# REVIEW OF NUMBER SYSTEMS & CODES.

## Number systems :-

→ Number systems are used to represent data in digital form

→ Number system is nothing more than a code that uses symbols to represent a number.

→ In general, in any number system, there is an ordered set of symbols known as digits.

→ A number is made up of a collection of digits and it has two parts.

- integer and fraction part.

→ Both are separated by a radix point (·). The number is represented as,

$$(\underbrace{d_{n-1}\, d_{n-2} \cdots d_1 d_0}_{\text{Integer part}} \cdot \underbrace{d_{-1}\, d_{-2} \cdots d_{-m}}_{\text{Fractional part}})_r \quad r \rightarrow \text{radix}$$

Radix point.

Number systems are classified as.

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

## Decimal number system :-

→ The decimal number system is a radix 10. It has 10 different digits or symbols to represent a number.

→ These are 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9.

→ The radix point is known as the decimal point.

→ The value of any mixed decimal number is

$$d_n . d_{n-1} d_{n-2} \cdots d_1 d_0 . d_{-1} d_{-2} d_{-3} \cdots d_{-m}.$$

is given by.

$$(d_n \times 10^n) + (d_{n-1} \times 10^{n-1}) + \cdots + (d_1 \times 10^1) + (d_0 \times 10^0) + (d_{-1} \times 10^{-1}) + (d_{-2} \times 10^{-2}) + \cdots$$

Consider the decimal number.

$$(135.56)_{10} = 1 \times 10^2 + 3 \times 10^1 + 5 \times 10^0 + 5 \times 10^{-1} + 6 \times 10^{-2}$$

$$= 100 + 30 + 5 + 0.5 + 0.06$$

$$= 135.56.$$

Binary number system :-

→ The Binary number system is a radix 2. It has Two independent ~~digits~~ Bits.

→ Binary numbers are represented in terms of '0' and '1'.

→ The radix point is known as the Binary point.

'0' or '1' is a bit.

Four number bits → Nibble

8 bits. → byte

Group of 16 bits → word.

→ The binary number system is used in digital computers because the switching circuits used in these computers use two-state devices such as transistors, diodes, etc.

## Octal Number System :-

→ The octal number system is a radix 8.

→ It has 8 different digits or symbols to represent a number

→ They are 0, 1, 2, 3, 4, 5, 6, and 7.

→ The radix point is known as the octal point.

→ octal number system is more efficient for us to write the number in octal rather than binary.

→ Electronically, it is easier and cheaper.

## Hexadecimal Number System :-

→ The Hexadecimal Number System is a radix 16.

→ It has 16 different digits or symbols to represent a number.

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

→ The radix point is known as the Hexadecimal point.

→ The decimal equivalent of A, B, C, D, E and F are 10, 11, 12, 13, 14 and 15.

→ In hexadecimal number system easier way of representing large binary numbers stored and processed inside the computer.

→ The contents of the memory when represented in hexadecimal form are very convenient to handle.

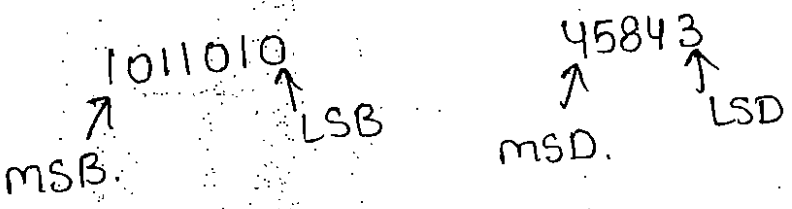
Radix - NO of symbols presented in a respective number system is called radix.

LSB - Least Significant bit.

The bit which having the least position weight bit is called LSB.

MSB - most significant bit.

The bit which having the most ~~significan~~ position weight is called MSB.




MSD - most Significant digit.

LSD - least Significant digit.

Conversion from one radix to another radix :-

In computer systems process binary data, but the information given by the user may be in the form of decimal number, hexadecimal number, & octal number.

→ Binary to decimal → octal to binary
→ octal to decimal → binary to octal
→ Hexadecimal to decimal → Hexadecimal to binary
→ decimal to binary → binary to Hexadecimal.
→ decimal to octal → octal to Hexadecimal
→ decimal to Hexadecimal → Hexadecimal to octal.

## Binary to decimal :-

$(101101.0110)_2 = (\quad)_{10}$.

$$(\underset{5\,4\,3\,2\,1\,0\;-1\,-2\,-3\,-4}{101101.0110})_2 = (1\times2^5)+(0\times2^4)+(1\times2^3)+(1\times2^2)+(0\times2^1)+(1\times2^0)+(0\times2^{-1})+(1\times2^{-2})+(1\times2^{-3})+(0\times2^{-4})$$

$$= 32+0+8+4+0+1+0+0.25+0.125+0.$$

$$= (45.375)_{10}.$$

## octal to decimal :-

→ $(4053.03)_8 = (2091\quad)_{10}$.

$$(\underset{3\,2\,1\,0\;-1\,-2}{4053.03})_8 = 4\times8^3+0\times8^2+5\times8^1+3\times8^0+0\times8^{-1}+3\times8^{-2}.$$

$$= 2048+0+40+3+0+$$

$$= 2091$$

$(532.78)_8 = (\quad)_{10}$.

In this given number system is octal but the value in the given problem 8 is not a octal number.

$$(\underset{2\,1\,0\;-1\,-2}{753.63})_8 = (491.79.)_{10}$$

$$= 7\times8^2+5\times8^1+3\times8^0+6\times8^{-1}+3\times8^{-2}.$$

$$= 448+40+3+0.75+0.046$$

$$= 491.79$$

Hexadecimal to decimal:-

→ $(5A3)_{16} = (1443.)_{10}$

$(5A3)_{16} = (5 \times 16^2) + (10 \times 16^1) + (3 \times 16^0)$

$= 1280 + 160 + 3$

$= (1443)_{10}$

→ $(64D)_{16} = (1613)_{10}$.

$(64D)_{16} = (6 \times 16^2) + (4 \times 16^1) + (13 \times 16^0)$.

$= 1,536 + 64 + 13$

$= (1613)_{10}$

Decimal to binary :-

→ To convert a decimal number to a binary number is known as repeated division and multiplication method

→ The integer and fractional parts of a decimal number are treated separately for the conversion and then the results are combined.

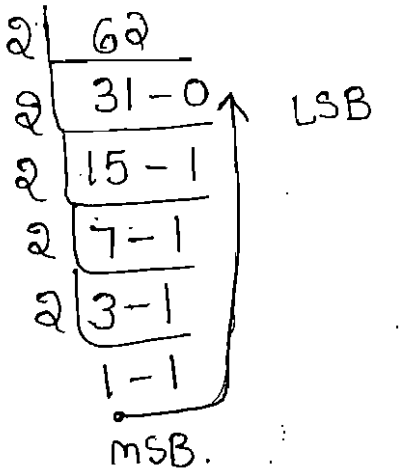
→ Integer part conversion.

To convert the integer part by using the repeated division method, the given decimal number is divided by 2

The remainder is found after each division until the quotient 0.

The remainder read from bottom to top give the equivalent binary integer number.

$(62)_{10} = (\quad)_2$




$(62)_{10} = (111110)_2$

Fractional part conversion.

To convert the fractional part by using the repeated multiplication method, the given decimal number is multiplied by 2.

The integer part of the multiplication is found after each multiplication operation until the fractional part of a decimal number becomes zero.

The integers read from top to bottom gives the equivalent Binary fraction.

$(0.625)_{10} = (\quad)_2$.

$0.625 \times 2 = 1.25 \quad 1$

$0.25 \times 2 = 0.5 \quad 0$

$0.5 \times 2 = 1.0 \quad 1$ ↓

$(0.625)_{10} = (101)_2$

→ $(105.15)_{10} = (\quad)_2$

| | |
|---|---|
| 2 \| 105 | |
| 2 \| 52 - 1 | ↑ |
| 2 \| 26 - 0 | |
| 2 \| 13 - 0 | |
| 2 \| 6 - 1 | |
| 2 \| 3 - 0 | |
| 1 - 1 | |

integer part.

$105_{10} = 1101001_2$

$0.15 \times 2 = 0.30$ 0
$0.30 \times 2 = 0.60$ 0
$0.60 \times 2 = 1.20$ 1
$0.20 \times 2 = 0.40$ 0
$0.40 \times 2 = 0.80$ 0
$0.80 \times 2 = 1.60$ 1 ↓

Fractional part

$0.15_{10} = 0.001001_2$.

$(105.15)_{10} = (1101001.001001)_2$.

Decimal to octal :-

Decimal conversion is same as decimal to binary except that in this case repeated multiplication and division are by 8 which is the base of octal number system.

$(183.62)_{10} = (\quad)_8 = (267.475)_8$.

| | |
|---|---|
| 8 \| 183 | |
| 8 \| 22 - 7 | |
| 2 - 6 | |

$0.62 \times 8 = 4.96$ 4
$0.96 \times 8 = 7.68$ 7
$0.68 \times 8 = 5.44$ 5.

$(145.93)_{10} = (\quad)_{8} = (221.734)_{8}$

8 | 145
8 | 18 - 1
8 | 2 - 2
0 - 2

$0.93 \times 8 = 7.44$
$0.44 \times 8 = 3.52$
$0.52 \times 8 = 4.16.$

## Decimal to Hexadecimal :-

Decimal to Hexadecimal conversion is same as decimal to octal. except that in this case repeated multiplication and division are by 16 which is the base of Hexadecimal number system.

→ $(138.58)_{10} = (80A.947.)_{16}$

$16 \times 8 = 128$
$138 - 128 = 10$

16 | 138
16 | 128 - 10
8 - 0

$0.58 \times 16 = 9.28$
$0.28 \times 16 = 4.48$
$0.48 \times 16 = 7.68.$

16 | 138 (L)
16 | 8 - 10 (A)
0 - 8 (M)

→ $(256.26)_{10} = (100.428F)_{16}$

16 | 256
16 | 16 - 0
1 - 0

$0.26 \times 16 = 4.16$
$0.16 \times 16 = 2.56.$
$0.56 \times 16 = 8.96$
$0.96 \times 16 = 15.36.$

## octal to binary conversion :-

To convert octal to binary. just replace each octal digit by its 3-bit binary equivalent.

→ $(563)_8 \rightarrow (101110011)_2$

→ $(725)_8 \rightarrow (111010101)_2$

→ $(326)_8 \rightarrow$ 3 2 6

011 010 110

$(011010110)_2$.

## Binary to octal conversion :-

To convert binary to octal, just replace each group of 3 bits by equivalent octal number.

→ Splitting the integer and fractional parts into Groups of three bits. starting from the binary point on both sides.

→ In integer part to making group of 3-bits from right to left

→ In fractional part to making group of 3-bits from left to right.

→ If needed add 0's on the extreme left of the integer part and extreme right of the fractional part.

→ $(110110.01101)_2 \longrightarrow (155.32)_8$

| 001 | 101 | 101 | . | 011 | 010 |
|---|---|---|---|---|---|
| 1 | 5 | 5 | . | 3 | 2 |

→ $(101101010.1101010)_2 \longrightarrow (552.650)_8$

| 101 | 101 | 010 | . | 110 | 101 | 000 |
|---|---|---|---|---|---|---|
| 5 | 5 | 2 | | 6 | 5 | 0 |

Hexadecimal to Binary conversion :-

To convert a hexadecimal number to binary, replace each digit by it's 4-bit binary equivalent.

Example :-

→ $(6A3)_{16} \longrightarrow (\quad)_2$

| 6 | A | 3 |
|---|---|---|
| 0110 | 1010 | 0011 |

$(6A3)_{16} \longrightarrow (011010100011)_2$

A - 10
B - 11
C - 12
D - 13
E - 14
F - 15

→ $(58C.26)_{16} \longrightarrow (\quad)_2$

| 5 | 8 | C | . | 2 | 6 |
|---|---|---|---|---|---|
| 0101 | 1000 | 1100 | | 0010 | 0110 |

$(58C.26)_{16} \longrightarrow (010110001100.0010\ 0110)_2$

→ Binary to Hexadecimal conversion :-

To convert binary to Hexadecimal number. by splitting the integer and fractional parts into Groups of 4-bits. Replace each 4-bit binary group by the equivalent hexadecimal digit.

→ $(1001011010101)_2 \longrightarrow (\quad)_{16}$.

| 0001 | 0010 | 1101 | 0101 |
|---|---|---|---|
| 1 | 2 | D | 5. |

$(1001011010101)_2 \rightarrow (12D5)$.

→ $(1101101010101.1010101)_2 \rightarrow (\quad)_{16}$.

| 0001 | 1011 | 0101 | 0101 | . | 1010 | 1010 | 1000 |
|---|---|---|---|---|---|---|---|
| 1 | B. | 5 | 5. | . | A | A | 8. |

$(1101101010101.10101010)_2 \rightarrow (1B55.AA8)_{16}$.

→ $(110101101.001101)_2$.

| 0011 | 0110 | 1101 | . | 0011 | 0100 |
|---|---|---|---|---|---|
| 3. | 6 | D | . | 3. | 4. |

→ Octal to Hexadecimal conversion

To convert an octal number to hexadecimal, first convert the given octal number to binary and then the binary number to hexadecimal.

→ $(356.63)_8 \longrightarrow (\quad)_{16}$.

1. octal to binary
2. binary to hexadecimal.

| 3 | 5 | 6 | . | 6 | 3 |
|---|---|---|---|---|---|
| 011 | 101 | 110 | . | 110 | 011 |

$(011\,101\,110\,.\,110\,011)_2$

| 0000 | 1110 | 1110 | . | 1100 | 1100 |
|---|---|---|---|---|---|
| 0 | E | E | . | C | C |

$(356.63)_8 \longrightarrow (0EE.CC)_{16}$

→ Hexadecimal to octal conversion

To convert a hexadecimal number to octal, first convert the given Hexadecimal number to binary and then the binary number to octal.

