DIGITAL LOGIC DESIGN

UNIT-I

BINARY SYSTEMS

* **Digital Systems:-**

→ A system which handles digital or discrete information that is called digital system.
→ Discrete information is restricted to a finite number of elements.
→ Generally information is either continuous or discrete in nature.
→ A system which handles analog or continuous or infinite information that is called analog system.
→ Compare with analog systems digital systems are more flexible because of limited number of elements.

Advantages:-

→ Easier to design.
→ Information storage is easy
→ More accuracy
→ Digital circuits are less effected by noise.

Applications:-

→ Digital systems are used in communication, business transactions, traffic control, medical treatment, weather monitoring, Internet etc.
→ We have digital telephones, digital television, digital cameras, and digital computers.
→ In digital systems generally discrete elements are digits.
→ Here we use two discrete elements called binary digits.
→ A binary digit called a bit. It has two values '0' & '1'.
→ Discrete elements of information are represented with groups of bits called binary code.
→ In this chapter we can study the various binary concepts.

* **Number Systems:-**

→ Number systems are useful in digital computers.
→ The knowledge of digital systems is necessary to perform easily understandable arithmetic operations.
→ Many digital systems are required with various digital codes to handle the data which is in the form of numbers, alphabets and symbols.

→ In any number system there is an ordered set of symbols called digits.
→ There are two types of number systems.
(1) Positional number systems
(2) Non Positional number systems.
→ In a positional number system, the position of each digit of a number indicates the significance to be attached to that digit.
→ In a non-positional number system, a digit of a number does not indicate any significance of its position.
→ Example for non-positional number system is roman numerical system. In this system zero is not present. So, it is difficult to use.
→ Most widely used system is positional number system.
→ A number is made up of a collection of digits and it has two parts - integer and fraction. Both are seperated by a radix point (.).
→ A positive number N can be represented by

$$N = a_{n-1} a_{n-2} \cdots a_1 a_0 \cdot a_{-1} a_{-2} \cdots a_{-m}.$$

N → number
n → number of digits in integer part
m → " " " " fractional part.
→ Number systems must contain a radix or base. It is defined as the weight of a digit which depends on its relative position within the number.
→ Also, radix of a number system is the number of symbols used in the number system.
→ Any positive number in the positional number system with base b can be represented by

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

Here a_m is called the least significant Digit (LSD)
a_{n-1} " " " most " " (MSD).

Decimal Number System:-

→ In day to day life, we can use decimal number system. It is a positional number system.
→ The base or radix of this system is 10.
→ This system contain 10 unique symbols to represent a number. They are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

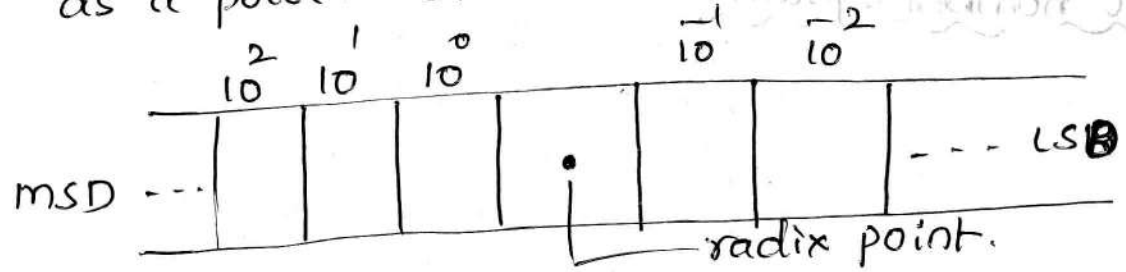
→ All the numbers in the system are represented using these digits.
→ We can express any decimal number in units, tens, hundreds, thousands and so on.
→ After reaching 9 in the decimal number system, we form combinations of decimal digits like 10, 11, 12...
→ For example a decimal number 456.72 can be represented by

$$(456.72)_{10} = 4\times10^{3-1} + 5\times10^{3-2} + 6\times10^{3-3} + 7\times10^{-1} + 2\times10^{-2}$$
$$= 400 + 50 + 6 + 0.7 + 0.02$$
$$= 456.72$$

(radix point)

Here MSD is 4
LSD is 2.

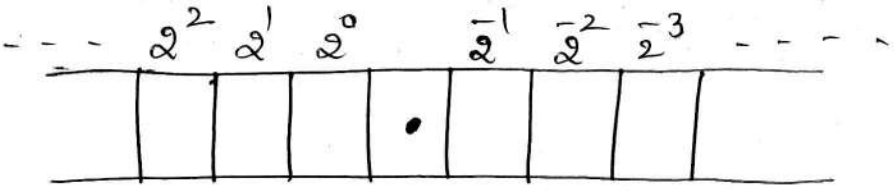
→ Below fig shows the digits and its weights represented as a power of 10.



Binary Number System:

→ Binary number system is mainly used in digital computers and in digital electronics.
→ It requires only two digits 1 and 0.
→ The base or radix of the system is 2. It is a positional number system.
→ The position of 1 or 0 in a number indicates its weight or value with in the number.
→ For example a binary number can be represented by 10110.1101. its value can be calculated by powers of 2. i.e.,

$$(10110.1101)_2 = 1\times2^{5-1} + 0\times2^{5-2} + 1\times2^{5-3} + 1\times2^{5-4} + 0\times2^{5-5} + 1\times2^{-1} + 1\times2^{-2} + 0\times2^{-3} + 1\times2^{-4}$$
$$= 16+0+4+2+1+0.5+0.25+0.125+0.0625$$
$$= (23.8375)_{10}$$

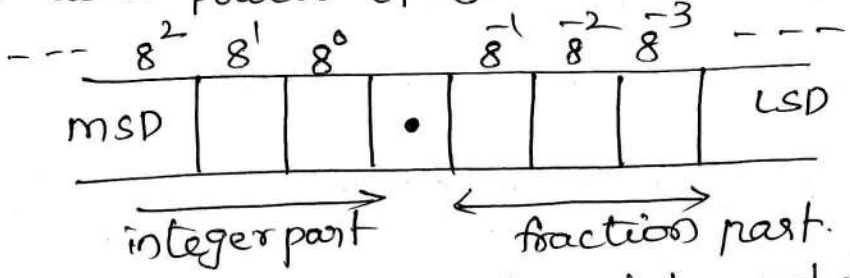


Octal number system:-

→ The base or radix of the octal system is eight.

→ Total symbols are 0 to 7. Here num 8 & 9 never appear. After 7 we will represent 10 & 11 and so on.

→ In the octal number system, each number weight is expressed as a power of 8.



Ex:- $(73.32)_8$ → valid octal number

But (79.23) is not a valid octal number

→ This type of number system used in UNIX & LINUX files.

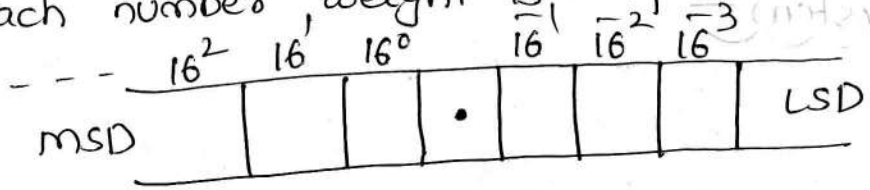
Hexa decimal number system:-

→ The base or radix of this system is 16.

→ symbols are 0,1,2,3,4,5,6,7,8,9, A, B, C, D, E & F.

→ It is also called as alphanumeric number system because it contains both alphabets as well as numbers.

→ Each number, weight is expressed as a power of 16



Ex:- A2H or $(A2)_{16}$.

→ This type of number systems used in microprocessors and microcontrollers.

Relation between decimal, binary, octal & Hexadecimal:-

Decimal	Binary	octal	Hexadecimal
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9

10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Other number systems:-

→ In any number system, any number can be represented as

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

Here N is a number b is the base or radix.

Radix (base) b	character (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

Number System Conversion:-

→ In digital system widely we use binary number systems.
→ In our everyday life we use decimal number system to represent quantities.
→ Therefore, it is necessary to convert decimal number into its binary and vice versa.
→ In some applications we use octal & hexadecimal also.
→ for this reason number system conversions are necessary.

① Binary-to-decimal Conversion:-

→ Any given binary number can be converted into its decimal equivalent by adding the products of each bit and its weight.
→ Representation of a binary number is

$$N_{10} = a_{n-1}2^{n-1} + a_{n-2}2^{n-2} + \cdots + a_1 2^1 + a_0 2^0 + a_{-1}2^{-1} + a_{-2}2^{-2} + \cdots + a_{-m}2^{-m}.$$

N_{10} is a decimal number
$a \rightarrow$ digit value (either 0 or 1).
$n \rightarrow$ number of digits in integer part
$m \rightarrow$ number of digits in fractional part

Ex ①:- Convert the binary number 10110 into decimal.

Sol:- Given binary number 10110 $\quad 4\ 3\ 2\ 1\ 0$

Corresponding weights are $2^4\ 2^3\ 2^2\ 2^1\ 2^0$

$\Rightarrow (2^4 \times 1) + (2^3 \times 0) + (2^2 \times 1) + (2^1 \times 1) + (2^0 \times 0)$

$\Rightarrow 16 + 0 + 4 + 2 + 0 = (22)_{10}$.

Ex ②:- Convert the binary number 1101.11 into decimal

Given binary number $\quad 1101.11 \quad 2\ 1\ 0 \cdot -1\ -2$

corresponding weights are $2^3\ 2^2\ 2^1\ 2^0 \cdot 2^{-1}\ 2^{-2}$

$\Rightarrow (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) + (1 \times 2^{-1}) + (1 \times 2^{-2})$

$\Rightarrow 8 + 4 + 0 + 1 + 0.5 + 0.25 \Rightarrow (13.75)_{10}$

② Binary-to-Octal Conversion:-

→ The base of the octal number is 8 and the base for binary number is 2. Thus the base for octal number is the third power of the base for binary number.

→ For integer part divide the binary number into group of three bits starting from the LSD & moving towards MSD. Then replace each group of 3 bits by its octal equivalent.

→ For fractional part grouping of 3 bits is made starting from the radix point & moving towards right.

Ex:- $\underleftarrow{10111} \cdot \underrightarrow{1010}$

Ex ③:- Convert the binary number 10110.01 into Octal.

Given binary number 10110.01

Grouping of bits $\quad \underbrace{10}_{②}\ \underbrace{110}_{①} \cdot \underbrace{010}_{③}$

convert each group into octal number.

① $\Rightarrow 110 = (1 \times 2^2) + (1 \times 2^1) + (0 \times 2^0) = 4 + 2 + 0 = 6$

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

③ $\Rightarrow 010 = (0 \times 2^0) + (1 \times 2^1) + (0 \times 2^2) = 0 + 2 + 0 = 2$.

octal number $= (26.2)_8$.

③ Binary-to-Hexadecimal Conversion:-

→ The base for hexadecimal number is 16 & the base for binary number is 2. The base for hexadecimal number is the fourth power of the base for binary number.

→ for integer part divide the binary number into group of four bits starting from the LSD & moving towards MSD. Then replace each group of 4 bits by its hexadecimal number.
→ For fractional part grouping of 4 bits is made starting from the radix point & moving towards right.

Ex:- Convert a binary number 10111100110.1100 into hexadecimal

Given binary number 10111100110.1100.

Grouping of 4 bits $\underbrace{101}_{③}\ \underbrace{1110}_{②}\ \underbrace{0110}_{①}.\ \underbrace{1100}_{④}$

Conversion of each group into hexadecimal

① ⇒ $0110 = (2^3\times0)+(2^2\times1)+(2^1\times1)+(2^0\times0) = 4+2=6$

② ⇒ $1110 = (2^3\times1)+(2^2\times1)+(2^1\times1)+(2^0\times0) = 8+4+2=14$
14 hexadecimal equivalent is E.

③ ⇒ $0101 = (2^2\times1)+(2^1\times0)+(2^0\times1) = 4+0+1=5$

④ ⇒ $1100 = (2^3\times1)+(2^2\times1)+(2^1\times0)+(2^0\times0) = 8+4=12$
12 hexadecimal equivalent is C.

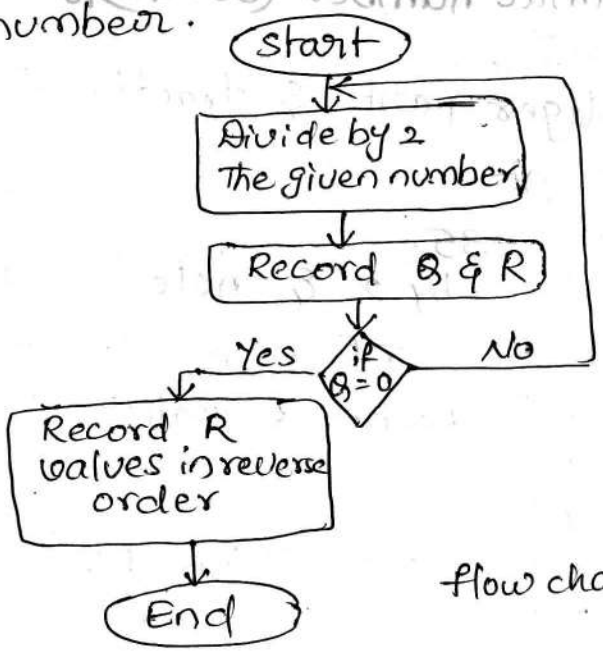
⇒ Hexadecimal number 5E6.C

④ Decimal-to-Binary Conversion:-

→ For the process of decimal to binary conversion we use double-dabble method.
→ In this method, successive division or successive multiplication by decimal number 2 is done for integer or fractional part respectively.

Successive division for integer part:-

→ Here we repeatedly divide the integer part by 2 until the quotient (Q) is zero.
→ Then the remainders are taken in the reverse order to form a binary number.



flow chart for integer part conversion.

Ex:- Convert a decimal number 25 into binary.

Given decimal number 25

Use successive division by 2 & note Q & R values.

25	Q	R	
25/2	12	1	LSD
12/2	6	0	
6/2	3	0	
3/2	1	1	
1/2	0	1	MSD

If Q=0 stop the process & Record R values in reverse order then

Binary number = $(11001)_2$

Successive multiplication for fractional part conversion:-

→ Here we use succession multiplication by 2 for fractional part conversion.

→ By multiplying a decimal number (fractional part) by 2, a product is produced and it has integer part & fractional part.

→ The integer part (carry) of the product becomes a bit in the binary number.

→ The fractional part is again multiplied by 2 & read carry until fractional part becomes zero.

Ex:- convert $(0.65625)_{10}$ into base-2.

Given decimal number 0.65625.

use successive multiplication by 2 & note carry values in order.

0.65625 × 2 → 1.3125 ↓ 1

0.3125 × 2 → 0.625 ↓ 0

0.625 × 2 → 1.250 ↓ 1

0.250 × 2 → 0.5 ↓ 0

0.5 × 2 → 1.0 ↓ 1 stop.

Binary number = $(0.10101)_2$.

Ex:- Convert a decimal number $(35.45)_{10}$ to binary num.

Given number $(35.45)_{10}$.

It consists both integer part & fractional part.

Integer part conversion:-

integer part = 35.

Use successive division by 2 & note Q, R values

35	Q	R
35/2	17	1
17/2	8	1
8/2	4	0
4/2	2	0
2/2	1	0
1/2	0	1

If Q=0 stop the process & Record R values in reverse order

for integer part binary number is 100011

fractional part conversion:-

Given fractional part is 0.45

Use successive multiplication by 2 & note carry values in order.

$0.45\times2 = 0.90$ → $0.90\times2 = 1.80$ → $0.80\times2 = 1.60$ → $0.60\times2 = 1.20$ → $0.2\times2 = 0.4$ → stop the process

0 1 1 1 0

fractional part binary equivalent = $(01110)_2$

Then Binary equivalent for decimal number

$(35.45)_{10} = (100011.01110)_2$

⑤ Decimal-to-octal conversion:-

→ For the conversion of decimal to octal we use octal dabble method.

→ In this method we use successive division or successive multiplication by 8 for integer part & or fractional part respectively.

Ex:- convert $(160)_{10}$ into octal number

Given number $(160)_{10}$ consists only integer part.

Use successive division by 8 & note Q, R values.

160	Q	R	
160/8	20	0	LSD
20/8	2	4	
2/8	0	2	MSD

If Q = 0 stop the process and note R values in reverse order.

$(160)_{10} = (240)_8$.

Ex:- convert $(10.20)_{10}$ into base-8 number

Given number $(10.20)_{10}$ consists both integer & fractional part.

Integer part conversion:-

Given integer part value 10

Use successive division by 8 & note Q, R values

10	Q	R	
10/8	1	2	↑
1/8	0	1	

If Q = 0 stop the process & note R values in reverse order.

Integer part equivalent is $(12)_8$.

fractional part conversion:-

Given fractional part 0.20

Use successive multiplication by 8 & note carry values in order

$0.20\times8 = 1.60$ → $0.60\times8 = 4.80$ → $0.80\times8 = 6.4$ → $0.40\times8 = 3.20$ → stop

1 4 6 3

fractional part equivalent is $(0.1463)_8$

Octal equivalent for $(10.20)_{10} = (12.1463)_8$

6) Decimal - to - Hexadecimal Conversion:-

→ Here we use hex-dabble method. Means successive division or multiplication by 16 for the conversion of integer part or fractional part respectively.

→ In integer part use successive division by 16 and record Remainder values in reverse order.

→ In fractional part use successive multiplication by 16 and note carry values in downward order.

Ex:- Convert $(3509)_{10}$ into Hexa decimal number

Given number $(3509)_{10}$

use successive division by 16 & note Q, R values

If Q=0; Record R values in reverse order.

3509	Q	R
3509/16	219	5
219/16	13	11 → B
13/16	0	13 → D

Hexadecimal number = $(DB5)_{16}$ (or) DB5H.

Ex:- Convert $(95.5)_{10}$ into $(X)_{16}$.

Given $(95.5)_{10}$ consists both integer & fractional part.

Integer part is 95.

Use successive division by 16 & note Q, R values.

If Q=0; Record R values in reverse order.

95	Q	R
95/16	5	15 → F
5/16	0	5

Equivalent integer part value = 5F H

fractional part is 0.5

use successive multiplication by 16 & note carry values.

0.5 × 16 → 0.00 stop

8.0 ↓ 8

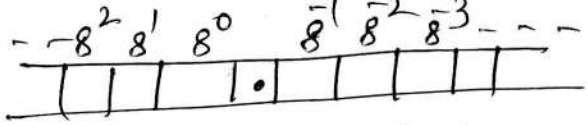
Equivalent fractional part value = 0.8 H

Equivalent Hexadecimal number is $(5F.8)_H$.

7) Octal - to - Decimal Conversion:-

→ To convert a octal number into decimal number, multiply each octal digit by its positional weight and add the resulting products.

→ octal positional weights are



Ex:- Convert $(475.25)_8$ to its decimal equivalent.

Given octal number $(475.25)_8$

positional weights are $8^2 8^1 8^0 . 8^{-1} 8^{-2}$.

$(N_{10}) = (4\times8^2)+(7\times8^1)+(5\times8^0)+(2\times8^{-1})+(5\times8^{-2})$

$= 256 + 56 + 5 + 0.25 + 0.078125$

$= (317.32825)_{10}$.

8) Octal-to-Binary Conversion:-

→ It is a reverse process of binary-to-octal conversion.

→ Here each octal digit converted into its equivalent 3bit binary numbers.

→ For conversion we use L.C.D process and not R values in reverse order.

Ex:- change octal number 65 into binary.

Given number $(65)_8$. It contains 2 digits.

Each digit converted into 3bit binary number using LSD by 2.

for digit

2	6
2	3 – 0
2	1 – 1
	0 – 1

for $(6)_8 = (110)_2$

for digit

2	5
2	2 – 1
2	1 – 0
	0 – 1

for $(5)_8 = (101)_2$.

Then $(65)_8 = (110101)_2$.

Ex:- Convert $(72.43)_8$ into binary equivalent.

Given number $(72.43)_8$ It contains 4 digits.

Each digit converted into 3 bit binary number using LSD by 2.

