STLD

UNIT-I

REVIEW OF NUMBER OF SYSTEMS & CODES

Representation of Numbers of Different Radix:—

In the Decimal system the 'Base' is also called as 'radix'. we have seen Number systems with radix (base) r equal to 10, 2, 8, & 16. Each number system has 'r' set of characters. for Example, in decimal NO.Sys 'r' equal to 10 and it has 10 characters from 0 to 9.

→ In binary No.sys 'r' equals to 2 and it has 2 characters 0 & 1 are used to represent any number. Individual digits in the binary system are called "bit's"

→ The base is usually indicated as a 'subscript."

Eg:- Decimal No $(23.625)_{10} = (10111.101)_2$

→ In general we can say that a number represented in radix 'r' has 'r' characters in its set and r can be any value.

Radix (Base) r	characters in set
2 (Binary)	0, 1
3	0, 1, 2
4	0, 1, 2, 3
⋮	⋮
7	0, 1, 2, 3, 4, 5, 6
8 (octal)	0, 1, 2, 3, 4, 5, 6, 7.
⋮	⋮
10 (Decimal)	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
⋮	
16 (Hexa Decimal)	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E F.

Philosophy of Number systems:—

→ Number system is a basis for counting various items we are familiar with decimal number system with its 10 digits i.e 0, 1, 2, 3, 4 - - - - 9

→ But modern computer communicate and operate with binary numbers which use only 0 and 1.

Eg: 18 - 10010

In decimal it is represented by two digits where as in binary 5 digits

∴ If decimal quantities are expressed in binary from they take more digits

→ For large decimal numbers people have to deal with very large binary

strings and ∴ they do not like working with binary numbers. this fact give rise to these new number systems.
i.e octal, Hexadecimal and Binary coded decimal.

→ These number systems represent binary number in a compressed form.
∴ these number systems are now widely used to compress long strings of binary numbers.

Decimal Number Systems:—

In thise we can express any number in units, hundreds, thousands and so on....

Eg:- 5678.9, it can be represented as
$5000 + 600 + 70 + 8 + 0.9 = 5678.9$

→ This NO can be written as 5678.9_{10}.
where 10 subscript indicates the radix (or) base.

→ In power of 10 we can write as
$5\times10^3 + 6\times10^2 + 7\times10^1 + 8\times10^0 + 9\times10^{-1}$

This shows that, the position of a digit w.r.to decimal point determines its value/weight. The sum of all the digits multiplied by their weights gives the total number being represented.

→ The left most digit which has "greatest weight" is called the "Most significant bit" (MSB).

→ The Right most bit which has the "least weight" is called "Least significant bit" (LSB).

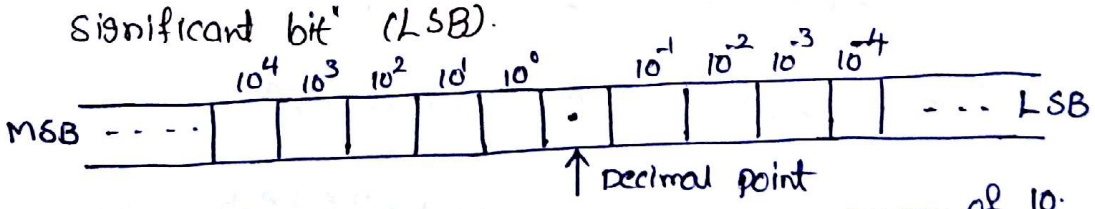


Fig:- Decimal position values as power of 10.

Eg:- Represent decimal number 98.72 in power of 10.

Sol:- $9\times10^1 + 8\times10^0 + 7\times10^{-1} + 2\times10^{-2}$

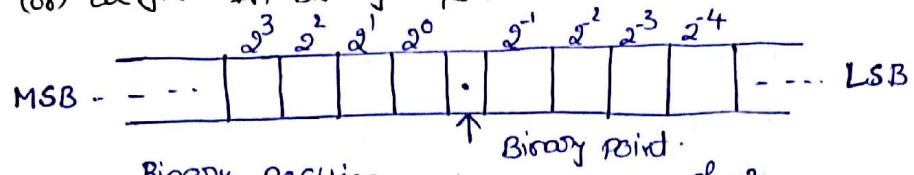
the digit 9 has a weight of 10,
8 " " " 1,
7 " " " of 1/10 &
2 " " of 1/100.

G. Naveen

Binary Number system:-

W.K.T Decimal system with it's ten digits is a base-ten system. Similarly binary system with it's two digits is a base-two system. The two binary digits (bits) are 1 and 0.

In binary system each binary digit commonly known as bit, has its own value (or) weight. In binary system weight is expressed as power of '2'.



Binary position values as a power of 2

Eg:- Represent binary number 1101.101 in power of 2 and find its decimal equivalent.

Sol:
$$N = 1\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0 + 1\times2^{-1} + 0\times2^{-2} + 1\times2^{-3}$$
$$= 8 + 4 + 0 + 1 + 0.5 + 0 + 0.125 = 13.625_{10}$$

Octal Number system:-

W.K.T the base of the decimal number system is 10 because it uses the digits 0 to 9. and the base of binary number system is '2' because it uses digits 0 & 1. the octal number system uses first eight digits of decimal number system i.e 0,1,2,3,4,5,6,& 7. As it uses 8 digits, its base is '8'.

Eg:- Rep octal number 567 in power of 8 and find its decimal equivalent

Sol:-
$$5\times8^2 + 6\times8^1 + 7\times8^0 = 5\times64 + 6\times8 + 7\times1$$
$$= 320 + 48 + 7 = 375_{10}$$

Hexa decimal Number System:-

The hexadecimal number system has a base of 16 having 16 digits

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

→ It is another number system that is particularly useful for human-communications with a computer

→ It is a base is power of 2 (2^4).

→ each hexadecimal digit represents a group of four binary 'digits' called "nibbles", that are fundamental parts of larger binary words.

Relation b/w Decimal, Binary and hexadecimal, octal Number:-

Decimal	Binary	Hexadecimal	octal
0	0000	0	0
1	0001	1	1
2	0010	2	2

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

Eg:- Represent hexadecimal number 3FD in power of 16 and find its decimal equivalent.

Sol:-

$$3\times16^2 + F\times16^1 + D\times16^0$$

(3, F, D)

$$\therefore 3\times256 + 15\times16 + 13\times1 = 768+240+13 = (1021)_{10}$$

Eg:- Find the decimal equivalent of $(231.23)_4$

Sol:

$$N = 2\times4^2 + 3\times4^1 + 1\times4^0 + 2\times4^{-1} + 3\times4^{-2}$$

$$= 32 + 12 + 1 + 0.5 + 0.1875 = (45.6875)_{10}$$

Counting the Radix in 5 :-

Eg:- 1. 5 in radix $(10)_5$ — $(5)_{10}$ $\Rightarrow 1\times5^1 + 0\times5^0 = 5+0 = 5_{10}$

2. $(11)_5$ — $(6)_{10}$ $\Rightarrow 1\times5^1 + 1\times5^0 = 5+1 = 6_{10}$

$(7)_{10}$ — $(12)_5$ $\Rightarrow 1\times5^1 + 2\times5^0 = 5+2 = 7_{10}$

$(8)_{10}$ — $(13)_5$ $\Rightarrow 1\times5^1 + 3\times5^0 = 5+3 = 8_{10}$

$(9)_{10}$ — $(14)_5$ $\Rightarrow 1\times5^1 + 4\times5^0 = 5+4 = 9_{10}$

Eg: Represent $(13)_{10}$ in octal

Sol: $(13)_{10}$ — $(?)_8$

$8 \mid 13$

$1 \quad -5$ $= (15)_8$

check: $(15)_8$ — $(?)_{10}$

$1\times8^1 + 5\times8^0$

$= 8+5 = (13)_{10}$

G. Naveen
EGE, NEC

Conversation from one Radix to another Radix :-

→ The human beings use decimal number system while computer uses binary number system. ∴ It is necessary to convert decimal number into its equivalent binary while feeding number into the computer and to convert binary number into its decimal equivalent while displaying result of operation to the human being's.

