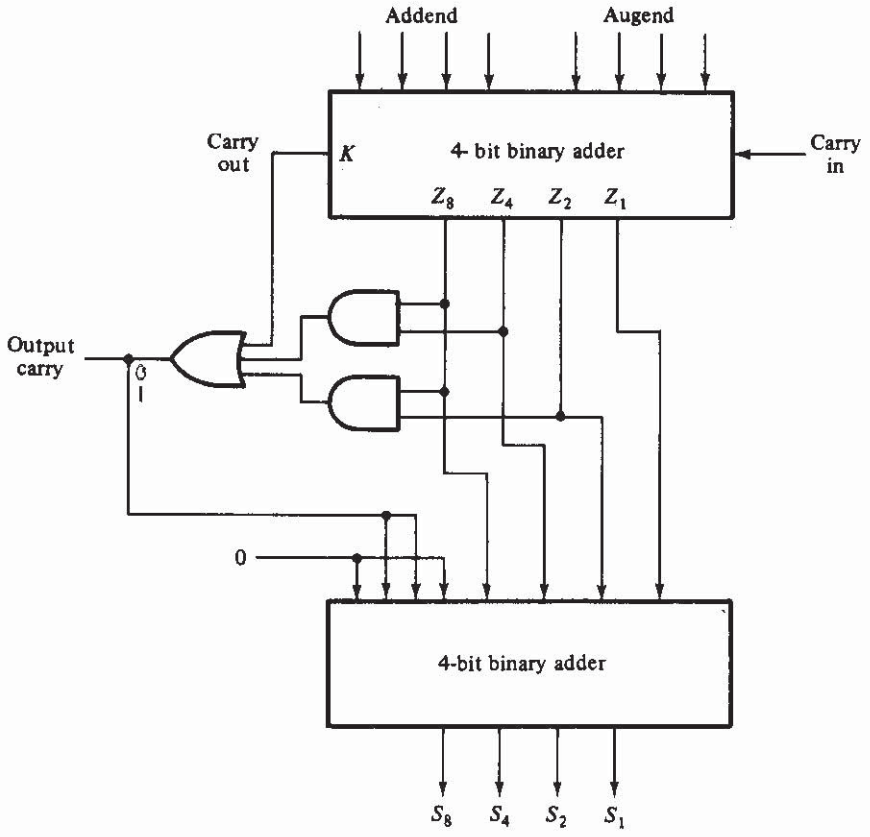


FIGURE 5-6
Block diagram of a BCD adder

output carry from one stage must be connected to the input carry of the next higher-order stage.

5-4 MAGNITUDE COMPARATOR

The comparison of two numbers is an operation that determines if one number is greater than, less than, or equal to the other number. A *magnitude comparator* is a combinational circuit that compares two numbers, A and B, and determines their relative magnitudes. The outcome of the comparison is specified by three binary variables that indicate whether $A > B$, $A = B$, or $A < B$.

The circuit for comparing two n-bit numbers has 2^{2n} entries in the truth table and becomes too cumbersome even with $n = 3$. On the other hand, as one may suspect, a comparator circuit possesses a certain amount of regularity. Digital functions that possess an inherent well-defined regularity can usually be designed by means of an al-

gorithmic procedure if one is found to exist. An *algorithm* is a procedure that specifies a finite set of steps that, if followed, give the solution to a problem. We illustrate this method here by deriving an algorithm for the design of a 4-bit magnitude comparator.

The algorithm is a direct application of the procedure a person uses to compare the relative magnitudes of two numbers. Consider two numbers, A and B, with four digits each. Write the coefficients of the numbers with descending significance as follows:

$$A = A_3A_2A_1A_0$$

$$B = B_3B_2B_1B_0$$

where each subscripted letter represents one of the digits in the number. The two numbers are equal if all pairs of significant digits are equal, i.e., if $A_3 = B_3$ and $A_2 = B_2$ and $A_1 = B_1$ and $A_0 = B_0$. When the numbers are binary, the digits are either 1 or 0 and the equality relation of each pair of bits can be expressed logically with an equivalence function:

$$x_i = A_iB_i + A_i'B_i' \qquad i = 0, 1, 2, 3$$

where $x_i = 1$ only if the pair of bits in position i are equal, i.e., if both are 1's or both are 0's.

The equality of the two numbers, A and B, is displayed in a combinational circuit by an output binary variable that we designate by the symbol $(A = B)$. This binary variable is equal to 1 if the input numbers, A and B, are equal, and it is equal to 0 otherwise. For the equality condition to exist, all x_i variables must be equal to 1. This dictates an AND operation of all variables:

$$(A = B) = x_3x_2x_1x_0$$

The *binary* variable $(A = B)$ is equal to 1 only if all pairs of digits of the two numbers are equal.

