which means the output voltage will always

[5]In all these examples the DAC is assumed to have outputs ranging from 0 to V volts. DACs with nonzero minimum values are dealt with later.



increase when the input value to the converter increases. For DACs with many bits of resolution, the manufacturer will sometimes only specify that the DAC is monotonic instead of giving an accuracy figure.

Some DACs can be set to have analog outputs which do not range from 0 to a positive voltage, but rather are in an interval from V_1 to V_2. (For instance, V_1 might be -5 V and V_2 might be $+10$ V.) The same principles exist except that the voltage V in the formulas for the weights is obtained by subtracting V_1 from V_2 and weights are added to V_1 to obtain output values. This would give 15 for the -5- to $+10$-V example, and the weight of the LSB for a 4-bit DAC with lower voltage -5 V and maximum $+10$ V would be 1 V. The outputs would then be $-5, -4, -3, \ldots, 9, 10$.

When DACs have the ability to be set to some interval, the manufacturer will generally specify two input lines to which the user can connect two input voltages which will control the DAC's upper and lower output limits.

ANALOG-TO-DIGITAL CONVERTERS—SHAFT ENCODERS

7.16 The most used type of an analog-to-digital converter which directly converts a physical position to a digital value is the *shaft encoder*. The shaft encoder is connected to a rotating shaft and reads out the angular position of the shaft in digital form.

In the A-to-D converter in Fig. 7.24(*a*), a coded-segment disk which can rotate is coupled to a shaft. A set of brushes is attached so that a single brush is positioned in the center of each concentric band of the disk. Each band is constructed of several segments made of either conducting material (the darkened areas) or some insulating material (the unshaded areas). A positive voltage is connected to the conducting sections. If a given brush makes contact with a segment of conducting material, a 1 signal will result; but if the brush is over the insulating material, the output from the brush will be a 0. The four output lines of the coder shown represent a 4-bit binary number. There are 16 distinct intervals around the

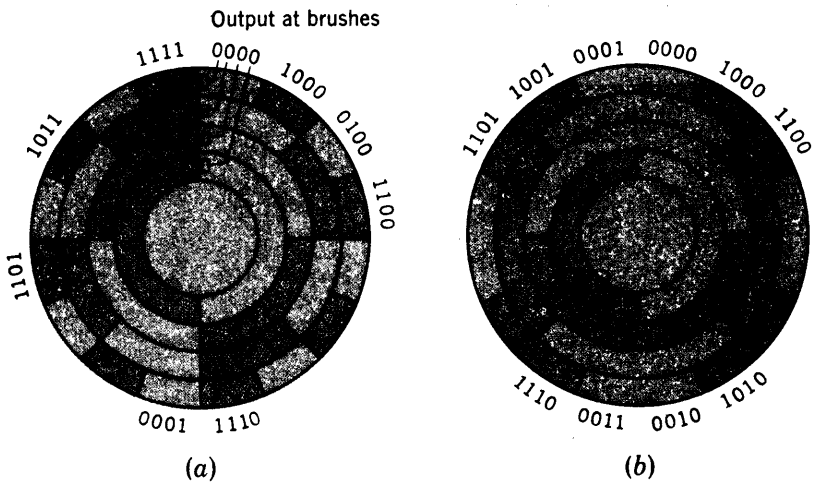


FIGURE 7.24

Shaft-position encoder disks. (*a*) Binary-coded disk. (*b*) Unit-distance code disk.

coder disk, each corresponding to a different shaft-position interval, and each causes the coder to have a different binary number output.

Photoelectric coders are constructed by using a coder disk with bands divided into transparent segments (the shaded areas) and opaque segments (the unshaded areas). A light source is put on one side of the disk, and a set of photoelectric cells on the other side, arranged so that one cell is behind each band of the coder disk. If a transparent segment is between the light source and a light-sensitive cell, a 1 output will result; and if an opaque area is in front of the photoelectric cell, there will be a 0 output. By increasing the number of bands around the disk, more precision may be added to the coder. The photoelectric type of coder has greater resolution than the brush type, and even greater resolution may be obtained by using gears and several disks. The state of the art is about 18 bits or 2^{18} positions per shaft revolution, but most commercial coders have 14 bits or fewer.

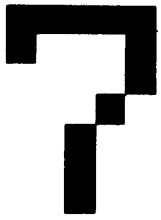
There is one basic difficulty with the coder illustrated: If the disk is in a position where the output number is changing from 011 to 100, or in any position where several bits are changing value, the output signal may become ambiguous. Since the brushes are of finite width, they will overlap the change in segments; and no matter how carefully it is made, the coder will have erroneous outputs in several positions. If this occurs when 011 is changing to 100, several errors are possible; the value may be read as 111 or 000, either of which is a value with considerable error. To circumvent this difficulty, a number of schemes have been devised, generally involving two sets of brushes with one set displaced slightly from the other. By logically choosing from the outputs available, the ambiguity may be eliminated at a slight cost in accuracy.

Another scheme for avoiding ambiguity involves the use of a *Gray*, or *unit-distance, code* to form the coder disk [Fig. 7.24(*b*)]. In this code, 2 bits never change value in successive coded binary numbers. By using a Gray-coded disk, a 6 may be read for a 7 or a 4 for a 5, but larger errors will not be made. Table 7.2 shows a listing of a 4-bit Gray code.

If the inputs to the machine are from a coder using a Gray code, the code groups must be converted to conventional binary or BCD before use.

TABLE 7.2

DECIMAL	GRAY CODE $a_3a_2a_1a_0$
0	0000
1	0001
2	0011
3	0010
4	0110
5	0111
6	0101
7	0100
8	1100
9	1101
10	1111
11	1110
12	1010
13	1011
14	1001
15	1000



There are straightforward ways to convert from Gray to binary or binary to Gray code. The conversion from binary to Gray code is as follows:

1 The leftmost digit of the binary number is also the leftmost digit of the Gray code.

2 The mod 2 sum ($0 \oplus 0 = 1 \oplus 1 = 0$ and $1 \oplus 0 = 0 \oplus 1 = 1$) of the two leftmost digits in the binary number will give the second leftmost digit in the Gray code.

3 The mod 2 sum of the second and third digits of the binary number give the third leftmost digit of the Gray code. This rule continues until the mod 2 sum of the two rightmost digits of the binary number give the rightmost Gray code digit.

Here is an example of conversion of 0111 binary to Gray code:

0111 binary	
0	leftmost digit
$0 \oplus 1 = 1$	2d leftmost digit
$1 \oplus 1 = 0$	next digit
$1 \oplus 1 = 0$	rightmost digit

So Gray code for 0111 binary is 0100.

Here is an example of conversion from 1010 binary to Gray code:

1010 binary	
1	leftmost digit
$1 \oplus 0 = 1$	2d leftmost digit
$0 \oplus 1 = 1$	next digit
$1 \oplus 0 = 1$	rightmost digit

Thus Gray code for 1010 binary is 1111.

Here is a five-digit example of converting 10111 to Gray code:

10111 binary	
1	leftmost digit
$1 \oplus 0 = 1$	next digit
$0 \oplus 1 = 1$	next digit
$1 \oplus 1 = 0$	next digit
$1 \oplus 1 = 0$	rightmost digit

So 10111 binary is 11100 in Gray code.

ANALOG-TO-DIGITAL CONVERTERS

7.17 When an analog voltage must be converted to a digital number, an analog-to-digital converter (ADC) is used. Figure 7.25(*a*) shows the block diagram symbol for a small ADC with a single analog input and 3 binary bits of output. The CONVERT input is normally a 0 and is changed to a 1 signal when a conversion

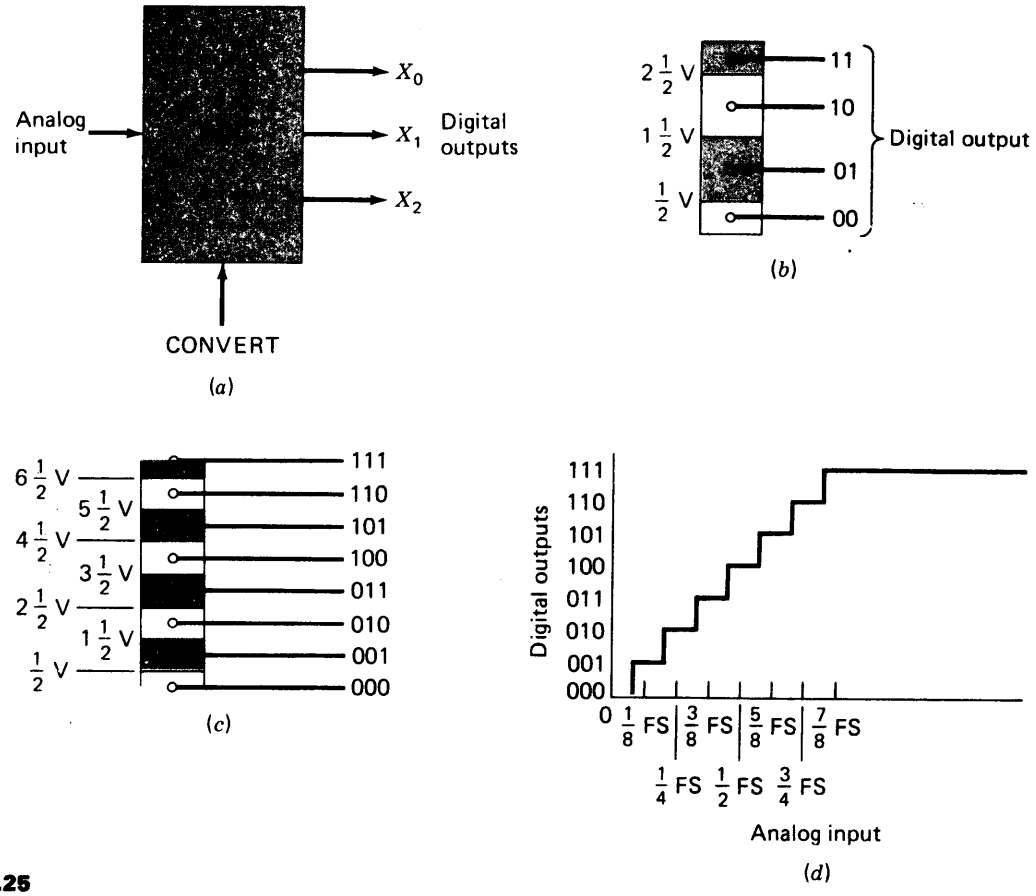


FIGURE 7.25

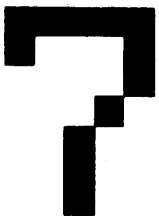
ADC operation and conversion intervals. (*a*) Block diagram (*b*) Two-output converter intervals. (*c*) Three-output converter intervals. (*d*) Graph of intervals.

is to occur. The ADC responds to the position transition on CONVERT by measuring the input voltage on the analog input and then outputting a binary number on the X outputs which represents the input voltage in digital form.

Converting an analog input signal such as a voltage to a digital number is called *quantizing* the input. Since the input can take infinitely many different voltage values and the digital representations will have only some finite number of values, each digital number at the output actually covers an interval of input values.

Figure 7.25(*b*) shows a graph of the input signal versus digital output numbers for a 2-bit ADC. The input voltage range is to be from 0 to 3 V, and the digital numbers at the outputs will range from 00 to 11. The output number 00 indicates the input voltage is in the interval from 0 to 0.5 V, the output number 01 indicates an input voltage value from 0.5 to 1.5 V, the number 10 indicates an input from 1.5 to 2.5 V, and the number 11 indicates an input of greater than $2\frac{1}{2}$ V.

This is the normal and most used system for ADCs. In this case, for example, the input voltage interval for the output number 01 has its center at 1 V, the interval



for 10 has its center at 2 V, etc. This means when the ADC reads out a 10, for example, the input is 2 V plus or minus $\frac{1}{2}$ V, as we would expect.

Figure 7.25(*c*) shows the intervals for a 3-bit converter which has a normal input voltage range from 0 to 7 V. In this example, the output 011 indicates the input voltage is 3 V $\pm$ $\frac{1}{2}$ V.

The final graph in Fig. 7.25(*d*) shows an analog input along the horizontal axis and digital values along the vertical axis. This graph is included in many manufacturers' manuals and specification sheets and again shows that the 0 interval is half the size of the other intervals which are of the same size (except that the final interval extends on).

In many converters there is an *overrange* feature to handle inputs outside the normal interval. This generally consists of an output line which indicates the input is "out of range" when it is 1.[6]

FLASH CONVERTERS

7.18 The fastest ADCs are called *simultaneous*, or *flash*, converters. Figure 7.26(*a*) shows a flash converter with two digital output lines X_0 and X_1. This converter realizes the converter input-output relations for Fig. 7.25(*b*).

The converter uses an analog circuit called a *comparator*. The block diagram symbol for a comparator is a triangle on its side. When the voltage at the upper (+) input to a comparator is relatively positive with respect to the lower (−) input, the comparator outputs a digital 1; when the upper input is negative with respect to the lower input, the comparator outputs[7] a digital 0. As an example, the lowest comparator in Fig. 7.26 has a lower (−) input of $\frac{1}{2}$ V. If the INPUT is at $\frac{1}{4}$ V, the comparator will have a 0 output; but if the INPUT is at $\frac{3}{4}$ V, the comparator will have a 1 output.

Analysis of the operation of this flash converter is as follows. (1) If the INPUT is at 0 to $\frac{1}{2}$ V, the three points *A*, *B*, and *C* will all be 0s and the X_0 and X_1 outputs will also be 0s. (2) If the input is from $\frac{1}{2}$ to $1\frac{1}{2}$ V, then points *A* and *B* will be 0s but point *C* will be a 1, and the X_0 output will be a 1 and the X_1 output a 0. (3) If the INPUT is between $1\frac{1}{2}$ and $2\frac{1}{2}$ V, points *B* and *C* will be 1s and *A* a 0, giving $X_0 = 1$ and $X_1 = 1$. (4) If the INPUT is greater than $2\frac{1}{2}$ V, then *A*, *B*, and *C* will be 1s and the output will be $X_0 = 1$ and $X_1 = 1$.

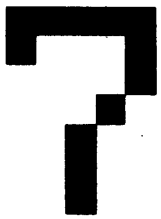
This example of a small flash converter shows the basic parts: the reference voltages, the comparators, and a gate network to connect the outputs from the comparators to the proper binary number.

Figure 7.27(*a*) shows a block diagram of a 6-bit flash converter which is packaged in a single IC container. This ADC operates at up to 100 million conversions per second. The resistor chain at the left provides the correct reference voltages for the inputs to the comparators. The CONV and $\overline{\text{CONV}}$ inputs which

---

[6]The overrange feature reduces the size of the first highest output interval to half the size of the other interval.

[7]The output of a comparator is unspecified when both inputs are equal (it can be a 0 or 1).