2	7
2	3 – 1
2	1 – 1
	0 – 1

$(7)_8 = (111)_2$

2	2
2	1 – 0
	0 – 1

$(2)_8 = (010)_2$

2	4
2	2 – 0
2	1 – 0
	0 – 1

$(4)_8 = (100)_2$

2	3
2	1 – 1
	0 – 1

$(3)_8 = (011)_2$

Then $(72.43)_8$ equivalent binary number is $(111010.100011)_2$

9) Octal-to-Hexadecimal Conversion:-

→ For the conversion of octal number into hexadecimal number we can use two phases.

(1) Convert octal number to its binary equivalent.

(2) Then convert the binary number to its hexadecimal equivalent.

→ We have other method for octal-to-hexa conversion i.e., convert octal-to-decimal and then decimal-to-hexadecimal.

Ex:- convert $(1234)_8$ into hexadecimal.

Given octal number 1234.

First convert octal-to-binary number. For this process use LCD of each digit and each digit converted into 3 bit binary.

2|1 → 0-1 ↑

2|2 → 2|1-0 ↑ → 0-1

2|3 → 2|1- ↑ → 0-1

2|4 → 2|2- ↑ → 2|1-0 ↑ → 0-1

$(1)_8 = (001)_2$ $(2)_8 = (010)_2$ $(3)_8 = (011)_2$ $(4)_8 = (101)_2$

→ $(1234)_8 = (001010011101)_2$.

→ Then convert binary-to-hexadecimal. Form a group of 4 bits & each group converted into hexadecimal number.

0010 (3) 1001 (2) 1100 (1)

(3) => 0010 weights are $2^3 2^2 2^1 2^0$.

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

(1) => 1100 => $(2^3\times1)+(2^2\times1) = 4+8 = 12 = C$

(2) => 1001 => $(2^3\times1)+(2^0\times1) = 8+1 = 9$.

=> $(1234)_8 = (29C)_{16}$

Ex:- Convert $(32.1)_8$ into Hexadecimal number.

Given octal number $(32.1)_8$

First convert octal-to-binary.

2|3 → 2|1- ↑ → 0-1

2|2 → 2|1-0 ↑ → 0-1

2|1 → 0-1 ↑

$(32.1)_8 = (011010.001)_2$.

Then convert binary to hexadecimal.

0001 (2) 1010 (1) . 0010 (3)

(1) => 1010 = $(2^3\times1)+(2^1\times1) = 2+8 = 10 = A$

(2) => 0001 = $2^0\times1 = 1$

(3) => 0010 = $2^1\times1 = 2$.

=> $(.32.1)_8 = (1A.2)_{16}$.

(10) Hexadecimal-to-decimal conversion:-

→ In hexadecimal, each digit position corresponds to a power of 16.

16^2	16^1	16^0	.	16^{-1}	16^{-2}

→ To convert hexadecimal number to decimal number, multiply each decimal digit by its weight and add the resulting products.

Ex:- Convert $(3A \cdot 2F)_{16}$ to $(X)_{10}$.

Given hexadecimal number 3A.2F.

Positional weights are $16^1\ 16^0 \cdot 16^{-1}\ 16^{-2}$.

$N_{10} = (3\times16^1) + (A\times16^0) + (2\times16^{-1}) + (F\times16^{-2})$.

Here A = 10 ; F = 15

$= 48 + 10 + 0.125 + 0.05859375$

$= (58.1816)_{10}$

11) Hexadecimal - to - Binary conversion:-

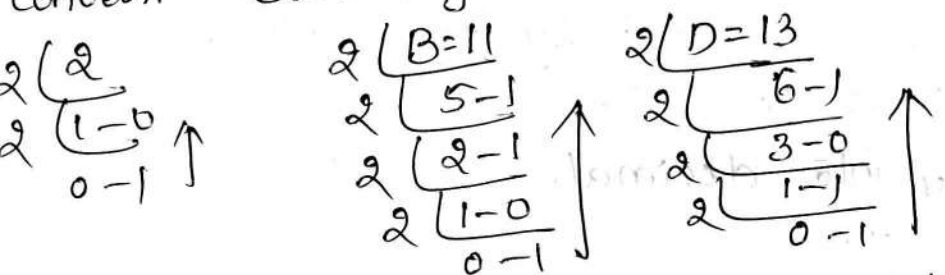
→ It is the reverse process of binary to hexadecimal conversion

→ Here every hexadecimal number is converted into its equivalent 4-bit binary number.

Ex:- Convert $(2BD)_{16}$ to binary.

Given hexadecimal number $(2BD)_{16}$.

Convert each digit into 4bit binary using LCD by 2



$(2)_{16} = (0010)_2$ $(B)_{16} = (1011)_2$ $(D)_{16} = (1101)_2$.

Binary number = $(10101111101)_2$.

12) Hexadecimal - to - octal conversion:-

1) First convert Hexa decimal number into binary number.

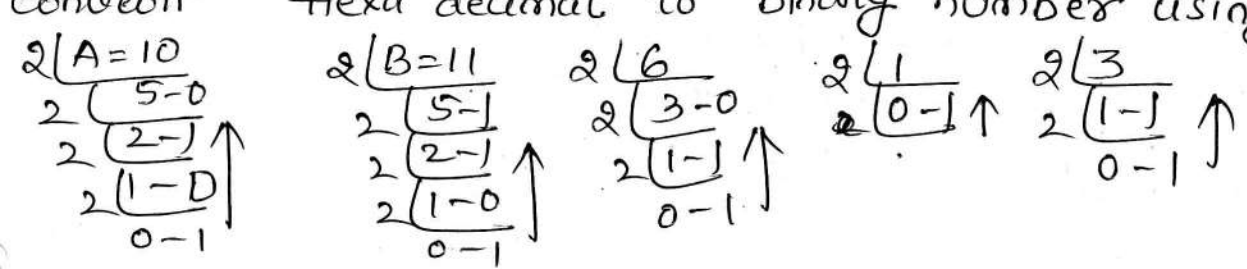
2) Then convert the binary number into its equivalent octal number.

→ other method is to convert hexa to decimal & then to octal number

Ex:- Convert $(AB6.13)_{16}$ into octal number.

Given $(AB6.13)_{16}$.

Convert Hexa decimal to binary number using LCD by 2.



$(A)_{16} = (1010)_2$ $(B)_{16} = (1011)_2$ $(6)_{16} = (0110)_2$ $(1)_{16} = (0001)_2$ $(3)_{16} = (0011)_2$.

$\Rightarrow (AB6.13)_{16} = (10101011011 0.00010011)_2$

→ Then Binary-to-octal conversion.
Form a group of 3 bits & convert each group into octal

⇒ $\underbrace{101}_{①}\ \underbrace{010}_{②}\ \underbrace{110}_{③}\ \underbrace{110}_{④} . \underbrace{000}_{⑤}\ \underbrace{100}_{⑥}\ \underbrace{110}_{⑦}$

① $101 = (2^2 \times 1) + (2^0 \times 1) = 4+1 = 5$
② $010 = 2^1 \times 1 = 2$
③ $110 = (2^2 \times 1) + (2^1 \times 1) + 0 = 4+2 = 6$
⑤ $000 = 0$
④ $110 = 6$
⑥ $100 = 2^2 \times 1 = 4$
⑦ $110 = (2^2 \times 1) + (2^1 \times 1) = 6$

⇒ octal number is $(5266.046)_8$

13) Other Radix/Base system-to-Decimal conversion:-

→ Any base system number can be converted into decimal system by multiplying the each digit by its positional weight and adding the resulting products.

Ex:- Convert $(23)_4$ into decimal.

Given number $(23)_4$.

positional weights $4^1\ 4^0$.

$N_{10} = (4^1 \times 2) + (4^0 \times 3) = 8+3 = 11$ $\Rightarrow (23)_4 = (11)_{10}$

14) Decimal-to-other radix/base system conversion:-

→ The integer part of a decimal number can be converted into its equivalent radix system by successive division by r and note remainder values in reverse order.

→ The fractional part of a decimal number can be converted into its equivalent radix system by successive multiplication by r and note carry values in order.

Ex:- Convert $(22.1)_{10}$ into base-3 system.

Given $(22.1)_{10}$.

Integer part 22. divide by 3.

22	Q	R
22/3	7	1
7/3	2	1
2/3	0	2

↑ note R values in reverse order.

$(22)_{10} = (211)_3$.

fractional part 0.1 multiplied by 3.

$0.1 \times 3 = 0.3$ → $0.3 \times 3 = 0.9$ → $0.9 \times 3 = 2.7$ → $0.7 \times 3 = 2.1$ → $0.1 \times 3 = 0.3$ stop.

0 0 2 2 0

$(0.1)_{10} = (0.0022)_3$.

Then $(22.1)_{10} = (211.0022)_3$.

* Binary Arithmetic Operations:-

→ Digital systems can handle only binary numbers. They do not process the decimal numbers.

→ Therefore it is necessary to learn the binary arithmetic operations. They are Binary addition
" Subtraction
" multiplication &
" division

Binary Addition:-

→ If A & B are two one bit binary numbers. Then the rules for addition of A & B are shown below.

A	B	sum	carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Here sum & carry are results after addition when both inputs are 1 then we will get carry.

→ If A & B are more than one bit numbers then we will follow the rules shown below.

carry-in	A	B	sum	carry-out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

In sum of 2 LSD bits if carry generated it is added to next sum of next significant bits. & it is represented as carry-in.

Ex:- Add $(1010)_2$ & $(0101)_2$.

Given two numbers are 1 0 1 0

(+) 0 1 0 1

Binary addition 1 1 1 1 $= (1111)_2$.

Binary Codes

→ Generally digital systems like computers, microprocessors use binary values means they can handle data in form of 1s and 0's.
→ But data consists numbers, alphabets, and some special characters.
→ So, all these numbers, alphabets & special characters are to be converted into binary format. "
→ This process of conversion is known as "binary coding."
→ The combinations of binary bits that represent numbers, alphabets & special characters are called "binary codes".
→ for the representation of 2^n distinct elements in a binary code requires a minimum of n bits.
→ If we consider $n=2$,
Total combinations are 4. They are 00, 01, 10, & 11.
They represent a decimal numbers from 0 to (2^n-1).
i.e., 0 to 3.

* Classification of binary codes:-

→ Binary codes are classified in different ways. They are
(1) Weighted codes (2) Non weighted codes (3) reflective codes (4) Sequential codes (5) Alphanumeric codes & (6) error detecting and correcting codes.

(1) **Weighted codes:-** The main characteristic of a weighted code is, each binary bit assigned by a weight and values depends on the position of the binary bit.
→ If $w_0, w_1, , w_2$ are the weights of the binary digits, and $x_0, x_1, , x_2$ are the corresponding bit values, then the decimal number $N = w_2x_2 + w_1x_1 + w_0x_0$ is represented by a binary sequence $x_2x_1x_0$ & it is called "code word."
Ex:- $\overset{2^2\ 2^1\ 2^0}{101}$ is a code word then

$$N = 2^0 \times 1 + 2^1 \times 0 + 2^2 \times 1 = 1+0+4 = 5 \checkmark$$

Ex:- Binary code, BCD codes.

(2) **Non weighted codes:-** (unweighted codes).
→ Here the bit value does not depends upon their position means each digit position within the number is not assigned fixed value.

Ex:- Gray codes, Excess-3 code & 5-bit BCD code.

(3) Reflective codes:- (self complimenting code).

→ A code is said to be a reflective code, if the codeword of 9's complement of N can be obtained from the code word of N by interchanging all the 1's as 0's & 0's as 1's.

Ex:- 5211, 4221, 3321 etc.

For example consider 5211 code → the codeword for 9 is the complement for the code 0.

Here 9 number 9's comp of number 9 is $\begin{array}{r} 9 \\ -9 \\ \hline 0 \end{array}$ ✓

and number 9 1's comp becomes

9 represents 1111

1's comp 0000 → value is 0 ✓

So, codeword for 9 is the complement for the code 0.

(4) Sequential code:- In sequential codes, each succeeding code is one binary number greater than its preceeding code.

Ex:- Binary codes, Ex-3 codes.

(5) Alphanumeric codes:- these codes are mainly used to represent numbers, alphabets & some special characters.

→ A set consists 10 decimal digits, 26 letters and number of special characters.

→ It contains between 36 to 64 elements if only capital letters are included, and between 64 to 128 elements if both Upper & lower case letters are included.

→ for 36 to 64 elements require 6 bit codeword.

→ " 64 to 128 " " 7 bit codeword.

Ex:- ASCII, EBCDIC & Hollerith code.

ASCII - American standard code for information interchange

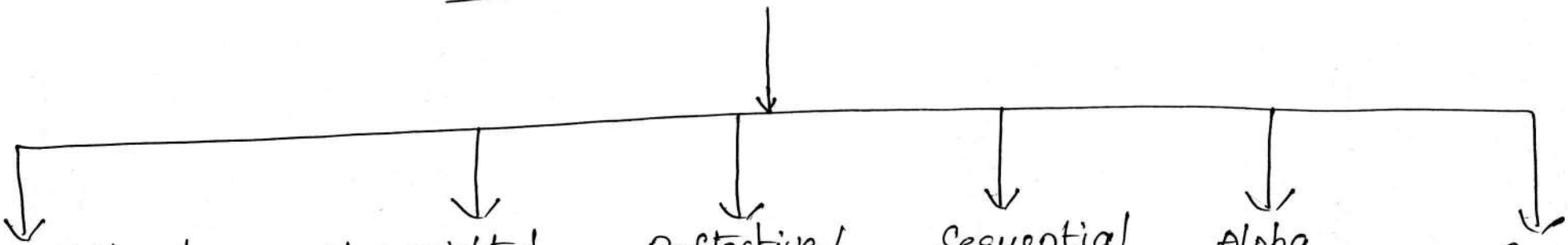
EBCDIC - Extended Binary coded decimal interchange code

(6) Error detecting & correcting codes:- when binary information is transmitted from transmitter to receiver, there is a chance of getting noise or error. Due to this error binary 0 may be changed as 1 or vice versa.

→ To avoid or eliminate this error we just add extra bits to binary data. These bits allow the detection & some times correction of errors.

Ex:- Parity (for detection), Hamming code (for correction).

classification of Binary codes



Examples:-

Weighted code	Nonweighted code	Reflective/ self complimenting code	Sequential code	Alpha numeric code	Error detecting & correcting code
Binary code BCD – 8421 4221 3321 5211 ~~5221~~ 5421 6311 7421 74$\bar{2}$1 84$\bar{2}$1	Gray code Ex-3 code 5-bit BCD code	2421 5211 Ex-3	8421 Ex-3	ASCII EBCDIC Hollerith	parity (error detection) Hamming code (error correction)

Application:-

Weighted code	Nonweighted code	Reflective/ self complimenting code	Sequential code	Alpha numeric code	Error detecting & correcting code
* To represent a decimal digits in calculators & voltmeters	* To perform certain arithmetic operations	* To perform 9's complement - Subtraction	* In mathematical manipulation of data	* To transfer information between computers & printers * So many we use applications of digital ASCII code Computers	* In communi-cation systems

BCD code :-

BCD – Binary coded decimal.

→ In BCD code decimal digits 0–9 are represented by their binary equivalents using four bits, i.e., BCD is a numeric code in which each decimal digit is represented by a seperate binary code of four bits.

→ The most common BCD code is 8-4-2-1 code.

→ In this code weights are 8, 4, 2, 1 from left to right.

Ex:- Representation of ~~Binary~~ decimal number 14 into BCD.

Given decimal number = 14 consists 2 digits.

Each digit represented by four bit binary.

2 | 1 → 0-1 ↑

2 | 4
2 | 2-0 ↑
2 | 1-0
0-1

$(14)_{10} = (0001\ 0100)_{BCD}$

Ex:- Convert $(78.216)_{10}$ into BCD.

Given decimal number = 78.216.

Each digit converted into 4 bit binary numbers by using successive division by 2.

2 | 7
2 | 3-1
2 | 1-1 ↑
0-1

2 | 8
2 | 4-0
2 | 2-0 ↑
2 | 1-0
0-1

2 | 2
2 | 1-0 ↑
0-1

2 | 1 → 0-1 ↑

2 | 6
2 | 3-0 ↑
2 | 1-1
0-1

7 8 . 2 1 6

$(0111\ \ 1000\ .\ 0010\ 0001\ 0110)_{BCD}$

BCD Addition :-

→ we can perform BCD addition when the addition of two digits does not exceed 9.

Ex:- Add the following numbers $(84)_{10} + (15)_{10}$ using BCD addition

8 4	=	1000 0100
1 5	= (+)	0001 0101
9 9	=	1001 1001

Ex:- Add $(98)_{10}$ & $(28)_{10}$ using BCD addition

9 8	=	1001 1000
2 8	=	0010 1000
1 2 6	=	1 1000 0000

It is not a valid BCD addition. Because sum of 2 digits greater than 9.

* Other 4-bit BCD codes:-

→ other 4-bit BCD codes are 4221, 3321, 5211, 5421, 6311, 7421, 74$\bar{2}$$\bar{1}$, 84$\bar{2}$$\bar{1}$.

→ In 4221 code, the weights are 4-2-2-1 means bit 1 has the weight 1, bit 2 & bit 3 have the same weight 2 and bit 4 has the weight 4.

→ Here $\bar{2}$$\bar{1}$ are different they represent negative weights.

Decimal number	4221	3321	5211	5421	6311	7421	74$\bar{2}$$\bar{1}$	84$\bar{2}$$\bar{1}$
0	0000	0000	0000	0000	0000	0000	0000	0000
1	0001	0001	0001	0001	0001	0001	0111	0111
2	0010	0010	0011	0010	0011	0010	0110	0110
3	0011	0011	0101	0011	0100	0011	0101	0101
4	0110	0101	0111	0100	0101	0100	1011	0100
5	1001	0110	1000	0101	0111	0101	1010	1011
6	1010	0111	1001	0110	1000	0110	1001	1010
7	1011	1101	1011	0111	1001	0111	1000	1001
8	1110	1110	1101	1011	1011	1001	1111	1000
9	1111	1111	1111	1100	1100	1010	1110	1111

Non weighted codes:-

Excess-3 code / Ex-3 / XS3:-

→ Ex-3 code is a modified form of a BCD code.

→ The Excess-3 code can be obtained from the natural BCD code by adding 3 to each coded number i.e., add 3 to each decimal number & then convert into 4 bit binary.

→ Here no fixed weights are assigned.

Decimal	BCD	Ex-3
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

Ex:- find Ex-3 of $(36)_{10}$.

Given decimal number 36. Add 3 to each digit.

$$\begin{array}{r} 36 \\ +\ 33 \\ \hline 69 \end{array}$$

Convert each digit into 4-bit binary.

$(36)_{10} = (0110\ 1001)_{Ex-3}$.

Ex:- find Ex-3 code for $(72.9)_{10}$.

Given decimal number 72.9. Add 3 to each digit

$$\begin{array}{r} 72.9 \\ +\ 3\ 3\ 3 \\ \hline 10\ 5.12 \end{array}$$

Convert each digit into 4-bit binary.

Ex-3 code = 10100101 . 1100

Note:- Ex-3 code is self complementing / reflective code.

→ Here 1's complement of an ex-3 code number is the Ex-3 code for the 9's complement of the respective decimal number

Ex:- Consider decimal number = 4.

Ex-3 code for 4 is = 4+3 = 7 = 0111

1's complement of ex-3 of 4 is = 1000 = (8) ✓

9's complement for 4 is = 9-4 = 5

Ex-3 code for 5 is = 5+3 = (8) ✓

Note:- for a digit in any number system to be converted into Ex-3 code, first it should be converted into Decimal and then to Ex-3 code and vice versa.

Ex:- Convert $(1011)_2$ into Ex-3 code.

Given binary number = 1011.

convert this binary number into decimal number.

$2^3\ 2^2\ 2^1\ 2^0$

$1\ 0\ 1\ 1 = (2^3 \times 1) + (2^1 \times 1) + (2^0 \times 1) = 8+2+1 = 11$

$\Rightarrow (1011)_2 = (11)_{10}$. add 3 to each digit

$$\begin{array}{r} 11 \\ +\ 3\ 3 \\ \hline 44 \end{array}$$

convert each digit into 4-bit binary

Ex-3 = 01000100.