To determine if A is greater than or less than B, we inspect the relative magnitudes of pairs of significant digits starting from the most significant position. If the two digits are equal, we compare the next lower significant pair of digits. This comparison continues until a pair of unequal digits is reached. If the corresponding digit of A is 1 and that of B is 0, we conclude that $A > B$. If the corresponding digit of A is 0 and that of B is 1, we have that $A < B$. The sequential comparison can be expressed logically by the following two Boolean functions:

$$(A > B) = A_3B_3' + x_3A_2B_2' + x_3x_2A_1B_1' + x_3x_2x_1A_0B_0'$$

$$(A < B) = A_3'B_3 + x_3A_2'B_2 + x_3x_2A_1'B_1 + x_3x_2x_1A_0'B_0$$

The symbols $(A > B)$ and $(A < B)$ are *binary* output variables that are equal to 1 when $A > B$ or $A < B$, respectively.

The gate implementation of the three output variables just derived is simpler than it seems because it involves a certain amount of repetition. The "unequal" outputs can use the same gates that are needed to generate the "equal" output. The logic diagram of the 4-bit magnitude comparator is shown in Fig. 5-7. The four x outputs are generated with

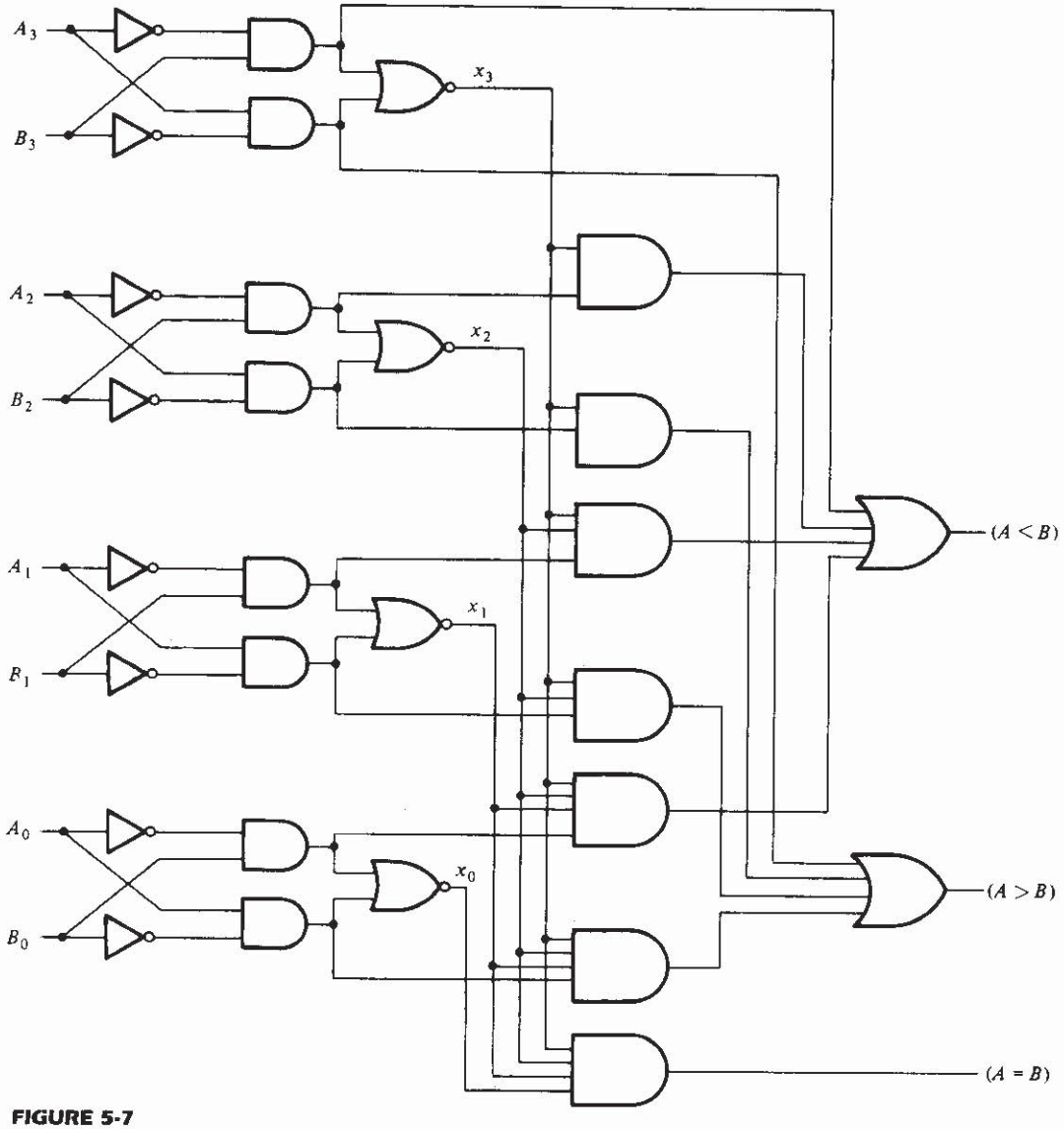


FIGURE 5-7
4-bit magnitude comparator

equivalence (exclusive-NOR) circuits and applied to an AND gate to give the output binary variable $(A = B)$. The other two outputs use the x variables to generate the Boolean functions listed before. This is a multilevel implementation and, as clearly seen, it has a regular pattern. The procedure for obtaining magnitude-comparator circuits for binary numbers with more than four bits should be obvious from this example. The same circuit can be used to compare the relative magnitudes of two BCD digits.

5-5 DECODERS AND ENCODERS