→ Dealing with a large quantity of binary numbers of many bits is in convenient for human being's.
∴ octal & hexadecimal numbers are used as short hand means of expressing large binary numbers.

→ But it is necessary to keep in mind that the digital em circuits and systems work strictly in binary. we are using octal & hexadecimal only as a convenience for the operator's of the systems.

Binary to octal conversion :-

For octal number's base is 8 and the base for binary is 2 ie the base for octal number is the third power of the base for binary numbers.

→ By grouping 3 digits of binary number & then converting each group digits to its octal equivalent. we can convert binary number to its octal equivalent.

Eg:- convert $(111\ 101\ 100)_2$ to octal equivalent

Sol:-

111	101	100
7	5	4

∴ octal number = $(754)_8$

Octal to Binary conversion :-

conversion from octal to binary is a reversal of the process explained in the previous section.

→ Each digit of the octal number is individually converted to its binary equivalent to get octal to binary conversion of the number

Eg:- convert $(634)_8$ to binary

Sol:-

6	3	4
110	011	100

∴ Binary number = 110 011 100

Eg:- convert $(725.63)_8$ to binary

Sol:

7	2	5	.	6	3
111	010	101		110	011

∴ Binary number = 111010101 . 110011

Binary to Hexadecimal conversion:-

→ Base for hexadecimal is '16' and the base for binary is '2'.

i.e the base for hexadecimal number is the "fourth power" of the base for binary numbers

↔ By grouping 4 digits of binary numbers and the converting each group digit to its hexadecimal equivalent we can convert binary number to its hexadecimal equivalent.

Eg:- convert $(1101\ 1000\ 1001\ 1011)_2$ to hexadecimal equivalent

Sol:-

1101	1000	1001	1011
D	8	9	B

∴ Hexa decimal number = $D89B_{16}$ (or) D89BH

Eg:- Convert $0101\ 1111\ 1011.0111\ 1100_2$ to hexadecimal

Sol:-

0010	1111	1011 .	0111	1100
2	F	B	7	C

∴ Hexadecimal Number = $2FB.7C_{16}$ (or) 2FB.76H

Hexa decimal to Binary conversion:-

It is reversal process.

→ Each digit of the hexadecimal number is individually converted to its binary equivalent to get hexa decimal to binary conversion of the number.

Eg:- Convert $(3FD)_H$ to binary

Sol:-

3	F	D
0011	1111	1101

∴ Binary number : $0011\ 1111\ 1101_2$

Eg:- convert $(5A9.B4)_H$ to binary

Sol:-

5	A	9	.	B	4
0101	1010	1001		1011	0100

∴ Binary number = 0101 1010 1001 . 1011 0100

Eg:- Convert $3A9E.B0D_{16}$ to binary.

3	A	9	E	.	B	0	D
0011	1010	1001	1110		1011	0000	1101

∴ Binary number = 0011 1010 1001 1110 . 1011 0000 1101

G.NAVEEN
EEE

Octal to Hexadecimal conversion:-

The easiest way to convert octal number to hexadecimal number is given below.

1. convert octal number to its binary equivalent.
2. convert binary number to its hexadecimal equivalent.

Eg:- convert $(615)_8$ to its hexadecimal equivalent

Sol:- Step 1:- octal to binary

6 1 5

110 001 101 ∴ Binary number = 110001101

Step 2:- Binary to hexadecimal

0001 1000 1101

1 8 D

∴ Hexadecimal Number = 18DH (or) $18D_{16}$

Eg:- convert 756.603_8 to hex.

Sol:

	7	5	6	.	6	0	3
octal to binary:	111	101	110	.	110	000	011
Group of 4 bits.	0001	1110	1110	.	1100	0001	1000
	1	E	E	.	C	1	8

∴ $1EE.C18_{16}$

Hexa decimal to octal conversion:-

The easiest way to convert hexadecimal number to octal number is given below.

1. convert hexadecimal number to its binary equivalent
2. Convert binary number to its octal equivalent.

Eg:- Convert $(25B)_H$ to its octal equivalent

Sol:- Step:1 Hexadecimal to binary.

2 5 B

0010 0101 1011 ∴ Binary no: 0010 0101 1011

Step:2: Binary to octal

001 001 010 011

1 1 3 3 ∴ octal No = 1133_8

Eg:- Convert $B9F.AE_{16}$ to octal.

Sol:

	B	9	F	.	A	E	
hex - binary	1011	1001	1111	.	1010	1110	
Group of 3 bits.	101	110	011	111	.	101	011	100
octal	5	6	3	7	.	5	3	4

∴ octal No = 5637.534_8

Converting any Radix to Decimal:-

In general, numbers can be represented as

$$N = A_{n-1}r^{n-1} + A_{n-2}r^{n-2} + \cdots + A_1 r^1 + A_0 r^0 + A_{-1} r^{-1} + A_{-2} r^{-2} + \cdots C_{-m} r^{-m}$$

where N = Number in decimal, A = Digit

r = Radix (or) base of a number system

n = the number of digits in the integer Position of number

m = the number of digits in the fractional Portion of number.

from this general equation we can convert number with any radix into its decimal equivalent.

Eg:- convert binary number $(1101.1)_2$ to its decimal equivalent

Sol:- $N = 1\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0 + 1\times2^{-1}$

$= 8 + 4 + 0 + 1 + 0.5 = 13.5_{10}$

Eg:- convert $(475.25)_8$ to decimal equivalent

Sol:- $N = 4\times8^2 + 7\times8^1 + 5\times8^0 + 2\times8^{-1} + 5\times8^{-2}$

$= 256 + 56 + 5 + 0.25 + 0.078125 = 317.32813_{10}$

Eg:- convert $(9B2.1A)_H$ to its decimal equivalent

Sol:- $N = 9\times16^2 + 11\times16^1 + 2\times16^0 + 1\times16^{-1} + 10\times16^{-2}$

$= 2304 + 176 + 2 + 0.0625 + 0.0390 = 2482.1_{10}$

Eg:- convert $(3102.12)_4$ to its decimal equivalent

Sol:- $N = 3\times4^3 + 1\times4^2 + 0\times4^1 + 2\times4^0 + 1\times4^{-1} + 2\times4^{-2}$

$= 192 + 16 + 0 + 2 + 0.25 + 0.125 = 210.375_{10}$

Eg:- convert $(614.15)_7$ to its decimal equivalent

Sol:- $N = 6\times7^2 + 1\times7^1 + 4\times7^0 + 1\times7^{-1} + 5\times7^{-2}$

$= 294 + 7 + 4 + 0.142857 + 0.102 = 305.24486_{10}$

Conversion of Decimal Numbers to any Radix Numbers:-

The conversion of decimal number to any radix number in two steps.

Step1:- we have to convert integer part - this is done by "Successive division method"

Step2:- " " " fractional part - " " "Successive multiplication" method.

G.NAVEEN
EEE

Successive Division for Integer Part conversion:-

In this method repeatedly divide the integer part of the decimal number by r (the new radix). until quotient is zero. The remainder of each division becomes the numeral in the new radix. The remainders are taken in the reverse order to form a new radix number.

This means that the first remainder is the least significant digit (LSD) and the last remainder is the most significant digit (MSD) in the new radix number.

Eg:- Convert decimal number 37 to its binary equivalent.

