CHAPTER 6

COMBINATIONAL LOGIC DESIGN USING MSI CIRCUITS

6.1 INTRODUCTION

The traditional methods of combinational circuit design have been discussed in Chapter 5. These methods involve simplification and realisation using gates. Using these methods, complex functions have been integrated (MSI) and are easily available in IC form. There is an attractive array of devices such as multiplexers, demultiplexers, adders, parity generators/checkers, priority encoders, decoders, comparators, etc. This chapter presents these complex integrated circuits and their applications in combinational system design. These devices significantly reduce IC package count thereby reducing the system cost. The system design is greatly simplified because the laborious and time consuming simplification methods are generally not required. This also improves the reliability of the system by reducing the number of external wired connections. Therefore, a designer must become familiar with the functions performed, the options available, and the limitations of these devices in order to make an effective and optimum use of such devices.

6.2 MULTIPLEXERS AND THEIR USE IN COMBINATIONAL LOGIC DESIGN

6.2.1 Multiplexer

The multiplexer (or data selector) circuit was introduced in section 5.8.4. It is a special combinational circuit that gates one out of several inputs to a single output, and it is one of the most widely used standard logic circuits in digital design. Because of its widespread use, it has been fabricated as MSI IC and is commercially available in various sizes, such as 2:1, 4:1, 8:1, and 16:1 multiplexers.

In a multiplexer, the input selected is controlled by a set of *select* inputs. Figure 6.1 shows the block diagram of a multiplexer with n input lines and one output line. For selecting one out of n inputs for connection to the output, a set of m select inputs is required, where $2^m = n$. Depending upon the digital code applied at the select inputs one out of n data sources is selected and transmitted to a single output channel. Normally, a *strobe*

(or *enable*) input (G) is incorporated which helps in cascading and it is generally *active-low*, which means it performs its intended operation when it is LOW.

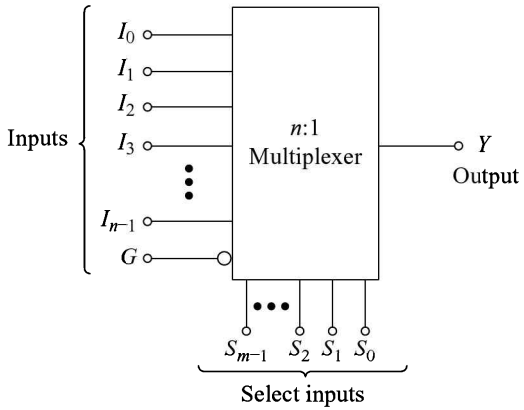


Fig. 6.1 ***Block Diagram of a Digital Multiplexer***

Its output Y is given by

$$Y = \overline{G} \cdot [\overline{S}_{m-1} \cdot \overline{S}_{m-2} \cdots \overline{S}_1 \cdot \overline{S}_0 \cdot I_0 + \overline{S}_{m-1} \cdot \overline{S}_{m-2} \cdots \overline{S}_1 \cdot S_0 \cdot I_1 + \cdots + S_{m-1} \cdot S_{m-2} \cdots S_1 \cdot \overline{S}_0 \cdot I_{n-2} + S_{m-1} \cdot S_{m-2} \cdots S_1 \cdot S_0 \cdot I_{n-1}] \quad (6.1)$$

Table 6.1 gives the truth table of a 4:1 multiplexer with active-low enable input (G).

Table 6.1 ***Truth Table of a 4:1 Multiplexer***

Enable input	Select inputs		Output
G	S_1	S_0	Y
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	X	X	0

Using Eq (6.1), its output Y will be

$$Y = (\overline{S}_1 \overline{S}_0 I_0 + \overline{S}_1 S_0 I_1 + S_1 \overline{S}_0 I_2 + S_1 S_0 I_3) \cdot \overline{G} \quad (6.2)$$

Equation (6.2) can be realised using NAND gates and the realisation is given in Fig. 6.2. This circuit is identical to the circuit of Fig.5.30 except the strobe (enable) input G.

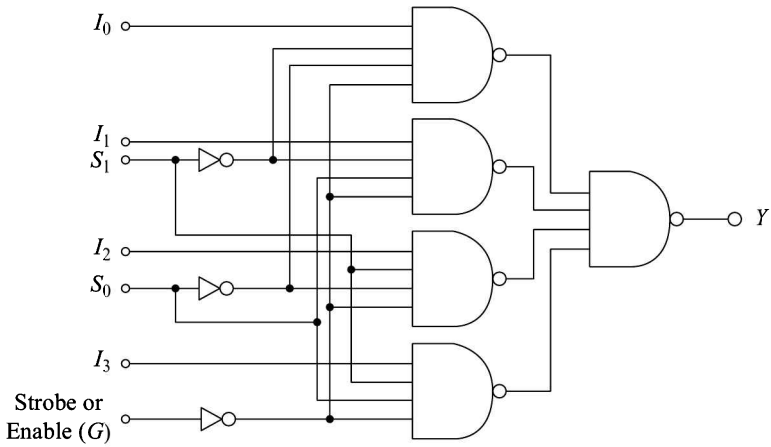


Fig. 6.2 ***A 4:1 Multiplexer with Strobe Input Using*** `NAND` ***Gates***

6.2.2 Combinational Logic Design Using Multiplexers

The multiplexing function discussed above can conveniently be used as a logic element in the design of combinational circuits. Standard ICs are available (Table 6.2) for 2:1, 4:1, 8:1, and 16:1 multiplexers.

Use of multiplexers offers the following advantages:

1. Simplification of logic expression is not required.
2. It minimises the IC package count.
3. Logic design is simplified.

Table 6.2 ***Available Multiplexer ICs***

IC No.	Description	Output
74157	Quad 2:1 Multiplexer	Same as input
74158	Quad 2:1 Multiplexer	Inverted input
74153	Dual 4:1 Multiplexer	Same as input
74352	Dual 4:1 Multiplexer	Inverted input
74151	8:1 Multiplexer	Complementary outputs
74152	8:1 Multiplexer	Inverted input
74150	16:1 Multiplexer	Inverted input

For using the multiplexer as a logic element, either the truth table or one of the canonical forms of logic expression must be available. The design procedure is given below:

1. Identify the decimal number corresponding to each minterm in the expression. The input lines corresponding to these numbers are to be connected to logic 1 level.
2. All other input lines are to be connected to logic 0 level.
3. The inputs are to be applied to select inputs.

The following examples illustrate the above procedure.

Example 6.1

Implement the expression using a multiplexer.

$$f(A, B, C, D) = \Sigma m(0, 2, 3, 6, 8, 9, 12, 14)$$

Solution

Since there are four variables, therefore, a multiplexer with four select inputs is required. The circuit of 16:1 multiplexer connected to implement the above expression is shown in Fig. 6.3. This implementation requires only one IC package. In case the output of the multiplexer is active-low, the logic 0 and logic 1 inputs of Fig. 6.3 are to be interchanged. The reader should verify the validity of this statement.

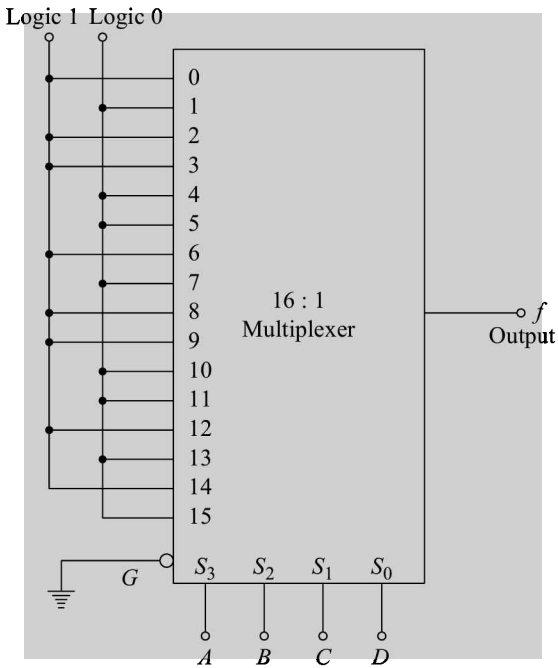


Fig.6.3 ***Implementation of Logic Expression of Ex. 6.1***

Example 6.2

Realise the logic function of the truth table given in Table 6.3.

Table 6.3 ***Truth table of Example 6.2***

Inputs				Output
A	***B***	***C***	***D***	***Y***
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Solution

(i) *First Method*: This can be realised using the method used in Example 6.1. Here, the input lines 2, 4, 6, 7, 9, 10, 11, 12, and 15 are to be connected to logic 1 and the input lines 0, 1, 3, 5, 8, 13, and 14 are to be connected to logic 0.

(ii) *Second Method*: A four variable truth table or logic expression can be realised by using an 8:1 multiplexer instead of a 16:1 multiplexer. For this, partition the truth table as shown by dotted lines. Here the inputs A, B, and C are to be connected to S_2, S_1 and S_0 select inputs respectively. Now, we observe the relationship between input D and output Y for each group of two rows. There are four possible values of Y and these are 0, 1, D, and $\bar{D}$. These are given in Table 6.4. From this table, we note the output Y for each of the combinations of A, B, and C, and then make the connections accordingly. The implementation of this function using an 8:1 multiplexer is shown in Fig. 6.4.

Table 6.4

Inputs			Output
A	*B*	*C*	*Y*
0	0	0	0
0	0	1	$\bar{D}$
0	1	0	$\bar{D}$
0	1	1	1
1	0	0	D
1	0	1	1
1	1	0	$\bar{D}$
1	1	1	D

The second method can also be used if the logic expression is specified.

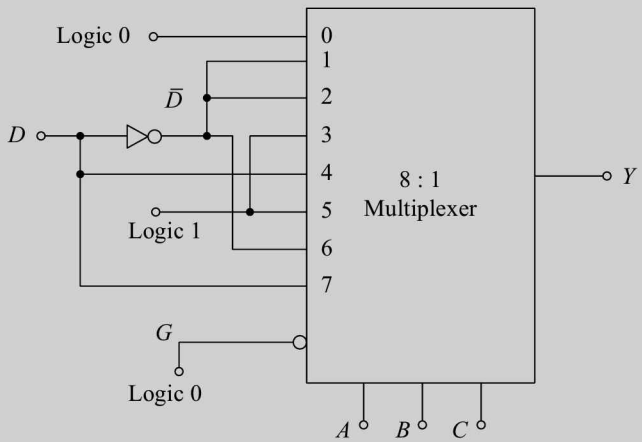


Fig. 6.4 ***Realisation of 4-variable Truth Table Using 8:1 Multiplexer***

6.2.3 Multiplexer Tree

Since 16-to-1 multiplexers are the largest available ICs, therefore, to meet the larger input needs there should be a provision for expansion. This is achieved with the help of enable/strobe inputs and multiplexers stacks or trees are designed. Two commonly used methods for this purpose are illustrated in Figs 6.5 and 6.6.

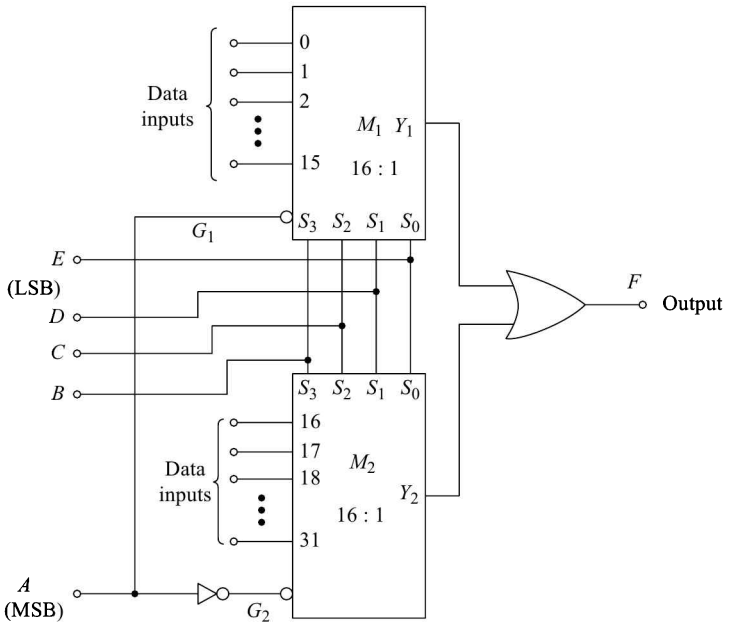


Fig. 6.5 ***32:1 Multiplexer Using Two 16:1 Multiplexers and One*** OR ***Gate***

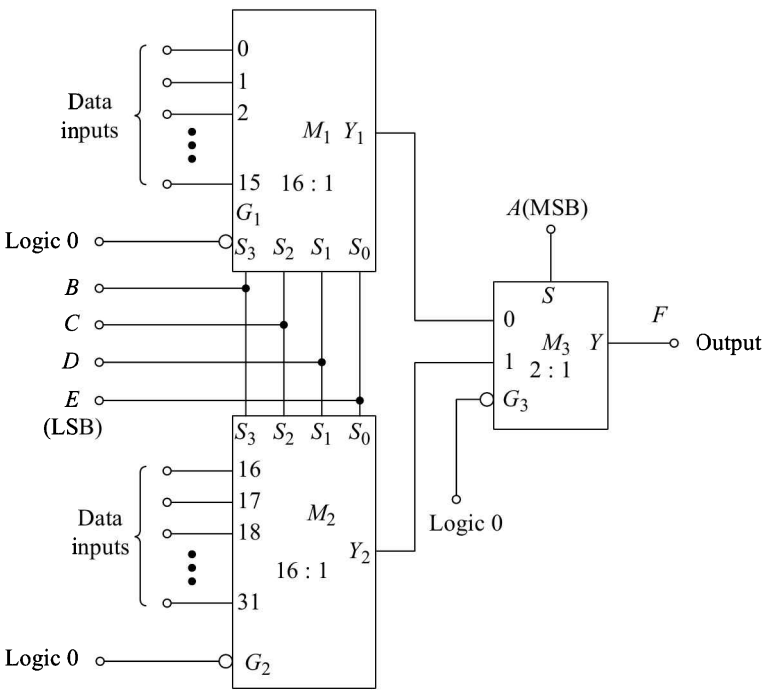


Fig. 6.6 ***32.1 Multiplexer Using Two 16:1 Multiplexers and One 2:1 Multiplexer***

The circuit of Fig. 6.5 uses two 16:1 multiplexers ($M_1$and M_2) for the realisation of a 32:1 multiplexer. The lower order 16 data input lines (I_0–I_{15}) are applied at the data input terminals of the multiplexer $M_1$and the higher order 16 data input lines (I_{16}–I_{31}) are applied at the data input terminals of the multiplexer M_2. For a 32:1 multiplexer, the number of select input lines is required to be 5. The 5-bit select inputs are *ABCDE*. The most-significant select input bit A is applied at G_1 and $\overline{A}$ is applied at G_2. *B,C,D*, and *E* are connected to S_3, S_2, S_1,and S_0 inputs of both the multiplexers. The output F of the multiplexer is obtained by using an OR gate, where $F = Y_1 + Y_2$.

