UNIT-I

(i) Number systems, conversions and codes:

Digital electronic circuits operate using only two voltage levels for all of their input and output signals

* This two-state operation allows us to use the binary number system when working with digital circuits.
* The binary number system (uses) consists only two digits, which are '0' and '1'. In most digital circuits binary '0' is used to represent 0 volts and binary '1' is used to represent +5 volts
* This two states can also be represented as

0 = Low = ~~True~~ = False = OFF = down

1 = High = True = ON = up

Along with binary number system several number systems are used, when operating with digital circuits

* Most commonly used Number systems are

1. Decimal Number system
2. Binary Number system
3. octal Number system
4. Hexadecimal Number system.

All number systems have a base or radix It specifies how many digits can be used in a number system

1. Decimal Number System:

The Decimal Number system is most familiar to us because it is used by all of us in our everyday world

- The decimal number system is composed of 10 numerals or symbols. So it is called a base-10 system.
- These 10 symbols are: 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9. By using these symbols as digits of a number we can express any quantity.

(It is evo) This number system has evolved naturally as a result of the fact that man has 10 fingers. In fact, the word "digit" is derived from the latin word for "finger"

- The decimal system is a positional-value system, in which the value of a digit depends on its position

→ The positional values of a decimal number system can be represented by the powers of 10 (ten) i.e

$$\cdots 10^{3}, 10^{2}, 10^{1}, 10^{0}, 10^{-1}, 10^{-2}, 10^{-3}, \cdots$$

- Here negative powers are used to calculate decimal points.

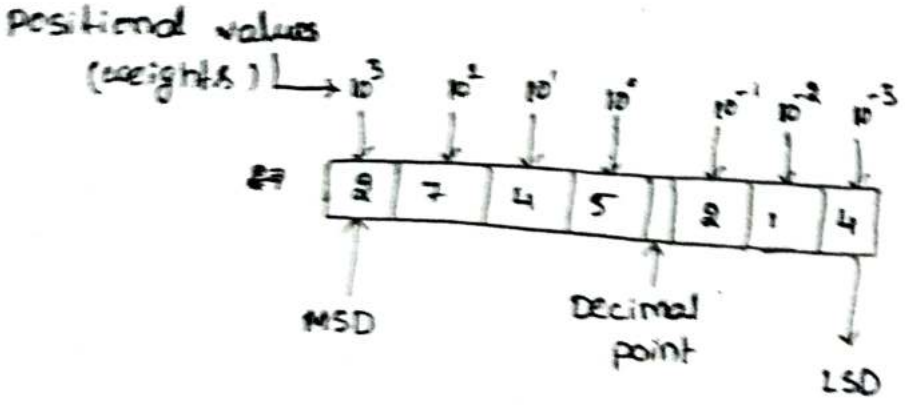


Figure: Decimal position values as powers of 10.

→ The Digit which carries most weight of is called most significant Digit (MSD) and the digit which carries least weight is called Least significant Digit (LSD).

→ The Decimal point seperates the positive power of 10 from the negative powers

→ In general, any number in the system can be written as "the sum of the products of each digit value and its positional value"

From the figure, the number 8745.814 can be written as

$$(8745.814)_{10} = (8\times10^{3}) + (7\times10^{2}) + (4\times10^{1}) + (5\times10^{0}) + (8\times10^{-1}) + (1\times10^{-2}) + (4\times10^{-3})$$

2. Binary Number system:

* In the binary number system there are only two symbols or possible digit values 0 and 1.
* So it is called base-2 number system can be used to represent any quantity that can be represented in decimal or other number systems
* The binary number system is also a positional-value system Here each binary digit has its own value or weight expressed as a power of 2. This is illustrated in below figure

$$\cdots, 2^{3}, 2^{2}, 2^{1}, 2^{0}, 2^{-1}, 2^{-2}, 2^{-3} \cdots$$

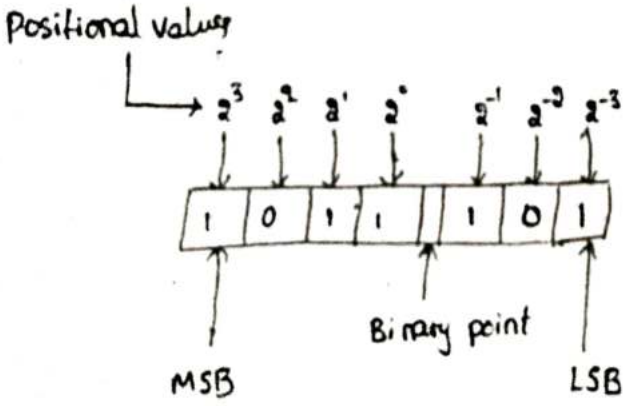


Fig: Binary position values as powers of 2.

* To find the equivalent decimal number to 1011.101. Simply take "the sum of the products of each digit value and its positional value

$$(1011.101)_2 = (1\times2^3) + (0\times2^2) + (1\times2^1) + (1\times2^0) + (1\times2^{-1}) + (0\times2^{-2}) + (1\times2^{-3})$$

$$= 8 + 0 + 2 + 1 + 0.5 + 0 + 0.125$$

$$= 11.625_{10}$$

Here left 4 bits left to binary point called integer part and 3 bits right to binary point represents fractional part

* The most significant bit (MSB) is the left most bit (largest weight) and the least Significant bit (LSB) is the right-most bit (small weight)

In the binary system, the term binary digit is often abbreviated to the term bit

* i.e each digit '0' or '1' is referred to as a 'bit' and A string of 4 bits is called a 'nibble' and Eight(8) bits makes a 'byte'

Almost every digital system uses the binary number system as the basic number system of its operations although other systems are often used in conjunction with binary

3. octal Number system:

* The octal number system is composed of Eight possible digits. So it is called base-8 system The possible digits are 0, 1, 2, 3, 4, 5, 6, and 7.