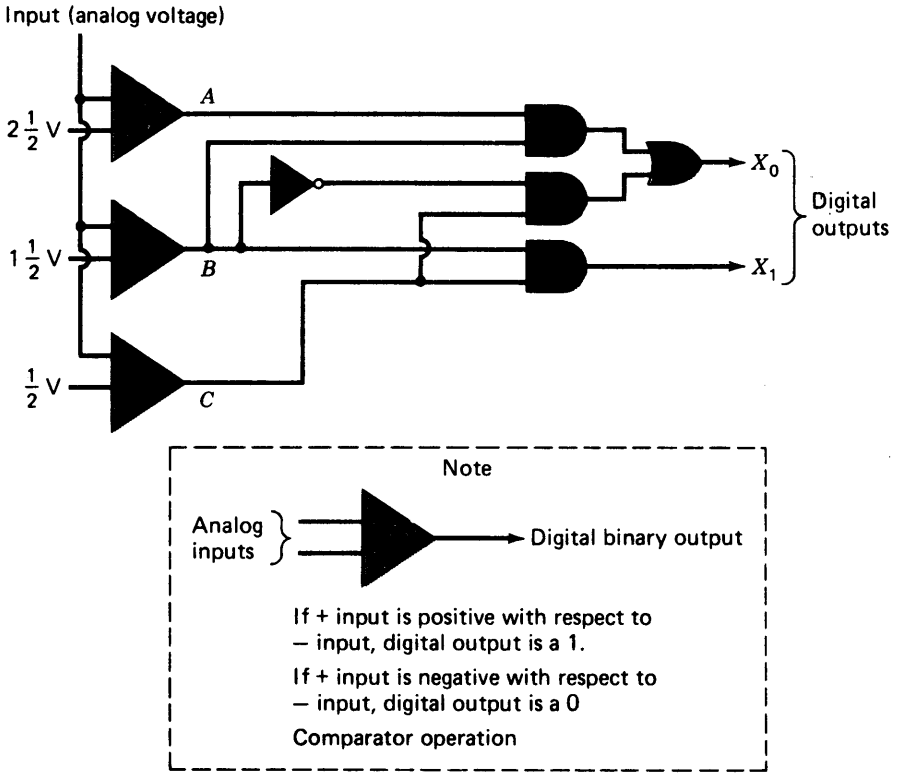


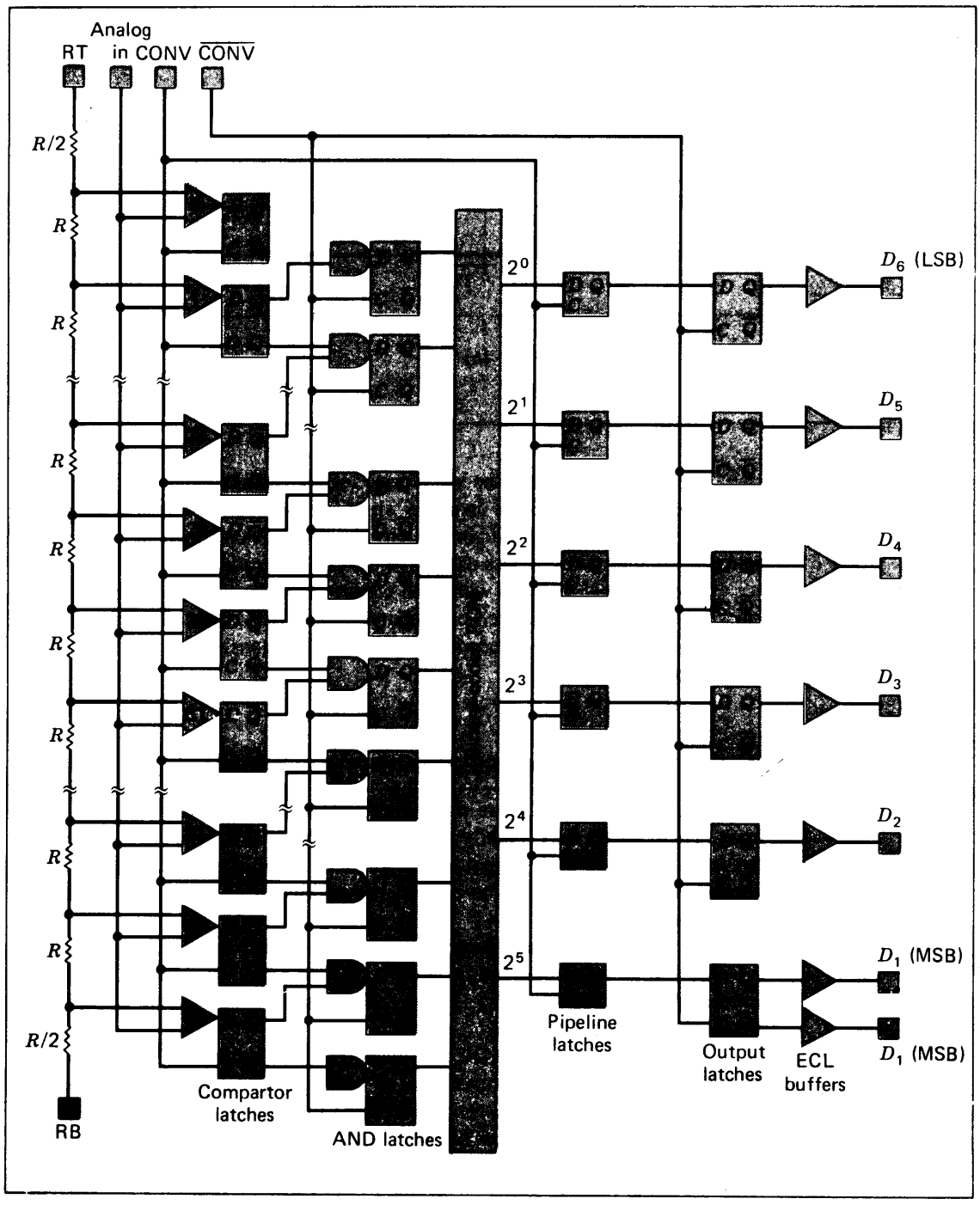
FIGURE 7.26

A 2-bit flash converter.

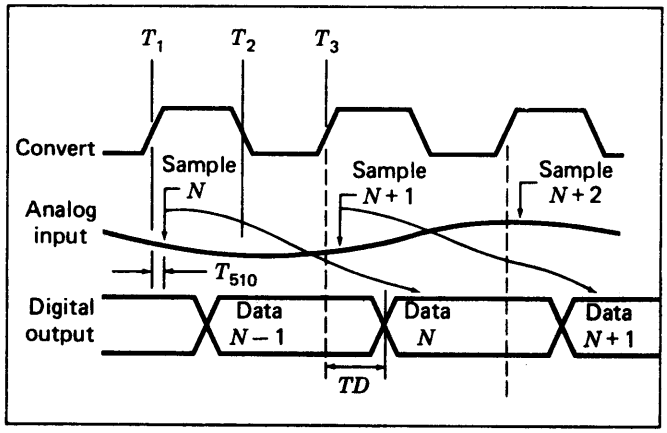
control conversions operate as follows: When the CONV is made a 1, the outputs from the 63 comparators go into the 63 latches (flip-flops) to their right. The *C* on the latches is the clock input which works on a positive 1 level. When $\overline{\text{CONV}}$ goes to 0, this transfers the outputs from these latches into the latches to their right. The ROM contains the conversion codes for the inputs from the latches, converting these inputs to the correct binary number. This ROM replaces the gate network shown in the preceding figure and provides an example of how ROMs and gate networks are sometimes interchangeable. When $\overline{\text{CONV}}$ is next a 1, the outputs from the ROM are transferred into the latches to its right. These latches now contain the correct conversion number. Finally, when $\overline{\text{CONV}}$ is made a 1, the output latches take the six output values for the ADC. (The triangles to the right of these latches are simply amplifiers or buffers providing drive for chip output; they do not shift 0 and 1 output levels from the latches.) Notice the upper and lower voltages which establish the interval through which the ADC converters are set by connecting the desired voltages to the *RT* and *RB* inputs to the ADC's container.

FIGURE 7.27

A high-speed flash A-to-D converter. (*Courtesy TRW, LSI Products Division*) (*a*) Block diagram of 6-bit 100-MHz flash converter. (*b*) Waveforms for flash converter in (*a*).


Analog in
RT
CONV
$\overline{CONV}$
R/2
R
R
R
R
R
R
R
R
R
R/2
RB
Compartor latches
AND latches
2^0
2^1
2^2
2^3
2^4
2^5
Pipeline latches
Output latches
ECL buffers
D_6 (LSB)
D_5
D_4
D_3
D_2
D_1 (MSB)
D_1 (MSB)

(a)



(*b*)

FIGURE 7.27 (*Cont.*)

Figure 7.27(*b*) shows the waveforms for conversion.

Flash converters come in many sizes and speeds. The fastest single-chip converters now perform 8-bit conversions at a 100 million conversions per second rate, but smaller, slower, less expensive devices are readily available.

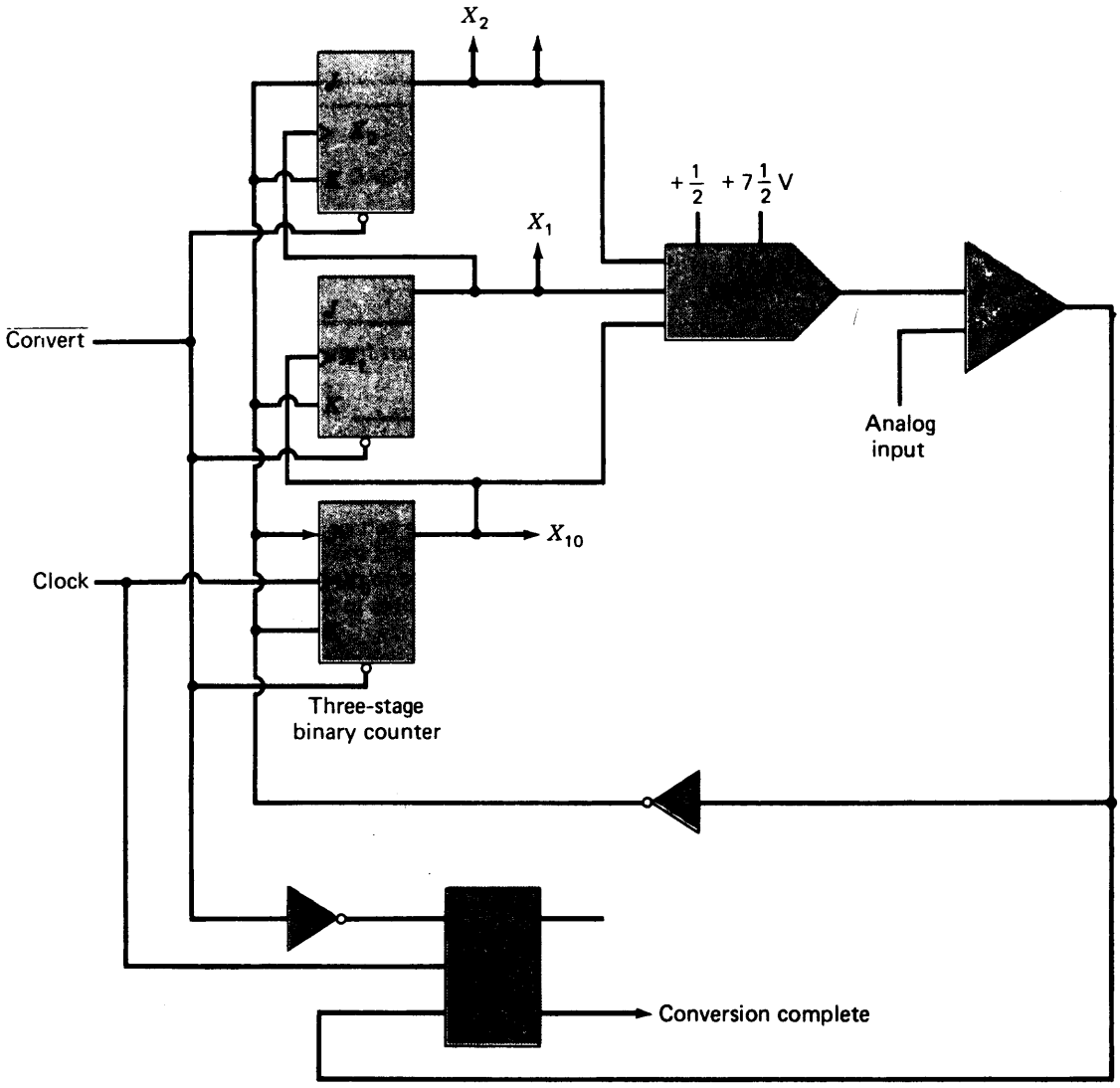
The principal problem with flash converters is the large number of comparators required as the number of output bits increases. For an n-bit converter, $2^n - 1$ comparators are needed, and this involves considerable circuitry if n is very large.

COUNTER AND SUCCESSIVE-APPROXIMATION CONVERTERS

7.19 Quite often ADCs are made from a DAC and some flip-flops and other logic. These ADCs are generally slower than the flash ADCs but are less expensive and ordinarily have more output bits (resolution) and thus greater accuracy. (Flash converters can be combined to give more bits, but more logic and one or more DACs are required.)

The most conceptually straightforward nonflash ADC involves a binary counter, a DAC, and a comparator and is called a *counter* ADC. The block diagram for a counter ADC is shown in Fig. 7.28(*a*). This uses a 3-bit counter, a 3-bit input DAC, and a comparator.

A conversion by this ADC is initiated by lowering the $\overline{\text{CONVERT}}$ line and then raising it (this line is normally a 1, so making it a 0 tells the ADC to convert). The actual conversion begins when the $\overline{\text{CONVERT}}$ line is returned to the 1 state which "frees" the counter that has been reset. The CLOCK input is supplied with clock signals continuously, and the three-flip-flop counter then begins to count. Also while the $\overline{\text{CONVERT}}$ is down, the CONVERSION COMPLETE flip-flop is set to 0 ($\overline{\text{CONVERT}}$ must be a 0 long enough for this flip-flop to be set).



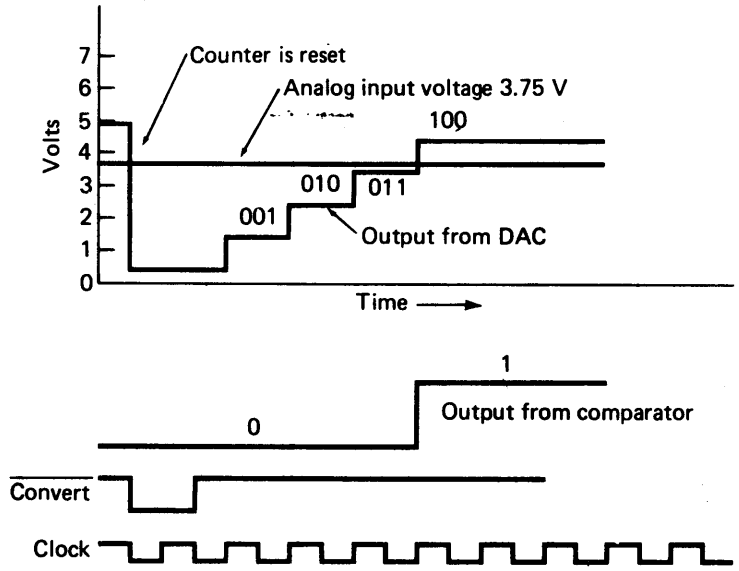
(*a*)

FIGURE 7.28

Counter ADC.
(*a*) Block diagram.
(*b*) Waveforms.

As the counter counts, the output from the DAC begins to increase in voltage[8] as shown in Fig. 7.28(*b*). Until the output from the DAC exceeds the analog input voltage, the comparator output will be a 0 and the *JK* inputs to the counter will be 1s (notice the inverter) and so the counter will count. When the DAC's output exceeds the analog input, the counter will be stopped and the CONVERSION COMPLETE signal will go to 1, indicating the conversion is complete.

[8]Note the DAC is biased $+\frac{1}{2}$ V positive. That is, a 0 input to the DAC would give a $+\frac{1}{2}$ V output. The DAC then increases its output voltages in 1-V steps as the counter is incremented.



Note: Either a voltage equal to ½ the voltage for the LSB must be added to each DAC output, as is shown here, or a voltage equal to ½ LSB must be added to the input voltage and a 1 subtracted from the final number in the counter.

(*b*)

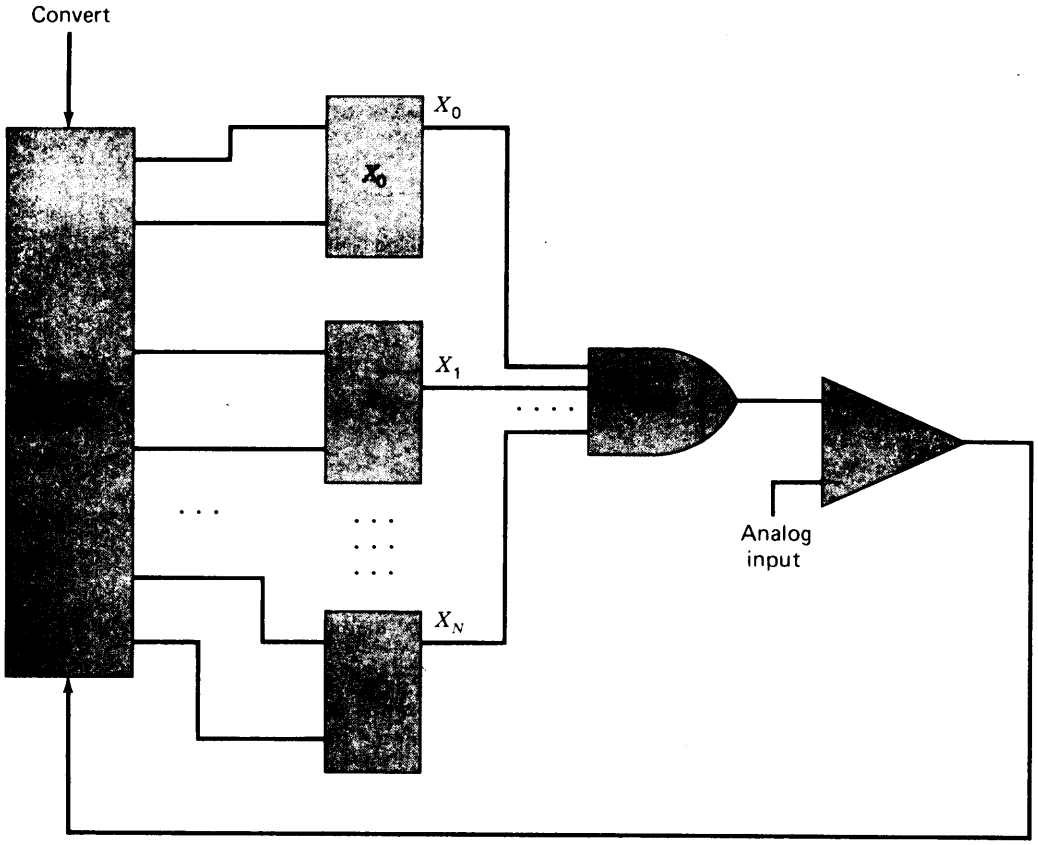
FIGURE 7.28 (*Cont.*)

As can be seen, the counter ADC is less expensive but slower than the flash converter. Another type of ADC, called a *tracking* ADC, simply follows the analog input up and down continuously, giving a continuous output of its value. This is formed by connecting the flip-flops in an up-down counter and then connecting the comparator's output to the DOWN input and the inverted comparator's output to the UP input. In this way the counter-DAC combination will continually track the analog input signal.