When $A = 0$, the multiplexer M_1 is enabled and M_2 is disabled, therby allowing one of the lower order 16 bits to Y_1, depending upon the value of *BCDE*. $Y_2 = 0$. Therefore, $F = Y_1$. Similarly, when $A = 1$, M_2 is enabled and M_1 is disabled and $F = Y_2$.

The circuit of Fig. 6.6 uses a 2:1 multiplexer instead of an OR gate. When $A = 0$, $F = Y_1$ and when $A = 1$, $F = Y_2$. Thus, both the circuits perform similar operation.

These two general techniques can be used to expand to an n input multiplexer without any difficulty.

6.3 DEMULTIPLEXERS/DECODERS AND THEIR USE IN COMBINATIONAL LOGIC DESIGN

6.3.1 Demultiplexer

The demultiplexer performs the reverse operation of a multiplexer. It accepts a single input and distributes it over several outputs. Figure 6.7 gives the block diagram of a demultiplexer. The select input code determines to which output the data input will be transmitted.

The number of output lines is n and the number of select lines is m, where $n = 2^m$. The data input D_i will appear on the output line selected by the select input. For example, if the decimal equivalent of the select input is 4, then the data will appear on D_4 output line. This circuit can also be used as binary-to-decimal decoder with binary inputs applied at the select input lines and the output will be obtained on the corresponding line. The data input line is to be connected to logic 1 level. This circuit can be designed using gates and it is left as an exercise for the reader. However, this device is available as an MSI IC and can conveniently be used for the design of combinational circuits. The device is very useful if multiple-output combinational circuit is to be designed, because this needs minimum package count. These devices are available (Table 6.5) as

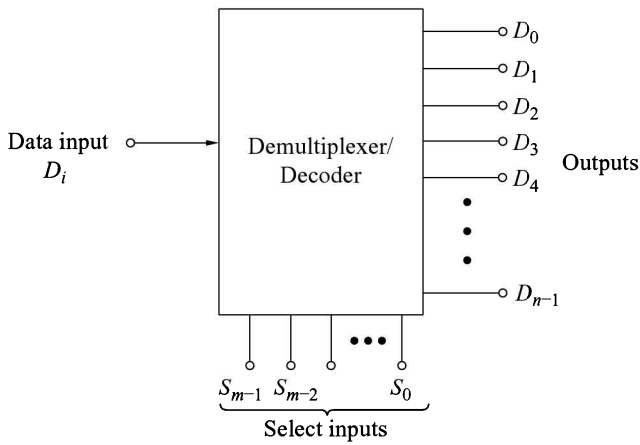


Fig. 6.7 ***Block Diagram of a Demultiplexer.***

2-line-to-4-line, 3-line-to-8-line, and 4-line-to-16-line decoders. The outputs of most of these devices are active-low, also there is an active-low enable/data input terminal available.

Unlike the multiplexer, the decoder does require some gates in order to realise Boolean expressions in the canonical SOP form. The following example illustrates its use in combinational logic design.

Table 6.5 ***Available Demultiplexer ICs***

IC No.	Description	Output
74139	Dual 1:4 Demultiplexer (2-line-to-4-line decoder)	Inverted input
74155	Dual 1:4 Demultiplexer (2-line-to-4-line decoder)	1 *Y*–Inverted input 2*Y*–Same as input
74156	-do-	Open-collector 1 *Y*–Inverted input 2 *Y*–Same as input
74138	1:8 Demultiplexer (3-line-to-8-line decoder)	Inverted input
74154	1:16 Demultiplexer (4-line-to-16-line decoder)	Same as input
74159	-do-	Same as input Open-collector

Example 6.3

Implement the following multi-output combinational logic circuit using a 4-to-16-line decoder.

$$F_1 = \Sigma m\ (1, 2, 4, 7, 8, 11, 12, 13)$$
$$F_2 = \Sigma m\ (2, 3, 9, 11)$$
$$F_3 = \Sigma m\ (10, 12, 13, 14)$$
$$F_4 = \Sigma m\ (2, 4, 8)$$

Solution

The realisation is shown in Fig. 6.8.

The four-bit input *ABCD* is applied at the Select input terminals S_3, S_2, S_1, and S_0. The output F_1 is required to be 1 for minterms 1, 2, 4, 7, 8, 11, 12, and 13. Therefore, a NAND gate is connected as shown. Similarly NAND gates are used for the outputs F_2, F_3, and F_4. Here, the decoder's outputs are active-low, therefore a NAND gate is required for every output of the combinational circuit.

In the combinational logic design using multiplexer, additional gates are not required, whereas design using demultiplexer requires additional gates. However, even with this disadvantage, the decoder is more economical in cases where nontrivial, multiple-output expressions of the same input variables are required. In such cases, one multiplexer is required for each output whereas it is likely that only one decoder will be required, supported with a few gates. Therefore, using a decoder could have advantages over using a multiplexer.

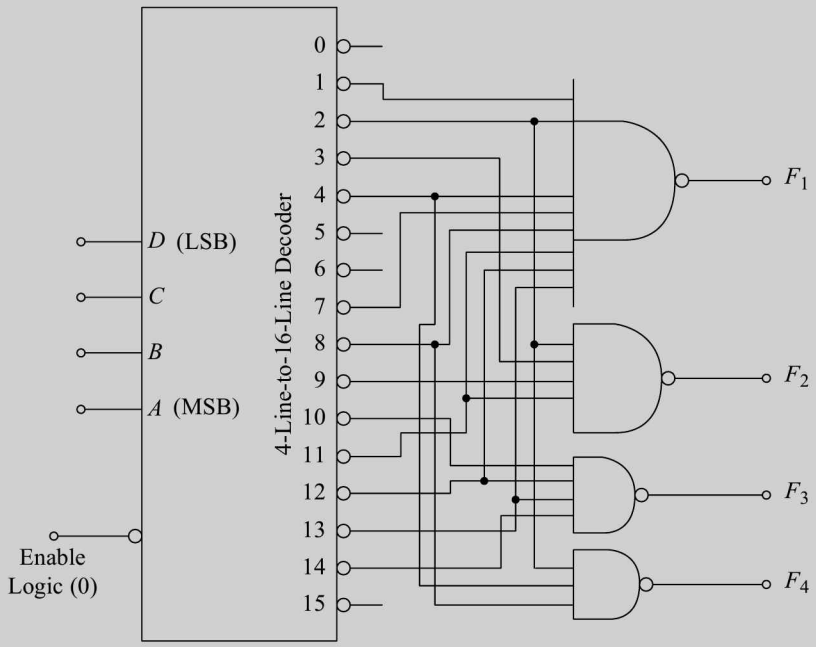


Fig. 6.8 ***Implementation of Combinational Logic Circuit of Ex. 6.3***

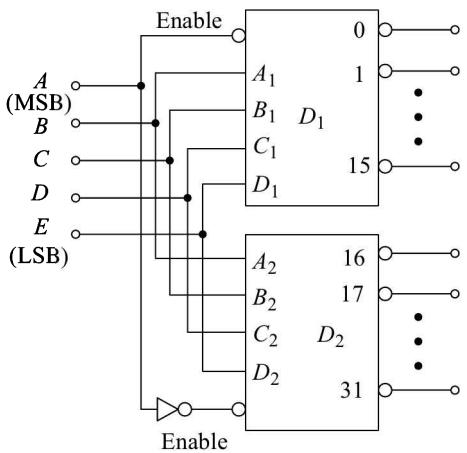


Fig. 6.9 ***5-Line-to-32-Line Decoder Using Two 4-line-to-16-Line Decoders***

6.3.2 Demultiplexer Tree

Since 4-line-to-16-line decoders are the largest available circuits in ICs, to meet the larger inputs need there should be a provision for expansion. This is made possible by using *enable* input terminal. Figure 6.9 shows a 5-line-to-32-line decoder and Fig. 6.10 shows a 8-line-to-256-line decoder using 4-line-to-16-line decoders. In a similar way, any *m*-line-to-*n*-line decoder can be implemented. However, if only a few codes of a large number need be recognised, the alternative approach, such as the one shown in Fig. 6.11, can be used. This is connected to detect the digital number 00011111. The most-significant 4-bits are applied at *A B C D* inputs and the least-significant bits are applied at *E F G H* inputs. The output goes low when the most-significant bits are 0 0 0 1 and the least-significant bits are 1 1 1 1. The circuit can be expanded to detect other 8-bit codes.

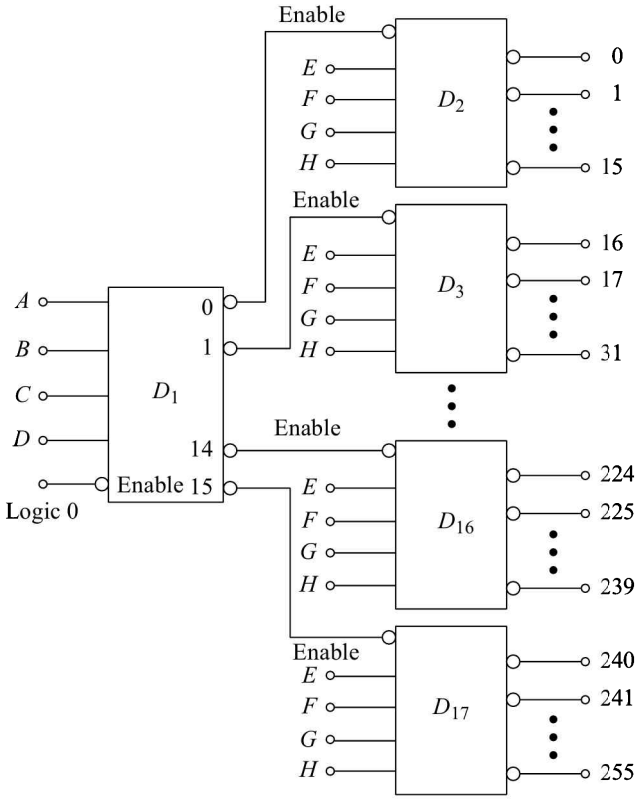


Fig. 6.10 ***8-Line-to-256-Line Decoder Using 4-Line-to-16-Line Decoders Tree***

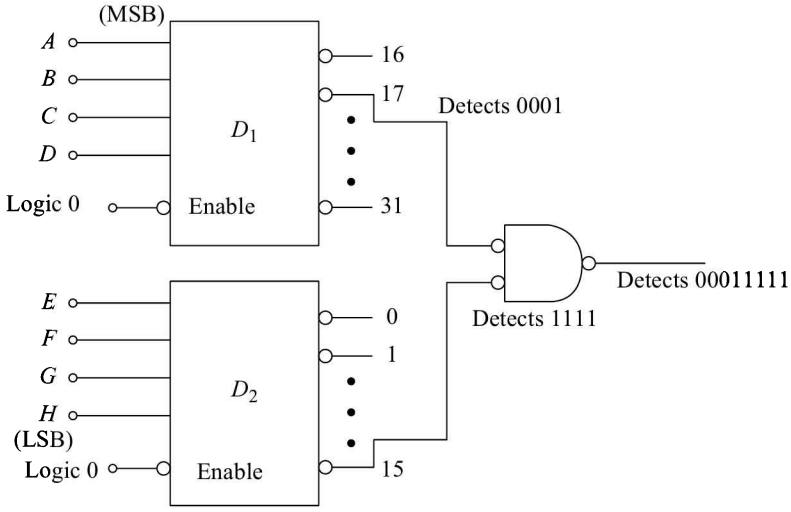


Fig. 6.11 ***An Alternative Approach to Decode Some Combinations***

6.4 ADDERS AND THEIR USE AS SUBTRACTORS

Half- and full-adders and subtractors for two one-bit binary numbers have been discussed in section 5.8.1. Using these circuits, we can design adders and subtractors for n-bit numbers. However, we can perform both addition and subtraction using only adders because the problem of subtraction becomes that of an addition when we use 1's and 2's complement representation of negative numbers. Therefore, adders have become widely-used standard circuits available in MSI.

An adder circuit for addition of two n-bit binary numbers consists of n full-adder circuits. It accepts two n-bit binary numbers as inputs and produces an $(n + 1)$-bit binary number as the sum. Figure 6.12*a* shows an n-bit adder using full-adders and Fig. 6.12*b* shows its block diagram.

Here, A and B are the two n-bit inputs to be added and $C_{n-1}\, S_{n-1}\, S_{n-2} \ldots S_2\, S_1\, S_0$ is their sum. In this, a half-adder may be used to add the least-significant bits A_0 and B_0. However, for cascading these adders to increase the number of bits to be added, the CARRY input terminal is required for the adder to add the least-significant bits. Therefore, all the adders used are full-adders. 2-bit and 4-bit adder circuits are available in ICs.

If the n-bit adder is implemented using the scheme of Fig. 6.12*a*, the carry has to ripple down the line of cascaded adders from the LSB to MSB position. This decreases the operating speed of the adder. The time required for addition operation to be completed is limited by the amount of time required to complete the ripple-carry operation (Prob. 6.11). A technique known as the *look-ahead carry* generation process is used for increasing the speed of operation. This technique anticipates the carry bits for each stage before the actual summing operation.

Adder ICs are designed incorporating the look-ahead carry generator circuit.

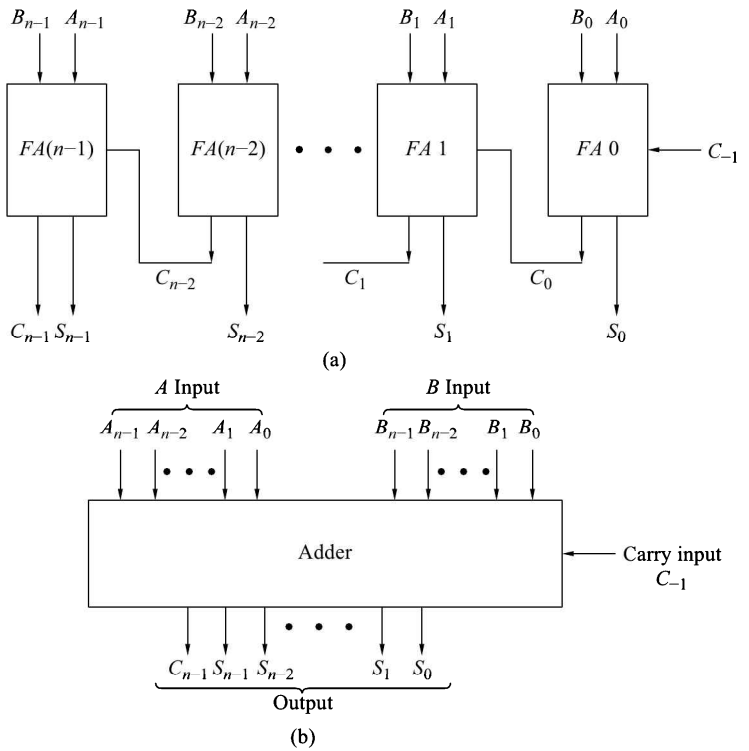


Fig. 6.12 ***(a) n-bit Binary Adder Using n Full-Adders***
(b) Block Diagram of an n-bit Binary Adder

6.4.1 Adder with Look-Ahead Carry

In the half-adder circuit of Fig. 5.19 and full-adder circuits of Fig. 5.21 and Prob. 5.19*b*, it was assumed that the inputs (augend and addend) and carry input are simultaneously present and the sum and carry outputs are generated instantaneously. However, in Prob. 5.20 it has been shown that even if the augend, addend, and carry inputs are present simultaneously, then the sum and carry outputs will be delayed due to the propagation delays of gates through which the signals are passing. Now, if we consider the *n*-bit parallel adder of Fig. 6.12*a*, we observe that the sum (S_0) and carry (C_0) outputs are delayed because of the propagation delays of the gates involved in FA0. The S_0 output, however, is the LSB of the final result and it is not required to be passed through other gates and hence does not get further delayed. The carry output C_0 acts as carry input of the full-adder FA1 and therefore, the outputs of FA1 S_1 and C_1, will reach steady-state only after arrival of C_0 and propagation delay introduced by FA1. This means the total delay upto FA1 will be the sum of delays introduced by FA0 and FA1. Similarly going further in the chain of adders towards MSB, the carry has to ripple through all the stages, thereby reducing the speed of the adder as the number of adder stages are increased (Prob. 6.11).