* The digit positions in an octal number have weight as follows. The positional values of each octal digit can be represented by the power of 8.

$$\ldots\ 8^3, 8^2, 8^1, 8^0, 8^{-1}, 8^{-2}, 8^{-3}\ \ldots$$

* octal number system is same as that of decimal number system but the numbers from 8, 9 onwards are taken as

*

Decimal number (digits)	octal number (digits)
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	10
9	11
⋮	⋮

* when dealing with a large quantity of binary numbers of many bits, it is convenient and more efficient for us to write the numbers in octal system rather than binary. when work with computers Each octal digit represents a group of 3 binary bits 8-1 4 2 1

*

octal Digit ⟶	0	1	2	3	4	5	6	7
Binary equivalent ⟶	000	001	010	011	100	101	110	111

Ex: $437.23_8 = (4\times8^2) + (3\times8^1) + (7\times8^0) + (2\times8^{-1}) + (3\times8^{-2})$

4. Hexadecimal Number system:

* Hexadecimal numbers are extensively used in the field of microcomputers

* Hexadecimal system has base-16. thus it has 16 possible digits. The first ten digits represented by the numbers from 0 to 9 and the letters from A to F are used to represent the Numbers from 10 to 15. i.e 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

* The weigh distribution for the Hexadecimal system can be shown in below: i.e the positional values of each Hexa digit can be represented by the powers of 16.

$$\cdots 16^3, 16^2, 16^1, 16^0, 16^{-1}, 16^{-2}, 16^{-3} \cdots$$

* The digit which carries highest weight is called MSD and the digit which carries lowest weight is called LSD

• Below table shows Binary and Decimal equivalents to Hex digits. Each hex digit represents a group of 4 binary digits

Hexadecimal	Decimal	Binary	octal
0	0	0000	0
1	1	0001	1
2	2	0010	2
3	3	0011	3
4	4	0100	4
5	5	0101	5
6	6	0110	6
7	7	0111	7
8	8	1000	10
9	9	1001	11
A	10	1010	12
B	11	1011	13
C	12	1100	14
D	13	1101	15
E	14	1110	16
F	15	1111	17

Conversions: To understand the operation of an digital system it is necessary to learn conversion between the number systems

(i)(a) Binary - To - Decimal Conversions:-

Any binary number can be converted to its decimal equivalent simply by summing together the weights of the various positions in the binary number.

Ex 1: 11011_2

$1 \quad 1 \quad 0 \quad 1 \quad 1$

$2^4 \quad 2^3 \quad 2^2 \quad 2^1 \quad 2^0$

$= (1\times2^4) + (1\times2^3) + (0\times2^2) + (1\times2^1) + (1\times2^0)$

$= 16 + 8 + 0 + 2 + 1$

$= 27_{10}$

$\therefore 11011_2 = 27_{10}$

Ex 2: 11010.11_2

$1 \quad 1 \quad 0 \quad 1 \quad 0 \quad . \quad 1 \quad 1$

$2^4 \quad 2^3 \quad 2^2 \quad 2^1 \quad 2^0 \quad 2^{-1} \quad 2^{-2}$

$= 16 + 8 + 0 + 2 + 0 + 0.5 + 0.25$

$\therefore 11010.11_2 = 26.75_{10}$

Ex 3: 10110101_2

$1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1$

$2^7 \quad 2^6 \quad 2^5 \quad 2^4 \quad 2^3 \quad 2^2 \quad 2^1 \quad 2^0$

$= (1\times2^7) + (0\times2^6) + (1\times2^5) + (1\times2^4) + (0\times2^3) + (1\times2^2) + (0\times2^1) + (1\times2^0)$

$= 128 + 32 + 16 + 0 + 4 + 0 + 1$

$= 181_{10}$

Ex: 1111_2, 10101_2 ...

(b) Decimal-To-Binary conversions:-

Repeated Division: The binary equivalent of decimal number can be obtained by taking repeated divisions by 2. and write down the remainder after each division until a quotient of '0' is obtained.

* Here the first remainder as LSB and the last remainder as the MSB.

Ex: $(85)_{10}$ =

		remainder
2 | 85		
2 | 42		1 LSB
2 | 6		0
2 | 3		0
2 | 1		1
	0	1 MSB

Read up ↑

$(85)_{10} = (1001)_2$

Ex: $(95.75)_{10} = (\quad)_2$

conversion of Integer part

2 | 95		
2 | 47	– 1	LSB
2 | 23	– 1	
2 | 11	– 1	
2 | 5	– 1	
2 | 2	– 1	
2 | 1	– 0	
	0 – 1	MSB

↑

$95_{10} = 1011111_2$

conversion of Fractional part

$(0.75)_{10} = (\quad)_2$

Required base × Fraction = Result	Fractional part result	Integer part result = coeff
$2 \times 0.75 = 1.5$	0.5	1
$2 \times 0.5 = 1.0$	0.0	1

$(0.75)_{10} = (0.11)_2$

$\therefore (95.75)_{10} = (1011111.11)_2$

Ex: $(37)_{10} = (\quad)_2$

2(a) Octal - to - Decimal conversion:-

An octal number can easily be converted to its decimal equivalent by multiplying each octal digit by its positional weight.

Ex 1: $372_8 = (\ )_{10}$

$(372)_8 = \underset{8^2}{3}\ \underset{8^1}{7}\ \underset{8^0}{2}$

$= (3 \times 8^2) + (7 \times 8^1) + (2 \times 8^0)$

$= 192 + 56 + 2$

$(372)_8 = (250)_{10}$

Ex 2: $(24.6)_8 = ()_{10}$

$(24.6)_8 = \underset{8^1}{2}\ \underset{8^0}{4}\ .\ \underset{8^{-1}}{6}$

$= (2 \times 8^1) + (4 \times 8^0) + (6 \times 8^{-1})$

$= 16 + 4 + 0.75$

$\therefore (24.6)_8 = (20.75)_{10}$

Ex 3: $(32.71)_8 = (\ )_{10}$

$(32.71)_8 = \underset{8^1}{3}\ \underset{8^0}{2}\ .\ \underset{8^{-1}}{7}\ \underset{8^{-2}}{1}$

$= (3 \times 8^1) + (2 \times 8^0) + (7 \times 8^{-1}) + (1 \times 8^{-2})$

$= 24 + 2 + 0.875 + 0.106$

$= 26.981_{10}$

Ex: $(476)_8 = (\ )_{10}$

(b): Decimal - TO - Octal conversion:-

A decimal number can be converted into octal number by taking repeated division by 8 and write down the remainder until quotient becomes zero

Ex1: $(266)_{10} = (\ )_8$

$(266)_{10}$:

		remainder
8 | 266		
8 | 33	- 2	LSD
8 | 4	- 1	
0	- 4	MSD

read up

$\therefore (266)_{10} = (412)_8$

Ex2: $(128)_{10} = $

8 | 128	-	
8 | 16	- 0	LSD
8 | 2	- 0	
0	- 2	MSD

$\therefore (128)_{10} = (200)_8$

Ex3: $(136.72)_{10} = (\ )_8$

Integer part : 136

8 | 136	
8 | 17	- 0
8 | 2	- 1
0	- 2

Fractional part : 0.72

$8 \times 0.72 = 5.76 \quad .5$

$8 \times 0.76 = 6.08 \quad .6$

$8 \times 0.8 =$

$(136)_{10} = (210)_8$

$(210.56)_{10}$

2(c): Octal - To - Binary Conversion :-

The conversion from octal - To - Binary is performed by converting each octal digit to its 3 - binary equivalent digits i.e any octal number is converted to binary by individually converting each digit

Ex 1: $(472)_8 = (\ )_2$

$$\begin{array}{ccc} 4 & 7 & 2 \\ \downarrow & \downarrow & \downarrow \\ 100 & 111 & 010 \end{array}$$

$\therefore (472)_8 = (100\ 111\ 010)_2$

Ex 2: $(5431)_8 = (\ )_2$

$$= \begin{array}{cccc} 5 & 4 & 3 & 1 \\ \downarrow & \downarrow & \downarrow & \downarrow \\ 101 & 100 & 011 & 001 \end{array}$$

$\therefore (5431)_8 = (101\ 100\ 011\ 001)_2$

2(d) Binary - To - octal conversion -

Converting from binary integers to octal integers is simply the reverse of the above process.

i.e the bits of binary numbers are grouped into groups of three bits starting at the LSB

Then Each group is converted to its octal equivalent.