→ $(B9F.AE)_{16} \longrightarrow (\quad)_8$

1. Hexadecimal to binary
2. binary to octal.

| B | 9 | F | . A | E |
|---|---|---|---|---|
| 1011 | 1001 | 1111 | 1010 | 1110 |

| 101 | 1001 | 1111 | . 1010 | 1110 0 |
|---|---|---|---|---|
| 5 | 6 | 3 | 7 | 5 | 3 | 4 |

$(B9F.AE)_{16} \longrightarrow (5637.534)_8$.

Conversion from any system to any system.:-

$(1321)_5 \longrightarrow (\quad)_{12}$.

To convert any system to any system, first convert the given number to decimal number and then the decimal number to any system.

Step1:- $(1321)_5 \longrightarrow (211)_{10}$.

$(1321)_5$ — 3 2 1 0

$= 1\times5^3 + 3\times5^2 + 2\times5^1 + 1\times5^0$

$= 125 + 75 + 10 + 1$

$= (211)_{10}$

Step2:- $(211)_{10} \longrightarrow (157)_{12}$.

12 | 211

12 | 17 - 7

12 | 1 - 5

0 - 1

$(1321)_5 \longrightarrow (157)_{12}$.

## r-1's complements and r's compliments :-

Complements are used in digital systems to simplify the subtraction operation base (radix) r system there are two useful types of complements, the r's-complement (Radix complement) and the (r-1)'s-complement (Diminished Radix complement).

## 1's and 2's Complements :-

The 1's complement of a binary number is obtained by complementing all its bits, that is, by replacing all 0's by 1's and all 1's by 0's.

| | | |
|---|---|---|
| Example → | 01101010 | 10101010 1100 |
| 1's complement → | 10010101 | 01010101 0011 |

The 2's complement of a binary number is obtained by adding '1' to its 1's complement.

| | | |
|---|---|---|
| Example | 01101010 | 10101101 |
| 1's complement | 10010101 | 01010010100 |
| | +1 | +1 |
| 2's complement | 10010110 | 01010010101 |

1's complement :-

In 1's complement subtraction, add the 1s complement of the subtrahend to the minuend. If there is a carry out, bring the carry around and add it to the LSB. This is called the end around carry. If the MSB is 0, the Result is positive and is in true binary. If the MSB is '1' The Result is negative and is in its 1s complement form.

Example :-

Subtract 20 from 36 using the 8-bit 1's complement for

36 - 20

36 - 00100100 ⟶ 00100100

20 - 00010100 ⟶ +11101011 (1's complement form

100001111

1 (Add the end arou carry)

00010000

MSB.

36 - 20 = 16

The MSB is 0. The result is positive.

In 1's Complement addition, add the 1's complement form of subtrahend and minuend. If there is a carry out add the carry in LSB position.

If the MSB of the result 0, then the answer is positive and MSB of the result 1, then the answer is negative.

Example 2

Add +36 to +20 using 8 bit 1's complement form.

$36 \rightarrow 00100100 \rightarrow 11011011$

$20 \rightarrow 00010100 \rightarrow 11101011$

$\boxed{1}\,11000110$

$\hookrightarrow \quad 1$ (add the end around carry).

$11000111$

MSB is 1, then the answer is negative

$00111000 = 56$

$36+20 = 56.$

Example 3

Add -36 to -20 using 8 bit 1's complement form.

$36 \rightarrow 00100100 \rightarrow 11011011$

$00010100 \rightarrow 11101011$

$\boxed{1}\,11000110$

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

$11000111 \rightarrow$ negative.

$00111000 = 56.$

$-36-20 = -56\ (11000111)$

Example 4

Add ~~-25~~ -36 to +20 using 8-bit 1's complement form.

$+36 \rightarrow 00100100$

$-36 \rightarrow 11011011$

$+20 \quad 00010100$

$11101111 \longrightarrow -16$

$00010000$

$+20$

$-36$

$-16$

## 2's complement :-

In 2's complement subtraction, add the 2's complement of the subtrahend to the minuend. If there is a carry out, ~~bring~~ ignore it. If the msb is 0, the result is positive and is in true binary. If the msb is '1' the result is negative and is in its 2's complement form.

Example 1 :-

Subtract 20 from 36 using the 8-bit 2's complement form.

36 - 20.

→ Take subtrahend. 20.

20 → 00010100

1's complement → 11101011

2's complement → 11101100

36 → 00100100

20 → 11101100 (2's complement form of 20).

1]00010000 (ignore the carry).

Example 2 :-

Add -36 to +20 using the 8-bit 2's complement form.

+36 ⟶ 00100100

− 11011011 (1's complement).

+1

-36 ⟶ 11011100 (2's complement).

20 → 00010100

-36 → 11011100

-16 = 11110000

msb is 1, then result is negative.

11110000

00001111

+1

00010000 (+16)

Example 3 :-

Add -45.75 to +87.5 using the 12-bit 2's complement arithmetic.

+87.5 → 0101 0111 . 1000

+45.75 → 00101101 . 1100

1101 0010 . 0011 (1's complement form)

1 1 1

1101 0010 . 0100 (2's complement form)

128 64 32 16 8 4 2 1

0 0 1 0 1 0 0 1

+87.5 → 0101 0111 . 1000

1101 0010 . 0100

1 1 1 1 1 1 1 1

1] 0010 1001 . 1100 (ignore the carry.

↓ ↓

41 . 75

Example 4 :-

Add -45.75 to -87.5 using the 12-bit 2's complement form.

+87.5 → 0101 0111 . 1000

(1's comple) 1010 1000 . 0111

(add 1) 1 1 1 1

(2's comp). 1010 1000 . 1000

-87.5

+45.75 → 00101101 . 1100

(1's comp) 1101 0010 . 0011

1

1 1

(2's comp). 1101 0010 . 0100

-45.75

-87.5. 1010 1000 . 1000

-45.75 1101 0010 . 0100

1] 0111 1010 . 1100 (ignore the carry).

1000 0101 . 0011

1

1 1 1

1000 101 . 0100

## 9's Complement:-

In 9's complement subtraction.

→ find the 9's complement of the subtrahend.
→ Add 9's complement of subtrahend to the minuend.
→ if there is a carry it indicates that the answer is positive. Add the carry to the LSD of this result to get answer.
→ If there is no carry, it indicates that the answer is negative and the result obtained is its 9's complement.
→ Find the 9's complement of the following decimal numbers.

→ 4986
9999
4986
5013

→ 738.65
999.99
738.65
261.34

→ 4526.075
9999.999
4526.075
5473.924

## 9's complement method of subtraction:-

→ 745.86 - 436.62

Step 1:- 999.99
436.62
563.37

Step 2:-
745.86
563.37
1) 309.23
1
309.24

The carry indicates the answer is positive.

+309.24.

436.62 – 745.86.

Step1:- 999.99
745.86
254.13

Step2:- 436.62
254.13
690.75

Step3:- If there is no carry. the answer is negative.

Step4:- 999.99
690.75
309.24

answer is -309.24.

10's complement:-

The 10's complement of a decimal number is obtained by adding a 1 to its 9's complement.

Find the 10's complement of the following decimal number.

→ 4986
9999
4986
5013
1
5014

→ 738.65.
999.99
738.65
261.34
1
261.35

→ 4526.075
9999.999
4526.075
5473.924
1
5473.925

10's complement method of subtraction :-

→ 745.86 - 436.62

```
  999.99
  436.62
 -------
  563.37
       1
 -------
  563.38   (10's complement)
```

```
  745.86
  563.38
 -------
(1)309.24   (ignore the carry)
```

→ 436.62 - 745.86

```
  999.99
  745.86
 -------
  254.13
       1
 -------
  254.14
```

```
  436.62
  254.14
 -------
  690.76   (No carry, negative answer)
```

```
  999.99
  690.76
 -------
  309.23
       1
 -------
  309.24   - 309.24.
```

15's Complement methode :-

Find the 15's complement of the following decimal number.

→ 6A36

| 15 | 15 | 15 | 15 |
|---|---|---|---|
| 6 | A | 3 | 6 |
| 9 | 5 | C | 9 |

→ 9AD.3A

| 15 | 15 | 15 | 15 | 15 |
|---|---|---|---|---|
| 9 | A | D | . 3 | A |
| 6 | 5 | 2 | . C | 5 |

15's complement methode of subtraction :-

69B - C14

→

| 15 | 15 | 15 | . |
|---|---|---|---|
| C | 1 | 4 | |
| 3 | E | B | . |

→

| 6 | 9 | B |
|---|---|---|
| 3 | E | B |
| A | 8 | 6 |

→ No carry, it indicates answer is negative.

| 15 | 15 | 15 |
|---|---|---|
| A | 8 | 6 |
| - | 5 | 79 |

$69B_{16} - C14_{16} = -579_{16}$

F - 15
10 - 16
11 - 17
12 - 18
13 - 19
14 - 20
15 - 21
16 - 22
17 - 23
18 - 1E
19 - 1F

20
21
22

16 | 22 | 6
1 | 6 ↑
16

16's complement methode :-

Find the 16's complement of the following decimal number.

→ A8C

```
  15  15  15
 - A   8   C
-------------
  5   7   3   — (15's complement)
  0   0   1   (add 1)
-------------
  5   7   4   (16's complement).
-------------
```

16's complement methode of subtraction :-

C9B − C14

```
  15  15  15
   C   1   4
-------------
   3   E   B   (15's complement).
           1
-------------
   3   E   C   (16's complement).
-------------

   1   1
   C   9   B
   3   E   C
-------------
1  0   8   7   (ignore it).
-------------
```

if carry present, the answer is positive.

C9B − C14 → $(087)_{16}$.

## 7's complement methode :-

Find the 7's complement of the following decimal number.

→ 536

777
536
241 (7's complement).

## 7's complement methode of subtraction :-

623 - 352.

777
352
425 (7's complement).

623 → 623
-352 → 425
1250
↳ 1 (add carry)
251

7 - 7
8 - 10
9 - 11
10 - 12
11 - 13
12 - 14
13 - 15

352 - 623

777
623
154.

352
154
526 (No carry, answer is negative

777
526
-251

## 8's complement methode

Find the 8's complement of the following decimal number

→ 536.

```
  777
  536   (
  241   (7's complement)
   +1   (add 1)
  242   (8's complement)
```

## 8's complement methode of subtraction :-

623 - 352

```
  777
  352
  425   (7's complement)
    1   (add 1)
  426   (8's complement)
```

```
  623
  426
  251   (ignore the carry).
```

623
−426
1049

if carry present, the answer is positive.

$623 - 352 = (251)_8$

## Number Representation in binary :-

There two types of numbers

→ unsigned numbers

→ signed numbers.

The numbers without positive or negative signs are known as unsigned numbers. The unsigned numbers are always positive numbers. (considered).

In signed number system, the number may be positive or negative. Different formats are used for representation of signed binary numbers. They are.

→ sign - magnitude representation

→ 1's complement representation.

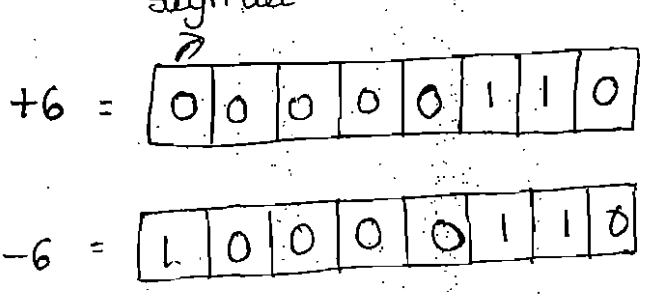
→ 2's complement representation.

## Sign - magnitude Representation :-

In sign - magnitude representation the MSB represents the sign and the remaining bits represents the magnitude. The MSB bit is 1, it indicates the sign is negative, and MSB bit is 0, it indicates the sign is positive.

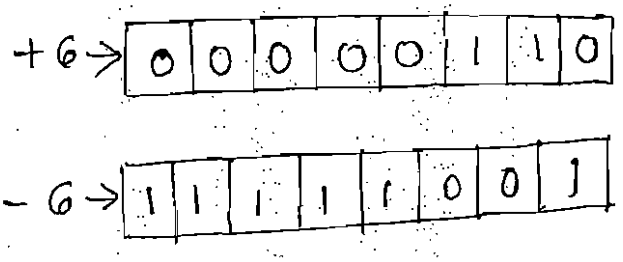
In eight bit representation, the MSB indicates the sign. and remaining seven bits represent the magnitude.

Consider the number +6 and -6 represented in binary.




## 1's Complement Representation :-

In the 1's complement representation, the positive numbers remain uncharged. 1's complement representation of negative number can be obtained by the 1's complement of the binary number.