Sol:-

Q	R	
2 | 37		
2 | 18	1	LSD
2 | 9	0	
2 | 4	1	
2 | 2	0	
2 | 1	0	
0	1	MSD

Q → Quotient

R → Remainder

∴ Binary equivalent = 100101_2

Eg:- Convert decimal number 214 to its octal equivalent

Sol:-

Q	R	
8 | 214		
8 | 26	6	LSD
8 | 3	2	
0	3	MSD

326_8 = octal equivalent

Eg: Convert decimal number 3509 to its hexadecimal equivalent

Sol:-

16 | 3509		LSD
16 | 219	5	
16 | 13	11 - B	
0	13 - D	MSD

∴ hexadecimal equivalent = $DB5_{16}$

Eg: Convert 54_{10} to radix 4

Sol:-

4 | 54		
4 | 13	2	LSD
4 | 3	1	
0	3	MSD

∴ radix 4 is = $(312)_4$

Successive multiplication for Fractional Part conversion:-

Conversion of fractional decimal numbers to another radix number is accomplished using a successive multiplication method. In this method, the number to be converted is multiplied by the radix of the new number, producing a product that has an integer part and a fractional part. the integer part (carry) of the product becomes a numeral in the new radix number. The fractional part is again multiplied by the radix and this process is repeated until fractional part reaches '0' or until the new radix number is carried out to

sufficient digits. The Integer Part (Carry) of each product is read downward to represent the new radix number.

Eg:- Convert 0.8125 decimal number to its binary equivalent.

Sol:-

Fraction	Radix	Result	Recorded Carries.
0.8125	x 2	= 1.625 = 0.625	with a carry of 1
0.625	x 2	= 1.25 = 0.25	" " 1
0.25	x 2	= 0.5 = 0.5	" " 0
0.5	x 2	= 1.0 = 0	" " 1

MSD ↓ LSD

Reading Carries downward we get,

binary fraction: 0.1101, which is equal to 0.8125 decimal.

Eg:- Convert 0.95 decimal number to its binary equivalent.

Sol:-

Fraction	Radix	Result	Recorded Carries
0.95	x 2	1.9 = 0.9	with a carry of 1
0.9	x 2	1.8 = 0.8	" " 1
0.8	x 2	1.6 = 0.6	" " 1
0.6	x 2	1.2 = 0.2	" " 1
0.2	x 2	0.4 = 0.4	" " 0
0.4	x 2	0.8 = 0.8	" " 0
0.8	x 2	1.6 = 0.6	" " 1

MSD ↓ LSD.

In this case, 0.8 is repeated & if we multiply further we will get repeated sequence. If we stop here, we get 7 binary digits .1111001. this number is an approximate answer.

To get more accurate answer we have to continue multiplying by 2 until we have as many digits as necessary for our application

Eg:- Convert 0.640625 decimal number to its octal equivalent.

Sol:

Fraction	Radix	Result	Recorded Carries
0.640625	x 8	= 5.125 = 0.125	with a carry of 5
0.125	x 8	= 1 = 0	with a carry of 1

MSD ↓ LSD

∴ octal = 0.51

Eg:- Convert 0.12890625 decimal number to its hexa equivalent.

Sol:

Fraction	Radix	Result	Recorded Carries.
0.12890625	x 16	= 2.0625	2. MSD ↓
0.0625	x 16	= 1.0	1 LSD

hexa = 0.21_{16}.

G.NAVEEN
EEE

Numbers having both Integer part & Fractional part:-

Eg: Convert decimal number 24.6 to a binary number.

Sol:-

Step:1 Separate out integer Part & fraction Part

Integer part = 24 Fractional part = 0.6

Step 2:- Find equivalent binary number for integer part

2	Q	R
2	24	
2	12	– 0
2	6	– 0
2	3	– 0
2	1	– 1
	0	– 1

LSD ↑ MSD

$= 11000_2$

Step 3:- Find equivalent binary no. for fractional part

Fraction	Radix	Result	Recorded Carries
0.6	× 2	= 1.2 = 0.2	1
0.2	× 2	= 0.4 = 0.4	0
0.4	× 2	= 0.8 = 0.8	0
0.8	× 2	= 1.6 = 0.6	1
0.6	× 2	= 1.2 = 0.2	1

MSD ↓ LSD

$= 0.10011$

$\therefore 24.6 = 11000.10011$

Eg:- Convert decimal number 35.45 to octal number.

Sol:- **Step 1:-** Integer part = 35 Fractional Part = 0.45

Step-II:-

8	Q	R
8	35	
	4	– 3
	0	– 4

↑ $= (43)_8$

Step-3:-

		Carry
0.45 × 8	= 3.6 = 0.6	3
0.6 × 8	= 4.8 = 0.8	" 4
0.8 × 8	= 6.4 = 0.4	" 6
0.4 × 8	= 3.2 = 0.2	" 3
0.2 × 8	= 1.6 = 0.6	" 1

MSD ↓ LSD

$= 0.34631$

$\therefore$ Octal NO = $(43.34631)_8$

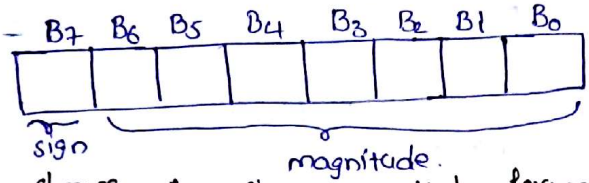
Signed Binary Numbers:—

→ In General we use '+' sign to represent positive numbers and '−' " " negative number.

→ But because of hardcoare limitations, in computers both +ve & −ve numbers are represented with only binary digits.

→ the left most bit (sign bit) in the number represents sign of the number. the sign bit is '0' for '+ve' numbers, & it is '1' for '−ve' numbers

→ These numbers are represented by the signed magnitude format



Above fig shows the sign magnitude format of 8 bit signed number. Here MSB represents sign of the Number.

If MSB is '1' NO is −ve &. If MSB is '0' NO is +ve. remaining bits represents magnitude of the NO.

Eg:- 1) +6 = 00000110 ③ +24 = 00011000

2) −14 = 10001110 ④ −64 = 11000000

→ In case of unsigned 8-bit binary numbers the decimal range is 0 to 255.

→ for signed magnitude magnitude 8-bit binary number the largest magnitude is reduced from 255 to 127 because we need to represent both +ve & −ve numbers.

max +ve NO 0111 1111 = +127

max −ve NO 1111 1111 = −128

This is only way to represent +ve numbers

→ Hoco ever there are too more way's to represent −ve numbers.

these are 1) signed −1's Complement representation

2) signed 2's " "

→ Complement Representation of Negative Numbers:—

→ In digital computers, to simplify the subtraction operation & for logical manipulation complements are used.

∴ These are two types of complements for each radix system's.

1) The Radix complement and

2) Diminished radix complement.

the 1st one is refered to as r's complement &

the 2nd as the (r−1)'s complement.

→ for example, In binary systems we substitute base value 2 in place of 'r' to refere Complements as 2's complement and 1's Complement

G.NAVEEN
EEE

→ In Decimal number system's we substitute base value 10 in place of 'r' to refere complements as 10's complement & 9's complement.

1's complement Representation:-

The 1's complement of a binary number is the number that results when we change all 1's to zero's & the zero's to one's.