Ex: $(100111010)_2 = (\ )_8$

$$= \begin{array}{|c|c|c|} \hline 100 & 111 & 010 \\ \hline 4 & 7 & 2 \\ \hline \end{array}$$

$\therefore (100111010)_2 = (472)_8$

Ex: $(011\ 010110)_2 = \begin{array}{|c|c|c|} \hline 011 & 010 & 110 \\ \hline 3 & 2 & 6 \\ \hline \end{array}$

$= (326)_8$

3(a) Hex-To-Decimal Conversion :-

A hex number can be converted to its decimal equivalent by multiplying each hex digit by its positional value.

Ex1: $(356)_{16} = (\quad)_{10}$

$= 3 \quad 5 \quad 6$
$\quad 16^2 \quad 16^1 \quad 16^0$

$= (3 \times 16^2) + (5 \times 16^1) + (6 \times 16^0)$

$= 768 + 80 + 6$

$(356)_{16} = (854)_{10}$

Ex2: $(2AF)_{16} = (\quad)_{10}$

$= 2 \quad A \quad F$
$\quad 16^2 \quad 16^1 \quad 16^0$

$= (2 \times 16^2) + (10 \times 16^1) + (15 \times 16^0)$

$= 512 + 160 + 15$

$\therefore (2AF)_{16} = (687)_{10}$.

Ex3: $(4FD.2C)_{16} = (\quad)_{10}$.

$= 4 \cdot F \quad D \cdot \quad 2 \quad C$
$\quad 16^2 \quad 16^1 \quad 16^0 \quad 16^{-1} \quad 16^{-2}$

$= (4 \times 16^2) + (15 \times 16^1) + (13 \times 16^0) + (2 \times 16^{-1}) + (12 \times 16^{-2})$

$= 1024 + 240 + 13 + 0.125 + 0.75$

$(ABCD)_{16} = (\quad)_{10}$
$\quad 16^3 \; 16^2 \; 16^1 \; 16^0$

$= (10 \times 16^3) + (11 \times 16^2) + (12 \times 16^1) + (13 \times 16^0)$

$= 40960 + 2816 + 192 + 13$

$= (43981)_{10}$.

$\therefore (ABCD)_{16} = (43981)_{10}$

(3b)

(b) Decimal-to-Hex conversion :-

Decimal-to-Hex conversion can be done by taking repeat divisions by Hex base i.e 16.

Ex1: $(423)_{10} = (\ )_{16}$

$= 16 \overline{)423}$

$16 \overline{)26}$ - 7 LSB

$16 \overline{)1}$ - 10 or A ↑ Read up

0 - 1 MSB

$\therefore (423)_{10} = (1A7)_{16}$

Ex2: $(214)_{10} = (\ )_{16}$

$= 16 \overline{)214}$

$16 \overline{)13}$ - 6 ↑

0 - 13 or D

$\therefore (214)_{10} = (D6)_{16}$

(c) Hex-to-Binary conversion :

Any Hexadecimal number can be converts into its binary equivalent by changing each hexadecimal digits to it's 4-bit binary number.

To convert a Hexadecimal number to a binary number. change each hexadecimal digit to its 4-bit binary equivalent.

Ex1: FAF_{16}

$(FAF)_{16} = (\ )_2$

$= F \quad A \quad F$

$15 \quad 10 \quad 15$

$1111 \quad 1010 \quad 1111$

$\therefore (FAF)_{16} = (1111\ 1010\ 1111)_2$.

Ex2: $(9F2)_{16} = 9 \quad F \quad 2$

$1001 \quad 1111 \quad 0010$

(9F2)

(d) Binary-To-Hexadecimal Conversion:

Conversion from Binary-To-Hex is just the reverse process of Hex-To-Binary conversion.

i.e the binary number is grouped into groups of 4-bits and each group is converted to its equivalent Hex digit zero's are added as needed to complete 4-bit group

Ex1:

$$(100111110010)_2 = 1001 \mid 1111 \mid 0010$$
$$9 \quad F \quad 2$$
$$\therefore (1001\ 1111\ 0010)_2 = (9F2)_{16}$$

Ex2: $(111010011\ 0)_2 = 0011 \mid 1010 \mid 0110$
$$3 \quad A \quad 6$$
$$\therefore$$
$$(1110100110)_2 = (3A6)_{16}$$

In this way hex, (octal) are so useful in representing large binary numbers

Ex3: $(\quad)_{16}$ Verify $(101011111)_2 = (\quad)_{16}$

(E) Hex-To-octal conversion:

And

(F) octal-To-Hex conversion:

when doing conversions from octal-To-Hex and Hex-To-octal, First convert Each digit into binary and then convert the binary into the desired (Hex or octal) number system.

Ex1 Hex-To-octal:

$(B2F)_{16} = (\quad)_8$

$\Rightarrow$ B 2 F

$=$ 1011 0010 1111 $\longrightarrow$ convert to Binary

$=$ 101 100 101 111

= 101 100 101 111 group into 3-bit grouping

= 5 4 5 7 octal Number

$\therefore (B2F)_{16} = (5457)_8$

Ex2: $(9C4)_{16} = (\ )_8$

= 9 C 4

= 1001 1100 0100 → Binary Form

= 100 | 111 | 000 | 100 → Grouping into 3-bits

4 7 0 4

$(9C4)_{16} = (4704)_8$

Ex: octal-To-Hex:

$(372)_8 = (\ )_{16}$

= 3 7 2

011 111 010 → Binary form

= 0000 1111 1010 → Grouping into 4-bits

= 0FA

$\therefore (372)_8 = (0FA)_{16}$

CODES:

The group of symbols is called a code

→ when numbers, letters or words are represented by a special group of symbols then we say that they are encoded

→ one of the most familiar codes is the "Morse code" where series of dots and dashes represent letters of the alphabet

→ Most Probably used codes in digital systems are

1. BCD (Binary-coded-Decimal) code
2. GRAY code
3. ASCII code

① Binary-coded-Decimal(BCD) code :-

when a decimal number is ~~represented~~ converted to by its equivalent binary number i.e in the group of '0's and '1's then the process is called "straight binary coding". By this coding the conversions between decimal and binary numbers become long and complicated for large numbers. For this reason encoding decimal numbers is used in certain situations. Ex: 4569873

BCD code :- If each digit of a decimal number is represented by its binary equivalent i.e in groups of 0's and 1's then the resultant code is called "Binary Coded Decimal (BCD) code"

→ Here 4-bits are required to code each Decimal digit

BCD system can code decimals from 0 to 9 and the BCD code does not use the decimal numbers from 10 to 15

→ If any of the "forbidden" 4-bit numbers occures in a machine by using this BCD code then the machine indicated that an error has occured.