To design fast operating parallel adders, we can use gates with lower propagation delay time. Even with this, the delay time of the adder will increase with increasing number of bits to be added. Another approach most commonly used is the concept of *look-ahead carry*. Although it requires additional circuitry but the speed of the adder becomes independent of the number of bits.

Let us consider a full-adder circuit in block diagram form and its EX-OR realisation (Prob. 5.19(b)) shown in Fig. 6.13. Here,

$$P_i = A_i \oplus B_i \tag{6.3a}$$
$$G_i = A_i B_i \tag{6.3b}$$
$$S_i = P_i \oplus C_{i-1} = A_i \oplus B_i \oplus C_{i-1} \tag{6.3c}$$
$$C_i = G_i + P_i C_{i-1} \tag{6.3d}$$

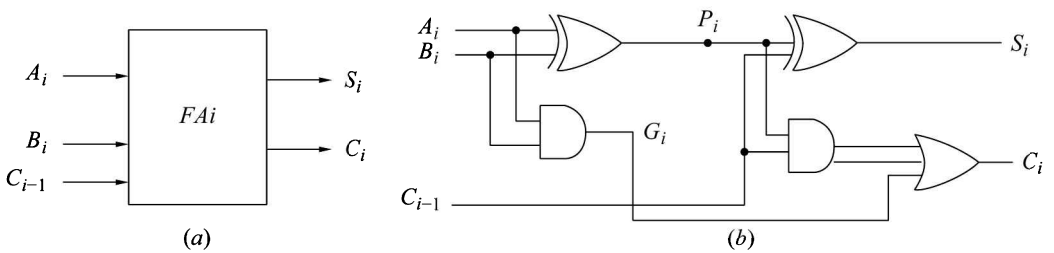


Fig. 6.13 ***(a) Block Diagram of ith Stage of a Full-Adder***
(b) EX-OR ***Implementation of Full-Adder***

The output G_i of the first half-adder is 1 if A_i and B_i both are 1 and a carry is generated. The variable G_i is known as a *carry generate*. Its value is independent of the input carry. The variable P_i is known as a *carry propagate* because this is the term associated with the propagation of the carry from C_{i-1} to C_i. Using Eq. (6.3d), we write Boolean expression for carry output of each stage:

$$C_{i+1} = G_{i+1} + P_{i+1} \cdot C_i$$

Substituting the value of C_i from Eq. (6.3d), we obtain,

$$C_{i+1} = G_{i+1} + P_{i+1}\, G_i + P_{i+1}\, P_i C_{i-1} \tag{6.4}$$

Similarly, for an *n*-stage adder, the final carry C_{n-1} can be determined. For clear understanding of the advantage of this approach, we consider a 4-bit adder and formulate Boolean expressions for the carry outputs C_0, C_1, C_2, and C_3. We obtain,

$$P_0 = A_0 \oplus B_0 \quad \text{and} \quad G_0 = A_0 B_0$$

$$C_0 = G_0 + P_0 C_{-1} \tag{6.5a}$$

$$C_1 = G_1 + P_1 G_0 + P_1 P_0 C_{-1} \tag{6.5b}$$

$$C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{-1} \tag{6.5c}$$

$$C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_{-1} \tag{6.5d}$$

Now, let us observe the logic variables involved in Eq. (6.5), these are,

$$G_0, G_1, G_2, G_3, P_0, P_1, P_2, P_3, \text{ and } C_{-1}$$

The G variables can be generated directly from A and B inputs using AND gates (Eq. 6.3b), the P variables are obtained again directly from A and B inputs using EX-OR gates (Eq. 6.3a). C_{-1} is the carry input. If Gs, Ps, and C_{-1} are available simultaneously, the carry outputs C_0, C_1, C_2, and C_3 are produced by using 2-level realisation (AND-OR or NAND-NAND) since Eq. 6.5 gives these outputs in SOP form. Therefore, for the generation of these carry outputs, propagation delay time of two gates only will be there. These carry outputs are connected to the carry inputs of the succeeding stages, thereby eliminating the problem of carry rippling through all the stages.

Logic circuit of a look-ahead carry generator can be prepared using Eqs. 6.5. A 4-bit adder with look-ahead carry is shown in Fig. 6.14. Thus we see that by generating all the individual carry terms needed by each full-adder in a 2-level circuit, the propagation delay time through the adder is considerably reduced and it becomes independent of the number of full-adders in the circuit.

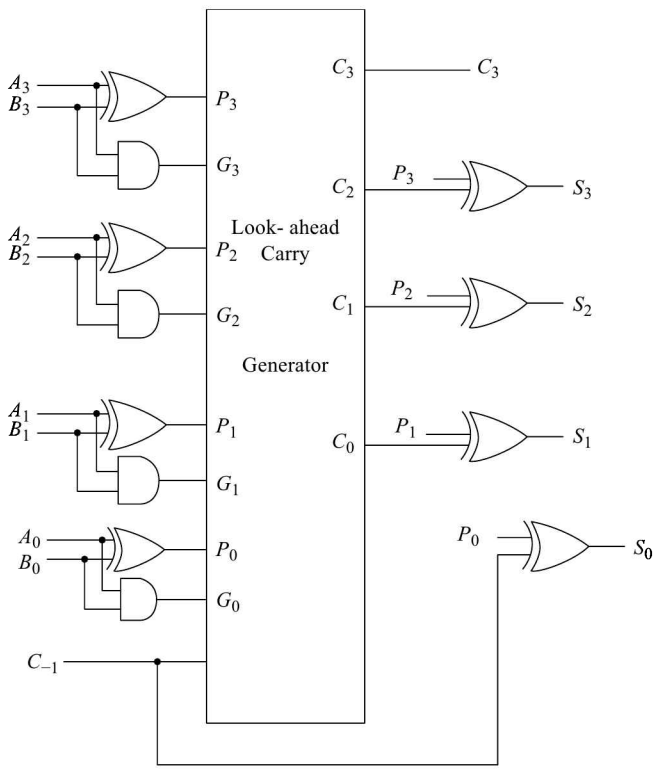


Fig. 6.14 ***A 4-bit Adder with Look-Ahead Carry***

6.4.2 Cascading of Adders

The bit lengths of the numbers to be added can be increased by cascading the adders. Figure 6.15 shows an 8-bit adder using two 4-bit adders. In a similar way, we can form an adder tree for making an n-bit adder.

Here $A_7A_6\cdots A_1A_0$ and $B_7B_6\cdots B_1B_0$ are the two 8-bit numbers to be added and $C_7S_7S_6\cdots S_1S_0$ is the sum.

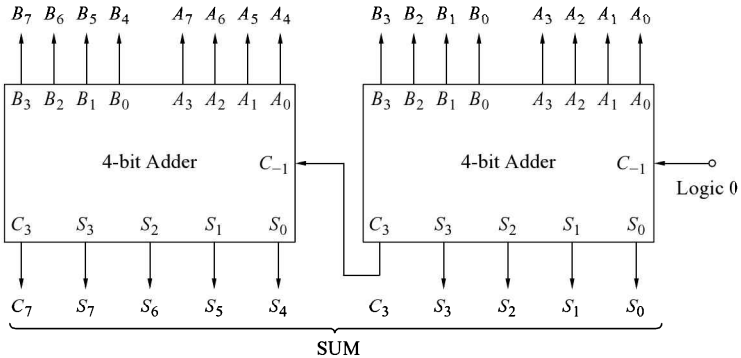


Fig. 6.15 ***An 8-bit Adder as a Cascade of Two 4-bit Adders***

6.4.3 Subtraction using Adder

As discussed in Chapter 2, the problem of subtraction gets converted into that of addition if 1's and 2's complement representation are used for representing negative numbers. The algorithm for a subtractor using adder is given in Fig. 6.16.

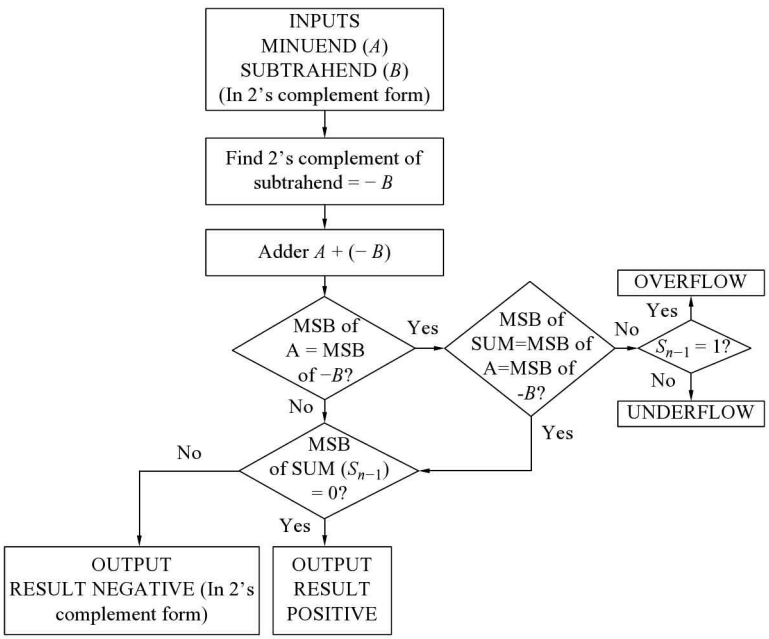


Fig. 6.16 ***Algorithm for Performing Subtraction Using Adder***

The two numbers A and B can be of the same sign or of opposite signs. If the two numbers are of unlike sign, we may come across the problem of *overflow* or *underflow*. Overflow occurs when the subtraction operation produces a number larger than the largest possible number which can be represented by n-bits. On the other hand, underflow occurs when the result produced is smaller than the smallest number which can be represented by n-bits. The overflow and underflow logic is illustrated in Fig. 6.16. The subtractor circuit is given in Fig. 6.17. The reader can verify the operation of this circuit. If overflow or underflow occurs then the result is wrong. This circuit can be converted into an ADDER/SUBTRACTOR circuit with ADD/SUB control (Problem 6.4).

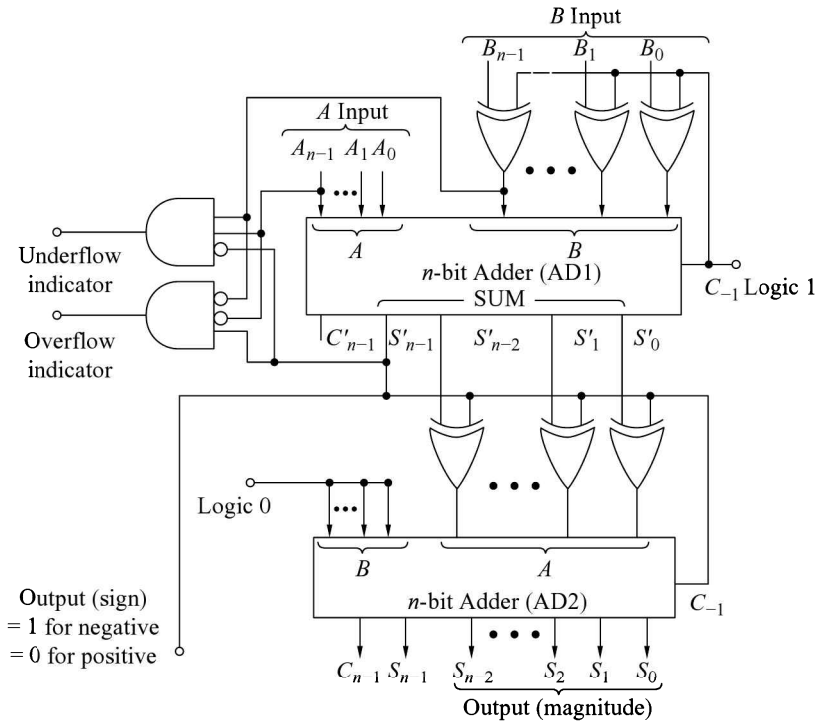


Fig. 6.17 ***n-bit Subtractor Circuit Using n-bit Adders***

6.5 BCD ARITHMETIC

Quite often, BCD code is used to represent decimal numbers, for example, in a calculator. Therefore, addition and subtraction are required to be performed in BCD code.

6.5.1 BCD Adder

The 4-bit binary adder IC (7483) can be used to perform addition of BCD numbers. In this, if the four-bit sum output is not a valid BCD digit, or if a carry C_3 is generated, then decimal 6(0 1 1 0 binary) is to be added to

the sum to get the correct result. Figure 6.18 shows a 1-digit BCD adder. BCD adders can be cascaded to add numbers several digits long by connecting the carry-out of a stage to the carry-in of the next stage.

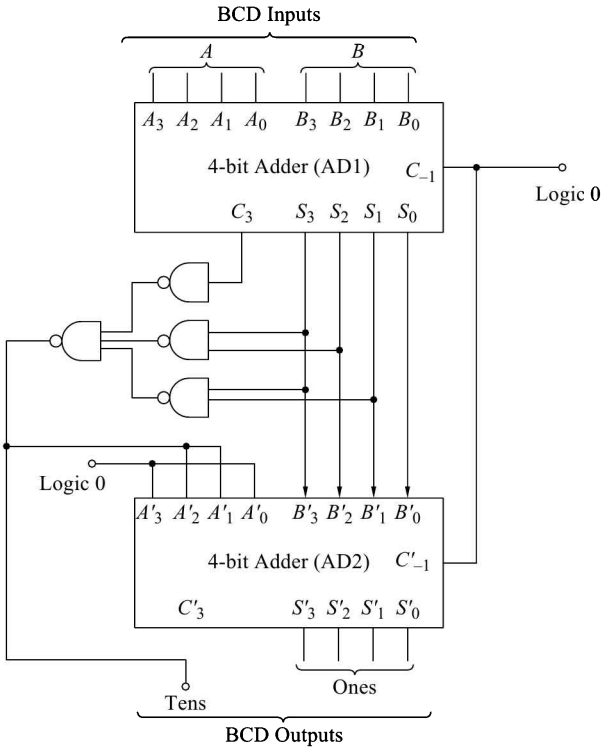


Fig. 6.18 ***One Digit BCD Adder***

6.5.2 BCD Subtractor

For BCD subtraction, nine's complement of the subtrahend is added to the minuend. The nine's complement of a BCD number is given by nine minus that number. For example, nine's complement of $7 = 9 - 7 = 2$. In BCD code we say that the nine's complement of 0 1 1 1 is 0 0 1 0. Examples of BCD subtraction using nine's complement to represent negative numbers are given below:

Example 6.4

(a) Subtract 5 from 9
(b) Subtract 1 from 8
(c) Subtract 8 from 4

Solution

(a)

9 =	1001	
– 5 =	(+) 0100	(nine's complement of 5)
	1101	(Invalid)
Add	0110	
	10011	
	→ 1	Add End Around Carry (EAC)
	0100	= + 4

(b)

8 =	1000	
– 1 =	(+) 1000	(nine's complement of 1)
	10000	(Invalid)
Add	0110	
	10110	
	→ 1	Add (EAC)
	0111	= + 7

(c)

4 =	0100	
– 8 =	(+) 0001	(nine's complement of 8)
	0101	

Nine's complement of 0101 = 4

Therefore, the answer is –4.