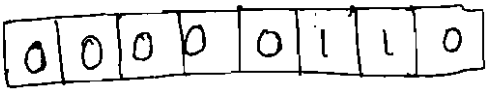


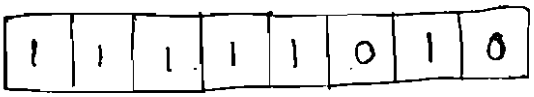

## 2's Complement Representation :-

In the 2's complement representation, the positive numbers remain uncharged, 2's complement representation of negative number can be obtained by

→ find the 1's complement of the number (by replacing 0 by 1 and 1 by 0)

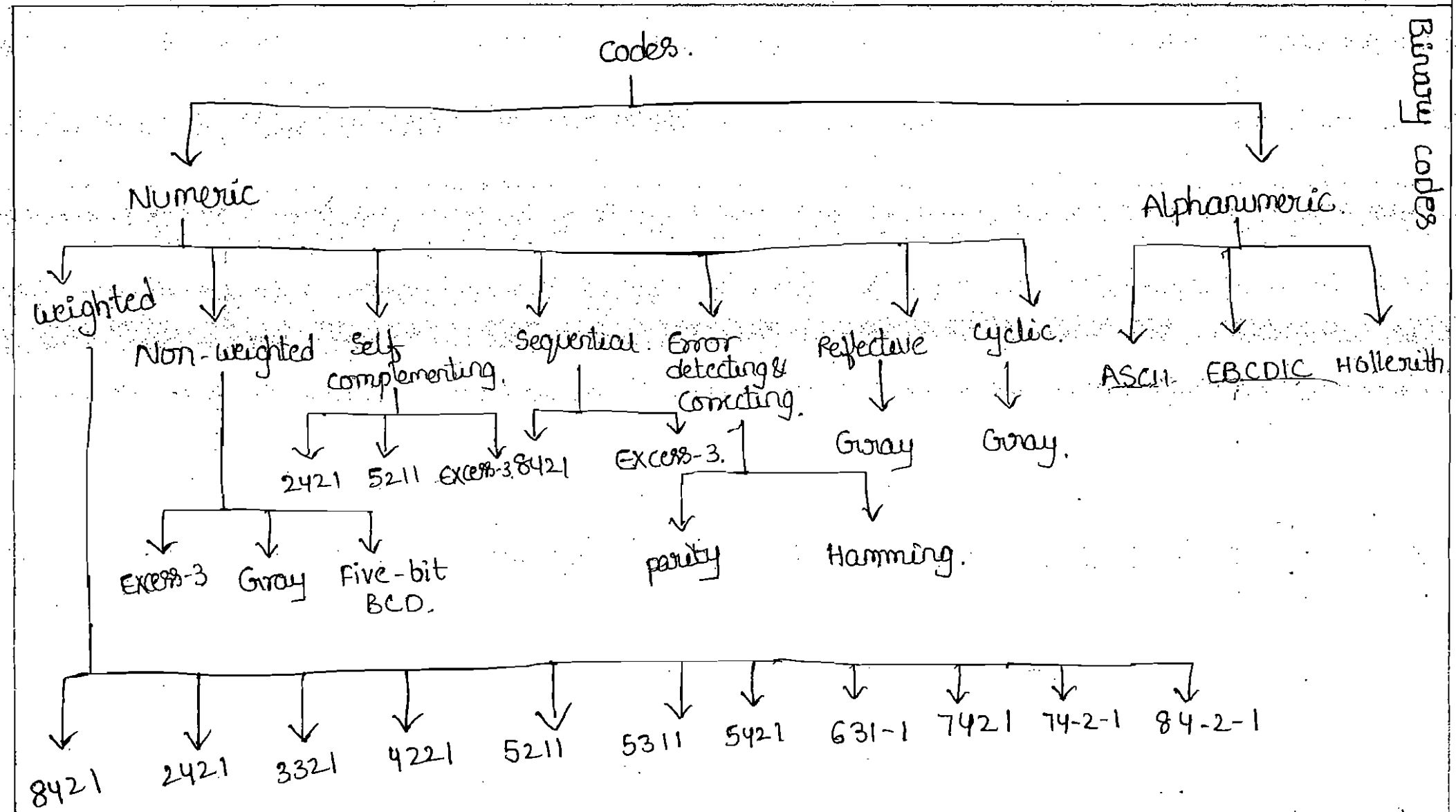
→ find the 2's complement of the number by adding 1 to 1's complement of the number.

+6 → 


-6 → 


| Decimal | Signed magnitude | Signed - 1's Complement | Signed - 2's Complement |
|---|---|---|---|
| +7 | 0111 | 0111 | 0111 |
| +6 | 0110 | 0110 | 0110 |
| +5 | 0101 | 0101 | 0101 |
| +4 | 0100 | 0100 | 0100 |
| +3 | 0011 | 0011 | 0011 |
| +2 | 0010 | 0010 | 0010 |
| +1 | 0001 | 0001 | 0001 |
| +0 | 0000 | 0000 | 0000 |
| -0 | 1000 | 1111 | - |
| -1 | 1001 | 1110 | 1111 |
| -2 | 1010 | 1101 | 1110 |
| -3 | 1011 | 1100 | 1101 |
| -4 | 1100 | 1011 | 1100 |
| -5 | 1101 | 1010 | 1011 |
| -6 | 1110 | 1001 | 1010 |
| -7 | 1111 | 1000 | 1001 |
| -8 | - | - | 1000 |

1001
1110



→ Numeric codes

Numeric codes are codes which represent numeric information that is only numbers as a series of 0s and 1s. Numeric codes used to represent the decimal digits are called Binary Coded decimal (BCD).

8421, 2421, 5211 are BCD codes.

8421, XS-3, Gray code are numeric codes.

Alphanumeric codes

Alphanumeric codes are binary codes which represent alphanumeric data. This code includes alphanumeric data, letters of the Alphabet, numbers, mathematical symbols and punctuation marks. ASCII and EBCDIC are commonly used Alphanumeric codes.

ASCII (American Standard code for Information Interchange).

EBCDIC (Extended Binary coded decimal Interchange code).

Alphanumeric codes are used to interface Input - output devices such as keyboards, printers, VDUs.

→ weighted and non-weighted codes:-

The weighted codes are those which obey the position - weighting principle. Each position of the number represents a specific weight.

Ex:- 8421, 2421, 84-2-1.

In decimal code, if number is 637 then weight of 6 is 100, weight of 3 is 10, and weight of 7 is 1

Non-weighted codes are codes which are not assigned with any weight to each digit position.

EX:- XS-3 (Excess-3) and Gray code.

→ Error detecting and correcting codes:-

Codes which allows only error detection are called error detecting codes. Codes which allows error detection and correction are called error-detec error correction codes. Error detection and correction involves the addition of extra bits, called check bits, to the transmitted data. These extra bits allows the detection and some time correction of errors in data.

EX:- parity and Hamming codes.

→ Self-complementing codes:-

A code is said to be a self-complementing if the code word of the 9's complement of N, of 9-N can be obtained by the 1's complement of the word of N. Therefore in a self complementing code, the code word for 9 is the complement for the code 0, 8 for 1, 7 for 2, 6 for 3 5 for 4.

Excess-3 code is example of self-complementary code.

→ Sequential codes:-

In sequential codes, each succeeding code is one binary number greater than its preceding code.

EX:- 8421 and excess-3 codes.

→ Cyclic codes :-

Cyclic codes are also called unit distance codes. The name itself indicates that there is unit distance between two consecutive codes. The unit distance codes have special advantages in that they minimize transitional errors or flashing.

EX:- Gray code.

→ Reflective codes :-

A Reflective code is a binary code in which the n least significant bits for code words $2^n$ through $2^{n+1}-1$ are the mirror images of those for 0 through $2^n-1$.

EX:- Gray code.

Binary coded decimal (BCD) code or 8421 code :-

In this code, each decimal digit, 0 through 9, is coded by a 4-bit binary number. it is also called the natural binary code. For example, the decimal number 15 can be represented as 1111 in binary but in BCD.

00010101

It is useful for mathematical operations. The main advantages of this code is its ease of conversion to and from decimal.

disadvantage of the BCD code is that, arithmetic operations are more complex and it requires more bits

## BCD addition :-

→ Convert the decimal numbers into their equivalent BCD codes.

→ Add the two BCD numbers, using the basic rules for Binary addition.

→ Check the result If sum is equal to or less than 9, it is a valid BCD number.

→ If the result is greater than 9 or if a carry generated from the 4-bit sum, the sum is invalid.

→ To correct the sum, add (0110) 6 to the sum term of that Group.

→ If carry results when 6 is added, simply add the carry to the next 4-bit Group.

Example :-

23 + 13.

$$\begin{array}{rl} 23 \rightarrow & 0010\ 0011 \\ 13 \rightarrow & 0001\ 0011 \\ \hline 36 & 0011\ 0110 \end{array}$$

683.5 + 256.2.

$$\begin{array}{rl} 683.5 \rightarrow & 0110\ 1000\ 0011.0101 \\ 256.2 \rightarrow & 0010\ 0101\ 0110.0010 \\ \hline 939.7 & 1000\ 1101\ 1001.0111 \\ & \quad\ 0110 \\ \hline & 1000\ 0011\ 1001.0111 \\ & \quad\ \curvearrowright \\ \hline & 1001\ 0011\ 1001.0111 \end{array}$$

## BCD Subtraction :-

→ Each 4-bit Group of subtrahend from the corresponding 4-bit Group of the minuend.

→ if there is no borrow from the next higher Group then no correction.

→ if there is a borrow from the next group, then 6(0110) is subtracted from the difference term of group.

Example :-

23 - 13

23 → 0010 0011

13 → 0001 0011

10 ← 0000 0000

1 0

236.8 - 142.5

236.8 → 0010 0011 0110. 1000

142.5 → 0001 0100 0010. 0101

94.3 0000 1111 0100.0011

0110

0000 1001 0100. 0011

0 9 4 . 3

BCD Subtraction using 9's complement :-

236.8 - 142.5

999.9 236.8 → 0010 0011 0110. 1000

142.5 857.4 → 1000 0101 0111. 0100

857.4 629.4

1010 1000 1101. 1100

0110 0110

1010 1001 0100. 0010

0110

1 0000 1001 0100. 0010

1

0000 1001 0100. 0011

0 9 4 . 3

## BCD Subtraction using 10's Complement:-

1) 236.8 - 142.5

```
  999.9
  142.5
 -------
  857.4
      1
 -------
  857.5
```

```
236.8 → 0010 0011 0110. 1000
857.5 → 1000 0101 0111 . 0101
        ---- ---- ---- ----
        1010 1000 1101 . 1101
        0110      0110 . 0110
        ---- ---- ---- ----
   1 ]  0000 1001 0100 . 0011
        ---- ---- ---- ----
          0    9    4 . 3
```

In 10's complement ignore the carry.

```
  236.8
  142.5
 -------
  094.3
 -------
```

## Excess Three Code (XS-3):-

The excess-3 code for a given decimal number is determined by adding 3 (0011) to each decimal digit in the Given number. It is a non-weighted BCD code, sequential and self-complementing code

It can be used for arithmetic operations.

XS-3 Addition:-

23+13

23 → 56 → 0101 0110

13 → 46 → 0100 0110

1001 1100

-0011 -0011

0110 1001 (answer in XS-3)

| 23 | 13 |
|---|---|
| 33 | 33 |
| 56 | 46 |

If carry generated add +0011

If carry not generated subtract -0011

→ 236.8 + 142.5

236.8 → 569.B → 0101 0110 1001 . 1011

142.5 → 475.8 → 0100 0111 0101 . 1000

1001 1101 1110 ① 0011

1001 1101 1111 0011

~~(answer in XS-3)~~

1001 1101 1111 0011

-0011 -0011 -0011 +0011

0110 1010 1100 . 0110 (answer in XS-3)

3 7 9 . 3

## XS-3 Subtraction :-

→ if borrow generated subtract 3 (0011)

→ if borrow not generated add 3 (0011).

→ 25.3 - 16.5.

25.3 → 58.6 → 0101 1000. 0110

16.5 → 49.8 → 0100 1001 . 1000

| | | |
|---|---|---|
| 0000 | 1110 | 1110 |
| +0011 | -0011 | -0011 |
| 0011 | 1011 | 1011 |

Answer in XS-3.

| | | |
|---|---|---|
| 3 | B | B |
| -3 | -3 | -3 |
| 0 | 8 . | 8 |

→ 267 - 175

267 → 5 9 10 → 0101 1001 1010

175 → 4 10 8 → 0100 1010 1000

| | | |
|---|---|---|
| 0000 | 1111 | 0010 |
| +0011 | -0011 | +0011 |
| 0011 | 1100. | 0101 |

Answer in XS-3).

| | | |
|---|---|---|
| 3 | C . | 5 |
| -3 | -3 | -3 |
| 0 | 9 . | 2 |

## XS-3 Subtraction using 9's complement

→ 687 - 348

348 → 999
348
651 (9's complement form).
333
984 (XS-3 form).

687 → 1001 1011 1010 (XS-3 form of 687)
984 → 1001 1000 0100

I] 0000 0011 1110
↳ 1

I) 0001 0011 1110
↳ 1

0001 0011 1111
+ 0011 + 0011 - 0011

0110 0110 1100 (answer in XS-3).

6 6 12
-3 -3 -3

3 3 9.

If borrow generated subtract 3 (0011)
if borrow not generated add 3 (0011).

XS-3 Subtraction using 10's complement :-

→ 687 - 348

```
  999
  348
 ---
  651   (9's complement form)
    1   (+add'1)
 ---
  652   (10's complement form)
```

```
  652
 +333
 ---
  985   (XS-3 form).
```

687 → XS-3 → 1001 1011 1010

1001 1000 0101

1 0010 0011 1111

(1) (A)

1 0011 0011 1111

↳ (ignore the carry)

0011 0011 1111

+0011 +0011 -0011

1 1

0110 0110 1100

answer in XS-3.

-3 -3 -3

3 3 9.

## Gray code :-

Gray code is a 4-bit numeric code. It is a unit distance code because successive code words in this code differ in one bit position only. It is also a reflective code, it is both reflective and unit distance.

| Decimal digit | ~~Gray~~ Binary Code | Decimal digit | Gray code |
|---|---|---|---|
| 0 | 0000 | 8 | 1100 |
| 1 | 0001 | 9 | 1101 |
| 2 | 0011 | A | 1111 |
| 3 | 0010 | B | 1110 |
| 4 | 0110 | C | 1010 |
| 5 | 0111 | D | 1011 |
| 6 | 0101 | E | 1001 |
| 7 | 0100 | F | 1000 |

## Binary to Gray code conversion :-

→ Record the MSB of the binary as the MSB of the Gray code.
→ Add the MSB of the binary to the next bit in binary ignore the carry
→ Add the 2nd bit of binary to the 3rd bit of the binary, the 3rd bit to the 4th bit

Convert binary 0110 to the Gray code.

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

Convert binary 1001 to the Gray code.

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

## Gray to Binary conversion :-

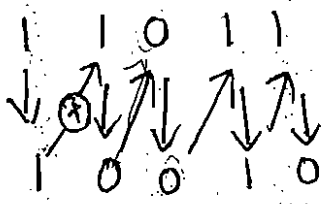
→ The MSB of the binary number is the same as the MSB of the Gray code

→ Add the MSB of the binary to the next significant bit of the Gray code, ignore the carry.