Discrete quantities of information are represented in digital systems with binary codes. A binary code of n bits is capable of representing up to 2^n distinct elements of the coded information. A *decoder* is a combinational circuit that converts binary information from n input lines to a maximum of 2^n unique output lines. If the n-bit decoded information has unused or don't-care combinations, the decoder output will have fewer than 2^n outputs.

The decoders presented here are called n-to-m-line decoders, where $m \leq 2^n$. Their purpose is to generate the 2^n (or fewer) minterms of n input variables. The name *decoder* is also used in conjunction with some code converters such as a BCD-to-seven-segment decoder.

As an example, consider the 3-to-8-line decoder circuit of Fig. 5-8. The three inputs are decoded into eight outputs, each output representing one of the minterms of the 3-input variables. The three inverters provide the complement of the inputs, and each one of the eight AND gates generates one of the minterms. A particular application of this decoder would be a binary-to-octal conversion. The input variables may represent a binary number, and the outputs will then represent the eight digits in the octal number

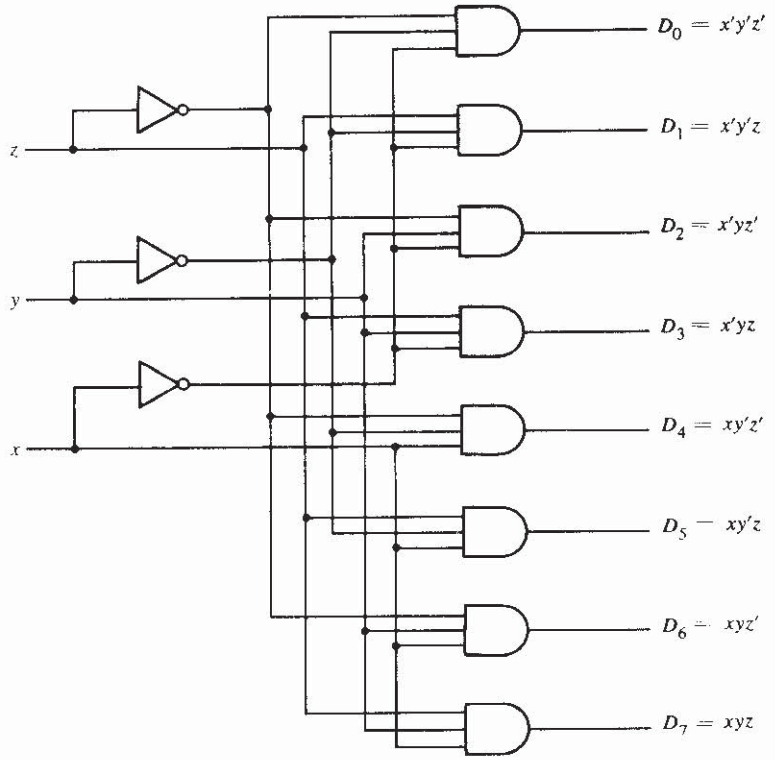


FIGURE 5-8
A 3-to-8 line decoder

TABLE 5-2
Truth Table of a 3-to-8-Line Decoder

Inputs			Outputs							
x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

system. However, a 3-to-8-line decoder can be used for decoding any 3-bit code to provide eight outputs, one for each element of the code.

The operation of the decoder may be further clarified from its input–output relationship, listed in Table 5-2. Observe that the output variables are mutually exclusive because only one output can be equal to 1 at any one time. The output line whose value is equal to 1 represents the minterm equivalent of the binary number presently available in the input lines.