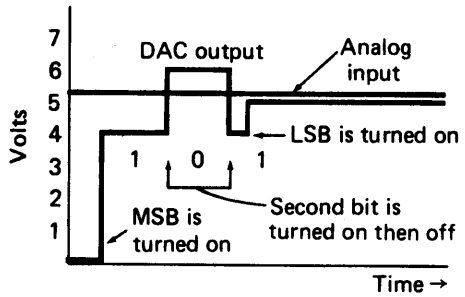
The most used ADC of this general type is the *successive-approximation* ADC. Figure 7.29(*a*) shows an ADC similar to the counter ADC but with control logic where the counter logic was. This ADC works as follows. First, all flip-flops are set to 0. Then the most significant bit (MSB) is set to 1. The output of the comparator is examined by the control logic: if it is a 1, the MSB flip-flop (the flip-flop connected to the MSB of the DAC) is turned off; if it is a 0, the flip-flop is left on. Next the second least significant bit's flip-flop is turned on. Again, if the comparator's output is a 1, the flip-flop is turned off; if the comparator's output is a 0, then the flip-flop is left on. This continues for each flip-flop up to and including the LSB flip-flop. Thus the final number in the flip-flop will represent the input voltage.

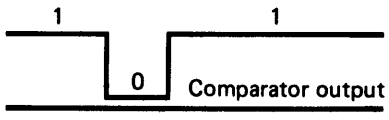
Figure 7.29(*b*) shows the waveform for a 3-bit successive-approximation ADC.

The important thing to notice is that a counter ADC with n binary outputs can take up to $2^n - 1$ clock signals to convert an input and will take $(2^n - 1)/2$



(a)



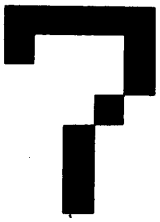


(b)

Note: Either a voltage equal to ½ LSB must be added to the analog input voltage or the output from the DAC must biased down ½ LSB

FIGURE 7.29

Successive-approximation ADC.



INPUT-OUTPUT DEVICES

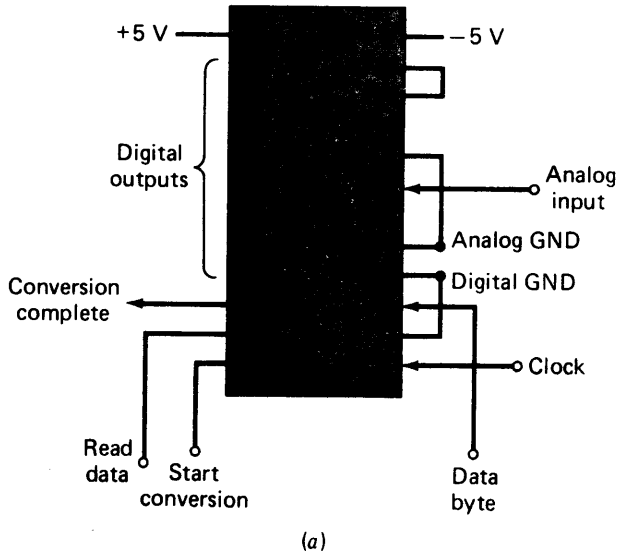


FIGURE 7.30

A successive-approximation A-to-D converter. (*Courtesy Advanced Micro Devices Inc.*) (*a*) Pin-out for AM 9517 showing input-output signals. (*b*) Waveforms for (*a*) and (*c*) showing how C/$\overline{D}$ controls multiplexing of digital output. (*c*) Block diagram of successive-approximation ADC in (*a*) and (*b*).

steps on the average. The successive-approximation ADC requires only n clock signals or steps to make each conversion. For a 12-bit ADC then, the successive-approximation ADC would require 12 steps while the counter ADC would require 4095/2 steps on average and could require up to 4095.

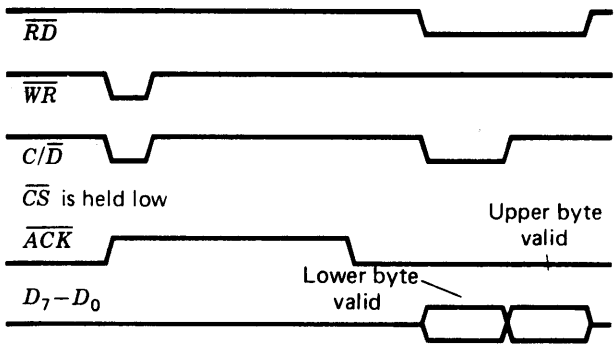
ADCs are packaged in IC containers or on printed-circuit boards when several IC chips are used; they are also sold in instrument cabinets. The block diagram and input-output signal lines for a 12-bit ADC which uses the successive-approximation technique and has a 3-μs conversion time are shown in Fig. 7.30(*a*) and (*c*). Waveforms for this device are seen in Fig. 7.30(*b*). Lowering $\overline{WR}$ to 0 initiates a conversion. At this time the $\overline{ACK}$ line goes high (a 1), and it goes back to 0 when the conversion is complete. Data are read from the chip in two steps.[9] (In order to conserve on in/out connections, the data output lines are time-multiplexed.) When RD is low, the ADC outputs appear on D_0 to D_7. If C/$\overline{D}$ is low, when RD is low, the 8 least significant bits are output on D_0 to D_7. If C/$\overline{D}$ is high, the 4 most significant bits appear on D_0 to D_3. $\overline{CS}$ is a chip select input (as in memories) which enables the ADC when low (a 0); holding this line high disables the chip (this permits easy connection to microcomputer buses). GND_A is analog ground, and GND_D is digital ground.

The outputs and inputs are TTL levels; approximately 0 V = 0 and 3.5 V = 1.

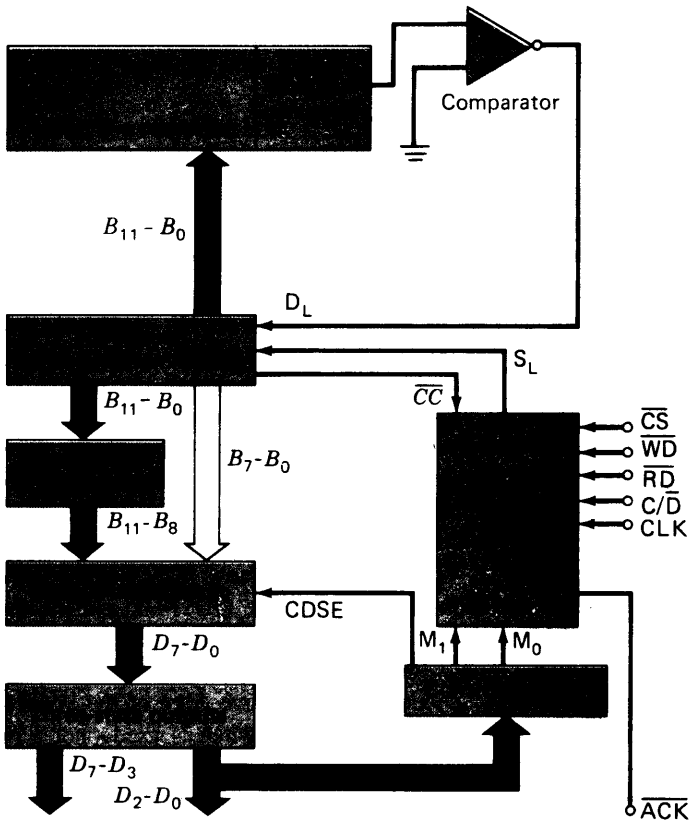
COMPUTER DATA ACQUISITION SYSTEMS

7.20 Computer data acquisition and computer control systems are important areas in computer technology. Computers have long been used to make measurements in laboratories and to monitor and control manufacturing processes. Machine

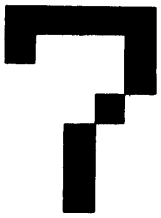
[9]These outputs utilize three-state drivers which are explained in Chap. 8.


$\overline{RD}$
$\overline{WR}$
$C/\overline{D}$
$\overline{CS}$ is held low
$\overline{ACK}$
D_7-D_0
Upper byte valid
Lower byte valid

(*b*)


Comparator
B_{11} - B_0
D_L
S_L
$\overline{CC}$
B_{11}- B_0
B_7-B_0
B_{11}-B_8
CDSE
D_7-D_0
M_1
M_0
D_7-D_3
D_2-D_0
$\overline{CS}$
$\overline{WD}$
$\overline{RD}$
$C/\overline{D}$
CLK
$\overline{ACK}$

(*c*)



tools are often controlled by computers as are plastic forming machines and other industrial devices. Now the advent of microprocessors is moving the level of control down to consumer items such as automobiles, stoves, temperature control of houses, etc. For example, for some time the automobile industry has used computers to test motors and perform other manufacturing operations. Now cars themselves have computer controls. Some Ford models use a microcomputer-based logic control system with inputs from six sensors. A vane airflow meter in the air induction system outputs a voltage proportional to the amount of air drawn into the engine. The temperature of this air is also measured by a second sensor while still another gives engine temperature. The amount of free oxygen in the exhaust gas is measured, too, as is the crankshaft position. By using data from these sensors as input, the ignition spark timing is set, and the amount of fuel discharged through the injector nozzles is controlled, as is the exhaust-gas recirculation system. Other manufacturers use computers to control transmissions, shock absorber pressure, fuel pumps, etc.

All the above applications rely on the measuring of analog inputs by computer, which involves extensive use of ADCs. As may be seen, in these and in laboratory data systems, blood processing laboratories, computer patient monitoring, and many other such systems, different sensors must be monitored by the computer continuously. This general area is called *data acquisition* and is rapidly expanding.

Figure 7.31(*a*) shows a section of a waveform to be monitored by a computer. This waveform is to be sampled periodically, and the value at sample points input to the computer.

The sampling process is not perfect, however. If counter or successive-approximation types of ADCs are used, the conversion times are liable to be long; and if the signal is changing during the conversion process, the output from the ADC can represent almost any point in the sampling interval. Figure 7.31(*b*) shows an expanded section of waveform, and clearly the output from the ADC could be from any point on this curve.

The indeterminacy in output value caused by the change of input during the sample interval may or may not be important. However, if the analog signal changes rapidly so that a significant change can occur during the sample interval, then mathematical analyses of the digital inputs, or attempts to reconstruct the original input signal from the sample values, can be severely degraded.

To alleviate this problem, a device called a *sample-and-hold amplifier* is often used. Figure 7.31(*c*) shows a functional representation of this device (it is an electronic circuit on a chip) and a block diagram symbol. The sample-and-hold amplifier works as follows: When a positive pulse is placed on the SAMPLE input, the current value at the input is placed on the OUTPUT and the value remains there until the SAMPLE input again is given a positive pulse. The functional representation shows a relay whose contacts are closed by the positive edge on the SAMPLE input and then opened by the negative edge of the positive pulse. (In actual practice, a high-speed semiconductor switch is used—not a relay.) When the contacts of the relay are closed, the leftmost amplifier charges the capacitor to the INPUT value. When the contacts are opened, the capacitor stores this value until the contacts are again closed.

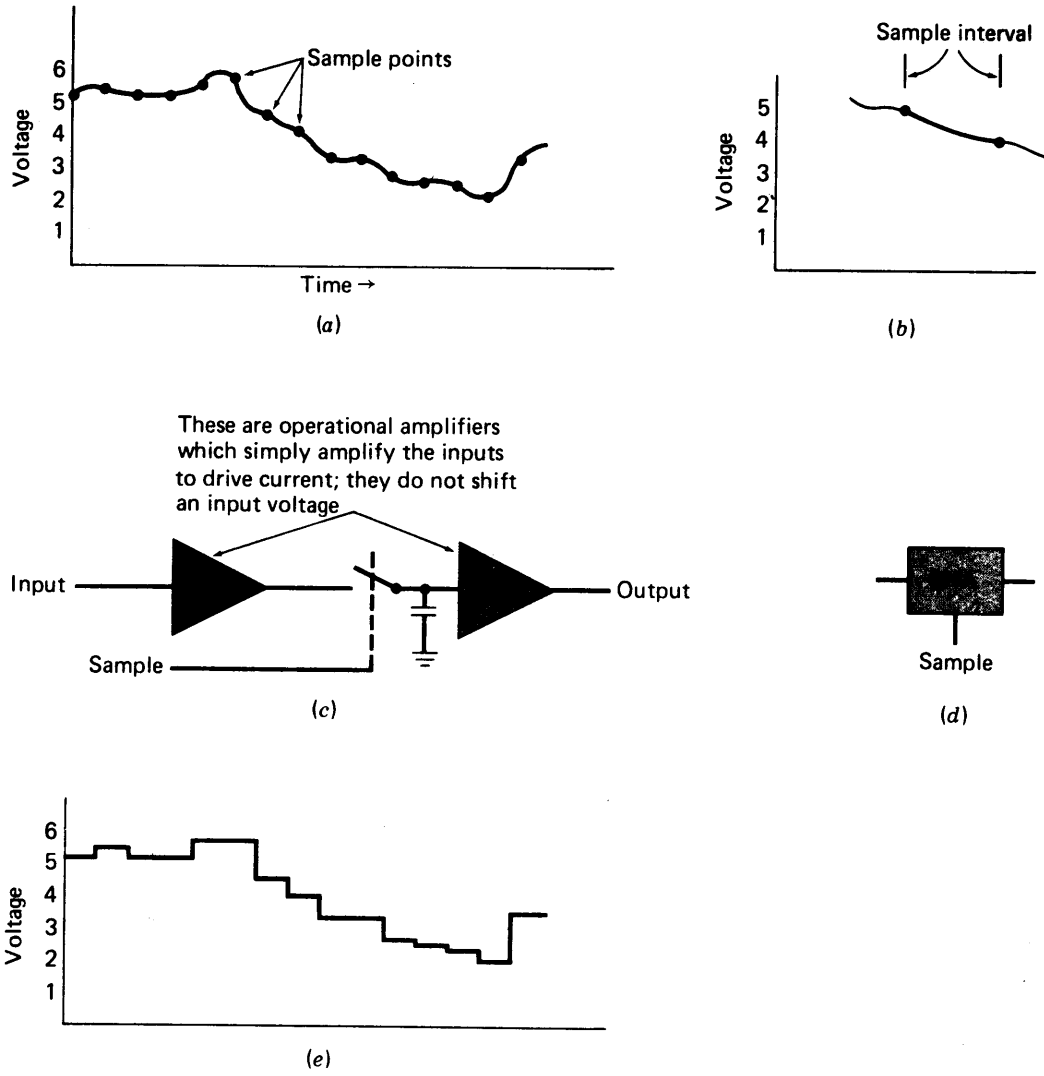


FIGURE 7.31

Sampling an analog signal. (*a*) Waveform to be sampled. (*b*) Typical sample interval. (*c*) Functional representation. (*d*) Block diagram symbol for sample-and-hold amplifier. (*e*) Output of sample-and-hold amplifier sampling waveform in (*a*).

Figure 7.31(*e*) shows the output from a sample-and-hold amplifier with input the waveform in (*a*) and sample points the same. Now, if an ADC is connected to the sample and holds output and told to convert, then the analog output from the sample-and-hold amplifier will be constant in the interval between the sample points and the ADC will have a constant input to convert. Then the output of the ADC will represent the value of the analog input at the time the sample-and-hold amplifier was told to sample.

There are important considerations concerning the operation of a sample-and-hold amplifier:

1 The *aperture time* is the time elapsing between the command to hold and the actual opening of the hold switch. (This can be as low a 1 ns for some switches.)

2 The *settling time* is the time required for the output to reach the input value when the switch is closed. If the switch is closed by the positive edge of a pulse, as in our example, and opened by the negative edge, then the time between positive and negative edges (pulse width) must be long enough for the output value to change from the prior to the new value. [This will involve charging the capacitor in Fig. 7.31(*e*).] The manufacturer's specification will give the settling time. Reasonable figures might be a 10-ns aperture delay, 0.25-ns aperture jitter (variation in delay), and 100-ns settling time.

3 *Droop* is the amount or rate of drift in the output between samples. A typical figure would be 100 μV/μs, which means a maximum change of 100 μV might occur during 1 μs.

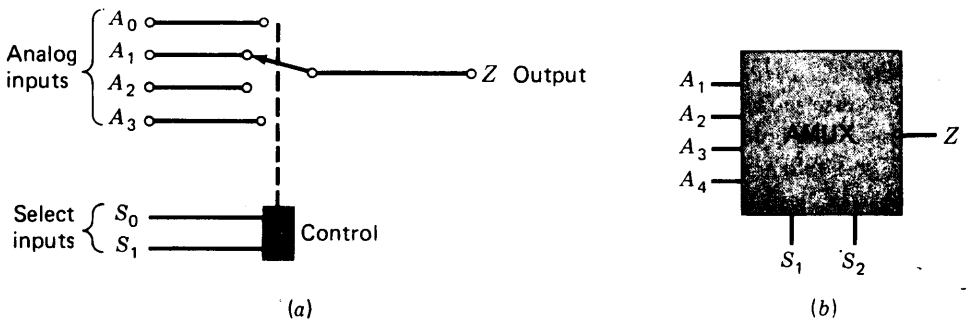
Another important device in data acquisition systems is the *analog multiplexer,* or AMUX. This is shown in Fig. 7.32(*a*). An AMUX has several analog inputs [four are shown in Fig. 7.32(*a*)], enough digital *select inputs* to select one of the analog inputs, and a single analog output. The function of the AMUX is to select one of the analog inputs from several possible inputs by using the digital select inputs and to output only this particular input. Figure 7.32(*a*) shows this in the functional diagram by a rotary switch where the position of the wiper arm on the switch selects and connects one of the four inputs to the output. In practice, AMUXs are made of semiconductors and are often packaged in a single IC container. The inputs and outputs are generally connected to amplifiers, and the switch is a semiconductor switch.

As can be seen, each select value will cause a particular input to be output. In Fig. 7.32(*a*), S_0, S_1 = 00 would cause A_0 to be output, S_0S_2 = 01 would output A_1, etc.