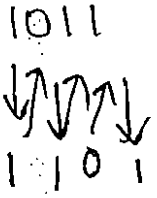
→ Add the 2nd bit of the binary to the 3rd bit of the Gray; the 3rd bit of the binary to the 4th bit of the Gray code.

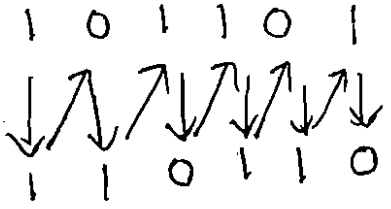
→ convert Gray to Binary.

11011




1011




## Applications of Gray code :-

→ Gray code is used in the transmission of digital signals as it minimizes the occurrence of errors.

→ The Gray code is better than the binary code in angle measuring devices.

→ The Gray code is used for labelling the axes of Karnaugh maps.

→ The use of Gray codes to address program memory in computers minimizes power consumption.

→ Another application of Gray code is position indication in rotating disk.

## Logic Gates:-

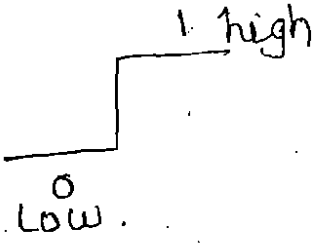
The logic Gates are the fundamental blocks of digital systems. The logic Gate is ability to make decisions (output). logic gate are electronic circuits because they are made up of a number of electronic devices and components.

Inputs and outputs of logic Gates can occur only in two levels.

### Positive Logic:-

high → 1

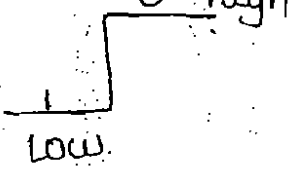
Low → 0.




### negative Logic

Low → 1

high → 0




### mixed Logic:-

mixed logics provides a simplified mechanism for the analysis and design of digital circuits. In mixed logic, the assignment of logical values to voltage values is not fixed, and it can be decided by the logic designers.

### Logic design:-

The interconnection of gates to perform a variety of logical operations is called logic design.

### Truth table:-

A table which lists all the possible combinations of input variables and the corresponding outputs is called a truth table.

## Types of logic gates:-

1. AND Gate
2. OR Gate
3. NOT Gate

} Basic Gates.

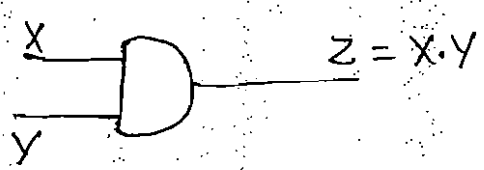
4. NAND Gate
5. NOR Gate

} universal Gates.

6. EXCLUSIVE OR Gate
7. EXCLUSIVE NOR Gate.

### AND Gate:-

An AND gate is a logic circuit with two or more input and one output that performs ANDing operation. The output of an AND gate is high only when all of its inputs are in the high state. In all other conditions the output is low.




Logic symbol.

Truth table

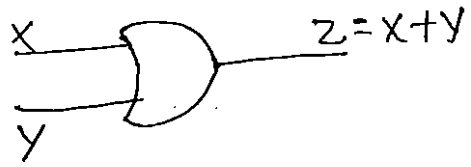
| X | Y | Z=X.Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

### OR Gate:-

An OR gate is a logic circuit with two or more inputs and one output that performs ORing operation. The output of an OR gate is high when any one of the input is high state. In all other conditions the output is low.

~~NOT Gate~~ :-

Symbol




Truth table.

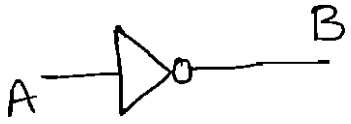
| X | Y | Z=X+Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

NOT Gate :-

A NOT gate is also called an inverter is a single input, single output logic circuit whose output is always the complement of the input. That is a low input produces a high output, high input produces a low output.

Truth table.

| A | B |
|---|---|
| 0 | 1 |
| 1 | 0 |



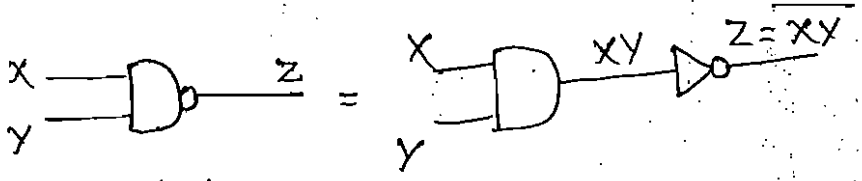

Symbol

NAND gate :-

A NAND gate is equivalent to AND gate followed by a NOT gate: The output of NAND gate is low when all inputs are in high state. In all other conditions the output is high.

Truth table.

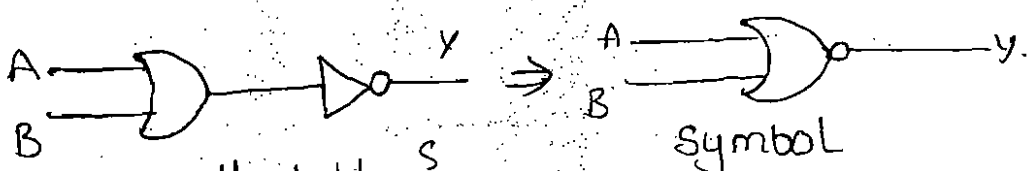
| X | Y | $Z = \overline{XY}$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |




Symbol

NOR Gate :-

The term NOR implies OR Gate followed by a NOT Gate. The output of a NOR Gate is Logic 1 when all the inputs are logic '0'. remaining all other conditions the output is Logic '0'.




Truth table

Symbol

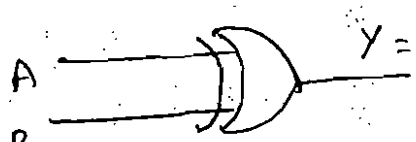
| A | B | Y |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

$Y = \overline{A+B}$

EX-OR Gate :- (Exclusive -OR Gate)

The output of an EX-OR Gate is a Logic 1 when the two inputs are different logic and Logic '0' when the two inputs are at the same logic.

Truth table $Y = \bar{A}B + A\bar{B}$



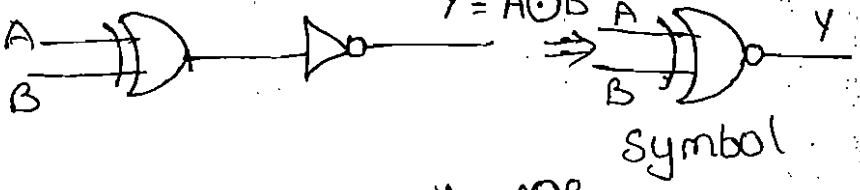

Symbol.

| A | B | Y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

EX-NOR Gate :- (EXclusive -NOR)

The output of an EX-NOR Gate is a Logic 1 when the two inputs are same and Logic 0 when the two inputs are different. EX-NOR Gate is EX-OR

Gate followed by NOT Gate




Symbol

$Y = A \odot B$.

Truth table

| A | B | Y |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Error-detecting codes :-

When the digital information in the binary form is transmitted from one system to another system an error may occur.

This means a signal '0' may change to '1' or '1' change to '0'.

Parity bit :-

To maintain data integrity between transmitter and receiver, extra bit or more than one bit are added in the data. These extra bits allow the detection and some times correction of error in the data. Those extra bits are called parity bits.

→ There are two types of parity - odd and even parity.

→ For odd parity, the parity bit is set to a 0 or 1 at the transmitter such that the total number of '1' bits in the word including the parity bit is an odd number.

→ For even parity, the parity bit is set to a 0 or 1 at the transmitter such that the total number of '1' bits in the word including the parity bit is an even number.

| decimal. | Binary | odd parity | even parity. |
|---|---|---|---|
| 0 | 0000 | 1 | 0 |
| 1 | 0001 | 0 | 1 |
| 2 | 0010 | 0 | 1 |
| 3 | 0011 | 1 | 0 |
| 4 | 0100 | 0 | 1 |
| 5 | 0101 | 1 | 0 |
| 6 | 0110 | 1 | 0 |
| 7 | 0111 | 0 | 1 |

⇒ In an even - parity scheme, which of the following words contain an error.

(a) 10110110
The no. of 1's in the word is odd (5). So, there is an error.

(b) 11011011
The no. of 1's in the word is even (6), so, there is no error.

(c) 10101000
The no. of 1's in the word is odd (3), so there is an error.

⇒ In an odd - parity scheme, which of the following words contain an error.

(a). 11011101
The no. of 1's in the word is even (6). So, there is an error.

(b) 11011010
The no. of 1's in the word is odd (5). So there is no error.

(c) 11011000
The no. of 1's in the word is even (4), so there is ~~no error~~ an error.

## Hamming code :- (Error - correcting codes).

A code is said to be an error - correcting code, which allow error detection and correction are called error detecting and correcting codes.

→ Hamming code not only provides the detection of a bit error, but also identifies which bit is in error.

→ The code uses a number of parity bits located at certain positions in the code group.

### 1-bit Hamming code :-

To transmit four data bits, three parity bits located at positions $2^0$, $2^1$ and $2^2$ from left are added to make a 7-bit code word which is then transmitted. The word format is.

$P_1$ $P_2$ $D_3$ $P_4$ $D_5$ $D_6$ $D_7$.

D for the data bits

P for the parity bits.

$P_1$ is the to be set to a '0' or a '1' so that it establishes even parity over bits 1,3,5 and 7 ($P_1, D_3, D_5, D_7$)

$P_2$ is to be set to a '0' or a '1' so that it establishes even parity over bits 2,3,6,7 ($P_2, D_3, D_6, D_7$)

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

At the receiving end, the message received in the Hamming code is decoded to see if any errors have occured.

Bits 1,3,5,7, bits 2,3,6,7, and bits 4,5,6,7 are all checked for even parity.

→ If they all check out, there is no error.

→ if there is an error, the error bit can be located by forming a 3-bit binary number.

$$C_1 = P_1 \oplus D_3 \oplus D_5 \oplus D_7.$$

$$C_2 = P_2 \oplus D_3 \oplus D_5 \oplus D_7$$

$$C_3 = P_4 \oplus D_5 \oplus D_6 \oplus D_7.$$

EX:- ~~1010~~ Encode data bits 1010 into the 7-bit even-parity hamming code.

The bit pattern.

| $P_1$ | $P_2$ | $D_3$ | $P_4$ | $D_5$ | $D_6$ | $D_7$ |
|---|---|---|---|---|---|---|
| | | 1 | | 0 | 1 | 0 |

Bits 1,3,5,7 ($P_1$ 1 1 1) must have even parity. So $P_1$ must be a '1'

Bits 2,3,6,7 ($P_2$ 1 1 0) must have even parity. So $P_2$ must be a '0'

Bits 4,5,6,7 ($P_4$ 0 1 0) must have even parity. So $P_4$ must be a '1'.

The final code with parity bits is

| $P_1$ | $P_2$ | $D_3$ | $P_4$ | $D_5$ | $D_6$ | $D_7$ |
|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 | 1 | 0 |

→ This data transmitted through a noisy channel.

The code is

1 0 1 0 0 1 0 (error occured).

$P_1$ $P_2$ $D_3$ $P_4$ $D_5$ $D_6$ $D_7$

To correct the code by using.

$C_1 = 1 \oplus 1 \oplus 0 \oplus 0$ = 0 put a '0' in the 1's position.

$C_2 = 0 \oplus 1 \oplus 1 \oplus 0$ = 0 put a '0' in the 2's position.

$C_3 = 0 \oplus 0 \oplus 1 \oplus 0$ = 1 put a '1' in the 4's position.

error in the 4th position.

1011010

12-bit hamming code :-

To transmit eight data bits, four parity bits located at positions $2^0, 2^1, 2^2$ and $2^3$ from left are added to make a 12-bit code word which is then transmitted.

$P_1\ P_2\ D_3\ P_4\ D_5\ D_6\ D_7\ P_8\ D_9\ D_{10}\ D_{11}\ D_{12}$

$P_1$ is set to a '0' or '1' ($P_1, D_3, D_5, D_7, D_9, D_{11}$).

Similarly. $P_2$ ($P_2, D_3, D_6, D_7, D_{10}, D_{11}$).

$P_4$ ($P_4, D_5, D_6, D_7, D_{12}$)

$P_8$ ($P_8, D_9, D_{10}, D_{11}, D_{12}$).

Example:- given - 01011010, generate the 12-bit code.

| $P_1$ | $P_2$ | $D_3$ | $P_4$ | $D_5$ | $D_6$ | $D_7$ | $P_8$ | $D_9$ | $D_{10}$ | $D_{11}$ | $D_{12}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 0 | | 1 | 0 | 1 | | 1 | 0 | 1 | 0 |

$P_1 (P_1, 0, 1, 1, 1, 1)$ even parity, so $= P_1 = 0$.

$P_2 (P_2, 0, 0, 1, 0, 1)$ even parity, so $= P_2 = 0$

$P_4 (P_4, 1, 0, 1, 0)$ even parity, so $= P_4 = 0$

$P_8 (P_8, 1, 0, 1, 0)$ even parity, so $= P_8 = 0$

000010101010 (final data).

## 15-bit hamming code :-

To transmit eleven data bits, four parity bits located at positions $2^0, 2^1, 2^2$ and $2^3$ from left are added to make a 15-bit code word which is then transmitted.

$P_1\ P_2\ D_3\ P_4\ D_5\ D_6\ D_7\ P_8\ D_9\ D_{10}\ D_{11}\ D_{12}\ D_{13}\ D_{14}\ D_{15}$

$P_1$ is set to be '0' or '1' $(P_1, D_3, D_5, D_7, D_9, D_{11}, D_{13}, D_{15})$

$P_2$ is set to be '0' or '1' $(P_2, D_3, D_6, D_7, D_{10}, D_{11}, D_{14}, D_{15})$

$P_4$ is set to be '0' or '1' $(P_4, D_5, D_6, D_7, D_{12}, D_{13}, D_{14}, D_{15})$.

