## 1.1 DIGITAL COMPUTERS AND INFORMATION

A computer is basically a programmable computing machine. Computers are used now a days by all kinds of people for a variety of tasks in a modern and industrialized society. Their presence is felt at almost every working place, namely, homes, schools, colleges, offices, industries, hospitals, banks, retail stores, research and design organizations and so on. Developments in software allow massive applications of computers for non computational jobs like text preparation, manipulation, storage and retrieval, transmission of texts, graphics and pictures from one place to another.

Computers which are in use today are digital computers. They operate on binary digits 0 and 1. They understand information composed of only 0s and 1s. In the case of alphabetic information, the alphabets are coded in binary digits. A binary digit is called bit. A group of 8 bits is called a byte. The most prominent characteristic of a digital computer is its generality. It usually follow a sequence of instructions, called a program, that operates on given data to perform a specific task. The user has the flexibility of manipulating the data or program according to specific needs. Consequently, a digital computer can perform a various information processing tasks that range over a wide spectrum of applications. The general purpose computer is the best known example of a digital system.

Digital system refers to the systems which manipulate discrete quantities of information that are represented in binary form. Any set that contains finite number of elements is said to process discrete information. Examples of discrete sets include 26 alphabetic letters, 10 decimal digits, 52 playing cards and so on. Earlier digital computers were used mostly for numeric computations, wherein discrete elements used were the digits. From such an application, the term digital computer emerged.

In digital systems, the discrete elements of information are represented by physical quantities called signals. The signals use two discrete values, and hence said to be binary. Usually these discrete values are represented by voltage ranges called HIGH and LOW. The other notations used to represent the two discrete values are TRUE and FALSE, ON and OFF, and 1 and 0, Fig. 1.1.1 depicts the input and output voltage ranges.

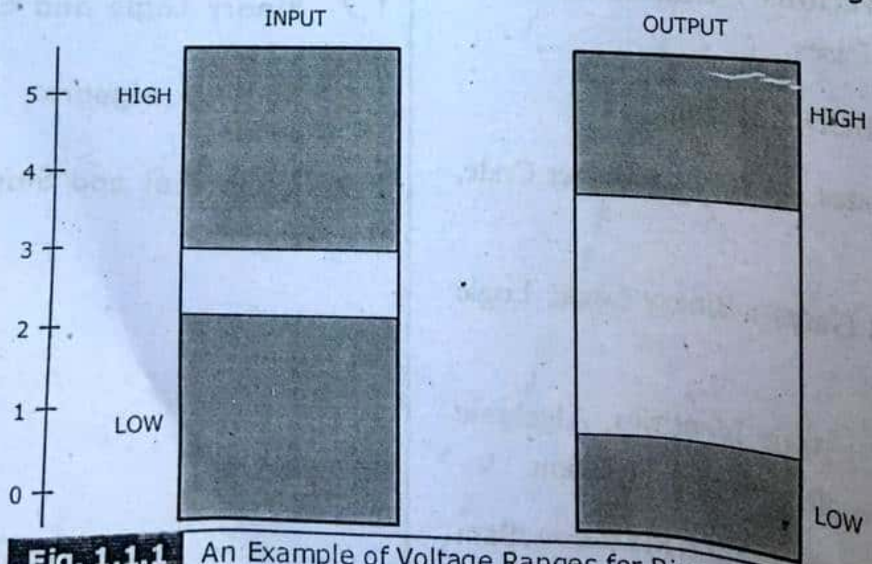


**Fig. 1.1.1** An Example of Voltage Ranges for Binary Signals

The high input voltage allows 3.0 to 5.5 volts to be recognized as a HIGH, and the low input voltage allows −0.5 to 2.0 volts to be recognized as a LOW. The high output voltage ranges between 4.0 to 5.5 volts, and the low output voltage ranges between −0.5 and 1.0 volts.

**NOTE**

*The input and output voltage ranges differs for the various logic families.*

## 1.1.1 Information Representation

In digital computers, information is represented in terms of binary digits 0 and 1. A binary digit (either '0' or '1') is called a bit. Group of bits can be made to represent information like numbers, alphabets and symbols. For this purpose various coding techniques have been used to transform numbers, alphabets and symbols into group of binary digits. A properly arranged group of bits can specify to the computer the instructions to be executed and the data to be processed.

Since digital computers understand information composed of only 0s and 1s and if any analog quantity is to be processed, it must be quantized (i.e., converted into digital) before processing. The quantization of the analog signal in magnitude and time is performed by an analog to digital convertor device.

## 1.1.2 Computer Structure

The block diagram of a digital computer is shown in Fig. 1.1.2.

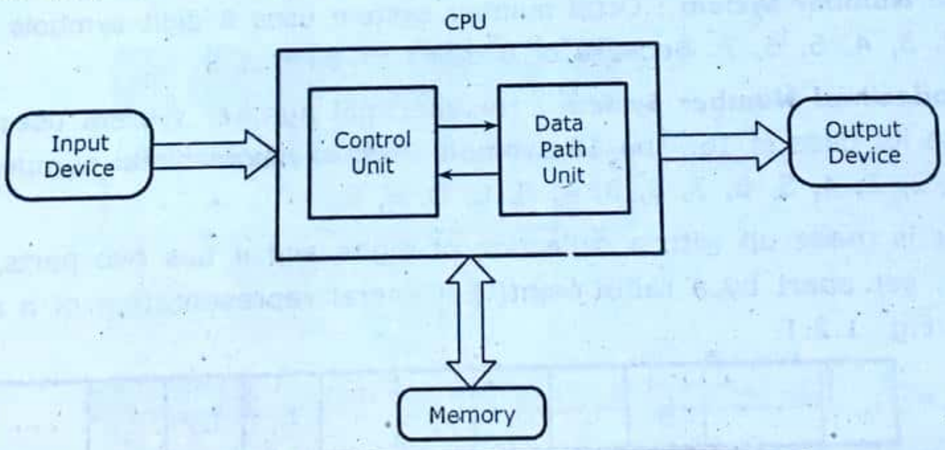


**Fig. 1.1.2** Block Diagram of a Digital Computer

- The central processing unit is the brain of a computer. Its primary function is to execute programs. The CPU also controls the operation of all other components such as memory, input and output devices. The datapath unit of CPU performs arithmetic and data processing operations as specified by program. The control unit provides the control signals to supervise the flow of information between the various components of a computer. The control unit retrieves the instructions, one by one, from the program stored in the memory. For each instruction, the control unit manipulates the datapath to execute the operation specified by the instruction.

m → Number of digits in fractional part.

i → digits in integer part:

f → digits in fractional part

Also, the weight of each digit (either integer or fractional part) can be determined as,

Weight of $i_{n-1}$ = $i_{n-1} \times (\text{base})^{n-1}$

And Weight of $f_{-m}$ = $f_{-m} \times (\text{base})^{-m}$

Here, weight of any digit can be defined as its relative position in number.

## 1.2.1 Decimal Number System

Decimal (or dernary) number system has a base of 10. This means that there are 10 different symbols that make up the number system namely, 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9. All the numbers are given using these basic symbols. Let us consider another number, say 5247.924. The numerical value of above number can be expressed as,

$$\begin{aligned} 5247.924 &= 5000 + 200 + 40 + 7 + 0.9 + 0.02 + 0.004 \\ &= (5 \times 10^3) + (2 \times 10^2) + (4 \times 10^1) + (7 \times 10^0) \\ &\quad + (9 \times 10^{-1}) + (2 \times 10^{-2}) + (4 \times 10^{-3}) \end{aligned}$$

Fig. 1.2.2 illustrates the positional values of digits of a given number relative to the decimal point.

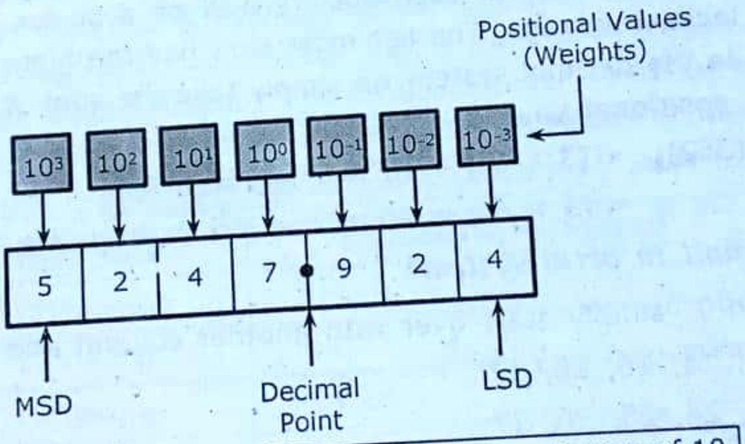


**Fig. 1.2.2** Decimal Positional Values as Powers of 10

Here the integer part (5247) of the number is separated from its fractional part (924) by a point called radix point (or base point). In decimal system, it is called as decimal point.

## 1.2.2 Binary Number System

- The binary number system operates on two voltage levels. The binary number system has a base of 2. This means that this system has only two symbols. They are 0 and 1. Every binary number is given as combination of these two symbols.
- Just like the decimal system, the binary system is also a positional-value system. Each bit has a positional value as a power of 2. Fig 1.2.3 illustrates the positional values for a binary number 1011.10.

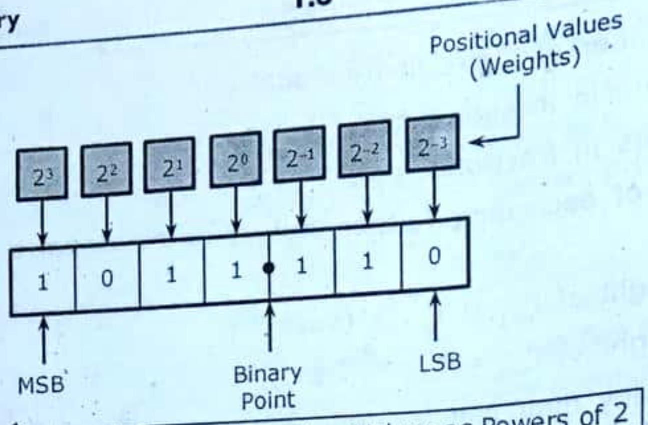


**Fig. 1.2.3** Binary Position Values as Powers of 2

➤ In Fig. 1.2.3, left of binary point (radix point in base-2 system) are positive powers of 2, and right of it are, negative powers of 2. The left most bit has the largest weight, hence called as Most Significant Bit (MSB). The right most bit has the smallest weight, hence called as Least Significant Bit (LSB).

## 1.2.3 Octal Number System

➤ The octal number system has a base of 8. This means that this system has eight symbols. They are 0, 1, 2, 3, 4, 5, 6, and 7. All the numbers given as combination of these symbols.

➤ As in the other number systems discussed, each of these digits has a positional value. In this case the weightage to each position will be a power of 8. The right most digit has the least weightage. The left most digit has the highest weightage. To find its equivalent in the decimal system we simply take the sum of the products of each value and its positional value,

$$(352)_8 = (3 \times 8^2) + (5 \times 8^1) + (2 \times 8^0)$$
$$= (3 \times 6^4) + 40 + 2 = 192 + 40 + 2 = (234)_{10}$$

➤ *How do we count in octal system?*

Once we get to 7, simply start over with another column and continue as follows,

10, 11, 12, 13, 14, 15, 16, 17

20, 21, 22, 23, 24, 25, 26, 27

30, 31, 32, 33, 34, 35, 36, 37 and so on....

## 1.2.4 Hexadecimal Number System

➤ The hexadecimal number system has a base f´16. This means that it has sixteen (Hex + Decimal) symbols. Since, the basic number system consists of only ten digits (from 0 to 9), the first six letters of the English alphabets are borrowed to make the remaining 6 symbols. Hence, the hexadecimal number system is made-up of the symbols, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F.

➤ Every hexadecimal number is given as combination of these symbols. In the above list of symbols, it should be noted that A corresponds to decimal number 10, 'B' to 11, 'C' to 12, 'D' to 13, 'E' to 14 and so finally 'F' to 15.

➢ Let us consider a hexadecimal number 5C. The value of the hexadecimal number 5C is given as,

$$5C = (5 \times 16^1) + (C \times 16^0)$$

Since, 'C' corresponds to the decimal number 12, the value of 5C can be written as,

$$5C = (5 \times 16^1) + (12 \times 16^0)$$

## 1.2.5 Count for Radix (Base)

In any number system, any number can be represented by multiplying each digit with its corresponding positional weights as shown in Fig. 1.2.4.