Figure 7.33 shows a typical *data acquisition system.* There are four analog signals, A_0, A_1, A_2, and A_3. Each is connected to the AMUX; the computer selects from these inputs the one it wants to be sampled and places its number on S_0S_1. Then the selected analog signal is connected to the sample-and-hold amplifier. When the correct time for a sample occurs, the computer raises and lowers the SAMPLE input, thereby causing the selected input voltage to be held at its value

FIGURE 7.32

Analog multiplexer. (*a*) Functional diagram of four-input analog multiplexer. (*b*) Block diagram.



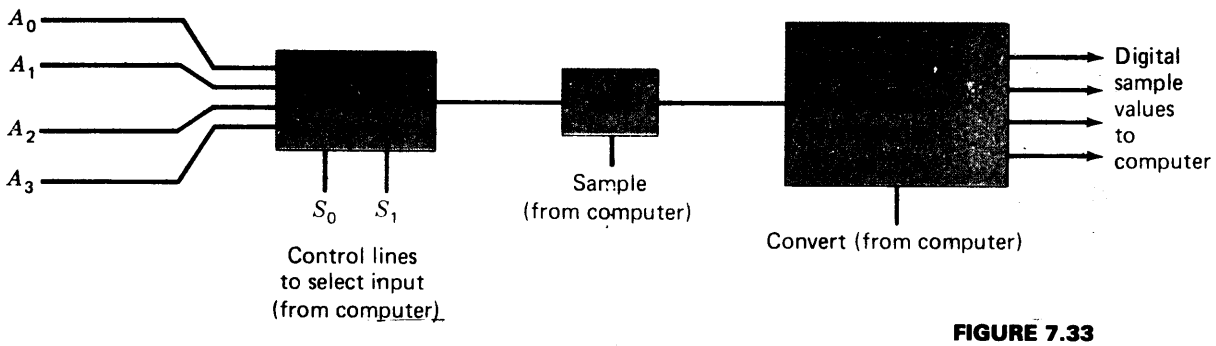


FIGURE 7.33

Data acquisition system.

at this particular time. Next the CONVERT signal to the ADC is raised, causing the ADC to measure and output the value of the selected analog signal. The computer can now read the output signal from the ADC and proceed to the sampling of another signal.

With the above data acquisition system, the analog inputs can be sampled at varying rates in case some signals change more rapidly than others or are more critical than the others. Also, the exact time at which the sample was taken will be known by the computer since it issues the SAMPLE signal (generally this signal is combined with some precise timing source).

AMUXs are produced with more than four analog inputs to handle large systems. Sometimes several sample-and-hold amplifiers are used, and these are connected to the analog inputs rather than the AMUX's output. This makes it possible to sample two or more imputs at the same time and to then read them by addressing one and then the other, using the AMUX.

Sometimes the analog signals have voltage amplitudes which are outside the normal range of the ADC. The sample-and-hold amplifiers (and sometimes the AMUX) contain amplifiers which often can be set to accommodate these inputs, increasing or decreasing signal values and even inverting (converting from negative to positive) input values. There are also circuits for *offsetting*, or *translating*, input signals which consists of adding a selected voltage to them, thereby changing their range. This area is called *signal conditioning* and generally involves the use of operational amplifiers. Some material on this can be found in the Questions and more in the Bibliography.

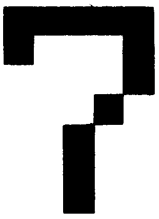
SUMMARY

7.21 This chapter presented an introduction to punched tapes, punched cards, and the punches and readers for these input-output media.

Alphanumeric codes are widely used in computer-human interfaces, and these were discussed and tables of the most used codes given.

Printers are the most used output devices, and they come in many kinds. The general operating characteristics of these and their principles of operation were presented. Oscilloscope displays were also discussed.

A typical keyboard and its operation were discussed as was the operation of terminals.



Digital-to-analog converters are used to convert binary output signals from computers to analog signals which can be used to position mechanical (and other) devices. Analog-to-digital converters are used to convert inputs from physical systems to digital form. The basic principles of both digital-to-analog and analog-to-digital converters were presented followed by examples of each, including flash and successive-approximation converters. The use of these devices in data collection and sampled data systems was discussed.

QUESTIONS

7.1 Using the code in Fig. 7.3, an operator keyboards the following:

ADD 641
CAD 932
SUB 841

How many lines will be punched in the paper tape? How many holes will be punched in the tape?

7.2 The code in Fig. 7.3 is a parity-checking code. Is the parity check odd or even?

7.3 Write out in binary form the first line of the program in Question 7.1, using the code in Fig. 7.3.

7.4 The code in Fig. 7.3 has at least one punch or hole for each character. This makes it possible for the reader to detect when to read. Explain the tape feed character.

7.5 If the short program in Question 7.1 is punched into cards according to the code in Fig. 7.5 by using normal procedures, how many holes will be punched into the cards used? If a mistake is made during keyboarding, will it be easier to correct if cards or tape are used? Explain why.

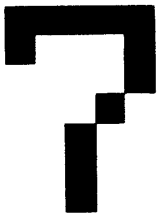
7.6 In EBCDIC, when the first 4 bits are 1s, the remaining 4 bits represent a digit. Is the code for these bits BCD or straight binary? Give a reason for your answer

7.7 Generally, programs are punched into cards with an instruction word per card. The first line of the program in Question 7.1 would go on one card. How many holes will be punched in the first card for the code in Fig. 7.5?

7.8 The tape feed character can be used to take out any character in the code in Fig. 7.3 except one. Which character and why? *Hint:* Remember the parity checks.

7.9 List the binary-code groups for each decimal digit in the excess-3 BCD code in Chap. 3, and assign a parity bit for an even-parity-bit checking system to each code group. List the values of the parity bits for the same excess-3 code for an odd-parity-bit checking system.

7.10 Discuss any problems you can foresee in attempting to read characters



optically which would not occur for magnetic characters. Do these reasons somewhat explain why banks adopted magnetic readers before optical readers?

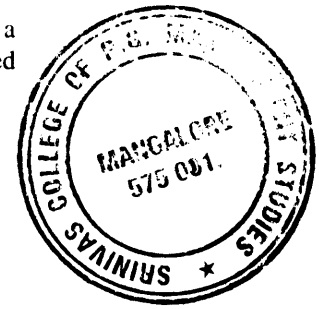
7.11 Each of the following rows of digits consists of a code group in ASCII. A single parity check has been added as the rightmost bit in each row, and a single row of parity checks has been added at the end, as explained in Question 7.15. In addition, errors have been added so that the data are not correct at present. Correct these groups of data, and then convert each seven-digit character to the alphanumeric character it represents. The parity checks are odd-parity checks.

(*a*)
10101000
10010001
10000011
10101000
01001111
10100111
01100000
10000100
10110010
10001111
10010001
10101000
11000101 parity-check row

(*b*)
10001001
10011110
10011101
01001111
10101000
01001000
10100111
10101000
10011110
10100001
01011011
10111001 parity-check row

7.12 Using the error-detecting and error-correcting scheme in Question 7.11, a message has been sent. It arrives as below. Determine whether errors have occurred and correct any you find in the following message:

10100100
10000010
10001101
10010001
10101000
01111111

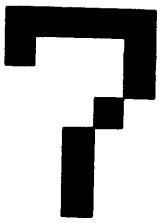


7.13 How many bands must a coder disk similar to that shown in Fig. 7.24 have for an A-to-D converter that has a precision of 10 binary digits? List the successive code groups for a 5-bit unit-distance Gray code which counts from 0 to 31_{10}.

7.14 A 3, 3, 2, 1 code for encoding the 10 decimal digits into 4 binary digits can be made so that no more than two positions change each time a single digit is increased by 1. Write this code down.

7.15 More powerful parity-check systems can be formed by adding columns with the number of 1s in each column written at the end as a binary number. For instance, if we wish to encode

1011
1101
1100
1110



we add

0
0
1
0 } parity checks

01001
01110
10000 } number of 1s in each column

10000 ——indicates two 1s

01001 indicates four 1s

Adding this to the data forms this encoded block of data:

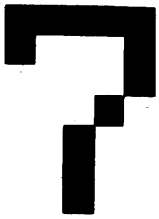
10110
11010
11001
11100
01001
01110 } check digits
10000

Now if one or more errors occur in the same column, they can be corrected by simply noting that the column does not agree with the number of 1s recorded at the end, and that there are parity checks in the rows of the block of data containing errors. By simply changing these errors, we convert the message back to its original form.

Here are two other blocks of data which include errors. Correct the errors in these blocks of data, and write the alphanumeric code for each set of seven digits in a row to the right of the rows. The code is that of Fig. 7.7, so the parity checks are as indicated in the figure, and not in the rightmost column.

(*a*) 00100011
01101010
01100011
00100001
00110010
00010000
01110110
01111001
01000101
01110101
00010100
01001101
01010011
00100000 } check digits

(*b*) 01100010
01000001
01000100
00010000
00100011
01100001
01010111
01110101
00110010
00010000
00110101
01100100
01110011
00000000 } check digits



7.16 If the ASCII code in Fig. 7.7 is transmitted serially in binary, draw the waveform for the character 6, assuming a 1 is +4 V and a 0 is −4 V.

7.17 Put errors into the message in Question 7.12 which the coding will neither correct nor detect.

7.18 Characters are generally read from punched paper tape a line at a time. When the code in Fig. 7.3(*a*) is used, the computer will be supplied with information bits each time a line is read. If the computer used is a serial computer, the bits will arrive in parallel and must be changed to serial form. By loading a seven-place shift register in parallel and then shifting the register at the machine's pulse-repetition frequency, the bits representing the character can be converted to serial form. Draw a block diagram of a seven-flip-flop shift register, along with the input lines necessary to load the register. (Assume that there are seven input lines—one to each flip-flop—from the tape reader and that a given input line will contain a pulse if a hole is in the respective position of the tape.)

7.19 Explain why, in large computer systems, output data to be printed are generally recorded on magnetic tape for offline printing and not printed at once.

7.20 Which of the sets or errors in Question 7.11 would have been detected if the error-catching system in Question 7.15 had been used? Which of these sets of errors in Question 7.15 could have been corrected by the error-detecting and error-correcting scheme in Question 7.11 and which would only have been detected?

7.21 Is it possible to invent a 7-binary-bit code which includes an odd-parity-check bit and which contains 70 characters? Give a reason for your answer.

7.22 The code in Question 7.15 is not generally able to correct double errors in a row or errors in the checking numbers at the end, but will almost always detect each of these. Explain this statement.

7.23 Show how a teletypewriter would read out the character B if this key were depressed. Draw the output waveform.

7.24 What is ASCII for "line feed," which is also LF?

7.25 What is EBCDIC for E?

7.26 Notice that the characters in the magnetic reader character set in Fig. 7.10 are "blocked," not curved. Why might this be a good idea?

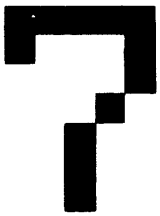
7.27 Notice the difference in a 0, a Q, and an O in Fig. 7.11. Find other letters that have similar printed shapes and have been altered to make them more machine-readable.

7.28 What is the difference between an acoustic coupler and a modem?

7.29 Why might it be a good idea for a computer to echo a character struck on a keyboard it is to read from instead of having the character printed immediately.

7.30 Why are start bits and at least one stop bit necessary for the teletypewriter code transmission explained for terminals?

7.31 Explain the difference between synchronous and asynchronous transmission of digital data.



7.32 Explain how the Bell 103 modem described would send the ASCII character G on a line in a chosen direction. How would it send in the other direction? Use the teletypewriter scheme to encode, using start and stop bits.

7.33 Design an A-to-D converter that converts using the successive-approximation technique.

7.34 Show how long it would take an A-to-D converter using the successive-approximation technique to convert 7 bits. Assume that it takes 10 ms for the D-to-A converter-resistor network to stabilize its output. Explain by showing how conversions are made for three specific input voltages.

7.35 Again assuming that it takes 10 ms for the D-to-A converter network to stabilize, show how a 6-bit converter uses the successive-approximation technique for three voltages +9, +1, and +7 V, assuming a 0- to 10-V range for inputs (0 and +10 V as voltage levels for the level converter outputs).

7.36 Show how the converter in Fig. 7.28 converts the three voltages in the preceding question and compare the conversion times.

7.37 Using the information in the preceding three questions, can you compare the average time for conversion for an A-to-D converter as shown in Fig. 7.28 with that of a successive-approximation converter? Assume 6 bits.

7.38 Show how a flash converter works for a 3-bit 0 to +7 volt system. Draw the gates from the comparator outputs to the binary numbers.

7.39 Explain how the converter in Figure 7.28 converts +0.5, +3.2, and +5.4 V.

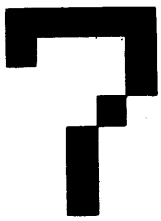
7.40 Discuss resolution and quantizing error. Can the quantizing error be less than the resolution? Why or why not?

7.41 Discuss A-to-D converters, bringing out the important characteristics which must be considered in choosing a converter. What is the primary advantage of a flash A-to-D converter, and what is its primary disadvantage? Can a flash converter convert an analog input directly to digital form, using a Gray code instead of binary? Justify your answer.

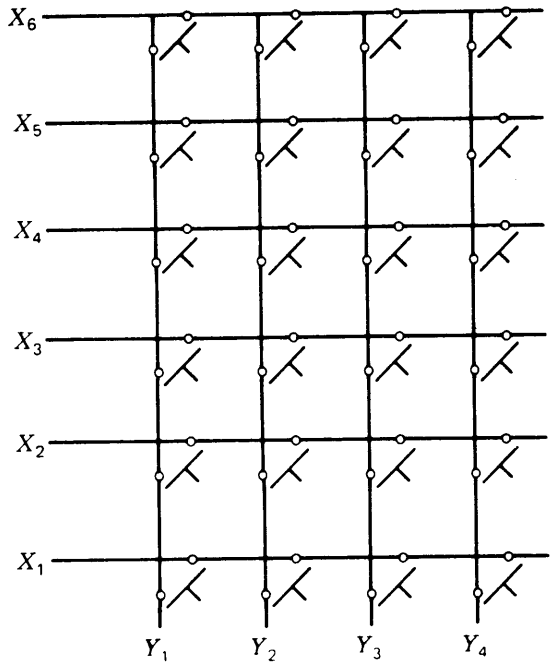
7.42 Design a gating network that converts a 3-bit Gray code to a 3-bit conventional binary number. Use only NOR gates in your design.

7.43 A straightforward technique for encoding a keyboard is shown in this chapter. Several other methods are sometimes used, and these are primarily intended to either reduce the number of semiconductors in the decoder mechanism or simplify the wiring.

One technique involves a two-dimensional array similar to the selection systems used in memories. The following figure shows a two-dimensional array. The horizontal wires are connected to the vertical wires by switches which are activated by keys on the keyboard. Thus depressing a key closes a switch which connects a single *X* wire to a single *Y* wire. Each key which is depressed produces a unique *X* wire, or *Y* wire, combination. Determining which key has been closed, however, is nontrivial. A common technique is to raise one of the *X* or horizontal wires and then scan (i.e., sample) each of the *Y* wires. If one of the wires is high and the



other wires are low, then the particular intersection of the X and Y wires which are connected can be determined. Sometimes a microprocessor is programmed to generate the X wire sequence and sample the Y wires, but special *keyboard encoding chips* are also made for this process. In each case the X wires are normally low except that, one at a time, the microprocessor, or scanner, raises a single X wire and then examines each of the Y wires to see whether one is high. If it is, a key has been depressed, and since the microprocessor is aware of which X wire has been raised and also which Y wire was raised, it can determine the unique key that has been depressed. By encoding the X wires with a unique 3-bit code on each wire and the Y wires with a unique 2-bit code, a unique pair of 5-bit combinations can be arranged. For larger keyboards the ASCII code can be used by proper choice of the X and Y values. Draw a simple 8-character encoder which encodes only 8 of the ASCII characters shown in Fig. 7.7, assigning values to the X and Y access wires.

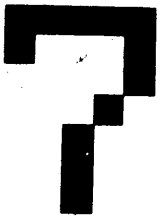


7.44 What is the weight of the least significant bit in a DAC with a resolution of 6 bits and an output voltage in the range of 0 to + 10 V?

7.45 What is the weight of the most significant bit for a DAC with 5 bits of resolution and an output voltage range of 0 to +5 V?

7.46 If a 6-bit DAC has a 0- to +5-V output range, what is the expected output for an input of 010101?

7.47 A 7-bit DAC has an output range of 0 to + 10 V. The manufacturer specifies that the DAC has a monotonic output with perfect values of 0 and + 10 V. In terms of voltage, what is the maximum error that can occur?