$P_8$ is set to be '0' or '1' $(P_8, D_9, D_{10}, D_{11}, D_{12}, D_{13}, D_{14}, D_{15})$

Example :-

11-bit group 01101110101 find out the final code.

| $P_1$ | $P_2$ | $D_3$ | $P_4$ | $D_5$ | $D_6$ | $D_7$ | $P_8$ | $D_9$ | $D_{10}$ | $D_{11}$ | $D_{12}$ | $D_{13}$ | $D_{14}$ | $D_{15}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 0 | | 1 | 1 | 0 | | 1 | 1 | 1 | 0 | 1 | 0 | 1 |

$P_1 (P_1, 0, 1, 0, 1, 1, 1, 1)$ even parity $P_1 = 1$

$P_2 (P_2, 0, 1, 0, 1, 1, 0, 1)$ even parity $P_2 = 0$

$P_4 (P_4, 1, 1, 0, 0, 1, 0, 1)$ even parity $P_4 = 0$

$P_8 (P_8, 1, 1, 1, 0, 1, 0, 1)$ even parity $P_8 = 1$

| $P_1$ | $P_2$ | $D_3$ | $P_4$ | $D_5$ | $D_6$ | $D_7$ | $P_8$ | $D_9$ | $D_{10}$ | $D_{11}$ | $D_{12}$ | $D_{13}$ | $D_{14}$ | $D_{15}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |

The final code is

100011011110101

## Standard sop and pos :-

### Sop (sum of products):-

product term → multiplying two or more variable is called product term. (EX:- ABC, $\bar{A}B\bar{C}$, AB, $AB\bar{C}\bar{D}E$)

sop → summiation of product term is called sop.

EX:- $AB+\bar{B}A+\bar{A}C$

It is also called the Disjunctive normal form (DNF).

### Standard sop :-

It is also called Disjunctive canonical form (DCF) It is also called the expanded sum of products form or canonical sum-of-products form. In this form, the function is the sum of a number of product terms where each product term contains all variables either in complemented or uncomplemented form.

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

### Sop to Standard SOP :-

→ write down all the terms

→ If one or more variables are missing in any term, expand that term by multiplying it with the sum of each one of the missing variable and its complement.

→ drop out the redundant term.

→ Replace the variables by 1's & 0's.
complemented variables by 0's
non-complemented variables by 1's.

* Expand $f(A,B) = \bar{A}B + B$.

The given expression is a two-variable function. In second term, the variable A is missing. So multiply it by $[A+\bar{A}]$.

$$\bar{A}B + B = \bar{A}B + B(A+\bar{A})$$
$$= \bar{A}B + AB + \bar{A}B \rightarrow \text{Drop out redundant}$$
$$= \bar{A}B + AB.$$
$$= 01 + 11 \quad \rightarrow \text{non-complemented} \rightarrow '1'$$
$$= m_1 + m_3. \quad \rightarrow \text{complemented} \rightarrow '0'.$$
$$= \Sigma m(1,3).$$

* $f(A,B,C) = AB\bar{C} + A\bar{B}C + AB + BC$
$$= AB\bar{C} + A\bar{B}C + AB(C+\bar{C}) + BC(A+\bar{A})$$
$$= AB\bar{C} + A\bar{B}C + ABC + AB\bar{C} + ABC + \bar{A}BC$$
$$= \bar{A}BC + ABC + A\bar{B}C$$
$$= 011 + 111 + 101$$
$$= m_3 + m_7 + m_5$$
$$= \Sigma m(3,5,7)$$

* $f(A,B,C) = A + AB + BCA.$
$$= A(B+\bar{B})(C+\bar{C}) + AB(C+\bar{C}) + BCA$$
$$= AB + A\bar{B}(C+\bar{C}) + ABC + AB\bar{C} + ABC.$$
$$= ABC + A\bar{B}C + A\bar{B}\bar{C} + AB\bar{C} + ABC + AB\bar{C} + ABC$$
$$= ABC + A\bar{B}C + A\bar{B}\bar{C} + AB\bar{C}$$
$$= 111 + 101 + 100 + 110 = m_7 + m_5 + m_4 + m_6$$
$$= \Sigma m(4,5,6,7).$$

POS (product of sum) :-

Sum term :- Sum of two or more variable is called sum term. (Ex :- (A+B), (A+B+C).

POS → product of sum terms is called POS.

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

It is also called conjunctive normal form (CNF).

Standard POS :-

It is also called conjunctive canonical for (CCF). It is also called the expanded product of sum form or canonical product of sum form. In this form, the function is product of a number of sum terms. where each sum term contains all variables either in complemented or uncomplemented form.

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

POS to standard POS :-

→ write down all ther terms.

→ If one or more variables are missing in any term expand that term by adding the product of each of the missing variable and its complement.

→ drop out the redundant one.

→ Replace the complemented variables by 1's and the non-complemented variables by 0's

* Expand $f(AB) = (\bar{A}+B)(A)$.

The given expression is a two-variable function. In second term, the variable B is missing. So add it by $(B \cdot \bar{B})$.

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

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

$$= (10)(11)(01)$$

$$= M_1, M_2, M_3$$

$$= \Pi M(1,2,3).$$

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

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

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

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

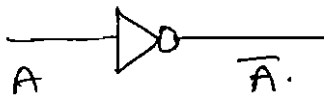
$$(000)\ (001)\ (010)\ (011)\ (100)\ (101)$$

$$= M_0, M_1, M_2, M_3, M_4, M_5.$$

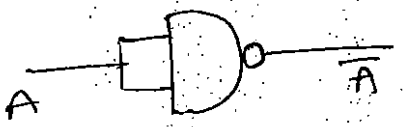
$$= \Pi M(0,1,2,3,4,5).$$

## NAND - NAND Realization :-

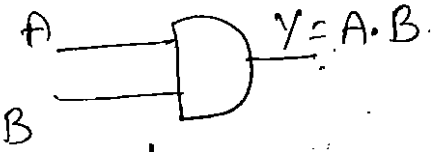
### 1) NOT Gate

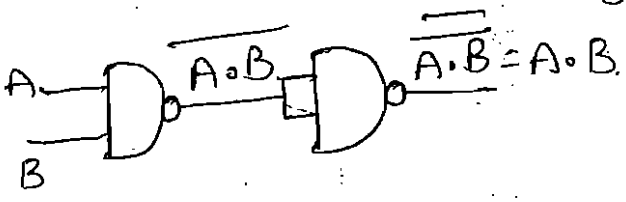



Using NAND gate.



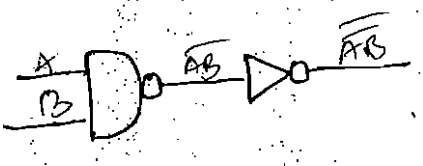

### 2) AND Gate.

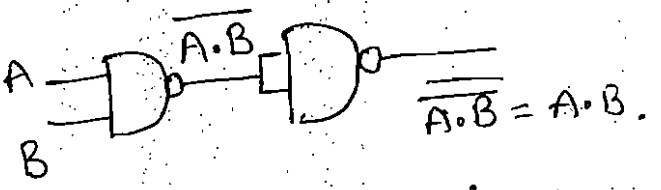


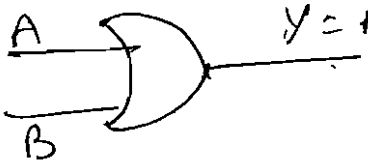

Step :- 1 add bubble on output


Step :- 2 add one NOT gate

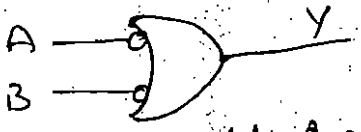


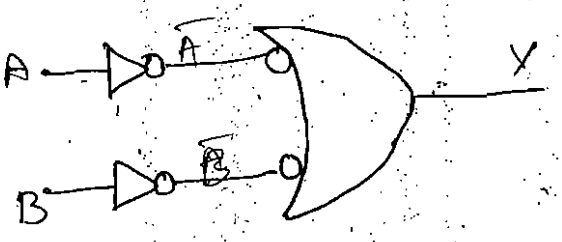

### 3) OR Gate

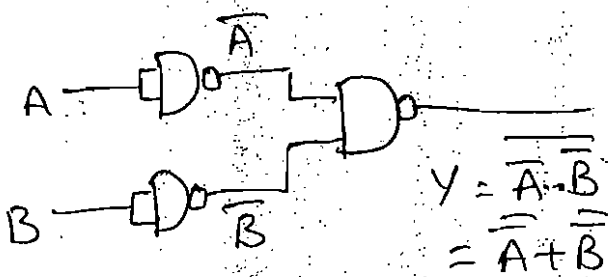



Step :- 1 add bubble on input




step 2 :- add not gate



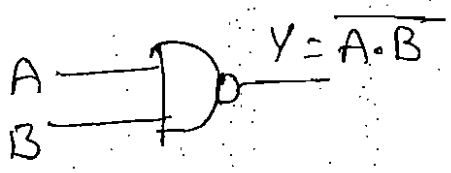




$Y = \overline{\overline{A} \cdot \overline{B}}$

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

$= A + B.$

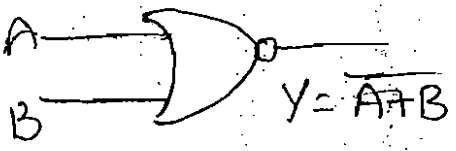
4) NAND gate

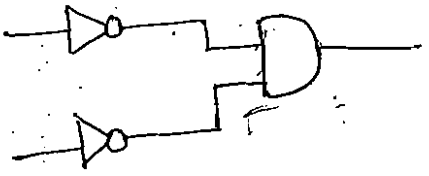



5) NOR Gate

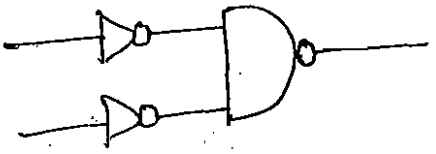
$Y = \overline{A+B}$.