Ex 1: $(874)_{10}$ = 8 → 1000, 7 → 0111, 4 → 0100 ---> BCD code

Ex 2: $(943)_{10}$ = 9 → 1001, 4 → 0100, 3 → 0011 ---> BCD code

Ex 3: convert 0110100001111001 (BCD) to its Binary equivalent

Sol: Divide the BCD Number into 4-bit groups and convert each to Decimal.

$$0110100001111001 = \underset{\downarrow}{0110}\ \underset{\downarrow}{1000}\ \underset{\downarrow}{0011}\ \underset{\downarrow}{1001} \rightarrow \text{BCD}$$

$$\underset{\downarrow}{6}\quad \underset{\downarrow}{8}\quad \underset{\downarrow}{3}\quad \underset{\downarrow}{9} \rightarrow \text{Decimal}$$

$$= (6839)_{10}$$

Ex 4: convert 011111000001 to its decimal equivalent

$$011111000001 = \underset{\downarrow}{0111}\ \underset{\downarrow}{1100}\ \underset{\downarrow}{0001} \rightarrow \text{BCD}$$

$$\underset{\downarrow}{7}\quad \underset{\downarrow}{1}$$

Forbidden code group indicates Error in BCD Number

(12-C)

Comparision of BCD and Binary :-

→ It is important to realize that BCD is not another number system like binary, octal, decimal and Hexadecimal.

→ It is also important to understand that a BCD ~~number~~ code is not same as a straight binary (number) coding.

→ A straight binary code takes the complete decimal number and represents it in binary but, the BCD code converts each decimal digit to binary nibble individually.

→ Ex: Take the number 137 and compare its straight binary and BCD codes:

$(137)_{10} = 10001001_2 \rightarrow$ binary

$(137)_{10} = 0001\ 0011\ 0111 \rightarrow$ BCD

2	137
2	68 - 1
2	34 - 0
2	17 - 0
2	8 - 1
2	4 - 0
2	2 - 0
2	1 - 0
	0 1

$(10001001)_2$

→ Thus the BCD code requires 12-bits while the straight binary code requires only 8-bits to represent a

possible 4-bit groups.

Importance: This case of conversion is especially important from a hardware standpoint because in a decimal system there are logic circuits that have to perform the conversions to and from decimal numbers.

Binary coded decimal values

Decimal	BCD code
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	0001 0000
11	0001 0001
12	0001 0010
13	0001 0011
14	0001 0100
15	0001 0101

Table: Decimal numbers and their equivalent BCD

Binary Codes - Before going to Gray code we have to learn addition. Binary subtraction. Binary multiplication and binary division.

Binary addition:

$0 + 0 = 0$

$0 + 1 = 1$

$1 + 0 = 1$

$1 + 1 = 0$ with carry 1

Ex1 ~~1 0 1 0~~ / ~~1 0 0~~

$$\begin{array}{r} 1\,0\,1\,1\,0\,1 \\ 1\,0\,0\,1\,1\,1 \\ \hline 1\,0\,1\,0\,1\,0\,0 \end{array}$$

④
$$\begin{array}{r} 1\,0\,0\,1\,0\,1\,1 \\ 1\,1\,1\,0\,0\,1\,1 \\ \hline 0\,0\,1\,1\,1\,0\,0 \\ \hline 1\,1\,0\,1\,1\,0\,0\,0 \end{array}$$

⑤ Binary Subtraction:-

Ex:
$0 - 0 = 0$
$0 - 1 = 1$ with barrow 1
$1 - 0 = 1$
$1 - 1 = 0$

Ex1:
$$\begin{array}{r} 1\,0\,1\,1\,0\,1 \\ 1\,0\,1\,0\,1\,0 \\ \hline 0\,0\,0\,0\,1\,1 \end{array}$$
(1 Barrow)

Ex2:
$$\begin{array}{r} 1\,1\,0\,1\,1\,0 \\ 1\,0\,1\,1\,0\,1 \\ \hline 0\,0\,1\,0\,0\,1 \end{array}$$
(1 Barrow)

⑥ Binary Multiplication:-

$0 \times 0 = 0$
$0 \times 1 = 0$
$1 \times 0 = 0$
$1 \times 1 = 1$

Ex1: 1101×101

$$\begin{array}{r} 1\,1\,0\,1 \\ 0\,0\,0\,0 \\ 1\,1\,0\,1 \\ \hline 1\,0\,0\,0\,0\,0\,1 \end{array}$$

Ex2: 1011×11

$$\begin{array}{r} 1\,0\,1\,1 \\ 1\,0\,1\,1 \\ \hline 1\,0\,0\,0\,1 \end{array}$$

⑦ Binary Division:-

$$10\overline{)1\,1\,0\,1\,1\,0\,1}\ (11011$$

$$\begin{array}{r} 10 \\ \hline 10 \\ 10 \\ \hline 0011 \\ 10 \\ \hline 10 \\ 10 \\ \hline 01 \end{array}$$

② Gray code:

Gray code belongs to a class of codes called "minimum-change codes", in which only one bit in the code group changes when going from one step to the next.

→ The Gray code is an unweighted code i.e the bit positions in the code group do not have any specific weight.

→ Because of this the Gray code is not suited for arithmetic operations.

→ It finds application in input/output devices and some types of analog-to-digital converters.

→ Below table shows the Gray-code representation for the decimal numbers from 0 to 15 together with the straight binary code.

Decimal number	Binary code	Gray code
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

By using Binary add

→ If we examine the Gray-code, it can be seen that in going from any one decimal number to the next, only one bit from the Gray

code changes This is the principle characteristic of the Gray code

→ For example

in going from 3 to 4 $\xrightarrow{\text{changing in Gray code}}$ 0010 to 0110 only the second bit from the left was changed

in 9 from 14 to 15 $\longrightarrow$ 1001 to 1000 here only the last bit was changed

→ But in case of binary code when going from one decimal number to other the next one bit all bits are changed.

Binary might produce erroneous or ambiguous results during those transitions in which more than one bit of the code is changing.

For example;

using binary code when going from 0111 to 1000 all i.e 4 bits have to be changed simultaneously.

→ Here the transition from 0111 to 1000 could produce one or more intermediate states, the following transitions will occur

For example:

9 1 0 1 $\longrightarrow$ decimal 9

0 1 1 1 $\longrightarrow$ Decimal 7

1 1 1 1 $\dashrightarrow$ erroneous code

1 0 0 0 $\longrightarrow$ Decimal 8

Here most significant bit changes faster than the rest

→ The occurrence of 1111 is only momentary but it could produce erroneous operation of the elements

→ obviously; in this situations the grad code is used.