From the above example, we conclude the following:

1. If the sum of minuend and the nine's-complement of subtrahend is an invalid BCD code (Ex. 6.4(a)) or if a carry is produced from the MSB (Ex. 6.4(b)), add decimal 6 (binary 0110) and the end around carry (EAC) to this sum. The result is positive number represented by this sum.
2. If the sum of the minuend and the nine's complement of the subtrahend is a valid BCD code, the result is negative and is in the nine's complement form (Ex. 6.4(c)).

Nine's complement of a number can be determined by adding 1010 to the one's complement of the number. Figure 6.19 shows a nine's complementer circuit using a 4-bit adder and EX–OR gates.

Figure 6.20 shows a one-digit BCD subtracter, using the nine's complementer circuit of Fig. 6.19.

We observe that the BCD arithmetic is rather complex in comparison to the straight binary arithmetic. Therefore, BCD arithmetic requires more hardware and results in reduction of speed. Also, more number of bits are required to represent a given number in BCD, therefore, BCD is normally not used in computers. However, digital calculators use BCD because the input data from the keyboard as well as the output display are decimal. Although the process of encoding BCD from decimal is simple and changing from decimal to binary is complex, even then the calculators have rather complex circuitry which are costly and considerably slower than computers.

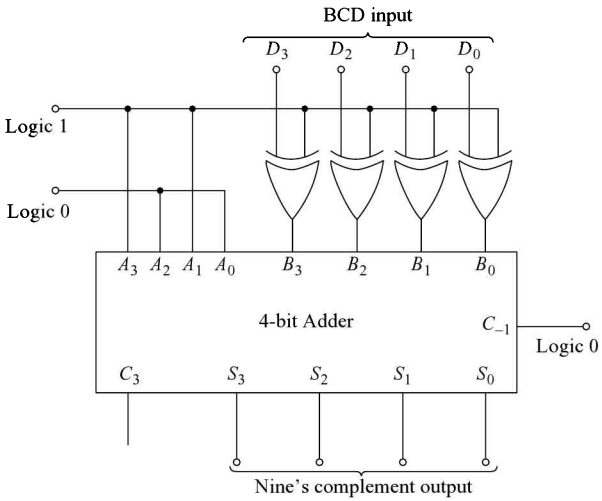


Fig. 6.19 ***A Nine's Complementer Circuit.***

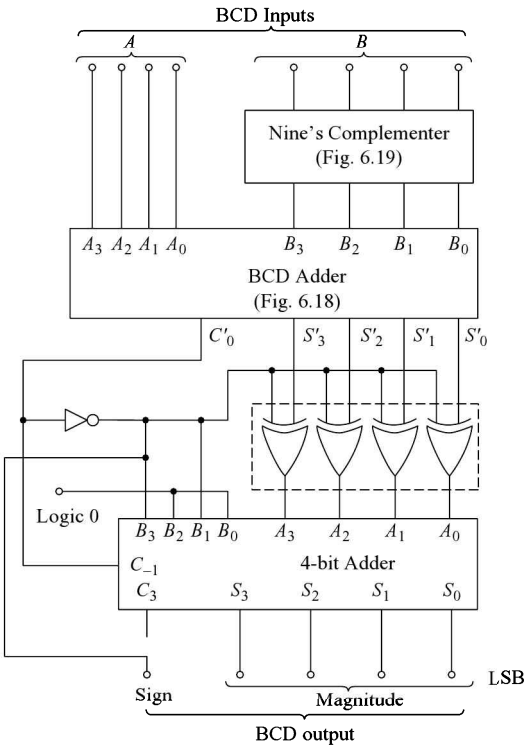


Fig. 6.20 ***A One-Digit BCD Subtractor***

6.6 ARITHMETIC LOGIC UNIT (ALU)

A very popular and widely used combinational circuit is ALU which is capable of performing arithmetic as well as logical operations. This is the heart of any microprocessor. Figure 6.21 shows the block diagram of 74181 ALU and Table 6.6 gives its function table.

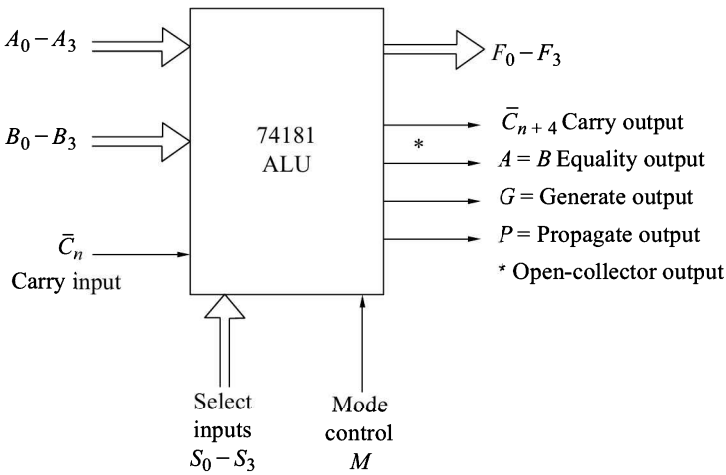


Fig. 6.21 ***Block Diagram of 74181 ALU***

The functions of various input, output and control lines are given below:

A and B: 4-bit binary data inputs

$\overline{C}_n$: Carry input (active-low)

F: 4-bit binary data output

$\overline{C}_{n+4}$: Carry output (active-low)
For subtraction operation, it indicates the sign of the output. Logic 0 indicates positive result and logic 1 indicates negative result expressed in 2's complement form.

$A = B$: Logic 1 on this line indicates $A = B$

G: Carry generate output

Table 6.6 ***Function Table of 74181 ALU***

						Active high data	
	Selection				$M = 1$ Logic	$M = 0$; Arithmetic operations	
Line	S_3	S_2	S_1	S_0	Functions	$\overline{C}_n = 1$ (no carry)	$\overline{C}_n = 0$ (with carry)
0	0	0	0	0	$F = \overline{A}$	$F = A$	$F = A$ PLUS 1
1	0	0	0	1	$\overline{A + B}$	$F = A + B$	$F = (A + B)$ PLUS 1

(*Continued*)

Table 6.6 **(*Continued*)**

Line	Selection S_3	S_2	S_1	S_0	$M = 1$ Logic Functions	Active high data: $M = 0$; Arithmetic operations $\overline{C}_n = 1$ (no carry)	$\overline{C}_n = 0$ (with carry)
2	0	0	1	0	$F = \overline{A} \cdot B$	$F = A + B$	$F = (A + \overline{B})$ PLUS 1
3	0	0	1	1	$F = 0$	F = MINUS 1 (2's COMPL)	F = ZERO
4	0	1	0	0	$F = \overline{AB}$	$F = A$ PLUS $A\overline{B}$	$F = A$ PLUS $A\overline{B}$ PLUS 1
5	0	1	0	1	$F = \overline{B}$	$F = (A + B)$ PLUS $A\overline{B}$	$F = (A + B)$ PLUS $A\overline{B}$ PLUS 1
6	0	1	1	0	$F = A \oplus B$	$F = A$ MINUS B MINUS 1	$F = A$ MINUS B
7	0	1	1	1	$F = A\overline{B}$	$F = A\overline{B}$ MINUS 1	$F = A\overline{B}$
8	1	0	0	0	$F = \overline{A} + B$	$F = A$ PLUS AB	$F = A$ PLUS AB PLUS 1
9	1	0	0	1	$F = \overline{A \oplus B}$	$F = A$ PLUS B	$F = A$ PLUS B PLUS 1
10	1	0	1	0	$F = B$	$F = (A + \overline{B})$ PLUS AB	$F = (A + \overline{B})$ PLUS AB PLUS 1
11	1	0	1	1	$F = AB$	$F = AB$ MINUS 1	$F = AB$
12	1	1	0	0	$F = 1$	$F = A$ PLUS A*	$F = A$ PLUS A PLUS 1
13	1	1	0	1	$F = A + \overline{B}$	$F = (A + B)$ PLUS A	$F = (A + B)$ PLUS A PLUS 1
14	1	1	1	0	$F = A + B$	$F = (A + \overline{B})$ PLUS A	$F = (A + \overline{B})$ PLUS A PLUS 1
15	1	1	1	1	$F = A$	$F = A$ MINUS 1	$F = A$

* Each bit is shifted to the next more significant position.

P: Carry propagate output

G and P outputs are used when a number of 74181 circuits are used in cascade along with 74182 Look-ahead Carry-generator circuit to make the arithmetic operations faster.

Select input (S) : Used to select any operation (Table 6.6)

Mode Control (M) : $M = 0$ Arithmetic operations

$M = 1$ Logic operations

The 74181 can be cascaded by connecting the carry-out of a stage to the carry-in of the succeeding stage.

Example 6.5

Design an 8-bit adder/subtractor using 74181s in cascade. Show how it works if
(a) $A = 97$ and $B = 29$ (b) $A = 24$ and $B = 58$.

Solution

For designing an 8-bit adder/subtractor, two 74181 ICs are to be cascaded. The least-significant four bits of A and B are applied at the A and B inputs of the least-significant 74181 and the most-significant four bits of A and B are applied at the A and B inputs of the most-significant 74181. The carry-out of the least-significant 74181 is connected

to the carry-in of the most-significant 74181. The 8-bit output will be available on F outputs. The select lines of both the 74181s are connected together.

Addition is performed with $M = 0$ and $S = 1001$, and subtraction is performed with $M = 0$ and $S = 0110$. Connect $\overline{C}_n$ of least-significant 74181 to 1 for addition and 0 for subtraction.

(a) $A = 97 = 01100001$
$B = 29 = 00011101$

(b) $A = 24 = 00011000$
$B = 58 = 00111010$

The various inputs and outputs are given in Table 6.7.

Table 6.7 ***Table for Example 6.5***

Part	Mode control	Select Inputs	Least-significant ALU					Most-significant ALU					Output	Remarks
			Inputs			Outputs		Inputs			Outputs			
	M	$S_3S_2S_1S_0$	A_L	B_L	$\overline{C}_n$	S_L	$\overline{C}_{n+4}$	A_H	B_H	$\overline{C}_n$	S_H	$\overline{C}_{n+4}$		
(a)	0	1 0 0 1	0001	1101	1	1110	1	0110	0001	1	0111	1	01111110	$(126)_{10}$
	0	0 1 1 0	0001	1101	0	0100	1	0110	0001	1	0100	0	01000100	$(68)_{10}$
(b)	0	1 0 0 1	1000	1010	1	0010	0	0001	0011	0	0101	1	01010010	$(82)_{10}$
	0	0 1 1 0	1000	1010	0	1110	1	0001	0011	1	1101	1	11011110	$-(34)_{10}$

6.7 DIGITAL COMPARATORS

The principle of comparing digital signals has been given in Section 1.5. Comparators can be designed for comparing multibit numbers. Figure 6.22 shows the block diagram of an n-bit comparator. It receives two n-bit numbers A and B as inputs and the outputs are $A > B$, $A = B$, and $A < B$. Depending upon the relative magnitude of the two numbers, one of the outputs will be HIGH. Table 6.8 gives the truth table of a 2-bit comparator. The reader is advised to simplify the expressions for $A > B$, $A = B$, and $A < B$ outputs using K-map and design the circuit using gates. However, 4-bit comparators are available in MSI (7485) which can compare straight binary and natural BCD codes. These ICs can be cascaded to compare words of greater lengths without external gates. The $A > B$, $A = B$, and $A < B$ outputs of a stage handling less-significant bits are connected to the corresponding $A > B$, $A = B$, and $A < B$ cascading inputs of the next stage handling more-significant bits. The stage handling the least-significant bits must have $A = B$ input connected to logic 1 level and $A > B$ and $A < B$ inputs connected to logic 0 or 1 level.

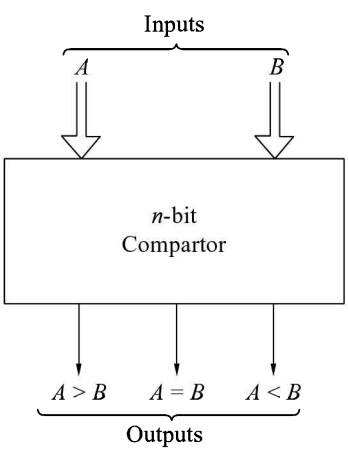


Fig. 6.22 ***Block Diagram of n-bit Comparator***

Table 6.8 ***Truth Table of a 2-Bit Comparator.***

Inputs				Outputs		
A_1	A_0	B_1	B_0	$A > B$	$A = B$	$A < B$
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

The function table of 7485 comparator is given in Table 6.9.

Table 6.9 ***Function Table of 7485 4-Bit Comparator***

Comparing inputs	Cascading inputs			Outputs		
A, B	$A > B$	$A = B$	$A < B$	$A > B$	$A = B$	$A < B$
$A > B$	X	X	X	1	0	0
	1	0	0	1	0	0
	X	1	X	0	1	0
$A = B$	0	0	1	0	0	1
	0	0	0	1	0	1
	1	0	1	0	0	0
$A < B$	X	X	X	0	0	1

Example 6.6

Design a 5-bit comparator using a single 7485 and one gate.

Solution

The circuit is shown in Fig. 6.23. The two 5-bit numbers to be compared are $A_4 A_3 A_2 A_1 A_0$ and $B_4 B_3 B_2 B_1 B_0$.

The operation of this circuit can be understood from the function table of 7485.