7.48 What is the Gray code for 10111 binary? If the Gray code is 11011, what is the corresponding binary code?

7.49 Derive a rule for converting from Gray code to binary.

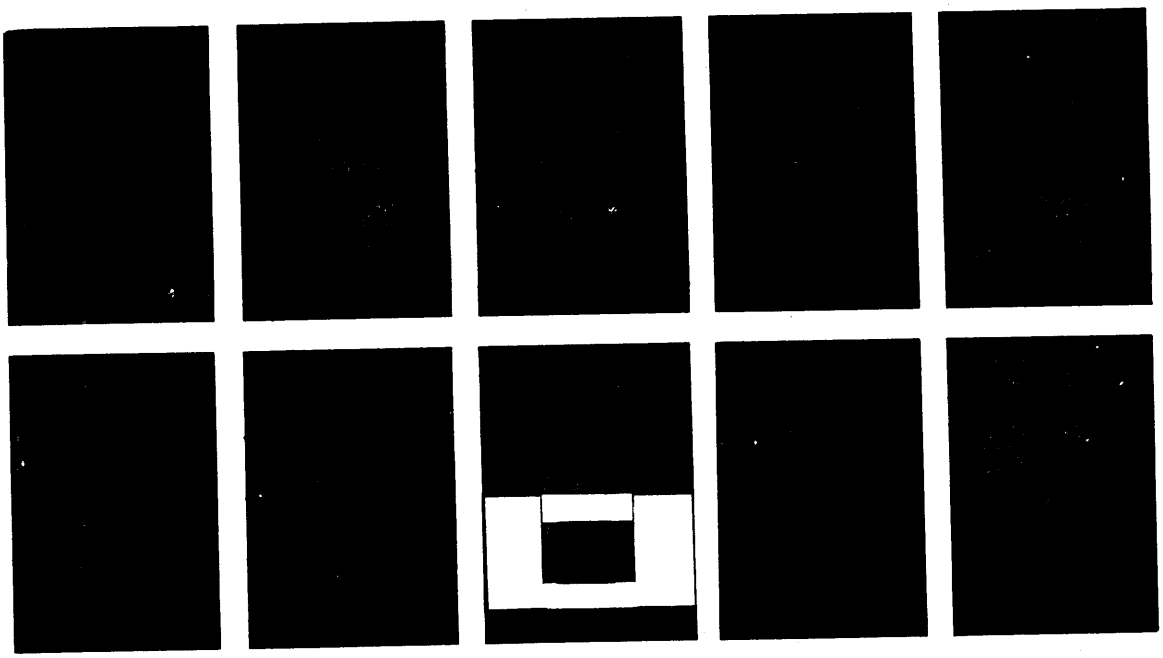
7.50 Draw the waveform from the DAC and input voltage for a 4-bit counter converter converting the input voltage of $+2\frac{3}{4}$ V. The conversion range for the ADC is 0 to +5 V.

7.51 Draw the DAC output waveform for a 5-bit successive-approximation ADC converting the input voltage of $+3\frac{1}{3}$ V. The ADC voltage has an input voltage range of 0 to +7 V.

7.52 Design a 4-bit ADC which uses the successive-approximation technique, showing gates, flip-flops, etc.

7.53 Design a 4-bit tracking ADC using an up-down counter as in Chap 4. The input voltage range should be 0 to +15 V.

7.54 Lay out the block diagram for a data acquisition system which samples the output from four DACs connected to temperature-measuring devices and a DAC connected to an accelerometer in a jet sled. Use sample-and-hold amplifiers, multiplexers, and an ADC. Have a computer control the sampling.



BUSES AND INTERFACES

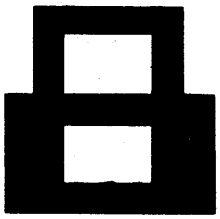
A computer can be conveniently broken into five sections, as explained in Chap. 1. It is necessary to interconnect these sections so that they will operate as a system. That is the subject of this chapter.

To interconnect memories, I/O devices, and other sections of a computer, most often a bus is used. Several examples of buses were given in Chap. 6, and buses are covered in greater depth in this chapter. When a given I/O device or memory is connected to a bus, an *interface* is required. This interface consists of the logic necessary for the I/O device or memory to successfully communicate with the bus. Since each device must be interfaced, there are at least as many interfaces in a system as there are devices. In general, whenever one part of a digital system is connected to another, the logic which effects the interconnection is called the interface.

To describe both buses and interfaces, examples from actual computers and buses are used. Also, interfaces are developed for a keyboard and printer, and a program to drive these interfaces is shown.

OBJECTIVES

1 There are several ways to interconnect system components in a computer, but the most used is a bus. The characteristics of buses are described followed by the overall organization of several computers and their interconnection strategies.



BUSES AND INTERFACES

2 When lines on a bus are shared, special circuits are used. These are discussed, followed by a description of the operational considerations. Then hand shaking for a bus and synchronous and asynchronous bus data transfers are discussed.

3 A design for a keyboard and printer interface for a microcomputer is presented. Program control of this interface is also described, and program sections are presented.

4 Input-output interrupts and other bus strategies are described for several systems. The general-purpose interface bus IEEE Standard 488-1978 is presented.

INTERCONNECTING SYSTEM COMPONENTS

8.1 The components of a computer system, that is, the memories, input-output devices, etc., must be interconnected to form a computer system. The way these components are put together and how they communicate with each other profoundly affect the system's performance characteristics.

The arithmetic-logic unit and control unit are generally placed physically together and called the central processing unit (CPU). The CPU is then "in charge" of the system's operation, directing the operation of the other parts of the system.

In the earlier computers the CPU was connected directly to each input-output device and memory unit by a separate cable for each connection.[1] This is shown in Fig. 8.1. Then, if a card is to be read from a card reader, the CPU must accept the information; and if it is to be stored in memory, the CPU must store it. The CPU is therefore central and involved directly in each transaction.

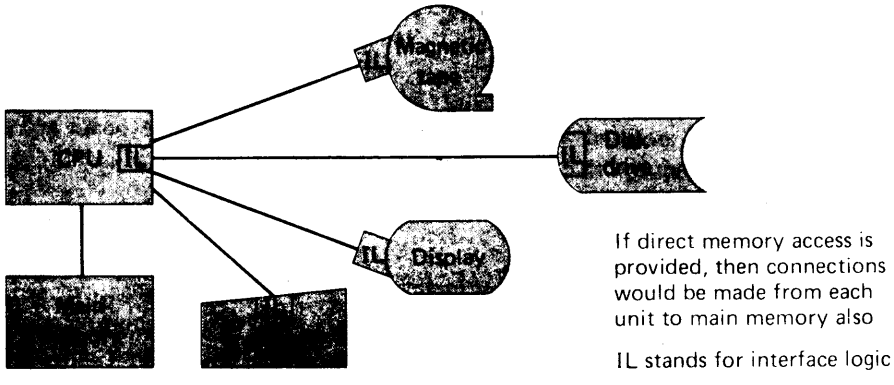
The above system has the disadvantage of many different cables and considerable interface logic (at each end of each cable).

To make interconnection of the system components less expensive and to standardize the interface logic used, a very popular technique involves interconnecting all components by using a single *bus*. This bus consists of a number of

[1] A cable is a set of electric conductors (wires). Connections are made to a cable only at each of the two ends, while a bus has connections made along the conductors.

FIGURE 8.1

Individual connections between computer system units.



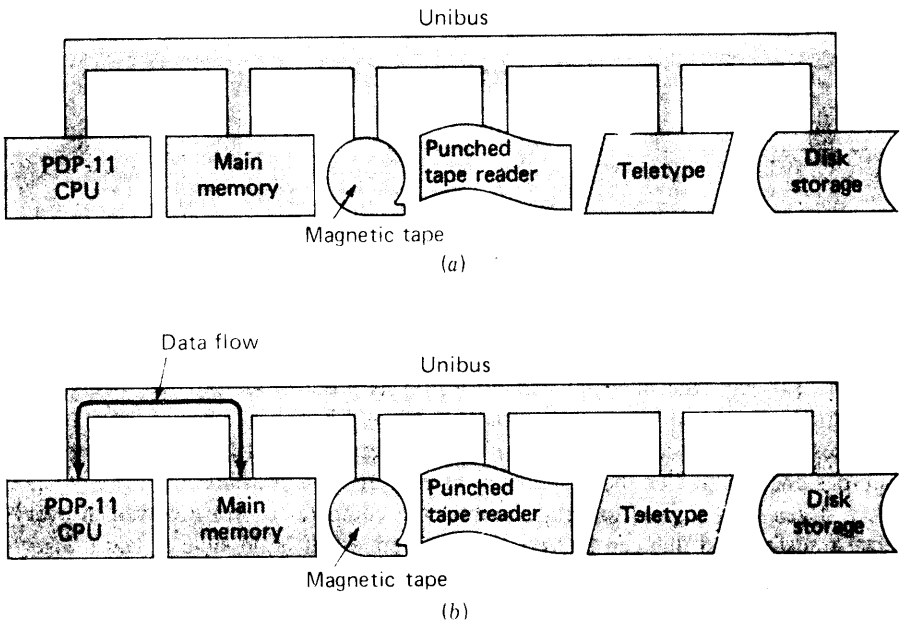


FIGURE 8.2

Bus for DEC PDP-11. (*a*) CPU, memory, and other devices are connected by a single bus. (*b*) CPU controls transfers of data in normal use. (*Digital Equipment Corp.*)

wires, or connections, and in the bus are provisions for addressing the components and transferring data from or to each component.

Figure 8.2 shows the organization for the DEC PDP-11 bus, which DEC calls a *unibus*. Notice that the same wires are used to transfer data from the CPU to the high-speed main memory as from the CPU to a tape punch or other input-output device.

In the simplest systems, the CPU is the director of all traffic on the bus. If a transfer of data must be made from, for instance, a disk pack to the core memory, then the CPU, under program control, will read each piece of data into its CPU general registers and then store each piece of data in the core memory.

There is a problem here in the computer's ability to know when a *peripheral device*[2] has performed a given operation. Suppose we wish to find some data on a magnetic tape and are unwilling to wait for the tape to be searched, desiring to perform other calculations while waiting. If the computer must continually look to see whether the tape drive now has the data available, then time is lost and programming complexity is increased. To alleviate this, the computer bus is generally provided with control lines[3] which are called *interrupt lines*, and a peripheral device can raise one of these lines when it has completed an action and is ready for attention.

The computer must be provided with some kind of interrupt facility so that

[2]*Peripheral devices* are the input-output devices, disk packs, tape drives, and other devices not including the main (IC) memory.

[3]The lines are normally 0; to raise a line means to place a 1 on it. In microcomputers and minicomputers there may be only one line. The word *lines* is often used for the conductors (wires) on a bus. Physically the lines are electric conductors.

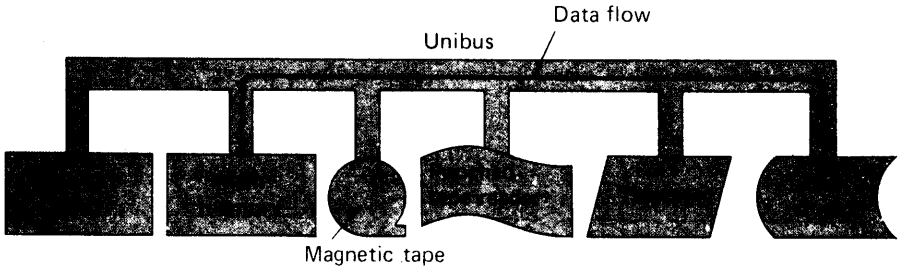


FIGURE 8.3

DEC bus in DMA mode. (*Digital Equipment Corp.*)

it can "service" the interrupt without losing its place in the program being executed. This problem becomes serious in systems where a number of input-output devices (such as A-to-D converters) must be serviced frequently, and computers are designed to service these interrupts as efficiently as possible.

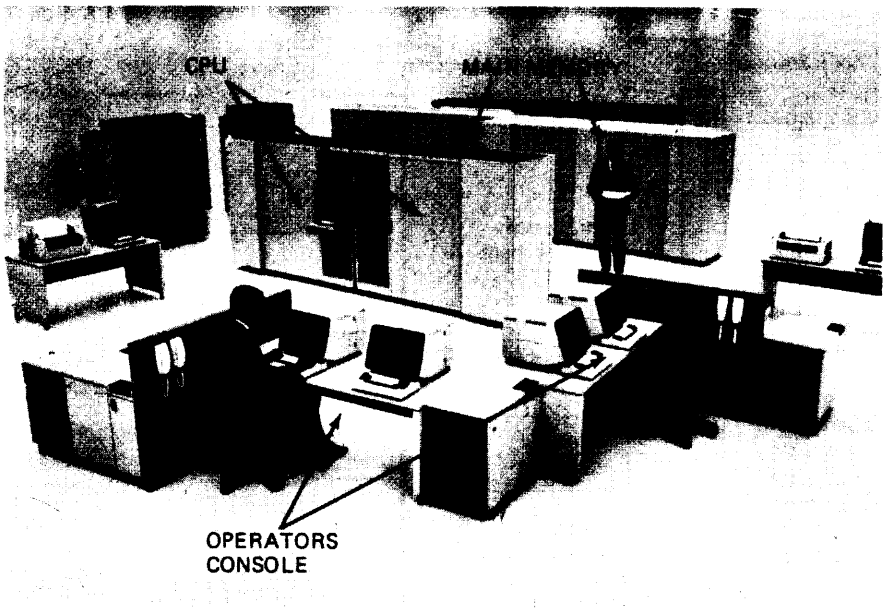
Even with a good interrupt facility, the computer is still involved in every data transfer, and this can be very time-consuming. It is possible to add a *direct memory access* (DMA) feature to most systems where a disk pack or tape reader transfers data directly into high-speed main memory using the bus but without passing the data through the CPU (see Fig. 8.3). This is done by "stealing" memory cycles, called *cycle stealing*. The CPU is simply held in its present state for one or more memory cycles while data are transferred from the disk pack directly into the main memory. The CPU does not "see" each transfer when it occurs, but simply continues executing its program, which is slowed down a little because of the cycle stealing, but not nearly as much as if the CPU had to make each transfer itself. The CPU must, of course, originate these DMA transfers by telling where in the disk pack the data are to be read from and into which locations in the main memory the data are to be read. (Transfers generally can be made in either direction when the DMA feature is added to a system, that is, for example, tape to main memory or main memory to tape.)

Because of the economy of operation, a bus is now the most used way to interconnect components of a microcomputer or minicomputer system.

Large systems have quite different problems from very small systems, and so different interconnection configurations are used. Since large systems contain many components, they are quite expensive, and it is important to utilize the CPU and other components to the maximum. Thus the cost of more expensive interconnection configurations is warranted.

As a result, in order to keep large processor configurations such as that shown in Fig. 8.4 working at their maximum speeds, the systems are operated in *multiprogramming mode*. This means that several programs are kept in memory at the same time. A given program is executed until it demands an input-output device or perhaps a disk drive. Since these devices are slow compared to the CPU, the device is started in its function, but then the CPU begins executing another different program until this program asks for input-output. When this happens, the CPU begins executing a third program, and this process continues.

When a program completes execution, another program is read in. As can be seen, the CPU must keep track of where it is in each program and must control



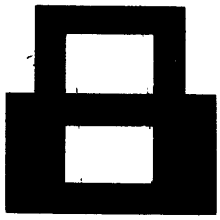


FIGURE 8.4

Parts of a large general-purpose computer. (*IBM Corp.*)

all the data transfers between system components, but would be hopelessly held up if it had to participate in each transfer.

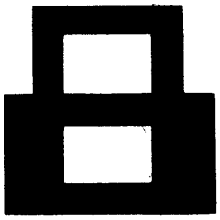
One way to configure a large system of this sort is shown in Fig. 8.5, which illustrates the IBM 3081 configuration. It shows a single CPU and two input-output processors, one of which IBM calls a *multiplexer channel* and the other a *selector channel*.[4] Some systems have more input-output processors, and others even have more than one CPU, in which case the system is called a *multiprocessor*.

The system operates as follows: All transfers to and from peripheral devices (such as card readers, printers, tape drives, etc.) are initiated by the CPU telling an input-output processor what is to be done. The actual transfers are then made by the special-purpose processors, which work independently of the CPU. To initiate a data transfer, the CPU tells the input-output processor where in memory to put (or find) the data, which input-output device to use, and (if necessary) where in that device the data are located. The actual transfer of data is guided by a *channel program* executed by the input-output processor which has been written in advance, and the CPU also sees that the correct channel program is used.

Once an input-output data transfer has been initiated by the CPU, the CPU can go about executing other programs. When the input-output processor completes its work, the CPU is notified so that it can return and continue where it was in the program which called for the input-output transfer.

Another large computer configuration is that in Fig. 8.6, which shows the

[4]A multiplexer channel has logic particularly suited to relatively slow devices with random characteristics. Selector channels are for fast bursts of data generated by disk packs and tape drives, for instance.



BUSES AND INTERFACES

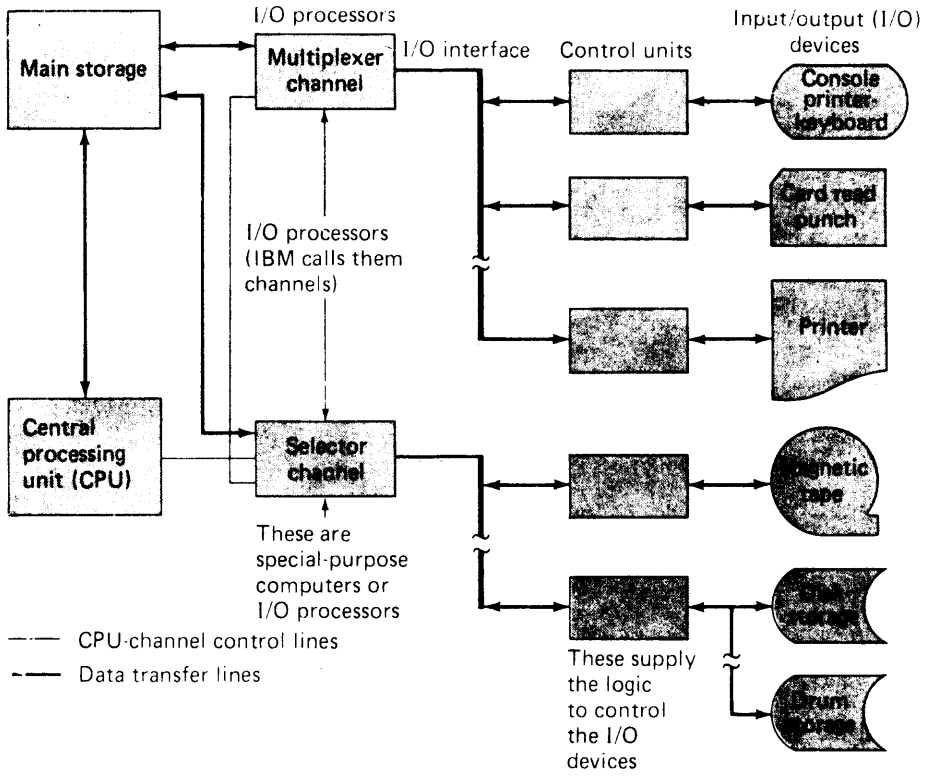


FIGURE 8.5

Organization of IBM 3081 computer series.