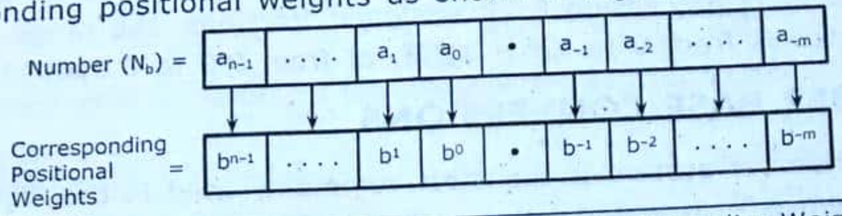


**Fig. 1.2.4** Representation of a Number with its Corresponding Weights

Thus,

$$N_b = a_{n-1}b^{n-1} + \ldots a_1 b^1 + a_0 b^0 + a_{-1}b^{-1} + a_{-2}b^{-2} + \ldots + a_{-m}b^{-m}$$

Where, 'N' is a number and 'b' is the radix or base. In the previous sections, we have seen number systems with radix (base) r equal to 10, 2, 8 and 16. Each number system has 'r' set of characters (symbols). For example, in decimal number system r equals to 10 has 10 characters from 0 to 9, in binary number system r equals to 2 has 2 characters 0 and 1 and so on. In general, we can say that a number represented in radix r, has r characters (symbols) in its set and r can be any integer value greater than one. The set of symbols used in various number systems is illustrated in Table 1.2.1.

**Table 1.2.1** Radix and Symbol Set

| Radix (Base) r | Characters (symbols) in set |
|---|---|
| 2 (Binary) | 0, 1 |
| 3 (Trinary) | 0, 1, 2 |
| 4 (Quinary) | 0, 1, 2, 3 |
| 5 | 0, 1, 2, 3, 4 |
| 6 | 0, 1, 2, 3, 4, 5 |
| 7 | 0, 1, 2, 3, 4, 5, 6 |
| 8 (Octal) | 0, 1, 2, 3, 4, 5, 6, 7 |
| 9 | 0, 1, 2, 3, 4, 5, 6, 7, 8 |
| 10 (Decimal) | 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 |
| 11 | 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, α |
| 12 | 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, α, β |
| 16 (Hexadecimal) | 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F |

## 1.2.6 Number Ranges

The range of numbers that can be represented in digital computers is based on the number of bits available in the hardware structures that store and process information. The number of bits available in these structures is expressed as a power of two, such as $8(=2^3)$bits, $16(=2^4)$ bits, $32(=2^5)$ bits and so on.

The range of numbers that can be handled by a digital computer processing N bit unsigned integers is from 0 to $2^N - 1$. For example, for a computer processing 8 bit unsigned integers is from 0 to $2^8 - 1$, that is, from 0 to 255. If unsigned integers, the range of integers is from 0 to $2^8 - 1$, that is, from 0 to 255. If unsigned integers, the range of fractions that the same computer is processing 8 bit unsigned fractions, the range of fractions that can be represented is from 0 to $(2^8 - 1)/2^8$, or from 0.0 to 0.99609375.

## 1.3 NUMBER BASE CONVERSIONS

The binary number system is the most commonly used in the digital systems, but the decimal system is also important since it is universally used to represent quantities outside a digital system. Therefore, it is necessary to convert decimal number into its equivalent binary while feeding number into the digital system (computer) and to convert the binary number into its decimal equivalent while displaying the result of operation on the output devices. While dealing with the large quantity of binary numbers of many bits, it is inconvenient to read or write. For this reason, octal and hexadecimal numbers are used as a shorthand means of expressing large binary numbers. So it is necessary to have the conversion in between these number systems.

### 1.3.1 Decimal to Radix 'r' System Conversion

There are many ways to convert a decimal number to any radix-r system, but most popular is the double-dabble method. In this method, successive division and successive multiplication by radix(r) is done for integer part and fractional part respectively.

#### 1.3.1.1 Successive Division for Integer Part Conversion

In this method, we repeatedly divide the integer part of the decimal number by radix 'r' until the quotient is zero. The remainder of each division becomes the number in the radix-r number system. The remainders are taken in the reverse order to form a number in base r. That is, the first remainder is the LSD and the last remainder is the MSD in the radix-r number system.

Following are the steps to be followed for integer part conversion,

**Step 1 :** Divide the integer part of the decimal number by radix-'r' and note the values of quotient (Q) and remainder (R).

**Step 2 :** Repeat step 1 until quotient becomes zero.

**Step 3 :** Remainders in reverse order gives the equivalent number in desired radix-r system.

#### 1.3.1.2 Successive Multiplication for Fractional Part Conversion

Following are the steps to be followed for fractional part conversion,

**Step 1 :** Multiply the fractional part of the decimal number by desired radix 'r'.

**Step 2 :** Integer part of the product becomes a digit in the desired radix r system number and fractional part as the new fractional part.

**Step 3 :** Repeat steps 1 and 2 until fractional part is zero or until necessity of digits required.

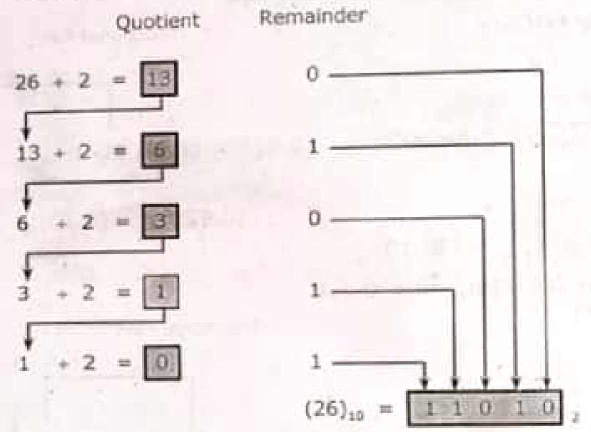
**Step 4 :** Read downwards to represent the equivalent fractional number in radix-r system.

**Example Problem 1.1**

**Convert $(26)_{10}$ into its binary equivalent.**

*Sol.* :

The process of repeated division by 2 is shown below,



Another way of showing the process of repeated division by 2 is as follows,

| | Q | R |
|---|---|---|
| 2 | 26 | |
| 2 | 13 | 0 |
| 2 | 6 | 1 |
| 2 | 3 | 0 |
| | 1 | 1 |

Read Upwards

$\therefore$ $(26)_{10} = (11010)_2$

## 1.3.3 Binary to Octal Conversion

The base for octal numbers (i.e., 8) is the third power of the base for binary numbers (i.e., 2), hence the group three bits to form an equivalent octal number. Table 1.3.1 gives the grouping of 3 bits to form its equivalent octal number.

**Table 1.3.1** Binary to Octal Conversion

| **Binary Equivalent** | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
|---|---|---|---|---|---|---|---|---|
| **Octal Digit** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

**Example Problem 1.6**

**Convert 1110101 . 1010011 into octal number system.**

*Sol.* :

The grouping of three binary digits can be done as follows,

**Step 1 :** For integer part grouping of 3 bits starts from LSB to MSB. For fractional part grouping of 3 bits starts from MSB to LSB.

$$\underset{\substack{\uparrow\\ \text{MSB}}}{1}\ \underbrace{1\,1\,0}\ \underbrace{1\,0\,1}_{\substack{\uparrow\\ \text{LSB}}}\ .\ \underbrace{1\,0\,1}_{\substack{\uparrow\\ \text{MSB}}}\ \underbrace{0\,0\,1}\ \underset{\substack{\uparrow\\ \text{LSB}}}{1}$$

**Step 2 :** Add zeros for making the groups of 3 bits. For integer part, add zeros on left of MSB and for fractional part, add zeros on right of LSB.

$$\underbrace{0\,0\,1}_{1}\ \underbrace{1\,1\,0}_{6}\ \underbrace{1\,0\,1}_{5}\ .\ \underbrace{1\,0\,1}_{5}\ \underbrace{0\,0\,1}_{1}\ \underbrace{1\,0\,0}_{4}$$

$$\therefore\ (1110101\ .\ 1010011)_2 = (165.514)_8$$

## 1.3.4 Binary to Hexadecimal Conversion

The base for haxadecimal numbers (i.e., 16) is the fourth power of the base of binary numbers (i.e., 2), hence group four bits to form an equivalent hexadecimal number.

Table 1.3.2 gives the hexadecimal decimal equivalent of group of four binary digits.

**Table 1.3.2** Binary to Hexadecimal Conversion

| **Binary Equivalent** | 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 |
|---|---|---|---|---|---|---|---|---|
| **Hexadecimal Digit** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

| **Binary Equivalent** | 1000 | 1001 | 1010 | 1011 | 1100 | 1101 | 1110 | 1111 |
|---|---|---|---|---|---|---|---|---|
| **Hexadecimal Digit** | 8 | 9 | A | B | C | D | E | F |

**Example Problem 1.7**

**Convert 1110101 . 1010011 into hexadecimal number.**

*Sol.* :

The conversion of binary number to hexadecimal number is done as follows,

**Step 1 :** For integer part, grouping of 4 bits starts from LSB to MSB. For fractional part grouping of 4 bits starts from MSB to LSB.

$$\underbrace{1\,1\,1}_{\substack{\uparrow\\ \text{MSB}}}\ \ \underbrace{0\,1\,0\,1}_{\substack{\uparrow\\ \text{LSB}}}\ .\ \underbrace{1\,0\,1\,0}_{\substack{\uparrow\\ \text{MSB}}}\ \ \underbrace{0\,1\,1}_{\substack{\uparrow\\ \text{LSB}}}$$

**Step 2 :** Add zeros for making the group of 4 bits. For integer part, add zeros on left of MSB and for fractional part add zeros on right of LSB.

$$\underbrace{0\,1\,1\,1}_{7}\ \ \underbrace{0\,1\,0\,1}_{5}\ .\ \underbrace{1\,0\,1\,0}_{A}\ \ \underbrace{0\,1\,1\,0}_{6}$$

$$\therefore\ (1110101\ .\ 1010011)_2 = (75.A6)_{16}$$

## 1.3.5 Octal to Binary Conversion

Octal numbers are converted into its equivalent binary numbers by replacing each octal digit with its equivalent binary number.

**Example Problem 1.8**

**Convert $(567.32)_8$ into binary number.**

*Sol.* :

Replacing each octal digit by 3 bit binary equivalent.

$$\begin{array}{cccccc} 5 & 6 & 7 & . & 3 & 2\\ \downarrow & \downarrow & \downarrow & & \downarrow & \downarrow\\ 1\,0\,1 & 1\,1\,0 & 1\,1\,1 & & 0\,1\,1 & 0\,1\,0 \end{array}$$

$$\therefore\ (567.32)_8 = (101110111\ .\ 011010)_2$$

## 1.3.6 Octal to Hexadecimal Conversion

The conversion of octal number to hexadecimal number is done in two steps and is given below,

**Step 1 :** Convert the octal number to its binary equivalent (octal-to-binary conversion).

**Step 2 :** Convert the binary number to its hexadecimal equivalent (binary-to-hexadecimal conversion).

Another method is to first convert the given octal number into decimal and then to hexadecimal.

**Example Problem 1.9**

**Convert $(675.42)_8$ into base 16 number.**

*Sol.* :

**Step 1 : Octal-to-binary Conversion**

$$\begin{array}{ccccccc} 6 & 7 & 5 & . & 4 & 2 \\ \downarrow & \downarrow & \downarrow & & \downarrow & \downarrow \\ 110 & 111 & 101 & & 100 & 010 \end{array}$$

$\therefore$ Binary equivalent = $(110111101.100010)_2$

**Step 2 : Binary-to-Hexadecimal Conversion**

$$\underset{1}{\underline{0001}}\ \ \underset{B}{\underline{1011}}\ \ \underset{D}{\underline{1101}}\ .\ \underset{8}{\underline{1000}}\ \ \underset{8}{\underline{1000}}$$

$\Rightarrow$ $(1BD.88)_{16}$ = 1BD.88H

## 1.3.7 Hexadecimal to Binary Conversion

Hexadecimal numbers are converted into equivalent binary numbers by replacing each hexadecimal digit with its equivalent binary number.