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

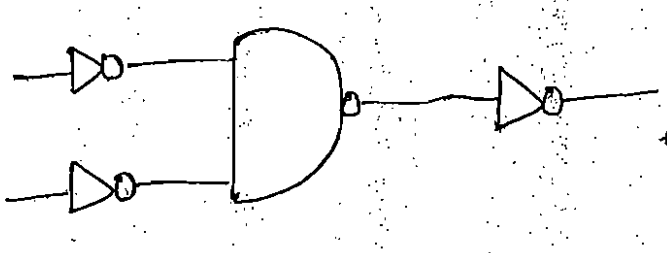


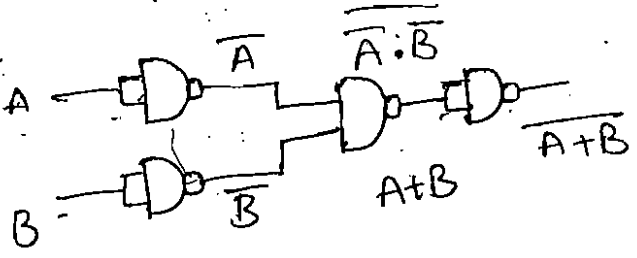
Step:-1



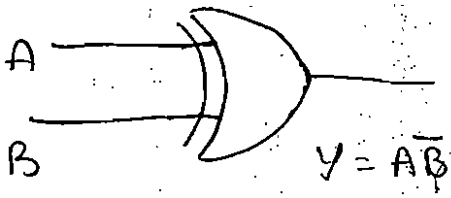
Step :- 2

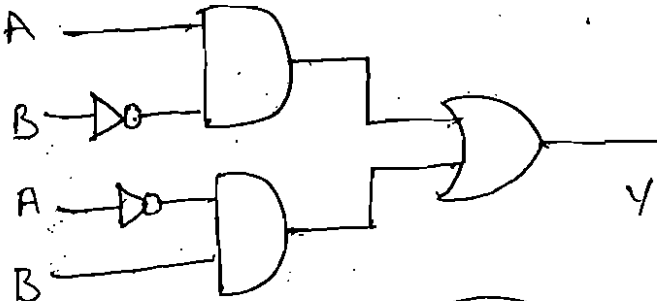


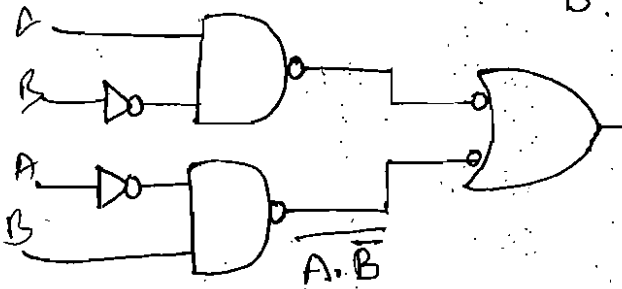
Step 3 :-

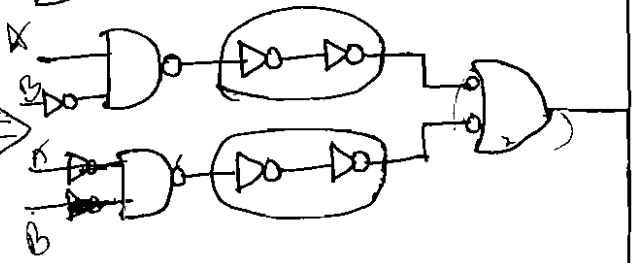


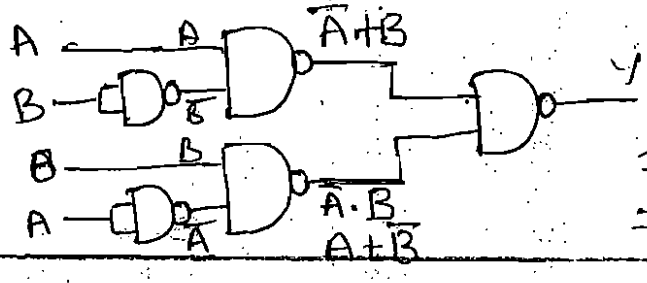

6) EX-OR Gate



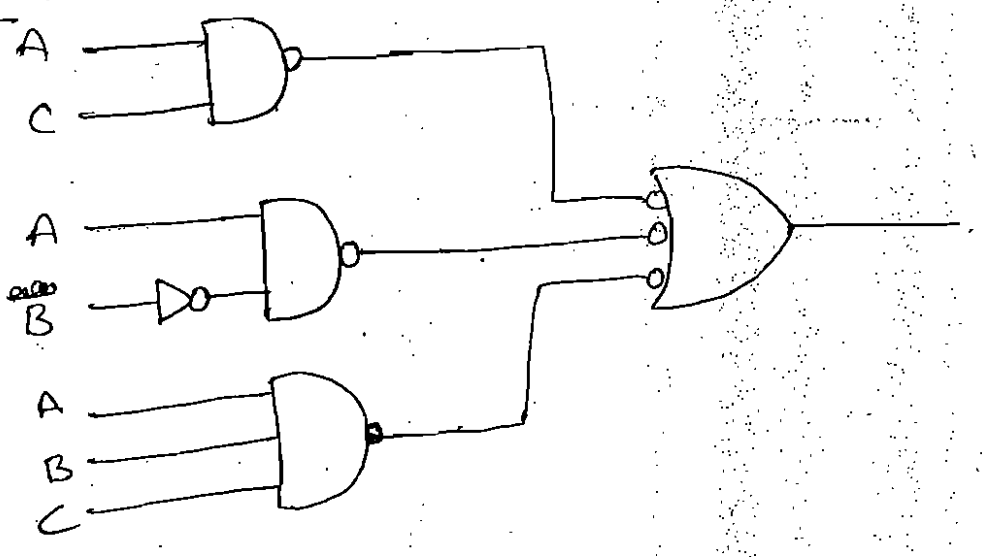




$Y = \overline{(\overline{A}+B)(A+\overline{B})}$

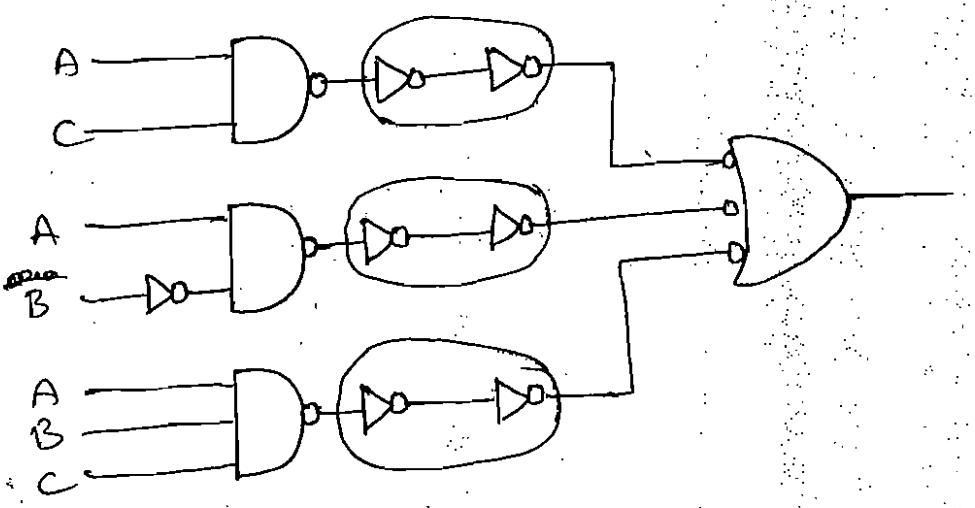
$= \overline{(\overline{A}+B)} + \overline{(A+\overline{B})}$

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

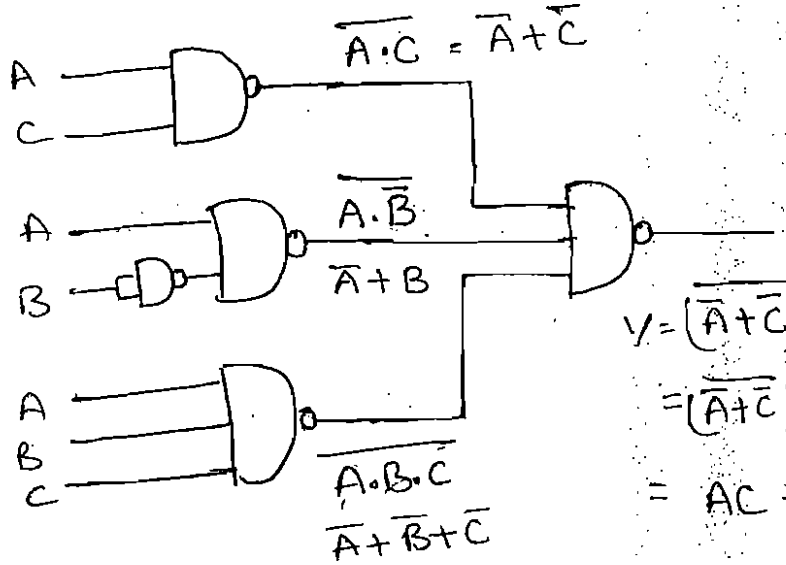
Step 1:-




Step 2:-




Step 3:-



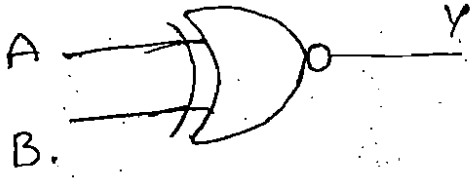

$$Y = \overline{(\overline{A}+\overline{C})(\overline{A}+B)(\overline{A}+\overline{B}+\overline{C})}$$

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

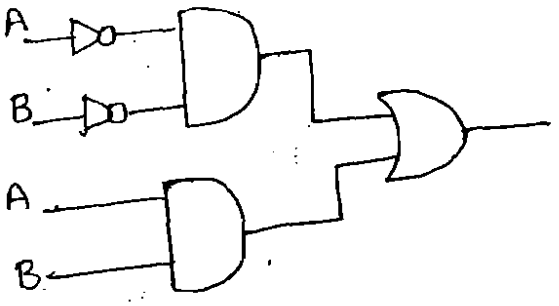
$$= AC + A\overline{B} + \overline{\overline{A}} \cdot \overline{\overline{B}} \cdot \overline{\overline{C}}$$

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

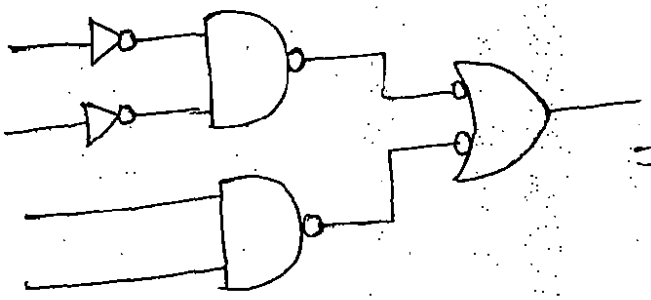
7) EX-NOR Gate :-

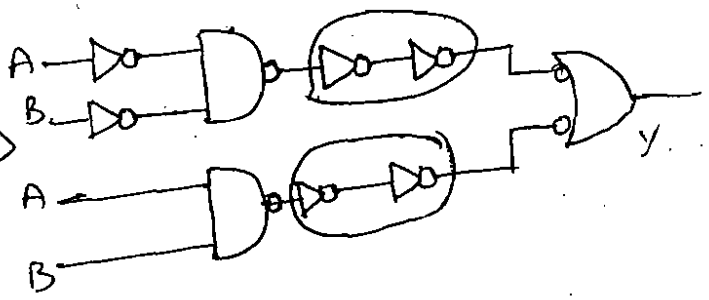


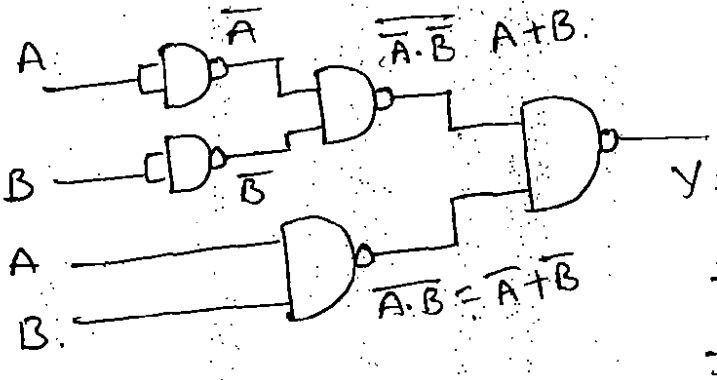

$Y = \overline{A}B + A\overline{B}$.




Step 1 :-





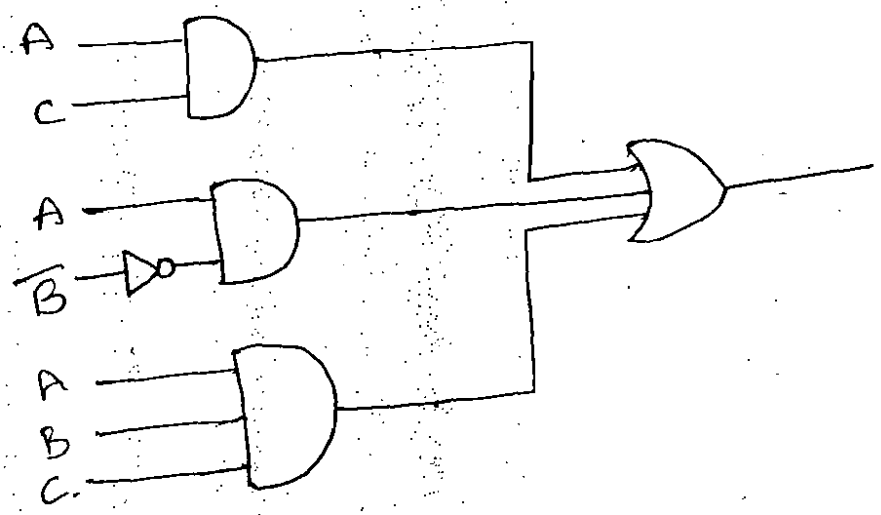

$Y = \overline{(A+B)(\overline{A}+\overline{B})}$

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

$= \overline{A}.\overline{B} + (AB)$

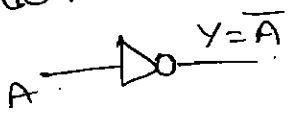
* Implement a given Boolean Expression by using NAND gate.

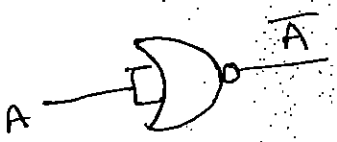
$Y = AC + A\overline{B} + ABC$.



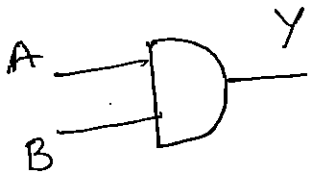
## NOR - NOR Realization :-

1) NOT Gate

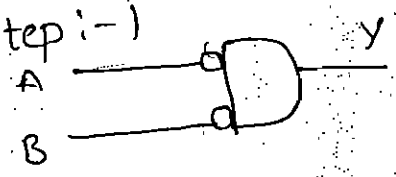


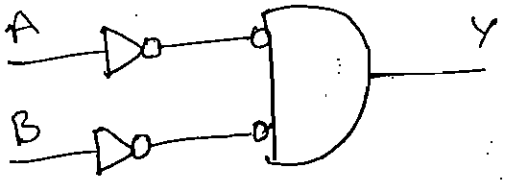

2) AND Gate.