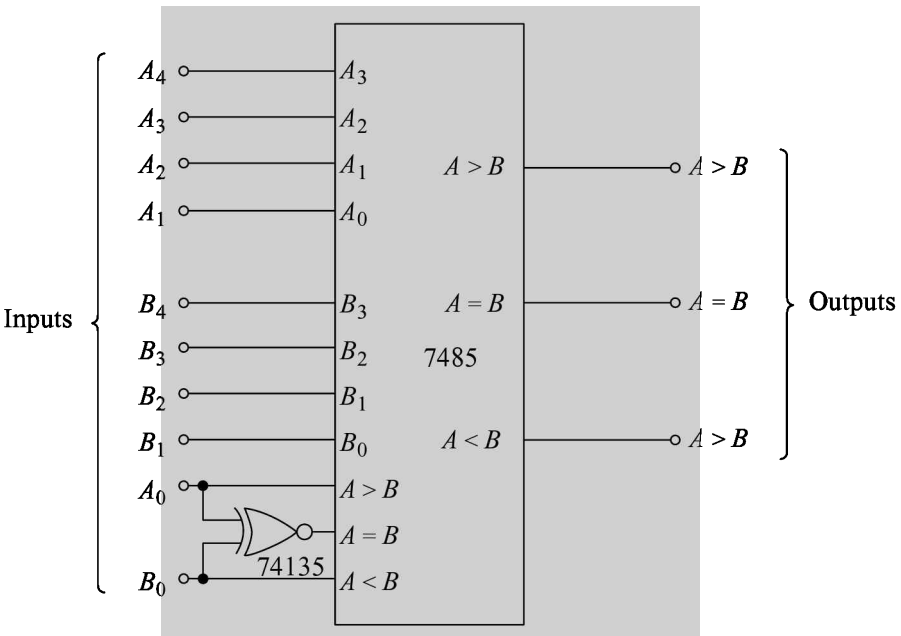


Fig. 6.23 ***A 5-bit Comparator.***

Example 6.7

Design a 24-bit comparator using six 7485 comparators in two levels.

Solution

Based on the approach used in Example 6.6, the circuit is given in Fig. 6.24. The reader is advised to verify the circuit assuming any two arbitrary 24-bit binary numbers. This method of cascading reduces comparison time in comparison to the straight method of cascading.

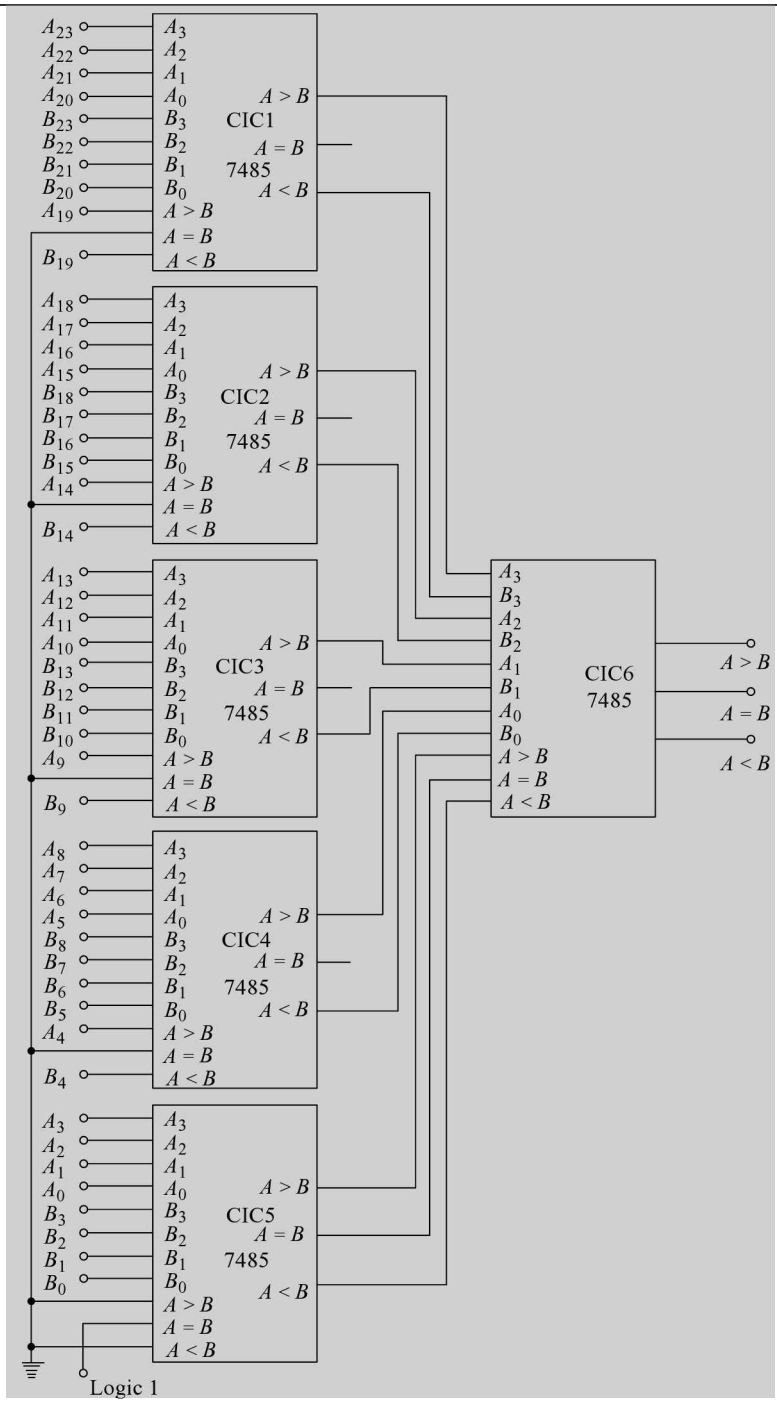


Fig. 6.24 ***A 24-bit Comparator***

6.8 PARITY GENERATORS/CHECKERS

The concept of parity, wherein an additional bit known as the *parity-bit* is added to a binary word to make the number of 1's, in the new word formed, even (*even parity*) or odd (*odd parity*), has been discussed in Section 2.10. The circuits for the generation of parity bits and checking the parity of a given word can be designed using gates (Problems 5.17 and 5.18). Because of its wide use, an 8-bit parity generator/checker circuit has been designed and is available as a MSI chip (74180).

Figure 6.25 gives the block diagram of 74180 in which there are eight parity inputs A through H and two cascading inputs. There are two outputs Σ EVEN and Σ ODD. Its function table is given in Table 6.10.

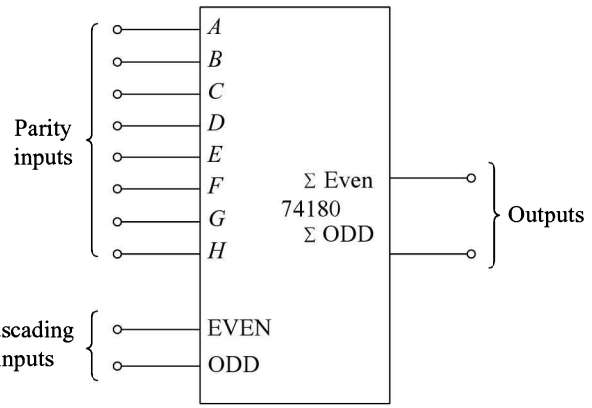


Fig. 6.25 ***Block Diagram of 74180 Parity Generator/Checker***

Table 6.10 ***Function Table of 74180***

Parity of inputs	Cascading inputs		Outputs	
A* through *H	***EVEN***	***ODD***	**Σ *EVEN***	**Σ *ODD***
EVEN	1	0	1	0
ODD	1	0	0	1
EVEN	0	1	0	1
ODD	0	1	1	0
X	1	1	0	0
X	0	0	1	1

The 74180 can be used as a parity generator as well as parity checker. Its function as a parity checker can be understood from the function table. Its function as a parity generator is given in Table 6.11.

Table 6.11 ***Function Table of 74180 As a 9-bit Parity Generator***

Parity of inputs *A* through *H*	Cascading inputs *EVEN*	Cascading inputs *ODD*	Parity of *A* through *H* and Σ *EVEN*	Parity of *A* through *H* and Σ *ODD*
ODD	1	0	ODD	EVEN
EVEN	1	0	ODD	EVEN
ODD	0	1	EVEN	ODD
EVEN	0	1	EVEN	ODD

The cascading inputs must not be equal and the unused parity inputs must be tied to logic 0 level.

Example 6.8

(a) Make a 9-bit odd parity checker using single 74180 and an inverter.
(b) Make a 10-bit even parity generator using single 74180 and an inverter.
(c) Make a 16-bit even parity checker using two 74180s.

Solution

(a) Eight of the nine bits are applied at *A–H* inputs and the ninth bit, *I*, is applied to the ODD input. Its operation can be verified with the help of Table 6.10. The circuit is given in Fig. 6.26*a*.

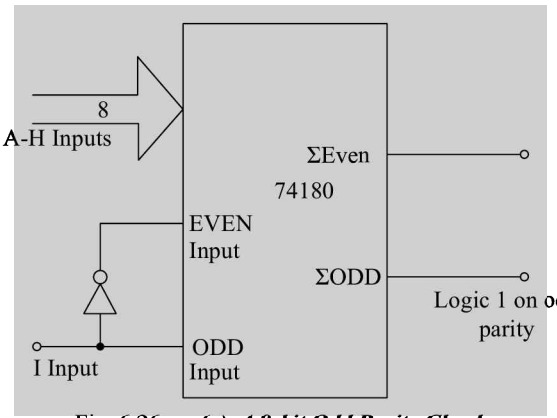
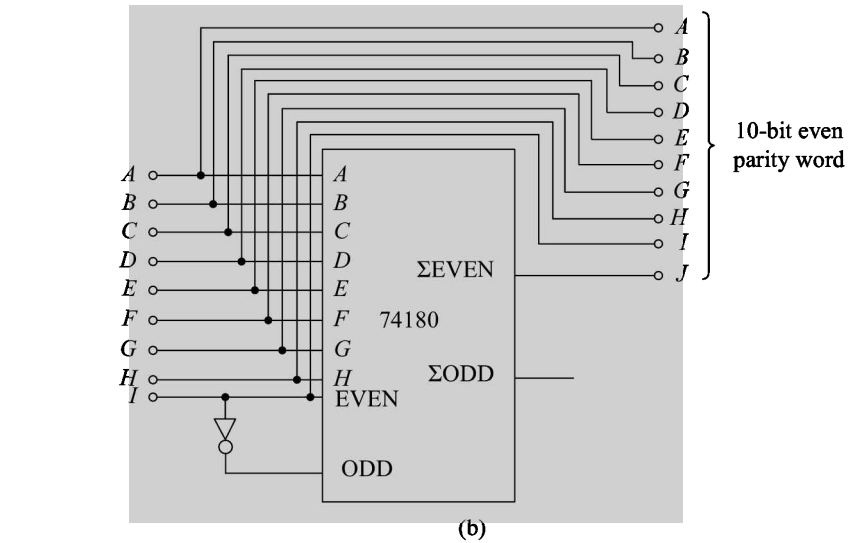
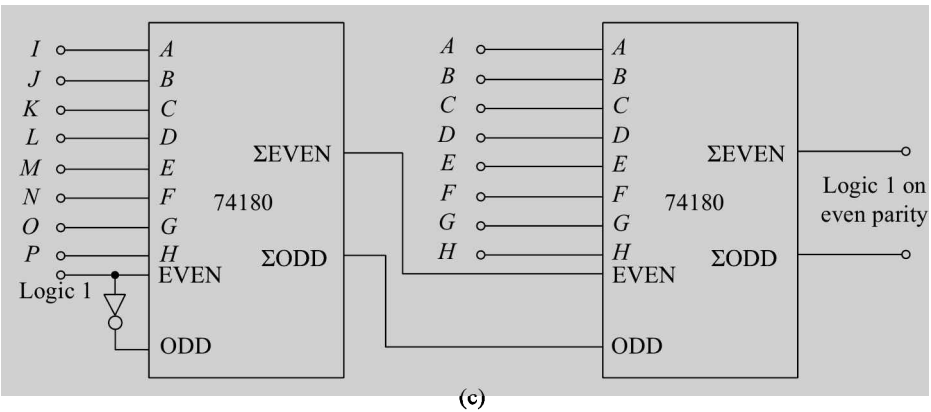


Fig. 6.26 ***(a) A 9-bit Odd Parity Checker***

(b) Here the 9-bit word consisting of *A* through *I* is converted to a 10-bit word with even parity. The circuit is given in Fig. 6.26*b*.
(c) The 16-bit even parity checker is shown in Fig. 6.26*c*.



(b)



(c)

Fig. 6.26 ***(b) A 10-bit Even Parity Generator***
(c) A 16-bit Even Parity Checker

6.9 CODE CONVERTERS

There is a wide variety of binary codes used in digital systems. Some of these codes are binary-coded-decimal (BCD), Excess-3, Gray, octal, hexadecimal, etc. Often, it is required to convert from one code to another. For example the input to a digital system may be in natural BCD and the output may be 7-segment LEDs. The digital system used may be capable of processing the data in straight binary format. Therefore, the data has to be converted from BCD to binary at the output.

The BCD output has to be converted to 7-segment code before it can be used to drive the LEDs. Similarly, octal and hexadecimal codes are widely used in microprocessors and digital computers as inputs and outputs. The various code converters can be designed using gates, multiplexers or demultiplexers. However, there are some MSI ICs available for performing these conversions and are extremely useful in the design of digital systems. These devices have been discussed below.

6.9.1 BCD-to-Binary Converter

The block diagram of BCD-to-binary converter IC 74184 is given in Fig. 6.27 and Table 6.12 gives its truth table. This device can be used as a $1\frac{1}{2}$ decade BCD-to-binary converter as shown in Fig. 6.28.

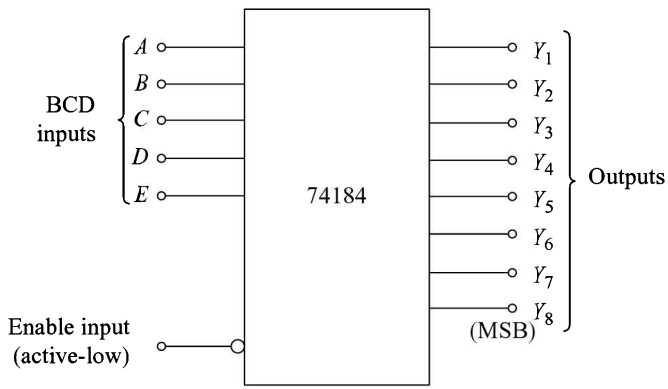


Fig. 6.27 ***Block Diagram of 74184.***

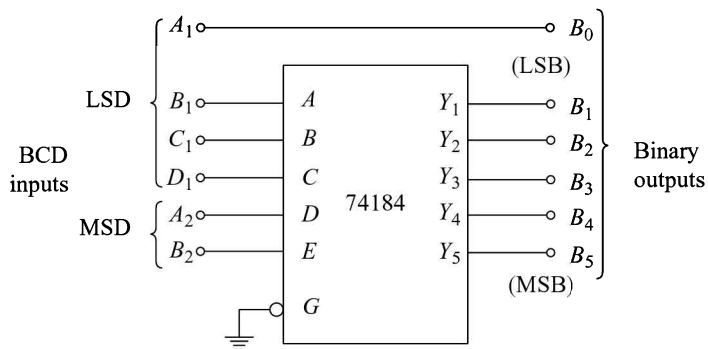


Fig. 6.28 *A* $1\frac{1}{2}$ ***Decade BCD-to-Binary Converter.***

The BCD inputs are applied at the input terminals A through E and the LSB of the least-significant BCD digit bypasses the converter and appears as the LSB of the binary output. It accepts two BCD digits–a full digit $D_1\ C_1\ B_1\ A_1$ and the two least-significant bits of a second digit $B_2\ A_2$. This means that the BCD inputs 00 through 39 can be converted to corresponding binary output by this circuit. Terminals Y_6, Y_7, and Y_8 are not used for BCD-to-binary conversion. These terminals are used to obtain the 9's complement and the 10's

complement of BCD numbers, useful for BCD arithmetic operations. Figure 6.29 gives the block diagram of BCD 9's complement converter and Table 6.13 gives its truth table.