**Example Problem 1.10**

**Convert $(567.A4)_{16}$ into binary number.**

*Sol.* :

Replacing each hexadecimal digit by 4 bit binary equivalent.

$$\begin{array}{cccccc} 5 & 6 & 7 & . & A & 4 \\ \downarrow & \downarrow & \downarrow & & \downarrow & \downarrow \\ 0101 & 0110 & 0111 & & 1010 & 0100 \end{array}$$

$\therefore$ $(567.A4)_{16}$ = $(10101100111\ .\ 101001)_2$

## 1.3.8 Hexadecimal to Octal Conversion

The simplest way to convert hexadecimal number to octal number contains two steps,

**Step 1 :** Convert the hexadecimal number to its binary equivalent.

**Step 2 :** Convert binary number to its octal equivalent.

Another method is to convert the given hexadecimal number into decimal and then decimal to the octal number.

**Example Problem 1.11**

**Convert $(AB6.13)_{16}$ into octal equivalent.**

*Sol.* :

**Step 1 :** Convert the hexadecimal number to its binary equivalent.

$$\begin{array}{cccccc} A & B & 6 & & 1 & 3 \\ \downarrow & \downarrow & \downarrow & & \downarrow & \downarrow \\ 1010 & 1011 & 0110 & . & 0001 & 0011 \end{array}$$

$\therefore$ Binary equivalent = $(101010110110.00010011)_2$

**Step 2 :** Convert the binary number to its octal equivalent.

Add one 0 to form complete group

$$\underbrace{1\,0\,1}_{5}\ \underbrace{0\,1\,0}_{2}\ \underbrace{1\,1\,0}_{6}\ \underbrace{1\,1\,0}_{6}\ .\ \underbrace{0\,0\,0}_{0}\ \underbrace{1\,0\,0}_{4}\ \underbrace{1\,1\,\boxed{0}}_{6}$$

$\therefore$ Octal equivalent= $(5266.046)_8$

## 1.4 ARITHMETIC OPERATIONS

The rules used for performing arithmetic operations for decimal numbers is also applicable for arithmetic operations with numbers in base-r. However, one must be careful to use only 'r' allowable digits and perform all computations with base-r digits.

### 1.4.1 Binary Arithmetic Operations

The arithmetic rules for the addition, subtraction, multiplication and division of binary number can be done by using the Table 1.4.1.

**Table 1.4.1** Arithmetic Rules for Addition, Subtraction, Multiplication and Division

| S.No. | Addition | Subtraction | Multiplication | Division |
|---|---|---|---|---|
| 1) | 0 + 0 = 0 | 0 − 0 = 0 | 0 × 0 = 0 | 0 ÷ 0 = NA * |
| 2) | 0 + 1 = 1 | 1°0 − 1 = 1 | 0 × 1 = 0 | 0 ÷ 1 = 0 |
| 3) | 1 + 0 = 1 | 1 − 0 = 1 | 1 × 0 = 0 | 1 ÷ 0 = NA * |
| 4) | 1 + 1 = 1'0 | 1 − 1 = 0 | 1 × 1 = 1 | 1 ÷ 1 = 1 |
|  | (a) | (b) | (c) | (d) |

'carry generated; °borrow taken; *Not Allowed

#### 1.4.1.1 Binary Addition

Addition of two binary numbers can be performed in the same way as the decimal numbers are added. The addition is started from the LSB and proceeds to higher significant bits, adding the carry results from the previous additions. Consider the addition of two binary numbers 1110 and 1010.

### 1.4.1.4 Binary Division

It is similar to the division procedure in decimal number system. If we are dividing any number by 0, is meaningless. The process of binary division can be defined as,

**Step 1 :** If the divisor is less than the dividend then subtract the divisor repeatedly from the dividend.

**Step 2 :** If the divisor is greater than the dividend then put a decimal in quotient and dividend; and increase the number of 0 in dividend.

**Example Problem 1.13**

**Divide the following**

**a) 11110 ÷ 101**

**b) 1111 ÷ 110**

*Sol.* :

**(a) 11110 ÷ 101**

```
       110
 101)11110
     101
     0101      |
      101      |
      000      v
      000
      000
```

**(b) 1111 ÷ 110**

```
      10.1
 110)1111.0
     110
     00110     |
       110     |
       000     v
```

In the example problem 1.13(b) division is not possible, so after placing 0 in the quotient, put decimal in the dividend and the quotient both. Then increase the 0's required in dividend.

## 1.4.2 Arithmetic Operations with Octal, Hexadecimal System

### 1.4.2.1 Octal Addition

Addition of octal number is carried out in the same way as the decimal addition is performed. The steps are given below,

**Step 1 :** First, add the least significant digits of the octal number in decimal.

**Step 2 :** During the process of addition, if the sum is less than or equal to 7, then it can be directly written as a octal digit.

**Step 3 :** If the sum is greater than 7, then subtract 8 from the digit and carry 1 to the next digit position.

**Step 4 :** This process is repeated for each larger significant digit of the octal number.

**Step 5 :** Carry generated from most significant bits addition will be included in the result.

**Comment**

*In this addition, we should remember that the largest octal digit is 7 instead of 9 in case of decimal system.*

**Example Problem 1.14**

**Add $(432)_8$ and $(746)_8$**

*Sol.* :

```
                                                   1   1   1    <---- Carry
                                       Augend =        4   3   2
                                      Addend  =        7   4   6
                Sum (Using decimal number system) =   1   2  8   8
Subtract 8 from digit whose sum is greater than 7 =  (-)  8   8   8
                                       Result =    1   4   0   0
```

$\therefore$ $(432)_8 + (746)_8 = (1400)_8$

**Comment**

*Carry generated from sum of most significant octal digits will be MSB in the result.*

**Example Problem 1.15**

**Add $(6352)_8$ and $(5143)_8$**

*Sol.* :

```
                                           1       1          <---- Carry
                             Augend =          6   3   5   2
                             Addend =          5   1   4   3
           Add digits using decimal system =  1  1   5   9   5
    Subtract 8 from digit whose sum is > 7 =      8       8
                             Result =      1   3   5   1   5
```

$\therefore$ $(6352)_8 + (5143)_8 = (13515)_8$

In the above example, only most significant digits and 2nd least significant dig have sum greater than 8 (i.e.,) 11, 9 respectively. Hence 8 is subtracted from the respective positions only.

### 1.4.2.2 Octal Subtraction

In the octal subtraction, the method, which we have adopted, is similar to that of binary subtraction method. The only difference lies in the carry part. During octal subtraction, instead of 1, we will borrow 8 and the rest of the steps are similar to that of binary subtraction.

**Example Problem 1.16**

**Subtract $(73)_8 - (64)_8$**

*Sol.* :

```
                 [8] ← Borrow          Decimal Equivalent
   Minuend    =   7  3                       5 9
   Subtrahend =   6  4        ⇨              5 2
   Resul      =      7                         7
```

$\therefore (73)_8 - (64)_8 = (7)_8$

**Example Problem 1.17**

**Subtract $(565)_8 - (376)_8$**

*Sol.* :

```
              [8]  [8] ← Borrow          Decimal Equivalent
   Minuend    =  5   6   5                    3 7 3
   Subtrahend =  3   7   6       ⇨            2 5 4
   Result     =  1   6   7                    1 1 9
```

$\therefore (565)_8 - (376)_8 = (167)_8$

### 1.4.2.3 Hexadecimal Addition

The addition operation performed with the hexadecimal numbers is similar to the decimal addition except with a few differences that are discussed in the following steps.

**Step 1 :** First, add the unit column of the hexadecimal digits in decimal.

**Step 2 :** During the process of addition, observe it the sum is 15 or less than it can be directly expressed as a hexadecimal digit.

**Step 3 :** If the sum is greater than 15, then subtract 16 from the particular digit and carry 1 to the next digit position.

**Step 4 :** This process is repeated for each larger significant digit of the hexadecimal number.

**Step 5 :** Carry generated from most significant bits addition will be included in the result.

**Example Problem 1.18**

**Add $(438)_{16}$ and $(538)_{16}$**

*Sol.* :

```
                                              1      ← Carry
                             Augend =     4   3   8
                             Addend = (+) 5   3   8
        Add digits using decimal system = 9   7  16
Subtract 16 from digits whose SUM > 15 = (-)     16
                             Result =     9   7   0
```

$\therefore (438)_{16} + (538)_{16} = (970)_{16}$

$\therefore (1A)_{16} + (2B)_{16} = (45)_{16}$

**Example Problem 1.19**

**Add $(3AF)_{16}$ and $(2BC)_{16}$**

*Sol.* :

In the given example, convert the values of A, B, C and F to their respective decimal equivalent and then proceed addition.

```
                                          1   1      ← Carry
                             Augend =     3   A   F
                             Addend = (+) 2   B   C
        Add digits using decimal system = 6  22  27
Subtract 16 from digits whose SUM > 15 =     16  16
                             Result =     6   6  11
                                                  B
```

Here F → 15
C → 12
So, F + C = 27
Also, A → 10
B → 11
So, A + B = 21

$\therefore (3AF)_{16} + (2BC)_{16} = (66B)_{16}$

### 1.4.2.4 Hexadecimal Subtraction

The hexadecimal subtraction is based on the same principles as that of binary subtraction. In this subtraction, 16 will be used as borrow instead of 1. The rest of the steps are similar to the binary subtraction.

**Example Problem 1.20**

**Subtract $(37)_{16}$ from $(4B)_{16}$**

*Sol.* :

```
   Minuend    =  4  B
   Subtrahend =  3  7
   Result     =  1  4
```

Here B → 11.
So, B - 7 = 4

$\therefore (4B)_{16} - (37)_{16} = (14)_{16}$

**Example Problem 1.21**

**Subtract $(4B)_{16}$ from $(54)_{16}$**

*Sol.* :

$$\begin{array}{rcc} & \overset{16}{} & \leftarrow \text{Borrow} \\ \text{Minuend} = & 5 & 4 \\ \text{Subtrahend} = & 4 & B \\ \hline \text{Result} = & & 9 \end{array}$$

Here 16 + 4 = 20
B → 11
So, 20 − 11 = 9

$\therefore$ $(54)_{16} - (4B)_{16} = (9)_{16}$

**Example Problem 1.22**

**Subtract $(6AB2)_{16}$ and $(5BF3)_{16}$**

*Sol.* :

$$\begin{array}{rcccc} & & 16 & 16 & 16 \leftarrow \text{Borrow} \\ \text{Minuend} = & 6 & A & B & 2 \\ \text{Subtrahend} = & 5 & B & F & 3 \\ \hline \text{Result} = & & E & B & F \end{array}$$

The above example explained as follows,

At LSB (1st bit) position : $(16 + 2) - 3 = 15 = F$

At 2nd bit position : Since B gives borrow of 16 to LSB (1st bit) hence now B will be one value less (i.e.,) $B \Rightarrow B - 1 = A$.

$\therefore$ $(16 + A) - F = (16 + 10) - 15 = 26 - 15 = 11 = B$

At 3rd bit position : A gives borrow to 2nd bit, hence A will be one value less (i.e.,) $A \Rightarrow A - 1 = 9$.

$\therefore$ $(16 + 9) - B = (25) - 11 = 14 = E$

At 4th bit position : Similarly 6 gives borrow to 3rd bit, hence 6 will be one value less (i.e.,) $6 \Rightarrow 6 - 1 = 5$. Thus $5 - 5 = 0$

$\therefore$ $(6AB2)_{16} - (5BF3)_{16} = (EBF)_{16}$

## 1.5 DECIMAL CODES

➤ *The electronic digital systems like computers, microprocessors, .etc., are required to handle data (represent and manipulate) which may be consisting of numbers, alphabets or special characters. But the digital systems use signals that have two distinct values and circuit elements that have two stable states. Hence the numerals, alphabets, special characters and control functions are to be converted into binary format.*

➤ The process of conversion into binary format is known as 'binary coding'. The combination of binary bits that represent numbers, (may be decimal numbers, octal, hexadecimal numbers etc.), alphabets, symbols or control functions are called 'binary codes (or) decimal codes'. When decimal number is represented by its binary equivalent binary, we call it as "straight binary coding".

➤ An n-bit binary code is a group of n bits that can represent upto $2^n$ distinct combinations of 1's and 0's. For example, a set of four elements can be coded with a 2-bit binary code, with each element assigned one of the following bit combinations 00, 01, 10, 11.

➤ If the number of elements in the set is not a power of 2, then binary code may have some unused bit combinations. One such decimal codes is binary coded decimal (BCD). In BCD, decimal digits 0 through 9 are represented by their binary equivalents using four bits.

Table 1.5.1 shows the 4 bit 8421 or BCD-code used to represent a single decimal digit.

