UNIT-4

DIGITAL ELECTRONICS

NUMBER SYSTEMS

The various number systems are as follows

i) Decimal number system

ii) Binary number system

iii) Octal Number system

iv) Hexadecimal number system.

DECIMAL NUMBER SYSTEM

* Decimal number system has base '10'
* The number starts from 0 to 9 [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

BINARY NUMBER SYSTEM

* Binary number system has base '2'
* It composed of only two digits 0 and 1

OCTAL NUMBER SYSTEM

* Octal number system has base '8'
* The octal number system uses the digits 0, 1, 2, 3, 4, 5, 6 & 7

HEXADECIMAL NUMBER SYSTEM

* Hexadecimal number system has base '16'
* The number includes 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
* A, B, C, D, E, F represents the decimals 10, 11, 12, 13, 14, 15. respectively

DECIMAL TO BINARY CONVERSION

PROCEDURE

* Given decimal number is divided by 2 progressively.
* When each division is performed, the remainder either '0' or '1' should write in side.
* The process ends when the quotient becomes zero.

Eg: 1

2 | 52	
2 | 26	- 0
2 | 13	- 0
2 | 6	- 1
2 | 3	- 0
1	- 1

$(110100)_2$

* Writing the remainders after each division in the reverse order, the binary number is obtained.
* This method is called as double - dabble method

Eg : 2

$(76.36)_{10} \rightarrow (\quad)_2$

2 | 76	
2 | 38	- 0
2 | 19	- 0
2 | 9	- 1
2 | 4	- 1
2 | 2	- 0
1	- 0

(1001100)

		Carry
.36 x 2	= 0.72	: 0
.72 x 2	= 1.44	: 1
.44 x 2	= 0.88	: 0
.88 x 2	= 1.76	: 1
.76 x 2	= 1.52	: 1
0.52 x 2	= 1.04	: 1
.04 x 2	= 0.08	: 0

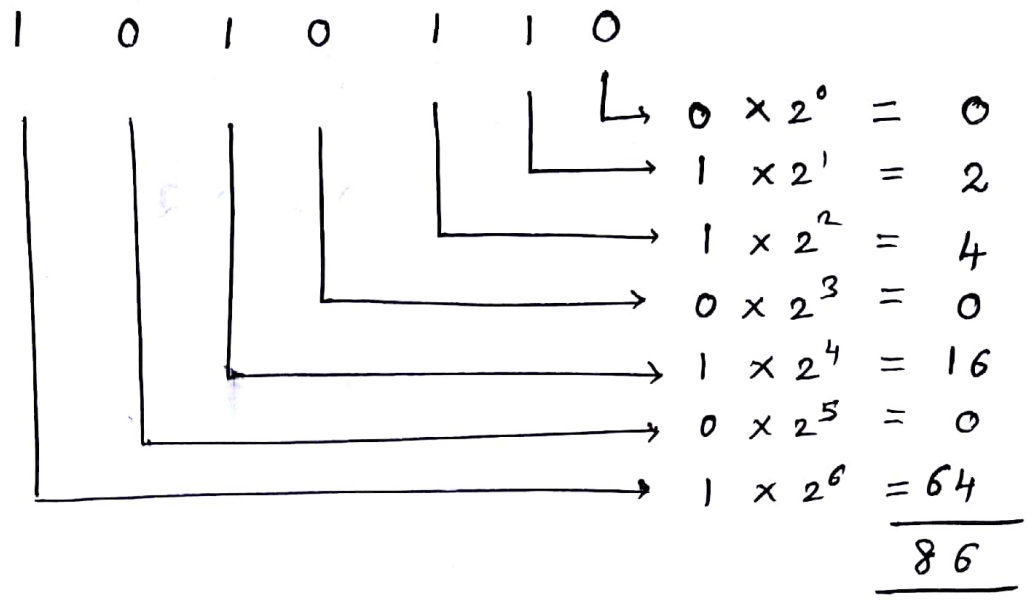
(0101110)

$(76.36)_{10} \rightarrow (1001100.0101110)_2$

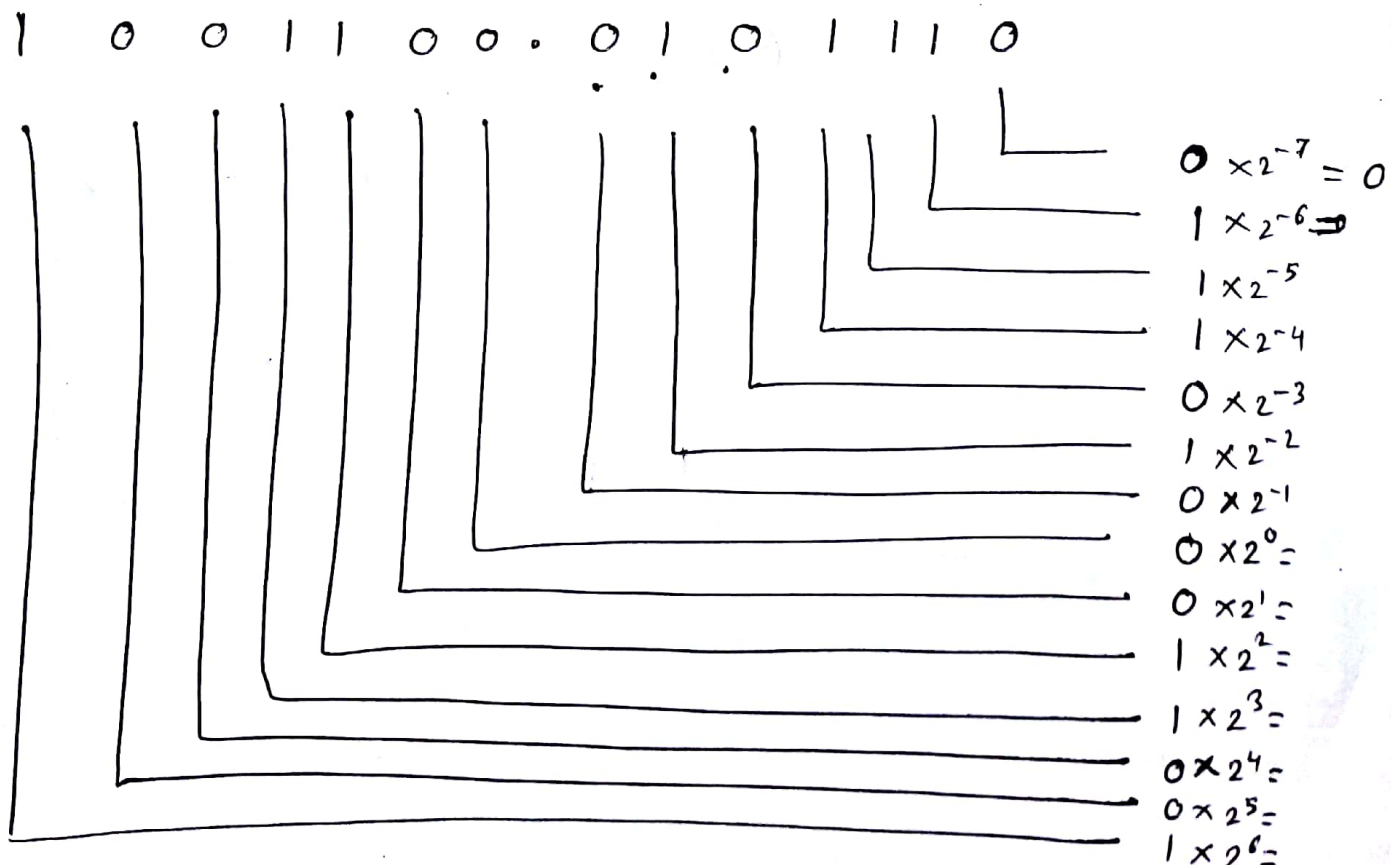
* Right hand side of decimal point is taken and multiplied by 2.
* From the obtained value the carry is written separately.
* The process continues till the product is zero
* If zero not obtained do upto 5 iteration.

Binary To Decimal Conversion

Convert the binary number 1010110_2 to the decimal equivalent



Convert the binary number $(1001100.0101110)_2$ into decimal equivalent.



$$
\begin{aligned}
&= \quad 0 \\
&\quad\;\; 0.015625 \\
&\quad\;\; 0.03125 \\
&\quad\;\; 0.0625 \\
&\quad\;\; 0.25 \\
&\quad\;\; 0 \\
&\quad\;\; 0 \\
&\quad\;\; 0 \\
&\quad 4.0 \\
&\quad 8.0 \\
&\quad\;\; 0 \\
&\quad\;\; 0 \\
&\quad 64 \\
&\hline
&\quad 76.359375
\end{aligned}
$$

OCTAL to BINARY SYSTEM

1) Convert $(634)_8$ to binary

6 3 4

(110) (011) (100)

Equivalent binary $(110011100)_2$

2) Convert $(725.63)_8$ to binary

7 2 5 . 6 3

111 010 101 . 110 011

Binary equivalent $[111010101.110011]_2$

BINARY To OCTAL SYSTEM

$[1110\ 1110]_2 \longrightarrow$ to equivalent octal.

0 1 1 | 1 0 1 | 1 1 0

3 5 6

$\Downarrow$

$(356)_8$

OCTAL to DECIMAL

$(567)_8$ in to decimal equivalent

5 6 7

$7 \times 8^0 = 7$

$6 \times 8^1 = 48$

$5 \times 8^2 = 320$

375

$(235.56)_8$ in to decimal equivalent

2 35 . 56

$6 \times 8^{-2} = 0.09375$

$5 \times 8^{-1} = 0.625$

$5 \times 8^0 = 5$

$3 \times 8^1 = 24$

$2 \times 8^2 = 128$

$[157.71875]_{10}$

DECIMAL TO OCTAL

$(489)_{10}$ to octal

8 | 489

8 | 61 - 1

7 - 5

$(751)_8$

Convert $(157.71875)_{10}$ to Octal.