Ex:- convert $(3A)_{16}$ into Ex-3 code

convert $(3A)_{16}$ into decimal = $3\ A_{10} = (3 \times 16^1) + (10 \times 16^0) = 48+10 = (58)_{10}$

add 3 to each digit.

$$\begin{array}{r} 58 \\ +\ 33 \\ \hline 811 \end{array}$$

Convert each digit into binary.

Ex-3 = 1000 1011

Ex-3 Addition :-

The addition operation may be explained in two cases.

Procedure :-

(1) Add the given binary numbers using rules for binary addition
(2) If any group produces a decimal carry, add 0011 to that group.
(3) If any group does not produce a decimal carry, subtract 0011 from that group.

Case (i) :- In Ex-3 code, If we add decimal digits whose sum is 9 or less, an Ex-6 number results. To get Ex-3 form we must subtract 3.

Ex :- Perform the following addition operations in Ex-3 code.

72.6 + 15.2

A = 7 2 . 6
    3 3 3
  ----
  1 0 5 . 9

add 3 to each digit

convert each digit into 4bit binary.

Ex-3 code for A = 1010 0101 . 1001

B = 1 5 . 2
    3 3 3
  ----
    4 8 . 5

Ex-3 code for B = 0100 1000 . 0101

(+)

Ex-6 of (A+B) 1110 1101 . 1001

Subtract '3' to get result in Ex-3 form.

1110 1101 . 1001
(-) 0011 0011 . 0011

1010 1010 . 0110 = [illegible]

Case (ii) :- In Ex-3 code, If we add decimal digits whose sum is greater than 9, add 3 to the result to get Ex-3 form.

Ex :- 7+3 use Ex-3 Addition operation

A = 7 add 3 to each digit
  = 7+3 = 10 convert it into 4 bit binary

Ex-3 code for A = 1010

B = 3 add 3 to each digit
  = 3+3 = 6

Ex-3 code for B = 0110

Ex-3 A = 1010
Ex-3 B = 0110
         1 1

10000

⇒ add 3 to the result to get Ex-3 form.

10000
  0011

10011

Ex:- Add 28 & 36 using BCD addition.

A = 28
33
---
511 = Ex-3 A = 0101 1011

B = 36
33
---
69 = Ex-3 B = 0110 1001
1111 11

1100 0100

[decimal sum < 9] subtract 0011 ↙ ↘ add 0011 [decimal sum > 9]

1100 0100
−0011 +0011

Ex-3 added result = 1001 0111

* Gray code:-

→ The code which exhibits only a single bit change from one number to the next is known as "gray code".
→ means In this code between any two successive code words there will be a change in only one position.
→ It is also called as "unit distance code" or "cyclic code".
→ No specific weight assigned to each bit position, so, it is a non weighted code.
→ Gray code is also called as 'reflected code' because n-bit gray code is obtained by reflecting the $(n-1)^{st}$ bit code.

2-bit gray code	3-bit gray code	4-bit gray code
g_1 g_0	g_2 g_1 g_0	g_3 g_2 g_1 g_0
0 0 ←	0 0 0	0 0 0 0
0 1 ←	0 0 1	0 0 0 1
1 1 ←	0 1 1 ←	0 0 1 1
1 0 ←	0 1 0 ←	0 0 1 0
	1 1 0 ←	0 1 1 0
	1 1 1 ←	0 1 1 1 ←
	1 0 1	0 1 0 1 ←
	1 0 0	0 1 0 0 ←
		1 1 0 0 ←
		1 1 0 1 ←
		1 1 1 1 ←
		1 1 1 0
		1 0 1 0
		1 0 1 1
		1 0 0 1
		1 0 0 0

* Gray to Binary Conversion :-

Procedure :-

(1) Number of bits in a gray code = no of bits in binary code.
(2) The MSB of the binary number is same as the MSB of the gray code, so write it down.
(3) To obtain the next binary bit, perform exclusive OR addition or addition with neglecting carry between the just written down binary bit and the next gray code bit. write down the result.
(4) Repeat the step 3 untill all gray code bits completed.

→ let $g_n g_{n-1} \cdots g_1 g_0$ → gray code.
$b_n b_{n-1} \cdots b_1 b_0$ → binary code.

$g_n = b_n$. $b_{n-1} = g_{n-1} \oplus b_n$.
$b_{n-2} = g_{n-2} \oplus b_{n-1}$ etc ...

Ex :- convert 10101111 gray code into binary code

A	B	Ex-OR
0	0	0
0	1	1
1	0	1
1	1	0

Gray code 1 0 1 0 1 1 1 1

Binary code 1 1 0 0 1 0 1 0

* Binary to gray Conversion :-

Procedure :-

(1) The MSB of binary number is same as the MSB of the gray code, so write it down.
(2) To obtain the next gray code bit, perform exclusive OR addition between preset binary code & next binary code bit. write it down.
(3) Repeat the step 2 untill all binary bits completed

Here $b_n = g_n$ $g_{n-1} = b_n \oplus b_{n-1}$
$g_{n-2} = b_{n-1} \oplus b_{n-2}$

Ex :- Convert 11001010 binary code into gray code

Binary code 1 1 0 0 1 0 1 0

Gray code 1 0 1 0 1 1 1 1

* 5 Bit BCD codes :-

→ 5 bit codes also exist. 5 bit BCD codes are useful for error detection. They are 51111 code and shift counter code. These 2 five bit BCD codes are nonweighted codes.

Decimal	51111	shift counter
0	00000	00000
1	00001	00001
2	00011	00011
3	00111	00111
4	01111	01111
5	10000	11111
6	11000	11110
7	11100	11100
8	11110	11000
9	11111	10000

* Alphanumeric Codes :-

ASCII Code :-

ASCII - American Standard Code for Information Interchange.

→ It is a widely used alphanumeric code. This is basically a 7-bit code.

→ Total number of different bit patterns are $2^7 = 128$.

→ But most computers manipulate an 8-bit quantity as a single unit called a byte. So the extra bit used for other purposes depending on the applications.

Ex :- Simple printer recognise 8 bit ASCII characters with the MSB set to 0.

→ ASCII can be used to convert both the uppercase, lowercase characters of alphabets (26+26) and numbers (0 to 9) and some special symbols.

→ Some used symbols are @ = 64 ; space = 32 ;

uppercase alphabets = 64+ ; lowercase alphabets = 96+

Ex :-

here, base = 47 + 8 = 55

55 converted into ASCII (7 bit binary)

= 0110111.

Ex:- Code @5CSE in ASCII

Here

	64	32	16	8	4	2	1
@ = 64 =	1	0	0	0	0	0	0
5 = 54 =	0	1	1	0	1	1	0
C = 67 =	1	0	0	0	0	1	1
S = 83 =	1	0	1	0	0	1	1
E = 69 =	1	0	0	0	1	0	1

@ 5CSE ASCII code = 1000000 1101101000011101001 1000101.

EBCDIC code:-

standard for Extended binary coded decimal interchange code.

→ It is a 8-bit code. [$2^8 = 256$ bit patterns]

→ It is a standard code for large computers.

Hollerith code:- This is a code essentially used with the punched cards. In this each character is represented as a sequence of 1's & 0's. This is a 12-bit code.

Error detecting codes:-

Parity Code:-

→ Parity code is a error detecting code. Here we can detect error just by adding an extra bit called as parity bit to each codeword before transmitting.

→ There are two types of parity

(1) Even parity

(2) Odd parity

Even parity:- For even parity, the parity bit is set to 0 or 1 at the transmitter in such a way that the total number of 1's in the word including the parity bit is an even number.

Odd Parity:- For odd parity, the parity bit is set to 0 or 1 at the transmitter in such a way that the total number of 1's in the word including the parity bit is an odd number.

odd & even parity in the 8421 BCD code

Decimal	8421 BCD	even parity	odd parity
0	0000	0	1
1	0001	1	0
2	0010	1	0
3	0011	0	1
4	0100	1	0
5	0101	0	1
6	0110	0	1
7	0111	1	0
8	1000	1	0
9	1001	0	1

→ When data is received, a parity checking circuit generates an error signal if the total number of 1's is even in odd-parity system or odd in an even-parity system.
→ This parity check can always detects a single bit error but cannot detect two or more errors within the same word.
→ But widely used odd parity system than even parity system because even parity does not detect the situation when all 0's are created by a short circuit or some other fault condition.

Ex:- In odd parity system, which of the following words contain an error
(a) 10110111 (b) 10011010 (c) 11101010

Sol:- (a) 10110111 = total num of 1's 6 [even] in odd parity system. So, this word has an error
(b) 10011010 = total num of 1's 4 [even]. So, this word has an error.
(c) 11101010 = total num of 1's 5 [odd]. So, this word does not have any error.

Ex:- In an even parity scheme, which of the following words contain error.
(a) 10101010 (b) 11110110 (c) 10111001

(a) 10101010 = total num of 1's 4 even. no error
(b) 11110110 = total num of 1's 6 even. no error.
(c) 10111001 = total num of 1's 5 odd. So, the word has an error

* Error Correcting Codes :-

→ Hamming code is an error correcting code.
→ First it can detect the error and the correct the codeword from an erroneous word.

→ Hamming Code :-

→ First we will discuss about procedure

Procedure :-

① Determine the number of parity bits required from the code that is obtained by calculating 'p' value which satisfy the following expression. where p is no of parity bits

$$2^p \geq m+p+1$$

m = number of information/data bits

→ To find 'P' Here we use trail and error method.

① Ex:- Findout the no of parity bits for constructing the hamming code for 1001.

Sol:- Consider trail and error method

let $P = 3$. $\quad 2^P \geq m + P + 1$

Given data 1001

$m = 4$

$P = 3$

$2^3 \geq 4 + 3 + 1$

$8 \geq 8$ (satisfied).

∴ Minimum number of parity bits are $\boxed{P=3}$.

② The 'k' parity checking bits denoted by $P_1, P_2 \text{----} P_k$ located at positions 2^{k-1} from left are added to form an (m+k) bit code word.

* Here 3 parity checking bits denoted by P_1, P_2, P_4 located at positions $2^{3-1} = 4$ from left are added to form an $(m+k) = (4+3) = 7$ bit code word.

→ The bits checked for parity can be noted from below table. This table lists the error positions and corresponding values of the position number for 15-bit, 12-bit & 7-bit Hamming codes.

Error Position	For 15-bit code P_8	P_4	P_2	P_1	For 12-bit code P_8	P_4	P_2	P_1	For 7-bit code P_4	P_2	P_1
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1	0	0	1
2	0	0	1	0	0	0	1	0	0	1	0
3	0	0	1	1	0	0	1	1	0	1	1
4	0	1	0	0	0	1	0	0	1	0	0
5	0	1	0	1	0	1	0	1	1	0	1
6	0	1	1	0	0	1	1	0	1	1	0
7	0	1	1	1	0	1	1	1	1	1	1
8	1	0	0	0	1	0	0	0			
9	1	0	0	1	1	0	0	1			
10	1	0	1	0	1	0	1	0			
11	1	0	1	1	1	0	1	1			
12	1	1	0	0	1	1	0	0			
13	1	1	0	1							
14	1	1	1	0							
15	1	1	1	1							

(3) Consider the table having Bit location, bit designation, information bits, parity bits.

(4) following table is an example for 7-bit hamming code with data 1001.

Bit location	7	6	5	4	3	2	1
Bit designation	D_7	D_6	D_5	P_4 D_4	D_3	P_2 D_2	P_1 D_1
message bits	m_7	m_6	m_5		m_3		
Parity bits				P_4		P_2	P_1

(5) Then findout parity bit values P_1, P_2 & P_4.

(6) from the table we observe that for 7-bit code P_1 is to be set to 0 or a 1 so that it establishes even parity bits 1, 3, 5, and 7 (i.e., P_1 D_3 D_5 D_7).

P_2 is to be set to 0 or 1 so that it establishes even parity over bits 2, 3, 6 & 7 (i.e., P_2 D_3 D_6 D_7)

P_4 is to be set to 0 or 1 so that it establishes even parity over bits 4, 5, 6, & 7 (i.e., P_4 D_5 D_6 D_7).

(7) Then in this example

Bit location	7	6	5	4	3	2	1	
Bit designation	D_7	D_6	D_5	D_4	D_3	D_2	D_1	
message bits	m_7	m_6	m_5		m_3			
Parity bits				P_4		P_2	P_1	
P_1 (even parity) at locations 1,3,5,7	1		0		1		0	$P_1=0$
P_2 (even parity) at locations 2,3,6,7	1	0			1	0		$P_2=0$
~~at~~ P_3 (even parity) at locations 4,5,6,7	1	0	0	1				$P_4=1$
Required code	1	0	0	1	1	0	0	

(8) finally represent Hamming code D_7 D_6 D_5 D_4 D_3 D_2 D_1.

Problem: Find out the Hamming code for the given data 01011011 [even parity]

Sol:- Given data/information '01011011' (8 bit data)

① First find Parity checking bits from the following expression $2^p \geq p+m+1$ here $m=8$

use trail & error method

let $p=3$ $\quad 2^3 \geq 3+8+1$

$8 \geq 12$ ✗

let $p=4$ $\quad 2^4 \geq 4+8+1$

$16 \geq 12$ ✓ satisfied.

Minimum number of parity checking bits are 4.

② Locate Parity checking bits for k bits $P_1, P_2 \cdots P_k$ with 2^{k-1} locations.

for 4 parity checking bits locations are

$P_k = P_4 = 2^{1-1} = 2^0 = 1$ (P_1)

$2^{2-1} = 2^1 = 2$ (P_2)

$2^{3-1} = 2^2 = 4$ (P_4)

$2^{4-1} = 2^3 = 8$ (P_8)

locations for 4 parity checking bits are P_1, P_2, P_4 and P_8.

③ Represent message bits, parity checking bits using table.

Bit location	12	11	10	9	8	7	6	5	4	3	2	1
Bit Designation	D_{12}	D_{11}	D_{10}	D_9	D_8	D_7	D_6	D_5	D_4	D_3	D_2	D_1
message bits	m_{12}	m_{11}	m_{10}	m_9		m_7	m_6	m_5		m_3		
Parity bits					P_8				P_4		P_2	P_1

④ From the table of error positions for 12-bit data we observe that

P_1 (even parity) locations 1, 3, 5, 7, 9, 11

P_2 (even parity) locations 2, 3, 6, 7, 10, 11

P_3 (~~odd~~ even parity) 4, 5, 6, 7, 12

P_4 (~~odd~~ even parity) 8, 9, 10, 11, 12

Error positions	P_8	P_4	P_2	P_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
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0

⑤ Find P_1, P_2, P_4 & P_8 using even and odd parity

Bit locations	12	11	10	9	8	7	6	5	4	3	2	1	
Designation	D_{12}	D_{11}	D_{10}	D_9	D_8	D_7	D_6	D_5	D_4	D_3	D_2	D_1	
msg bits	m_{12}	m_{11}	m_{10}	m_9		m_7	m_6	m_5		m_3			
Parity bits					P_8				P_4		P_2	P_1	
Given msg	1	1	0	1		1	0	1		0			
P_1 (even parity) locations 1,3,5,7,9,11		1		1		1		1		0		0	$P_1=0$
P_2 (even parity) 2,3,6,7,10,11		1	0			1	0			0	0		$P_2=0$
P_4 (even parity) locations 4,5,6,7,12	1					1	0	1	1				$P_4=1$
P_8 (even parity) locations 8,9,10,11,12	1	1	0	1	1								$P_8=1$
Hamming code	1	1	0	1	1	1	0	1	1	0	0	0	

Hamming code $m_{12}\ m_{11}\ m_{10}\ m_9\ P_8\ m_7\ m_6\ m_5\ P_4\ m_3\ P_2\ P_1$ is 110111010000 [4 parity bits + 8 data bits].

Problem:- A 7-bit even parity Hamming code is 0100011. Determine the error detection and correct the data.

Sol:- Given Hamming code is 0100011.

① find number of parity checking bits using expression

$$2^P \geq P+m+1 \qquad \text{Given } P+m=7 \text{ bits}$$

Let $P=2$ $\quad 2^2 \geq 7+1$

$4 \geq 8$ ✗

Let $P=3$ $\quad 2^3 \geq 7+1$

$8 \geq 8$ ✓ satisfied.

Number of parity bits in given hamming code are 3.

② Parity checking bits locations are P_1, P_2 and P_4.

then message bits " " $m_3\ m_5\ m_6\ m_7$

Given Hamming code = $m_7 m_6 m_5 P_4 m_3 P_2 P_1$ is

0 1 0 0 0 1 1

→ from the table error positions are

P_1 (even parity locations 1,3,5,7

P_2 (even parity locations 2,3,6,7

P_4 (even parity 4,5,6,7

	P_4 P_2 P_1
0	0 0 0
1	0 0 1
2	0 1 0
3	0 1 1
4	1 0 0
5	1 0 1
6	1 1 0
7	1 1 1

→ Draw the table

Bit locations	7	6	5	4	3	2	1	
Designation	D_7	D_6	D_5	D_4	D_3	D_2	D_1	
Hamming code	0	1	0	0	0	1	1	
message bits	m_7	m_6	m_5		m_3			
Parity bits				P_4		P_2	P_1	
P_1 even parity 1,3,5,7	0		0		0		1	→ error present
P_2 even parity 2,3,6,7	0	1			0	1		→ no error
P_4 even parity 4,5,6,7	0	1	0	0				→ error present

→ from the observation [common bit location between P_1 and P_4 checking bits is] error detected at location 5.

→ So, change the m_5 bit with correct information.

then m_5 becomes 1.

→ Then correct Hamming code is 0 1 1 0 0 1 1.

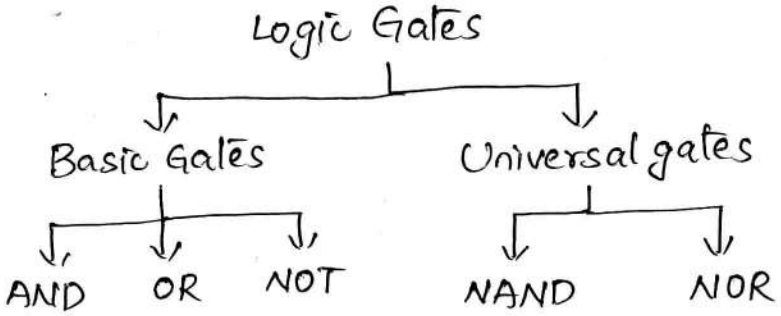
Data is 0 1 1 0.

PART-B

BOOLEAN ALGEBRA AND LOGIC GATES

* Logic Gates:-

→ Logic gates are fundamental building blocks of digital systems.
→ Logic gates are made up of a number of electronic devices and components.
→ Logic gate is a device implementing a boolean function, that is, it performs a logical operation on one or more logical inputs and produces a single logical output.
→ Logic gates are classified into two types. They are



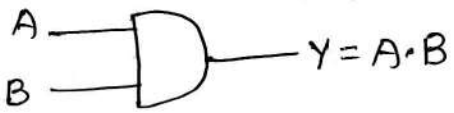
* Basic Gates:-

→ Basic gates are . AND, OR and NOT gates.

① AND gate:-

→ An AND gate has two or more inputs but only one output.
→ 'AND' is denoted by $\cdot$, $\wedge$, $\cap$

LOGIC SYMBOL



TRUTH TABLE

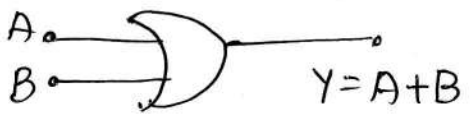
Inputs A	Inputs B	Output Y
0	0	0
0	1	0
1	0	0
1	1	1

Logical operation:-

→ The output becomes logic 1 state, only when each one of its inputs is at logic 1 state.
→ The output becomes logic 0 even if one of its inputs is at logic 0.
→ AND gate is also called an 'all' or 'nothing' gate.
→ If we consider two input variables A, B and output variable Y then the Boolean expression for the output can be written as $\boxed{Y = A \cdot B}$ (or) Y = A and B.
→ If we consider more than 2 input variables we can represent $Y = A \cdot B \cdot C \cdot \cdots$ Y = A and B and C ...