**Table 1.5.1** 8421 or BCD Code

| Decimal Digit | BCD Code | | | |
|---|---|---|---|---|
| | **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 |

**With four bits, $2^4$ i.e., sixteen numbers can be represented. But in this code, we are using only ten. The result of six codes combinations are not used i.e., 1010 to 1111 are invalid in this BCD code.**

In multi digit coding, each decimal digit is individually coded with BCD code. For example, 67.8 in decimal can be encoded in BCD as,

$$\begin{array}{cccc} 6 & 7 & \bullet & 8 \\ \downarrow & \downarrow & & \downarrow \\ 0110 & 0111 & \bullet & 1000 \end{array}$$

Though BCD numbers are represented in bits, they are decimal numbers and not binary numbers. One key distraction between a BCD number and a decimal number is that the BCD numbers use the binary codes 0000, 0001, 0010,...., 1001, while the decimals are written with the symbols 0, 1, 2, 3,...., 9.

**Example Problem 1.23**

**Convert the given binary number 11011.101 into BCD number system.**

*Sol.* :

Any given binary number can be converted into its equivalent BCD by first converting it into decimal number and then each decimal is coded individually with four bits.

➢ $(11011.101)_2 \rightarrow (?)_{10}$

**Step 1 :** $1 \quad 1 \quad 0 \quad 1 \quad 1 \quad . \quad 1 \quad 0 \quad 1$

**Step 2 :** $2^4 \quad 2^3 \quad 2^2 \quad 2^1 \quad 2^0 \quad . \quad 2^{-1} \quad 2^{-2} \quad 2^{-3}$

**Step 3 :** $1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}$

$= 16 + 8 + 2 + 1 + 0.5 + 0.125 = 27.625$

➢ $(27.625)_{10} \rightarrow (\ )_{BCD}$

$$\begin{array}{cccccc} 2 & 7 & \bullet & 6 & 2 & 5 \\ \downarrow & \downarrow & & \downarrow & \downarrow & \downarrow \\ 0010 & 0111 & & 0110 & 0010 & 0101 \end{array}$$

$\therefore \ (11011.101)_2 \rightarrow (00100111.011000100101)_{BCD}$

## 1.5.1 BCD Addition

Procedure for BCD addition is as follows,

**Step 1 :** Add the two-binary numbers of BCD similar to binary addition.

**Step 2 :** If the 4-bit group sum is ≤ 9, it a valid BCD.

**Step 3 :** If the 4-bit group sum is > 9, or if carry is generated, it is not a valid BCD. Add 6 i.e., $(0110)_2$ to the 4-bit group sum to skip the six invalid states and return to the code 8421.

Notice that if the 4-bit group sum results into a carry, the carry is added in the next significant 4-bit sum group.

**Case 1 (Sum Equals 9 or Less) :** Consider adding 4 and 3 using BCD to represent each digit.

$$\begin{array}{rl} \text{BCD code for 4} = & 0100 \\ \text{BCD code for 3} = & 0011 \\ \hline \text{Sum} = & \underset{\text{BCD code for 7}}{0111} \end{array} \Longrightarrow \begin{array}{r} 4 \\ 3 \\ \hline 7 \end{array}$$

The addition is carried out as in normal binary addition and the sum is 0111 which is the BCD code for 7. As another example, take 45 added to 33.

$$\begin{array}{rl} \text{Augend in BCD} = & 0100\ 0101 \\ \text{Addend in BCD} = (+) & 0011\ 0011 \\ \hline \text{Sum} = & \underbrace{0111}_{7}\ \underbrace{1000}_{8} \end{array} \Longrightarrow \begin{array}{r} 45 \\ 33 \\ \hline 78 \end{array}$$

In this example, the four-bit codes for 5 and 3 are added in binary to produce 1000, which is BCD for 8. Similarly, adding the second-decimal-digit positions produces 0111, which is BCD for 7. The total is 01111000, which is the BCD code for 78.

In the examples above, none of the sums of the pairs of decimal digits exceeded 9, therefore, no decimal carries were produced. For these cases, the BCD addition process is straightforward and is actually the same as binary addition.

**Case 2 (When Sum Greater than 9 with No Carry) :** In this case, sum of two digits greater than 9 will result an invalid BCD number. Whenever this arises, add $(6)_{10} \rightarrow 0110$ to that corresponding sum term.

**Example Problem 1.24**

**Add 368.5 and 534.6 in BCD form**

*Sol.* :

$$\begin{array}{rcccccc} \text{Augend in BCD} = & 0011 & 0110 & 1000 & . & 0101 \\ \text{Addend in BCD} = & 0101 & 0011 & 0100 & . & 0110 \\ \hline \text{ADD } 0110_2 \text{ to illegal codes} = & 1000 & 1001 & 1100 & . & 1011 \\ & & & 0110 & . & 0110 \\ \hline \text{Add } 0110_2 \text{ to illegal code} = & 1000 & 1010 & 0011 & . & 0001 \\ & & 0110 & & & \\ \hline \text{Correct result} = & \underbrace{1001}_{9} & \underbrace{0000}_{0} & \underbrace{0011}_{3} & . & \underbrace{0001}_{1} & = 903.1 \end{array}$$

**Case 3 (When Sum Greater than 9 with Carry) :** If there is a carry out of one sum term or if the sum term is an illegal code, then add $6_{10}$ = $(0110)_2$ to the sum for correction.

**Example Problem 1.25**

**In BCD Formate Add 59 and 38.**

*Sol. :*

```
                                         1
             BCD code for 59 =      0101 | 1001         59
             BCD code for 38 = (+)  0011 | 1000   ==>   38
                                   ------------         ---
Carry propagated to next group =    1001 (1) 0001       97
             ADD 6 for correction =          0110
             Result in BCD =        1001     0111
                                    \__/     \__/
                                     9        7
```

In the above example, least significant digits (i.e.,) 9, 8 addition produces a sum of 10001 = 17. Thus generates a carry into the next digit position to be added to the codes for 5 and 3. Also, since 17 > 9, a correction factor of +6 is added to the LSB sum. Addition of this correction does not generate a carry. The carry was already propagated in the original addition.

## 1.6 ALPHANUMERIC CODES

The codes which represent numbers, alphabets (both upper and lower case) and special symbols are called as alphanumeric codes.

### 1.6.1 ASCII Character Code

An input output code known as the American Standard Code for Information Interchange (pronounced as ask-ee) used in most computers by its manufacturer. It represents a character set with 7 bits, which can be stored as one byte, with an extra unused bit. The ASCII is a 7 bit code whose format is $X_6$ $X_5$ $X_4$ $X_3$ $X_2$ $X_1$ $X_0$, where each bit is either 0 or 1.

***Example :*** *A is coded as 100 0001. The Table 1.6.1 shows the capital letter A has an $X_6$ $X_5$ $X_4$ of 100 and $X_3$ $X_2$ $X_1$ $X_0$ of 0001.*

Table 1.6.1 ASCII Character Code

| $X_3 X_2 X_1 X_0$ | $X_6 X_5 X_4$ | | | | | |
|---|---|---|---|---|---|---|
| | 010 | 011 | 100 | 101 | 110 | 111 |
| 0000 | SP | 0 | @ | P | | P |
| 0001 | ! | 1 | A | Q | a | q |
| 0010 | " | 2 | B. | R | b | r |
| 0011 | # | 3 | C | S | c | s |
| 0100 | $ | 4 | D | T | d | t |
| 0101 | % | 5 | ! | U | e | u |
| 0110 | & | 6 | F | V | f | v |
| 0111 | ' | 7 | G | W | g | w |
| 1000 | ( | 8 | H | X | h | x |
| 1001 | ) | 9 | I | Y | i | y |
| 1010 | * | : | J | Z | j | z |
| 1011 | + | ; | K | | k | |
| 1100 | ' | < | L | . | l | |
| 1101 | ? | = | M | | m | |
| 1110 | ? | > | N | | n | |
| 1111 | / | ? | O | | o | |

### 1.6.2 Parity Bit

In the process of binary data transmission, errors may occur. In order to detect errors, sometimes an extra bit, known as parity bit is added to each ASCII character being transmitted. The parity bit is added to ASCII character code to make the total number of 1's either even or odd. The two ASCII characters with their even and odd parity are as shown in Table 1.6.2.

Table 1.6.2 ASCII Characters with Even and Odd Parity

| ASCII Character | With Odd Parity | With Even Parity |
|---|---|---|
| A = 1000001 | 11000001 | 01000001 |
| X = 1011000 | 01011000 | 11011000 |

Table 1.7.2 Truth Table for 2-input AND Gate

| Inputs | | Output |
|---|---|---|
| A | B | F |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

From Table 1.7.2, it can be noticed that the logical AND operation is such that the "output is HIGH only when all of the inputs are HIGH. When any of the inputs are LOW, the output is LOW".

The logic operation AND can be represented by a switching-circuit as shown in Fig. 1.7.2. The lamp glows (logic 1) when both switches $S_1$ and $S_2$ are closed (logic 1). Obviously the lamp is OFF (logic 0) if any one or both switches are OPEN (logic 0).

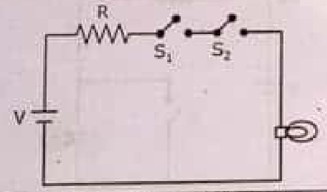


Fig. 1.7.2 Switching Circuit Implementing AND Gate

## 1.7.1.3 Logical OR Operation

The logical OR operation is performed on a two or multiple input variables. The logical OR operation between two inputs variables A and B can be represented as,

$$F = A + B$$

The common symbol used for logical OR operation is addition symbol (+). Table 1.7.3 shows the result of OR operation on the variables A and B.

Table 1.7.3 2-input OR Gate Truth Table

| Inputs | | Output |
|---|---|---|
| A | B | F |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

From Table 1.7.3 logical OR operation can be simply stated as "The output is HIGH when one or more inputs are HIGH". The logical OR operation can be represented by a switching circuit as shown in Fig. 2.2.3. The lamp glows when one or both switches $S_1$ and $S_2$ are closed. The lamp is OFF when both the switches are open.

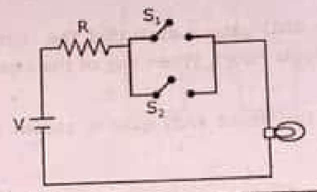


Fig. 1.7.3 Logic Operation for 2-input OR Gate

## 1.7.2 Logic Gates

- Logic gates are the functional building blocks of digital systems. Logic gate is an electronic circuit that operates on one or more input signals to produce an output signal.
- The inputs and outputs of logic gates can occur only in two levels. These two levels are termed as HIGH and LOW or TRUE and FALSE or ON and OFF or simply 1 and 0. The output level (logic HIGH or logic LOW) depends upon the combinations of input levels. Any logic gate operation can be understood with the help of Truth Table.
- Digital electronics deals with different types of logic gates. They are,
  - OR, AND, NOT gates (also known as basic gates) which are mostly used in combinational logic design.
  - NAND, NOR gates (also known as universal gates) because all the other gates can be realized using either, only by NAND gates or only by NOR gates.
- **NOT Gate**
  - NOT operation is performed on a single input variable. If A is an input variable to the NOT gate, then the output (F) can be expressed as,

    $$F = \bar{A} \text{ (or) } A'.$$

    This expression is read as, "F equals NOT A" or "F equals the inverse of A" or "F equals the complement of A".
  - NOT operation is also referred as "inversion or complementation". It changes a logic 1 to a logic 0 and a logic 0 to a logic 1.
  - Fig. 1.7.4(a) shows the symbol for NOT gate, which is more commonly called as "INVERTER". It always has a single input and single output.

➤ **AND Gate**

- The AND gate has one output and two or more inputs. If two inputs A and B are combined using the AND operation, the result F, can be represented as,

$$F = A.B$$

- The operation of the AND gate is such that the "output is logic HIGH only when all of the inputs are logic HIGH. When any of the inputs are logic LOW, the output is logic LOW".
- The logic symbol of two input AND gate is shown in Fig. 1.7.4(b).

➤ **OR Gate**

- The OR gate has one output and two or more inputs. Let A and B represent two independent input logic variables to the OR gate, the output F can be represented as,

$$F = A + B$$

- An OR gate produces a logic 1 (HIGH) output when any of the inputs is at logic 1. The output is logic 0 (LOW) only when all the inputs are at logic 0. Simply logical OR operation is stated as "the output is HIGH when one or more inputs are HIGH".
- The logic symbol for the two input OR gate is shown in Fig. 1.7.4(c).