$8 \mid 157$

$8 \mid 19 - 5$

$2 - 3$

$0.71875 \times 8 = 5.75 : 5$ ↓

$0.75 \times 8 = 6.0 : 6$ ↓

$$[235.56]_8$$

HEXADECIMAL TO BINARY SYSTEM

Convert $[8D]_{16}$ to binary

	8	D
	↓	(13)
Equivalent binary →	1000	1101

$\Rightarrow (10001101)_2$

BINARY TO HEXADECIMAL

Convert $[1110\ 1110]_2$ to Hexadecimal

1110	1110	
14	14	→ Equivalent decimal.
↓	↓	
E	E	

$$[EE]_{16}$$

HEXADECIMAL To DECIMAL

Convert $(5F)_{16}$ to decimal.

5 F

$15 \times 16^0 = 15$

$5 \times 16^1 = 80$

95

$[95]_{10}$

Convert $(7F.8E)_{16}$ to decimal.

7 F . 8 E

$14 \times 16^{-2} = 0.0546875$

$8 \times 16^{-1} = 0.5$

$15 \times 16^0 = 15$

$7 \times 16^1 = 112$

127.5546875

DECIMAL To HEXADECIMAL

Convert $(189)_{10}$ to Hex

16 | 189

11 - 13 ↑

$11 \rightarrow B$

$13 \rightarrow D$

$[BD]_{16}$

Convert $[127.5546875]_{10}$ to Hex

$16 \mid 127$

$7 - 15$

7 15

↓ ↓

7 F

$0.5546875 \times 16 = 8.875 \;:\; 8$

$0.875 \times 16 = 14.0 \;:\; 14\,(E)$ ↓

8 E

$[7F.8E]_{16}$

OCTAL TO HEXA DECIMAL

Convert $(675)_8$ to Hex

6 7 5

110 111 101

⇓

000 | 101 | 111 | 101

1 11 13

↓ ↓ ↓

1 B D

$[1BD]_{16}$

HEXA DECIMAL TO OCTAL

Convert $(AF5)_{16}$ to Octal

A F 5

⇓ ⇓ ⇓

1010 1111 0101

⇓

101 | 011 | 110 | 101

5 3 6 5

$[5365]_8$

BINARY ARITHMETICS

BINARY ADDITION

RULES

A	B	SUM	CARRY
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

EXAMPLE

Add $(1010)_2$ and $(0011)_2$

SOLUTION

$$\begin{array}{llll} 1\ 0\ 1\ 0 & \rightarrow & 10 \\ 0\ 0\ 1\ 1 & \rightarrow & 03 \\ \hline 1\ 1\ 0\ 1 & & 13 \end{array}$$

Add $(28)_{10}$ and $(15)_{10}$ with binary equivalent

$$\begin{array}{lll} & \overset{1}{} \quad \overset{1}{} & \\ (28)_{10} \rightarrow & 1\ 1\ 1\ 0\ 0 & \\ (15)_{10} \rightarrow & 0\ 1\ 1\ 1\ 1 & (+) \\ \hline (43)_{10} & 10\ 1\ 0\ 1\ 1 & \end{array}$$

$$\begin{array}{r|l} 2 & 43 \\ 2 & 21-1 \\ 2 & 10-1 \\ 2 & 5-0 \\ 2 & 2-1 \\ & 1-0 \end{array} \qquad \begin{array}{r|l} 2 & 28 \\ 2 & 14-0 \\ 2 & 7-0 \\ 2 & 3-1 \\ & 1-1 \end{array} \qquad \begin{array}{r|l} 2 & 15 \\ 2 & 7-1 \\ 2 & 3-1 \\ & 1-1 \end{array}$$

$$(43)_{10} \Rightarrow (1\ 0\ 1\ 0\ 1\ 1)_2$$

BINARY SUBTRACTION

RULES

A	B	Diff	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

EXAMPLE:

Subtract $(0101)_2$ from $(1011)_2$

1 → borrow

$\not{1}\ \ 0\ 1\ 1$ → Decimal 11

$0\ 1\ 0\ 1$ → 5

$1\ 1\ 0$ → 6

Subtract $(1100001)_2$ from $(1011101)_2$

$\overset{0}{\not{1}}\ \overset{10}{\not{1}}\ 0\ 1\ 1\ 1\ 0\ 1$ → 109

$0\ 1\ 1\ 0\ 0\ 0\ 1$ → 049

$(1\ 1\ 1\ 1\ 0\ 0)_2$ → 060

Subtract 01111001 from 11000010

$$\begin{array}{rl} 11000010 & \longrightarrow 194 \\ 01111001 & \longrightarrow 121 \\ \hline [1001001]_2 & \Rightarrow 73 \end{array}$$

BINARY MULTIPLICATION

9 x 6

$9 \Rightarrow 1001$

$6 \Rightarrow 0110$ ×

$$\begin{array}{r} 0000 \\ 1001\times \\ 1001\times\times \\ 0000\times\times\times \\ \hline 110110 \end{array} \Rightarrow 54$$

Multiply 1011.01 × 110.1

$$\begin{array}{r} 1011.01 \\ 110.1 \quad \times \\ \hline 101101 \\ 0000000\times \\ 1011 01\times\times \\ 1011 01\times\times\times \\ \hline 100 1001.001 \end{array}$$

1's Complement Representation

Find 1's Complement of $(1101)_2$ and $(10111001)_2$

Solution

1101 ← Binary number

0010 ← 1's Complement

10111001 ← Binary number

01000110 ← 1's Complement

2's Complement of $(1001)_2$ and $(10100011)_2$

* 1's Complement + 1 will give 2's Complement

Solution

1001

0110 1's Complement

\+ 1

0111 ← 2's Complement

10100011

01011100 1's Complement

\+ 1

01011101 ← 2's Complement

1's Complement Subtraction

CASE: 1

Subtraction of smaller number from larger number

Method

1. Determine the 1's Complement of the smaller number
2. Add the 1's Complement to the larger number
3. Remove the carry and add it to the result.

Problem: subtract 101011_2 from 111001 using 1's complement method.

```
   1
   111001     → larger number
   010100     → 1's Complement of smaller number
  --------
[1] 001101
 └───+   1    → Add the carry
  --------
   001110
  ========
```

CASE: 2

Subtraction of larger number from Smaller number.

Method

1. Determine the 1's Complement of larger number.
2. Add the 1's Complement to the smaller number.
3. The answer is in ones complement form to get the true value find 1's complement again.

Example:

Subtract 111001 from 101011 using 1's Complement method.

```
  1 1 1
  1 0 1 0 1 1   → Smaller number
  0 0 0 1 1 0   → 1's Complement of larger number
---------------
  1 1 0 0 0 1
---------------  ⇩⟹ Finding 1's Complement
- 0 0 1 1 1 0
---------------
```

2's Complement Subtraction

CASE: 1 Subtraction of smaller number from larger number.

Method : 1) Determine the 2's complement of the smaller number.

2) Add the 2's complement to the larger number

3) Discard the carry.

EXAMPLE: Subtract 101011_2 from 111001_2 using 2's Complement

```
  1     0
  1 1 1 0 0 1   → Smaller number
  0 1 0 1 0 1   → 2's Complement of larger number
---------------
1 0 0 1 1 1 0
```

discard the Carry

001110 //

2's Complement of 101011

(1) 010100

+ 1

010101

CASE: 2 Subtraction of larger number from smaller number using 2's Complement.

Method: 1) Determine 2's complement of larger number

2) Add the 2's Complement to the smaller number

3) The answer is in 2's Complement form, to get the true value find 2's complement again.

Example Subtract 111001_2 from 101011_2 using 2's Complement method.

```
  1 1 1 1
  1 0 1 0 1 1   → smaller number
  0 0 0 1 1 1   → 2's Complement
-------------     of larger number
  1 1 0 0 1 0
-------------
```

↳ Find 2's Complement

```
      1
  0 0 1 1 0 1
        1  +
-------------
  0 0 1 1 1 0
-------------
```

↳ 001110 //

2's complement of larger number

```
  1 1 1 0 0 1
      ↓
  0 0 0 1 1 0
        1  +
-------------
  0 0 0 1 1 1
-------------
```

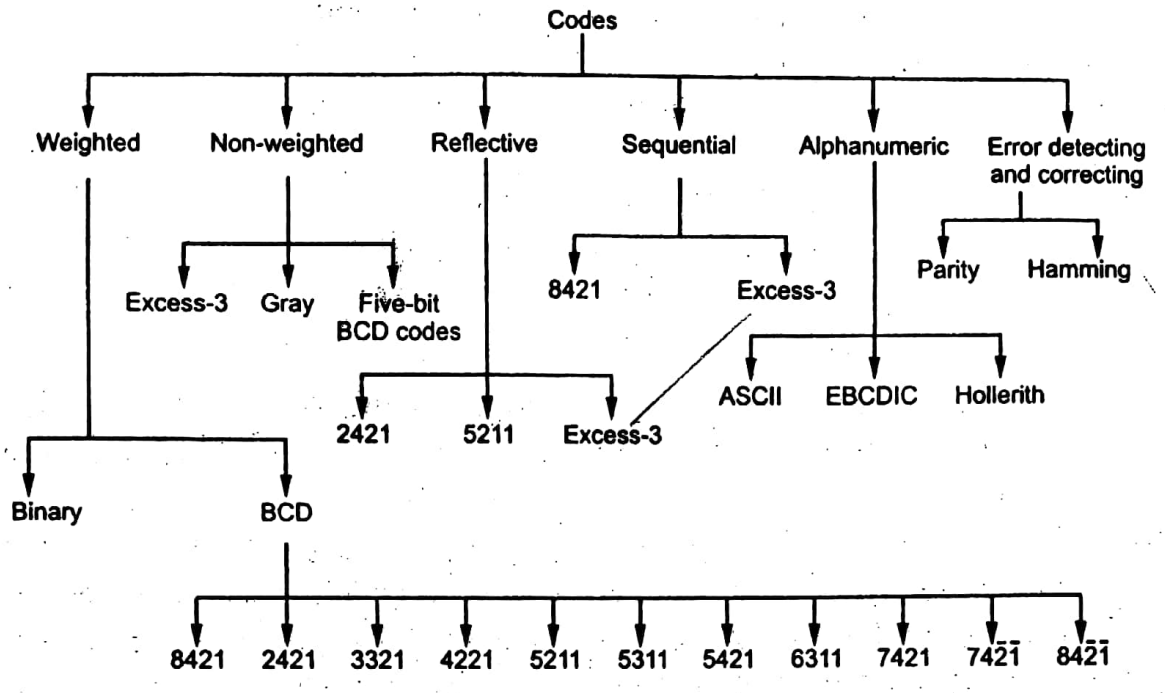


Fig. 1.6 Classification of various binary codes

1.12.2 BCD (8-4-2-1)

BCD is an abbreviation for binary-coded-decimal. BCD is a numeric code in which each digit of a decimal number is represented by a separate group of bits. The most common BCD code is 8-4-2-1 BCD, in which each decimal digit is represented by a 4-bit binary number. It is called 8-4-2-1 BCD because the weights associated with 4 bits are 8-4-2-1 from left to right. This means that, bit 3 has weight 8, bit 2 has weight 4, bit 1 has weight 2 and bit 0 has weight 1.