Combinational Logic Implementation

A decoder provides the 2^n minterm of n input variables. Since any Boolean function can be expressed in sum of minterms canonical form, one can use a decoder to generate the minterms and an external OR gate to form the sum. In this way, any combinational circuit with n inputs and m outputs can be implemented with an n-to-2^n-line decoder and m OR gates.

The procedure for implementing a combinational circuit by means of a decoder and OR gates requires that the Boolean functions for the circuit be expressed in sum of minterms. This form can be easily obtained from the truth table or by expanding the functions to their sum of minterms (see Section 2-5). A decoder is then chosen that generates all the minterms of the n input variables. The inputs to each OR gate are selected from the decoder outputs according to the minterm list in each function.

Example 5-1

Implement a full-adder circuit with a decoder and two OR gates.

From the truth table of the full-adder (Section 4-3), we obtain the functions for this combinational circuit in sum of minterms:

$$S(x, y, z) = \Sigma(1, 2, 4, 7)$$
$$C(x, y, z) = \Sigma(3, 5, 6, 7)$$

Since there are three inputs and a total of eight minterms, we need a 3-to-8-line decoder. The implementation is shown in Fig. 5-9. The decoder generates the eight

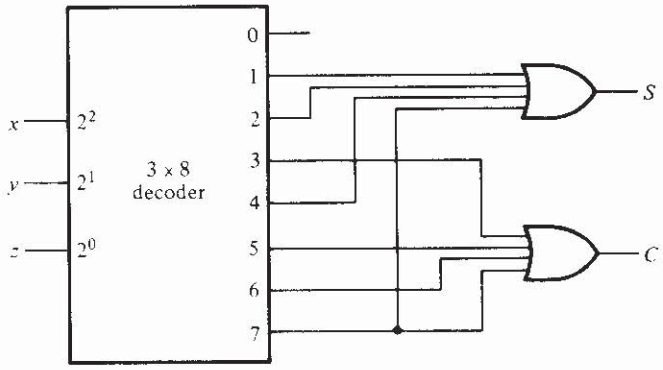


FIGURE 5-9
Implementation of a full-adder with a decoder

minterms for x, y, z. The OR gate for output S forms the sum of minterms 1, 2, 4, and 7. The OR gate for output C forms the sum of minterms 3, 5, 6, and 7. ■

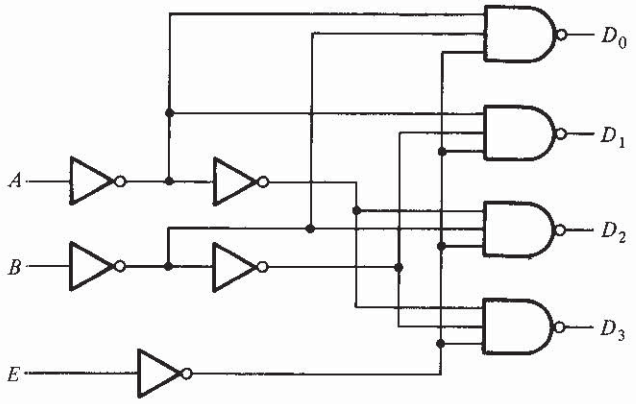
A function with a long list of minterms requires an OR gate with a large number of inputs. A function F having a list of k minterms can be expressed in its complemented form F' with $2^n - k$ minterms. If the number of minterms in a function is greater than $2^n/2$, then F' can be expressed with fewer minterms than required for F. In such a case, it is advantageous to use a NOR gate to sum the minterms of F'. The output of the NOR gate will generate the normal output F.

The decoder method can be used to implement any combinational circuit. However, its implementation must be compared with all other possible implementations to determine the best solution. In some cases, this method may provide the best implementation, especially if the combinational circuit has many outputs and if each output function (or its complement) is expressed with a small number of minterms.

Demultiplexers

Some IC decoders are constructed with NAND gates. Since a NAND gate produces the AND operation with an inverted output, it becomes more economical to generate the decoder minterms in their complemented form. Most, if not all, IC decoders include one or more *enable* inputs to control the circuit operation. A 2-to-4-line decoder with an enable input constructed with NAND gates is shown in Fig. 5-10. All outputs are equal to 1 if enable input E is 1, regardless of the values of inputs A and B. When the enable input is 0, the circuit operates as a decoder with complemented outputs. The truth table lists these conditions. The X's under A and B are don't-care conditions. Normal decoder operation occurs only with $E = 0$, and the outputs are selected when they are in the 0 state.

The block diagram of the decoder is shown in Fig. 5-11(a). The small circle at input E indicates that the decoder is enabled when $E = 0$. The small circles at the outputs indicate that all outputs are complemented.