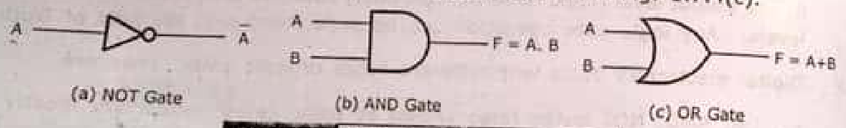


**Fig. 1.7.4** Logic Symbols of Basic Gates

➤ The operation of these logic gates can be illustrated using truth tables as shown in Table 1.7.4.

**Table 1.7.4** Truth Table of Basic Logic Gates

(a) Not Gate

| Input (A) | Output (F) |
|---|---|
| 0 | 1 |
| 1 | 0 |

(b) AND Gate

| Inputs | | Output |
|---|---|---|
| A | B | F |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

(c) OR Gate

| Inputs | | Output |
|---|---|---|
| A | B | F |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

➤ **Timing Diagrams of Basic Logic Gates :** Fig. 1.7.5 shows the timing diagram for the corresponding output signal for each type of gate. The horizontal axis of a timing diagram represents time and the vertical axis shows a signal as it changes between the two possible voltage levels.

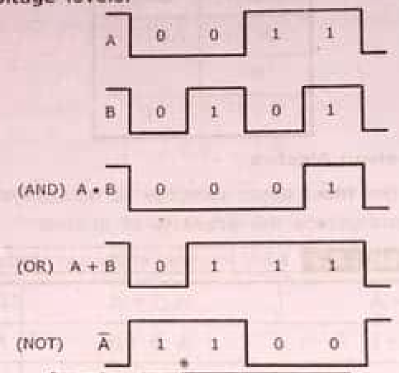


**Table 1.7.5** Timing Diagram

## 1.8 BOOLEAN ALGEBRA

'Boolean algebra' is a set of rules, laws and theorems by which logical operations can be mathematically expressed. Among all the Boolean algebras, the two-element Boolean algebra $B_2$ ({0, 1}, ., +, ', 0, 1) known as "switching algebra" is used for analysis and design of switching circuits that make up digital systems. It contains two extreme elements, the largest number represented by 1 and the smallest number represented by 0. Boolean functions defined on the switching algebra are called 'switching functions'.

Boolean Algebra is different from the normal algebra in many ways, such as :

1) Boolean algebra deals with finite elements while normal algebra deals with real numbers, which are infinite.
2) Complement operator is available in Boolean algebra.
3) Boolean algebra obeys the distributive law i.e., A + (B.C) = (A + B).(A + C).

### 1.8.1 Basic Identities

It is often possible to obtain a simpler expression for the Boolean function by manipulating a Boolean expression according to Boolean algebra rules. This simpler expression reduces the number of logic gates and the number of inputs to gates. In this section we will study the basic rules of Boolean algebra necessary to obtain the simple expressions for Boolean function.

➤ **Principle of Duality :** Duals are opposites or mirror images of original operators or constants. In Boolean algebra, we can produce dual by changing all '+' signs to '.' signs, all '.' to '+' signs and complementing all 0's and 1's. Keep a note that the variables are not complemented in this procedure.

**Table 1.8.1** Duals for Binary Operations

| Operator | Dual |
|---|---|
| AND (+) | OR (.) |
| OR (.) | AND (+) |
| 0 | 1 |
| 1 | 0 |

➢ **Basic Identities of Boolean Algebra**

- Table 1.8.2 lists the most basic identities of Boolean algebra arranged into two columns that demonstrate the property of duality.

**Table 1.8.2** Basic Identities of Boolean Algebra

| | | |
|---|---|---|
| $A = 0 = A$ | $A\,.1 = A$ | Identity |
| $A + 1 = 1$ | $A\,.0 = 0$ | Null |
| $A + A = A$ | $A\,.\,A = A$ | Idempotency |
| $A + \overline{A} = 1$ | $A\,.\,A = 0$ | Complement |
| $\overline{\overline{A}} = A$ | | Involution |
| $A + B = B + A$ | $AB = BA$ | Commutative |
| $A + (B + C) = (A + B) + C$ | $A(BC) = (AB)Z$ | Associative |
| $A(B + C) = AB + AC$ | $A + BC = (A + B)(A + C)$ | Distributive |
| $\overline{A+B} = \overline{A}\cdot\overline{B}$ | $\overline{A\cdot B} = \overline{A}+\overline{B}$ | DeMorgan's |

The first five identities involving a single variable can be easily verified by substituting each of the two possible values for X. For example, to show that X + 1 = X, let X = 1 to obtain 1 + 1 = 1 and then let X = 0 to obtain 0 + 0 = 0. Both equations are true according to the definition of the OR logic operation.

***Identity 5 :*** *It states that double complementation (involution) restores the variable to its original value. Thus, if X = 0, then $\overline{X} = 1$ and $\overline{\overline{X}} = 0 = X$.*

***Identity 6 :*** *The commutative law state that the order in which the variables are written will not affect the result when using the OR and AND operations.*

***Identities 7 :*** *The associative laws, state that the result of forming an operation among three variables is independent of the order that is taken and therefore, the parentheses can be removed altogether as follows,*

$$X + (Y + Z) = (X + Y) + Z = X + Y + Z$$

$$X(YZ) = (XY)Z = XYZ.$$

- Identity 9 is referred to as DeMorgan's theorem. This is a very important theorem and is used to obtain the complement of an expression.
- **Demorgans First Law :** According to the DeMorgan's first law, the complement of a sum is equal to the product of the individual complements, that is,

  $$\overline{A + B} = \overline{A}.\overline{B}$$

  If there exists 'N' variables, then the Demorgan's first law gives,

  $$\overline{A + B + C + \ldots\ldots + N} = \overline{A}.\overline{B}.\overline{C}\ldots\ldots\overline{N}$$

- **Demorgan's Second Law :** According to the DeMorgan's second law, the complement of a product is equal to the sum of individual complements, that is,

  $$\overline{A\,.\,B} = \overline{A} + \overline{B}$$

  If there exists 'N' variables, then the DeMorgan's second law gives,

  $$\overline{A\cdot B\cdot C\ldots\ldots + N} = \overline{A} + \overline{B} + \overline{C} + \overline{D} + \ldots\ldots + \overline{N}$$

- These Laws can be verified using truth table as shown in Table 1.8.3.

**Table 1.8.3** Truth Tables to Verify DeMorgan's Theorem

| DeMorgans First Theorem | | | | DeMorgans Second Theorem | | | | |
|---|---|---|---|---|---|---|---|---|
| A | B | A + B | $\overline{X\cdot Y}$ | A | B | $\overline{A}$ | $\overline{B}$ | $\overline{A}\cdot\overline{B}$ |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 |

## 1.8.2 Algebraic Manipulations

When a Boolean expression is implemented with logic gates, each literal in the function designated as input to the gate. The literal may be primed or unprimed variable. Minimization of the number of literals and the number of terms leads to less complex circuit as well as less number of gates, which should be a designer's objectives.

The number of literals in a Boolean function can be reduced by algebraic manipulations. There is no such rules that will guarantee the final answer. The only method available is a cut and try procedure using the postulates, the basic theorems. The following examples illustrates this procedure.

**Example Problem 1.26**

**Simplify the following boolean functions to least number of literals by manipulation of Boolean algebra.**

**a)** $A\bar{B}\bar{C}D + \bar{A}\bar{B}D + BC\bar{D} + \bar{A}B + B\bar{C}$ **b)** $x + xyz + \bar{x}yz + xw + x\bar{w} + \bar{x}y$

**c)** $\overline{ABC} + \bar{A}BC + ABC + A\bar{B}\bar{C}$ **d)** $(B + \bar{C})(\bar{B} + C) + \bar{A} + B + \bar{C}$

**e)** $(x + y + z)(x + y)x$ **f)** $(x + \bar{y}\,\bar{x})\left[xz + x\bar{z}(y + \bar{y})\right]$

*Sol.* :

a) $f = A\bar{B}\bar{C}D + \bar{A}\bar{B}D + BC\bar{D} + \bar{A}B + B\bar{C}$

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

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

$= (\bar{A} + \bar{C})(D + B \;\; + BC\bar{D} = \bar{A}B + \bar{A}D + \bar{C}B + \bar{C}D + BC\bar{D}$

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

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

b) $f = x + xyz + \bar{x}yz + xw + x\bar{w} + \bar{x}y$

$= x(1 + yz + w + \bar{w}) + \bar{x}yz + \bar{x}y = x + \bar{x}y(z + 1) = x + \bar{x}y = (x + \bar{x})\cdot(x + y)$

$= x + y$

c) $f = \overline{ABC} + \bar{A}BC + ABC + A\bar{B}\bar{C}$

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

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

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

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

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

e) $f = (x + y + z)(x + y)(x)$

$= (x + y + z)\ (x + xy)$

$= x.x + xy.x + x.y + xy.y + xz + xyz = x + xy + xz + xyz$

$= x(1 + y + z + yz) = x$

$f = (x + \bar{x}\bar{y})[(xz + x\bar{z}(y + \bar{y}))] = (x + \bar{x})\,(x + \bar{y})\,[(xz + x\bar{z}(y + \bar{y}))]$

$= (x + \bar{y})[xz + x\bar{z}(y + \bar{y})] = (x + \bar{y})(xz + x\bar{z})$

$= (x + \bar{y})[x(z + \bar{z})] = (x + \bar{y})x$

**Example Problem 1.27**

**Simplify the following logic expressions using Boolean algebra.**

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

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

**c)** $A + AB + ABC + ABCD + \ldots\ldots$

*Sol.* :

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

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

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

$= A[B + C]$ $(\because x + (y.z) = (x + y).(x+y))$

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

$= A\bar{A} + \bar{A}A\bar{B} + B\bar{A}.A + \bar{A}B\bar{B} + AC + A\bar{B}C + ABC + CB\bar{B}$

$= 0 + 0 + 0 + 0 + AC + A\bar{B}C + ABC + 0$ $(\because x\cdot\bar{x} = 0)$

$= AC[1 + \bar{B} + B]$

$= [AC\ (1 + 1)]$ $(\because 1 + A = 1)$

$= AC.1$

$= AC$ $(\because x.1 = x)$

c) $A + AB + ABC + \ldots. = A[1 + B + BC + \ldots.]$

$= A[1]$ $(\because 1 + A = 1)$

$= A$ $(\because x\ .\ 1 = x)$

**Example Problem 1.28**

**Simplify the following expressions**

**a)** $(\bar{A} + B)(A + B + D)\bar{D}$ **b)** $\bar{A}C(\overline{\bar{A}BD}) + \bar{A}B\bar{C}\bar{D} + A\bar{B}C$

**c)** $B(A + \bar{B})(B + C)$ **d)** $\overline{AB}(B + C)$

**e)** $B(A + C) + C$ **f)** $\overline{A\bar{B}(A + C)} + \bar{A}B\overline{(A + \bar{B} + \bar{C})}$

*Sol.* :

a) $f = (\bar{A} + B)(A + B + D)\bar{D} = (\bar{A}A + \bar{A}B + \bar{A}D + BA + B + DB)\bar{D}$

$= (\bar{A}B + \bar{A}D + BA + B + DB)\bar{D}$ $(\because x\bar{x} = 0)$

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

**Table 1.8.4** Truth Table of Given Boolean Function

| A | B | C | f | $\bar{f}$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 |

Thus from Table 1.8.4, we have complement of a given function as,

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

**Example Problem 1.31**

***Find the complement of a function, F = AB + $\bar{C}$***

*Sol.* :

The value of a *F* computed for every combination of A, B and C is as shown in Table 1.8.5.

**Table 1.8.5** Truth Table of Given Boolean Function

| A | B | C | F | $\bar{F}$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 |

The complement of a function F is obtained by complementing each entry in column F as shown in fifth column of Table 1.8.5. From Table 1.8.5, we have complement of a function as,

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

➤ **Second Method (Using DeMorgan's Theorem) :** The complement of a function may also be obtained algebraically through Demorgan's Theorem. The following examples illustrates the complement of a function obtained using Demorgan's theorem.