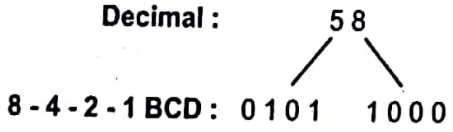
The Table 1.5 shows the 4-bit 8-4-2-1 BCD code used to represent a single decimal digit. The 8-4-2-1 BCD code is so widely used that it is common practice to refer to it simply as BCD.

Decimal	BCD Code			
Digit	8	4	2	1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1

6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

Table 1.5 8-4-2-1 BCD code

In multidigit coding, each decimal digit is individually coded with 8-4-2-1 BCD code. For example, 58 in decimal can be encoded in 8-4-2-1 BCD as :



As seen from the above example, in multidigit coding of 8-4-2-1 BCD numbers we require 4-bits per decimal digit. Therefore, total 8-bits are required to encode 58_{10} in 8-4-2-1 BCD. When we represent the same number (58) in binary : 111010_2, we require only 6 digits. This means that, for representing numbers, 8-4-2-1 BCD is less efficient than pure binary number system. The advantage of a BCD code is that it is easy to convert between it and decimal. The principle disadvantage of a BCD, besides its low efficiency, is that arithmetic operations are more complex than they are in pure binary. Let us see the arithmetic operations using 8-4-2-1 BCD.

1.12.2.1 BCD Addition

The addition of two BCD numbers can be best understood by considering the three cases that occur when two BCD digits are added.

Sum equals 9 or less with carry 0

Let us consider additions of 3 and 6 in BCD.

```
   6     0 1 1 0     ← BCD for 6
 + 3     0 0 1 1     ← BCD for 3
 ------  ------------------------------
   9     1 0 0 1     ← BCD for 9
```

The addition is carried out as in normal binary addition and the sum is 1 0 0 1, which is BCD code for 9.

Sum greater than 9 with carry 0

Let us consider addition of 6 and 8 in BCD

```
   6     0 1 1 0  ← BCD for 6
 + 8     1 0 0 0  ← BCD for 8
 ----------------------------------------
  14     1 1 1 0  ← Invalid BCD number (1110) > 9
```

The sum 1 1 1 0 is an invalid BCD number. This has occurred because the sum of the two digits exceeds 9. Whenever this occurs the sum has to be corrected by the addition of six (0110) in the invalid BCD number, as shown below

```
   6      0 1 1 0     ← BCD for 6
 + 8      1 0 0 0     ← BCD for 8
 ------   ----------------------
  1 4     1 1 1 0     ← Invalid BCD number
 +        0 1 1 0     ← Add 6 for correction
 ------------------------------------------
 0 0 0 1  0 1 0 0     ← BCD for 14
   1        4
```

After addition of 6 carry is produced into the second decimal position.

Sum equals 9 or less with carry 1

Let us consider addition of 8 and 9 in BCD

```
   8                1 0 0 0     ← BCD for 8
 + 9                1 0 0 1     ← BCD for 9
 ------   ----------------------------
  17      0 0 0 1 0 0 0 1     ← Incorrect BCD result
```

In this, case, result (0001 0001) is valid BCD number, but it is incorrect. To get the correct BCD result correction factor of 6 has to be added to the least significant digit sum, as shown.

```
   8                1 0 0 0     ← BCD for 8
 + 9                1 0 0 1     ← BCD for 9
 ------   ----------------------------
  17      0 0 0 1 0 0 0 1     ← Incorrect BCD result
        + 0 0 0 0 0 1 1 0     ← Add 6 for correction
 ------------------------------------
          0 0 0 1 0 1 1 1     ← BCD for 17
```

Going through these three cases of BCD addition we can summarise the BCD addition procedure as follows :

1. Add two BCD numbers using ordinary binary addition.
2. If four-bit sum is equal to or less than 9, no correction is needed. The sum is in proper BCD form.
3. If the four-bit sum is greater than 9 or if a carry is generated from the four-bit sum, the sum is invalid.
4. To correct the invalid sum, add 0110_2 to the four-bit sum. If a carry results from this addition, add it to the next higher-order BCD digit.

1.12.5 Gray Code

Gray code is a special case of **unit-distance code.** In unit-distance code, bit patterns for two consecutive numbers differ in only one bit position. These codes are also called **cyclic codes.** The Table 1.10 shows the bit patterns assigned for gray code from decimal 0 to decimal 15.

Decimal Code	Gray code
0	0 0 0 0
1	0 0 0 1
2	0 0 1 1
3	0 0 1 0
4	0 1 1 0
5	0 1 1 1
6	0 1 0 1
7	0 1 0 0
8	1 1 0 0
9	1 1 0 1
10	1 1 1 1
11	1 1 1 0
12	1 0 1 0
13	1 0 1 1
14	1 0 0 1
15	1 0 0 0

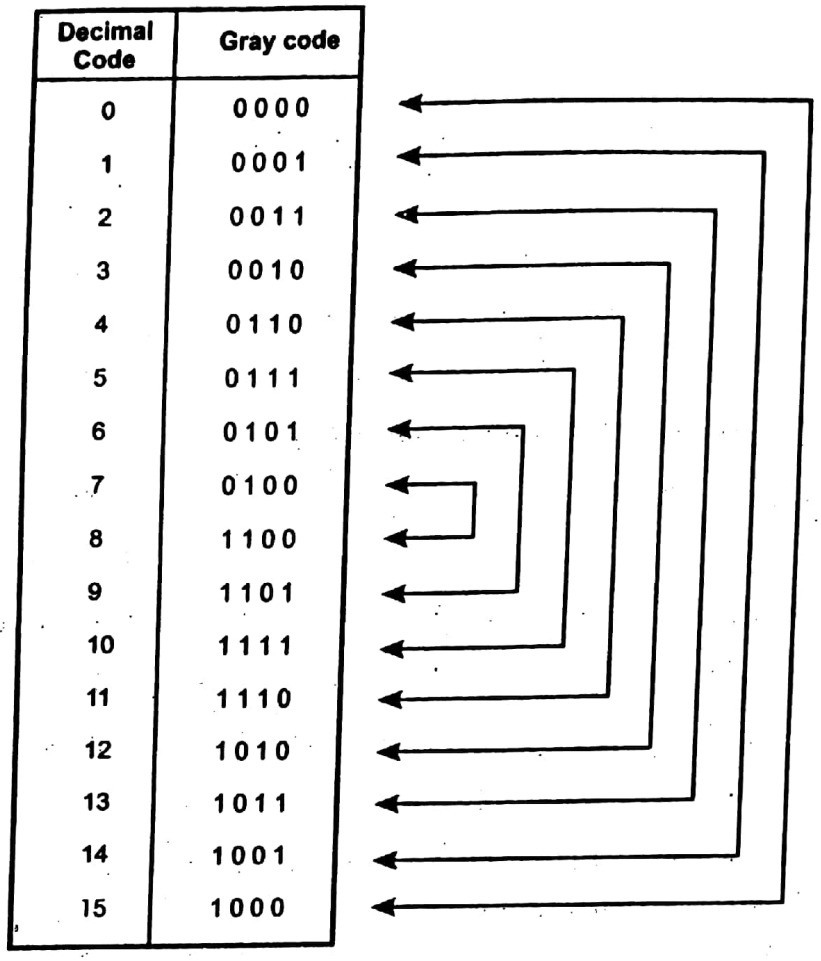


Table 1.10 Gray code

As shown in the Table 1.10 for gray code any two adjacent code groups differ only in one bit position. The gray code is also called **reflected code.** Notice that the two least significant bits for 4_{10} through 7_{10} are the mirror images of those for 0_0 through 3_{10}. Similarly, the three least significant bits for 8_{10} through 15_{10} are the mirror images of those for 0_{10} through 7_{10}. In general, the n least significant bits for 2^n through $2^{n+1} - 1$ are the mirror images of those for 0 through $2^n - 1$. This observation gives us a method for counting in gray code.

Another property of gray code is that the gray-coded number corresponding to the decimal number $2^n - 1$, for any n, differs from gray coded 0 (0000) in one bit position only. For example, for n = 2, 3 and 4, we see that

$2^2 - 1 = 3_{10}$ = 0 0 1 0 in gray

$2^3 - 1 = 7_{10}$ = 0 1 0 0 in gray and

$2^4 - 1 = 15_{10}$ = 1 0 0 0 in gray

Each differ from 0000 in one bit position only. This property places the gray code for the largest n-bit binary number at unit distance from 0.

1.12.5.1 Advantage of Gray Code

Let us consider an application where 3-bit binary code is provided to indicate position of the rotating disk. As shown in the Fig. 1.7, brushes are used to indicate black and white portions on the disk. When the brushes are on the black portion, they output a 1. When on the white portion, they output a 0.