Eg:- Find 1's complement of $(1101)_2$

Sol:-
1 1 0 1
0 0 1 0 → 1's complement

Eg: Find 1's complement of 10111001

Sol:
1 0 1 1 1 0 0 1
0 1 0 0 0 1 1 0 → 1's complement

2's complement Representation:-

The 2's complement is the binary number that results when we add 1 to the 1's complement. It is given as

2's complement = 1's complement + 1

The 2's complement form is used to represent negative number's

Eg:- Find 2's complement of $(1001)_2$

Sol:
1 0 0 1
0 1 1 0 1's complement
\+ 1
0 1 1 1 2's complement

Eg: Find 2's complement of $(10100011)_2$

Sol:
1 0 1 0 0 0 1 1
0 1 0 1 1 1 0 0 1's comp
\+ 1
0 1 0 1 1 1 0 1 2's comp

1's complement subtraction:-

Subtraction of binary number's can be accomplished by the direct method by using 1's complement method, which allow's to perform subtraction using only addition for subtraction of two number's we have two cases.

→ Subtraction of smaller number from larger number and
→ Subtraction of larger number from smaller number

Subtraction of Smaller Number from larger Number:-

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

This is called "end-around carry"

Eg:- Subtract $1010\,11_2$ from $111\,001_2$ using the 1's complement

Sol:-
1 1 1 0 0 1
\+ 0 1 0 1 0 0 → 1's complement of 101011
(1) 0 0 1 1 0 1
→ + 1 Add end around carry
0 0 1 1 1 0 Final answer

Subtraction of larger number from smaller number:-

Method:- 1. Determine the 1's complement of the larger number
2. Add the 1's complement to the smaller number
3. Answer is in 1's complement form so get the answer in true form take the 1's complement and assign negative sign to the answer

Eg:- Subtract 111001_2 from 101011_2 using the 1's complement method.

Sol:

```
  0 0 0 1 1 0     -> 1's complement of 111001
  1 0 1 0 1 1
-------------
  1 1 0 0 0 1     Answer in 1's complement form
- 0 0 1 1 1 0     true form
```

Advantages of 1's complement subtraction:-

1. The 1's complement subtraction can be accomplished with an binary adder
   ∴ This method is useful in arithmetic logic circuits
2. The 1's complement of a number is easily obtained by inverting each bit in the no

2's complement subtraction:-

Like 1's complement subtraction, in 2's complement subtraction, the subtraction is accomplished by only addition. let us see the methods for 2's complement subtraction

Subtraction of smaller number from larger number:-

Method:- 1. Determine the 2's complement of a smaller number
2. Add the 2's complement to the larger number
3. Discard the carry.

Eg:- Subtract 101011_2 from 111001_2 using 2's complement method.

Sol:-

```
    1 0 1 0 1 1
(1) 0 1 0 1 0 0   is complement of 101011
  +           1
  -------------
    0 1 0 1 0 1   2's complement of 101011

(2) 1 1 1 0 0 1
    0 1 0 1 0 1
  -------------
carry (1) 0 0 1 1 1 0
discard

    00 1 1 1 0   final answer
```

G. NAVEEN
ECE

Subtraction of Larger Number from Smaller Number:-

Method:-
1. Determine the 2's complement of the larger number
2. Add the 2's complement to the smaller number
3. Answer is in the 2's complement form. To get the answer in the true form take 2's complement and assign negative sign to the answer.

Eg:- Subtract 111001_2 from 101011_2 using 2's complement method.

Sol:-

```
  000110      is complement of 111001
+      1
--------
  000111      2's complement of 111001

  101011
  ①①①①
--------
  110010      Answer is in 2's complement form

- 001101      is complement
      1
--------
- 001110      True form
```

Binary codes:-

Usually the digital data is represented, stored & transmitted as groups of binary digits (bits). The group's of binary digits (or) bits also known as "binary code".

Classification of Binary codes:-

1. Weighted codes → 8421, 2421 & 5211 (Binary & BCD)
2. Non-weighted codes → Excess 3 & Grey code.
3. Reflective codes → code for 9's complement, for the code for 0, 8, 1, 7. --2421, 4221
4. Sequential codes
5. Alphanumeric codes.
6. error detecting & correcting codes.

Weighted codes:- BCD

In weighted codes, each digit position of the number represents a specific weight.

Eg:- In Decimal code, if number is 567 then weight of 5 is 100, weight of 6 is 10 & weight of 7 is 1.

In weighted binary codes each digit has a weight 8, 4, 2 or 1.

The codes 8421, 2421 & 5211 are weighted codes.

BCD (8421) Binary coded Decimal:-

→ In BCD each digit of a decimal number is represented by a separate Group of bits.

Eg:- 5 8

0101 1000

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

→ It is called 8*4-2-1 BCD because the weights associated with 4 bits are 8-4-2-1 from "left to right".

This mean that, bit 3 has weight 8, bit 2 has weight 4, bit 1 has weight 2 and bit '0' has weight 1.

→ The 4bit 8-4-2-1 BCD code used to represent a single decimal digit.

Decimal	BCD code			
Digit	8	4	2	1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

In multidigit coding, each decimal digit is individually coded with 8-4-2-1 code.

→ multidigit coding of 8-4-2-1 BCD numbers we require 4-bits per decimal digit. ∴ Total 8-bits are required to encode 58_{10} in 8-4-2-1 BCD.

→ when we represent the same number 58 in binary i.e 111010_2 we require only 6 digits

→ This means that, for representing numbers, 8-4-2-1 BCD is less efficient than pure binary number system.

→ Advantage of BCD code is that it is easy to convert between it and decimal.

→ Disadvantage of BCD, besides its less efficiency, is that arithmetic operations are more complex than they are in pure binary.

G.NAVEEN
EEE

Arithmetic operations using 8-4-2-1 BCD:-

BCD Addition:-

Addition of two BCD numbers can be understood by considering 3 cases that occur when two BCD digits are added.

1. Sum equals 9 (or) less with carry 0
2. Sum greater than 9 with carry 0
3. Sum equals 9 (or) less with carry 1

1. Sum equals 9 or less with carry 0:-

Let us consider additions of 3 and 6 in BCD

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

The addition is carried out as in normal binary addition and the sum is 1001 which is BCD code for 9.

2. Sum greater than 9 with carry 0:-

Addition of 6 & 8 in BCD

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

→ The sum 1110 is an Invalid BCD number. This has occured because the sum of the two digits exceeds 9. when ever this occurs the sum has to be corrected by the addition of six (0110) in the invalid BCD number.

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

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

3. Sum equals 9 (or) less with carry 1:-

Addition of 8 and 9 in BCD

```
  8        1 0 0 0   → BCD for 8
+ 9        1 0 0 1   → BCD for 9
---      -----------
 17   0001   0 0 0 1  → Incorrect BCD result
```

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

8	1000	→ BCD for 8
+9	1001	→ BCD for 9
17	0001 0001	→ Incorrect BCD result
	0000 0110	→ Add 6 for correction
	0001 0111	→ BCD for 17

Going through these 3 cases of BCD addition we can summarise the BCD addition procedure as follows.

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

Eg:- Decimal Addition in 8-4-2-1 BCD for 175, 326

Sol:-

175	0001	0111	0101	
+326	0011	0010	0110	
501	0100	1001	1011	>9 so add 6 (1011 >9)
			0110	
		1		
	0100	1010	0001	(1010) >9 so add 6
		0110		
	1			
	0101	0000	0001	Corrected sum.

BCD Subtraction:-

Addition of signed BCD numbers can be performed by using 9's or 10's complement methods.

A -ve BCD number can be expressed by taking the 9's (or) 10's complement.

Subtraction using 9's Complement:-

The 9's complement of a decimal number is found by subtracting each digit in the number from 9.

The 9's complement of each of the decimal digits is.

G. NAVEEN
EEE

Digit	9's complement
0	9
1	8
2	7
3	6
4	5
5	4
6	3
7	2
8	1
9	0

In 9's complement subtraction when 9's complement of smaller number is added to the larger number carry is generated. It is necessary to add this carry to the result.