→ Thus the use of Gray code reduces or eliminates ambiguous or erroneous results. since in this code only one bit can change at transition and no "race" between bits can occur.

Binary-to-Gray code conversion :-

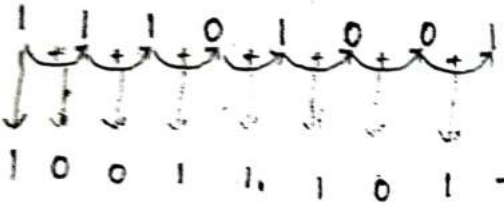
$G_3 = A_3$; G for Gray code, A for Binary code

$G_2 = A_3 + A_2$

$G_1 = A_2 + A_1$

$G_0 = A_1 + A_0$

Ex: 11101001 =



Gray code-to-Binary code conversion :-

$A_3 = G_3$; A for Binary code bit, G for Gray code bit

$A_2 = A_3 + G_2$

$A_1 = A_2 + G_1$

$A_0 = A_1 + G_0$

Ex: 10011101 =

3) ASCII code :

The ASCII code is an alphanumeric code. ASCII (ask-key) stands for: American Standard code for Information Interchange.

→ The ASCII code is a 7-bit binary code. So, it has $2^N - 1 = 2^7 - 1 = 127$ possible code groups. The ASCII code has been used to standardize codes for numbers, letters and symbols in equipment such as printers, keyboards and computer displays.

→ Each keystroke on an ASCII keyboard produces a corresponding binary code for the designated character.

ASCII code most widely used in minicomputers and microcomputers and in many main frames

→ The ASCII code is used to transfer alphanumeric information between a computer and input/output devices

Binary values for each number, letter and symbol in ASCII code are shown in below table.

$X_3X_2X_1X_0$	$X_6X_5X_4$ = 010	011	100	101	110	111
0000						
0001						
0010						
0011						
0100						
0101						
0110						
0111						
1000						
1001						
1010						
1011						
1100						
1101						

$X_3X_2X_1X_0$ \ $X_6X_5X_4$	010	011	100	101	110	111
0000	blank space	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	E	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	DEL

Examples

① ASCII code for @ = $X_6X_5X_4X_3X_2X_1X_0$ = 1000000

for & = 0100110

for = = 0111101

for I = 1001001

for H E L P = 1001000 1000101 ~~10~~ 1001100 1010000

GOTO 25 = 1000111 1001111 1010100 1001111 0100000 (space) 0110010 0110101

1.4 BINARY ARITHMETIC :

1.4.1 Binary Addition and Subtraction :

Binary addition is performed in the same manner as decimal addition. The binary addition is as follows :

0 + 0	= 0	
0 + 1	= 1	
1 + 0	= 1	
1 + 1	= 0	with a carry of 1

Notice that the first three additions result in a single bit, and that the addition of two 1's yields a binary two (10_2). Since 1 is the largest digit in the binary system, any sum greater than 1 requires that a digit be carried over. When binary numbers are added, a sum of 0 in a given column and a carry of 1 over to the next higher column should be observed as illustrated below :

$$\begin{array}{cccc} & \textcircled{1} & \textcircled{1} & \\ & 0 & 1 & 1 \\ + & 0 & 0 & 1 \\ \hline & 1 & 0 & 0 \\ \hline \end{array}$$

[where the carry is encircled as ①]

The following examples illustrate the binary addition.

Examples

(a)

Binary	Decimal
10	2
+ 11	+ 3
101	5

(b)

Decimal	Binary
7	111
+ 4	+ 100
11	1011

(c)

Decimal	Binary
8	1000
+ 2	+ 10
10	1010

(d)

Binary	Decimal
1111	15
+ 10100	+ 20
100011	35

(e)

Decimal	Binary
4.25	100.01
+ 7.75	+ 111.11
12.00	1100.00

(f)

Decimal	Binary
5.25	101.01
+ 6.50	+ 110.10
11.75	1011.11

Binary Subtraction : It is a special case of addition. In fact the addition of a negative number to a positive number is subtraction.

To subtract, it is essential to establish a procedure for subtracting a larger from a smaller digit. But, when 1 is subtracted from 0, it is necessary to borrow 1 from the next column to the left. The binary subtraction is as follows :

$0 - 0 = 0$

$1 - 0 = 1$

$1 - 1 = 0$

$0 - 1 = 1$ with a borrow of 10. This borrowed '1' will have a value of 10 in the current position.

Let us examine exactly what was done to subtract the two binary numbers in the following example :

	1	0	1	5
–	0	1	0	–2
	0	1	1	3

In this example;

(1) In the first column; we have $1 - 0$ the result is 1.

(2) In the second column; we have $0 - 1$; then borrow 1 from next column; making a 10 in this column, hence $10 - 1 = 1$.

(3) In the third column; when a 1 is already borrowed, a 0 is left; hence $0 - 0 = 0$.

The following examples illustrate the binary subtraction.

Examples

(a)

Decimal	Binary
6	110
– 4	– 100
2	010

(b)

Decimal	Binary
10	1010
– 7	– 111
3	0011

(c)

Decimal	Binary
12	1100
– 8	– 1000
4	0100

(d)

Decimal	Binary
16	10000
– 10	1010
6	00110

(e)

Decimal	Binary
7.25	111.01
– 4.50	– 100.10
2.75	10.11

(f)

Decimal	Binary
10.75	1010.11
– 6.25	– 0110.01
4.50	100.10

1.4.2 Binary Multiplication and Division :

Binary multiplication is performed in the same manner as with decimal numbers. The simple multiplication with binary digits is as follows :

$0 \times 0 = 0$
$0 \times 1 = 0$
$1 \times 0 = 0$
$1 \times 1 = 1$

Note that the binary multiplication also involves forming the partial products, shifting each successive partial product left to one place, and then adding all the partial products. The following examples illustrate the procedure and the equivalent decimal multiplication.

Examples

(a)

Decimal	Binary
$3 \times 2 = 6$	11×10
	00
	11
	110

(b)

Decimal	Binary
$4 \times 3 = 12$	100×11
	100
	100
	1100

(c)

Decimal	Binary
$7 \times 6 = 42$	111×110
	000
	111
	111
	101010

(d)

Decimal	Binary
$12 \times 10 = 120$	1100×1010
	0000
	1100
	0000
	1100
	1111000

(e)

Decimal	Binary
$15 \times 11 = 165$	1111×1011
	1111
	1111
	0000
	1111
	10100101

Binary division is very simple. As in the decimal system, division by zero is meaningless. The division by binary digits is as follows :

$0 \div 1 = 0$
$1 \div 1 = 1$

Binary division can also be implemented by following a procedure similar to that used in long hand division with appropriate modifications. Let us consider the different examples for binary division.