**Example Problem 1.32**

**Find the complement of the functions,**

**a) F = A'BC' + AB'C**

**b) F = A(B'C + B'C')**

*Sol.* :

a) The complement of a function obtained by applying Demorgans theorem as many times as necessary is as follows,

$$\begin{aligned} F' &= (A'BC' + AB'C)' \\ &= (A'BC')' \cdot (AB'C)' && (\because (A+B)' = A' \cdot B') \\ &= [(A')' + B' + (C)'] \cdot [A' + (B')' + (C')'] && (\because (A.B)' = A'+B') \\ &= (A + B' + C)(A' + B + C) \end{aligned}$$

b)
$$\begin{aligned} F' &= [A(B'C + B'C')]' \\ &= A' + [B'C + B'C']' && (\because (AB)' = A'+B') \\ &= A' + [(B'C)' \cdot (B'C')'] && (\because (A+B)' = A' \cdot B') \\ &= A' + [(B')' + C'] \cdot [(B')' + (C')'] \\ &= A' + (B + C') \cdot (B + C) \end{aligned}$$

➤ **Third Method (Using Duals) :** The complement of a given Boolean function obtained using duals are illustrated using the following examples,

**Example Problem 1.33**

**Determine the complements of the following functions,**

**i) $f_1 = \bar{A}\bar{B}C + A\bar{B}\bar{C}$.**

**ii) $f_2 = \bar{B}(AC + \bar{A}.\bar{C})$.**

*Sol.* :

i) $f_1 = \bar{A}\bar{B}C + A\bar{B}\bar{C}$

**Step 1 :** The dual of $f_1$,

$$f_1^D = (\bar{A} + \bar{B} + C).(A + \bar{B} + \bar{C})$$

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

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

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

$= (B + \bar{A}D)\bar{D}$ $(\because\ x.(y + z) + (x.z))$

$= B\bar{D} + \bar{A}D\bar{D} = B\bar{D}$ $(\because\ x\bar{x} = 0)$

b) $f = \bar{A}C(\overline{\bar{A}BD}) + \bar{A}B\bar{C}\bar{D} + A\bar{B}C$

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

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

$= \bar{B}C(\bar{A} + A) + \bar{A}\bar{D}(C + B\bar{C})$ $(\because\ x\bar{x} = 0)$

$= \bar{B}C + \bar{A}\bar{D}(C + B)$ $(\because\ x + \bar{x} = 1)$

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

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

$= B\left[AB + AC + \bar{B}C\right]$ $(\because\ x\bar{x} = 0)$

$= AB + ABC + \bar{B}CB$ $(\because\ A.A = A)$

$= AB(1 + C) + 0 = AB$ $(\because\ 1 + A = 1)$

d) $f = \overline{AB}(B + C)$,

$= (\bar{A} + \bar{B})(B + C)$ $(\because$ DeMorgan's second law : $\overline{x.y} = \bar{x} + \bar{y})$

$= \bar{A}B + \bar{A}C + \bar{B}C$ $(\because\ x\bar{x} = 0)$

e) $f = B(A + C) + C$

$= AB + BC + C$ $(\because\ x.(y+z) = x.y + x.z)$

$= AB + C(B + 1)$

$= AB + C$ $(\because\ 1 + A = 1)$

$= (C + B)(A + C)$ $(\because\ x + yz = (x + y).(x + z))$

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

$= \overline{A\bar{B} + A\bar{B}C} + \bar{A}B(\overline{A + \bar{B} + \bar{C}})$ $(\because\ x.(y + z) = x.y + x.z)$

$= (\bar{A} + B)(\bar{A} + B + \bar{C}) + \bar{A}B.\bar{A}BC$ $(\because$ By DeMorgan's theorem$)$

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

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

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

**Example Problem 1.29**

**Simplify the following Boolean expressions**

**a)** $xy + xyz + xy\bar{z} + \bar{x}yz$ **b)** $\bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + \bar{A}BC$

**c)** $\bar{A}B\bar{C}D + \bar{A}BCD + ABD$ **d)** $AB + \overline{AC} + A\bar{B}C(AB + C)$

**e)** $(A + B)(\bar{A} + C)(\bar{B} + \bar{C})$

*Sol.* :

a) $f = xy + xyz + xy\bar{z} + \bar{x}yz$

$= xy(1 + z + \bar{z}) + \bar{x}yz$

$= xy + \bar{x}yz$ $(\because\ 1 + A = 1)$

$= y(x + \bar{x}z)$ $(\because\ x.(y + z) = x.y + x.x)$

$= y(x + z)$ $(\because\ A + \bar{A}B = A + B)$

b) $f = \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + \bar{A}BC$

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

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

$= \bar{A}(\bar{C} + BC)$ $(\because\ x.(y + z) = x.y + x.z)$

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

c) $f = \bar{A}B\bar{C}D + \bar{A}BCD + ABD$

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

$= \bar{A}BD + ABD$ $(\because\ x + \bar{x} = 1)$

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

$= BD$

**STEP 2 :** The complement of each literal :

$$f_1^{D'} = (A + B + \bar{C}).(\bar{A} + B + C)$$

For verification consider $f_1' = \overline{\bar{A}\,\bar{B}\,C + A\,\bar{B}\,\bar{C}}$

$$= (\overline{\bar{A}\,\bar{B}\,C}).(\overline{A\,\bar{B}\,\bar{C}})$$

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

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

Hence verified.

ii) $f_2 = \bar{B}.(AC + \bar{A}\,\bar{C})$

**STEP 1 :** (Dual of $f_2$) : $f_2^D = \bar{B} + \left[(A + C).(\bar{A} + \bar{C})\right]$

**STEP 2 :** (Complement each literal) : $\overline{f_2^D} = B + [(\bar{A} + \bar{C}).(A + C)]$

## 1.9 STANDARD FORMS

Logical functions are generally expressed in terms of various combinations of logical variables either in its complement or in its normal form. In general, Boolean function can be expressed in following forms,

1) SOP/FOS Forms.

2) Canonical SOP/POS Forms.

In the canonical form, all 'n' variables or literals are present in the Boolean expression either in complement or in its normal form. In SOP/POS forms, one, two or any number of variables or literals may present in the Boolean expression either in complement or in its normal form.

### 1.9.1 Minterms and Maxterms

➤ **Minterms :** The product of n-variables in Boolean expression either in its complement or in its normal form is called a minterm or a standard product. Suppose that there are two binary variables a and b combined with AND operation and each variable may appear in either form, then there are four possible combinations : $\bar{a}\bar{b}$, $\bar{a}b$, $a\bar{b}$ and ab.

In general, for n number of variables, $2^n$ minterms will be determined. Some properties of minterms are listed below,

- Each minterm is obtained by the AND operation of all the variables of the function.
- In minterms, each variable may appear in either in its normal form or in its complemented form.

- If the corresponding bit of the binary number is 0 each variable is in complemented form and in its normal form if the corresponding bit of the binary number is 1. Minterm is denoted by the symbol Σ.

➤ **Maxterms :** The sum of n variables Boolean expression either in its complemented or in its normal form is called a maxterm or standard sum. Suppose that there are two binary variables a and b associated with OR operation and each variable may appear in either form, then there are four possible combinations : a + b, $a + \bar{b}$, $\bar{a} + b$ and $\bar{a} + \bar{b}$.

In general, for n number of variables $2^n$ maxterms are determined. Some properties of maxterms are listed below,

- Each maxterm is obtained by the OR operation of all variables of the function.
- In maxterms each variable may appear in either in its normal form or in its complemented form.
- If the corresponding bit of the binary number is 1 each variable is in complemented form and if the corresponding bit of the binary number is 0 each variable is in its normal form. Maxterm is denoted by the symbol π.

➤ Table 1.9.1 represents the eight possible minterms and maxterms for three binary variables A, B and C.

Table 1.9.1 Three Variable Minterms and Maxterms

| A | B | C | Minterms | | Maxterms | |
|---|---|---|---|---|---|---|
| | | | Representing Terms | Designation | Representing Terms | Designation |
| 0 | 0 | 0 | $\bar{A}\bar{B}\bar{C}$ | $m_0$ | $A + B + C$ | $M_0$ |
| 0 | 0 | 1 | $\bar{A}\bar{B}C$ | $m_1$ | $A + B + \bar{C}$ | $M_1$ |
| 0 | 1 | 0 | $\bar{A}B\bar{C}$ | $m_2$ | $A + \bar{B} + C$ | $M_2$ |
| 0 | 1 | 1 | $\bar{A}BC$ | $m_3$ | $A + \bar{B} + \bar{C}$ | $M_3$ |
| 1 | 0 | 0 | $A\bar{B}\bar{C}$ | $m_4$ | $\bar{A} + B + C$ | $M_4$ |
| 1 | 0 | 1 | $A\bar{B}C$ | $m_5$ | $\bar{A} + B + \bar{C}$ | $M_5$ |
| 1 | 1 | 0 | $AB\bar{C}$ | $m_6$ | $\bar{A} + \bar{B} + C$ | $M_6$ |
| 1 | 1 | 1 | $ABC$ | $m_7$ | $\bar{A} + \bar{B} + \bar{C}$ | $M_7$ |

From the above Table 1.9.1, we can clearly say, that each maxterm is the complement of the corresponding minterm. For example, if the maxterm is $(\bar{A} + \bar{B} + \bar{C})$, then its complement is,

$$\overline{(\bar{A} + \bar{B} + \bar{C})} = ABC, \text{ which is its corresponding minterm.}$$

## 1.9.2 SOP/POS Forms

In standard forms, the terms that form the Boolean function may contain one, two, or any number of literals. There are two types of standard forms,

1) **Sum of Products :** The sum of products is a Boolean expression containing AND terms, called product terms, of one, two or more literal each. The sum denotes ORing of these terms. Following is an example of SOP expression.

**Example :** $F_1$ (A, B, C) = A'B + AB'C + A'

The 3 variable function $F_1$ has three product terms of two, three and one literals each, respectively. All these product terms are connected with OR operation.

2) **Product of Sums :** The product of sums is a Boolean expression containing OR terms, called sum terms, of one, two or more literals each. The product denotes ANDing of these terms.

**Example :** $F_1(A, B, C) = (A + B + C) \bullet (A + B) \bullet (\bar{A})$

*In this example, Boolean function, $F_1$, of three variables has three sum terms of three, two and one literals each respectively. All these sum terms are connected with AND operation.*

## 1.9.3 Canonical Forms / Standard Forms

*In Canonical forms, the terms that form the Boolean function must contain all n variables either in complemented or in its normal form. There are two types of canonical forms,*

1) **Canonical Sum of Products (CSOP) :** *It is an SOP form in which each product term contains all the variables either in complemented or true (uncomplemented) form i.e., for a 3 variable function, each product term must contain 3 variables. So, here each product term is called as minterm. This type of representation is called "sum of minterms" (or) canonical SOP (or) disjunctive normal expression.*

Examples of canonical SOP with 3 variables are,

i) $A\bar{B}C + A\overline{BC} + \bar{A}B\bar{C}$.

ii) $xyz + \bar{x}\bar{y}z + x\bar{y}\bar{z}$.

2) **Canonical Product of Sums (CPOS) :** It is a POS form in which each sumterm contains all the variables either in complemented or uncomplemented form i.e., for a 3-variable Boolean function, each sum term must contain 3-variables.

Here, the sum is also called as "maxterm" and this type of expression is called canonical POS or product of maxterms or Conjuctive Normal Form (CNF) expression. The following are the examples of canonical POS form for a three input variable.

i) $(x + y + z).(x + \bar{y} + \bar{z}).(\bar{x} + \bar{y} + z)$.

ii) $(A + B + C).(\bar{A} + B + \bar{C}).(A + \bar{B} + \bar{C})$.

## 1.9.4 Conversion of a SOP/POS Form to the Canonical Form

### 1.9.4.1 Conversion From SOP to Canonical SOP Form

To convert SOP to canonical SOP form, the following steps are required,

**Step 1 :** Examine each term, if it is a minterm, retain it and continue to the next term.

**Step 2 :** If the product term is not a minterm, then check the literals that do not occur.

**Step 3 :** Multiply (AND) the product terms having missing literal $x_i$ with $x_i + \bar{x}_i$.

For example, for a three variable function (x, y, z) if a term contains only x, y, since it is not a minterm hence multiply xy with $(z+\bar{z}) = xy(z+\bar{z}) = xyz + xy\bar{z}$.