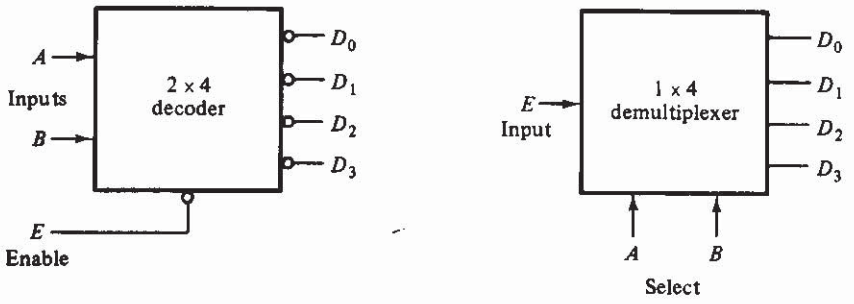
(a) Logic diagram

E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

(b) Truth table

FIGURE 5-10
A 2-to-4-line decoder with enable (E) input

A decoder with an enable input can function as a demultiplexer. A *demultiplexer* is a circuit that receives information on a single line and transmits this information on one of 2^n possible output lines. The selection of a specific output line is controlled by the bit values of n selection lines. The decoder of Fig. 5-10 can function as a demultiplexer if the E line is taken as a data input line and lines A and B are taken as the selection lines. This is shown in Fig. 5-11(b). The single input variable E has a path to all four outputs, but the input information is directed to only one of the output lines, as specified by the binary value of the two selection lines, A and B. This can be verified from the truth table of this circuit, shown in Fig. 5-10(b). For example, if the selection lines $AB = 10$, output D_2 will be the same as the input value E, while all other outputs are maintained at 1. Because decoder and demultiplexer operations are obtained from the same circuit, a decoder with an enable input is referred to as a *decoder/demultiplexer*. It is the enable input that makes the circuit a demultiplexer; the decoder itself can use AND, NAND, or NOR gates.



(a) Decoder with enable

(b) Demultiplexer

FIGURE 5-11
Block diagrams for the circuit of Fig. 5-10

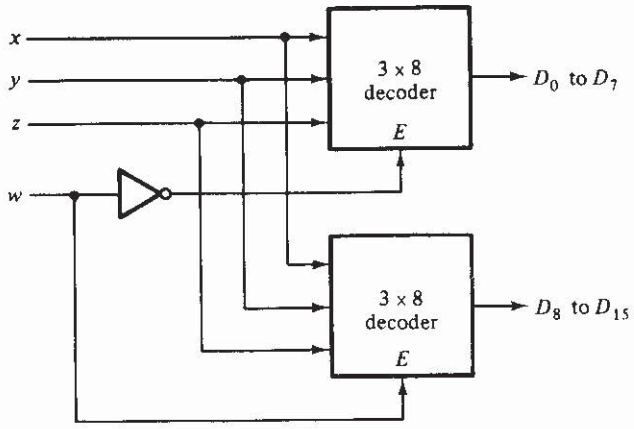


FIGURE 5-12
A 4 × 16 decoder constructed with two 3 × 8 decoders

Decoder/demultiplexer circuits can be connected together to form a larger decoder circuit. Figure 5-12 shows two 3 × 8 decoders with enable inputs connected to form a 4 × 16 decoder. When $w = 0$, the top decoder is enabled and the other is disabled. The bottom decoder outputs are all 0's, and the top eight outputs generate minterms 0000 to 0111. When $w = 1$, the enable conditions are reversed; the bottom decoder outputs generate minterms 1000 to 1111, while the outputs of the top decoder are all 0's. This example demonstrates the usefulness of enable inputs in ICs. In general, enable lines are a convenient feature for connecting two or more IC packages for the purpose of expanding the digital function into a similar function with more inputs and outputs.

Encoders

An encoder is a digital circuit that performs the inverse operation of a decoder. An encoder has 2^n (or fewer) input lines and n output lines. The output lines generate the binary code corresponding to the input value. An example of an encoder is the octal-to-binary encoder whose truth table is given in Table 5-3. It has eight inputs, one for each of the octal digits, and three outputs that generate the corresponding binary number. It is assumed that only one input has a value of 1 at any given time; otherwise the circuit has no meaning.

The encoder can be implemented with OR gates whose inputs are determined directly from the truth table. Output z is equal to 1 when the input octal digit is 1 or 3 or 5 or 7. Output y is 1 for octal digits 2, 3, 6, or 7, and output x is 1 for digits 4, 5, 6, or 7. These conditions can be expressed by the following output Boolean functions:

$$z = D_1 + D_3 + D_5 + D_7$$

$$y = D_2 + D_3 + D_6 + D_7$$

$$x = D_4 + D_5 + D_6 + D_7$$

TABLE 5-3
Truth Table of Octal-to-Binary Encoder

Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

The encoder is implemented with three OR gates, as shown in Fig. 5-13.