→ when larger number is subtracted from smaller one. there is no carry. and the result is in 9's complement form and negative.

Eg:- ① 8 − 2 = 6

9's complement

8 + 7 (9's complement of 2) = 15

+1 (add carry to result)

6

② 9 − 5 = 4

9 + 4 (9's complement of 5) = 13

+1 (Add carry to result)

4

③ 4 − 8 = −4

4 + 1 (9's complement of 8) = 5

Also carry indicate that the answer is negative and in complement form

9's complement of result = −4

From the above examples we can summarize steps for 9's complement BCD subtraction as follows:

1. Find the 9's complement of negative number
2. Add two numbers using BCD addition
3. If carry is generated add carry to the result other wise find the 9's complement of the result.

Eg:- Perform the decimal subtractions in 8-4-2-1 BCD using 9's complement method ① 79 − 26 ② 89 − 54

① 79 − 26 = 53

0	1	1	1	1	0 0 1
+ 0	1	1	1	0	0 1 1
1	1	1	0	1	1 0 0
		(1)		0	1 1 0
1	1	1	1	0	0 1 0

73 → 9's complement for BCD 26

1100 > 9 so add 6

```
  1 1 1 Φ      0 0 1 0        1110 > 9  so add 6
+ 0 1 1 0
-----------------------
1 0 1 0 1      0 0 1 0
  └──────────→ + 1          (End around carry)
-----------------------
  0 1 0 1      0 0 1 1       BCD for 53
```

② 89 − 54 = 35

```
  1 0 0 0      1 0 0 1
  0 1 0 0      0 1 0 1       (45 → 9's complement of 54)
-----------------------
  1 1 0 0      1 1 1 0
                             1110 > 9  so add 6
             + 0 1 1 0
-----------------------
1 1 0 1      0 1 0 0
                             1101 > 9  so add 6
  0 1 1 0
-----------------------
1 0 0 1 1      0 1 0 0
  └──────────→ +     1       End around carry
-----------------------
  0 0 1 1      0 1 0 1       BCD for 35
```

2-4-2-1 BCD code:-

→ The 2421 is another self complementing code unlike excess-3 it has the additional feature that its 4-bit code groups are weighted.

→ In this case the weights are 2-4-2-1 meaning that bit 1 & bit 3 have the same weight (2). it is some times refered to as 2*-4-2-1 code.

→ when the asterisk simply distinguishes one position with weight 2 from the other since two positions have the same weight. there are two possible bit patterns that could be used to represent some decimal digits, but only one of those patterns is actually assigned.

Decimal	2	4	2	1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	1	0	1	1
6	1	1	0	0
7	1	1	0	Φ
8	1	1	1	0
9	1	1	1	1

G. NAVEEN

Excess - 3 code :-

Excess - 3 code is a modified form of a BCD number. the excess - 3 code can be derived from the natural BCD code by adding 3 to each coded number

Eg:- Decimal 12 can be represented in BCD as 0001 0010. Now adding 3 to each digit we get excess - 3 code as 0100 0101 (12 in decimal)

0001	0010
0011	0011
0100	0101

Excess - 3 codes to represent single decimal digit :-

Decimal Digit	Excess - 3 code
0	0 0 1 1
1	0 1 0 0
2	0 1 0 1
3	0 1 1 0
4	0 1 1 1
5	1 0 0 0
6	1 0 0 1
7	1 0 1 0
8	1 0 1 1
9	1 1 0 0

In BCD Subtraction we have to compute 9's complement of the number before subtraction

In Excess - 3 code we get 9's complement of a number by just complementing each bit. Due to this excess - 3 code is called "self - complementing code".

Eg:- Find the excess - 3 code and its 9's complement for following decimal numbers

(a) 592 (b) 403

Sol:-

592_{10} = 1000 1100 0101

9's complement of 592_{10} = 0111 0011 1010

→ 403_{10} = 0111 0011 0110

9's complement of 403_{10} = 1000 1100 1001

Excess - 3 Addition :-

To Perform excess - 3 addition we have to

1. Add two Excess - 3 numbers
2. If carry = 1 → Add 3 to the sum of two digits
   = 0 → Subtract 3

Eg:- 8, 6 (14)

```
  1 0 1 1   → Excess-3 for 8
+ 1 0 0 1   → Excess-3 for 6
-----------
1 0 1 0 0
```

Add 3: `0011 0011`

```
0100 0111   → Excess-3 for 14
 4    7
```

Eg: 1, 2

1+2 = 3

```
0 1 0 0   → Excess-3 for 1
0 1 0 1   → "  "  "  2
---------
1 0 0 1
```

Sub 3. - `0011`

```
0 1 1 0   → Excess-3 for 3
```

Excess-3 Subtraction:—

To perform excess-3 subtraction we have to

1. Complement the subtrahend (using 9's complement)
2. Add complement subtrahend to minuend
3. Added If carry = 1 Result is positive. Add 3 and end around carry + Add 3
   If carry = 0 Result is negative. subtract 3

Truth table

0 - 0 = 0
0 - 1 = 1 with borrow 1
1 - 0 = 1
1 - 1 = 0

Eg:- 8 - 5

Sol:

```
  1 0 1 1   → Excess-3 for 8
+ 0 1 1 1   → Complement of 5 in excess-3
-----------
1 0 0 1 0
```

```
  0 0 1 1   → add 3
-----------
  0 1 0 1
→     1     → add end around carry
-----------
  0 1 1 0   → Excess-3 for 3.
```

(ii) 5 - 8

```
1 0 0 0   → Excess-3 for 5
0 1 0 0   → Complement of 8 in excess-3
---------
1 1 0 0
- 0 0 1 1
---------
1 1 1 1
```

Eg:- Perform the subtraction $645_{10} - 319_0$ in excess-3 using the 9's complement method

Sol:-

Excess-3 for 645 : 1001 0111 1000
" 319 : 0110 0100 1100
9's Complement of 319 : 1001 1011 0011

645
- 319
326

```
1 0 0 1   0 1 1 1   1 0 0 0
1 0 0 1   1 0 1 1   0 0 1 1
-------------------------------
1 0 0 1 1   0 0 1 0   1 0 1 1
→                     1          end around carry
-------------------------------
  0 0 1 1   0 0 1 0   1 1 0 0
```

Add 3 to correct 0011, 0010

```
  0 0 1 1   0 0 1 0   1 1 0 0
+ 0 0 1 1   0 0 1 1             Add 3
-----------------------------
  0 1 1 0   0 1 0 1   1 1 0 0
(-)           0 0 1 1           subtract (3)
-----------------------------
  0 1 1 0   0 1 0 1   1 0 0 1
```

Excess-3 for 326

G. NAVEEN
EEE

Sequential codes:—

In this codes each succeeding code is one binary number greater than its preceding code. this greatly aids mathematical manipulation of data

8-4-2-1, excess-3 are sequential, whereas 2421 & 5211 codes are not sequential

Alphanumeric codes:—

The codes which consists of both numbers and alphabetic characters are called Alphanumeric codes.

ASCII:- American Standard code for Information Interchange.

EBCDIC:- Extended Binary Coded Decimal Interchange code.

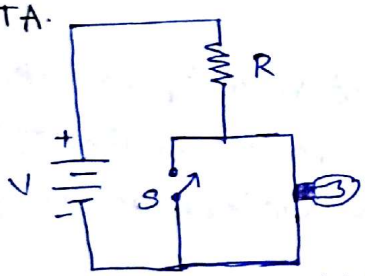
Logic Operations:—

Logic operators:-

To represent and slove arithmetic expressions we use arithmetic operations such as +, −, × and ÷. Similarly we can use logical operators to represent and slove logical expressions. There are three basic logical operators: NOT/INVERT, AND and OR.