**Step 4 :** Multiply all products and eliminate the redundant terms.

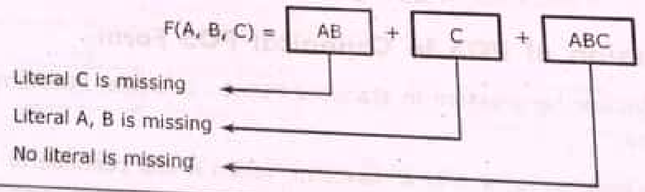
**Example Problem 1.34**

**Convert the following Boolean expression into canonical SOP form.**

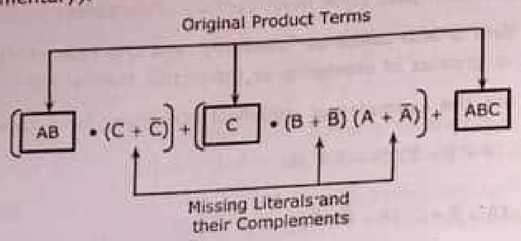
**f = AB + C + ABC**

*Sol.* :

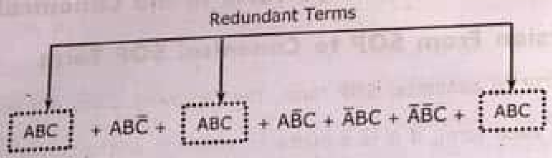
**Step 1 :** Check the product terms or minterms for the missing literals in it.



**Step 2 :** Multiply (AND) the product term having missing literal with (missing literal + its complementary).


Original Product Terms
AB • (C + C̄) + (C • (B + B̄) (A + Ā)) + ABC
Missing Literals and their Complements

**Step 3 :** Multiply all products and eliminate redundant term.



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

**Example Problem 1.35**

**Convert the following Boolean expression into canonical SOP form.**

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

*Sol.* :

*In the given Boolean expression, first minterm* $A\bar{C}$ is having one literal (i.e., B) less. *So multiply it with* $(B + \bar{B})$. Similarly the minterm $\bar{A}\bar{B}$ has missing literal C. So multiply it with $(C + \bar{C})$. Thus,

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

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

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

### 1.9.4.2 Conversion of POS to Canonical POS Form

To convert Boolean expression in standard POS to canonical POS form the following steps are required,

**Step 1 :** Examine each term, if it is a maxterm, retain it and continue to the next term.

**Step 2 :** In each sum term which is not a maxterm, check the variables that do not occur, for each $x_i$ that does not occur, add the sum by $x_i\bar{x}_i$.

For example, for a three variable function (x, y, z) if a term contains only x, y, since it is not a maxterm, hence, add x + y with $z.\bar{z} = x + y + z\bar{z}$

**Step 3 :** Expand the maxterm by applying (Distributive law of '+' over '.')

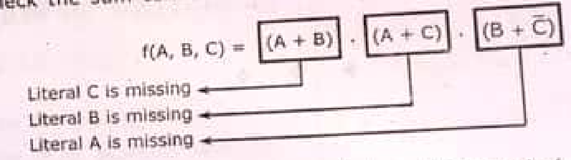
$$(x + y + z\bar{z}) = (x + y + z)(x + y + \bar{z})$$

**Example Problem 1.36**

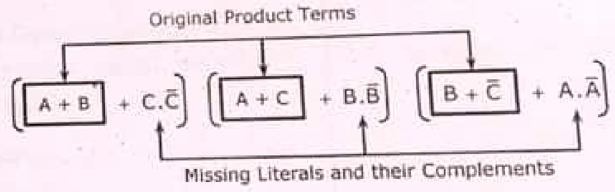
**Convert the Boolean expression f(A, B, C) = (A+B) (A+C) (B+C) into standard POS form.**

*Sol.* :

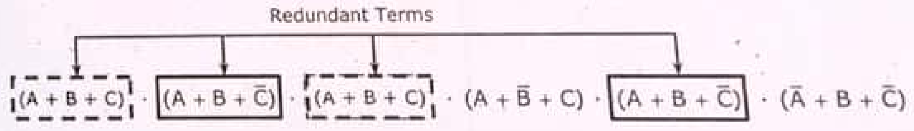
**Step 1 :** Check the sum terms or maxterms for the missing literals in it.



**Step 2 :** Add (OR) the maxterm having missing literal with (missing literal + its complement).


Original Product Terms
(A + B + C.C̄) (A + C + B.B̄) (B + C̄ + A.Ā)
Missing Literals and their Complements

**Step 3 :** Expand the maxterm by using distributive law $X + Y + Z\bar{Z} = (X + Y + Z)(X + Y + \bar{Z})$ and eliminate redundant terms.


Redundant Terms
(A + B + C) · (A + B + C̄) · (A + B + C) · (A + B̄ + C) · (A + B + C̄) · (Ā + B + C̄)

$$\therefore \; f(A, B, C) = (A + B + C)(A + \bar{B} + \bar{C})(A + \bar{B} + C)(\bar{A} + B + \bar{C})$$

**Example Problem 1.37**

**Convert the Boolean expression $f(A, B, C) = (\bar{A}+\bar{B})(A+\bar{C})(A+B)$ into its CPOS form.**

*Sol.* :

In the given expression, maxterm $\bar{A}+\bar{B}$ has the missing literal C, so add it with $C.\bar{C}$. And maxterm $A+\bar{C}$ has the missing literal B, so add it with $B+\bar{B}$. Also A + B has the missing literal C, so add it with $C.\bar{C}$. Thus,

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

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

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

❖ ❖ ❖

f) $f = AB+\overline{AC}+A\bar{B}C(AB+C)$

$= AB+\overline{AC}+A\bar{B}C.AB+A\bar{B}C.C$ $(\because x.\bar{x}=0)$

$= AB+\overline{AC}+A\bar{B}C$ $(\because A.A = A)$

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

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

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

$= 1+B+\bar{C}$ $(\because x+\bar{x}=1)$

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

$= 1$

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

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

$= A\bar{B}C+\bar{A}\bar{B}B+B\bar{B}C+AC\bar{C}+\bar{A}B\bar{C}+BC\bar{C}$ $(\because x.\bar{x}=0)$

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

**Example Problem 1.30**

**Simplify the following Boolean expressions.**

*a)* $\overline{\overline{\overline{AB}C}D}$    **b)** $a+a\bar{b}+a\bar{b}\bar{c}+a\bar{b}\bar{c}\bar{d}+\ldots\ldots$

*c)* $\bar{x}\bar{y}z+yz+xz$    **d)** $(x+y)(\overline{\bar{x}(\bar{y}+\bar{z})})+\bar{x}\bar{y}+\bar{x}\bar{z}$

*e)* $(A\bar{B}C)+(\bar{B}+\bar{C})(\bar{B}+\bar{D})+\overline{A+C+D}$

*Sol.* :

a) $f = \overline{\overline{\overline{AB}C}D}$

$= \overline{\overline{\overline{AB}C}}+\bar{D}$ $(\because$ By De Morgan's theorem)

$= \overline{AB}C+\bar{D} = (\bar{A}+\bar{B})C+\bar{D}$

b) $f = a+a\bar{b}+a\bar{b}\bar{c}+a\bar{b}\bar{c}\bar{d}+\ldots\ldots\ldots$

$= a(1+\bar{b}+\bar{b}\bar{c}+\bar{b}\bar{c}\bar{d})$ $(\because x(y+z) = x.y+y.z)$

$= a.1$ $(\because 1+A = 1)$

$= a$ $(\because x.1 = x)$

c) $f = \bar{x}\bar{y}z+yz+xz$

$= z(x+y+\bar{x}\bar{y})$ $(\because x(y+z) = x.y+x.z)$

$= z(x+y+\bar{y})$ $(\because A+\bar{A}B = A+B)$

$= z(x+1)$ $(\because x+\bar{x}=1)$

$= z$ $(\because 1+A = 1)$

d) $f = (x+y)\left[\overline{\bar{x}(\bar{y}+\bar{z})}\right]+\bar{x}\bar{y}+\bar{x}\bar{z}$

$= (x+y)(\overline{\bar{x}\bar{y}+\bar{x}\bar{z}})+\bar{x}\bar{y}+\bar{x}\bar{z}$

$= (x+y)(x+yz)+\bar{x}\bar{y}+\bar{x}\bar{z} = x.x+xyz+x.y+y.yz+\bar{x}\bar{y}+\bar{x}\bar{z}$

$= x+xyz+xy+yz+\bar{x}\bar{y}+\bar{x}\bar{z} = x(1+yz)+xy+yz+\bar{x}\bar{y}+\bar{x}\bar{z}$

$= x(1+y)+yz+\bar{x}\bar{y}+\bar{x}\bar{z} = x+yz+\bar{x}\bar{y}+\bar{x}\bar{z}$

$= x+yz+\bar{x}(\bar{y}+\bar{z}) = [x+\bar{x}(\bar{y}+\bar{z})]+yz$

$= [x+(\bar{y}+\bar{z})]+yz = x+\bar{y}+\bar{z}+yz$

$= x+\bar{y}+\bar{z}+z = x+\bar{z}+1$

$= x+1 = 1$

e) $f = A\bar{B}C+(\bar{B}+\bar{C})(\bar{B}+\bar{D})+\overline{A+C+D}$

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

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

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

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

## 1.8.3 Complement of a Function

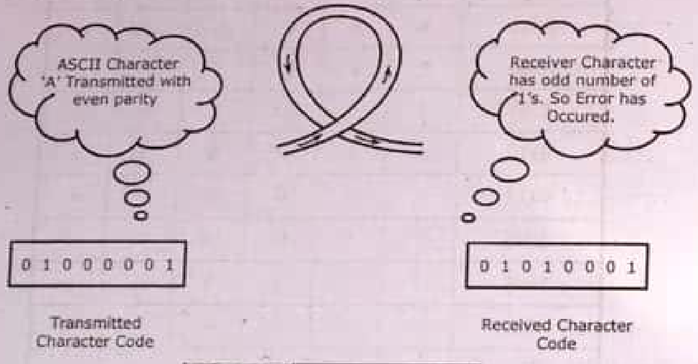
The complement of a Boolean function $f'(x_1, x_2, \ldots\ldots, x_n)$ is a function whose value is 0 whenever the value of true function $f(x_1, x_2, \ldots\ldots, x_n)$ is 1 and 1 whenever the value of true function $f(x_1, x_2, \ldots\ldots, x_n)$ is 0.

➢ **First Method (Using Truth Table)** : If a function is specified by the Truth Table, then it's complement can be obtained by complementing each entry in the column f.

**Example :** Consider the truth table of a Boolean function $f = AB+\bar{A}C+B\bar{C}$ shown in Table 1.8.4. The complement of function is obtained by complementing each entry in column f as shown in Table 1.8.4.

From Table 1.6.2, it can be noticed that inorder to produce an odd number of 1's in the character for odd parity and an even number of 1's in the character for even parity, an extra bit is added in the left most position of the code.

➢ **How the Errors are Detected :** Suppose if an ASCII character is transmitted with even parity. At the receiving end the parity of the transmitted character is checked, if the parity of the received character is not even, it implies that atleast one bit has changed its value during transmission. This is illustrated in Fig. 1.6.1. This method detects one, three, or any odd number of errors in each character being transmitted.



ASCII Character 'A' Transmitted with even parity
Receiver Character has odd number of 1's. So Error has Occured.
0 1 0 0 0 0 0 1
0 1 0 1 0 0 0 1
Transmitted Character Code
Received Character Code

**Fig. 1.6.1** Detection of Errors

➢ **What is done After an Error is Detected :** Once an error is detected, one must request the transmitter to retransmit the character on the assumption that the error was random and will not occur again. Thus, if the receiver detects a parity error, it sends back a NAK (Negative Acknowledge) control character having even parity 8 bits 10010101 (shown in Table 1.6.1). If no error is detected, the receiver sends back an ACK (Acknowledge) control character having even parity eight bits 00000110.

## 1.7 BINARY LOGIC AND GATES

### 1.7.1 Binary Logic

➢ Binary logic consists of variables that takes on two discrete values, and logical operations. The variables are designated by using either upper or lower case letters such as a, b, c or A, B, C. The two values the variables take may be represented by (HIGH, LOW) or (ON, OFF) or (TRUE, FALSE) or (YES, NO) or (0, 1).

➢ There are three types of logic operations AND, OR and NOT.

➢ Binary logic is used to define manipulation and processing of binary information using a mathematical expression.

➢ Binary logic is also used to analyze and design of digital systems.

#### 1.7.1.1 Logical NOT Operation

NOT operation is performed on a single input variable. If A is an input variable to the NOT gate, then the output (F) can be expressed as,

$$F = \bar{A} \text{ (or) } A'.$$

This expression is read as, "F equals NOT A" or "F equals the inverse of A" or "F equals the complement of A".

NOT operation is also referred as "inversion or complementation". It changes the one logic level to it's opposite level. In terms of bits, it changes a logic 1 to a logic 0 and a logic 0 to a logic 1. In terms of logic levels, it changes a logic HIGH to a logic LOW and vice-versa.