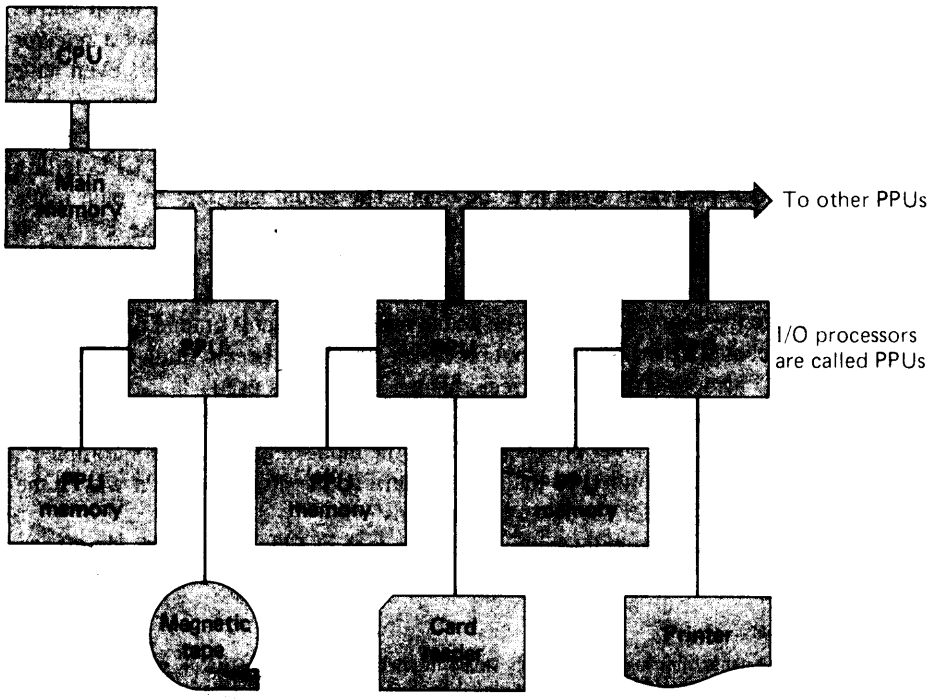
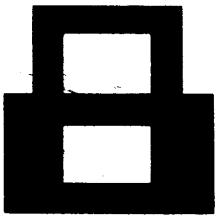


FIGURE 8.6

Cyber computer organization.



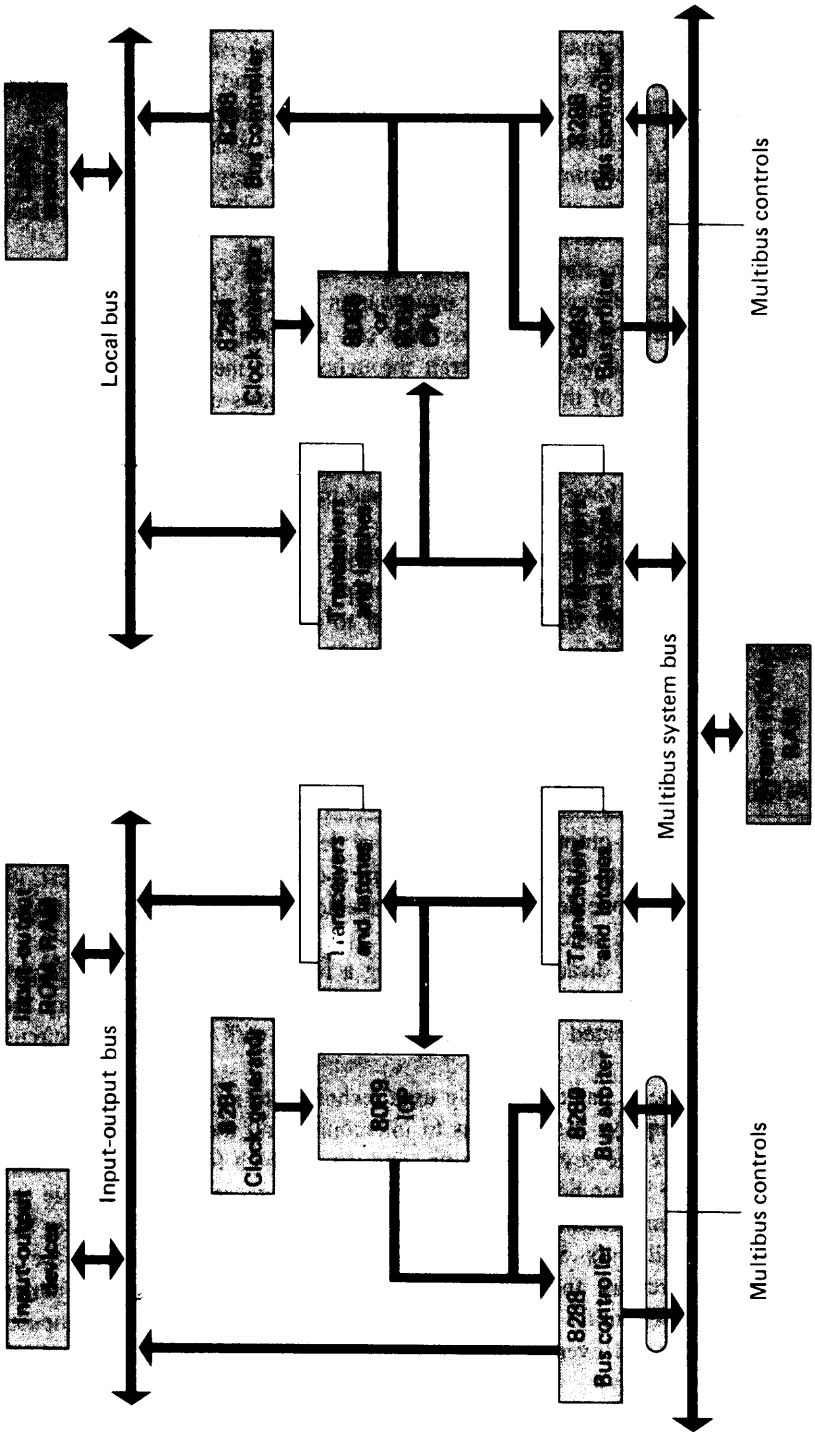
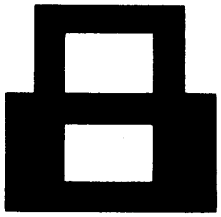


FIGURE 8.7

Multiprocessor configuration using 8086 and 8088 microprocessors.



organization of the CDC Cyber computer. In this case the input-output operations are all handled by small "computers" called *peripheral processing units* (PPUs). The PPUs have their own memories and programs and work independently of the Cyber CPU. The way the Cyber commands input-output operations is to "plant" messages in a specified area in its memory, telling what it would like. The PPUs then search this memory, looking for orders; when they find one, they execute the necessary operations and plant a message telling the CPU that its orders have been fulfilled and the necessary operations performed.

Complex structures such as the IBM 3081 and Cyber make good sense because a number of peripheral or input-output devices can be operating simultaneously at the relatively low speeds they can maintain, while the CPU races from job to job. In this way the overall throughput for the computer system can be increased because of the parallel operation of all parts of the system.

The idea of providing several CPUs which execute programs in parallel is an attractive one. As just mentioned, such systems are called *multiprocessor systems*. Again, they have high throughput and make good usage of both large memories and the several input-output devices.

Figure 8.7 shows how the 8086, 8088, and 8089 microprocessor chips can be combined with a number of support chips to produce a multiprocessor system. More 8086 or 8088 CPUs can be added to the multiprocessor as desired, each using the same configuration.

INTERFACING—BUSES

8.2 When input-output devices, memory devices, the arithmetic-logic unit, and the control unit are all combined to form a computer system, all these must be connected. When one device or unit is connected to another, an interface is required which includes the necessary logic.

The primary disadvantages of using a large number of individual cables to interconnect parts of a system are cost and complexity. The necessary interface logic must be repeated for each connection along with cable-driving circuits and receiving circuits.

As has been stated, a widely used technique to interface modules efficiently at low cost employs a single bus to interconnect all the units. This is shown in Fig. 8.8, where the several lines or conductors which form the bus pass through and connect to a number of units, or modules. In general, each module can read from or write onto the bus. The bus interface is usually standardized since the same bus connects all units. Since each unit connects only once to the bus, the amount of interface circuitry and logic required tends to be lower than for separate connections between units. As a result, buses are widely used in microcomputers and minicomputers and even in large computer systems for modules where the data flow is not excessive.

Often the modules which are bused together share the same data lines. Then it is necessary for each module to be able to both write onto and read from a given line.

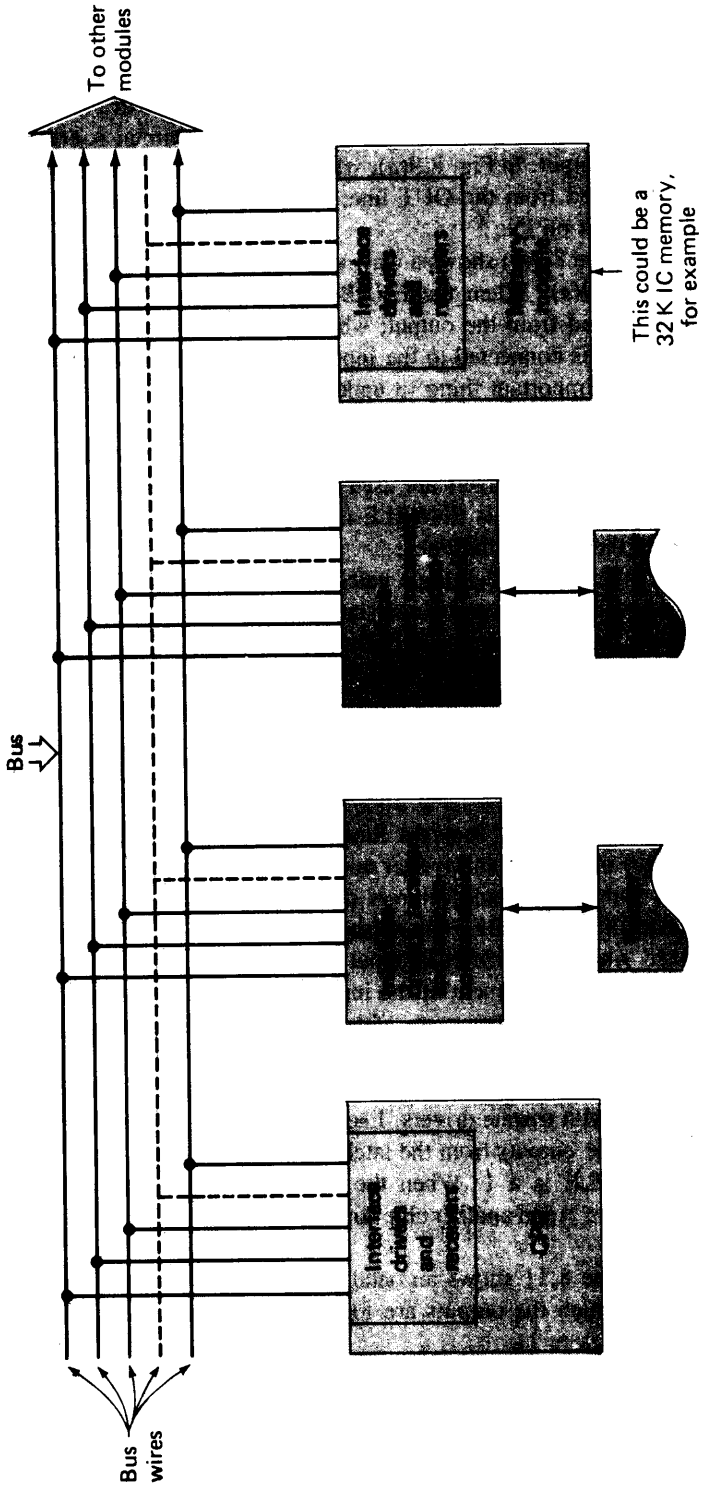
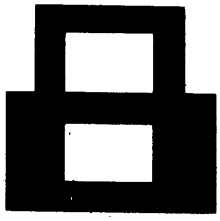


FIGURE 8.8

Computer organization of single-bus system.



To share lines on a bus,[5] a logic circuit called a *three-state* or *tristate driver* is used. The block diagram for the three-state driver is shown in Fig. 8.9(*a*). There are IN and ENABLE inputs and a single OUT line. A table showing the operation of the three-state driver is also given. The three-state driver has three output states: a 0 output, a 1 output, and a state in which the circuit is effectively disconnected from the output. In Fig. 8.9(*a*), where the ENABLE is a 0, the circuit is effectively disconnected from the OUT line; where ENABLE is a 1, the output is the same as the input on IN.

Figure 8.9(*c*) shows a way of looking at the operation of the three-state driver in Fig. 8.9(*a*). When the ENABLE is a 0, the relay is open and so the IN is disconnected from the output; when the ENABLE is a 1, the relay is closed and the output is connected to the input, and so the logic values are the same.

The important thing to understand is that if several three-state drivers have their outputs connected to the same line and if only one of the drivers has as its ENABLE input a 1, then that particular driver will control the state of the line. When three-state drivers are used for interfaces, only one interface on a line (conductor) must have its ENABLE input a 1 at a given time, and this interface will control the state of the line.

Inverters or other logic gates can have their inputs connected to a line which is controlled by several three-state drivers. Generally, manufacturers arrange so that the three-state driver will drive a line of considerable length and with several gate inputs connected to it, which is often the case for a bus line.

Figure 8.9(*b*) shows a three-state driver that inverts its input. The table shows this: When the ENABLE is a 0, the driver is effectively disconnected from the output; and when the ENABLE is a 1, the OUT value is the complement of the IN value.

Figure 8.9(*d*) shows a functional representation of (*b*) showing the driver operates much like an inverter and a relay connected.

Sometimes manufacturers make three-state drivers with a DISABLE input instead of an ENABLE input [see Fig. 8.9(*e*)]. In this case the driver is disconnected when the DISABLE input is a 1 and the output, which is inverted in this case, is the complement of the input when the DISABLE is a 0.

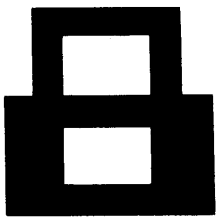
Three-state drivers are widely used in the interfaces for buses. They enable control of bus lines to pass from interface to interface as is appropriate.

Figure 8.10 shows a tristate octal *D*-type latch IC package with eight latches equipped with tristate drivers. Each latch reads the input at *D* when the clock input is high. The outputs from the latches are forced on the outputs from the chip when the ENABLE is a 1. When the ENABLE is a 0, the outputs represent "high impedances" (and another chip can drive the lines as they are connected to a desired state).

Figure 8.11 shows an octal tristate buffer with positive-edge-triggered flip-flops in which the outputs are forced to the input state only when the ENABLE input is high (a 1).

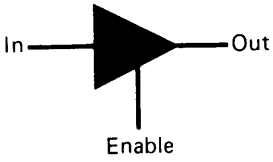
Since several units are sharing the same bus lines, the interface procedures for bused modules must be carefully worked out so that, for instance, two modules

[5]Three-state drivers are used in many applications other than on buses, but the sharing of a line is always the reason for their use.



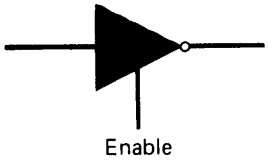
In	Enable	Out
0	0	–
1	0	–
0	1	0
1	1	1

– indicates driver is electrically disconnected



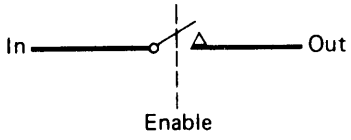
(*a*)

In	Enable	Out
0	0	–
1	0	–
0	1	1
1	1	0

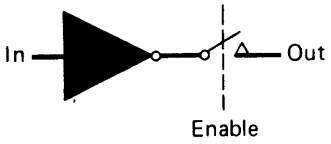


(*b*)

In	Enable	Out
0	0	Relay open
1	0	Relay open
0	1	0
1	1	1

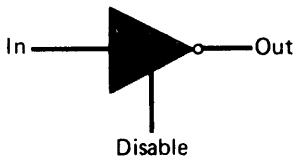


(*c*)



(*d*)

In	Disable	Out
0	0	1
1	0	0
0	1	–
1	1	–



(*e*)

FIGURE 8.9

Three-state driver operation. (*a*) Three-state driver. (*b*) Three-state driver with inverted output. (*c*) Functional representation of (*a*). (*d*) Functional representation of (*b*). (*e*) Three-state driver with disable input.