Table 6.12 ***Truth Table of 74184 BCD-to-Binary Converter***

BCD	Inputs						Outputs				
words	*E*	*D*	*C*	*B*	*A*	*G*	Y_5	Y_4	Y_3	Y_2	Y_1
0 – 1	0	0	0	0	0	0	0	0	0	0	0
2 – 3	0	0	0	0	1	0	0	0	0	0	1
4 – 5	0	0	0	1	0	0	0	0	0	1	0
6 – 7	0	0	0	1	1	0	0	0	0	1	1
8 – 9	0	0	1	0	0	0	0	0	1	0	0
10 – 11	0	1	0	0	0	0	0	0	1	0	1
12 – 13	0	1	0	0	1	0	0	0	1	1	0
14 – 15	0	1	0	1	0	0	0	0	1	1	1
16 – 17	0	1	0	1	1	0	0	1	0	0	0
18 – 19	0	1	1	0	0	0	0	1	0	0	1
20 – 21	1	0	0	0	0	0	0	1	0	1	0
22 – 23	1	0	0	0	1	0	0	1	0	1	1
24 – 25	1	0	0	1	0	0	0	1	1	0	0
26 – 27	1	0	0	1	1	0	0	1	1	0	1
28 – 29	1	0	1	0	0	0	0	1	1	1	0
30 – 31	1	1	0	0	0	0	0	1	1	1	1
32 – 33	1	1	0	0	1	0	1	0	0	0	0
34 – 35	1	1	0	1	0	0	1	0	0	0	1
36 – 37	1	1	0	1	1	0	1	0	0	1	0
38 – 39	1	1	1	0	0	0	1	0	0	1	1
Any	×	×	×	×	×	1	1	1	1	1	1

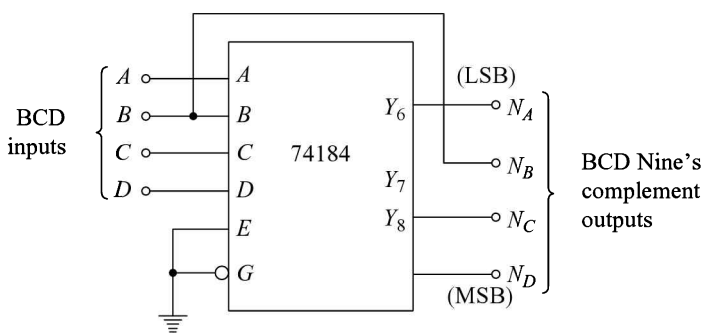


Fig. 6.29 ***Block Diagram of BCD 9's Complement Converter Using 74184***

Table 6.13 ***Truth Table of 74184 as BCD 9's Complement Converter***

BCD	Inputs						Outputs			*BCD* 9's complement			
word	*E*	*D*	*C*	*B*	*A*	*G*	Y_8	Y_7	Y_6	N_D	N_C	N_B	N_A
0	0	0	0	0	0	0	1	0	1	1	0	0	1
1	0	0	0	0	1	0	1	0	0	1	0	0	0
2	0	0	0	1	0	0	0	1	1	0	1	1	1
3	0	0	0	1	1	0	0	1	0	0	1	1	0
4	0	0	1	0	0	0	0	1	1	0	1	0	1
5	0	0	1	0	1	0	0	1	0	0	1	0	0
6	0	0	1	1	0	0	0	0	1	0	0	1	1
7	0	0	1	1	1	0	0	0	0	0	0	1	0
8	0	1	0	0	0	0	0	0	1	0	0	0	1
9	0	1	0	0	1	0	0	0	0	0	0	0	0
Any	×	×	×	×	×	1	1	1	1				

BCD input is applied at DCBA input terminals and its 9's complement appears at $N_D N_C N_B N_A$ terminals. Figure 6.30 gives the block diagram of BCD 10's complement converter and Table 6.14 gives its truth table.

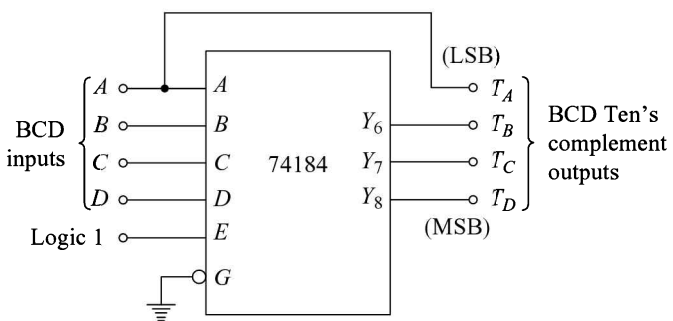


Fig. 6.30 ***Block Diagram of BCD 10's Complement Converter Using 74184***

Table 6.14 ***Truth Table of 74184 as BCD 10's Complement Converter***

BCD	Inputs						Outputs			*BCD* 10's complement			
word	*E*	*D*	*C*	*B*	*A*	*G*	Y_8	Y_7	Y_6	T_D	T_C	T_B	T_A
0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	1	0	1	0	0	1	0	0	1
2	1	0	0	1	0	0	1	0	0	1	0	0	0

(*Continued*)

Table 6.14 (*Continued*)

BCD word	Inputs						Outputs			*BCD* 10's complement			
	E	*D*	*C*	*B*	*A*	*G*	Y_8	Y_7	Y_6	T_D	T_C	T_B	T_A
3	1	0	0	1	1	0	0	1	1	0	1	1	1
4	1	0	1	0	0	0	0	1	1	0	1	1	0
5	1	0	1	0	1	0	0	1	0	0	1	0	1
6	1	0	1	1	0	0	0	1	0	0	1	0	0
7	1	0	1	1	1	0	0	0	1	0	0	1	1
8	1	1	0	0	0	0	0	0	1	0	0	1	0
9	1	1	0	0	1	0	0	0	0	0	0	0	1
Any	×	×	×	×	×	1	1	1	1				

The design of circuits for the conversion of 2 or more decades of BCD is quite involved. The circuit for 2-decade BCD-to-binary converter is given in Fig. 6.31. The number of 74184 ICs required increases tremendously as the number of input BCD decades increases. Table 6.15 gives the number of ICs required for a given number of input BCD decades. From the table we observe that the circuits become unmanageable for longer BCD words and it may be worthwhile to examine an alternative approach.

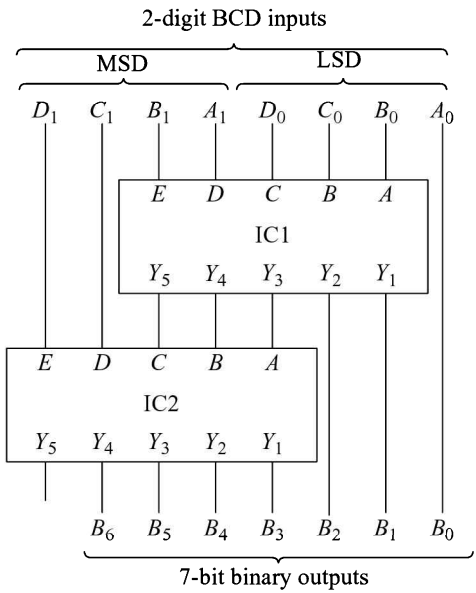


Fig. 6.31 ***A 2-Decade BCD-to-Binary Converter***

Table 6.15 ***Number of 74184 ICs Required***

Number of	
BCD decades	**74184 ICs required**
2	2
3	6
4	11
5	19
6	28

A read-only-memory (ROM) may be a convenient alternative, which will be discussed later.

Example 6.9

Verify the operation of a 2-decade BCD-to-binary converter of Fig. 6.31 for ar input of 29.

Solution

The input is

$D_1 C_1 B_1 A_1 D_0 C_0 B_0 A_0 = 00101001$

The binary output $B_0 = A_0 = 1$

The inputs of IC1 are
$E = B_1 = 1$
$D = A_1 = 0$
$C = D_0 = 1$
$B = C_0 = 0$
$A = B_0 = 0$

The outputs of IC1 are
$Y_5 = 0$
$Y_4 = 1$
$Y_3 = 1$
$Y_2 = 1$
$Y_1 = 0$

The binary outputs B_1 and B_2 are
$B_1 = Y_1 = 0$
$B_2 = Y_2 = 0$

The inputs of IC2 are
$E = D_1 = 0$
$D = C_1 = 0$
$C = Y_5$ of IC1 = 0
$B = Y_4$ of IC1 = 0
$A = Y_3$ of IC1 = 1

The outputs of IC2 are
$Y_5 = 0$
$Y_4 = 0$
$Y_3 = 0$
$Y_2 = 1$
$Y_1 = 1$

The binary outputs B_3, B_4, B_5, and B_6 are

$$B_3 = Y_1 = 1$$
$$B_4 = Y_2 = 1$$
$$B_5 = Y_3 = 0$$
$$B_6 = Y_4 = 0$$

Therefore the full binary output is 0011101.

6.9.2 Binary-to-BCD Converter

The block diagram of binary-to-BCD converter IC 74185A is given in Fig. 6.32 and Table 6.16 gives its truth table. It has the same pin out as the 74184. The binary inputs are applied at terminals A through E and the BCD outputs are available at terminals Y_1 through Y_6 (Y_7 and Y_8 are not used and these are always at logic 1). Similar to the BCD-to-binary converter, the least-significant bit bypass the circuit as shown in Fig. 6.32.

Table 6.16 ***Truth Table of 74185A Binary-to-BCD Converter***

Binary Words	Inputs						Outputs					
	E	*D*	*C*	*B*	*A*	*G*	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1
0 – 1	0	0	0	0	0	0	0	0	0	0	0	0
2 – 3	0	0	0	0	1	0	0	0	0	0	0	1
4 – 5	0	0	0	1	0	0	0	0	0	0	1	0
6 – 7	0	0	0	1	1	0	0	0	0	0	1	1
8 – 9	0	0	1	0	0	0	0	0	0	1	0	0
10 – 11	0	0	1	0	1	0	0	0	1	0	0	0
12 – 13	0	0	1	1	0	0	0	0	1	0	0	1
14 – 15	0	0	1	1	1	0	0	0	1	0	1	0
16 – 17	0	1	0	0	0	0	0	0	1	0	1	1
18 – 19	0	1	0	0	1	0	0	0	1	1	0	0
20 – 21	0	1	0	1	0	0	0	1	0	0	0	0
22 – 23	0	1	0	1	1	0	0	1	0	0	0	1
24 – 25	0	1	1	0	0	0	0	1	0	0	1	0
26 – 27	0	1	1	0	1	0	0	1	0	0	1	1
28 – 29	0	1	1	1	0	0	0	1	0	1	0	0
30 – 31	0	1	1	1	1	0	0	1	1	0	0	0
32 – 33	1	0	0	0	0	0	0	1	1	0	0	1
34 – 35	1	0	0	0	1	0	0	1	1	0	1	0

(*Continued*)

Table 6.16 (***Continued***)

Binary words	Inputs						Outputs					
	E	*D*	*C*	*B*	*A*	*G*	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1
36 – 37	1	0	0	1	0	0	0	1	1	0	1	1
38 – 39	1	0	0	1	1	0	0	1	1	1	0	0
40 – 41	1	0	1	0	0	0	1	0	0	0	0	0
42 – 43	1	0	1	0	1	0	1	0	0	0	0	1
44 – 45	1	0	1	1	0	0	1	0	0	0	1	0
46 – 47	1	0	1	1	1	0	1	0	0	0	1	1
48 – 49	1	1	0	0	0	0	1	0	0	1	0	0
50 – 51	1	1	0	0	1	0	1	0	1	0	0	0
52 – 53	1	1	0	1	0	0	1	0	1	0	0	1
54 – 55	1	1	0	1	1	0	1	0	1	0	1	0
56 – 57	1	1	1	0	0	0	1	0	1	0	1	1
58 – 59	1	1	1	0	1	0	1	0	1	1	0	0
60 – 61	1	1	1	1	0	0	1	1	0	0	0	0
62 – 63	1	1	1	1	1	0	1	1	0	0	0	1
All	×	×	×	×	×	1	1	1	1	1	1	1

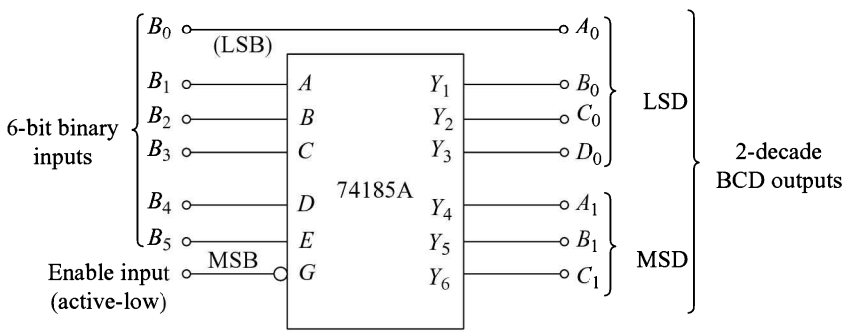


Fig. 6.32 ***A 6-Bit Binary-to-BCD Converter***

The method of expansion to accommodate larger number of binary inputs is quite complicated. The circuit for conversion of 8-bit binary number to BCD is given in Fig. 6.33. This requires three ICs. Table 6.17 gives the number of ICs required for a given number of input binary bits. Similar to the BCD-to-binary converters we may have to resort to ROM for conversion of longer binary words.

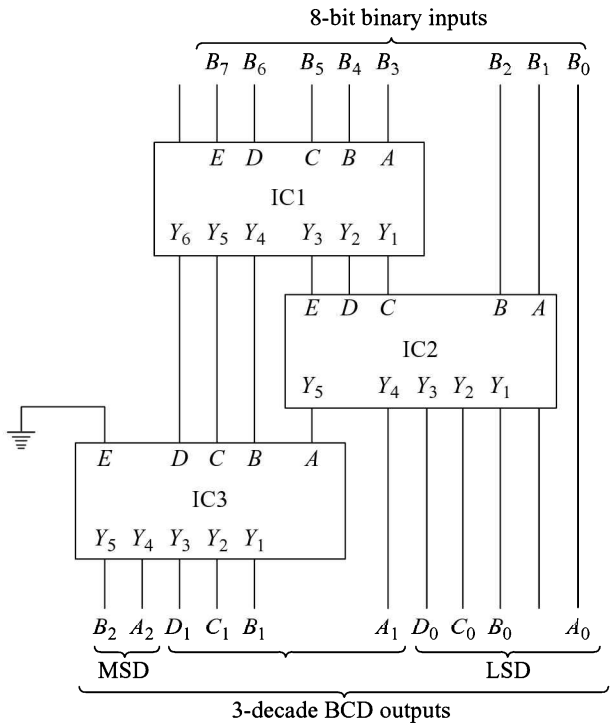


Fig. 6.33 ***An 8-bit Binary-to-BCD Converter***

Table 6.17 ***Number of 74185A ICs Required***

Number of	
Binary bits	**74185 *ICs* required**
4 – 6	1
7, 8	3
9	4
10	6
11	7
12	8
13	10
14	12
15	14
16	16

Example 6.10

Verify the operation of 8-bit binary-to BCD converter of Fig. 6.33 for the binary input of 11010011.