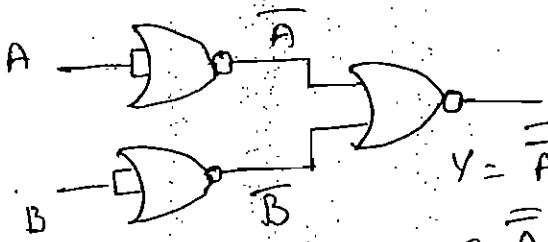

Step :-1




Step 2




Step 3

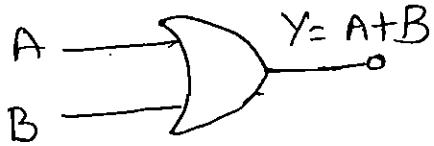



$Y = \overline{\bar{A} + \bar{B}}$

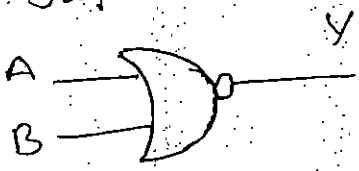
$= \bar{\bar{A}} \cdot \bar{\bar{B}}$

$= A \cdot B.$

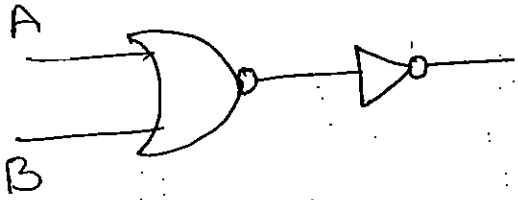
3) OR Gate

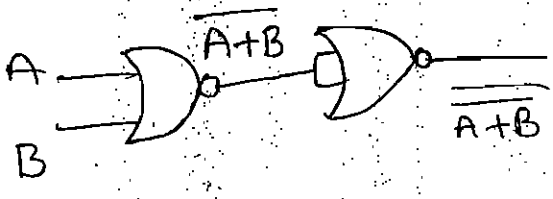



Step :-1



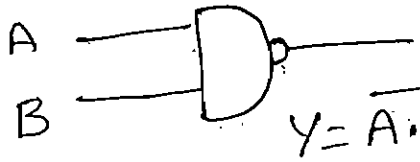

Step :- 2




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

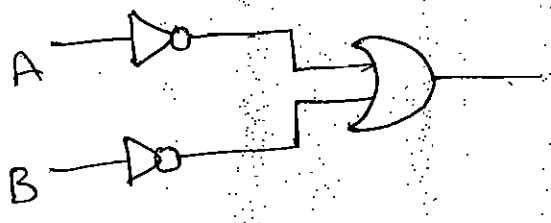
4) NAND Gate



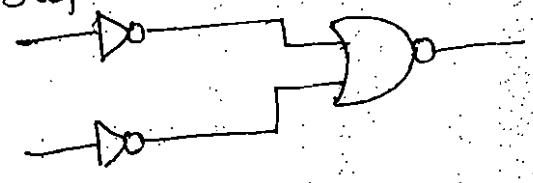

$Y = \overline{A \cdot B}$

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

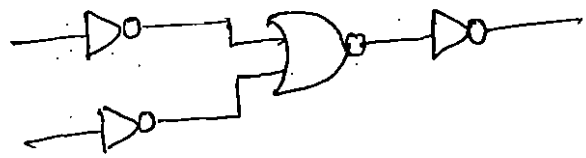
Step :- 1



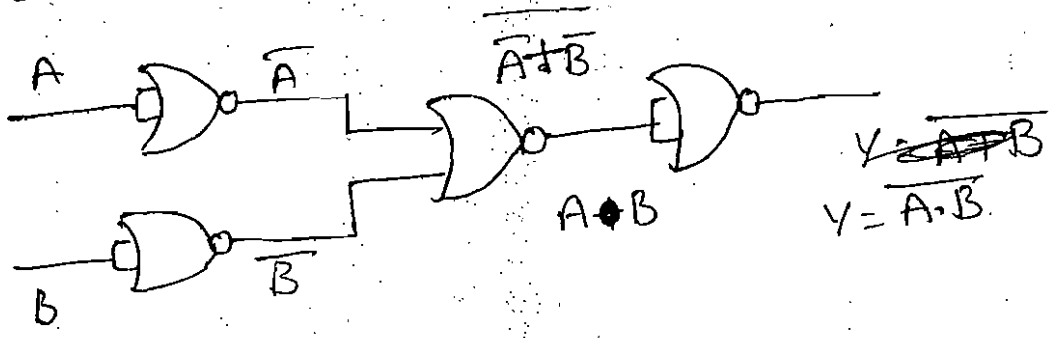
Step 2:-



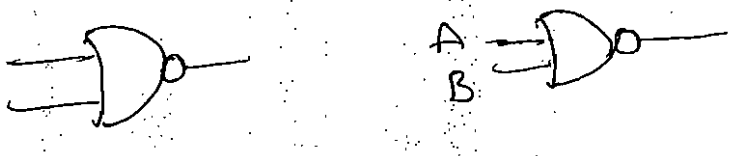
Step 3



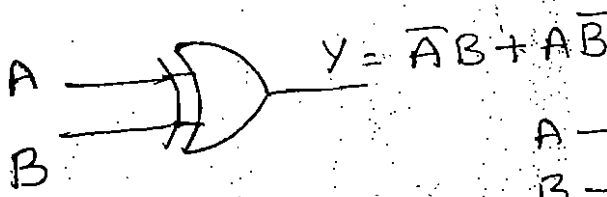
Step 4 :-



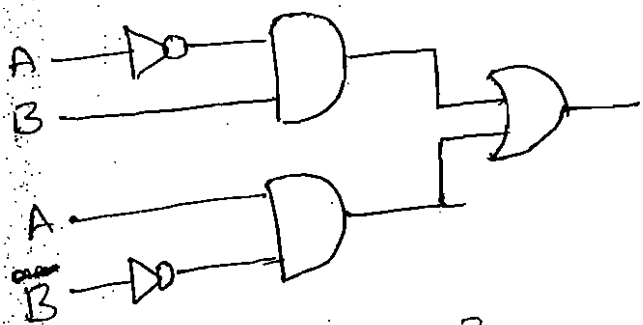

5) NOR Gate :-



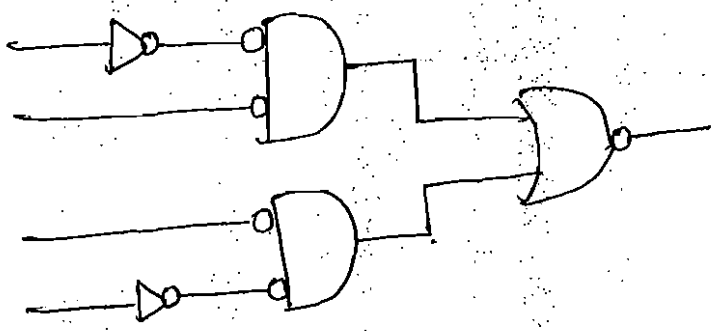

6) EX-OR Gate :-



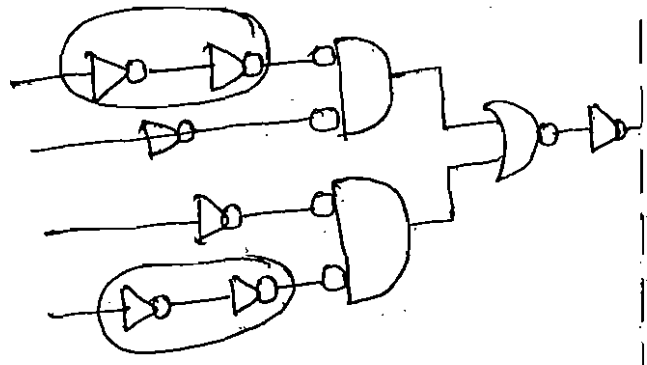

Step :- 1



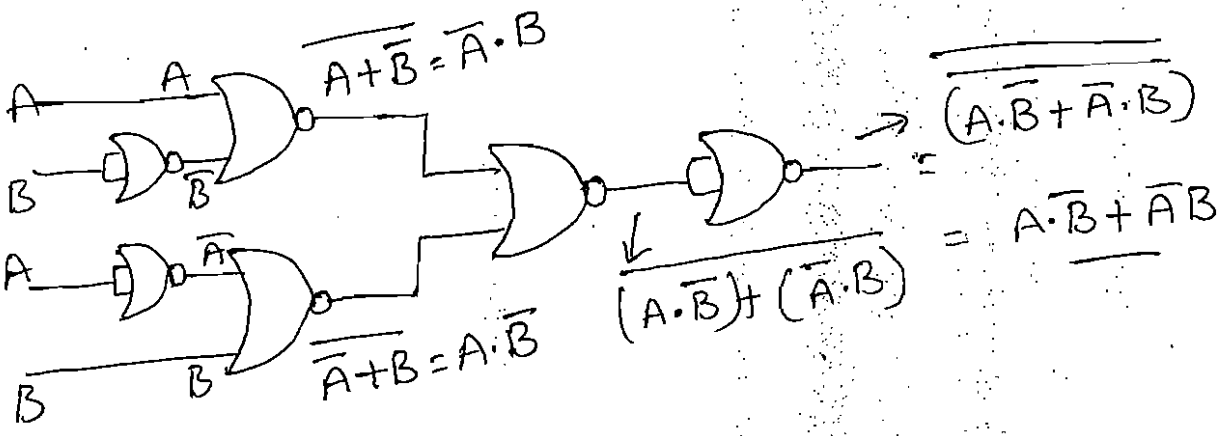

Step :- 2



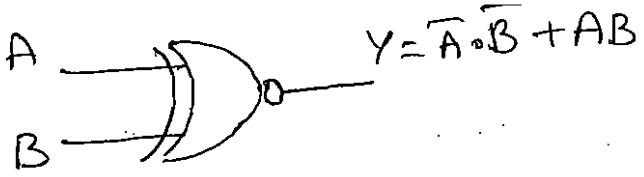
Step :- 3



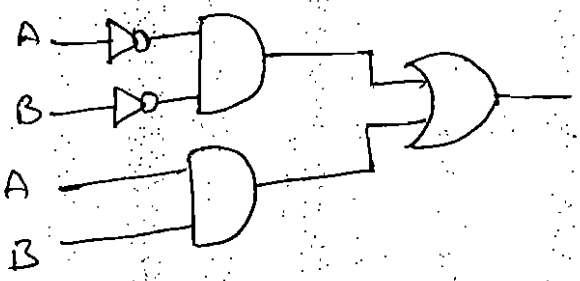
Step 4 :-



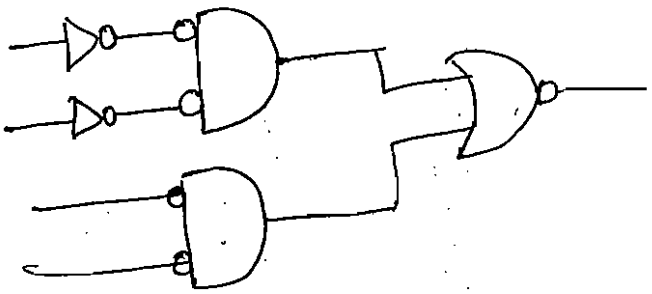

7) EX-NOR gate :-



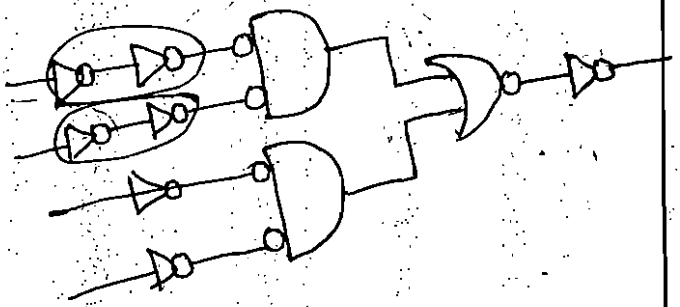

Step 1 :-



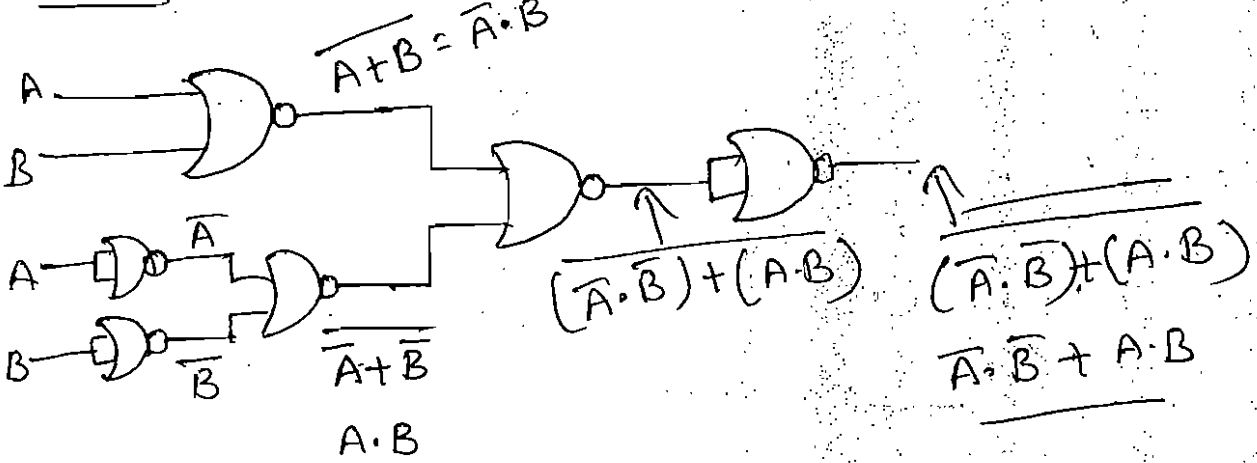

Step 2 :-



Step 3 :-

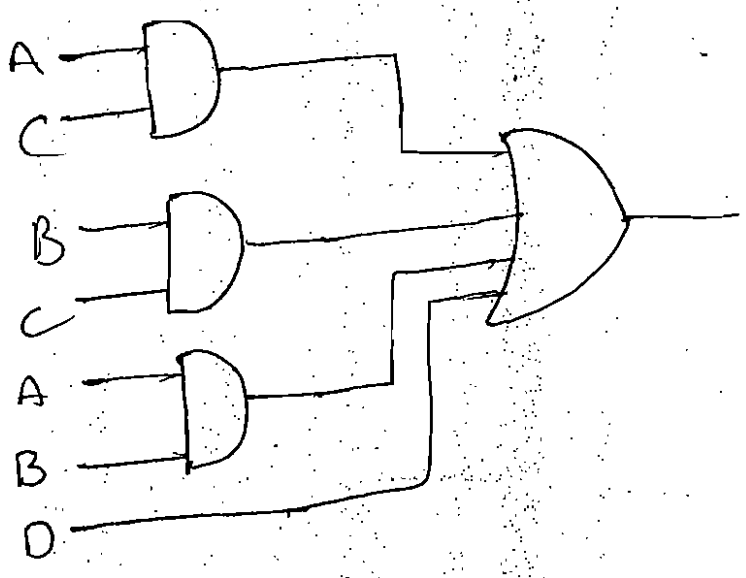


Step 4 :-

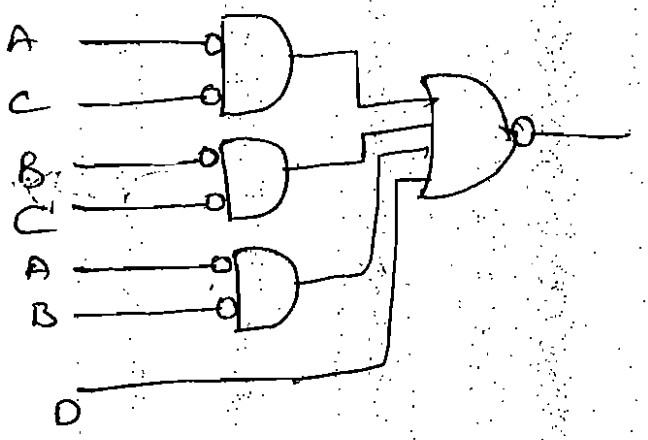


* $Y = AC + BC + AB + D$, implement Boolean Expression by using NOR Gate.