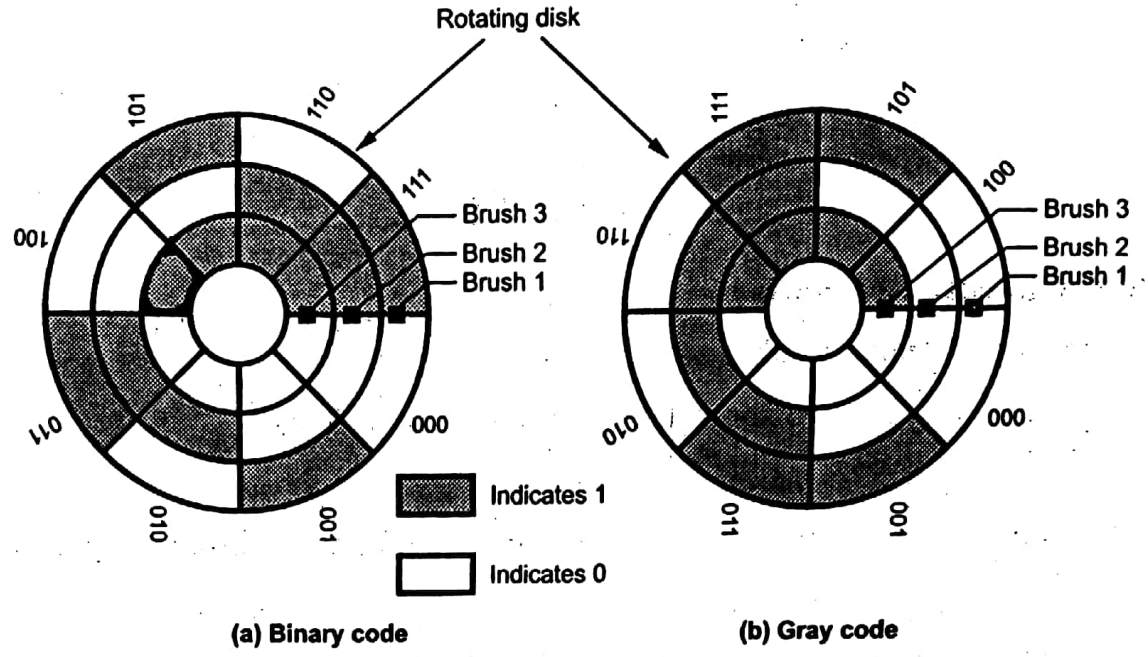


Fig. 1.7 Position indication using binary and gray codes

Now consider what happens when the brushes are on the 111 sector and almost ready to enter the 000 sector. If one brush is slightly ahead of the other, say the 3rd brush, then the position is indicated by a 011 instead of a 111 or 000. This results, a 180° error in the disk position. Since it is physically impossible to have all the brushes precisely aligned, some error will always be present at the edges of the sectors.

If we use gray code to represent disk position then error due to improper brush alignment can be reduced. This is because the gray code assures that only one bit will change each time the decimal number is incremented. So in 3-bit code, error may occur due to one bit position. Other two bits positions of two adjacent sectors are always same and hence there is no possibility of error. Therefore in 3-bit code probability of error is reduced upto 66%. In case of 4-bit code it is reduced upto 75%.

1.12.5.2 Gray-to-Binary Conversion

The gray to binary code conversion can be achieved using following steps.

1. The most significant bit of the binary number is the same as the most significant bit of the gray code number. So write it down.

2. To obtain the next binary digit, perform an exclusive-OR-operation between the bit just written down and the next gray code bit. Write down the result.
3. Repeat step 2 until all gray code bits have been exclusive-ORed with binary digits. The sequence of bits that have been written down is the binary equivalent of the gray-code number.

Example 1.58 : *Convert gray code 101011 into its binary equivalent.*

Solution :

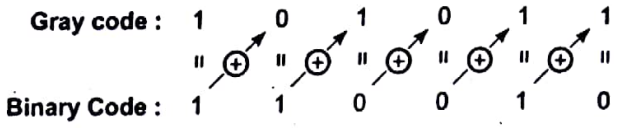


Fig. 1.8

1.12.5.3 Binary to Gray Conversion

Let us represent binary number as

$B_1\ B_2\ B_3\ B_4\ \ldots\ B_n$ and its equivalent gray code as

$G_1\ G_2\ G_3\ G_4\ \ldots\ G_n$.

With this representation gray code bits are obtained from the binary bits as follows :

$$G_1 = B_1$$

$$G_2 = B_1 \oplus B_2$$

$$G_3 = B_2 \oplus B_3$$

$$G_4 = B_3 \oplus B_4$$

$$\vdots$$

$$G_n = B_{n-1} \oplus B_n$$

Example 1.59 : *Convert 10111011 in binary into its equivalent gray code.*

Solution :

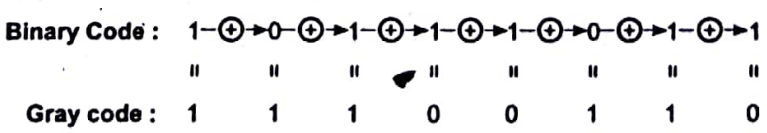


Fig. 1.9

BOOLEAN ALGEBRA

RULES

RULE 1: $0 + A = A$

$A + 0 = A$

RULE 2: $1 + A = 1$

$A + 1 = 1$

RULE 3: $A + A = A$

RULE 4: $A + \bar{A} = 1$

$\bar{A} + A = 1$

RULE 5: $0 \cdot A = 0$

$A \cdot 0 = 0$

RULE 6: $1 \cdot A = A$

$A \cdot 1 = A$

RULE 7: $A \cdot A = A$

RULE 8: $A \cdot \bar{A} = 0$

$\bar{A} \cdot A = 0$

RULE 9: $\overline{\bar{A}} = A$ (Involution)

LAWS

LAW 1: $A + B = B + A$

$AB = BA$

} Commutative law.

LAW 2: $A + (B + C) = (A + B) + C$

$(AB)C = A(BC)$

} Associative law.

LAW 3: $A(B + C) = AB + AC$

$A + BC = (A + B)(A + C)$

} Distributive law.

ADDITIONAL RULES

* $A + AB = A$

* $A + \bar{A}B = A + B$

* $(A+B)(A+C) = A + BC$

THEOREMS

DeMorgan's Theorems

1) $\overline{AB} = \bar{A} + \bar{B}$

2) $\overline{A+B} = \bar{A}\,\bar{B}$

CONSENSUS THEOREM

$$AB + \bar{A}C + BC = AB + \bar{A}C$$

PROOF FOR ADDITIONAL RULES

1) $A + AB = A$

$A(1+B) = A(1) = A$ //

2) $A + \bar{A}B$

According to distributive law

$(A+\bar{A})(A+B) = A+B$

$(A+\bar{A}) = 1$

3) $AB + \bar{A}C + BC$

$= AB + \bar{A}C + BC(A + \bar{A})$

$= AB + \bar{A}C + ABC + \bar{A}BC$

$= AB(1+C) + \bar{A}C(1+B) = AB + \bar{A}C$ //

$(1+C) = 1$, $(1+B) = 1$

PROBLEMS

1) Simplify the Boolean Expression.

i) $Y = A + \bar{A}B + \bar{A}\bar{B}C + \bar{A}\bar{B}\bar{C}D$

Solution

$$A + B + \bar{A}\bar{B}C + \bar{A}\bar{B}\bar{C}D \quad \{\because A + \bar{A}B = A + B$$

$$A + B + \bar{A}\bar{B}[C + \bar{C}D]$$

$$A + B + \bar{A}\bar{B}[C + D] \quad \{\because C + \bar{C}D = C + D$$

$$A + B + \bar{A}\bar{B}C + \bar{A}\bar{B}D$$

$$A + \bar{B}C + B + \bar{A}D$$

$$A + D + B + C$$

$$\therefore A + \bar{A}B + \bar{A}\bar{B}C + \bar{A}\bar{B}\bar{C}D = A + B + C + D$$

---

ii) $Y = \bar{A}B + ABD + A\bar{B}C\bar{D} + BC$

Solution

$$Y = B[\bar{A} + AD] + C[B + \bar{B}A\bar{D}]$$

$$= B(\bar{A} + D) + C(B + A\bar{D}) \quad \{\because A + \bar{A}B = A + B$$

$$= \bar{A}B + BD + BC + AC\bar{D}$$

$$= \bar{A}B + BD + BC(A + \bar{A}) + AC\bar{D} \quad (A + \bar{A} = 1)$$

$$= \bar{A}B + BD + ABC + \bar{A}BC + AC\bar{D}$$

$$= \bar{A}B(1 + C) + BD + ABC + AC\bar{D}$$

$= \bar{A}B + BD + AC\bar{D} + ABC$ $\quad \{\because 1+C=1$

$= \underline{\bar{A}B + BD + AC\bar{D}}$

Consensus theorem

$AB + \bar{A}C + BC = AB + \bar{A}C$

$BB + \bar{D}AC + BAC$

$= DB + \bar{D}AC$

$= BD + AC\bar{D}$

---

iii) $A\bar{B}C + \bar{A}BC + \bar{A}\bar{B}C$

Solution

$= C\left[A\bar{B} + \bar{A}B + \bar{A}\bar{B}\right]$

$= C\left[A\bar{B} + \bar{A}(B + \bar{B})\right]$ $\quad \{\because B + \bar{B} = 1$

$= C\left[A\bar{B} + \bar{A}(1)\right]$

$= C\left[A\bar{B} + \bar{A}\right]$ $\quad \{\because A + \bar{A}B = A + B$, $\bar{A} + A\bar{B} = \bar{A} + \bar{B}$

$= C(\bar{A} + \bar{B})$ //

(iv) $A(\bar{A} + AB)$

$= A(\bar{A} + AB)$

$= A(\bar{A} + B)$ $\quad \{\because \bar{A} + AB = \bar{A} + B$

$= A \cdot \bar{A} + A \cdot B$

$= 0$

$= AB$ // $\quad \{\because A \cdot \bar{A} = 0$

2.6 Minimization of Boolean Expression

We know that Boolean functions can be realised by logic gates. The number of gates required and the number of input terminals for the gates for the implementation of a Boolean function, in general, get reduced considerably if the Boolean function can be simplified. Therefore, the simplification of Boolean function is very important as it saves the hardware required and hence the cost for design of specific Boolean function. Let us see some examples to illustrate the Boolean equation simplification.