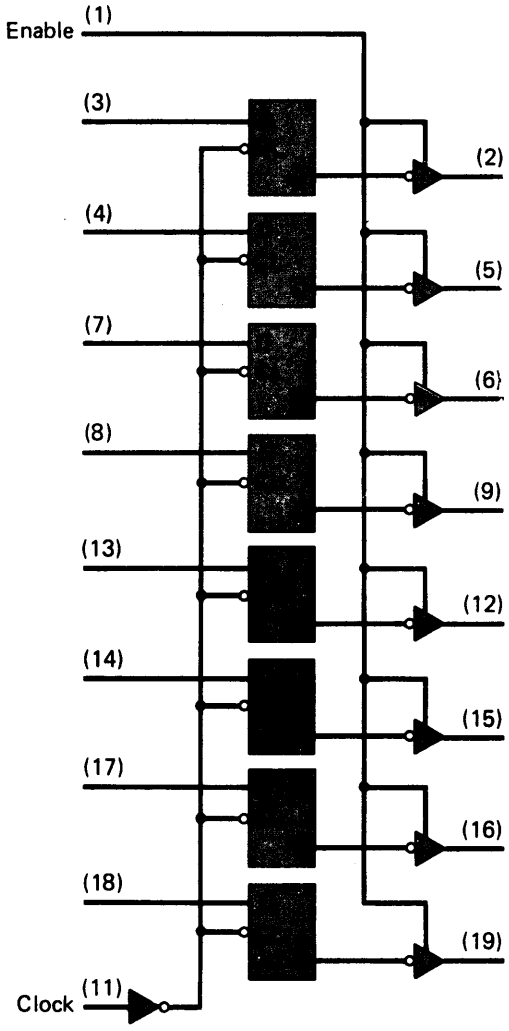


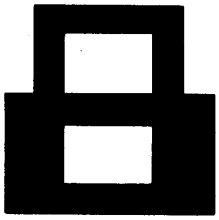
FIGURE 8.10

Octal latches with tristate drivers.

do not attempt to write data on the bus at the same time and the module for which the data are intended knows it is the selected module, etc.

BUS FORMATS AND OPERATION

8.3 A number of different types and several different standards for buses exist. All buses can be divided into three major sections, however: the *address*, *data*, and *control* sections. This is shown in Fig. 8.12 as are two commonly used ways to represent multiple lines on a bus. Figure 8.12(*a*) shows the three sections, using a wide, ribbonlike representation for the multiple lines (wires). Figure 8.12(*b*) shows that sometimes a single printed line is used with a starting slash through the printed line to indicate that there are actually a number of lines (wires). For both



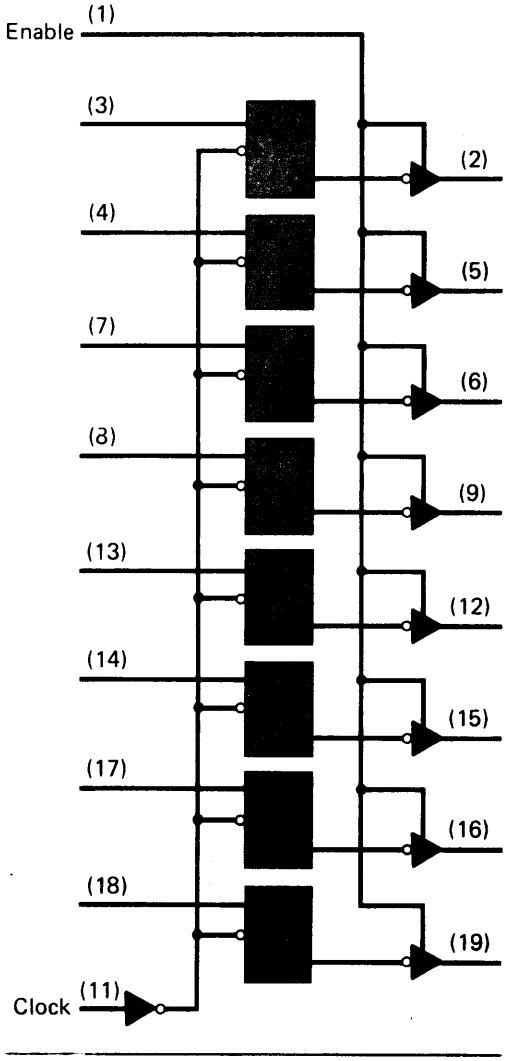
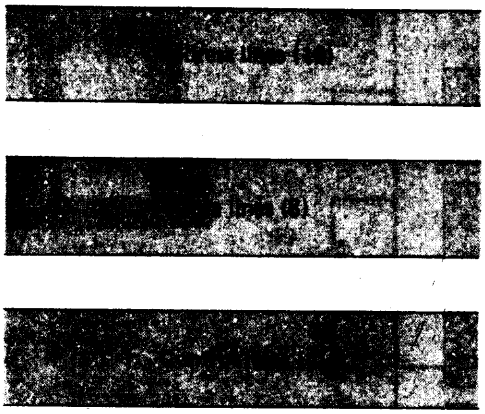


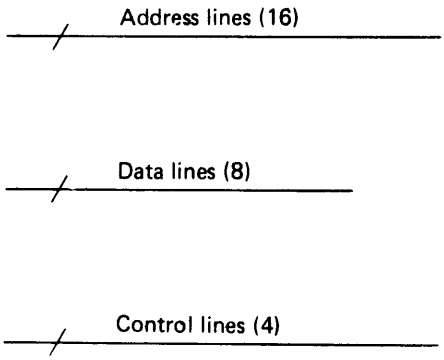
FIGURE 8.11

Octal flip-flop chip with tristate drivers.

Fig. 8.12(*a*) and (*b*) there are 16 address lines, 8 data lines, and 4 control lines. Both representations in (*a*) and (*b*) are frequently used.

Most buses now use tristate drivers to write data on the data lines. In the most straightforward systems, the address lines are completely controlled by the *bus master* which is generally a miniprocessor or microprocessor CPU. If there is a single bus master, the remaining devices connected to the bus are called *slaves*. Each slave then has an address number, and the bus master uses the address lines to control who is to use the bus. In some systems, however, devices other than the CPU can take control of the bus. In this case, a controlling device is called the bus master only when it has control, and at that time the responding devices are called slaves. For buses in which control is shared among several devices (that is, more than one device can take control of the bus), the address lines also are driven by tristate drivers.





(a) (b)

FIGURE 8.12

Address, data, and control sections of a bus. (*a*) The three sections of a bus. (*b*) Alternative representation for (*a*).

Some of the control lines may be permanently controlled by the CPU, and others may be used by several devices and thus require tristate drivers (or wired OR or wired AND drivers).

Note that any device can read from a line driven by a tristate driver and conventional inverters or gates can be connected to these lines. (The only restriction is that the gates connected to the lines not load the lines unduly.)

Buses transfer information over data wires by using either a *synchronous* or an *asynchronous* technique. Different manufacturers and bus designers have different philosophies about which is better, with the result that there is no standard.

For synchronous transfers, the bus works as follows. Let us assume a CPU wishes to read and write from peripheral devices. Each device is given a separate number, and a device is selected by the CPU placing that number on the address lines.[6]

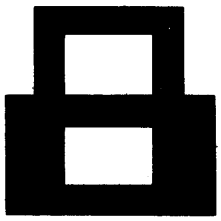
Figure 8.13 shows the timing of synchronous data transfers involving a set of address wires, data wires, and a READ and WRITE control line. All transfers are controlled by the CPU. Note that the convention is that the meaning of READ and WRITE is always relative to the master or CPU on buses. Therefore, a READ means the CPU (master) reads data from the bus, and a WRITE means the CPU (master) writes data on the bus.

Figure 8.13 shows that in order to read, the CPU places the number of the device to be read from on the address lines. Then the CPU lowers[7] the $\overline{\text{READ}}$ line, and the selected device must place data on the DATA lines, which means enabling its tristate drivers connected to the bus DATA wires. When the $\overline{\text{READ}}$ line goes back to a 1, the device must disable its tristate drivers.

When the CPU desires to write data (to give data to a device), the device's number is placed on the address line and then the data are placed on the DATA lines. The CPU lowers the $\overline{\text{WRITE}}$ line to 0, and the device must read the data at

[6]A disk drive might be given the number 1, a keyboard the number 3, etc.

[7]It is common practice to use a 0 signal to activate devices on a bus. This is indicated by the bar over the name of the line ($\overline{\text{READ}}$). If a device is activated by a 1 signal, no bar is used.



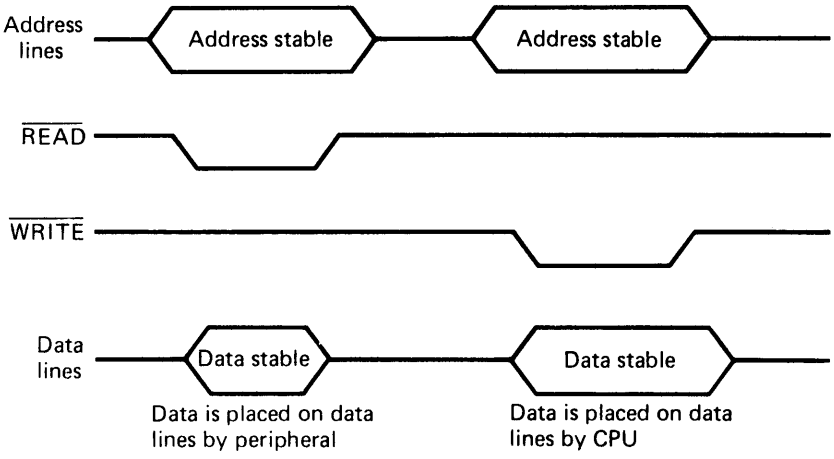


FIGURE 8.13

Timing signals for synchronous transfers.

that time. The data must have been read when the WRITE line goes back to a 1. The interface designer must know how long the WRITE will be a 0 so that the data can be read safely during the time the WRITE line is a 0.

Note here that the device being written into or read from must respond during the fixed time period permanently established by the CPU. If a READ is performed, the device must place data on the data lines at once and keep them there while the READ line is down. Similarly, for a WRITE the device must have already read the data by the time that the WRITE line goes high.

Synchronous transfers are generally thought to be the fastest way to transfer data and are used for memory data transfers and sometimes for transferring data to other types of devices. The problem is that all devices must be able to respond at the same speed unless the CPU has READ and WRITE signals of different durations for different devices.[8] To alleviate this problem (that is, to accommodate devices with differing response times), the asynchronous transfer technique is often used.

When an asynchronous bus transfer technique is used, another control line is required, called DATA VALID OR RECEIVED (see Fig. 8.14). This line is controlled by the devices and not the CPU, however. And if there is more than one device on the bus, a tristate driver will be required for each device using this line.[9]

The sequence of timing steps performed in Fig. 8.14 is as follows. To read from the bus, the CPU sets the number of the device on the ADDRESS lines and then lowers the READ line. The device which is selected then places data on the DATA lines and a 1 on the DATA VALID OR RECEIVED line. This tells the CPU that the data are on the lines and can be read. The CPU cannot read from the DATA lines until the DATA VALID OR RECEIVED line is a 1. If the device is slow in preparing its data, the CPU must wait until the data are on the DATA lines

[8]This would be a complicated strategy. Another bad alternative is for all devices to operate at the speed of the slowest devices.

[9]A driver which wire-ORs its output could also be used.

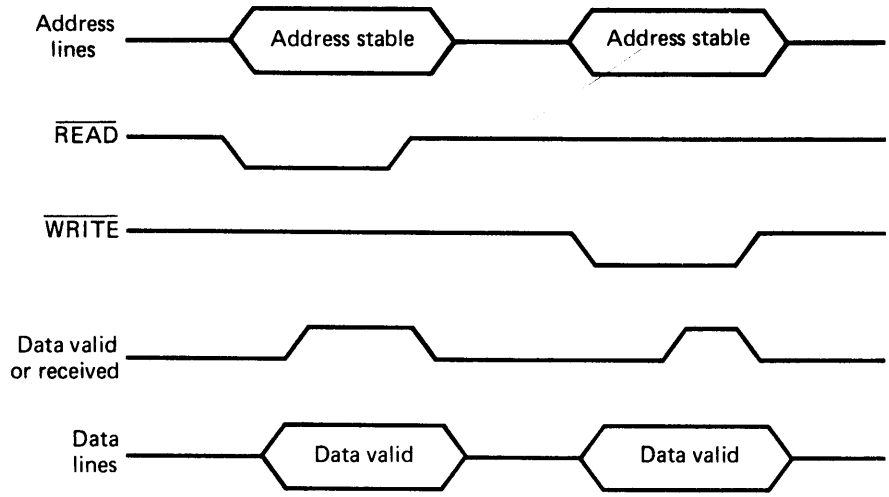


FIGURE 8.14

Timing signals for asynchronous transfers.

and the DATA VALID OR RECEIVED line is raised. Next the CPU reads the data and raises the $\overline{\text{READ}}$ line to a 1. This means the selected device must keep its data on the DATA line until $\overline{\text{READ}}$ goes to a 1. The selected device then turns off its tristate drivers to the DATA lines and lowers the DATA VALID OR RECEIVED line.

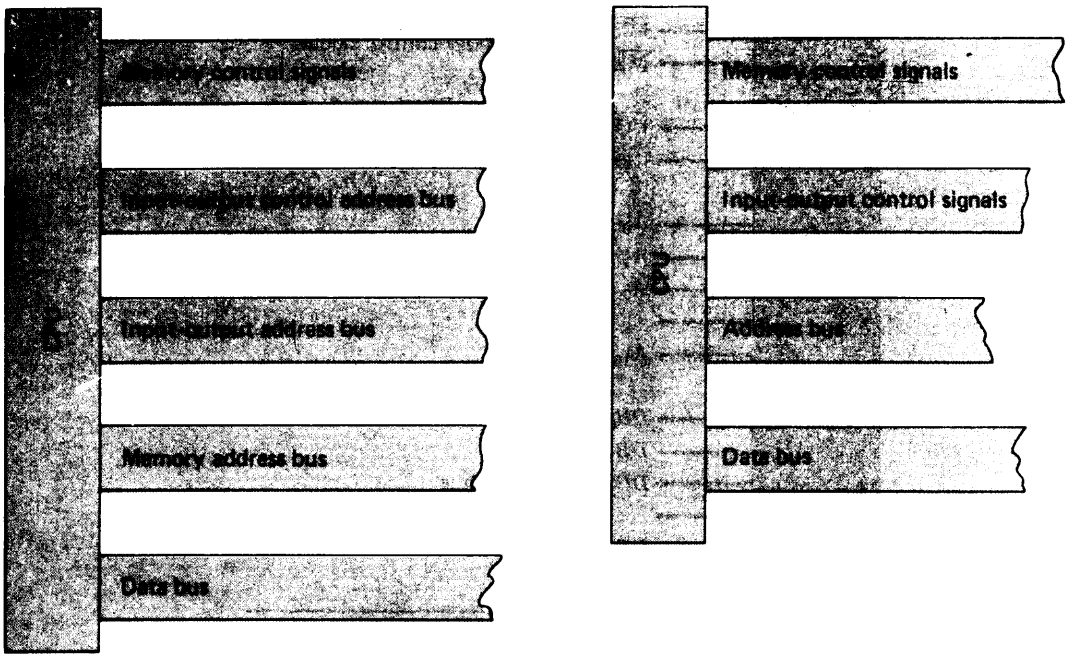
A write operation is performed similarly. First the CPU places the number of the selected device on the ADDRESS line and the data on the DATA lines. Then the CPU lowers the $\overline{\text{WRITE}}$ line. The selected device now reads the data and raises the DATA VALID OR RECEIVED line. The CPU must keep its DATA and ADDRESS lines stable until it receives this signal, so the device can be "slow to read" and still get the data. After the CPU receives the DATA VALID OR RECEIVED high signal, it removes the address and data from the bus and then raises the $\overline{\text{WRITE}}$ line; this allows the selected device to lower the DATA VALID OR RECEIVED lines.

The asynchronous procedure involves what is called a *handshake*. The effect is that the CPU tells which is selected if it is reading or writing and then waits for the selected device to respond (with a handshake) before continuing. Thus means fast and slow devices can be accommodated on the same bus. This is the reason for the wide use of asynchronous buses.

For both synchronous and asynchronous buses, there are several variations on Figs. 8.13 and 8.14. Often, instead of separate READ and WRITE lines, a single R/$\overline{\text{W}}$ line is used (1 for read, 0 for write) and a separate ADDRESS VALID line is used to indicate when the address is on the address lines. Sometimes the DATA VALID and DATA RECEIVED lines are separate. The principles of synchronous and asynchronous data transfers remain, and the combining or splitting of control lines can be figured out easily if the general principles are understood.

ISOLATED AND MEMORY-MAPPED INPUT-OUTPUT

8.4 Two general techniques for identifying input-output (I/O) devices have been the most used for bus operation. The first separates the I/O devices from the



(a) (b)

FIGURE 8.15

Bus structure for input-output and memory. (*a*) Separate memory and input-output address lines. (*b*) Shared memory and input-output address bus.

memory and addresses (or numbers), for I/O devices are separate from memory addresses; this is often called *isolated I/O*. The second, called *memory-mapped I/O*, mixes memory addresses with I/O device numbers.

In the isolated I/O technique, the I/O devices are each given a separate number on the bus. For instance, a printer would be assigned device number 3, a keyboard device number 4, a disk drive device number 5, etc. These numbers are placed in binary on the address section of the bus when an I/O device is to be read from or written into by the CPU. The I/O devices may even have a separate I/O address bus, as shown in Fig. 8.15(*a*), or the I/O devices may share the address section of the bus with memory, as in Fig. 8.15(*b*). In either case, the I/O device number is placed on the bus by the CPU, and then the control lines tell the I/O device whether to place data on the lines, to read data, etc. If the I/O devices share bus lines with the main (IC) memory, the control signals must tell whether an address on the bus is for I/O devices or for memory.