The encoder defined in Table 5-3 has the limitation that only one input can be active at any given time. If two inputs are active simultaneously, the output produces an undefined combination. For example, if D_3 and D_6 are 1 simultaneously, the output of the encoder will be 111 because all three outputs are equal to 1. This does not represent binary 3 nor binary 6. To resolve this ambiguity, encoder circuits must establish a priority to ensure that only one input is encoded. If we establish a higher priority for inputs with higher subscript numbers, and if both D_3 and D_6 are 1 at the same time, the output will be 110 because D_6 has higher priority than D_3.

Another ambiguity in the octal-to-binary encoder is that an output with all 0's is generated when all the inputs are 0. The problem is that an output with all 0's is also generated when D_0 is equal to 1. This ambiguity can be resolved by providing an additional output that specifies the condition that none of the inputs are active.

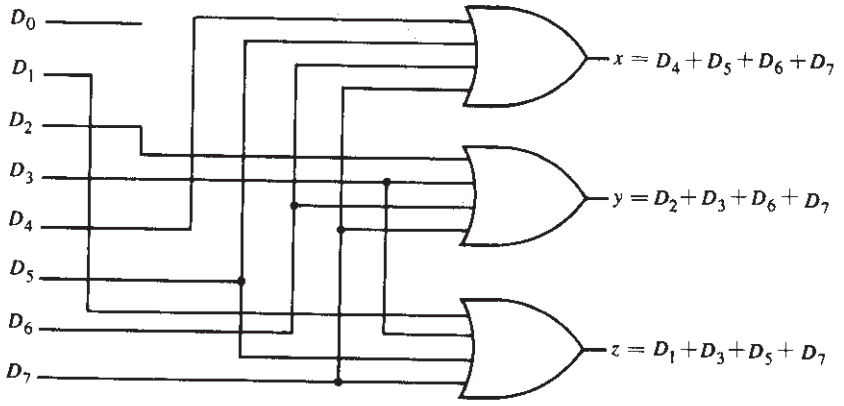


FIGURE 5-13
Octal-to-binary encoder

TABLE 5-4
Truth Table of a Priority Encoder

Inputs				Outputs		
D_0	D_1	D_2	D_3	x	y	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

Priority Encoder

A priority encoder is an encoder circuit that includes the priority function. The operation of the priority encoder is such that if two or more inputs are equal to 1 at the same time, the input having the highest priority will take precedence. The truth table of a four-input priority encoder is given in Table 5-4. The X's are don't-care conditions that designate the fact that the binary value may be equal either to 0 or 1. Input D_3 has the highest priority; so regardless of the values of the other inputs, when this input is 1, the output for xy is 11 (binary 3). D_2 has the next priority level. The output is 10 if $D_2 = 1$ provided that $D_3 = 0$, regardless of the values of the other two lower-priority inputs. The output for D_1 is generated only if higher-priority inputs are 0, and so on down the priority level. A *valid*-output indicator, designated by V, is set to 1 only when one or more of the inputs are equal to 1. If all inputs are 0, V is equal to 0, and the other two outputs of the circuit are not used.