② OR Gate :-

→ An OR gate have two or more inputs but only one output.
→ OR operation denoted by '+'.
→ LOGIC SYMBOL



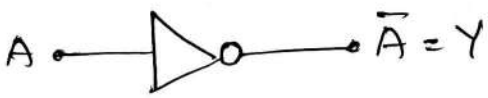
TRUTH TABLE

Inputs A	Inputs B	Output Y
0	0	0
0	1	1
1	0	1
1	1	1

Logical Operation :-
→ The output becomes logic 1 state, even if one of its inputs is in logic 1 state.
→ Its output becomes logic 0 state, only when each one of its inputs is in logic 0 state.
→ OR gate is also called an 'any' or 'all' gate.
→ With the input variables to the OR gate are A, B & output Y then the Boolean expression for the output can be written as $Y = A+B$; Y = A or B
→ If we consider more than 2 inputs then $Y = A+B+C+\cdots$; Y = A or B or C ...

③ NOT gate :-

→ NOT gate has only one input and single output.
→ NOT operation denoted by '–' (bar)
LOGIC SYMBOL



TRUTH TABLE

Input A	output Y
0	1
1	0

Logical Operation :-
→ It is a device whose output is always the complement of its input i.e., the output of a NOT gate becomes the logic '1' state when its input is in logic '0' and becomes logic '0' when its input is in logic '1'.
→ NOT gate also called an inverter.
→ When the input variable to the NOT gate is represented by A and the output variable by Y, the Boolean expression for the output is $Y = \bar{A}$; Y = not A.
→ Logic circuits which use these three logic gates only are called AOI logic circuits. [AND-OR-INVERT].

Universal Gates :-

→ If logic circuits have any complexity with the use of three basic gates, there are two universal gates.
→ Universal gates are NAND and NOR gates.
→ Both NAND and NOR gates can perform all the three basic logic functions (AND, OR & NOT). Therefore NAND & NOR gates are called universal gates.

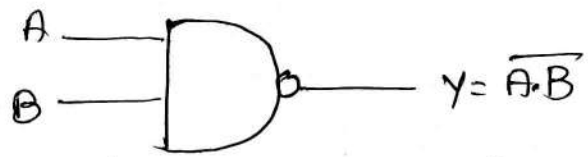
① NAND gate :-

→ NAND means NOT AND i.e., AND output is NOTed.
→ NAND gate is a combination of AND gate and a NOT gate.

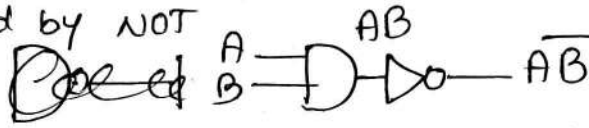
Truth Table

Inputs A	Inputs B	outputs Y
0	0	1
0	1	1
1	0	1
1	1	0

Logic Symbol



(or) AND followed by NOT



Logical Operation :-

→ The output is logic 0 level, only when each of the inputs assumes a logic 1 level. For any other combination of inputs, the output is a logic 1 level.
→ If A, B are input variables, Y is output variable, then the Boolean expression for NAND operation is represented by $\boxed{Y = \overline{A \cdot B}}$ $Y = A$ nand B.
→ If we consider more than 2 inputs

$$Y = \overline{ABCD\ldots}$$

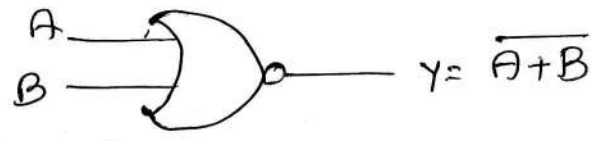
② NOR gate :-

→ NOR means NOT OR i.e., OR output is NOTed.
→ NOR gate is a combination of OR gate and a NOT gate

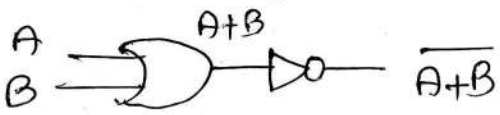
Truth Table

Inputs A	Inputs B	output Y
0	0	1
0	1	0
1	0	0
1	1	0

logic Symbol



OR followed by NOT



Logical operation:-

→ The output is logic 1 level, only when each one of its inputs becomes a logic 0 level. for any other combination of inputs, the output is a logic 0 level.

→ If A, B are 2 input variables & Y is output variable, then the Boolean expression for NOR logical operation is denoted by $\boxed{Y = \overline{A+B}}$ $Y = A \text{ nor } B$.

→ If we consider more than 2 inputs then

$$Y = \overline{A+B+C+\cdots}$$

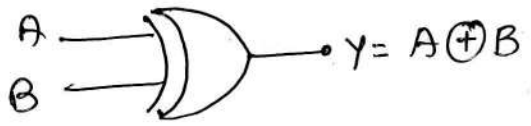
→ Here we have other two logic gates. They are EX-OR gate and EX-NOR gate.

Ex-OR gate:- (Exclusive-OR gate)

→ An Ex-OR gate is a two input and one output logic circuit.

→ The output assumes a logic 1 state when one and only one of its two inputs becomes logic 1. when both the inputs are at logic 0 or 1 then output becomes 0.

Logic symbol



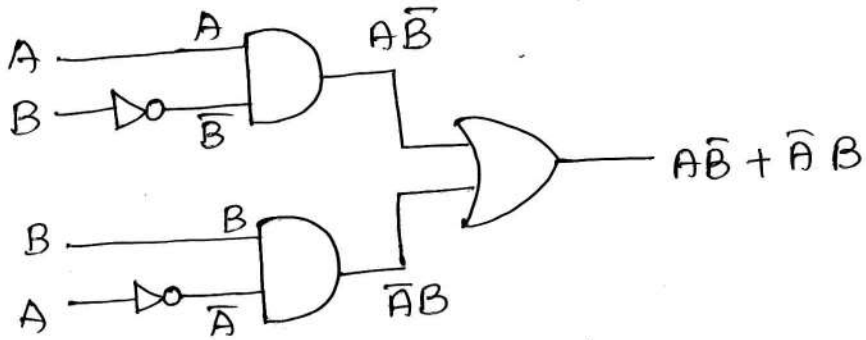
Truth Table

Inputs		output
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

→ If A, B are input variables and Y is an output variable, then the Boolean expression for Ex-OR operation is represented by $\boxed{Y = A \oplus B}$

Logic diagram for Ex-OR gate:-

$$A \oplus B = A\bar{B} + \bar{A}B.$$



→ To construct Ex-OR gate we require 2 NOT gates, 2 AND gates & 1 OR gate.

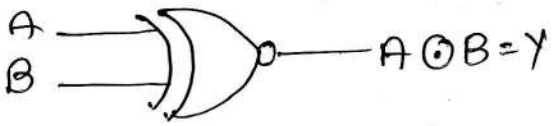
Ex-NOR Gate :-

→ An EX-NOR gate is a combination of an x-OR gate and a NOT gate.

→ The X-NOR gate is a two input, one output logic circuit, whose output becomes logic 1 only when both the inputs are 0 or when both the inputs are 1.

→ The output becomes logic 0 when one of the inputs becomes 0 & the other 1.

Logic Symbol



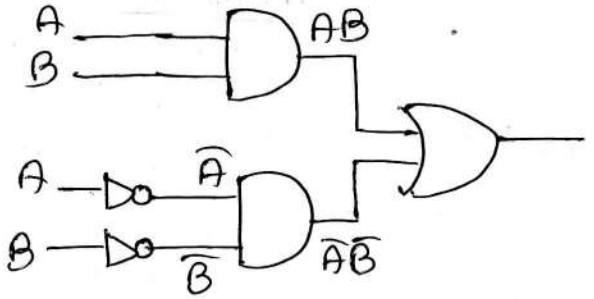
Truth Table

Inputs		output
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

→ If A, B are two input variables, Y is output variable, then the boolean expression for Ex-nor operation is denoted by $Y = A \odot B$

Logic diagram for Ex-NOR gate :-

$$Y = A \odot B = \overline{A \oplus B} = A\bar{B} + \bar{A}B = ? \; AB + \bar{A}\bar{B}$$



Here also we require 2 AND, 2 NOT & 1 OR gate.

* Properties of Boolean Algebra :- [Duality]

* If we observe Huntington postulates which consists two parts (a) & (b). One part may be obtained from the other if the binary operators and the identity elements are interchanged.
* This property of Boolean algebra is called the duality principle.

Duality principle :- It states that every algebraic expression deducible from the postulates of boolean algebra remains valid if the operators & identity elements are interchanged.

* In a two valued Boolean algebra, the identity elements and the elements of the set B are the same: 1 & 0.
* If we observe above theorem's (1) b is the dual of (1) a. Any dual theorem can be similarly derived from the proof of its corresponding pair.

* Boolean functions :-

→ A Boolean function described by an algebraic Expression consists of binary variables and some logic operation symbols.

→ For a given value of the binary variables, the function can be equal to either 1 or 0.

→ Ex :- $f_1 = x + y'z$.

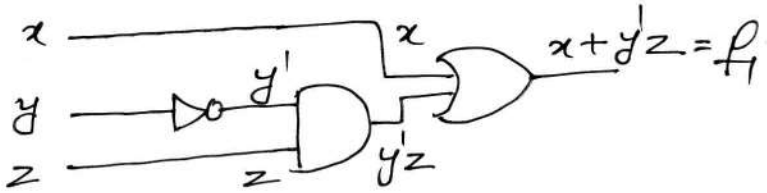
Function f_1 consists 3 variables, and 2 terms.

→ we can find f_1 values based on truth table.

x	y	z	y'	y'z	x+y'z
0	0	0	1	0	0
0	0	1	1	1	1
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	1	0	1
1	0	1	1	1	1
1	1	0	0	0	1
1	1	1	0	0	1

→ The function f_1 is equal to 1 if x is equal to 1 or if both y' & z are equal to 1

→ we can implement function by using logic gates. To implement f_1 we require 1 OR, 1 AND and 1 NOT gates.



→ We define a literal to be a single variable within a term that may be complemented or not.
→ In above example f_1 consists 3 literals.

Simplification of Boolean function:-

→ We can simplify a Boolean function by applying some of the postulates of Boolean algebra.

Ex:- $f_2 = x(x'+y) = xx'+xy \quad (xx'=0)$
$= 0+xy = xy.$

→ Normally f_2 consists 3 literals. By simplification reduced to 2 literals.

Ex:- $f_3 = x'y'z + x'yz + xy'$
$= x'z(y+y') + xy' = x'z \cdot 1 + xy' = x'z + xy'.$

Ex:- $(x+y)(x+y') = x \cdot x + x \cdot y' + y \cdot x + y \cdot y'$
$= x + xy' + xy + 0 = x + xy' + xy = x[1+y+y']$
$= x[1+0] = x$

Complement of a function:-

→ The complement of a function is obtained from an interchange of 0s for 1s & 1's for 0's in the value of F.
→ Algebraically we can find complement of a function by using Demorgan's theorem.

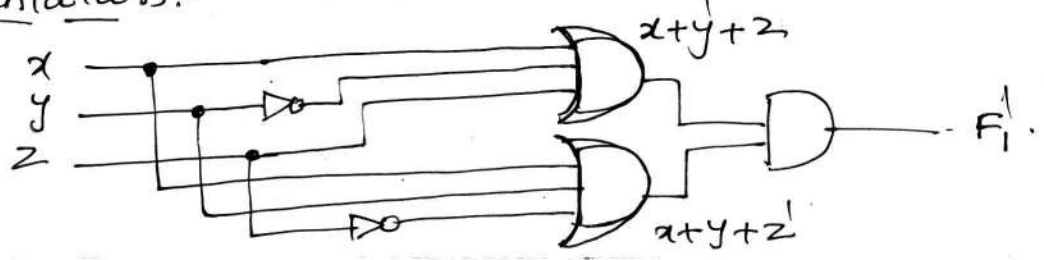
Ex:- $F = (A+B+C')'$ find complement of F.

$F = (A+B+C')$
$F' = (A+B+C')'$ $\{(x+y)' = x' \cdot y'\}$
$= A' \cdot B' \cdot (C')$ $\{(x')' = x\}$
$= A'B'C$

Ex:- $F_1 = x'yz' + x'y'z$ find complement of F_1

Given $F_1 = x'yz' + x'y'z$
$F_1' = [x'yz' + x'y'z]'$ $\{(x+y)' = x'y'\}$
$= (x'yz')'(x'y'z)'$ $\{(xy)' = x'+y'\}$
$= [(x')' + y' + (z')'][(x')' + (y')' + z']$
$= (x+y'+z)(x+y+z')$

implementation:-



* Canonical and Standard forms :-

→ Minterms and Maxterms :-

Generally a binary variable may appear in its normal form (x) or in its complement form (x').

→ Now consider two binary variables x & y combined with an AND operation. Here each variable may appear in either form, There are four possible combinations

$x'y'$ — 00 ⇒ These combinations are called minterms or standard products.

$x'y$ — 01

xy' — 10 ⇒ In minterm normal form (x) represented by 1, complement form (x') represented by 0.

xy — 11

→ If we consider n variables, they can be combined to form 2^n minterms.

→ Similarly, n variables forming an OR term, we will get 2^n maxterms.

Ex :- x & y 2 variables we will get 4 maxterms

$x+y$ — 00 → In maxterm normal form (x) represented by 0

$x+y'$ — 01 Complement form (x') represented by 1.

$x'+y$ — 10 → maxterms also called standard sums.

$x'+y'$ — 11

Minterms and maxterms for Three Binary variables

			Minterms		Maxterms		
x	y	z	Term	Designation	Term	Designation	
0	0	0	$x'y'z'$	m_0	$x+y+z$	M_0	Here maxterm is Complement of corresponding minterm.
0	0	1	$x'y'z$	m_1	$x+y+z'$	M_1	
0	1	0	$x'yz'$	m_2	$x+y'+z$	M_2	
0	1	1	$x'yz$	m_3	$x+y'+z'$	M_3	
1	0	0	$xy'z'$	m_4	$x'+y+z$	M_4	
1	0	1	$xy'z$	m_5	$x'+y+z'$	M_5	
1	1	0	xyz'	m_6	$x'+y'+z$	M_6	
1	1	1	xyz	m_7	$x'+y'+z'$	M_7	

* Canonical forms :-

(1) Canonical SOP Sum of Products (or) Sum of minterms

(2) Canonical POS ~~sum~~ product of sums (or) Product of maxterms.

(1) Canonical Sum of Minterms:-

→ We can express a boolean function as a sum of minterms

→ Now consider a truth table, we can form a minterm for each combination of the variables that produces a 1 in the function, and then take the OR of all those terms.

Ex:-

x	y	z	f_1	f_2
0	0	0	0	1
0	0	1	1	1
0	1	0	0	1
0	1	1	0	1
1	0	0	1	0
1	0	1	1	0
1	1	0	0	0
1	1	1	0	0

$f_1 = 1$ for combinations 001, 100, 101

⇒ minterms are,

001 — $x'y'z$

100 — $xy'z'$

101 — $xy'z$

add all these 3 minterms

⇒ $f_1 = x'y'z + xy'z' + xy'z$.

Here 'f_1' is a Boolean function in form of SOP.

$f_1 = x'y'z + xy'z' + xy'z$.

$\underbrace{001}_{1}$ $\underbrace{100}_{4}$ $\underbrace{101}_{5}$

we can also represent it like

$f_1 = m_1 + m_4 + m_5$ (or) $f_1(x,y,z) = \Sigma(1,4,5)$.

IIIly $f_2 = 1$ combinations 000, 001, 010, 011 are

$f_2 = x'y'z' + x'y'z + x'yz' + x'yz$

$= m_0 + m_1 + m_2 + m_3 = \Sigma(0,1,2,3)$

(2) Canonical Product of Maxterm:-

→ we can express a boolean function as product of maxterms.

→ Consider a truth table, form a maxterm for each combination of the variables that produces a 0 in the function, then take the AND of all those terms.

→ In above example

$f_1 = 0$ for combinations 000, 010, 011, 110, 111

maxterms are $x+y+z$, $x+y'+z$, $x+y'+z'$, $x'+y'+z$, $x'+y'+z'$.

product all these 5 maxterms.

⇒ $f_1 = (x+y+z)(x+y'+z)(x+y'+z')(x'+y'+z)(x'+y'+z')$

$\underbrace{000}_{0}$ $\underbrace{010}_{2}$ $\underbrace{011}_{3}$ $\underbrace{110}_{6}$ $\underbrace{111}_{7}$

$f_1 = M_0 M_2 M_3 M_6 M_7$ (or) $f_1(x,y,z) = \Pi(0,2,3,6,7)$.

IIIly $f_2 = 0$ combinations 100, 101, 110, 111

⇒ $f_2 = (x'+y+z)(x'+y+z')(x'+y'+z)(x'+y'+z')$

$= M_4 M_5 M_6 M_7 = \Pi(4,5,6,7)$

Problem ① :- Express $F = xy + x'z$ in a sum of minterms.

Sol :- Given $F = xy + x'z$

In SOP form each minterm must consists 3 variables either in normal form or in complement form.

$$F = xy \cdot 1 + x'z \cdot 1$$
$$= xy[z+z'] + x'z[y+y']$$
$$= xyz + xyz' + x'yz + x'y'z \quad \text{in SOP form.}$$

Problem ② :- Express $F = AB + B'C$ in a product of maxterms.

Sol :- Given $F = AB + B'C$

In POS form each maxterm must consists 3 variables either in normal form or in complement form.

$$F = AB + B'C$$
$$\{a + bc = (a+b)(a+c)\}$$
$$= (AB + B')(AB + C)$$
$$= (B' + AB)(C + AB)$$
$$= (B'+A)(B'+B)(A+C)(B+C)$$
$$= (A+B')(A+C)(B+C)$$
$$= (A+B'+0)(A+C+0)(B+C+0)$$
$$= (A+B'+CC')(A+C+BB')(B+C+AA')$$
$$= (A+B'+C)(A+B'+C')(A+B+C)(A+B'+C)(A+B+C)(A'+B+C)$$
$$= (A+B'+C)(A+B'+C')(A+B+C)(A'+B+C)$$

Problem ③ :- Calculate min and maxterms of $F = AB + \bar{B}C + C\bar{A}$

Sol :- Given $F = AB + \bar{B}C + C\bar{A}$

$$= AB \cdot 1 + \bar{B}C \cdot 1 + C\bar{A} \cdot 1$$
$$= AB \cdot (C + \bar{C}) + \bar{B}C(A + \bar{A}) + \bar{A}C(B + \bar{B})$$
$$= ABC + AB\bar{C} + A\bar{B}C + \bar{A}\bar{B}C + \bar{A}BC + \bar{A}\bar{B}C$$
$$= ABC + AB\bar{C} + A\bar{B}C + \bar{A}\bar{B}C + \bar{A}BC$$
$$\underbrace{111}_{7} \quad \underbrace{110}_{6} \quad \underbrace{101}_{5} \quad \underbrace{001}_{1} \quad \underbrace{011}_{3}$$

$$F = m_1 + m_3 + m_5 + m_6 + m_7 = \Sigma(1,3,5,6,7)$$

Minterms are 1, 3, 5, 6 & 7.

$$F' = \Sigma(0,2,4) = m_0 + m_2 + m_4$$

$$F = (F')' = (m_0 + m_2 + m_4)' = m_0' \cdot m_2' \cdot m_4' = M_0 M_2 M_4$$

maxterms are 0, 2 & 4.

* Conversion between Canonical forms:-

→ The complement of a function is expressed as the SOP equal the SOP of missing terms from the original function.

Ex:- $F(A,B,C) = \Sigma(1,4,5,6,7)$.

The complement of F is

$F'(A,B,C) = \Sigma(0,2,3) = m_0 + m_2 + m_3$.

Take complement of F', then we will get F in different form.

$(F'(A,B,C))' = F(A,B,C) = (m_0 + m_2 + m_3)'$

$= m_0' \cdot m_2' \cdot m_3' = M_0 M_2 M_3$.