Solution

The inputs of IC1 are
$E = B_7 = 1$
$D = B_6 = 1$
$C = B_5 = 0$
$B = B_4 = 1$
$A = B_3 = 0$

The outputs of IC1 are
$Y_6 = 1$
$Y_5 = 0$
$Y_4 = 1$
$Y_3 = 0$
$Y_2 = 0$
$Y_1 = 1$

The inputs of IC2 are
$E = Y_3$ of IC1 = 0
$D = Y_2$ of IC1 = 0
$C = Y_1$ of IC1 = 1
$B = B_2 = 0$
$A = B_1 = 1$

The outputs of IC2 are
$Y_6 = 0$
$Y_5 = 0$
$Y_4 = 1$
$Y_3 = 0$
$Y_2 = 0$
$Y_1 = 0$

The inputs of IC3 are
$E = 0$
$D = Y_6$ of IC1 = 1
$C = Y_5$ of IC1 = 0
$B = Y_4$ of IC1 = 1
$A = Y_5$ of IC2 = 0

The outputs of IC3 are
$Y_6 = 0$
$Y_5 = 1$
$Y_4 = 0$
$Y_3 = 0$
$Y_2 = 0$
$Y_1 = 0$

The BCD outputs are
$B_2 = Y_5$ of IC3 = 1
$A_2 = Y_4$ of IC3 = 0
$D_1 = Y_3$ of IC3 = 0
$C_1 = Y_2$ of IC3 = 0
$B_1 = Y_1$ of IC3 = 0
$A_1 = Y_4$ of IC2 = 1
$D_0 = Y_3$ of IC2 = 0
$C_0 = Y_2$ of IC2 = 0

$B_0 = Y_1$ of IC2 = 0
$A_0 = B_0 = 1$
Therefore, the BCD output is 10 0001 0001 = 211.

6.10 PRIORITY ENCODERS

The encoder was introduced in Section 5.8.3 and a decimal-to-BCD encoder was designed using gates. The process of encoding is reverse of that of decoding. A number of priority encoder MSI ICs are available.

6.10.1 Decimal-to-BCD Encoder

One of the most commonly used input device for a digital system is a set of ten switches, one for each numeral between 0 and 9. These switches generate 1 or 0 logic levels in response to turning them OFF or ON. When a particular number is to be fed to the digital circuit in BCD code, the switch corresponding to that number is pressed. There is an IC available for performing this function (74147) which is a priority encoder. The block diagram of 74147 IC is given in Fig. 6.34 and Table 6.18 gives its truth table. It has active-low inputs and outputs. The meaning of the word priority can be seen from the truth table, for example, if inputs 2 and 5 are LOW, the output will be corresponding to 5 which has a higher priority than 2, i.e. the highest numbered input has priority over lower numbered inputs.

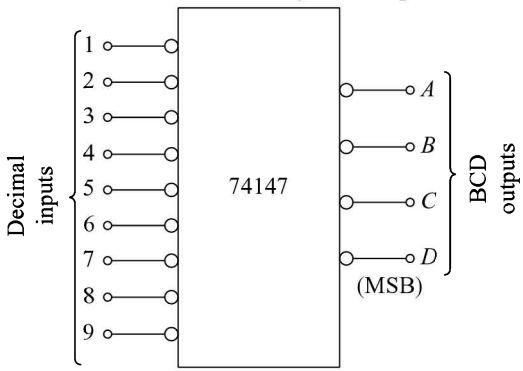


Fig. 6.34 ***Block Diagram of 74147 Decimal-to-BCD Priority Encoder***

Table 6.18 ***Truth Table of 74147***

Active-low decimal inputs									Active-low BCD outputs			
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	*D*	*C*	*B*	*A*
1	1	1	1	1	1	1	1	1	1	1	1	1
0	1	1	1	1	1	1	1	1	1	1	1	0
×	0	1	1	1	1	1	1	1	1	1	0	1
×	×	0	1	1	1	1	1	1	1	1	0	0
×	×	×	0	1	1	1	1	1	1	0	1	1
×	×	×	×	0	1	1	1	1	1	0	1	0
×	×	×	×	×	0	1	1	1	1	0	0	1
×	×	×	×	×	×	0	1	1	1	0	0	0
×	×	×	×	×	×	×	0	1	0	1	1	1
×	×	×	×	×	×	×	×	0	0	1	1	0

6.10.2 Octal-to-Binary Encoder

The octal code is often used at the inputs of digital circuits that require manual entering of long binary words. Priority encoder 74148 IC has been designed to achieve this operation. Its block diagram is given in Fig. 6.35 and Table 6.19 gives its truth table. This circuit also has active-low inputs and active-low outputs. The enable input and carry outputs, which are also active-low, are used to cascade circuits to handle more inputs. A hexadecimal-to-binary encoder, which is also a very useful circuit because of widespread use of hexadecimal code in computers, microprocessors, etc. can be designed using this facility.

The priority encoders can conveniently be used for handling priority interrupts in computers, microprocessors, etc.

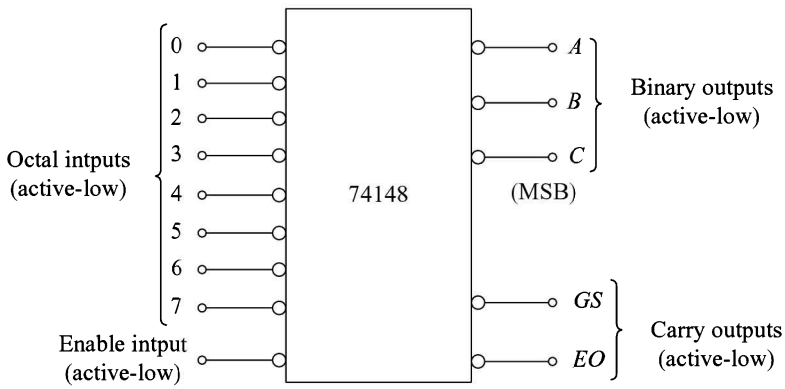


Fig. 6.35 ***Block Diagram of 74148 Octal-to-Binary Priority Encoder***

Table 6.19 ***Truth Table of 74148***

	Inputs								Outputs				
EI	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	***C***	***B***	***A***	***GS***	***EO***
1	X	X	X	X	X	X	X	X	1	1	1	1	1
0	0	1	1	1	1	1	1	1	1	1	1	0	1
0	X	0	1	1	1	1	1	1	1	1	0	0	1
0	X	X	0	1	1	1	1	1	1	0	1	0	1
0	X	X	X	0	1	1	1	1	1	0	0	0	1
0	X	X	X	X	0	1	1	1	0	1	1	0	1
0	X	X	X	X	X	0	1	1	0	1	0	0	1
0	X	X	X	X	X	X	0	1	0	0	1	0	1

(*Continued*)

Table 6.19 (*Continued*)

	Inputs								Outputs				
EI	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	***C***	***B***	***A***	***GS***	***EO***
0	X	X	X	X	X	X	X	0	0	0	0	0	1
0	1	1	1	1	1	1	1	1	1	1	1	1	0

Example 6.11

Design a hexadecimal-to-binary encoder using 74148 encoders and 74157 multiplexer.

Solution

Since there are sixteen symbols (0 – *F*) in hexadecimal number system, two 74148 encoders are required. Hexadecimal inputs 0 through 7 are applied to IC1 input lines and inputs 8 through *F* to IC2 input lines. Whenever one of the inputs of IC2 is active (LOW), IC1 must be disabled, on the other hand, if all the inputs of IC2 are HIGH then IC1 must be enabled. This is achieved by connecting the *EO* line of IC2 to the *EI* line of IC1. A quad 2:1 multiplexer is required to get the proper 4-bit binary outputs. The complete circuit is shown in Fig. 6.36. The *GS* output of 74148 goes LOW whenever one of its inputs is active. Therefore, *GS* of IC2 is connected to select input of 74157, which selects *A* inputs if it is LOW, otherwise *B* inputs are selected. The outputs of the multiplexer are the required binary outputs and are active-low. This circuit is also a priority encoder.

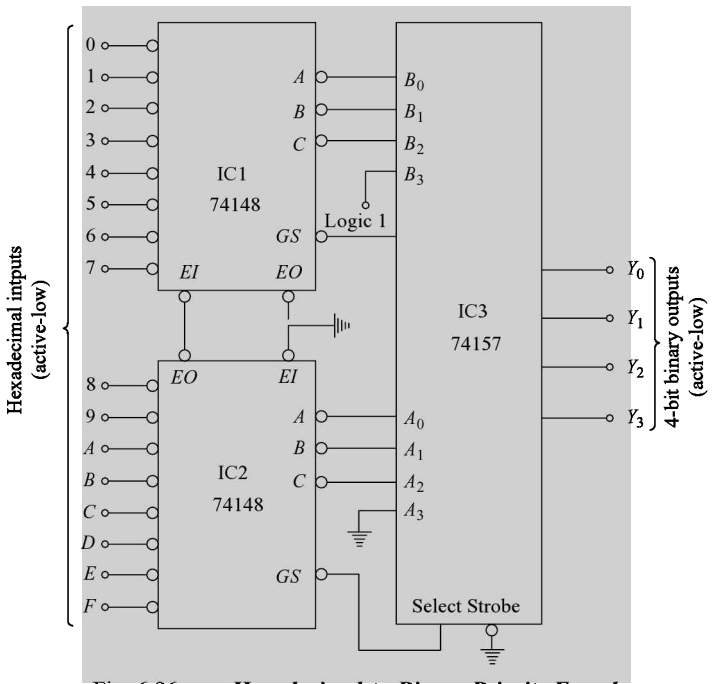


Fig. 6.36 ***Hexadecimal-to-Binary Priority Encoder***

6.11 DECODER/DRIVERS FOR DISPLAY DEVICES

6.11.1 BCD-to-Decimal Decoder/Driver

In many digital systems, we prefer to see the output in a decimal format. The outputs can either be displayed using display devices like LEDs, Nixie tubes, etc. or can be used to actuate some indicators or relays. Table 6.20 gives the available BCD-to-decimal decoder/driver ICs. All these ICs have active-high inputs and active-low outputs.

Table 6.20 ***Available BCD-to-decimal decoder/driver ICs***

IC No.	Output circuit	Application
7441	Open-collector	Nixie tube driver
7442	Totem-pole	LED driver
7445	Open-collector	Indicator/relay driver
74141	Open-collector	Nixie tube driver
74145	Open-collector	Indicator/relay driver
74445	Open-collector	Indicator/relay driver

6.11.2 BCD-to-7-Segment Decoder/Driver

Seven segment display is the most popular display device used in digital systems. For displaying data using this device, the data have to be converted from BCD to 7-segment code. A number of MSI ICs are available (Table 6.21) for performing this function. The decoder/driver circuit has 4 input lines for BCD data and 7 output lines to drive a 7-segment display. Output terminals *a* through *g* of the decoder are to be connected to *a* through *g* terminals of the display respectively. If the outputs are active-low, then the 7-segment LED must be of the common anode type, whereas if the outputs are active-high then the 7-segment LED must be of the common cathode type. The function of LT, RBI, RBO and BI are given in Table 6.22 and are explained below.

Table 6.21 ***Available BCD-to-7-Segment Decoder/Driver ICs***

IC No.	Output	Rating (Max voltage, Sink current)	Facilities available			
			Lamp Test LT	Ripple blanking Input RBI	Ripple blanking Output RBO	Blanking Input BI
7446, 74246	Active-low; Open-collector	30 V, 40 mA	Yes	Yes	Yes	Yes
7447, 74247	Active-low; Open-collector	15 V, 40 mA	Yes	Yes	Yes	Yes

(*Continued*)

Table 6.21 (*Continued*)

IC No.	Output	Rating (Max voltage, Sink current)	Facilities available: Lamp Test LT	Ripple blanking Input RBI	Ripple blanking Output RBO	Blanking Input BI
7448, 74248	Active-high; Pull-up resistor = 2 kΩ	5.5 V, 6.4 mA	Yes	Yes	Yes	Yes
7449, 74249	Active-high; Open-collector	5.5 V, 8 mA	No	No	No	Yes

Table 6.22 ***Summary of BCD-to-7-Segment Decoder Functions***

LT	RBI	BI/RBO	BCD inputs	Display mode
0	X	1 output	X	Lamp test
X	X	0 input	X	Display blank
1	1	1 output	Any number	Normal decoding
1	0	Normally at logic 1 output. Goes to logic 0 during zero blanking interval	Any number	Normal decoding with zero blanking

LT This is used to check the segments of LED. If it is connected to logic 0 level, all the segments of the display connected to the decoder will be ON. For normal decoding operation, this terminal is to be connected to logic 1 level.

RBI It is to be connected to logic 1 for normal decoding operation. If it is connected to 0 level, the segment outputs will generate data for normal 7-segment decoding for all BCD inputs except zero. Whenever the BCD inputs correspond to zero, the 7-segment display switches off. This is used for blanking out leading zeros in multi-digit displays.

BI If it is connected to logic 0 level, the display is switched off irrespective of BCD inputs. This is used for conserving power in multiplexed displays.

RBO This output, which is normally at logic 1 goes to logic 0 during zero blanking interval. This is used for cascading purposes and is connected to RBI of the succeeding stage.

Example 6.12

(a) Set up a single 7-segment LED display using 7447 BCD-to-7-segment decoder/driver.
(b) Design a 4-digit 7-segment LED display system with leading zero blanking.
(c) Design a 4-digit multiplexed 7-segment LED display system with leading-zero blanking.

Solution

(a) The set up is given in Fig. 6.37.

(b) A 4-digit display system can display numbers from 0000 to 9999. If the number to be displayed is smaller than a 4-digit number then there is a problem of leading zeros. It is very often desirable to design a display system so that any number less than 1000 will not show leading zeros. For example, the number 0203 should appear as 203. The block diagram of a 4-digit display system with leading zero blanking is shown in Fig. 6.38. The reader should verify the operation of this circuit.

(c) When a multi-digit, 7-segment LED display system is used, it requires power which is n-times the power required for a single 7-segment LED, where *n* is the number of such LEDs in the system. Typically, the current requirement of one segment is 20 mA at full brightness which makes the maximum current requirement for a single 7-segment LED to be 140 mA (corresponding to zero). If there are four such LEDs in the display system, then the maximum current required would be 560 mA. It is possible to reduce the overall current drain of the display system by using the concept of multiplexing the display, i.e. to energize the output digits one at a time. If this is done in rapid succession, the display would appear to be continuous to the human eyes and the overall current drain is equal to that of a single 7-segment display. Figure 6.39 gives the block diagram of a 4-digit multiplexed 7-segment LED display system. Here, a counter (to be discussed later) drives the BCD-to-decimal decoder 7442 and the output of the decoder through an inverter pulls up the common anode terminal to HIGH. Each anode is excited in sequence at the rate determined by the clock frequency. Multi-digit 7-segment display assemblies which can be used only in the multiplexed mode are also commercially available.