Examples

Ex. 2.7 : $A \cdot \overline{A}C = 0.C$ — Rule 8 : [$A\overline{A} = 0$]

$= 0$ — Rule 5 : [$A.0 = 0$]

Ex. 2.8 : $ABCD + ABD = ABD\,(C + 1)$ — Distributive law

$= ABD \,.\, 1$ — Rule 2 : [$A + 1 = 1$]

$= ABD$ — Rule 6 : [$A \cdot 1 = A$]

Ex. 2.9 : $ABCD + A\overline{B}CD = ACD\,(B + \overline{B})$ — Distributive law

$= ACD.1$ — Rule 4 : [$B + \overline{B} = 1$]

$= ACD$ — Rule 6 : [$A \cdot 1 = A$]

Ex. 2.10 : $A\,(A + B) = AA + AB$ — Distributive law

$= A + AB$ — Rule 7 : [$A \cdot A = A$]

$= A\,(1 + B)$ — Distributive law

$= A$ — Rule 2 : [$A + 1 = 1$]

Ex. 2.11 :

$AB + ABC + AB\,(D + E) = AB\,(1 + C + D + E)$

$= AB$ — Rule 2 : [$A + 1 = 1$]

Ex. 2.12 : $XY + XYZ + XY\overline{Z} + \overline{X}YZ$

$= XY\,(1 + Z) + XY\overline{Z} + \overline{X}YZ$

$= XY + XY\overline{Z} + \overline{X}YZ$ — Rule 2 : [$A + 1 = 1$]

$= XY\,(1 + \overline{Z}) + \overline{X}YZ$

$= XY + \overline{X}YZ$ — Rule 2 : [$A + 1 = 1$]

$= Y\,(X + \overline{X}Z)$

$= Y\,(X + Z)$ — Rule 11 : [$A + \overline{A}B = A + B$]

Ex. 2.13 :

$\overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,B\,\overline{C} + \overline{A}\,B\,C = \overline{A}\,\overline{C}\cdot(\overline{B} + B) + \overline{A}BC$

$= \overline{A}\,\overline{C} + \overline{A}BC$ — Rule 4 : [$A + \overline{A} = 1$]

$= \overline{A}(\overline{C} + BC)$ — Distributive law

$= \overline{A}(\overline{C} + B)$ — Rule 11 : [$A + \overline{A}B = A + B$]

2.14 : $A\bar{B}C + A\bar{B}C + AB\bar{C} = A(C + B)$

L.H.S. $= AC(B + \bar{B}) + AB\bar{C}$ — Distributive law

$= AC \cdot 1 + AB\bar{C}$ — Rule 4 : $[A + \bar{A} = 1]$

$= AC + AB\,\bar{C}$ — Rule 6 : $[A \cdot 1 = A]$

$= A(C + B\bar{C})$ — Distributive law

$= A(C + B)$ — Rule 11 : $[A + \bar{A}B = A + B]$

2.15 : $\bar{A}\,B\,C\bar{D} + BC\bar{D} + B\bar{C}\,\bar{D} + B\,\bar{C}\,D$

$= BC\bar{D}(\bar{A} + 1) + B\bar{C}\,\bar{D} + B\bar{C}D$ — Distributive law

$= BC\bar{D} + B\bar{C}\,\bar{D} + B\bar{C}D$ — Rule 2 : $[A + 1 = 1]$

$= B\bar{D}(C + \bar{C}) + B\bar{C}D$ — Distributive law

$= B\bar{D} + B\bar{C}D$ — Rule 4 : $[A + \bar{A} = 1]$

$= B(\bar{D} + \bar{C}\,D)$ — Distributive law

$= B(\bar{D} + \bar{C})$ — Rule 11 : $[A + \bar{A}B = A + B]$

2.16 : $AC + C(A + \bar{A}B)$

$= AC + AC + \bar{A}BC$ — Distributive law

$= AC + \bar{A}BC$ — Rule 3 : $[A + A = A]$

$= C(A + \bar{A}B)$ — Distributive law

$= C(A + B)$ — Rule 11 : $A + \bar{A}B = A + B$

2.17 : $\bar{A}\,B\,\bar{C}\,D + \bar{A}BCD + ABD$

$= \bar{A}BD(\bar{C} + C) + ABD$ — Distributive law

$= \bar{A}BD + ABD$ — Rule 4 : $[A + \bar{A} = 1]$

$= BD(\bar{A} + A)$ — Distributive law

$= BD$ — Rule 4 : $[A + \bar{A} = 1]$

Ex. 2.18 : $A + \bar{A}B + A\bar{B} = A + B$

L.H.S $= A + \bar{A}B + A\bar{B}$

$= A + B + A\bar{B}$ — Rule 11 : $[A + \bar{A}B = A + B]$

$= A + A + B$ — Rule 11 : $[A + \bar{A}B = A + B]$

$= A + B$ — Rule 5 : $[A + A = A]$

Ex. 2.19 : $\overline{A\bar{B} + \bar{A} + AB}$

$= \overline{\bar{A} + \bar{B} + \bar{A} + AB}$ — Theorem 1 : $[\overline{A\bar{B}} = \bar{A} + \bar{B}]$

$= \overline{\bar{A} + \bar{B} + \bar{A} + B}$ — Rule 11 : $[A + \bar{A}\,B = A + B]$

$= \overline{\bar{A} + 1}$ — Rule 3 : $[A + A = A]$ and Rule 4 : $[A + \bar{A} = 1]$

$= \overline{1}$ Rule 2 : [A + 1 = 1]

$= 0$

Ex. 2.20 : $AB + \overline{AC} + A\overline{B}C\,(AB + C)$

$= AB + \overline{AC} + AA\overline{B}\,BC + A\overline{B}\,CC$ Distributive law

$= AB + \overline{AC} + A\overline{B}\,CC$ Rule 8 : [A · $\overline{A}$ = 0]

$= AB + \overline{AC} + A\overline{B}\,C$ Rule 7 : [A · A = A]

$= AB + \overline{A} + \overline{C} + A\,\overline{B}\,C$ Theorem 1 : [$\overline{AB} = \overline{A} + \overline{B}$]

$= \overline{A} + B + \overline{C} + A\overline{B}\,C$ Rule 11 : [A + $\overline{A}$B = A + B]

$= \overline{A} + A\,\overline{B}\,C + B + \overline{C}$ Commutative law

$= \overline{A} + \overline{B}\,C + B + \overline{C}$ Rule 11 : [A + $\overline{A}$B = A + B]

Here B = $\overline{B}$C

$= \overline{A} + B + \overline{C} + \overline{B}\,C$ Commutative law

$= \overline{A} + B + \overline{C} + \overline{B}$ Rule 11 : [A + $\overline{A}$B = A + B]

$= \overline{A} + \overline{C} + 1$ Rule 4 : [A + $\overline{A}$ = 1]

$= 1$ Rule 2 : [A + 1 = 1]

Example 2.21 : *Simplify the expression* $Z = A\,B + A\,\overline{B} \cdot (\overline{A}\,\overline{C})$.

Solution :

Step 1 : Apply the DeMorgan's theorem and multiply out all terms to get expression in sum of products form

$Z = A\,B + A\,\overline{B} \cdot (\overline{\overline{A}\,\overline{C}})$

$= A\,B + A\,\overline{B} \cdot (\overline{\overline{A} + \overline{C}})$ De Morgan's theorem 1

$= A\,B + A\,\overline{B} \cdot (A + C)$ Rule 9 : [$\overline{\overline{A}}$ = A]

$= A\,B + A\,\overline{B}\,A + A\,\overline{B}\,C$ Distributive law

Step 2 : Search for common terms for factorization and apply boolean rules

$Z = A\,B + A\,\overline{B}\,A + A\,\overline{B}\,C$

$= A\,B + A\,\overline{B} + A\,\overline{B}\,C$ Rule 7 : [AA = A]

$= A\,B + A\,\overline{B}\,(1 + C)$

$= A\,B + A\,\overline{B}$ Rule 2 : [1 + C = 1]

$= A\,(B + \overline{B})$ Rule 4 : [A + $\overline{A}$ = 1]

$= A$

Example 2.48 : *Simplify the following expression*

$Y = (A + B)\left(\overline{A}+C\right)\left(\overline{B}+\overline{C}\right)$

Solution :

$= \left(A\overline{A}+A\,C+\overline{A}\,B+BC\right)\left(\overline{B}+\overline{C}\right)$

$= \left(AC+\overline{A}B+BC\right)\left(\overline{B}+\overline{C}\right)$ Rule 8 : $\left[A\overline{A}=0\right]$

$= A\overline{B}C+AC\overline{C}+\overline{A}\,B\overline{B}+\overline{A}\,B\,\overline{C}+B\,\overline{B}C+B\,C\overline{C}$

$= A\overline{B}C+\overline{A}B\overline{C}$ Rule 8 : $\left[A\overline{A}=0\right]$

Example 2.49 : *Simplify each of the following using Demorgan's theorem*

a) $\overline{\left(A+\overline{B}\right)\left(\overline{A}+B\right)}$ *b)* $\overline{\overline{A\,B\,C\,D}}$

Solution :

a) $\overline{\left(A+\overline{B}\right)\left(\overline{A}+B\right)}$

$= \overline{A+\overline{B}}+\overline{\overline{A}+B}$ DeMorgan's theorem 1

$= \overline{A}\cdot\overline{\overline{B}}+\overline{\overline{A}}\cdot\overline{B}$ DeMorgan's theorem 2

$= \overline{A}\cdot B+A\cdot\overline{B}$ Rule 9 : $\left[\overline{\overline{A}}=A\right]$

$= A \oplus B$

b) $\overline{\overline{A\,B\,C\,D}}$

$= \overline{\overline{A\,B\,C}}+\overline{D}$ DeMorgan's theorem 1

$= \overline{A\,B\,C}+\overline{D}$ Rule 9 : $\left[\overline{\overline{A}}=A\right]$