$F(A,B,C) = \Pi(0,2,3)$

Prob:- Convert $F(A,B,C,D) = \Sigma(1,4,5,8,9,11)$ into POS.

$F(A,B,C,D) = \Sigma(1,4,5,8,9,11)$

$F'(A,B,C,D) = \Sigma(0,2,3,6,7,10,12,13,14,15)$

$(F'(A,B,C,D))' = [m_0 + m_2 + m_3 + m_6 + m_7 + m_{10} + m_{12} + m_{13} + m_{14} + m_{15}]'$

$= M_0 M_2 M_3 M_6 M_7 M_{10} M_{12} M_{13} M_{14} M_{15}$

$= \Pi(0,2,3,6,7,10,12,13,14,15)$.

Prob:- Convert $F(A,B,C) = \Pi(1,2,3)$ into SOP.

$F(A,B,C) = \Pi(1,2,3)$

$F'(A,B,C) = \Pi(0,4,5,6,7) = M_0 M_4 M_5 M_6 M_7$

$F(A,B,C) = [M_0 M_4 M_5 M_6 M_7]'$

$= m_0 + m_4 + m_5 + m_6 + m_7$.

Standard form:-

→ The two canonical forms of Boolean algebra are basic forms that one obtains from reading a function from the truth table.

→ In these two canonical forms contains more number of literals because each minterm or maxterm must contain all the variables either in normal or complement form.

→ Another way to express Boolean function is in standard form.

→ There are two types of standard forms

- standard SOP
- standard POS.

This function may contain one, two or any number of literals.

UNIT-II

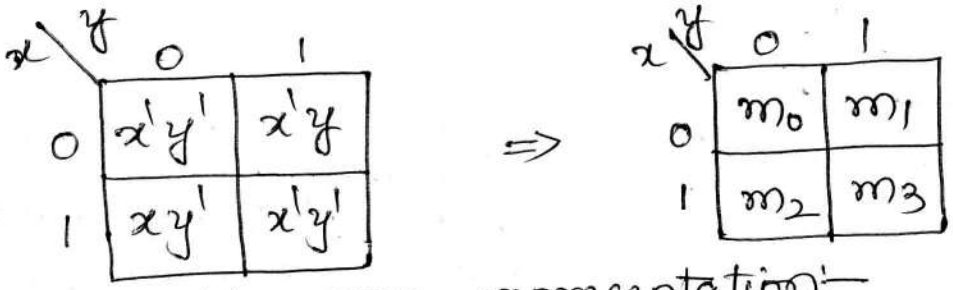
Gate Level Minimization

* The Map Method :- (Vietech Diagram)

→ The map method provides a simple straight-forward procedure for minimizing Boolean functions.
→ The map method is also known as the Karnaugh map or K-map.
→ The map is a diagram made up of squares / cells, with each square representing minterm of the function.
→ Any Boolean function can be expressed as sum of minterms.
→ The simplified expression produced by the map are always in one of the two standard forms: SOP or POS.
→ It contains minimum number of literals as well as with least terms.

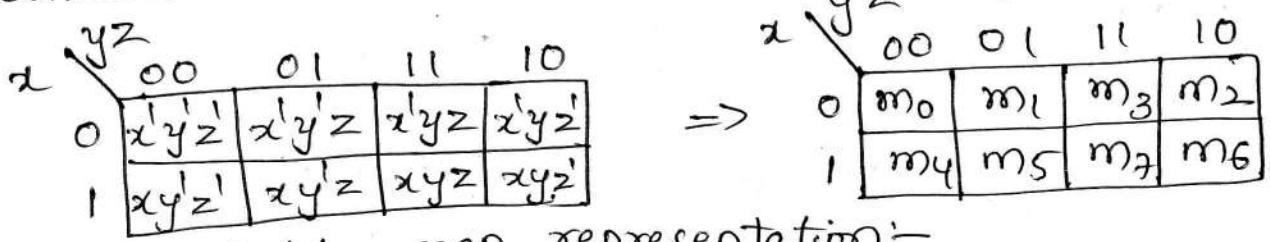
* 2-Variable map representation :-

→ There are 4 minterms for 2 variables x, y. It contains 4 cells. ($2^2 = 4$). In minterm $x = 1$; $x' = 0$.



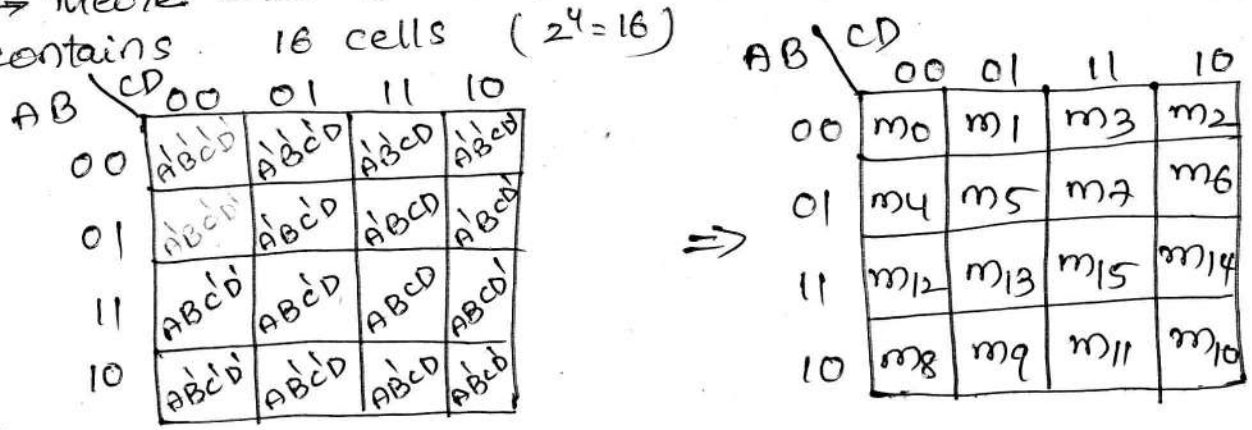
* 3-Variable map representation :-

→ There are 8 minterms for 3 variables x, y & z. It contains 8 cells. ($2^3 = 8$).



* 4-Variable map representation :-

→ There are 16 minterms for 4 variables A, B, C & D. It contains 16 cells ($2^4 = 16$)



* Grouping of cells for Simplification:-

* Plotting of a truth table on the K-map:-

Given truth table we can represent 'f' values in K-map.

A	B	f
0	0	0
0	1	0
1	0	1
1	1	1

Each cell represents minterms of corresponding variables.

consider 2V K-map.

A \ B	0	1
0	0 (0)	0 (1)
1	1 (2)	1 (3)

* Plotting of a canonical SOP form on K-map:-

→ A Boolean function in SOP form can be plotted on a K-map by placing 1 in each cell corresponding to the minterm in the canonical SOP expression.

Ex:- $f = \bar{A}\bar{B}C + \bar{A}BC + AB\bar{C} = f = \Sigma(1,3,5)$

001 011 110

→ f has '3' variables and it can be plotted using 3-variable K-map.

A \ BC	00	01	11	10
0	0 (0)	1 (1)	1 (3)	0 (2)
1	0 (4)	0 (5)	0 (7)	1 (6)

other cells are represented by 0's.

* plotting of a canonical POS form on K-map:-

→ A Boolean function in POS form can be plotted on a K-map by placing 0 in each cell corresponding to the maxterm in the canonical POS expression.

Ex:- $f = (A+B+C)(A+\bar{B}+C)(\bar{A}+B+\bar{C})(A+B+\bar{C})$.

000 010 101 001

$f = \Pi(0,1,2,5)$.

→ f has 3 variables and it can be plotted using 3-variable K-map.

A \ BC	00	01	11	10
0	0 (0)	0 (1)	1 (3)	0 (2)
1	1 (4)	0 (5)	1 (7)	1 (6)

other cells are represented by 1's.

* Grouping of cells for ~~simplifi~~ simplification :-

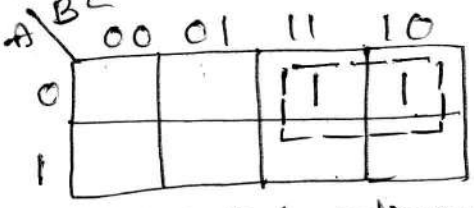
→ Once the logic function is plotted on the K-map, we have to use grouping technique to simplify the logic function

→ Grouping means combining the adjacent logic cells which contains 1's or 0's.

→ In SOP we can group adjacent 1's ; in POS we can group adjacent 0's.

* Grouping of 2 adjacent cells (pairs):-

(a)



(Two horizontal adjacent cells)

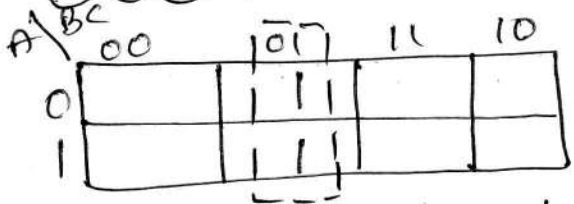
Then simplified function becomes

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

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

$= \bar{A}B$

(b)



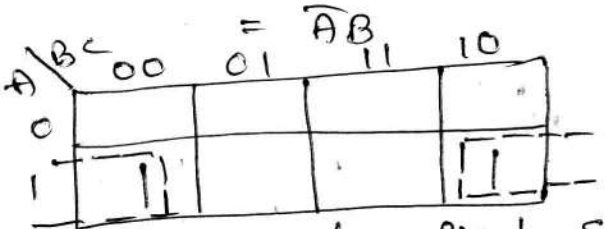
(Two vertical adjacent cells)

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

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

$= \bar{B}C$

(c)

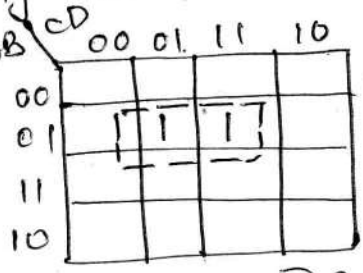


$\Rightarrow f = A\bar{B}\bar{C} + AB\bar{C} = A\bar{C}(B+\bar{B})$

$= A\bar{C}.$

Any two cells of first & last columns and belong to same row.

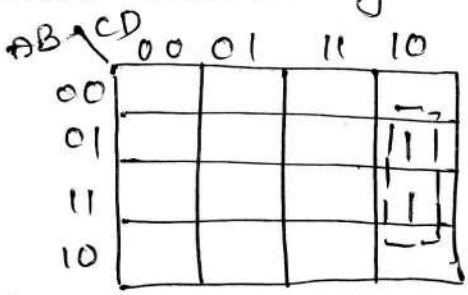
(d)



(2 horizontal adjacent cells)

$f = \bar{A}B\bar{C}D + \bar{A}BCD = \bar{A}BD$

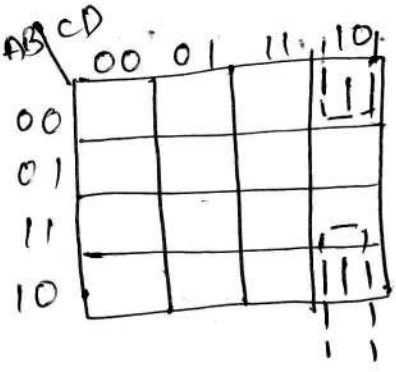
(e)



(Two vertical adjacent cells)

$f = \bar{A}BC\bar{D} + ABC\bar{D} = BC\bar{D}$

(f)

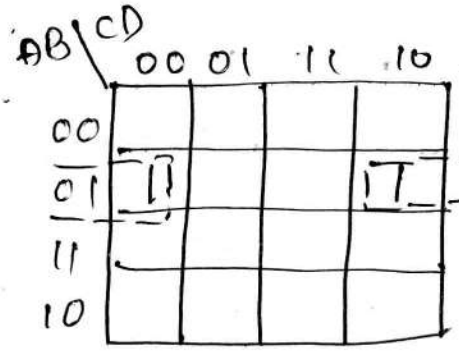


Any 2 cells of first & last row & belong to same column

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

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

(g)



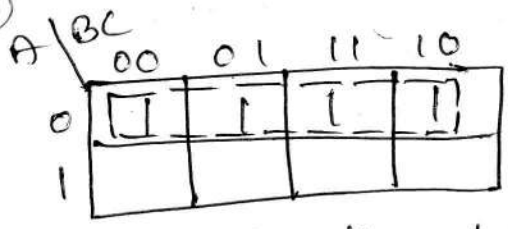
Any 2 cells of 1st & last column & belong to same row

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

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

* Grouping 4 adjacent cells :- (quad)

(a)

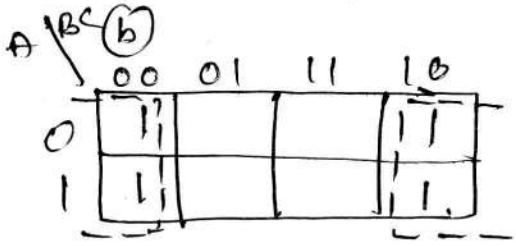


4 horizontal adjacent cells

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

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

(b)

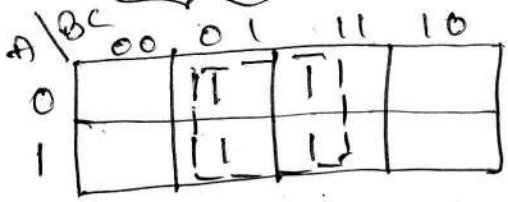


Two first & last column pairs.

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

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

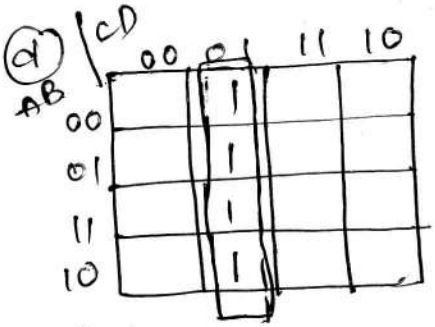
(c)



Any two adjacent pairs.

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

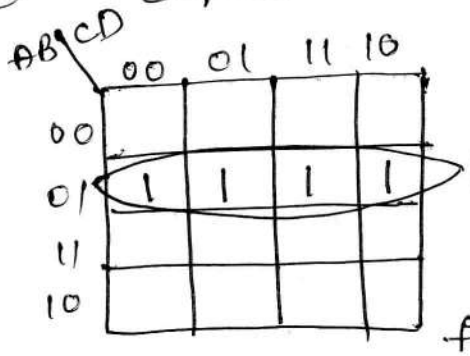
(d)



Any vertical column form a quad

$f = \bar{C}D$

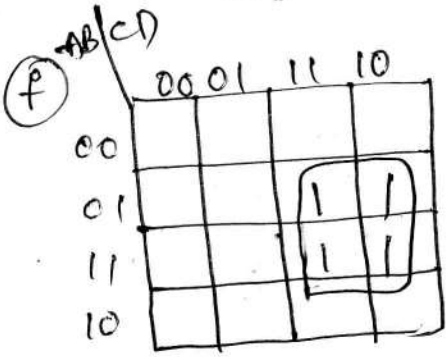
(e)



Any horizontal row form a quad

$f = \bar{A}B$

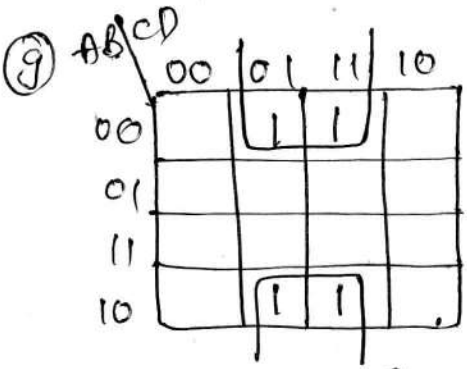
(f)



Any two adjacent pairs

$f = BC$

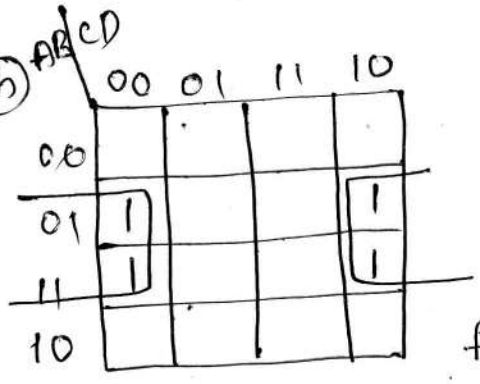
(g)



Any 2 adjacent pairs

$f = \bar{B}D$

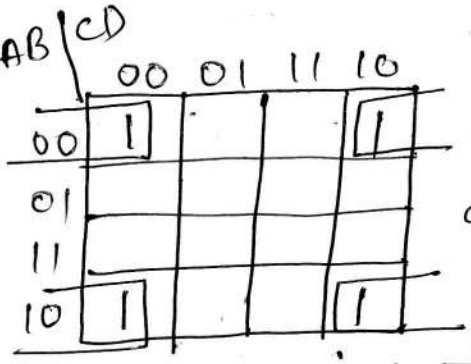
(h)



Any two adjacent pairs

$f = B\bar{D}$

(i)

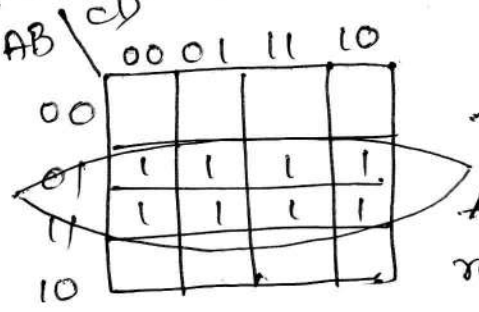


Any 2 adjacent pairs

$f = \bar{B}\bar{D}$

* Grouping of 8 adjacent cells (octet):-

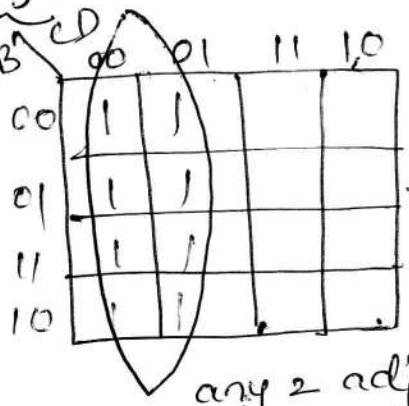
(a)



$f = B$

Any 2 adjacent rows form octet

(b)



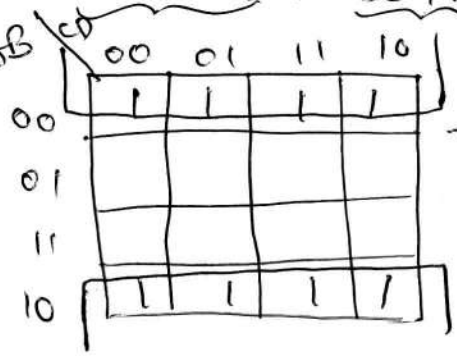
$f = \bar{C}$

any 2 adjacent columns form octet

$f = \bar{A}B\bar{C}\bar{D} + \bar{A}B\bar{C}D + \bar{A}BCD + \bar{A}BC\bar{D} + AB\bar{C}\bar{D} + AB\bar{C}D + ABCD + ABC\bar{D}$

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

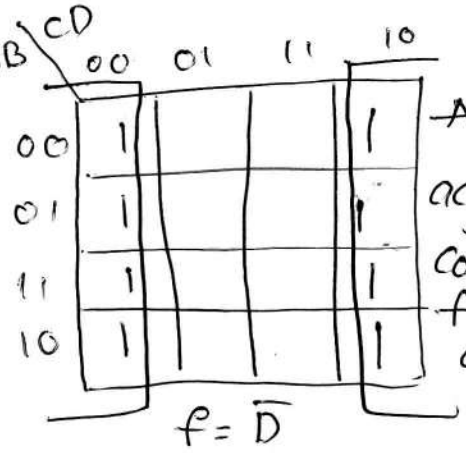
(c)



Any 2 adjacent rows form octet

$f = \bar{B}$

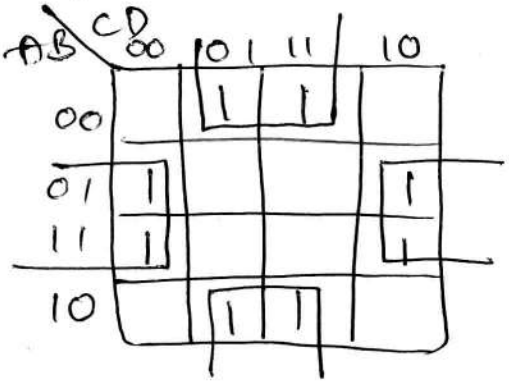
(d)



Any 2 adjacent columns form octet

$f = \bar{D}$

(e)



4 adjacent pairs form octet

$f = \bar{A}\bar{B}\bar{C}D + \bar{A}\bar{B}CD + \bar{A}B\bar{C}\bar{D} + AB\bar{C}\bar{D} + \bar{A}BCD + ABCD + AB\bar{C}D + AB\bar{C}D$

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

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

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

Simplification of SOP Expressions:-

Prob:- Simplify the following expression by using K-map & implement by using logic gates.