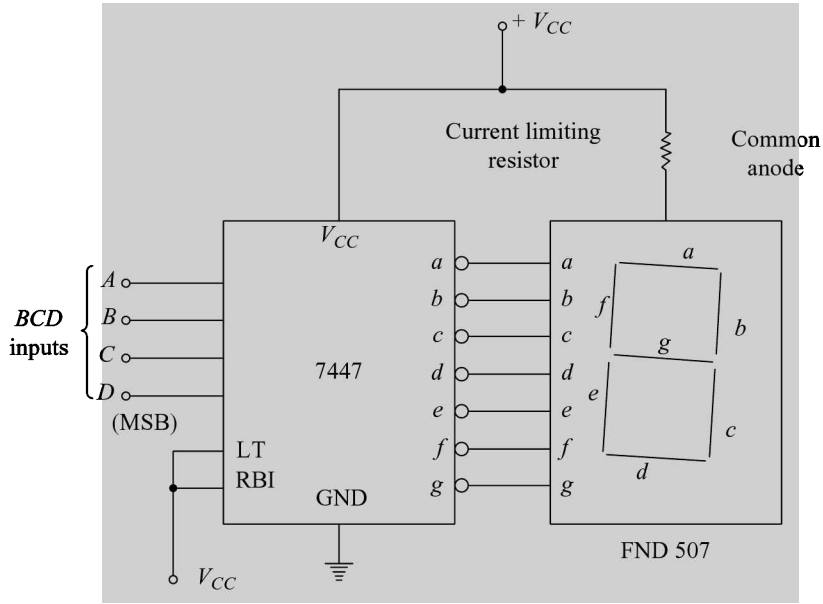


Fig. 6.37 ***A 7447 Driving a 7-Segment LED Display***

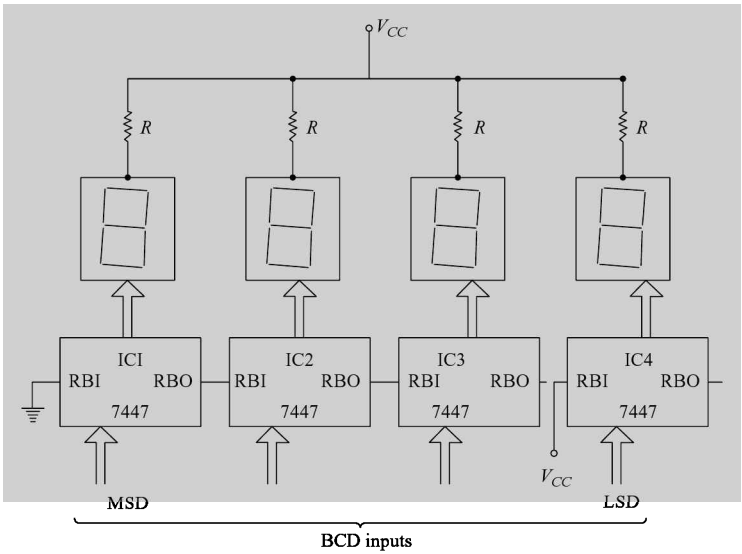


Fig. 6.38 ***A 4-digit LED Display System with Leading Zero Blanking***

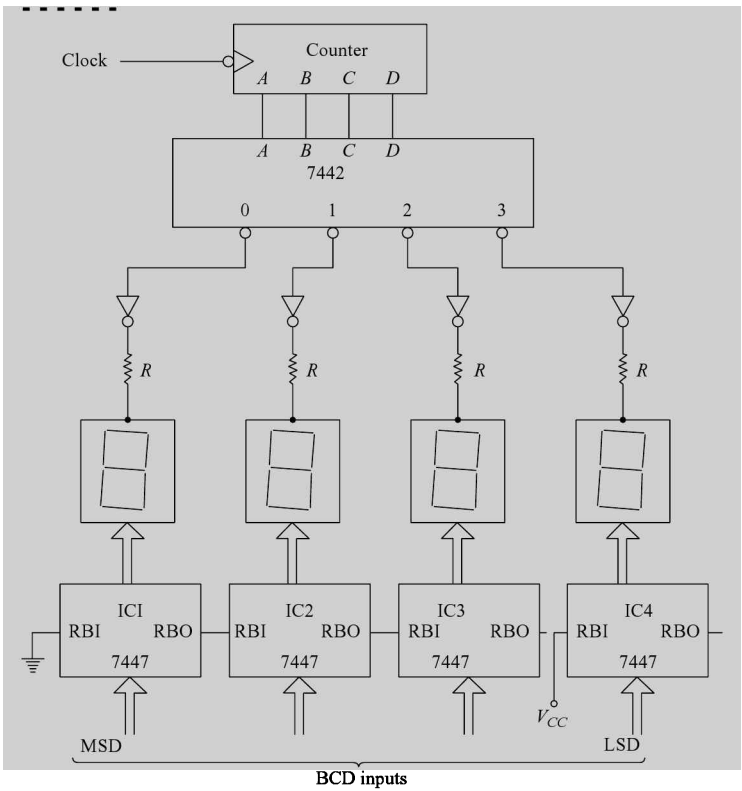


Fig. 6.39 ***A 4-digit Multiplexed Display System with Leading Zero Blanking***

SUMMARY

The most popular and commonly available MSI ICs and their applications have been discussed in this chapter. Since the logic equations for these circuits are very complex, therefore, their applicability must be recognised at the system level. Quite often, the system specifications are affected by the initial knowledge of available standard circuits. Each technique discussed can be thought of as a tool and each tool has its place. The designer has to make a proper choice of the tool for the job. Though more than one tool may work for a given job, the key is to select the right one. Although system design has been oriented around making use of higher levels of integration, a lot of little jobs of interfacing the MSI devices are still best done with discrete gates.

Active-low inputs and outputs have been indicated by small circles throughout. Recently, many authors have started using an alternative symbol (small right triangle) to represent the active-low input/output to avoid confusion (of inversion) associated with the small circle. Figure 6.35 is redrawn in Fig. 6.40 to indicate this new system.

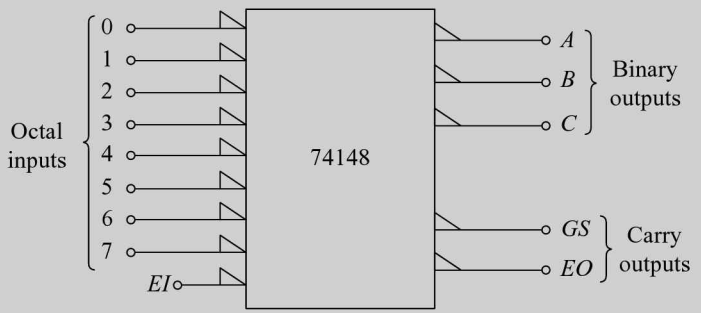


Fig. 6.40 ***An Alternative Representation of Active-Low Input/Output***

GLOSSARY

Arithmetic-logic unit (ALU) A logic circuit that performs arithmetic and logical operations.

Comparator A logic circuit that compares two binary numbers and produces an output indicating whether they are equal or which is greater if they are unequal.

Demultiplexer A logic circuit that connects a single input line to one of the several output lines.

Dot matrix display A display device consisting of elements in the form of dots arranged in a matrix form. It is used to display alphanumeric and special characters.

Driver A circuit capable of supplying high load current.

Hardware The physical parts forming a system.

Look-ahead carry A method of generating carry for fast addition.

Multiplexed display A display system used in multiplexed mode.

Priorty encoder An encoder that responds to the highest number when two or more numbers are applied simultaneously.

Strobe An input signal to a digital circuit that can enable or disable the circuit.

Subtractor A logic circuit used for subtraction.

Ten's complement It is 9's complement of a decimal number plus one.

REVIEW QUESTIONS

6.1 A logic circuit that gates one out of several inputs to single output is known as a _____ .
6.2 A _____ is a logic circuit that accepts one data input and distributes it over several outputs.
6.3 The minimum number of selection inputs required for selecting one out of 32 inputs is _____ .
6.4 A logic circuit required for converting BCD code to 7-segment code is known as _____ .
6.5 In a BCD adder _____ is added in case the sum is not valid BCD number.
6.6 A logic circuit used to compare binary numbers is _____ .
6.7 _____ is used in display systems to save_____ .
6.8 A 4-variable logic expression can be realised using single _____ multiplexer.
6.9 A decoder with 64 output lines has _____ data inputs.
6.10 ALU stands for _____ .
6.11 A bubble at the output of a multiplexer indicates _____ output.
6.12 Chip select input of a digital IC has _____ symbol. This indicates _____ .
6.13 A 4-digit display system with leading zero blanking displays 47 as _____ .
6.14 _____ complementation is used for performing BCD subtraction.
6.15 Subtractors are designed using _____ ICs.

PROBLEMS

6.1 Realise the logic function of table 6.3 using
(a) A 16:1 multiplexer IC 74150, and
(b) An 8:1 multiplexer IC 74152.
6.2 Design a 32:1 multiplexer using two 16:1 multiplexer ICs
6.3 Design a full adder using 8:1 multiplexer ICs. Compare the IC package count with the NAND-NAND realisation.
6.4 Design a 4-bit ADDER/SUBTRACTOR circuit with ADD/SUB control line.
6.5 Design a Gray-to-BCD code converter using
(a) Two dual 4:1 multiplexer ICs (74153) and some gates, and
(b) One 1:16 demultiplexer IC 74154 and NAND gates.

6.6 Design a BCD-to-7-segment decoder with active-low outputs using
(a) Dual 4:1 multiplexers and some gates,
(b) 1:16 demultiplexer and some gates, and
(c) A BCD-to-decimal decoder and NAND gates.
(d) Compare the IC package count.

6.7 Design a BCD-to-Gray code converter using
(a) 8:1 multiplexers,
(b) Dual 4:1 multiplexers and some gates,
(c) Quad 2:1 multiplexers and some gates,
(d) A BCD-to-decimal decoder and NAND gates, and
(e) NAND gates only.
(f) Compare package count for the various approaches.

6.8 Realise the following functions of four variables using
(a) 8:1 multiplexers,
(b) 16:1 multiplexers, and
(c) 4-to-16-line decoder with active-low outputs.
(i) $f_1 = \sum m(0, 3, 5, 6, 9, 10, 12, 15)$
(ii) $f_2 = \sum m(0, 1, 2, 3, 11, 12, 14, 15)$
(iii) $f_3 = \Pi M(0, 1, 3, 7, 9, 10, 11, 13, 14, 15)$

6.9 Design a 40:1 multiplexer using 8:1 multiplexers.

6.10 Design a 1:40 demultiplexer using BCD-to-decimal decoders.

6.11 In the n-bit binary adder shown in Fig. 6.12*a*, determine the time required for the final carry C_{n-1} to reach steady state. Assume each full-adder to be consisting of half-adder circuits (Prob. 5.19(b)) and propagation delays of gates as given in Prob. 5.20.

6.12 Design a 4-digit BCD adder using 7483 adders.

6.13 Design a 2-bit comparator using gates.

6.14 Design an 8-bit comparator using only two 7485s.

6.15 Verify the operation of the 24-bit comparator of Fig. 6.24 for the following numbers:
$A = 100110000111011001010010$
$B = 101110000111011000100011$

6.16 Design a one digit-BCD-to-binary converter using 74184.

6.17 Show that the hexadecimal-to-binary encoder of Fig. 6.36 is a priority encoder.

6.18 Design a 6-bit odd/even parity checker using 74180.

6.19 Design a parity generator circuit using 74180 to add an odd parity bit to a 7-bit word.

6.20 Design a parity generator circuit to add an even parity bit to a 14-bit word. Use two 74180 packages.

6.21 Design a 10-bit parity checker using one 74180 and an EX-OR gate (7486).

6.22 Design a two level parity checker tree using ten packages of 74180 to check the parity of 81 bits.

6.23 Design a two level parity checker tree using three packages of 74180 to check the parity of 25 bits.

6.24 Design the following circuits:
(a) 7442 BCD-to-decimal decoder driving ten LEDs.
(b) 74141 BCD-to-decimal decoder driving a Nixie tube

6.25 Suggest a suitable alternative circuit for Fig. 6.11 which does not require an OR gate.

6.26 Consider the four digit 7-segment display system shown in Fig. 6.41. Modify the circuit of Fig. 6.39 if this display system is to be used.

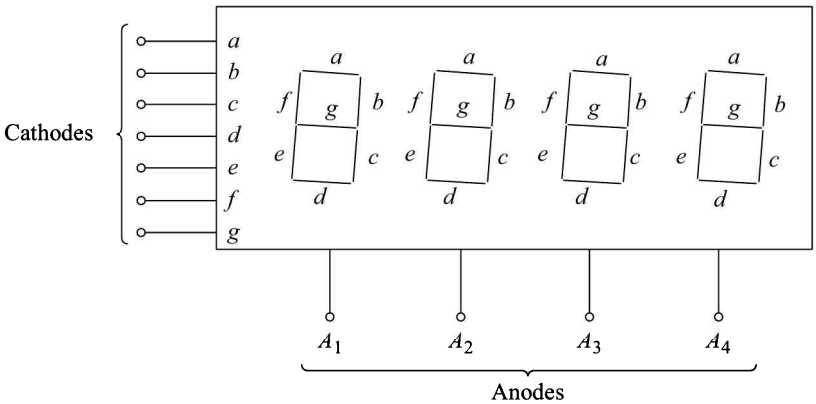


Fig. 6.41 ***A 4-digit 7-Segment Display System***

6.27 A commonly used display system known as 7 × 5 dot matrix is shown in Fig. 6.42*a*. This is used to display alphanumeric characters and some special symbols. It is desired to display *R* using this display system whose pattern of glowing of the dots is shown in Fig. 6.42*b*. Design a suitable circuit for this.

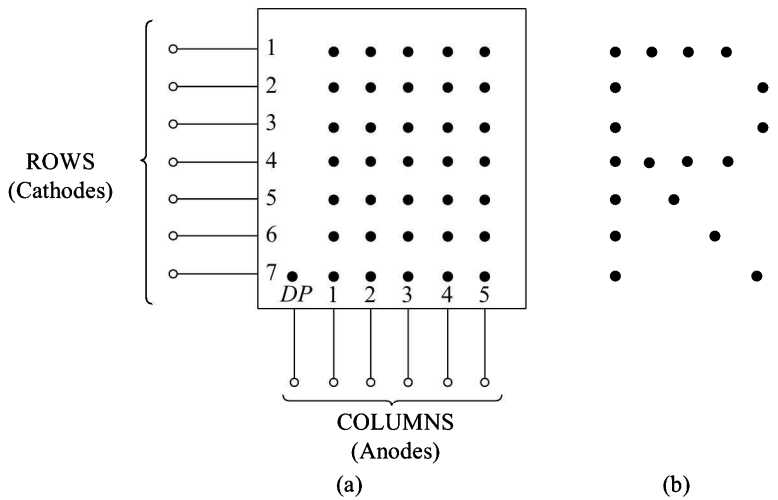


Fig. 6.42 ***A 7 × 5 Dot Matrix Display***