$= \left(\overline{A}+\overline{B}\right)C+\overline{D}$ DeMorgan's theorem 1

Example 2.50 : *Verify the following Boolean algebraic manipulation. Justify each step with a reference to a postulate or theorem :*

i) $(X+\overline{Y}+XY)\,(X+\overline{Y})\,\overline{X}Y = 0$

ii) $(AB+C+D)\,(\overline{C}+D)\,(\overline{C}+D+E) = AB\overline{C}+D$

Solution : i) $(X+\overline{Y}+XY)\;(X+\overline{Y})\,(\overline{X}Y)$

$= (X+\overline{Y}+X)\;(X+\overline{Y})\,(\overline{X}Y)$ $\because\ A+\overline{A}B = A+B$

$= (X+\overline{Y})\;(X+\overline{Y})\,(\overline{X}Y)$ $\because\ A+A = A$

EXAMPLE

Simplify the following expressions using Boolean algebra:

(a) $A + AB + A\bar{B}C$

According to the rule $A + AB = A$

$= A + A\bar{B}C$

$= A[1 + \bar{B}C]$

According to the $1 + A = 1$

$1 + \bar{B}C = 1$

$= A(1)$

$= A$ //

(b) $(\bar{A} + B)C + ABC$

$= \bar{A}C + BC + ABC$

$= \bar{A}C + BC(1 + A)$

$= \bar{A}C + BC$ $\quad \because A + 1 = 1$

$= C(\bar{A} + B)$ //

(c) $A\bar{B}C(\bar{B}D + CDE) + A\bar{C}$

$= A\underbrace{\bar{B}B}_{0}CD + A\bar{B}\underbrace{CC}_{C}DE + A\bar{C}$

$\because A \cdot \bar{A} = 0$

$A \cdot A = A$

$= A\bar{B}CDE + A\bar{C}$

$= A[\bar{B}CDE + \bar{C}]$

$= A[\bar{C} + \bar{B}DE]$ $\quad \because A + \bar{A}B = A + B$

$\bar{C} + C\bar{B}DE = \bar{C} + \bar{B}DE$

$= A[\bar{B}DE + \bar{C}]$ //

15.6 LOGIC GATES

The basic elements that make up a digital system are called as logic gates. The most common logic gates are OR, AND, NOT, NAND and NOR gates. The NAND and NOR gates are called as universal gates. Exclusive-OR gate is another logic circuit which can be constructed using AND, OR and NOT gates.

15.6.1 OR Gate

The OR gate performs logical addition, commonly known as OR function. The OR gate has two or more inputs and only one output. The operation of OR gate is such that a high (1) on the output is produced when any of the inputs is high (1). The output is low (0) only when all the inputs are low (0).

As shown in Fig. 15.1, A and B represent the inputs and Y the output. Resistance R is the load resistance.

If $A = 0$ and $B = 0$ then $V_0 = 0$ and $Y = 0$.

If $A = 1$ and $B = 0$, diode D_1 will conduct and so the output $Y = 1$.

If $A = 0$ and $B = 1$, diode D_2 will conduct and the output $Y = 1$.

If $A = 1$ and $B = 1$, both the diodes will conduct and so the output $Y = 1$.

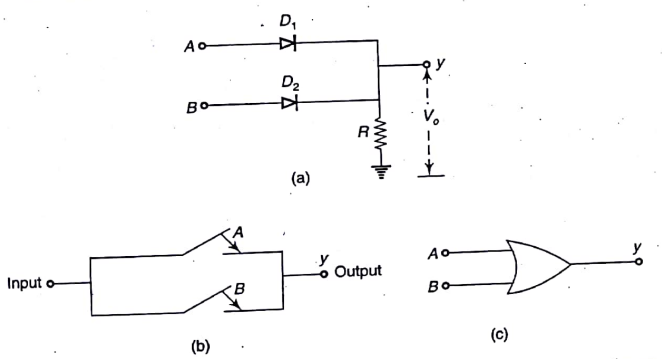


Fig. 15.1 (a) Circuit diagram of an OR gate, (b) Electrical equivalent of an OR gate, (c) Logic symbol

The electrical equivalent circuit of an OR gate is shown in Fig. 15.1(b) where switches A and B are connected in parallel with each other. If either A, B or both are closed, then the output will result. The logic symbol for OR gate is shown in Fig. 15.1(c). The logic operation of the two input OR gate is described in the truth table shown in Table 15.3.

Table 15.3 Truth table for the two-input OR gate

Input		Output
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

15.6.2 AND Gate

The AND gate performs logical multiplication, commonly known as AND function. The AND gate has two or more inputs and a single output. The output of AND gate is high only when all the inputs are high. When any of the inputs is low, the output is low.

As shown in the Fig. 15.2(a), A and B represent the inputs and Y represents the output.

If $A = 0$ and $B = 0$, both diodes conduct as they are forward biased and the output $Y = 0$.

If $A = 0$ and $B = 1$, diode D_1 conducts and D_2 does not conduct, and again the output $Y = 0$.

If $A = 1$ and $B = 0$, diode D_1 does not conduct and D_2 conducts, and the output $Y = 0$.

If $A = 1$ and $B = 1$, both the diodes do not conduct as they are reverse based and so the output $Y = 1$.

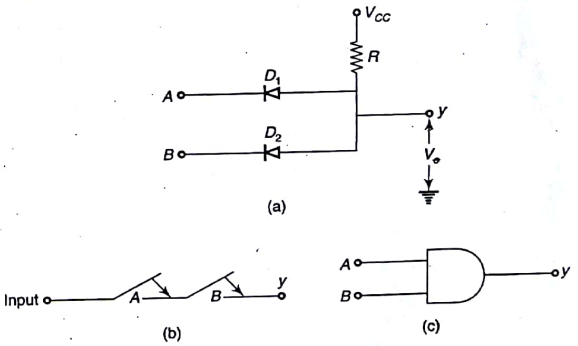


Fig. 15.2 (a) Circuit diagram of an AND gate, (b) Electrical equivalent of an AND gate, (c) Logic symbol

The electrical equivalent circuit of an AND gate is shown in Fig. 15.2 (b) where two switches A and B are connected in series. If both A and B are closed, then only output will result. Logic symbol of the AND gate is shown in Fig. 15.2(c). The logic operation of the two input AND gate is described in the truth table shown in Table 15.4.

Table 15.4 Truth table for a two-input AND gate

Input		*Output*
A	*B*	*Y*
0	0	0
0	1	0
1	0	0
1	1	1

15.6.3 NOT Gate (Inverter)

The NOT gate performs a basic logic function called inversion or complementation. The purpose of the gate is to change one logic level to opposite level. It has one input and one output. When a high level is applied to an inverter input, a low level will appear at its output and vice-versa. The operation of the circuit can be explained as follows. When a high voltage is applied to the base of the transistor, base current increases and the transistor is saturated. The transistor now acts as a closed switch and conducts heavily. Thus the output voltage is logic 0. On the other hand, when a low voltage is applied at the base, the transistor is cut-off due to very low or no base current. Now, the transistor can be considered as an open switch, with no current flowing through it. The output is now clamped to the supply voltage. The transistor when operated between cut-off and saturation will act as a switch.

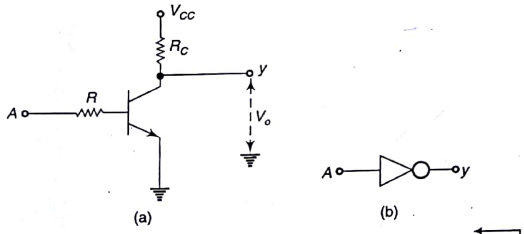


Fig. 15.3 (a) Circuit diagram of an INVERTER gate (b) Logic symbol

As shown in Fig. 15.3(a), A represents the input and y represents the output. If the input is high, the transistor is in ON state and the output is low. If the input is low, the transistor is in OFF state and the output is high. The symbol for the inverter is shown in Fig. 15.3(b). The truth table is given in Table 15.5.

Table 15.5 Truth table for an INVERTER

Input	*Output*
A	*Y*
0	1
1	0

15.6.4 NAND Gate

NAND is a contraction of NOT–AND. It has two or more inputs and only one output. When all the inputs are high, the output is low. If any of the inputs is low, the output is high. The logic symbol for the NAND gate is shown in Fig. 15.4.

The truth-table for the NAND gate is shown in Table 15.6.

Table 15.6 Truth table for NAND gate

Input		*Output*
A	*B*	*y*
0	0	1
0	1	1
1	0	1
1	1	0

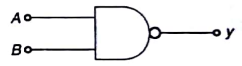


Fig. 15.4 Logic symbol for the NAND gate

The NAND gate is a very popular logic function because it is an universal function; that is, it can be used to construct an AND gate, an OR gate, and INVERTER or any combination of these functions. Figure 15.5 shows how NAND gates can be connected to realize various logic gates.

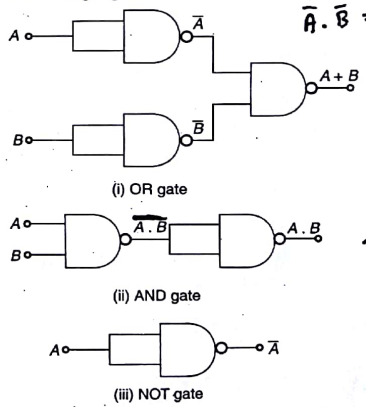


Fig. 15.5 NAND gates connected to realize (a) OR (b) AND and (c) NOT gates

15.6.5 NOR Gate

NOR is a contraction of NOT–OR. It has two or more inputs and only one output. Only when all the inputs are low, the output is high. If any of the inputs is high, the output is low. The logic symbol for the NOR gate is shown in Fig. 15.6.

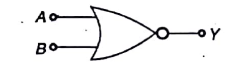


Fig. 15.6 Logic symbol for NOR gate

The truth-table for the NOR gate is shown in Table 15.7.

Table 15.7 Truth table for NOR gate

Input		Output
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

The NOR gate is also a very popular logic function because it is also an universal function; that is, it can be used to construct an AND gate, an OR gate, and INVERTER or any combination of these functions. Figure 15.7 shows how NOR gates can be connected to realize various logic gates.