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

Sol:- Step(i):- Given function is in standard SOP form. To simplify any function by using K-map, function must be in canonical form. Convert this standard SOP into canonical SOP.

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

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

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

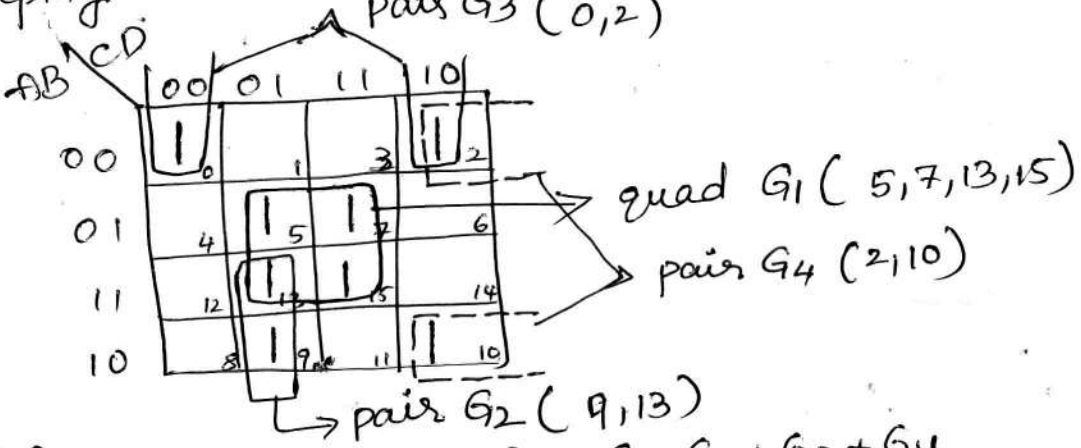
$f = A\bar{B}\bar{C}D + \bar{A}\bar{B}\bar{C}\bar{D} + AB\bar{C}D + \bar{A}BCD + A\bar{B}C\bar{D} + \bar{A}\bar{B}C\bar{D} + \bar{A}B\bar{C}D +$
$\underbrace{1001}_{9}\ \underbrace{0000}_{0}\ \underbrace{1101}_{13}\ \underbrace{0111}_{7}\ \underbrace{1010}_{10}\ \underbrace{0010}_{2}\ \underbrace{0101}_{5}\ \underbrace{1111}_{15}$ ABCD

$f = \Sigma(0,2,5,7,9,10,13,15) = m_0 + m_2 + m_5 + m_7 + m_9 + m_{10} + m_{13} + m_{15}$

step(2):- Given function is a 4-variable function. To simplify this function use 4-variable K-map.

step(3):- Represent minterms marked by 1's in corresponding cells.

step(4):- Grouping of cells for simplification.



step(5):- Simplified expression $f = G_1 + G_2 + G_3 + G_4$

$G_1 = \bar{A}B\bar{C}D + \bar{A}BCD + AB\bar{C}D + ABCD$

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

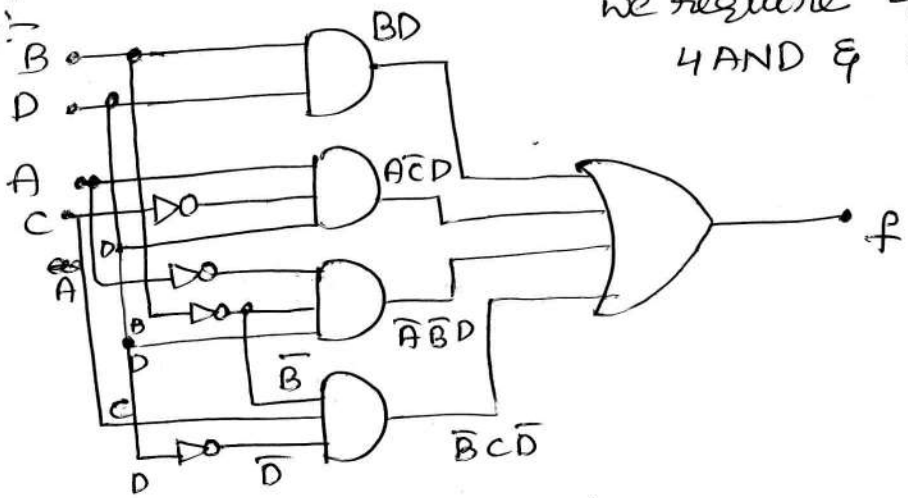
$G_2 = AB\bar{C}D + A\bar{B}\bar{C}D = (A\bar{C}D)$

$G_3 = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}C\bar{D} = (\bar{A}\bar{B}D)$

$G_4 = \bar{A}BC\bar{D} + A\bar{B}C\bar{D} = (\bar{B}C\bar{D})$

Simplified expression $f = BD + A\bar{C}D + \bar{A}\bar{B}D + \bar{B}C\bar{D}$.

Implimentation:-



we require 4 NOT, 4 AND & ONE OR gates.

Simplification of POS expressions:-

Prob:- Minimize the following expression in POS form. implement using gates

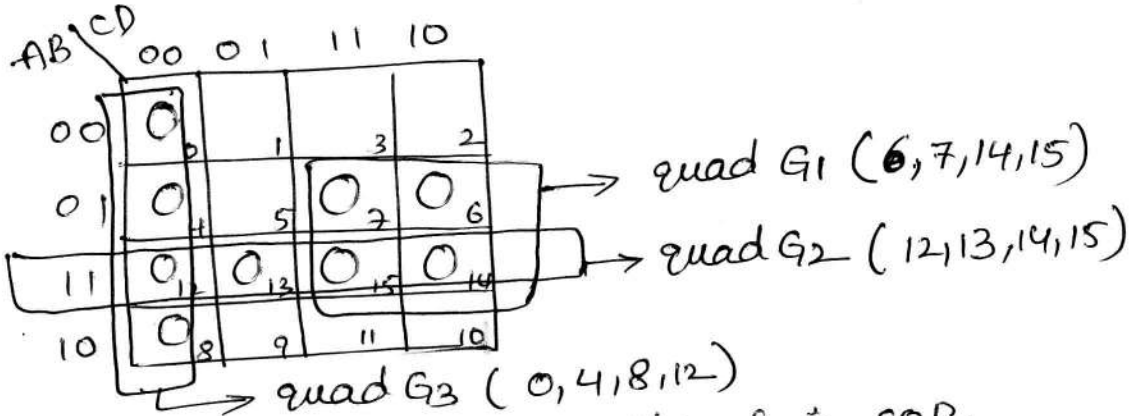
$f = \Pi M(0,4,6,7,8,12,13,14,15)$.

Sol:- Given function $f = \Pi M(0,4,6,7,8,12,13,14,15)$. (4)

step (1):- Given function is a 4-variable function. So, plot 4-variable K-map.

step (2):- Given function is in pos form. So, maxterms are marked by 0's in the corresponding cells.

step (3):- form possible pair, octet & quads for corresponding cells.



step (4):- All 0's are covered. first write f in SOP.

f in SOP = Get SOP of G_1 + SOP of G_2 + SOP of G_3

SOP of $G_1 = \bar{A}BCD + \bar{A}BC\bar{D} + ABCD + ABC\bar{D}$

$= \bar{A}BC + ABC = (BC)$

SOP of $G_2 = AB\bar{C}\bar{D} + AB\bar{C}D + ABCD + ABC\bar{D}$

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

SOP of $G_3 = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}B\bar{C}\bar{D} + AB\bar{C}\bar{D} + A\bar{B}\bar{C}\bar{D}$

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

f in SOP = $BC + AB + \bar{C}\bar{D}$.

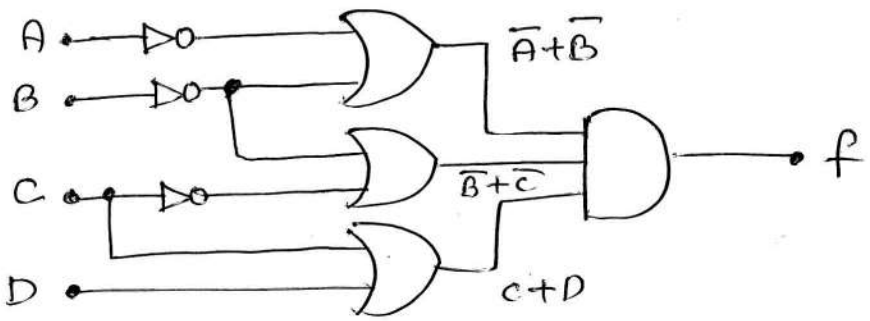
step (5):- find the complement of F in SOP, we will f in POS. Then we will get simplified f.

f in POS = $BC + AB + \bar{C}\bar{D}$ $\{(x+y+z)' = x'\cdot y'\cdot z'\}$

$= \overline{BC} \cdot \overline{AB} \cdot \overline{\bar{C}\bar{D}}$ $\{(xyz)' = x'+y'+z'\}$

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

Implementation:- We require 3 OR; 1 AND & 3 NOT gates



* Don't Care Conditions :-

→ In some logic circuits there are certain input conditions for which there are no specified output levels, because these input conditions will never occur.
→ These combinations of input levels are called don't care conditions. [means output is either 0 or 1].
→ ~~Don't~~ these output levels are indicated by 'x' or 'd' or 'ϕ'
→ An incompletely specified function containing k don't care combinations for 2^k distinct functions.

Ex:- for 8 combinations we can specify 3 don't care combinations.

Prob:- Find the reduced SOP for the following functions.
$f(A,B,C,D) = \Sigma m(4,6,7,13,14) + \Sigma d(5,10,12,15)$.

Sol:- Given function $f(A,B,C,D) = \Sigma(4,6,7,13,14) + \Sigma d(5,10,12,15)$.

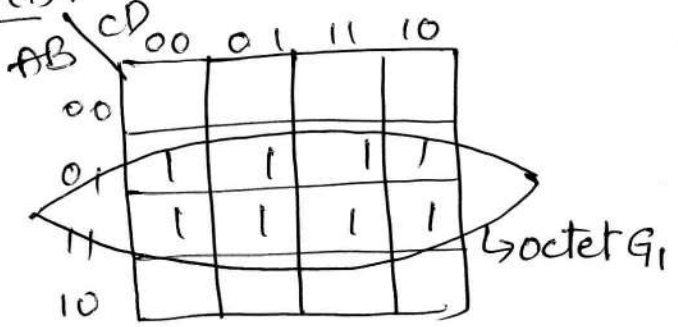
Step(1):- Plot 4-variable K-map to implement given function

Step(2):- minterms are marked by 1's and don't cares are marked by 'x'.

Step(3):- Grouping of cells.
→ To form a quad, the don't care in cell 5 is treated as 1, cell 12 is treated as 1 & cell 15 treated as 1.
→ The remaining cell 10 treated as 0.

AB \ CD	00	01	11	10
00	0	1	3	2
01	1 (4)	X (5)	1 (7)	1 (6)
11	X (12)	1 (13)	X (15)	1 (14)
10	8	9	11	X (10)

Step(4):- Then modified K-map becomes



Simplified expression becomes

$f = B$.

Step(5):- Simplified SOP becomes

$f = G_1$

$G_1 = \bar{A}B\bar{C}\bar{D} + \bar{A}B\bar{C}D + \bar{A}BCD + \bar{A}BC\bar{D} + AB\bar{C}\bar{D} + AB\bar{C}D + ABCD + ABC\bar{D}$

$G_1 = \bar{A}B\bar{C} + \bar{A}BC + AB\bar{C} + ABC$

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

* 5-Variable K-Map Representation :-

→ The 5-variable K-map contains $2^5 = 32$ cells.
→ To represent 32-cell map we require two 16-cell K-maps.
→ If the variables are A, B, C, D & E then the two 16-cell K-maps

contain B, C, D & E variables with A=1 and A=0.

→ Representation:

BC \ DE	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

(A=0)

BC \ DE	00	01	11	10
00	16	17	19	18
01	20	21	23	22
11	28	29	31	30
10	24	25	27	26

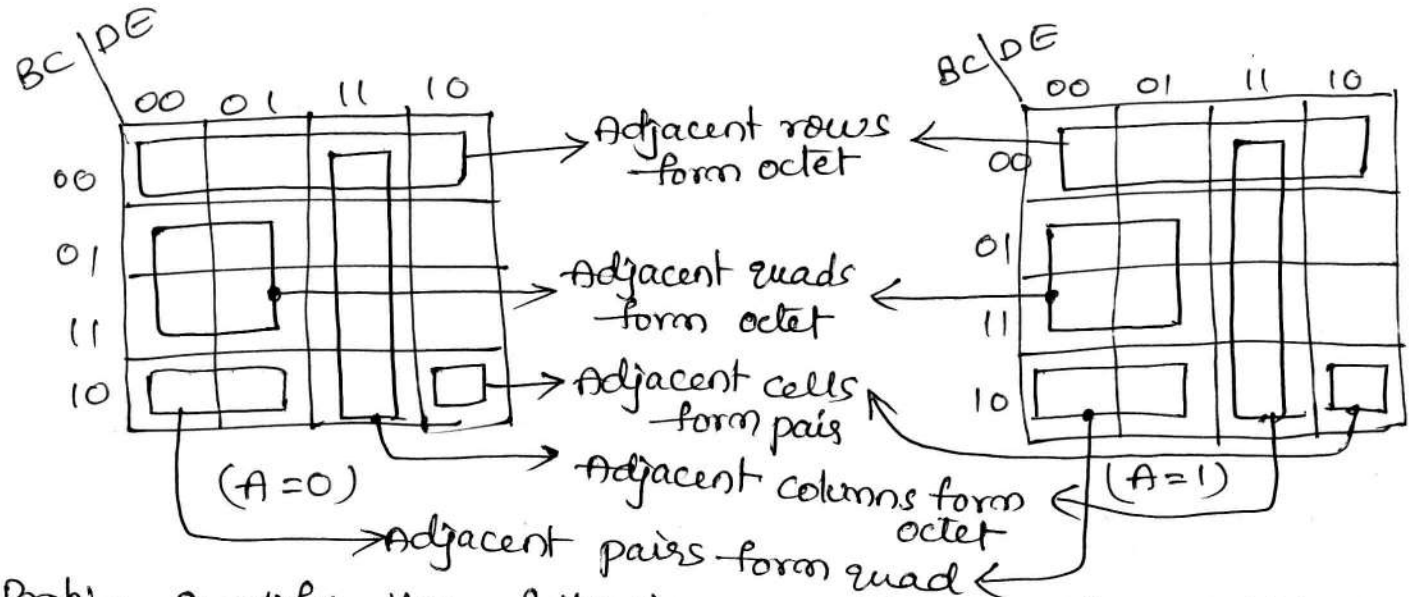
(A=1)

Grouping of cells for simplification:-

→ Here one 16-cell K-map is mirror image of other 16-cell K-map.

→ means Every cell in one map is adjacent to the corresponding cell in the other map because only one variable change between such corresponding cells.

→ Ex:- 0th cell adjacent to 16th cell.
00000 → 10000 only one variable change.



Prob:- Simplify the following Boolean functions using K-map. & implement using gates.

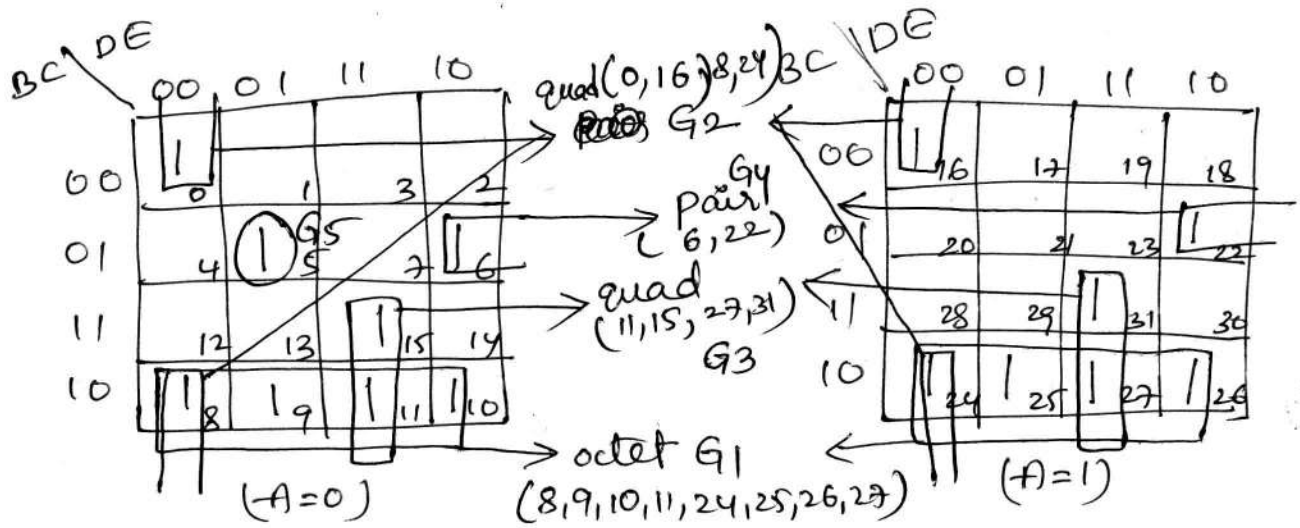
$f = \Sigma(0,5,6,8,9,10,11,15,16,22,24,25,26,27,31)$

Sol:- Given function $f = \Sigma(0,5,6,8,9,10,11,15,16,22,24,25,26,27,31)$.

Step(1):- function contains 5-variables. Plot 5-V K-map to represent given function.

step(2):- minterms are marked by 1's in corresponding cells.

step(3):- group possible combinations like pais, quad and octet.