NOT / INVERT:—

The inversion (or complementing or negation) operator is written as a bar over its argument. Some times it is written "NOT". Thus the inverse of A is $\bar{A}$ (or NOT A.



Input	output
Switch open (Low)	Lamp ON (High)
Switch close (High)	Lamp OFF (Low)

when the (S) Switch is 'open', lamp is 'ON', when the switch 'S' is 'close', lamp is 'OFF'

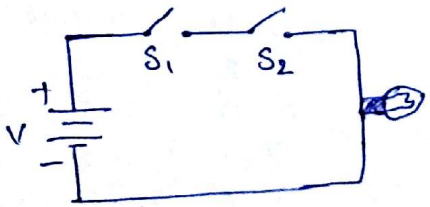
AND:—

It is denoted by '×' or '·'. These signs may be omitted.

Eg: A·B, A×B (or) AB has a same meaning may be read as A and B, or A times B.

→ A·B is high if A and B both are high otherwise Low

→ AND gate denotes lowest value of the i/p variable.



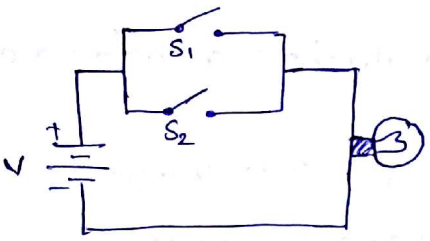
Input		output
S_1	S_2	
open (Low)	open (Low)	Lamp OFF (Low)
open (Low)	close (High)	Lamp OFF (Low)
close (High)	open (Low)	Lamp OFF (Low)
close (High)	close (High)	Lamp ON (High)

It is clear that lamp will be 'ON' only when both the switches are closed.

OR:—

It is written '+', '.', 'A+B' is read as A OR B, A+B is high if either A is high or B is high. or both are high.

OR gate denotes highest value of the i/p variable.



Input		Output
S_1	S_2	
Open (Low)	Open (Low)	Lamp OFF (Low)
Open (Low)	Close (High)	Lamp ON (High)
Close (High)	Open (Low)	Lamp ON (High)
Close (High)	Close (High)	Lamp ON (High)

If either (or) both of two separate switches S_1 OR S_2 are closed, the lamp will be ON. only when both S_1 & S_2 are OFF, the lamp will be OFF.

Logic Gates:—

Logic Gates are the fundamental building blocks of digital systems. the name Logic gate is derived from the ability of such a device to make decisions, in the sense that it produces one output level when some combinations of i/p levels are present and a different o/p level when other combinations of i/p levels are present.

there are 3 basic types of gates: AND, OR and NOT.

I/p's & O/p's of logic gates can occur only in two levels. these two levels are termed High & low. (or) TRUE and FALSE (or) ON and OFF (or) simply 1 & 0.

A table which lists all the possible combinations of i/p variables and the corresponding o/p is called "A Truth table". It shows how the logic circuits O/p responds to various combinations of logic levels at the i/p's

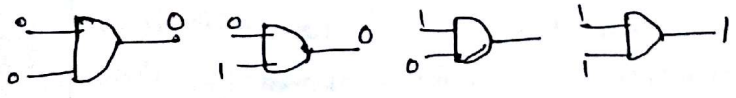
AND GATE:—

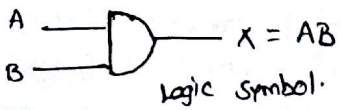
An AND gate has two (or) more i/p's but only one output the o/p assumes the logic 1 state, even if two or more of its inputs is in logic '1' state. Its o/p assumes the logic '0' state, only if each one of its inputs is in logic '0' state.

the AND gate is also called an "all" or "nothing gate"

The symbol for the AND operation is '.' (or) we use no symbol at all

→ Logic symbol and truth table





Logic symbol.

Truth table

Inputs		output
A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

G. NAVEEN (EEE)

with the i/p variables to the AND gate represented by A, B, C ----, the Boolean expression for the o/p can be written as $X = A \cdot B \cdot C$ ----, which is read as 'X' is equal to A and B and C ----', (or) X is equal to A dot B dot C ----

Discrete AND gates may be realized by using diodes (or) transistors.

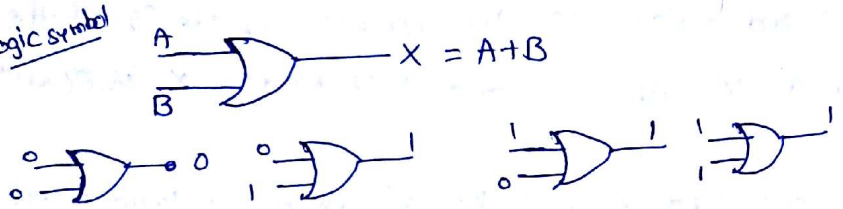
OR GATE:—

Like an AND gate an 'OR' gate may have two (or) more i/p's but only one o/p. the o/p assumes the logic '1' state, even if one of its i/p's is in logic 1 state. It's o/p assumes the logic '0' state, only when each one of its i/p's is in logic '0' state.

→ OR gate is also called an 'any' (or) "all gate". (or) "Inclusive OR gate"

The logic symbol and the truth table of a two i/p OR Gate are.

logic symbol



Truth table

i/p's		o/p
A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

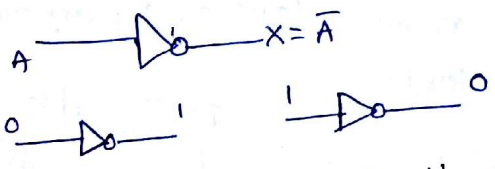
with the i/p variables to the OR gate represented by A, B, C ----, the Boolean expression for the o/p can be written as $X = A+B+C$ ---- This is read as X is equal to A or B or C or ----' (or) X is equal to A Plus B plus C ----!

NOT GATE:—

A NOT gate is also called an Inverter, The inverter performs a basic logic function called "Inversion" (or) Complementation".

It has only one i/p and one o/p. It is a device whose o/p is always the complement of its i/p. That is the o/p of a NOT Gate assumes the logic 1 gate. when its i/p is in Logic '0' state and vice-versa.

The logic symbol and truth table of an Inverter



Truth table

A	X
0	1
1	0

they symbol for NOT operation is '—' (bar).

ie if the i/p is A. the o/p X is expressed as $\bar{A} = X$

Logic circuits of any complexity can be realized using only AND, OR and NOT gates

Logic circuits which use these three gates only are called AND/OR/INVERT ie "AOI" Logic circuits

Logic circuits which use AND gates & OR gates only are called 'AO' logic circuits (AND/OR)

Universal Gates:—

Any logic circuit can be designed by using only three basic gates (AND, OR and NOT). There are two universal gates (NAND & NOR) each of which can also realize logic circuits single handedly.

∴ the NAND and NOR gates are called "universal building blocks".

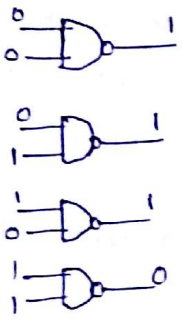
Both NAND and NOR gates can perform all the three basic logic functions (AND, OR and NOT).

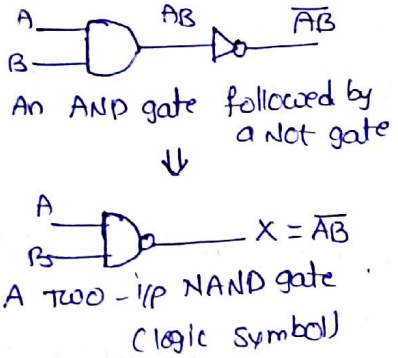
∴ AOI logic can be converted to NAND logic (or) NOR logic.