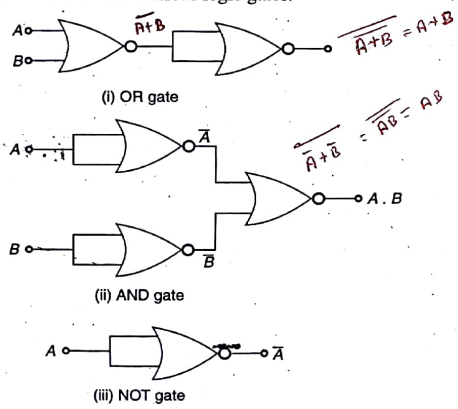
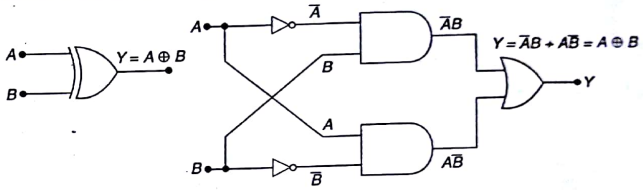


Fig. 15.7 NOR gates connected to realize (a) OR (b) AND and (c) NOT gates

15.6.6 Exclusive–OR (Ex-OR) Gate

An Exclusive-OR Gate is a gate with two or more inputs and one output. The output of a two-input Ex-OR gate assumes a HIGH state if one and only one input assumes a HIGH state. This is equivalent to saying that the output is HIGH if either input A or input B is HIGH exclusively, and low when both are 1 or 0 simultaneously.

The logic symbol for the Ex-OR gate is shown in Fig. 15.8 (a) and the truth table for the Ex-OR operation is given in Table 15.8.



(a) Logic symbol (b) Using AND-OR-NOT gates

Fig. 15.8 Ex-OR gate

Table 15.8 Truth table of a 2-input Ex-OR gate

Input		Output
A	B	$Y = A + B$
0	0	0
0	1	1
1	0	1
1	1	0

The truth table of the Ex-OR gate shows that the output is HIGH when any one, but not all, of the inputs is at 1. This exclusive feature eliminates a similarity to the OR gate. The Ex-OR gate responds with a HIGH output only when an odd number of inputs is HIGH. When there is an even number of HIGH inputs, such as two or four, the output will always be LOW. From the truth table of a 2-input Ex-OR gate, the Ex-OR function can be written as $Y = \overline{A}B + A\overline{B} = A \oplus B$.

The above expression can be read as Y equals A Ex-OR B. Using the above expression, a 2-input Ex-OR gate can be implemented using basic gates like AND, OR and NOT gates as shown in Fig. 15.8.

The 2-input Ex-OR gate can also be implemented using NAND gates as shown in Fig. 15.9.

The main characteristic property of an Ex-OR gate is that it can perform modulo-2 addition. It should be noted that the same Ex-OR truth table applies when adding two *binary digits* (bits). A 2-input Ex-OR circuit is, therefore, sometimes called a *module-2-adder* or a *half-adder* without carry output. The name half-adder refers to the fact that possible carry-bit, resulting from an addition of two preceding bits, has not been taken into account. A full addition is performed by a second Ex-OR circuit

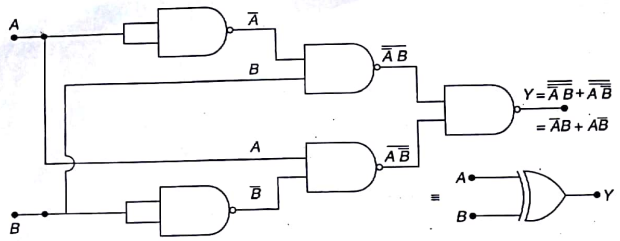


Fig. 15.9 Ex-OR gates using NAND gates

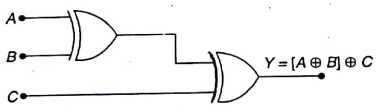


Fig. 15.10 Cascading of two Ex-OR circuits

with the output signal of the first circuit and the carry as input signals, as shown in Fig. 15.10.

The configuration of Fig. 15.10 is a cascading of two Ex-OR circuits resulting in an Ex-OR operation of three variables A, B, and C. Consequently, the sum output of a full adder for two bits is an Ex-OR operation of the 2 bits to be added and the carry of the preceding adding stage. The logic expression of the Ex-OR operation of three variables A, B, and C is given by

$$A \oplus B \oplus C = (A\overline{B} + \overline{A}B)\overline{C} + \overline{(A\overline{B} + \overline{A}B)}C$$
$$= (A\overline{B} + \overline{A}B)\overline{C} + (\overline{A}\,\overline{B} + AB)C$$
$$A \oplus B \oplus C = A\overline{B}\,\overline{C} + \overline{A}B\overline{C} + \overline{A}\,\overline{B}C + ABC$$

In general, Ex-OR operation of n variables results in a logical 1 output if an *odd* number of the input variables are 1s. An Ex-OR operation of n variables can be obtained by cascading 2-input Ex-OR gates.

Another important property of an Ex-OR gate is that it can be used as a *controlled inverter*, i.e. by using an Ex-OR gate, a logic variable can be complemented or allowed to pass through it unchanged. This is done by using one Ex-OR input as a control input and the other as the logic variable input as shown in Fig. 15.11. When the control input is HIGH, the output $Y = \overline{A}$ and when the control input is LOW, the output $Y = A$.

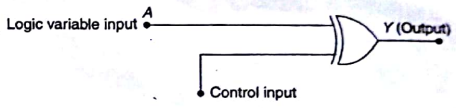


Fig. 15.11 Ex-OR gate as a controlled inverter

15.6.7 Exclusive-NOR (Ex-NOR) Gate

The exclusive-NOR gate, abbreviated Ex-NOR, is an Ex-OR gate, followed by an inverter. An Exclusive-NOR gate has two or more inputs and one output. The output of a two-input Ex-NOR gate assumes a HIGH state if both the inputs assume the same logic state or have an even number of 1s, and its output is LOW when the inputs assume different logic states or have an odd number of 1s. The logic symbol of Ex-NOR gate is shown in Fig. 15.12 and its truth table is given in Table 15.9. From the truth table, it is clear that the Ex-NOR output is the complement of the Ex-OR gate. The Boolean expression for the Ex-NOR gate is

$$Y = \overline{A \oplus B}$$

Read the above expression as "Y equals A Ex-NOR B". According to DeMorgan's theorem.

$$\overline{A \oplus B} = \overline{A}B + A\overline{B}$$
$$= \overline{A}B \cdot A\overline{B}$$
$$= (A + \overline{B})(\overline{A} + B)$$
$$= AB + \overline{A}\,\overline{B}$$

Table 15.9 Truth table of 2-input Ex-NOR gate

Input		*Output*
A	B	$Y = \overline{A \oplus B}$
0	0	1
0	1	0
1	0	0
1	1	1

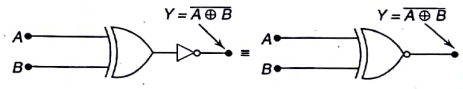


Fig. 15.12 Logic symbol of 2-input Ex-NOR gate

An important property of the Ex-NOR gate is that it can be used for bit comparison. The output of an Ex-NOR gate is 1 if both the inputs are similar, i.e. both are 0 or 1; otherwise, its output is 0. Hence, it can be used as a one-bit comparator. It is also called a *coincidence circuit.*

Another property of the Ex-NOR gate is that it can be used as an *even-parity checker*. The output of the Ex-NOR gate is 1 if the number of 1 s in its inputs is even; if the number of 1s is odd, the output is 0. Hence, it can be used as an even/odd parity checker. Hence, the 2-input Ex-NOR gate is immensely useful for bit comparison and parity checking.

Example 15.14 Realise the logic expression $Y = \overline{B}\overline{C} + \overline{A}\overline{C} + \overline{A}\overline{B}$ using basic gates.

Solution: In the given expression, there are 3 product terms each with two variables which can be implemented using three 2-input AND gates, and the product terms can be OR operated together using a 3-input OR gate. The complemented form of individual variable can be obtained by 3 NOT gates. Thus, the circuit for the given expression is realised as shown in Fig. 15.13.

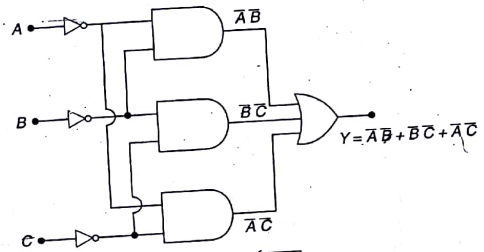


Fig. 15.13

Example 15.15 Realise the logic expression $Y = (A + B)\,(\overline{A} + C)\,(B + D)$ using basic gates.

Solution: In the given expression, there are 3 sum terms which can be implemented using three 2-input OR gates and their outputs are AND operated together by a 3-input AND gate. A NOT gate can be used to obtain the inverse of A. Now, the realized circuit is shown in Fig. 15.14.

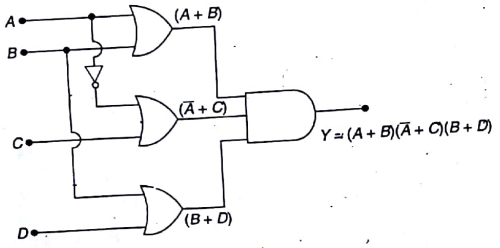


Fig. 15.14

Example 15.16 Implement $Y = \overline{AB} + A + (\overline{B + C})$ using NAND gates only.

Solution: The implementation of the given function is shown in Fig. 15.15.