The NOT operation can be realized by using a switching circuit for better understanding. For this, Fig. 1.7.1 is considered and the analogy of NOT is shown in Table 1.7.1.

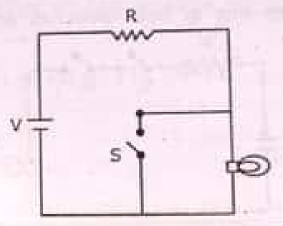


**Fig. 1.7.1** Switching Circuit

**Table 1.7.1**

| Switch Condition (Input) | Lamp Condition (Output) |
|---|---|
| Switch open (low) | Lamp ON (high) |
| Switch close (high) | Lamp OFF (low) |

When switch 'S' is open, lamp is 'ON' and when switch 'S' is close, lamp is OFF.

#### 1.7.1.2 Logical AND Operation

Logical AND operation is performed on a two or multiple input variables. If two inputs A and B are combined using the AND operation, the result F, can be represented as,

$$F = A.B$$

The common symbol for logical AND operation is multiplication symbol (.). Table 1.7.2 shows the result of the AND operation on the variables A and B.

```
                     MSB        LSB        Decimal Equivalent
                      1  1  1  0                 14
                    + 1  0  1  0      ⇨       + 10
Carry Generated → 1 1  0  0  0                  24
```

The process can be explained as,

**Step 1 :** The least significant bits are added i.e., 0 + 0 = 0.

**Step 2 :** The carry in the previous step is 0. The next higher significant bits are added i.e., 1 + 1 = 0 with a carry 1.

**Step 3 :** The carry generated in the second step is 1, added to the next higher significant bits i.e., 0 + 1 + 1 = 0 with a carry 1.

Hence, the sum obtained is 11000. Its equivalent in decimal numbers system is shown adjacently.

## 1.4.1.2 Binary Subtraction

Subtraction of two binary numbers is same as in the decimal number system. The subtraction is started from the LSB and proceeded to the MSB. In subtraction there are always two cases,

➢ **Case 1 (When Borrow is not Taken) :** Suppose that the minued is 1100 and subtrahend is 1000.

```
                MSB        LSB       Decimal Equivalent
(minued)         1  1  0  0               12
(subtrahend)   - 1  0  0  0      ⇨      - 8
                 0  1  0  0                4
```

The process of subtraction in this case can be explained as,

- **Step 1 :** The LSB in first column are 0 and 0. Hence 0 – 0 = 0.
- **Step 2 :** In the next column, subtraction is again performed between 0 and 0. So, 0 – 0 = 0.
- **Step 3 :** In the next column, the difference is obtained by subtracting 1 and 0. So, 1 – 0 = 1.
- **Step 4 :** At the last, subtract 1 – 1 = 0, so the difference is 0100.

➢ **Case 2 (When Borrow is Taken) :** Suppose that minuend is 100 and subtrahend is 0110.

```
     1  10               Decimal Equivalent
  1  0  0  0                  8
- 0  1  1  0        ⇨         6
  0  0  1  0                  2
```

The process of subtraction in this case is as follows,

- **Step 1 :** The LSB's in the first column are 0 and 0, so the difference is 0 – 0 = 0.
- **Step 2 :** In the second column, the subtraction of 1 from 0 is not possible. So, the borrow will be taken from the adjacent higher bit, but that bit is also 0. So, the borrowing has to be done from next higher bit. Hence, the minuend in fourth, third and second column will become 0, 1, 10 respectively. Thus the result obtained in the second column is 10 – 1 = 1.
- **Step 3 :** In the third column, the result is 1 – 1 = 0.
- **Step 4 :** The difference obtained in the fourth column is 0 – 0 = 0.

Finally, the result is 0010.

## 1.4.1.3 Binary Multiplication

It is much simpler than decimal multiplication. The process is similar to that of decimal multiplication. The procedure is as follows,

**Step 1 :** The LSB of multiplier is taken. If,

➢ The multiplier bit is 1, copy the multiplicand as it is given.

➢ The multiplier bit is 0, place 0 at all the bit positions.

**Step 2 :** By taking the next higher multiplier bit, find the partial product and write it by shifting 1 bit towards left.

**Step 3 :** Repeat the step 2 for all bits in multiplier.

**Step 4 :** Find the sum of all the partial products.

**Example Problem 1.12**

**Multiply the given binary numbers**

**a) 1010 and 110**

**b) 11.01 and 1.01.**

**Sol. :**

```
(a)         1  0  1  0    Multiplicand
          ×    1  1  0    Multiplier
            0  0  0  0
         1  0  1  0       ← Shift 1 bit left
      1  0  1  0          ← Shift 1 bit left
      1  1  1  1  0  0    ← Result

(b)         1  1 .0  1    Multiplicand
          ×    1 .0  1    Multiplier
            1  1  0  1
         0  0  0  0       ← Shift 1 bit left
      1  1  0  1          ← Shift 1 bit left
      1  0 .0  0  0  1    ← Result
```

**Example Problem 1.2**

**Convert the decimal number 56.625 into binary, hexadecimal and octal system.**

*Sol.* :

**a) Conversion of $(56.625)_{10}$ into Binary**

Integer Part

$$\begin{array}{r|l}
 & Q \quad R \\
2 & 56 \\
2 & 28 - 0 \\
2 & 14 - 0 \\
2 & 7 - 0 \\
2 & 3 - 1 \\
2 & 1 - 1
\end{array} \uparrow \text{Read upwards}$$

Fractional Part

$$\begin{array}{lcll}
0.625 \times 2 = & 1\,.25 & 1 & \\
0.25 \times 2 = & 0\,.50 & 0 & \downarrow \text{Read Downwards} \\
0.50 \times 2 = & 1\,.00 & 1 &
\end{array}$$

$(56)_{10} = (111000)_2$ $(0.625)_{10} = (.101)_2$

$\therefore$ $(56.625)_{10} = (111000.101)_2$

**b) Conversion of $(56.625)_{10}$ into Hexadecimal**

Integer Part

$$\begin{array}{r|l}
 & Q \quad R \\
16 & 56 \\
 & 3 - 8
\end{array} \uparrow \text{Read upwards}$$

Fractional Part

$0.625 \times 16 = 10.0$ 10

$(56)_{10} = (38)_{16}$ $(0.625)_{10} = (0.10)_{16}$

$\therefore$ $(56.625)_{10} = (38.10)_{16}$

**c) Conversion of $(56.625)_{10}$ into Octal**

Integer Part

$$\begin{array}{r|l}
8 & 56 \\
 & 7 - 0
\end{array} \uparrow \text{Read upwards}$$

Fractional Part

$0.625 \times 8 = 5.0$ 5 $\downarrow$ Read Downwards

$(56)_{10} = (70)_{18}$ $(0.625)_{10} = (0.5)_8$

$\therefore$ $(56.625)_{10} = (70.5)_8$

## 1.3.2 Radix 'r' System to Decimal System Conversion

Any radix-r system number can be converted into decimal system, by multiplying the each radix digit by its positional weight and adding the resulting products.

$$N_b = a_{n-1}\, b^{n-1} + a_{n-2}\, b^{n-2} + \ldots + a_0\, b^0 + a_{-1}\, b^{-1} + a_{-2}\, b^{-2} + \ldots a_{-m}\, b^{-m}$$

where b represents the base of the system.

**Example Problem 1.3**

**Convert the binary number 1101.011 into decimal equivalent.**

*Sol.* :

**Step 1 :** $1 \quad 1 \quad 0 \quad 1 \quad . \quad 0 \quad 1 \quad 1$

**Step 2 :** $2^3 \quad 2^2 \quad 2^1 \quad 2^0 \qquad 2^{-1} \quad 2^{-2} \quad 2^{-3}$

**Step 3 :** $(2^3 \times 1) + (2^2 \times 1) + (2^1 \times 0) + (2^0 \times 1) + (0 \times 2^{-1}) + (1 \times 2^{-2}) + (1 \times 2^{-3})$

$= 8 + 4 + 0 + 1 + 0 + 0.25 + 0.125 = (13.375)_{10}$

$\therefore$ $(1101.011)_2 = (13.375)_{10}$

**Example Problem 1.4**

**Convert the octal number $(432.763)_8$ into the decimal equivalent.**

*Sol.* :

**Step 1 :** $4 \quad 3 \quad 2 \quad . \quad 7 \quad 6 \quad 3$

**Step 2 :** $8^2 \quad 8^1 \quad 8^0 \quad . \quad 8^{-1} \quad 8^{-2} \quad 8^{-3}$

$64 \quad 8 \quad 1 \quad . \quad \frac{1}{8} \quad \frac{1}{64} \quad \frac{1}{512}$

**Step 3 :** $(64 \times 4) + (8 \times 3) + (1 \times 2) + \left(\frac{1}{8} \times 7\right) + \left(\frac{1}{64} \times 6\right) + \left(\frac{1}{512} \times 3\right)$

$= 256 + 24 + 2 + 0.875 + 0.093 + 0.0058 = (282.9738)_{10}$

$\therefore$ $(432.763)_8 = (282.9738)_{10}$

**Example Problem 1.5**

**Convert decimal number $(3A.2F)_{16}$ to its decimal equivalent.**

*Sol.* :

**Step 1 :** $3 \quad A \quad . \quad 2 \quad F$

**Step 2 :** $16^1 \quad 16^0 \quad . \quad 16^{-1} \quad 16^{-2}$

$16 \quad 1 \quad . \quad \frac{1}{16} \quad \frac{1}{256}$

**Step 3 :** $(16 \times 3) + (1 \times A) + \left(\frac{1}{16}\right) \times 2 + \left(\frac{1}{256} \times F\right)$

$= (16 \times 3) + (1 \times 10) + \left(\frac{1}{16} \times 2\right) + \left(\frac{15}{256}\right)$ $(\because A = 10, F = 15)$

$= 48 + 10 + 0.125 + 0.0585 = 58.1835$

$\therefore$ $(3A.2F)_{16} = (58.1835)_{10}$

➢ The memory of a digital computer stores the results, program and data.

➢ The program and data given by the user are transferred into memory by means of an input device such as keyboard. An output device, such as a CRT monitor, displays the results of the computations to the user. An output device, printer gives the printed results of the computations to the user.

## 1.2 NUMBER SYSTEMS

➢ Life without numbers is quite impossible to imagine. A number can be denoted using different symbols. A system of symbols used to represent numbers is called a number system.

These number systems are classified according to the values of the base of the number system. Base (or radix) represents the total number of symbols (or digits) used in a system.

➢ In digital systems, following number systems are frequently used,

- **Decimal Number System :** In decimal number system, there are 10 digit symbols used to represent any number. So its base is 10 and the symbols used to represent any number are 0,1, 2, 3, 4, 5, 6, 7, 8, 9.
- **Binary Number System :** In binary number system, 2 digit symbols (called bit) are used to represent any number. So its base is 2 and symbols used to represent any binary number are 0, 1. In binary number system, a group of 4 and 8 bits is called a nibble and a byte respectively.
- **Octal Number System :** Octal number system uses 8 digit symbols. They are 0, 1, 2, 3, 4, 5, 6, 7. Because of 8 digits its base is 8.
- **Hexadecimal Number System :** Hexadecimal number system uses 16 symbols hence its base is 16. The 16 symbols used in hexadecimal number system are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

➢ A number is made up with a collection of digits and it has two parts, integer and fractional, set apart by a radix point(.). General representation of a number is as shown in Fig. 1.2.1.

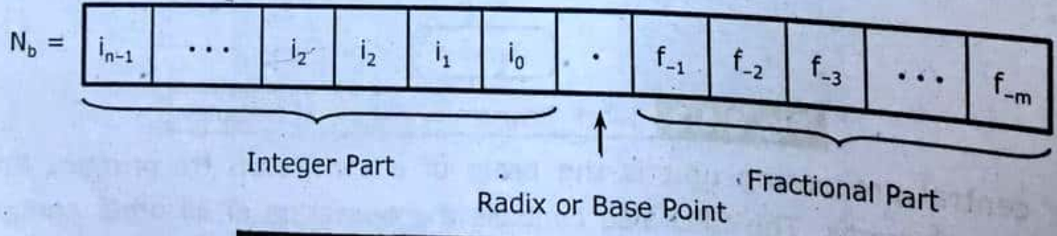


**Fig. 1.2.1** Representation of a Number

Where,

| | | |
|---|---|---|
| $N_b$ | → | A number. |
| b | → | Radix or base of the number system. |
| n | → | Number of digits in integer part. |