G_1 — octet — 8,9,10,11,24,25,26,27

G_2 — quad — 0,16,8,24

G_3 — quad — 11,15,27,31

G_4 — pair — 6,22

G_5 — 5 — single group.

step(4):- Simplified expression $f = G_1 + G_2 + G_3 + G_4 + G_5$

$G_1 = \bar{A}B\bar{C}\bar{D}\bar{E} + \bar{A}B\bar{C}\bar{D}E + \bar{A}B\bar{C}DE + \bar{A}B\bar{C}D\bar{E} + AB\bar{C}\bar{D}\bar{E} + AB\bar{C}\bar{D}E + AB\bar{C}DE + AB\bar{C}D\bar{E}$

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

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

$G_2 = \bar{A}\bar{B}\bar{C}\bar{D}\bar{E} + \bar{A}B\bar{C}\bar{D}\bar{E} + A\bar{B}\bar{C}\bar{D}\bar{E} + AB\bar{C}\bar{D}\bar{E}$

$= \bar{A}\bar{C}\bar{D}\bar{E} + A\bar{C}\bar{D}\bar{E} = \boxed{\bar{C}\bar{D}\bar{E}}$

$G_3 = \bar{A}BCDE + \bar{A}B\bar{C}DE + ABCDE + AB\bar{C}DE$

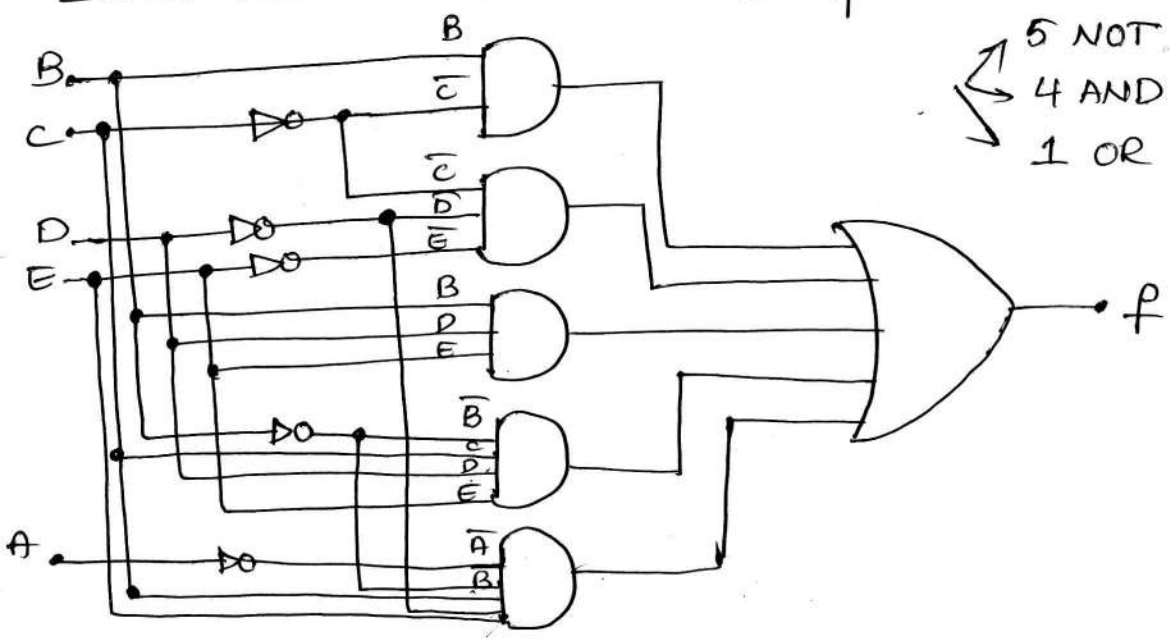
$= \bar{A}BDE + ABDE = \boxed{BDE}$

$G_4 = \bar{A}\bar{B}CD\bar{E} + AB\bar{C}D\bar{E} = \boxed{\bar{B}CD\bar{E}}$; $G_5 = \bar{A}\bar{B}C\bar{D}E$

Simplified expression $f = B\bar{C} + \bar{C}\bar{D}\bar{E} + BDE + \bar{B}CD\bar{E} + \bar{A}\bar{B}C\bar{D}E$

Implementation:-

To implement we require
5 NOT gates
4 AND gates &
1 OR gate.



* NAND- NOR Implementation:-

→ NAND, NOR gates are called universal gates. Because we can construct any logic gate by using these gates.

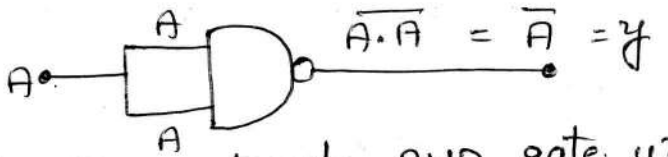
Construction of logic gates by using NAND:-

(i) **NOT gate:-** To construct NOT gate we require one NAND gate. General expression for NOT gate is

$$y = \overline{A}$$ here A is input, y is output.

we can write it $x \cdot x = x$

$$\Rightarrow \boxed{y = \overline{A \cdot A}}$$



(ii) **AND gate:-** To construct AND gate we require 2 NAND gates

General expression for AND gate is

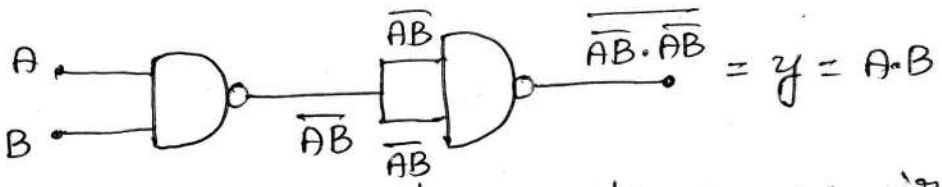
$$y = A \cdot B$$

A, B are inputs, y is output.

we can write it $(x')' = x$

$$\Rightarrow y = \overline{\overline{A \cdot B}}$$

$$\boxed{y = \overline{\overline{AB} \cdot \overline{AB}}} \quad [x \cdot x = x]$$



(iii) **OR gate:-** To construct OR gate we require 3 NAND gates

General expression for OR gate is

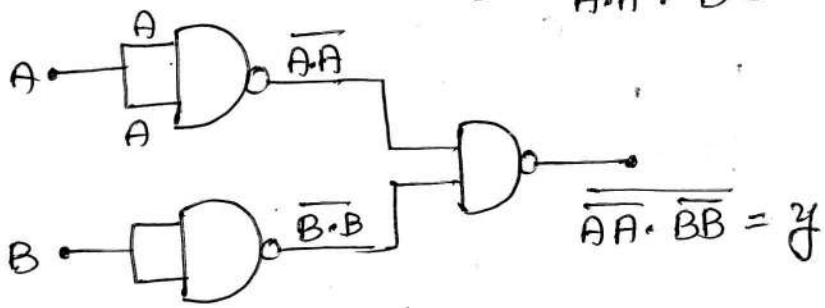
$$y = A + B$$

A, B are inputs, y is output

we can write it $(x+y)' = x' \cdot y' \Rightarrow x+y = (x' \cdot y')'$

$$(x')' = x.$$

$$\Rightarrow y = \frac{\overline{\overline{A} \cdot \overline{B}}}{\overline{A \cdot A} \cdot \overline{B \cdot B}} \quad [x \cdot x = x]$$

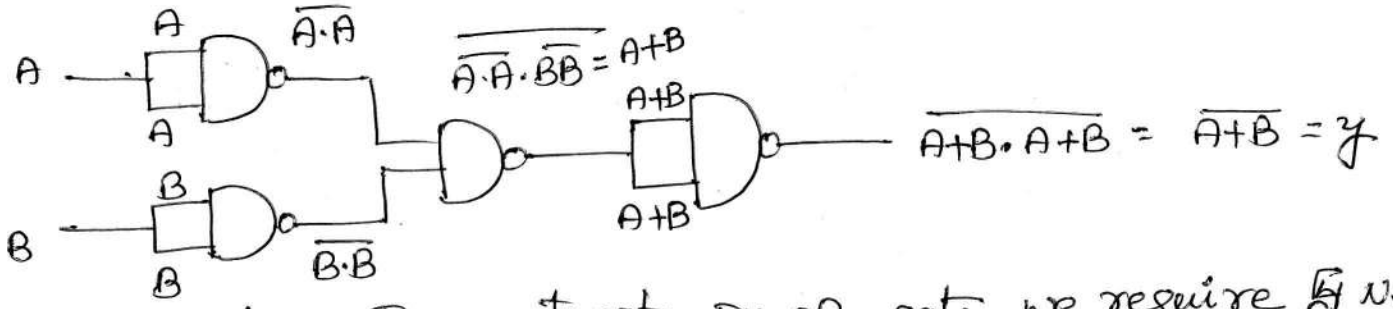


(iv) NOR gate:- To construct NOR gate we require 4 NAND gates. The general expression is

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

$$y = \overline{\overline{A}\cdot\overline{B}} = A+B \qquad \{(x+y)' = x'\cdot y' \quad ((x'\cdot y')')' = x'\cdot y\}$$

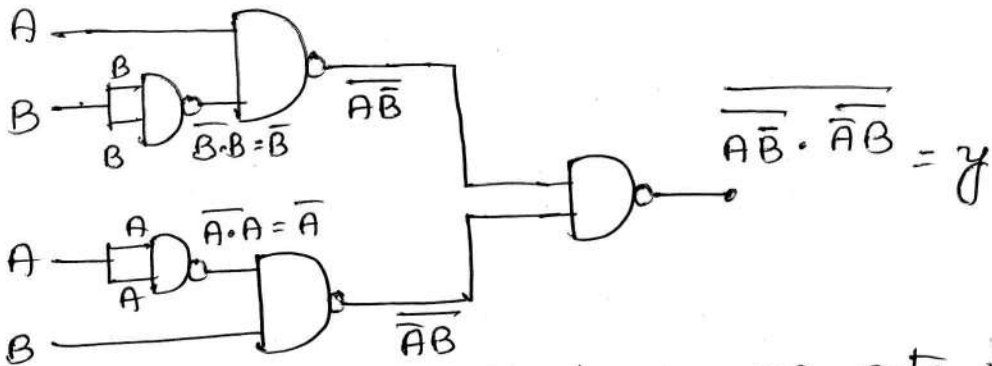
$$y = \overline{\overline{AA}\cdot\overline{BB}} = \overline{A+B}$$



(v) EX-OR gate:- To construct EX-OR gate we require 5 NAND gates. The general expression is

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

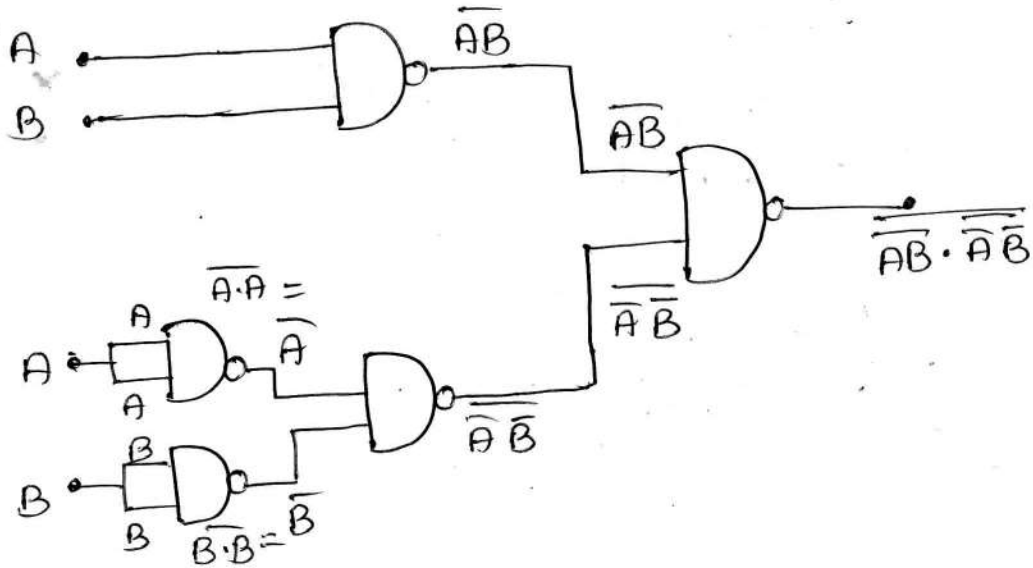
$$y = [\overline{A\overline{B}}\cdot\overline{\overline{A}B}] \qquad [x+y = (x'\cdot y')']$$



(vi) EX-NOR gate:- To construct EX-NOR gate we require 5 NAND gates. The general expression is

$$y = AB + \overline{A}\overline{B}$$

$$= [\overline{AB}\cdot\overline{\overline{A}\overline{B}}] \qquad [x+y = (x'\cdot y')']$$



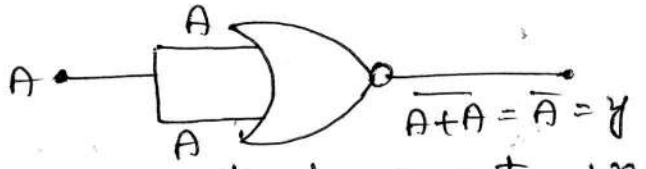
Construction of logic gates by using NOR:-

(i) NOT gate:- To construct NOT gate we require one NOR gate.

The general expression is $y = \bar{A}$

We can write it $x + x = x$

$$y = \overline{A+A} = \bar{A}$$

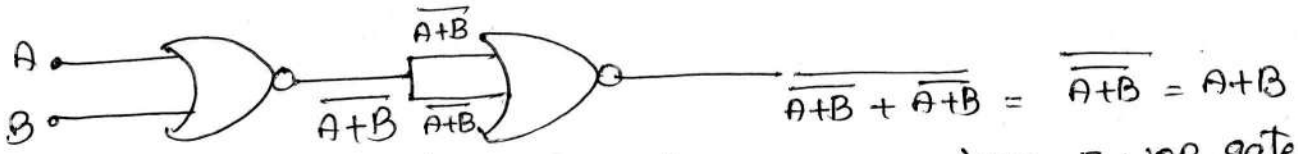


(ii) OR gate:- To construct OR gate we require 2 NOR gates.

The general expression is $y = A+B$ $\quad [x+x=x]$

We can write it like $(x')' = x$

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



(iii) AND gate:- To construct AND gate we require 3 NOR gates.

The general expression is $y = A \cdot B$

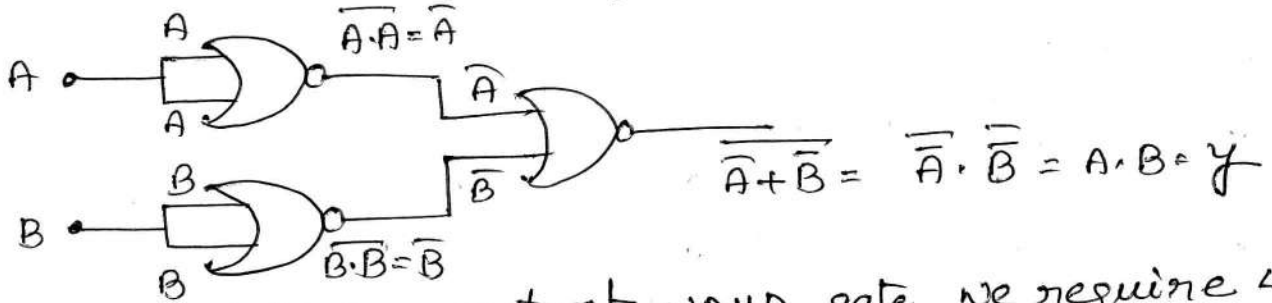
$$\Rightarrow y = \overline{[\bar{A} + \bar{B}]}$$

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

$[(x \cdot y)' = x' + y']$

$((x \cdot y)')' = x \cdot y$

$= [(x')' + (y')']$

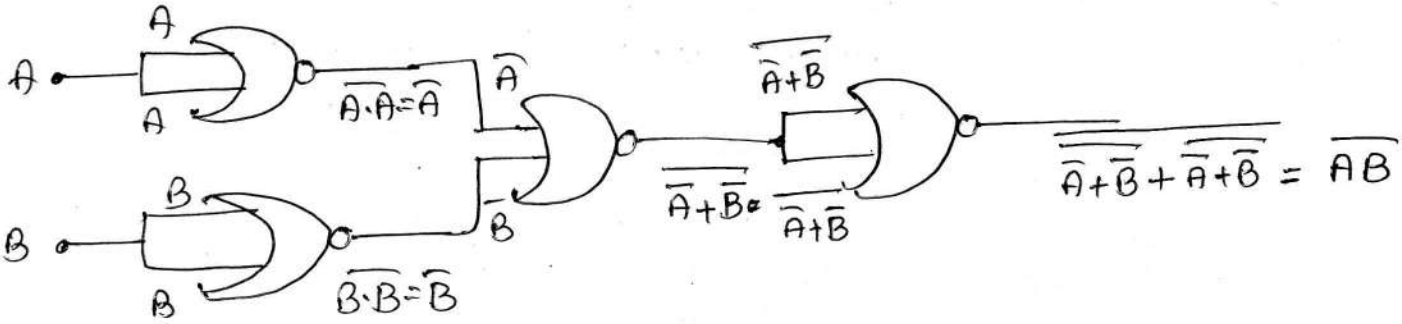


(iv) NAND gate:- To construct NAND gate we require 4 NOR gates.

The general expression is $y = \overline{A \cdot B}$

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

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



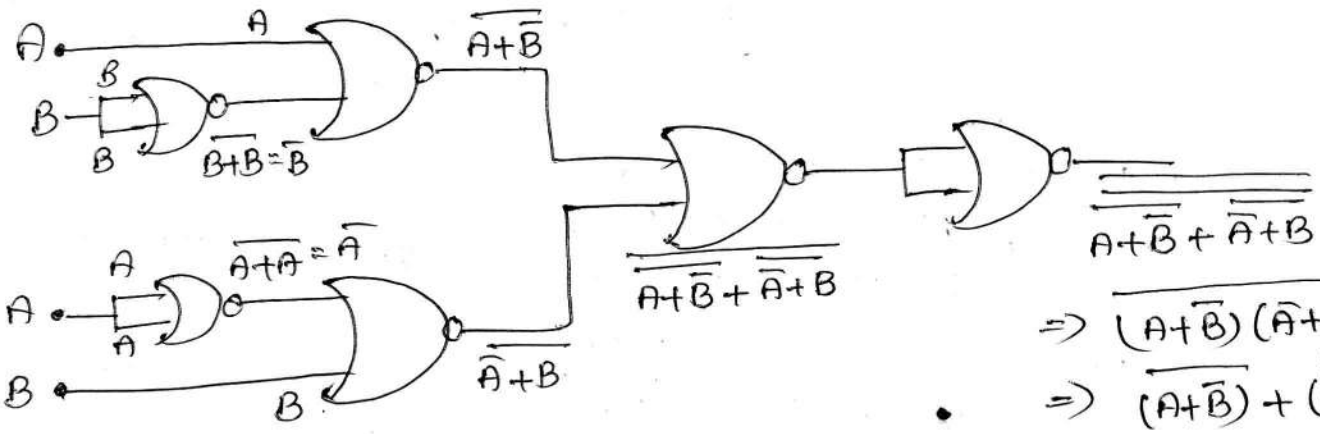
(v) EX-OR gate:- To construct Ex-OR gate we require 6 NOR gates. The general expression is

$$Y = A\bar{B} + \bar{A}B$$
$$= \overline{\overline{A+\bar{B}} + \overline{\bar{A}+B}}$$

$(x')' = x$

$[x'+y']' = x \cdot y$

$(x \cdot y)' = x'+y'$



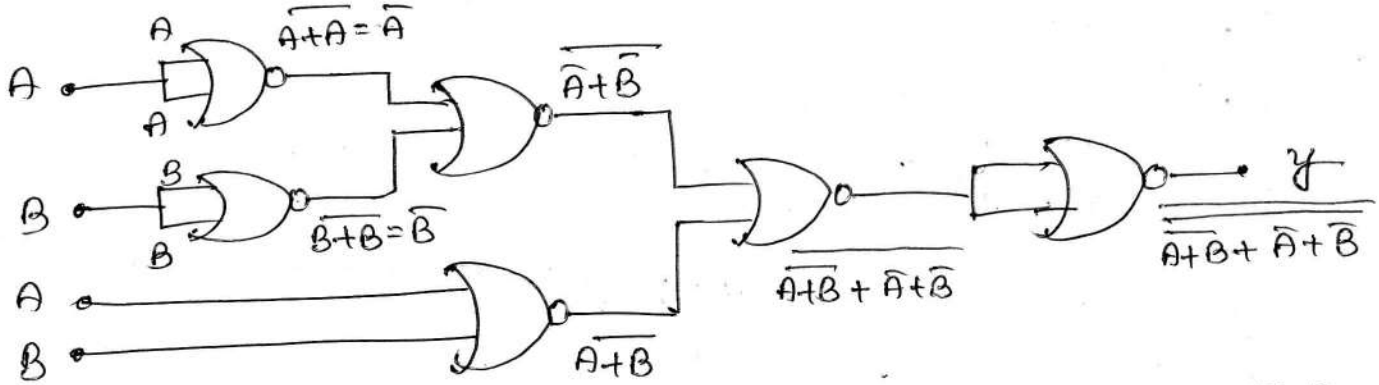
$\Rightarrow \overline{(A+\bar{B})(\bar{A}+B)}$

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

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

(vi) EX-NOR gate:- To construct Ex-NOR gate we require 6-NOR gates. The general expression is

$$Y = AB + \bar{A}\bar{B}$$
$$= \overline{A+\bar{B}} + \overline{\bar{A}+\bar{B}} \qquad (x')' = x$$
$$= \overline{(A+\bar{B})(\bar{A}+B)}$$
$$= \overline{(\bar{A}+B)} + \overline{(\bar{A}+\bar{B})}$$



* Two level Implementation:- [Using NAND] [NAND-NAND]

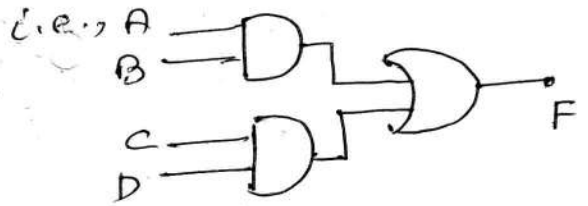
→ If we want to implement a Boolean function with NAND gates, the function must be in sum of products form. It is also called NAND-NAND implementation.