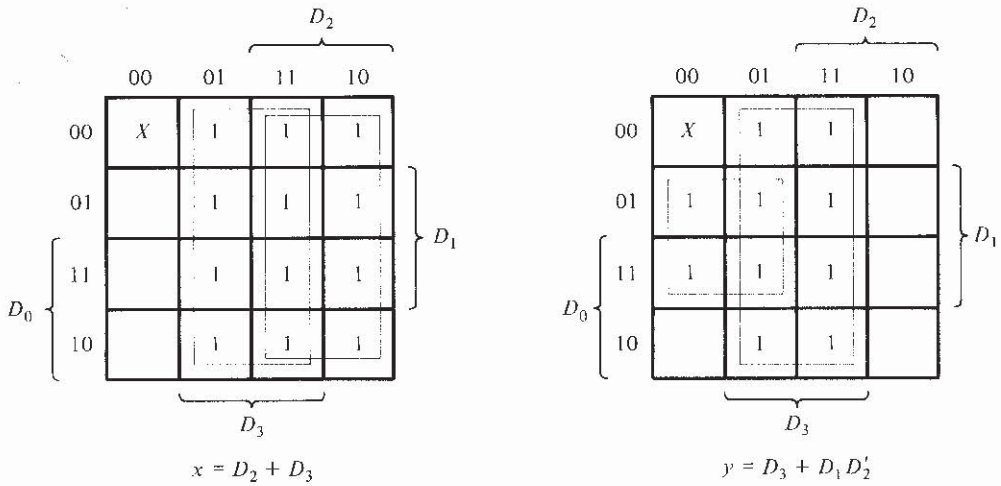


FIGURE 5-14
Maps for a priority encoder

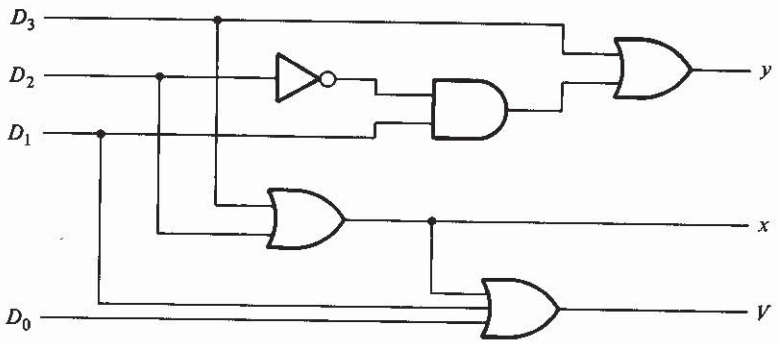


FIGURE 5-15
4-input priority encoder

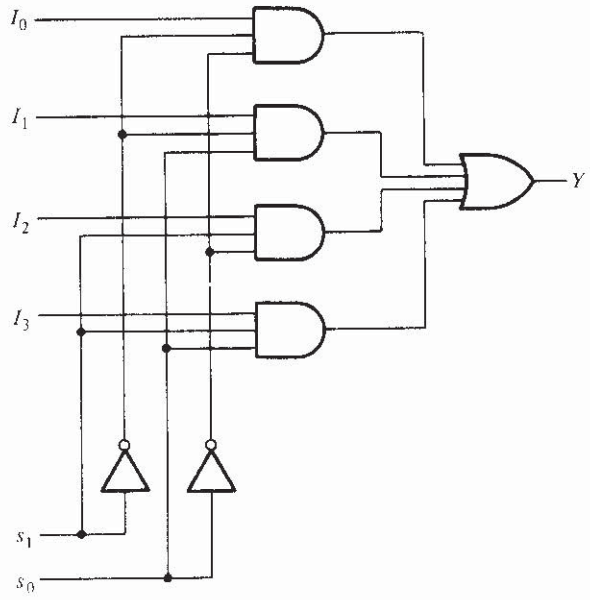
The maps for simplifying outputs x and y are shown in Fig. 5-14. The minterms for the two functions are derived from Table 5-4. Although the table has only five rows, when each don't-care condition is replaced first by 0 and then by 1, we obtain all 16 possible input combinations. For example, the third row in the table with $X100$ represents minterms 0100 and 1100 since X can be assigned either 0 or 1. The simplified Boolean expressions for the priority encoder are obtained from the maps. The condition for output V is an OR function of all the input variables. The priority encoder is implemented in Fig. 5-15 according to the following Boolean functions:

$$x = D_2 + D_3$$
$$y = D_3 + D_1 D_2'$$
$$V = D_0 + D_1 + D_2 + D_3$$

5-6 MULTIPLEXERS

Multiplexing means transmitting a large number of information units over a smaller number of channels or lines. A *digital multiplexer* is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line. The selection of a particular input line is controlled by a set of selection lines. Normally, there are 2^n input lines and n selection lines whose bit combinations determine which input is selected.

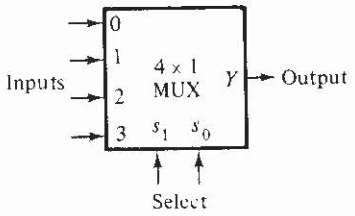
A 4-to-1-line multiplexer is shown in Fig. 5-16. Each of the four input lines, I_0 to I_3, is applied to one input of an AND gate. Selection lines s_1 and s_0 are decoded to select a particular AND gate. The function table, Fig. 5-16(b), lists the input-to-output path for each possible bit combination of the selection lines. When this MSI function is used in the design of a digital system, it is represented in block diagram form, as shown in Fig. 5-16(c). To demonstrate the circuit operation, consider the case when $s_1 s_0 = 10$. The AND gate associated with input I_2 has two of its inputs equal to 1 and the third input connected to I_2. The other three AND gates have at least one input equal to 0, which