1.5 BCD CODE OR 8421 CODE :

It is known as Binary Coded Decimal (BCD) code; in which decimal digits 0 through 9 are represented by their binary equivalents using four bits. In otherwords, Binary coded decimal means that each decimal digit is represented by a binary code of four bits. In this code, the designation 8421 indicates the binary weights of the four bits; i.e. 2^3, 2^2, 2^1, 2^0.

Let us consider the ten binary combinations as shown in table 1.3 that represent the ten decimal digits in BCD. From this, we should realise one thing. With the four bits, 2^4 i.e. sixteen numbers can be represented. But in this code, we are using only ten. The rest of six code combinations that are not used. In otherwords; 1010, 1011, 1100, 1101, 1110 and 1111 are invalid in this BCD code.

Table 1.3 : The BCD Code

Decimal	8421 (BCD)
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

Example :

Convert each of the following decimal numbers into BCD.

(i) 9 (ii) 7.5 (iii) 26.8 (iv) 70 (v) 726 (vi) 7287

Solutions :

(i) 9 → 1001

(ii) 7 • 5 → 0111 0101

(iii) 2 6 • 8 → 0010 0110 1000

(iv) 7 0 → 0111 0000

(v) 7 → 0111

(vi) 2 6 → 0010 0110

7 2 8 7 → 0111 0010 1000 0111

We can also consider the determination of a decimal number from a BCD. In this process, start at the decimal point and break the code into groups of four bits. Then write down the decimal digit corresponding to that four-bit group of binary word. The following example illustrates.

Example :

Find the decimal numbers represented by the following BCD or 8421 codes :

(i) 10000001 (ii) 10010100 (iii) 00111000.0001

(iv) 010101111000 (v) 1001011001111000

Solutions :

(i) 1000 0001 → 8 1

(ii) 1001 0100 → 9 4

(iii) 0011 1000 • 0001 → 3 8 • 1

(iv) 0101 0111 1000 → 5 7 8

(v) 1001 0110 0111 1000 → 9 6 7 8

BCD Addition :

In this code, we can also perform BCD addition, provided that in each case the *sum in any four-bit column doesnot exceed 9*. Then only the results are valid BCD numbers. The following example illustrate both the valid BCD addition and invalid addition.

Example :

Add the following BCD numbers.

(i) 0001 0100 + 1000 0001

(ii) 1001 1001 + 0101 0111

Solutions :

(i) 0001 0100 + 1000 0001 = 1001 0101 → 14 + 81 = 95

This is *valid* BCD addition. In all the cases, the four-bit sum is equal to or less than 9.

(ii) 1001 1001 + 0101 0111 → 99 + 57

This is *invalid* BCD addition. Because; four-bit sum is greater than 9 and also carry-out of the group is generated. Hence BCD addition can't be performed.

1.6 ALPHANUMERIC CODES :

Many applications of digital computer require the data not only in the form of numbers, but al it contains alphabet letters and certain special characters i.e., +, –, *, %, =. Hence to get such informati for a computer, we need to use a code which contains letters, numbers and other symbols, known 'Alphanumeric code'. In addition to the symbols and numbers, most of these codes also represent th various instructions for conveying the information.

To handle the data with the computers, at a minimum, an alphanumeric code must represent te decimal digits and 26 letters of the alphabet, for a total of 36 items. Hence it requires six bits in eac code combination. Thus there are ($2^6 = 64$) 64 total combination of six bits, so we have 28 (64 – 36 unused code combinations.

At one time, manufacturers used their own alphanumeric codes, which led to all kinds o confusion. Hence the industry settled the important alphanumeric codes for the usage. These ar ASCII code (Americal Standard Code for Information Interchange) and EBCDIC (Extended Binar Coded Decimal Interchange Code).

(i) ***ASCII :*** (Pronounced as **ask–ee**) : It is a widely used standardised alphanumeric code. It is a seven bit code in which the decimal digits are represented by the 842 coded preceded by 011. The letters of the alphabet and other symbols and instructions are represented by other code combina-tions are shown in the table 1.6.

For example the letter 'B' is represented by 1000010 $(42)_{16}$, the positive sign (+) is represented by 0101011 $(2B_{16})$ and ESC (Escape) is by 0011011 $(1B_{16})$.

(ii) ***EBCDIC :*** (Pronounced as **ebb'see dick**) It is an eight bit alphanumeric code. In this code, the decimal digits are represented by 8421 code preceded by 111. The letters, symbols, commands characters and instructions are shown in table 1.7.

For example, the capital letter 'C' is represented by 11000011, small letter 'c' is represented by 10000011 and the positive sign (+) is by 01001110.

(iii) In addition to the above two codes, Telegraphic code and Hollerith code are also known as 'alphanumeric codes'. Telegraphic code / Morse code is used in Telegraphy. Hollerith code is used in punched cards.

Table 1.6 : ASCII (American Standard Code for Information Interchange)

	MSBs							
LSBs	000 (0)	001 (1)	010 (2)	011 (3)	100 (4)	101 (5)	110 (6)	111 (7)
0000 (0)	NUL	DLE	SP	0	@	P	`	p
0001 (1)	SOH	DC_1	!	1	A	Q	a	q
0010 (2)	STX	DC_2	"	2	B	R	b	r
0011 (3)	ETX	DU_3	#	3	C	S	c	s
0100 (4)	EOT	DU_4	$	4	D	T	d	t
0101 (5)	ENQ	NAK	%	5	E	U	e	u
0110 (6)	ACK	SYN	&	6	F	V	f	v
0111 (7)	BEL	ETB	'	7	G	W	g	w
1000 (8)	BS	CAN	(	8	H	X	h	x
1001 (9)	HT	EM	)	9	I	Y	i	y
1010 (A)	LF	SUB	*	:	J	Z	j	z
1011 (B)	VI	ESC	+	;	K	[	k	{
1100 (C)	FF	FS	,	<	L	\	l	|
1101 (D)	CR	GS	-	=	M	]	m	}
111D (E)	SO	RS	.	>	N	↑	n	~
1111 (F)	SI	US	/	?	O	←	o	DEL

Definitions of control abbreviations :

ACK	Acknowledge	ETB	End of transmission block
BEL	Bell	ETX	End text
BS	Back space	FF	Form feed
CAN	Cancel	FS	Form separator
CR	Carriage return	GS	Group separator
$DC_1 - DC_4$	Direct control	HT	Horizontal tab
DEL	Delete idle	LF	Line feed
DLE	Data link escape	NAK	Negative acknowledge
EM	End of medium	NUL	NULL
ENQ	Enquiry	RS	Record separator
EOT	End of transmission	SI	Shift in
ESC	Escape	SO	Shift out
SOH	Start of heading	STX	Start text
SUB	Substitute	SYN	Synchronous idle
US	Unit separator	VT	Vertical tab

1.7 PARITY BIT :

The parity check is based on the use of an additional bit, known as a *'parity bit'* or *'parity-check bit'*. This parity can be associated with each group. An even–parity–bit checking code has a parity bit such that the sum of the 1's in the code group plus the parity bit is always an even number. Similarly for an odd–parity–bit checking system. The example shown in Table 1.8 which uses 8421 code contains the both odd parity bit and even parity bit, at independent level.