The 8080 microprocessor is an example of an I/O system which has device numbers on the address lines that are shared with memory, as in Fig. 8.15(*b*). Figure 8.16 shows the sections of the 8080 bus. To read from a memory device, the 8080 places the memory address to be read from on A_0 to A_{15} and then lowers $\overline{\text{MEMR}}$; next the memory places the contents at this address on data wires DB_0 to DB_7. To write into memory, the 8080 places the address to be written into on A_0 to A_{15} and the data to be written on DB_0 to DB_7 and lowers $\overline{\text{MEMW}}$; the data are then written at the selected location.

The 8080 uses only eight of the address lines A_0 to A_{15} to select I/O devices. These lines are A_0 to A_7. As a result, only 256 I/O devices can be used. Suppose

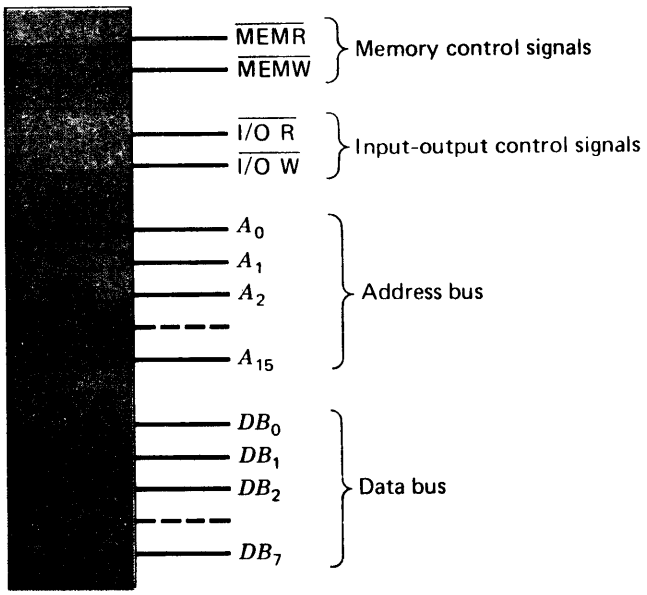


FIGURE 8.16

8080 memory and intput-output bus sections.

a keyboard has device number 3_{10}. Then to read from the keyboard, the CPU[10] places 00000011 on A_7 to A_0 and then lowers $\overline{\text{I/O R}}$; next the keyboard places an 8-bit character (data) on DB_0 to DB_7.

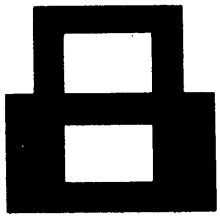
As another example, if a printer has device number 5 and the CPU wishes to print a character, the CPU places 00000101 on A_7 to A_0, the character on DB_0 to DB_7 and then lowers $\overline{\text{I/O W}}$. This bus is synchronous, and so the devices must read or write data at the specified time because the $\overline{\text{I/O W}}$ and $\overline{\text{I/O R}}$ lines are lowered only for a time determined by the clock rate of the 8080.

The 8080 is only the original member of a line of 8-bit microprocessors produced by Intel. Other processors have numbers such as 8085, 8089, etc. Each has particular features. The bus interfaces for each are compatible; however, the speed of transfer is determined by the clock rate of the particular chip which varies from device line to device line. (A memory or I/O device which responds in 100 ns could interface any of the current devices.)

The second general technique for addressing I/O devices is called *memory-mapped I/O*. When this is used, the I/O devices are assigned addresses in memory. These locations must not conflict with addresses given to memory devices such as ROM and RAM chips. Then the I/O devices are read from and written into by using the same control lines as for the IC memory chips.

The memory-mapped I/O technique requires making a map of memory showing which locations are devoted to IC memory and which to I/O devices. Figure 8.17 shows a memory map for a bus with 16 address bits and 8 data bits. The computer has a 4K-word read-only memory, a 4K-word random-access memory, a printer, and a keyboard. The ROM is connected to addresses 0 to 4095 and the

[10]The most significant bit is A_7; the least significant bit is A_0. For numbers on DB_7 to DB_0, the sign bit goes on DB_7 and the least signficant bit is DB_0.



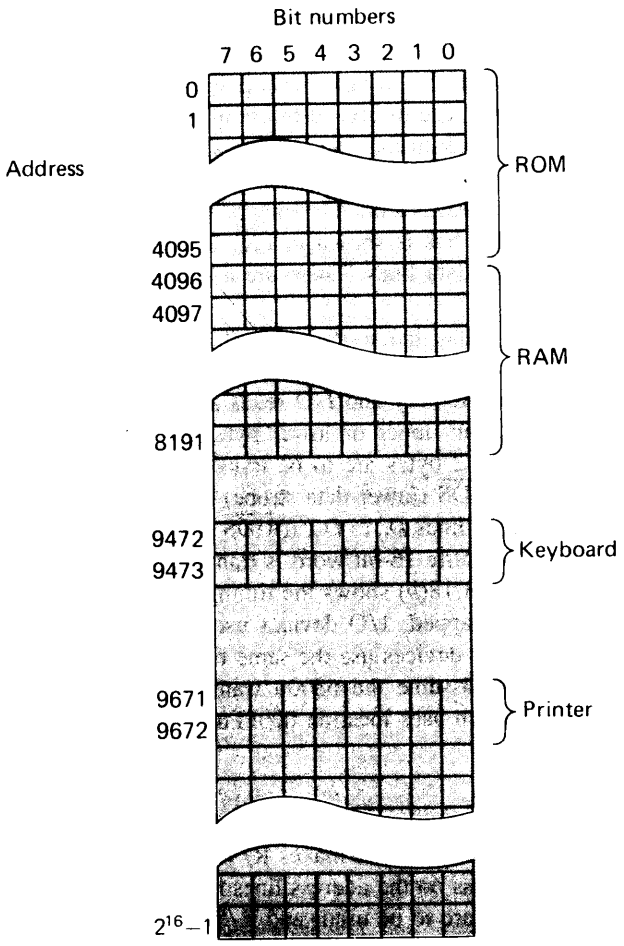


FIGURE 8.17

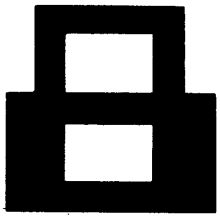
Map of memory layout for memory-mapped I/O.

RAM to addresses 4096 to 8191; the keyboard is given locations 9472 and 9473, and the printer is given locations 9671 and 9672. (The keyboard and printer are each given two locations because each requires a status register, we discuss later.)

The programmer for this system must know where the RAM, ROM, keyboard, and printer are located. To print a character, the particular address allocated to the printer must be used; to read a character from the keyboard, the address given to the keyboard must be employed.

Memory-mapped computers have no I/O instructions in their list of instructions. To read a character from a keyboard, the keyboard address is simply used; any instruction which reads from memory can read from that address. Data from a keyboard would be added to or ANDed with the current contents of an accumulator by using a single instruction, for example; or a character from the keyboard could be simply transferred into an accumulator. The PDP-11 series and 6800 and 68000 microprocessors are examples of memory-mapped computers.

In systems having separate I/O control lines and device numbers, the CPU will have specific I/O instructions. For example, the 8080 has IN and OUT in-



structions, and these specifically cause transfers to and from I/O devices. The programmer must then know the I/O device numbers and the memory addresses.

Advocates of separate I/O point out that interfaces may be simpler and that programmers' use of I/O instructions seems more natural. Advocates of memory-mapped I/O claim that the CPU is simpler as is the bus and that the instructions in the CPU for data manipulation can be used for I/O data, simplifying programs.

The 68000 is a memory-mapped and asynchronous microcomputer. A section of the 68000 bus is shown in Fig. 8.18(*a*). This is a large bus with 23 address lines and 16 data lines. There are a number of control signals, five of which are shown.

The 68000 has a 16-bit word, which is the normal unit for data transfer. However, each word is divided into two 8-bit bytes, called the *upper byte* and *lower byte*. Memory and I/O reads and writes can transfer either a complete 16-bit word or an upper or lower byte. To indicate to an I/O device or memory whether 1 or 2 bytes are to be transferred, two control signals $\overline{\text{UDS}}$ (upper data strobe) and $\overline{\text{LDS}}$ (lower data strobe) are used [Fig. 8.18(*a*)]. If $\overline{\text{LDS}}$ is low, data are placed on lines D_0 to D_7; if $\overline{\text{UDS}}$ is low, data lines D_8 to D_{15} are used. If both are low, an entire 16-bit word is transferred.

Figure 8.18(*b*) shows the timing for both a read and write. Since the 68000 is memory-mapped, I/O devices use addresses in memory and data transfers to and from I/O devices use the same timing signals as memory transfers.

Let us assume the 68000 wants to read a 16-bit word from a disk drive interface which uses location $000FF6_{16}$. The timing would be as follows [refer to Fig. 8.18(*b*)]:

1 The 68000 places the address 000FF6 on the address lines and then lowers $\overline{\text{AS}}$, $\overline{\text{UDS}}$, and $\overline{\text{LDS}}$ and makes R/$\overline{\text{W}}$ a 1. A 0 on $\overline{\text{AS}}$ tells the devices on the bus that the address on the address lines is valid, both $\overline{\text{UDS}}$ and $\overline{\text{LDS}}$ low indicates all 16 data lines are to be used, and R/$\overline{\text{W}}$ indicates a read operation.

2 The disk drive interface places 16 bits of data on D_0 to D_{15} and then lowers $\overline{\text{DTACK}}$ (for data acknowledge), which indicates the data are on lines D_0 to D_{15}.

3 The 68000 reads the data from D_0 to D_{15}.

4 The 68000 raises $\overline{\text{AS}}$ to indicate the disk interface can release the data because they have been read.

5 The disk interface turns off its tristate drivers to lines D_0 to D_{15} and raises $\overline{\text{DTACK}}$.

A write is performed similarly. Assume a tape drive interface accepts data on address $00076F4_{16}$. When data are written into the tape drive interface, the following occurs:

1 The 68000 places 00076F4 on address lines A_1 to A_{23}; places the data to be transferred on D_0 to D_{15}; and then lowers $\overline{\text{AS}}$, $\overline{\text{UDS}}$, $\overline{\text{LDS}}$, and R/$\overline{\text{W}}$. This indicates that the address is valid, all of D_0 to D_{15} is to be used, and the operation is a write operation.

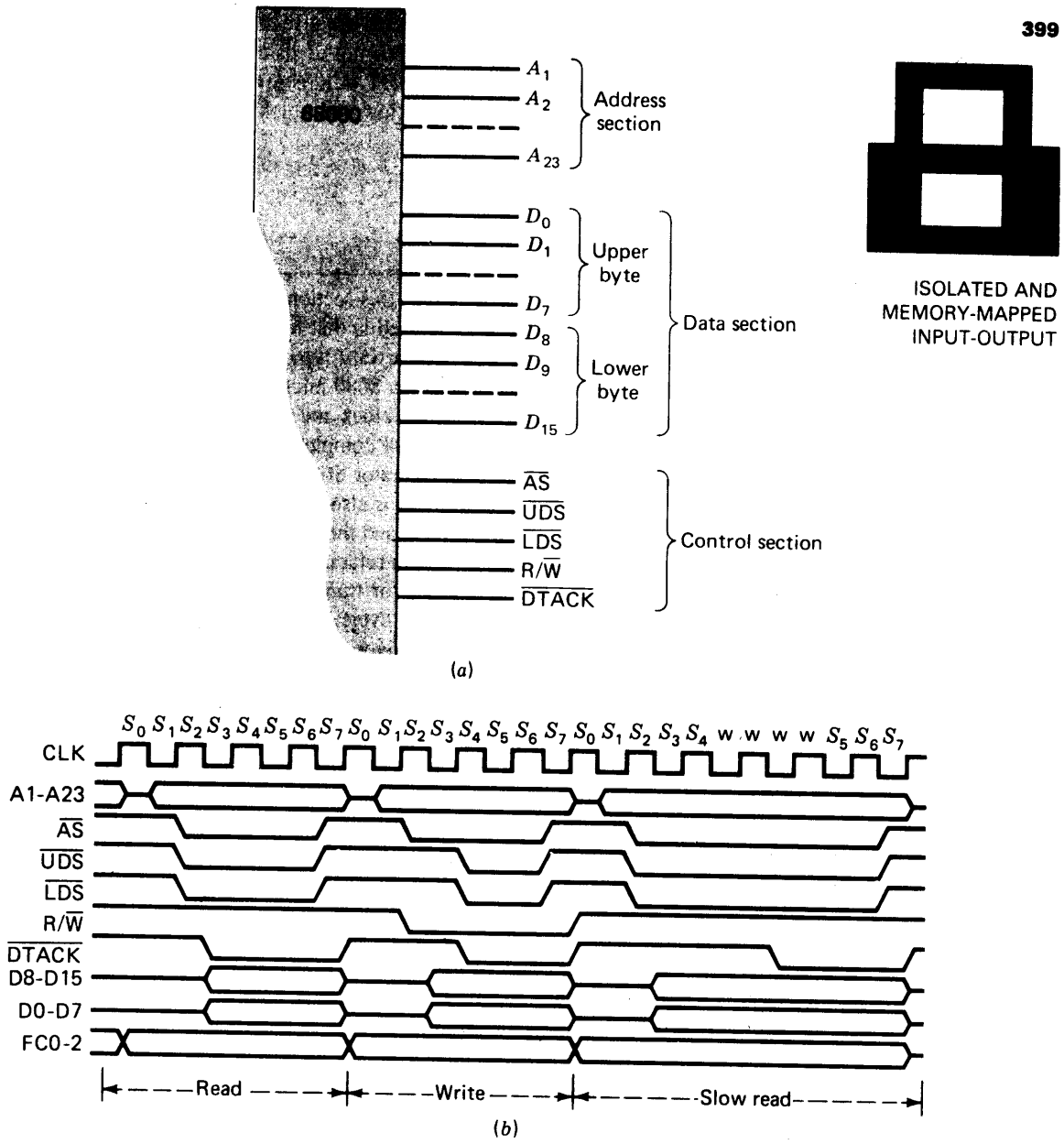


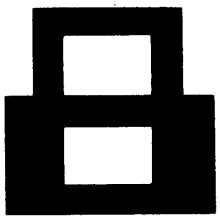
FIGURE 8.18

68000 bus signal. (*a*) Bus layout. (*b*) Read-write timing.

2 The tape drive interface reads the data from D_0 to D_{15}.

3 After it has the data, the tape drive interface lowers $\overline{\text{DTACK}}$, to indicate the data transfer is complete.

4 The 68000 releases its address from the address bus and the data from the data bus and then raises $\overline{\text{AS}}$, R/$\overline{\text{W}}$, $\overline{\text{UDS}}$, and $\overline{\text{LDS}}$.



Note that these transfers are asynchronous because the I/O device or memory must signal the acceptance of data or the placing of data on D_0 to D_{15} by using $\overline{\text{DTACK}}$. The 68000 will not proceed until $\overline{\text{DTACK}}$ is lowered. This is shown by "slow read" shown where $\overline{\text{DTACK}}$ is not lowered quickly, resulting in a longer read time.

INTERFACING A KEYBOARD

8.5 Section 7.12 described keyboards. In this section we describe the interfacing of a keyboard with a bus. The bus to be used is that for the 8080 microprocessor. The interface developed will be a straightforward typical design.

Figure 8.19 shows the basic bus for an 8080 microprocessor. The 8080 CPU chip requires a separate chip to generate the clock and several other timing signals. It will not affect the interface design, but for completeness we show a chip developed for this purpose, the 8224 clock generator driver.

An 8228 bidirectional bus driver chip is also used in this design.[11] The 8080 output lines have limited drive capabilities, and the 8228 bus driver has TTL levels and drive capabilities which are useful for interfacing. Also, and more importantly, the 8080 bus uses its data lines D_0 to D_7 for transmitting several control signals (status bits) during an early section of each cycle. These status bits are considered a part of the bus for the 8080. The 8228 driver strobes these values into flip-flops and then outputs them as $\overline{\text{INTA}}$, $\overline{\text{MEMR}}$, I/O R, $\overline{\text{I/O W}}$, etc., which are then considered to be a part of the 8080 system bus.

Notice that the 8080 bus has three basic classes of input-output lines: address lines A_{15} to A_0, data lines D_7 to D_0, and control lines such as $\overline{\text{WR}}$, DBIN, and $\overline{\text{I/O R}}$.

The address signals are used both to address the IC memory and to select which input device is to be written into or read from. The data lines are bidirectional; that is, data are written into the 8080 CPU chip by using D_7 to D_0. These same lines are also used to output data to memories, input-output devices, etc. Bidirectional lines are widely used in buses for computers, the main advantage being fewer connections to and from chips and fewer pins on chips. If the data wires D_7 to D_0 were not bidirectional, a set of both eight input wires and eight output wires would be required instead of the eight bidirectional wires.[12]

Using bidirectional data lines means that the various system components such as memories and keyboards must be carefully controlled and timed in their operations so that only one device writes on a wire at a time and so that system components know exactly when to examine wires with signals on them.

Each input and output device which interfaces an 8080 system is given a unique *device number*. The numbers given devices can have up to 8 bits. Thus 256 different devices can be handled directly.

---

[11]This is a chip developed by Intel to facilitate interfacing with input-output devices. Often microprocessor chips have limited power output lines and require extra chips for interfacing.

[12]Three-state drivers are normally used to drive these lines.