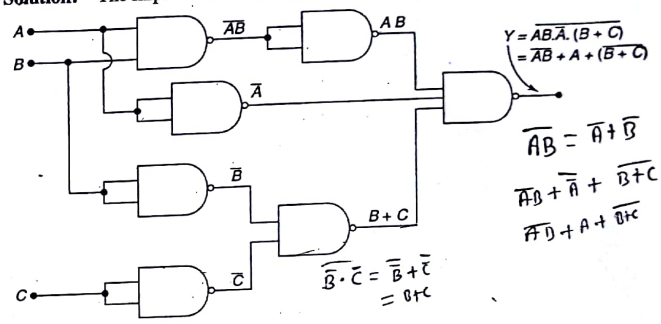


Fig. 15.15

Example 15.17 Simplify the logic circuit shown in Fig. 15.16(a)

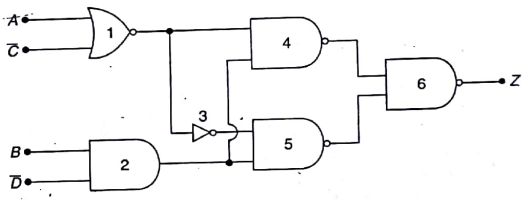


Fig. 15.16 (a)

Solution: From the given logic circuit, the expression for Z can be written as

$$Z = \overline{\overline{(A + \overline{C})\cdot B\overline{D}}\cdot\overline{(A + \overline{C})\cdot B\overline{D}}}$$

$$= \overline{\overline{[(A + \overline{C}) + B\overline{D}]}\cdot\overline{(A + \overline{C}) + B\overline{D}}}$$

$$= \overline{[(A + \overline{C}) + B\overline{D}]\cdot[(A + \overline{C}) + B\overline{D}]}$$

$$= \overline{B\overline{D}} + (A + \overline{C})(A + \overline{C}) \qquad [\because (A + B)\,(A + C) = A + BC]$$

$$= \overline{B\overline{D}} \qquad [\because A\overline{A} = 0]$$

$$= B\overline{D}$$

Therefore, the above logic circuit can be simplified as shown in Fig. 15.16 (b).

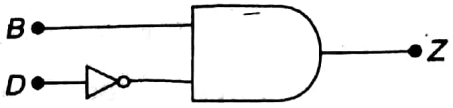


Fig. 15.16 (b)

Instead of using Boolean algebra, the logic circuit can be simplified directly as shown below. In the given logic circuit shown in Fig. 15.16 (a) the NAND gate (6) can be replaced by an OR gate with a bubble at its inputs as shown in Fig. 15.16 (c).

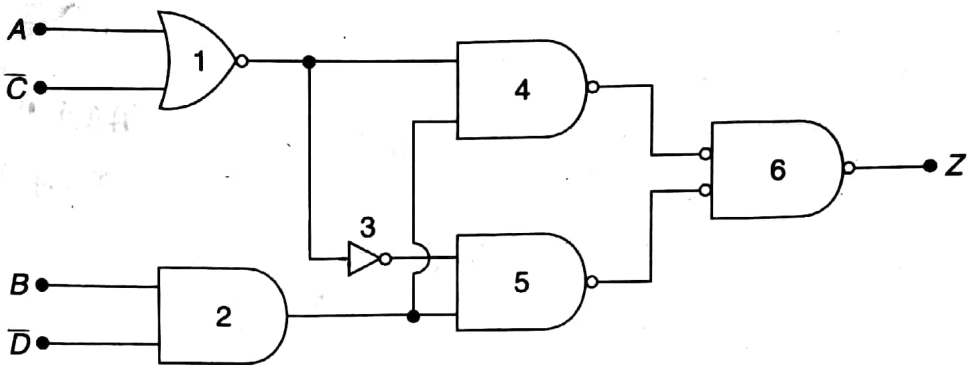


Fig. 15.16 (c)

Now, using $\overline{\overline{A}} = A$, the bubble at the outputs of gates 4 and 5 get cancelled with the bubble at the inputs of gate 6 as shown in Fig. 15.16 (d).

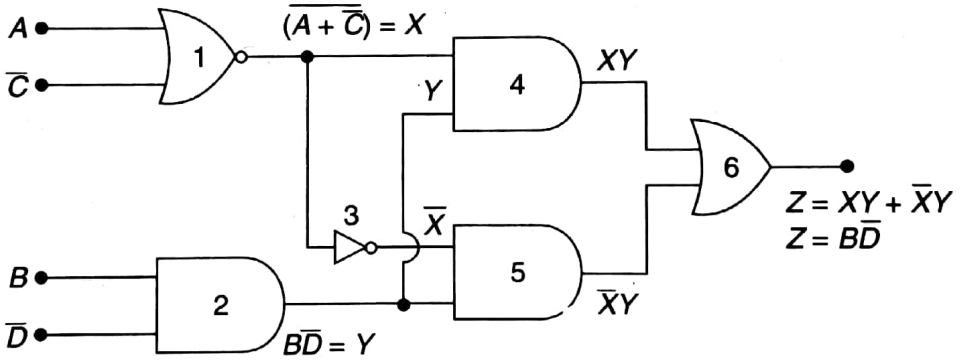


Fig. 15.16 (d)

In the above figure, if we assume the output of gate 1 $\overline{(A + \overline{C})} = X$ and the output of gate 2 $B\overline{D} = Y$, then the output of gate 4 is XY and the output of gate 5 is $\overline{X}\overline{Y}$. If XY and $\overline{X}\overline{Y}$ are OR operated in gate 6, then $XY = \overline{X}\overline{Y} = Y(X + \overline{X}) = Y = B\overline{D}$. Therefore, the above circuit can be simplified as shown in Fig. 15.16 (e).

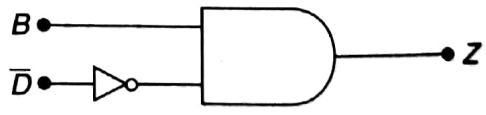


Fig. 15.16 (e)

Example 15.18 Realise (a) $Y = A + BC\overline{D}$ using NAND gates and (b) $Y = (A + C)(A + \overline{D})(A + B + \overline{C})$ using NOR gates.

Solution Using NAND gates

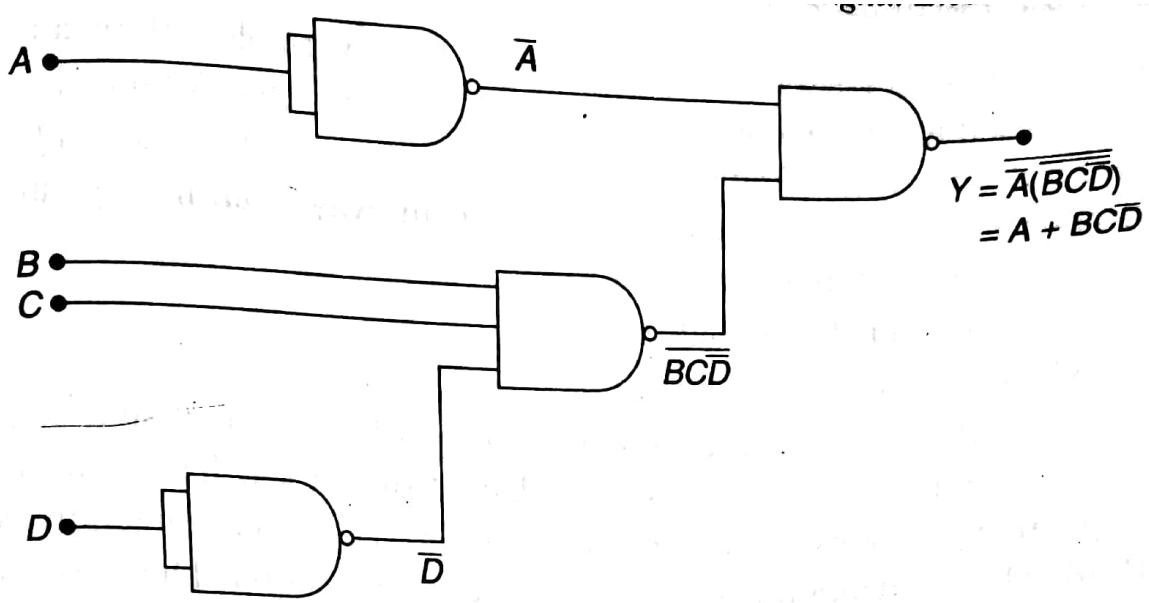


Fig. 15.17 (a)

(b) Using NOR gates

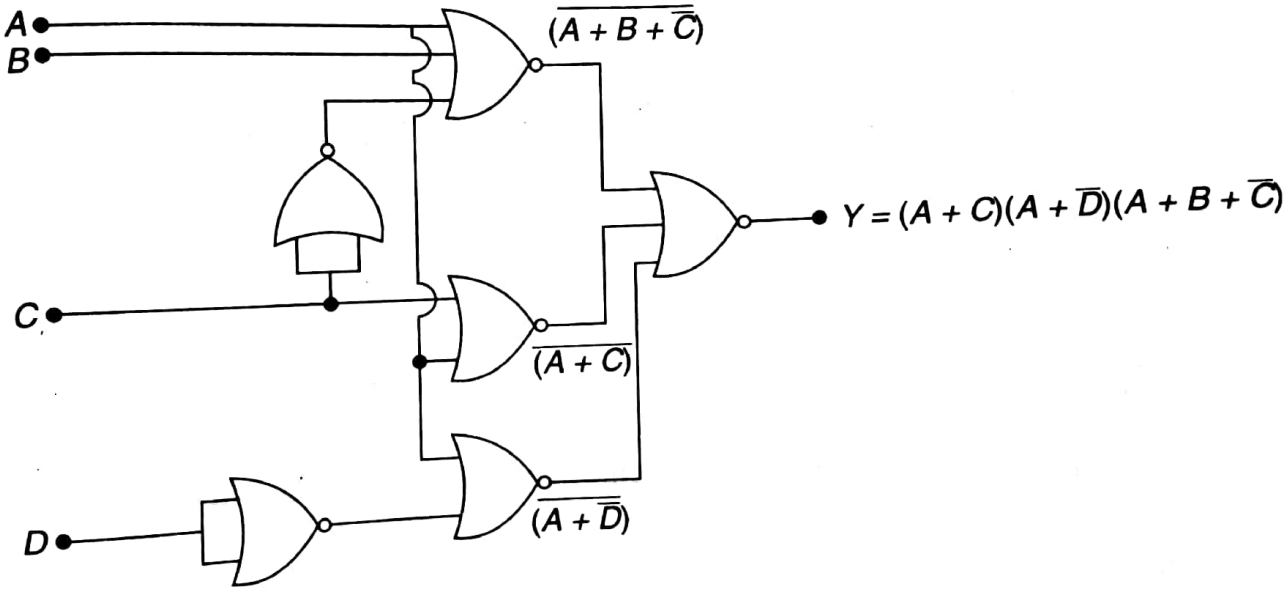


Fig. 15.17 (b)