Table 1.8 : ODD and EVEN Parity Bit

Decimal	8421 Code (BCD)	ODD Parity Bit	EVEN Parity Bit
0	0000	1	0
1	0001	0	1
2	0010	0	1
3	0011	1	0
4	0100	0	1
5	0101	1	0
6	0110	1	0
7	0111	0	1
8	1000	0	1
9	1001	1	0

To detect or correct the errors, the parity bit can also be used in ASCII code; when it is used for sending the data over the transmission lines. Then a parity checker at the receiving end can test for even or odd parity, whichever parity has been prearranged between the transmitter and the receiver. Since ASCII code uses 7 bits, the addition of a parity bit to the transmitted data produces an 8–bit number, as shown below :

$X_7\ X_6\ X_5\ X_4\ X_3\ X_2\ X_1\ X_0$

↑ parity bit.

Thus the technique of parity checking is the most popular method of detecting errors in stored code groups for storage devices and whenever the data transfer occurs between the digital systems.

(II) BOOLEAN ALGEBRA AND GATES

1.8 INTRODUCTION TO BOOLEAN ALGEBRA :

'Boolean Algebra' (also known as 'Switching Algebra') is a set of rules, laws and theorems by which all logical operations can be mathematically expressed. It is only the convenient way of expressing the functions of digital circuits and systems.

This algebra was first introduced by George Boole, an England mathematician. It also provides an economical way of describing the circuitry used in digital systems. It is a basic tool to determine the logical operations of digital electronic systems. This chapter also presents the simplification of logic expressions using Boolean algebric postulates and theorems.

1.9 POSTULATES OF BOOLEAN ALGEBRA :

This algebra is introduced by first defining a set of elements which are allowed, a set of operations which operate with the elements and a set of 'postulates' or Axioms. Any thing which is not proved but assumed to be true, is known as '*postulate*'. The theorems of the Boolean Algebra are derived from these postulates.

To begin with the postulates of the Boolean algebra, if M is used as a set and 'x' and 'y' are the two objects; then the notation $x, y \varepsilon M$ implies that x and y are members of the set M. Also note that if $y \notin M$, implies that y is not a member of the set M.

Postulate 1 :

(a) An operation '+' is defined such that if $z = x + y$ then $z \varepsilon M$ for every pair of elements $x, y \varepsilon M$.

(b) An operation '$\bullet$' is defined such that if $w = x \bullet y$ then $w \varepsilon M$ for every pair of elements $x, y \varepsilon M$.

Postulate 2 :

(a) There exists an elements '0' in M such that $x + 0 = x$ for every element $x \varepsilon M$.

(b) There exists an element '1' in M such that $x \bullet 1 = x$ for every element $x \varepsilon M$.

Postulate 3 :

For $x, y \varepsilon M$ the following commutative laws hold as follows :

(a) $x + y = y + x$

(b) $x \bullet y = y \bullet x$

Postulate 4 :

For $x, y, z \varepsilon M$ the following distributive laws hold as follows :

(a) $x \bullet (y + z) = x \bullet y + x \bullet z$

(b) $x + (y \bullet z) = (x + y) \bullet (x + z)$

Postulate 5 :

For every element $x \varepsilon M$, there exists an element $\overline{x}$ such that

(a) $x + \overline{x} = 1$

(b) $x \bullet \overline{x} = 0$

Postulate 6 :

There are at least two elements $x, y \varepsilon M$ such that $x \neq y$.

Thus Boolean Algebra is defined on a set of elements M, together with two binary operations '+' and '$\bullet$' and unary * operation – (a bar over the variable), for which the above postulates are satisfied. The list of the postulates of Boolean Algebra are brief up indicated in the table 1.9.

* Note that unary operation is the operation over a single variable.

Table 1.9 : Postulates of Boolean Algebra

Sl. No. of Postulate	Operation	Expressions
1. (a)	+ operation	(a) $z = x + y$ } then z and w ∈ M
(b)	• operation	(b) $w = x \cdot y$ }
		for every pair of elements x, y ∈ M
2. (a)	Existance of an element 0	$x + 0 = x$ } for every element x ∈ M
(b)	Existance of an element 1	$x \cdot 1 = x$ }
3.	Commutative operation	(a) $x + y = y + x$
		(b) $x \cdot y = y \cdot x$
4.	Distributive operation	(a) $x \cdot (y + z) = x \cdot y + x \cdot z$
		(b) $x + (y \cdot z) = (x + y) \cdot (x + z)$
5.	Unary operation (–)	(a) $x + \bar{x} = 1$
	(a bar over the variable)	(b) $x \cdot \bar{x} = 0$
6.	Presence of the elements	Two elements x, y ∈ M such that $x \neq y$

1.10 THEOREMS OF BOOLEAN ALGEBRA :

Theorems of Boolean Algebra are useful in manipulating and simplifying the Boolean Algebraic expressions. Table 1.10 shows various theorems with their specific purposes.

Table 1.10 Theorems and Laws of Boolean Algebra

Sl. No.	Theorem	Specific Purpose
1.	$A + 0 = A$	OR operation
2.	$A + 1 = 1$	
3.	$A + A = A$	
4.	$A + \bar{A} = 1$	
5.	$A \cdot 0 = 0$	AND operation
6.	$A \cdot 1 = A$	
7.	$A \cdot A = A$	
8.	$A \cdot \bar{A} = 0$	
9.	$\bar{\bar{A}} = A$ (If A = 0, then $\bar{A} = 1$; A = 1, then $\bar{A} = 0$	Complement (NOT) operation (Involution)

10. 11.	$\overline{A+B} = \overline{A} \cdot \overline{B}$ $\overline{A \cdot B} = \overline{A} + \overline{B}$	De-Morgan's Theorems
12. 13.	$A + B = B + A$ $A \cdot B = B \cdot A$	Commutative Laws
14. 15.	$(A + B) + C = A + (B + C)$ $(A \cdot B) \cdot C = A \cdot (B \cdot C)$	Associative Laws
16. 17.	$A \cdot (B + C) = A \cdot B + A \cdot C$ $(A + B) \cdot C = A \cdot C + B \cdot C$	Distributive Laws

From these theorems, we must note the following

(1) **Boolean Algebra** is the algebra of binary variables 0 and 1; and these theorems are used to construct an example or realisation of a Boolean Algebraic expressions.

(2) The mathematical symbols '+' and '•' represent OR and AND logical operations respectively. The symbol ¯ (a bar over the variable) or unary operator represents NOT (or) the 'complement' of the logic operation.