(a) Logic diagram

S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

(b) Function table



(c) Block diagram

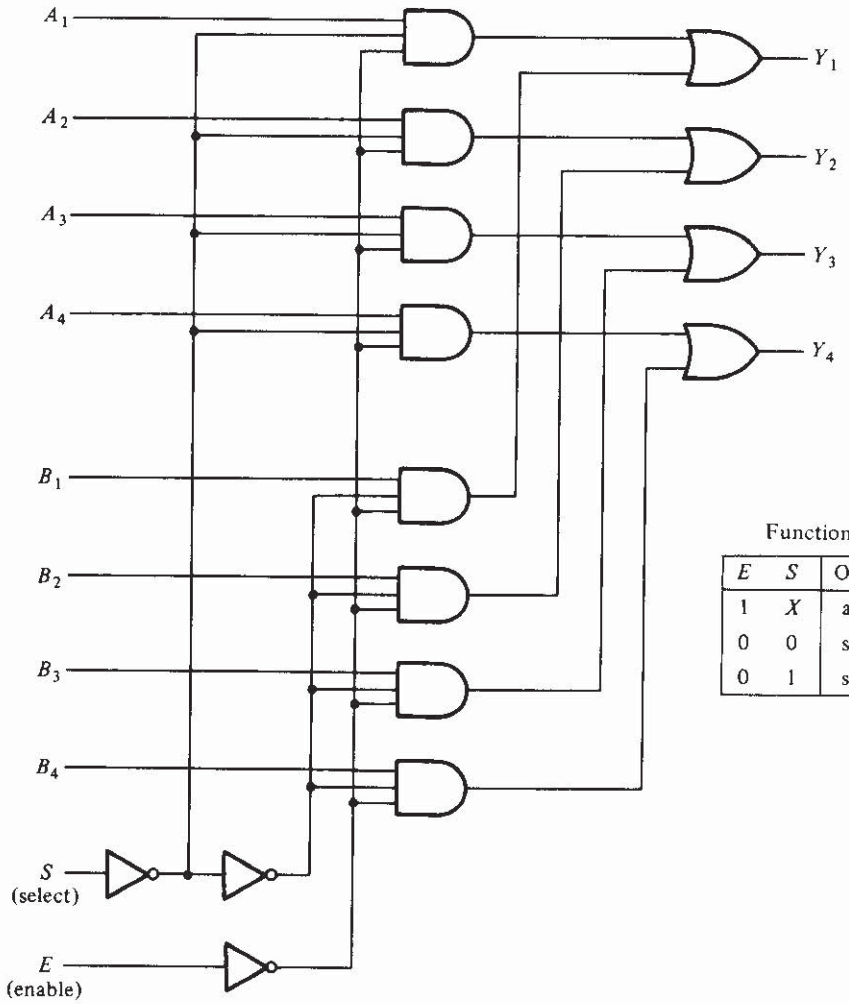
FIGURE 5-16
A 4-to-1-line multiplexer

makes their outputs equal to 0. The OR gate output is now equal to the value of I_2, thus providing a path from the selected input to the output. A multiplexer is also called a *data selector*, since it selects one of many inputs and steers the binary information to the output line.

The AND gates and inverters in the multiplexer resemble a decoder circuit and, indeed, they decode the input-selection lines. In general, a 2^n-to-1-line multiplexer is constructed from an n-to-2^n decoder by adding to it 2^n input lines, one to each AND gate. The outputs of the AND gates are applied to a single OR gate to provide the 1-line output. The size of a multiplexer is specified by the number 2^n of its input lines and the single output line. It is then implied that it also contains n selection lines. A multiplexer is often abbreviated as MUX.

As in decoders, multiplexer ICs may have an *enable* input to control the operation of the unit. When the enable input is in a given binary state, the outputs are disabled, and when it is in the other state (the enable state), the circuit functions as a normal multiplexer. The enable input (sometimes called *strobe*) can be used to expand two or more multiplexer ICs to a digital multiplexer with a larger number of inputs.

In some cases, two or more multiplexers are enclosed within one IC package. The selection and enable inputs in multiple-unit ICs may be common to all multiplexers. As an illustration, a quadruple 2-to-1-line multiplexer IC is shown in Fig. 5-17. It has four multiplexers, each capable of selecting one of two input lines. Output Y_1 can be selected



Function table

E	S	Output Y
1	X	all 0's
0	0	select A
0	1	select B

FIGURE 5-17
Quadruple 2-to-1-line multiplexer

to be equal to either A_1 or B_1. Similarly, output Y_2 may have the value of A_2 or B_2, and so on. One input selection line, S, suffices to select one of two lines in all four multiplexers. The control input E enables the multiplexers in the 0 state and disables them in the 1 state. Although the circuit contains four multiplexers, we may think of it as a circuit that selects one in a pair of 4-input lines. As shown in the function table, the unit is selected when $E = 0$. Then, if $S = 0$, the four A inputs have a path to the outputs. On the other hand, if $S = 1$, the four B inputs are selected. The outputs have all 0's when $E = 1$, regardless of the value of S.