$Y = AC + BC + AB + D$,

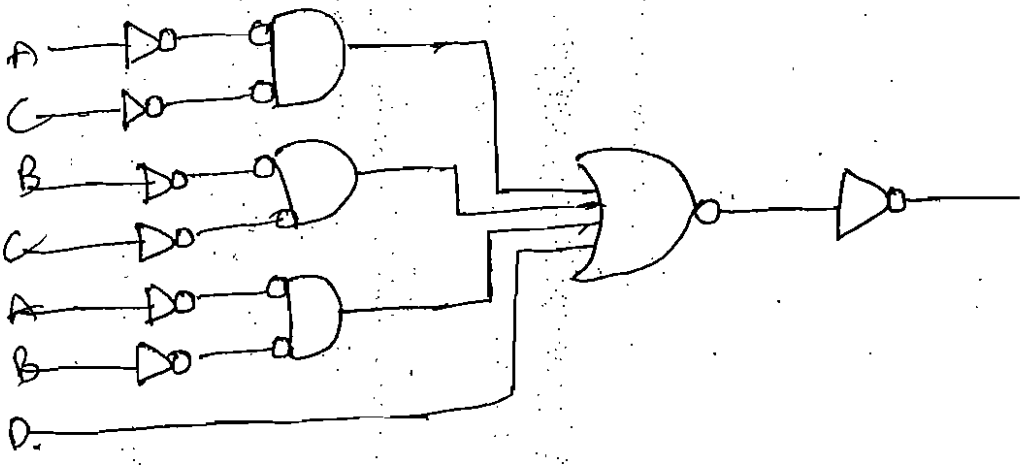



Step :- 1

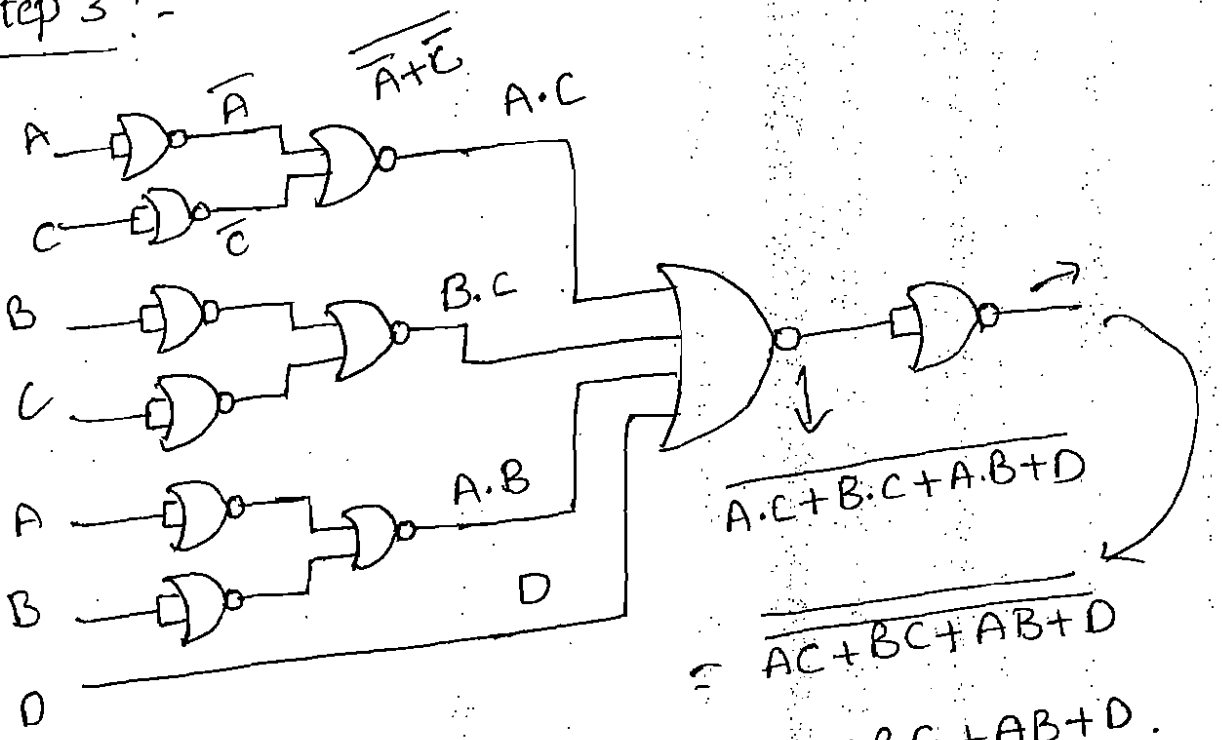



Step :- 2

Step :- 2



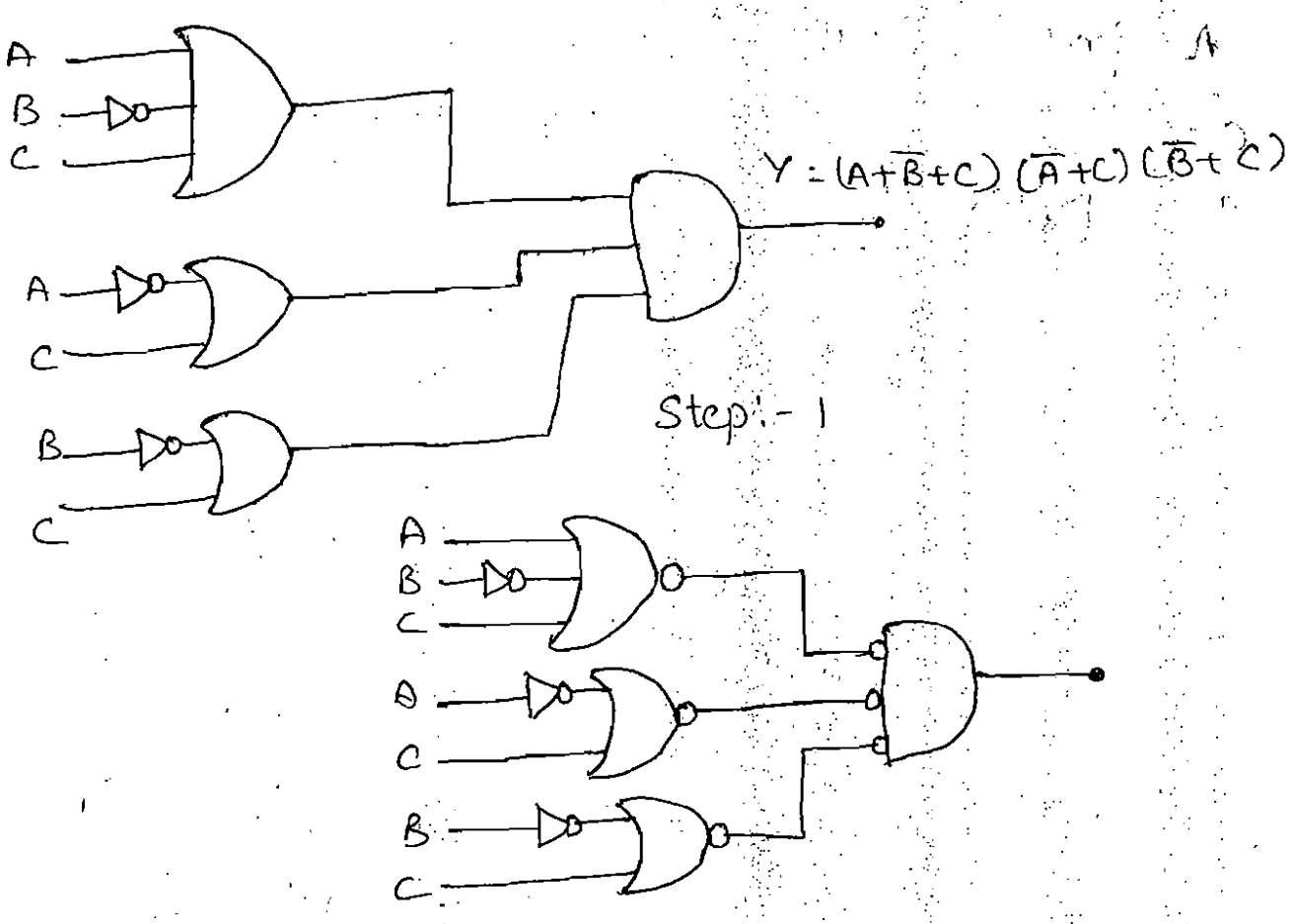
Step 3 :-



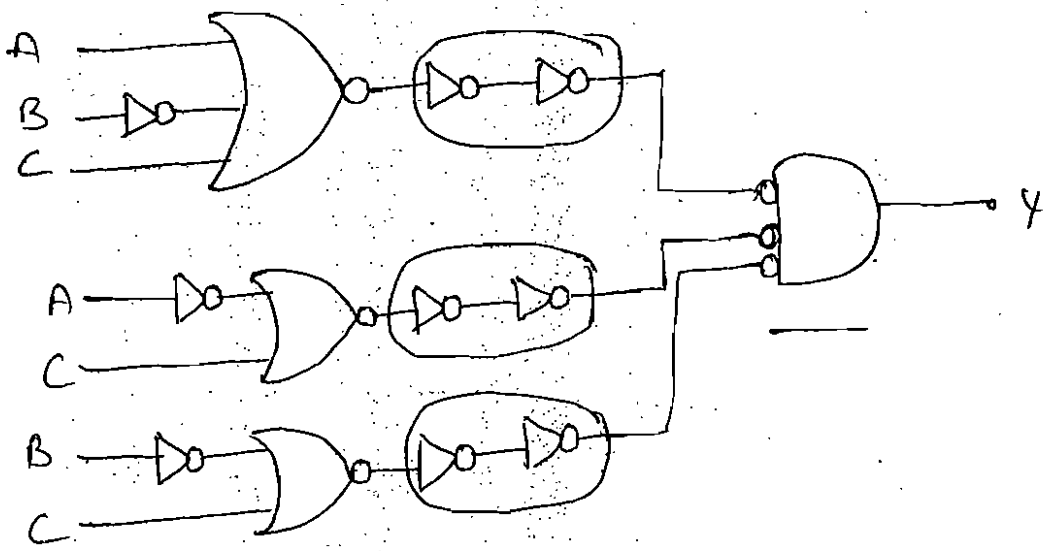

$= \overline{\overline{AC+BC+AB+D}}$

$= AC+BC+AB+D.$

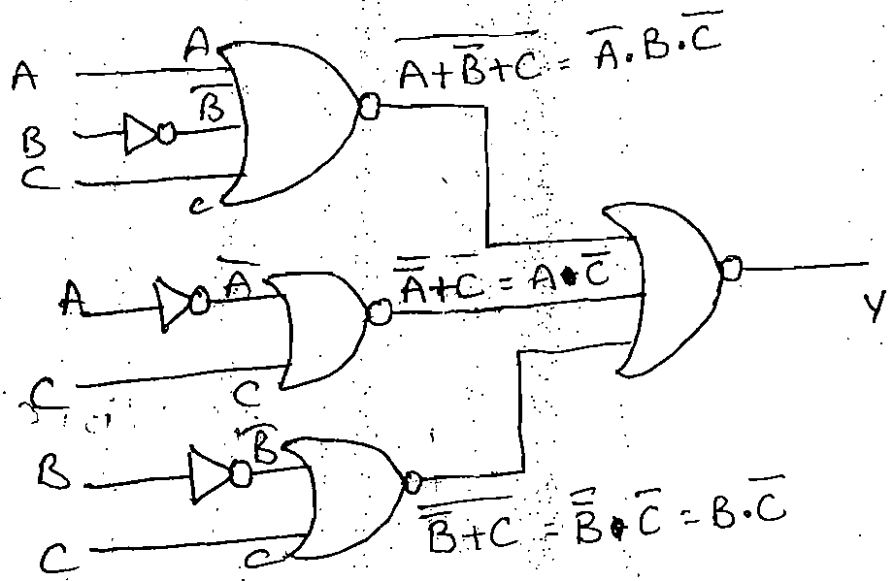
* implement the Boolean Expression by using NOR Gate.
$(A+\bar{B}+C)\ (\bar{A}+C)\ (\bar{B}+C)$.



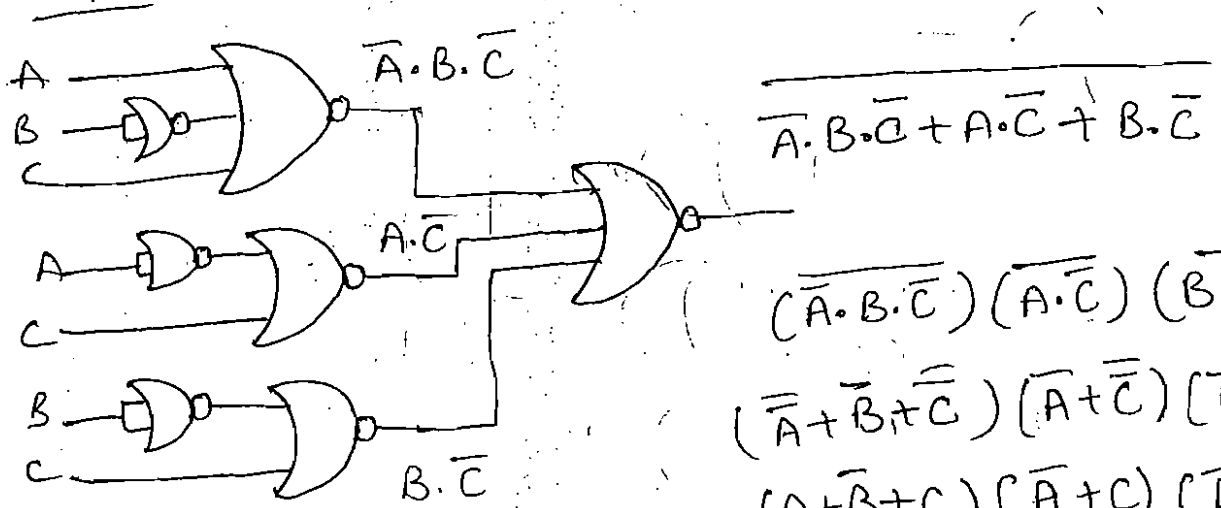
Step 2 :-




Step 3 :-




Step 4 :-




$$\overline{\bar{A}.B.\bar{C} + A.\bar{C} + B.\bar{C}}$$

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

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

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