→ Consider one example

Given function $F = AB + CD$

It is in standard SOP form.

→ To implement this function we require 2 AND gates & 1 OR Gate.

i.e., 

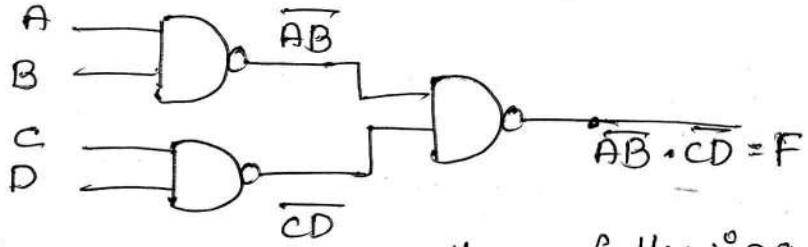
⇒ We can also implement this function by using NAND gates only.

$F = AB + CD$

$F = \overline{\overline{AB} \cdot \overline{CD}}$

$\{ x' \cdot y' = (x+y)' \}$
$\{ (x' + y')' = x+y \}$

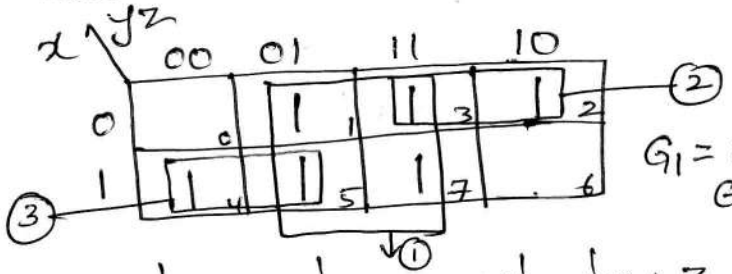
Then implementation of F requires 3 NAND gates



⇒ This implementation is called two-level implementation.

Prob:- Implement the following Boolean function with NAND gates. $F(x,y,z) = \Sigma(1,2,3,4,5,7)$

Sol:- Simplify the function in sum of products.



$F = G_1 + G_2 + G_3$

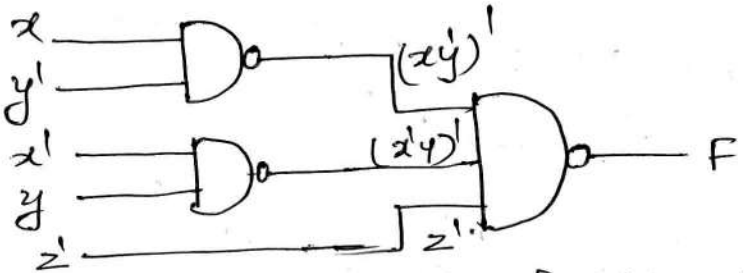
$G_1 = z \quad G_2 = x'y \quad G_3 = xy'$

$F = z + x'y + xy' = xy' + x'y + z$

implementation of F by using NAND gates.

$F = [(xy')' \cdot (x'y)' \cdot z']'$

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



* Two level Implementation:- [Using NOR] [NOR-NOR]

→ If we want to implement a Boolean function with NOR gates, the function must be in product of sums form.

→ It is also called NOR-NOR implementation.

→ Consider one example

Given function $F = (A+B)(C+D)E$.

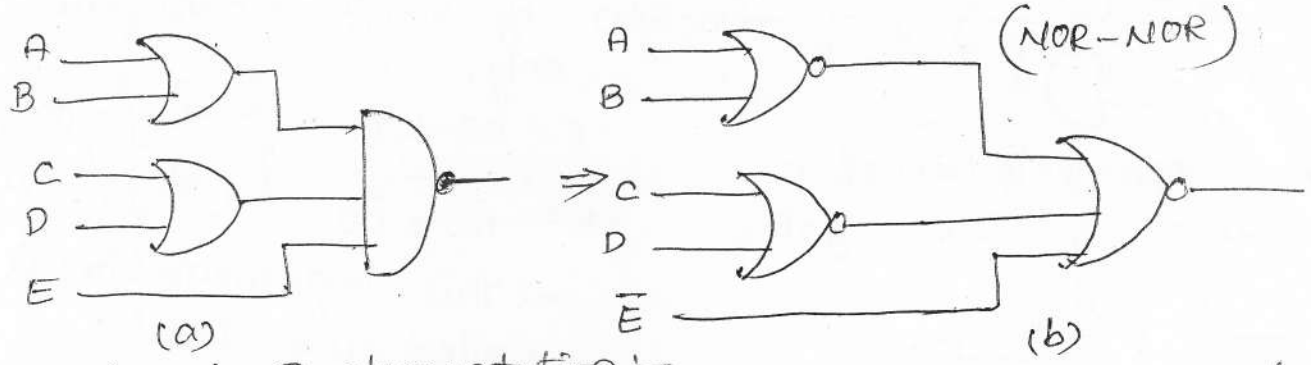
→ To implement this function generally we require 2 OR gates & ONE AND gate.

→ We can also implement this function F using NOR gates. i.e.,

$F = \overline{\overline{(A+B)(C+D)E}}$

$= \overline{[\overline{A+B} + \overline{C+D} + \overline{E}]}$

$\{ (xyz)' = x'+y'+z' \}$
$\{ xyz = (x'+y'+z')' \}$



* Multilevel Implementation :-

→ we have to express the Boolean function in terms of AND, OR and complement operations.

→ Ex:- $F = A(CD+B) + B\bar{C}$ implement using NAND gates.

Sol :- $F = A(CD+B) + B\bar{C}$

$\{x+y = (x'\cdot y')'\}$

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

$$F = \overline{\overline{A\,[\overline{CD}\cdot\bar{B}]}\cdot\overline{B\bar{C}}}$$

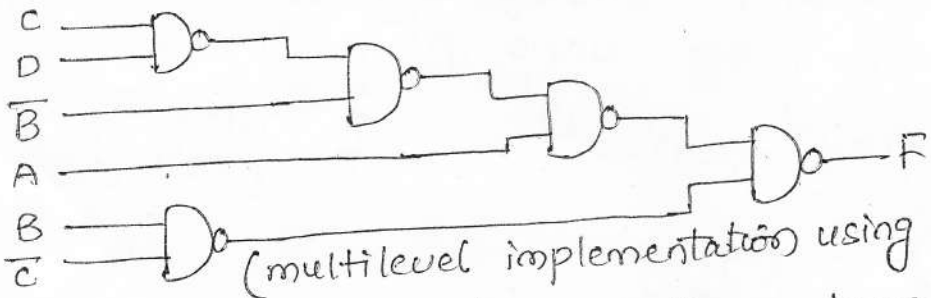
$$\overline{A(CD+B)} = \overline{\overline{A(CD+B)}}$$

$$= \overline{\overline{A} + \overline{(CD+B)}}$$

$$= \overline{\overline{A} + \overline{CD}\cdot\overline{B}}$$

$$= \overline{\overline{\overline{A}}\ (\overline{CD}\cdot\overline{B})}$$

$$= \overline{A\ (\overline{CD}\cdot\overline{B})}$$



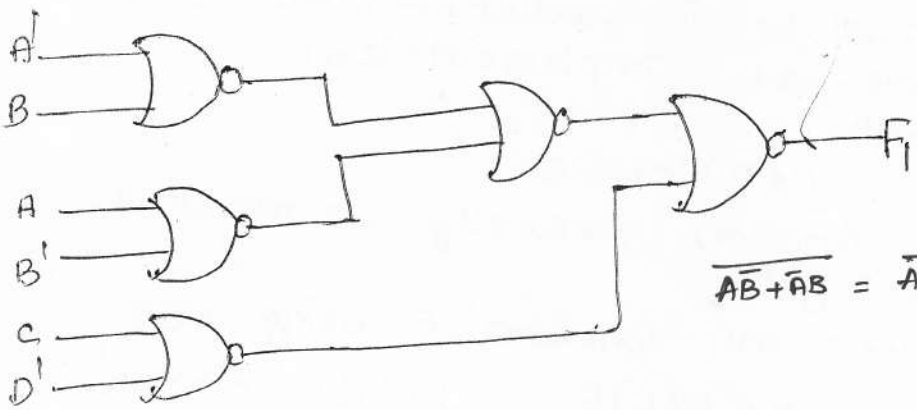
(multilevel implementation using NAND gates)

→ Ex:- $F_1 = (AB' + A'B)(C + D')$ implement using NOR gates.

$\{x\cdot y = (x'+y')'\}$

$$= \left[\left[AB' + A'B\right]' + (C+D')'\right]'$$

$$= \left[\left\{\left[A'+B\right]' + \left[A+B'\right]'\right\}' + (C+D')'\right]'$$



(multilevel implementation using NOR gates)

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

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

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

* Other Two-level Implementation :-

→ other two-level implementations are

(1) AND-OR-INVERT implementation (AOI)

(2) OR-AND-INVERT implementation (OAI)

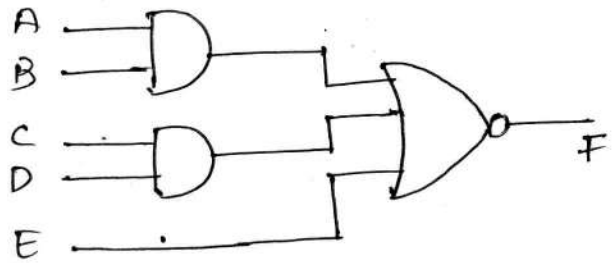
(1) AOI implementation :- function should be in sum of products with whole complement.

Ex :- $F = (AB+CD+E)'$.

we can implement this function by using AND-NOR and NAND-AND functions. $F = (AB+CD+E)'$

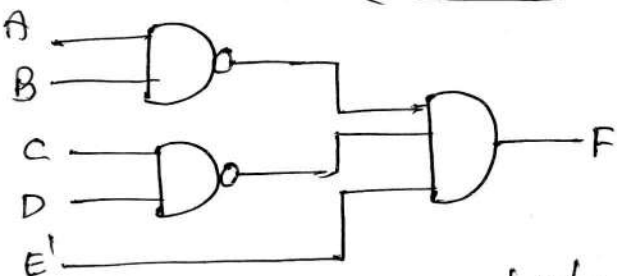
$= (AB)' \cdot (CD)' \cdot E'$

(AND-NOR)



⇒

(NAND-AND)



(2) OAI implementation :- function should be in product of sums with whole complement.

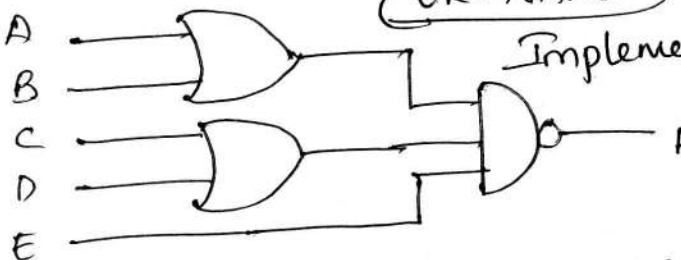
Ex :- $F = [(A+B)(C+D)E]'$

we can implement this function by using OR-NAND and NOR-OR functions

$F = [(A+B)(C+D)E]'$

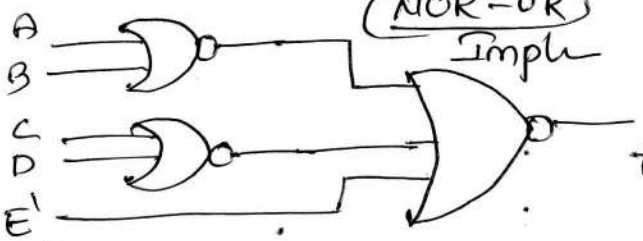
$= [(A+B)' + (C+D)' + E']$

OR-NAND Implementation



⇒

NOR-OR Implementation



* Other Minimization Methods :-

→ Quine-Mccluskey (Tabulation) (Q-M) Method :-

→ To simplify a boolean function of large number (more than 6) of variables, we use Tabulation method.

→ It was first introduced by Quine and then improved by Mccluskey.

→ To find minimal expression here we take two terms called Prime Implicant (PI) & Essential Prime Implicant (EPI).

Prime Implicant (PI) :- Here Implicant denotes the set of adjacent minterms.
→ An implicant is called a prime implicant if it is not a subset of another implicant of the function.
Essential Prime Implicant (EPI) :- A prime implicant which includes at least 1 cell (minterm) that not included in any other prime implicant is called essential prime implicants.

* Tabulation method Procedure :-

Problem :- Simplify the following boolean function by using Q-M method. $f(A,B,C,D) = \Sigma m(0,1,2,5,7,8,9,10,13,15)$.

Step ① :- List all the minterms in the binary form and arrange them in the form of groups according to the number of 1's contained in binary format.
→ This is done in below table.

minterm	Binary Representation ABCD	Number of 1's
m_0	0000	0
m_1	0001	1
m_2	0010	1
m_5	0101	2
m_7	0111	3
m_8	1000	1
m_9	1001	2
m_{10}	1010	2
m_{13}	1101	3
m_{15}	1111	4

minterms	Index	Binary representation ABCD
m_0	Index 0	0000
m_1	Index 1	0001
m_2		0010
m_8		1000
m_5	Index 2	0101
m_9		1001
m_{10}		1010
m_7	Index 3	0111
m_{13}		1101
m_{15}	Index 4	1111

Step ② :- Compare each binary term with every term of the next index, and if they differ by one position, copy the term in the next column with '_' in the position they differed.
→ Apply the same process for the resultant column and continue these cycles until no further elimination of literals. Below tables shown step 2 procedure.

(a)

Minterm group	Binary representation ABCD
0,1	000– ✓
0,2	00–0 ✓
0,8	–000 ✓
1,5	0–01 ✓
1,9	–001 ✓
2,10	–010 ✓
8,9	100– ✓
8,10	10–0 ✓
5,7	01–1 ✓
5,13	–101 ✓
9,13	1–01 ✓
7,15	–111 ✓
13,15	11–1 ✓

(b)

minterm groups	Binary representation ABCD
0,1,8,9	–00– (P)
0,2,8,10	–0–0 } Both are same (Q)
0,8,2,10	–0–0 }
1,5,9,13	–-01 } Both are same (R)
1,9,5,13	–-01 }
5,7,13,15	–1–1 } Both are same (S)
5,13,7,15	–1–1 }

step③:- from above 2 tables there are 4 uncombined terms. These uncombined terms are called prime Implicants. These are represented by capital alphabets. let P, Q, R, S.

step④:- List all the prime Implicants in the table & this table called as Prime–Implicant table.

	PI's	Binary representation ABCD	Literal representation
P	0,1,8,9	–00–	$\bar{B}\bar{C}$
Q	0,2,8,10	–0–0	$\bar{B}\bar{D}$
R	1,5,9,13	–-01	$\bar{C}D$
S	5,7,13,15	–1–1	BD

step⑤:- List essential prime implicants from PI table. Represent corresponding minterms by 'x'.

	PI's	Binary repres–	Literal represe	m_0	m_1	m_2	m_5	m_7	m_8	m_9	m_{10}	m_{13}	m_{15}
P	0,1,8,9	–00–	$\bar{B}\bar{C}$	X	X				X	X			
Q	0,2,8,10	–0–0	$\bar{B}\bar{D}$	X		(X)			X		(X)		
R	1,5,9,13	–-01	$\bar{C}D$		X		X			X		X	
S	5,7,13,15	–1–1	BD				X	(X)				X	(X)

→ In the columns m_2, m_7, m_{10}, m_{15} contain only one ×. and theis corresponding PI must be included in the minimal function and they called EPI. (10)

→ Here Q & S are EPI.

Step (6):- Represent minimal expression by using EPI's.

Then ~~minimal expression~~ is ~~Q+S+P (or) Q+R+S~~

$= \bar{B}\bar{D} + BD$

Because Q & S covered 0, 2, 8, 10, 5, 7, 13, 15 except 1 & 9. those 1 & 9 can be covered using either P or R.

Then minimal expression is

$f = Q + S + P$ (or) $Q + R + S$

$= \bar{B}\bar{C} + \bar{B}\bar{D} + BD$ (or) $\bar{B}\bar{D} + \bar{C}D + BD$.

Prob(2):- Simplify the following Boolean function using Q-M method. $f(A,B,C,D) = \Sigma m(1,2,3,5,9,12,14,15) + \Sigma d(4,8,11)$.

Sol:- Step(1):-

minterm	Binary representation ABCD	No of 1's
m_1	0001	1
m_2	0010	1
m_3	0110	2
m_5	0101	2
m_9	1001	2
m_{12}	1100	2
m_{14}	1110	3
m_{15}	1111	4
d_4	0100	1
d_8	1000	1
d_{11}	1011	3

Minterms	Index	Binary representation
m_1	Index 1	0001
m_2		0010
d_4		0100
d_8		1000
m_3	Index 2	0011
m_5		0101
m_9		1001
m_{12}		1100
m_{14}	Index 3	1110
d_{11}		1011
m_{15}	Index 4	1111

step(2):-

(a)

Minterm group	Binary representation
1,3	00–1 ✓
1,5	0–01 (Q)
1,9	–001 ✓
2,3	001– (R)
4,5	010– (S)
4,12	–100 (T)
8,9	100– (U)
8,12	1–00 (V)
3,11	–01 ✓
9,11	10–1 ✓
12,14	11–0 (W)
14,15	111– (X)
11,15	1–11 (Y)

(b) (11)

Minterm group	Binary representation
1,3,9,11	–0–1 } Both are same (P)
1,9,3,11	–0–1 }
step (3) :- Here only one combinations :	we will get 10 unmatched i.e., 10 PI.

step (4) :- Draw PI table.

PI	Binary representation	literal representation
P– 1,3,9,11	–0–1	$\bar{B}D$
Q– 1,5	0–01	$\bar{A}\bar{C}D$
R– 2,3	001–	$\bar{A}\bar{B}C$
S– 4,5	010–	$\bar{A}B\bar{C}$
T– 4,12	–100	$B\bar{C}\bar{D}$
U– 8,9	100–	$A\bar{B}\bar{C}$
V– 8,12	1–00	$A\bar{C}\bar{D}$
W– 12,14	11–0	$AB\bar{D}$
X– 14,15	111–	ABC
Y– 11,15	1–11	ACD

step ⑤ :- List EPI's

PI	Binary representation	Literal representation	minterms 1	2	3	5	9	12	14	15	4	8	11
P 1,3,9,11	-0-1	$\bar{B}D$	X		X		X						X
Q 1,5	1-01	$A\bar{C}D$	X			X							
R 2,3	001-	$\bar{A}\bar{B}C$		⊗	X								
S 4,5	010-	$\bar{A}B\bar{C}$				X					X		
T 4,12	-100	$B\bar{C}\bar{D}$						X			X		
U 8,9	100-	$A\bar{B}\bar{C}$					X					X	
V 8,12	1-00	$A\bar{C}\bar{D}$						X	●			X	
W 12,14	11-0	$AB\bar{D}$						✖	X	●			
X 14,15	111-	ABC							X	X			
Y 11,15	1-11	ACD								X			X

→ Here we will get only one EPI i.e., R.
R covers only 2 minterms 2,3.

→ other minterms are 1, 5, 9, 12, 14, 15

Q covers 1,5 ; T covers 12 ;
X covers 14,15 ; U covers 9.

→ Then minimal expression is $f = R + Q + X + T + U$

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