NAND Gate:—

NAND means NOT AND, i.e the AND output is NOTed. So, a NAND gate is a combination of an AND gate and a NOT gate. the expression for the o/p of the NAND gate can be written as $X = \overline{ABC\cdots}$ and it is read as 'X' is equal to A·B·C ··· whole bar

the o/p is logic '0' only when each of the i/p's assumes a logic '1'. For any other combination of i/p's, the o/p is logic '1'





Truth Table

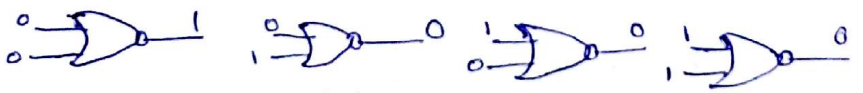
i/p's		o/p
A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

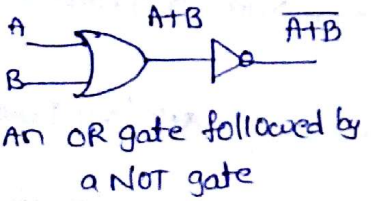
NOR Gate:—

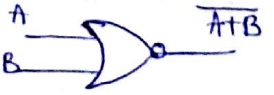
NOR means NOT OR, i.e the OR output is NOTed. So a NOR gate is a combination of an OR gate and NOT gate.

the expression for the output of the NOR gate is, $X = \overline{A+B+C+\cdots}$ and is read as 'X' is equal to A plus B plus C plus ··· whole bar

The o/p is logic 1, when all i/p's assumes a logic 0. For any other combination of i/p's, the o/p is logic '0' level.







Logic symbol

Truth Table

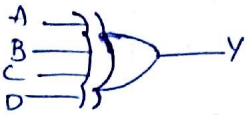
i/p's		o/p
A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

G. NAVEEN
(EEE)

Exclusive - OR Gate:-

The EX-OR gate is an abbreviation for Exclusive-OR gate. An EX-OR gate has two (or) more i/p's and one o/p as indicated by the standard logic symbol.

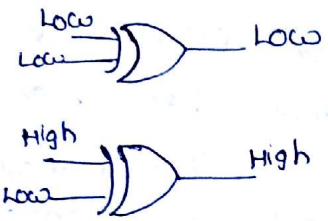
Two i/p EX-OR



Four i/p EX-OR

It recognizes only the words that have an odd No. of 1's. This means that for odd number of ones, o/p of EX-OR gate is high.

logic operation for a two i/p EX-OR gate for all 4 i/p combinations.

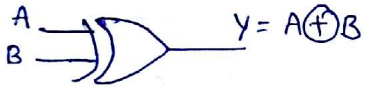


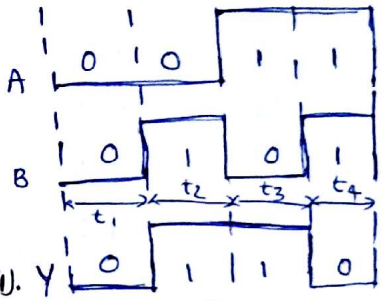
Truth Table

I/p's		O/p
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

The Truth Table can be expanded for any number of i/p's, however, regardless of the No. of i/p's, the o/p is high only when odd number of i/p's are high

Let us discuss the operation of an EX-OR gate with pulsed inputs



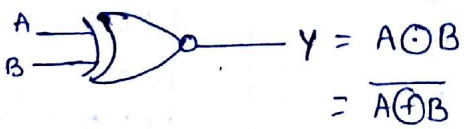


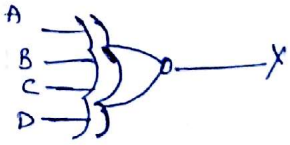
Timing diagram.

During time interval t_1, both i/p's A & B are Low(0), making o/p Low, During time interval t_2 input A is low but i/p B is High. Hence odd number of i/p's High making o/p High(1). During t_3, i/p A is HIGH(1) and i/p B is LOW(0), making o/p HIGH(1). During t_4, both i/p's are HIGH. Hence even no. of i/p's HIGH making o/p Low(0).

Exclusive NOR Gate:-

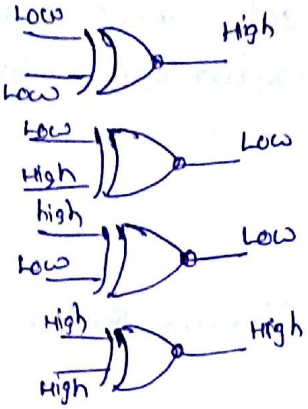
The term EX-NOR is a contraction of NOT-X-OR, NOT Exclusive OR gate. It is logically equivalent to an EX-OR gate followed by an inverter. An EX-NOR has two (or) more i/p's and one o/p, as indicated by the standard logic symbol





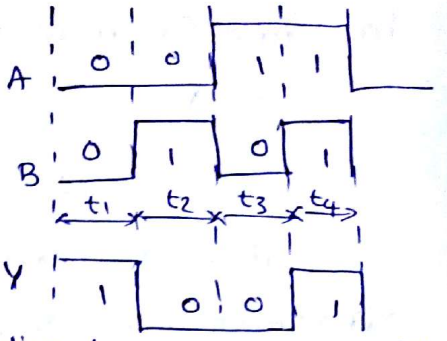
It recognizes only the words that have an even No. of ones, and i/p's having zero's. i.e for even no. of ones at the i/p, (or) i/p's having all zero's, the o/p of EX-NOR gate is high.

logic operation for a two i/p EX-NOR gate for all four possible i/p combinations



Truth Table

I/p's		O/p
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1



operation of an EX-NOR gate with Pulsed i/p's

GRAY CODE:- (Reflective code)

A Gray code is a non-weighted, or reflective unit distance code. In unit-distance code, bit Patterns for two consecutive numbers differ in only one bit position. These codes are also called "cyclic codes". In it is the most popular of the unit distance codes and is not suitable for arithmetic operations.

Reflection of Gray codes.

Gray code.				Decimal	4-bit binary
1-Bit	2-Bit	3-Bit	4-Bit		
0	00	000	0000	0	0000
1	01	001	0001	1	0001
	11	011	0011	2	0010
	10	010	0010	3	0011
		110	0110	4	0100
		111	0111	5	0101
		101	0101	6	0110
		100	0100	7	0111
			1100	8	1000
			1101	9	1001
			1111	10	1010
			1110	11	1011
			1010	12	1100
			1011	13	1101
			1001	14	1110
			1000	15	1111

We can obtain an N-bit Gray code from an N-1 bit Gray code by reflecting the N-1 bit code about an axis at the end of the code and putting the MSB of '0' above the axis and the MSB of '1' below the axis

G.NAVEEN
(EEE)

Advantage:—

It is ease of conversion to and from binary

Disadvantage:—

It is not suitable for mathematical operations because it is not a sequential code

Applications:—

Gray codes are used in instrumentation and data acquisition systems. where linear (or) angular displacement is measured. They are also used in shaft encoders. I/O devices, A/D converters and other peripheral equipment.

Binary to GRAY conversion:—

If an n-bit binary number is represented by $B_n . B_{n-1} \cdots B_1$ and its Gray code equivalent by $G_n G_{n-1} \cdots G_1$, where B_n and G_n are the MSBs then the Gray code bits are obtained from the binary code as

$$G_n = B_n \qquad G_{n-1} = B_n \oplus B_{n-1} \qquad G_{n-2} = B_{n-1} \oplus B_{n-2} \cdots \qquad G_1 = B_2 \oplus B_1$$

where $\oplus$ stands for the Exclusive OR (X-OR) operation explained below

The conversion procedure is as follows.