(3) In these theorems, one part may be obtained from the other if '+' is interchanged with '•', and '0' is interchanged with '1'. This is known as '*duality*' and can be greatly utilised in the theorems.

(4) The knowledge of the simple logic expressions (or) operations is essential for simplification of various logical expressions and for the design of the digital systems.

(5) Based on the De–Morgan's theorems, we have two important universal operations, i.e. NAND and NOR.

(6) Each theorem may be proved by using the proof by *perfect induction*. For example, for the theorem 7 as shown in the table 1.10; the variable A can have only the value 0 or 1. If $A = 0, \Rightarrow 0 \cdot 0 = 0$ or if $A = 1, \Rightarrow 1 \cdot 1 = 1$. Hence $A \cdot A = 1$. Thus, this same technique may be utilised to prove the remaining theorems.

(7) The theorem 9 states that *double complementation* of a variable results in the original variable. If $A = 1$, then $\overline{A} = 0$ and $\overline{\overline{A}} = 1$ i.e., original value. If $A = 0$, then $\overline{A} = 1$ and $\overline{\overline{A}} = 0$ i.e., original value. Hence $\overline{\overline{A}} = A$. This operation is also known as 'involution'.

(8) The three laws; commutative, Associative and distributive (as stated in the table 1.10, the theorems from 12 to 17), may be extended to any number of terms; as follows :

$$A + B + C + Y + M = A + Y + M + B + C,$$

$$ABCM = BCMA,$$

$$A(B + C + D) = AB + AC + AD$$

Basically these laws are useful in regrouping the terms of an equation.

(9) The following simple logic expressions may be proved by using the preceding theorems, as follows :

(i) $A + AB = A(1 + B) = A$ ($\because (B + 1) = 1$ as per the theorem 2)

$\therefore A + AB = A$

(ii) $A \cdot (A + B) = A \cdot A + A \cdot B$

$= A + A \cdot B$ ($\because A \cdot A = A$ as per the theorem 7)

$= A(1 + B) = A$ ($\because (1 + B) = 1$ as per the theorem 2)

$\therefore A \cdot (A + B) = A$

(iii) $(A + B)(A + C) = AA + AC + BA + BC$

$= A + AC + BA + BC$ ($\because AA = A$ as per the theorem 7)

$= A(1 + C) + BA + BC$

$= A + BA + BC$ ($\because (1 + C) = 1$ as per the theorem 2)

$= A(1 + B) + BC$

$= A + BC$ ($\because (1 + B) = 1$ as per the theorem 2)

$\therefore (A + B)(A + C) = A + BC.$

(10) As we know that the theorems can also be proved by 'Perfect Induction'. Then it is necessary to construct the truth table for both left hand and right hand of the equation and compare the result. For example, let us verify the expression, i.e., $A + \overline{A}B = A + B$ by perfect induction, in the following truth table.

(Note that the truth table is a table that shows all possibilities of a variable)

A	B	$\overline{A}$	AB	A + AB	A + B
0	0	1	0	0	0
0	1	1	1	1	1
1	0	0	0	1	1
1	1	0	0	1	1

1.10.1 Simplification of Expressions :

We can simplify various Boolean expressions by using theorems or rules of Boolean Algebra; as illustrated in the following examples :

Example :

(1) Simplify the following logic expressions using Boolean Algebra.

(i) $(A + B)(A + \bar{B})$ (ii) $ABC + A\bar{B}C + AB\bar{C}$

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

Solutions :

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

$\Rightarrow A \cdot A + A \cdot \bar{B} + BA + B\bar{B}$

$\Rightarrow A + A\bar{B} + BA + 0$

($\because A \cdot A = A$ as per the theorem 7 and $B \cdot \bar{B} = 0$ as per the theorem 8)

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

$\Rightarrow A + A$ ($\because B + \bar{B} = 1$ as per the theorem 4)

$\Rightarrow A$ ($\because A + A = A$ as per the theorem 3)

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

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

$\Rightarrow AC(B + \bar{B}) + AB\bar{C}$

$\Rightarrow AC + AB\bar{C}$ ($\because B + \bar{B} = 1$ as per the theorem 4)

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

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

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

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

$\Rightarrow (A \cdot A + A\bar{B} + BA + B\bar{B})(\bar{A} + C)$

$\Rightarrow (A + A\bar{B} + BA + 0)(\bar{A} + C)$

($\because A \cdot A = A$ as per the theorem 7 and $B\bar{B} = 0$ as per the theorem 8)

$\Rightarrow [A + A(\bar{B} + B)][\bar{A} + C]$

$\Rightarrow (A + A)(\bar{A} + C)$ ($\because B + \bar{B} = 1$ as per the theorem 4)

$\Rightarrow A(\bar{A} + C)$ ($\because A + A = A$ as per the theorem 3)

$\Rightarrow A\bar{A} + AC \Rightarrow AC$ ($\because A\bar{A} = 0$ as per the theorem 8)

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

(iv) $AB + A(B + C) + B(B + C)$

$\Rightarrow AB + AB + AC + BB + BC$

$\Rightarrow AB + AC + B + BC$

($\because$ $AB + AB = AB$ as per two theorem 3 and $BB = B$ as per the theorem 7)

$\Rightarrow AB + AC + B(1 + C)$

$\Rightarrow AB + AC + B$ ($\because$ $1 + C = 1$ as per the theorem 2)

$\Rightarrow B(A + 1) + AC$

$\Rightarrow B + AC$ ($\because$ $1 + A = 1$ as per the theorem 2)

$\therefore AB + A(B + C) + B(B + C) = B + AC$

Example :

(2) Prove that (i) $A + \bar{A}B = A + B$

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

Solutions :

(i) $A + \bar{A}B = A(1 + B) + \bar{A}B$ ($\because$ as per the theorem 2, $1 + B = 1$)

$\Rightarrow A + AB + \bar{A}B$

$\Rightarrow AA + AB + \bar{A}B$ ($\because$ as per the theorem 7, $AA = A$)

$\Rightarrow AA + AB + A\bar{A} + \bar{A}B$ ($\because$ as per the theorem 8, $A\bar{A} = 0$)

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

$\Rightarrow A + B$ ($\because$ $A + \bar{A} = 1$ as per the theorem 4)

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

(ii) $AB + BC + \bar{B}C$

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

$\Rightarrow AB + C$ ($\because$ $B + \bar{B} = 1$ as per the theorem 4)

$\therefore AB + BC + \bar{B}C = AB + C$

1.11 BASIC LOGIC OPERATIONS :

From the study of the theorems, we know that the only logical operations involved in Boolean Algebra are OR, AND and NOT. These operations are also known as basic logical operations and frequently involved in the design of digital systems. Each of these functions is performed by an electronic circuit known as 'gate', such as OR gate, AND gate and NOT gate. These logic gates are the basic building blocks of digital systems. The study of these logic gate is essential for the design of digital systems.