1. Record the MSB of the binary as the MSB of the Gray code.
2. Add the MSB of the binary to the next bit in binary, recording the sum and ignoring the carry, if any i.e x-OR the bits. This sum is the next bit of the Gray code.
3. Add the 2nd bit of the binary to the 3rd bit of the binary, the 3rd bit to the 4th bit and so on...
4. Record the successive sums as the successive bits of the Gray code until all the bits of the binary number are exhausted.

⇒ Another way to convert a binary number to its Gray code is to Exclusive OR (i.e to take the modulo sum of) the bits of the binary number with those of the binary number shifted one position to the right. The LSB of the shifted number is discarded and the MSB of the Gray code number is the same as the MSB of the original binary number

eg:- convert the binary 1001 to the Gray code

Sol: (a) Binary 1 ⊕→ 0 ⊕→ 0 ⊕→ 1

Gray 1 1 0 1

(b) Binary (or) 1 0 0 1

Shifted binary 1 0 0 1

Gray. 1 1 0 1

② **eg:** 1 0 1 1 1 0 1 1

Sol: 1 ⊕→ 0 ⊕→ 1 ⊕→ 1 ⊕→ 1 ⊕→ 0 ⊕→ 1 ⊕→ 1

Gray code: 1 1 1 0 0 1 1 0

Gray to Binary conversion:-

If an n-bit Gray number is represented by $G_n, G_{n-1} \cdots G_1$ and its binary equivalent by $B_n, B_{n-} \cdots B_1$, then binary bits are obtained from Gray bits as follows.

$B_n = G_n \quad B_{n-1} = B_n \oplus G_{n-1} \quad B_{n-2} = B_{n-1} \oplus G_{n-2} \cdots \quad B_1 = B_2 \oplus G_1$

The conversion procedure is:

1. The MSB of the binary number is the same as the MSB of the Gray code number. record it.
2. Add the MSB of the binary to the next significant bit of the Gray code, ie X-OR them; record the sum and ignore the carry.
3. 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. and so on. each time recording the sum and ignoring the carry.
4. continue this till all the Gray bits are exhausted. the sequence of bits that has been written down is the binary equivalent of the Gray code number.

Eg:- convert Gray code 1101 to binary

Sol:

Gray	1	1	0	1
	1 ⊕	1 ⊕	1 ⊕	1
Binary	1	0	0	1

Eg: Convert the following into Gray Number

① $3A7_{16}$

0011 1010 0111
0011 1010 0111
0010 0111 0100 (Gray)

② 527_8

101010111
111111100 (Gray)

Eg:- Convert the Gray number 10110010 into

(i) hex (ii) Octal (iii) decimal

Sol:- 1 0 1 1 0 0 1 0 → Gray
1 1 0 1 1 1 0 0 → binary

① hex 1101 1100
D C
DC_{16}

② octal:
011 011 100
3 3 4
334_8

③ Decimal

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	0	1	1	1	0	0

$1\times2^7 + 1\times2^6 + 1\times2^4 + 1\times2^3 + 1\times2^2$
$128 + 64 + 16 + 8 + 4 = 220_{10}$

G.NAVEEN

Sum of Product form:-

The words sum and product are derived from the symbolic representations of the OR (sum) and AND (product) functions by + and . resp.

But we realize that these are not arithmetic operator's in the usual sense.

→ A product term is any group of literals that are ANDed together.

eg: ABC, xy & so on...

→ A sum term is any group of literals that are ORed together eg:- A+B+C, x+y.....

→ A sum of products (SOP) is a group of product terms ORed together.

eg:- ① $f(A,B,C) = ABC + A\bar{B}\bar{C}$ (sum; product terms)

② $f(P,Q,R,S) = \bar{P}Q + QR + RS$ (sum; product terms)

→ The sum of products expression consists of two (or) more product terms (AND) that are ORed together

→ Each product term consists of one or more literals appearing in either complemented (or) un complemented form.

→ eg:- In this expression $ABC + AB'c'$, the first product term contains literals A, B and C, in their uncomplemented form

→ The second product term contains B and C in their complemented form

→ The sum of product form is also known as "normal form" (or) "Disjunctive normal formula".

Product of sum form:-

A product of sums is any groups of sum terms ANDed together.

eg:- ① $f(A,B,C) = (A+B).(\bar{B}+C)$ (product; sum terms.)

② $f(P,Q,R,S) = (P+Q).(R+\bar{S}).(P+S)$ (product; sum terms.)

→ These product of sums expressions consists of two (or) more sum terms (OR) that are ANDed together

→ Each sum term consists of one (or) more literals appearing in either complemented (or) un complemented form.

→ The product of sum form is also known as "conjunctive normal form" (or) "conjunctive normal formula".

Canonical form (standard SOP and POS forms):—

The canonical forms are the special cases of SOP forms. These are also known as standard SOP and POS forms.

Standard SOP form (or) Minterm canonical form:—

In SOP form, all the individual terms do not involve all literals.

→ for example, in expression $AB + A\bar{B}\bar{C}$ the first product term do not contain literal 'C'.

→ If each term in SOP form contains all the literals then the SOP form is known as standard (or) canonical SOP form.

→ Each individual term in the standard SOP form is known as "minterm".

∴ canonical SOP form is also known as "minterm canonical form".

→ In expression $AB\bar{C} + ABC + \bar{A}BC + A\bar{B}C$ all the literals are present in each product term.

→ one standard sum of products expression is as shown in fig.

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

each product term consists of all literals in either complemented form (or) uncomplemented form.

Standard POS Form (or) Maxterm canonical Form:—

→ If each term in POS form contains all the literals then the POS form is known as "standard" (or) "canonical POS form".

→ Each individual term in the standard POS form is called "maxterm".

∴ canonical POS form is also known as "maxterm canonical form".

→ one standard product of sum's expression is as.

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

Each sum term consists of all literals in either complemented form (or) uncomplemented form.

Converting Expressions in Standard SOP (or) POS Forms:—

Sum of products form can be converted to standard sum of products by ANDing the terms in the expression with terms formed by ORing literals and its complement which are not present in the form.

→ for example, a three literal expression with literals A, B and C.

→ If there is a term AB, where C is missing, then we form term $(C+\bar{C})$ and AND it with AB.

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

G.NAVEEN

Steps to convert SOP to standard SOP form:-

Step 1:- Find the missing literal in each product term if any.

Step 2:- AND each product term having missing literals with terms form by ORing the literal and its complement.

Step 3:- Expand terms by applying distributive law and reorder the literals in the product terms.

Step 4:- Reduce the expression by omitting repeated product terms if any.

Because $A+\bar{A}=A$.

Eg:- Convert the given expression in standard sop form $f(A,B,C) = AC+AB+BC$

Sol:- Step 1:- Find the missing literals in each product term.

$f(A,B,C) = \underbrace{AC}_{\text{literal B is missing}} + \underbrace{AB}_{\text{literal C is missing}} + \underbrace{BC}_{\text{literal A is missing}}$

Step 2:- AND product term with (missing literal + it's complement)

$f(A,B,C) = \overbrace{AC \cdot (B+\bar{B}) + AB \cdot (C+\bar{C}) + BC(A+\bar{A})}^{\text{original product terms}}$

missing literals and their complements.

Step 3:- Expand the terms and reorder literals

Expand: $f(A,B,C) = ACB + AC\bar{B} + ABC + AB\bar{C} + ABA + BC\bar{A}$

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

Step 4:- Omit repeated product terms.

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

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

Eg:- Convert the given expression in standard SOP form $f(A,B,C) = A + ABC$

Sol:- (1) $f(A,B,C) = A + ABC$ → literals B and C are missing.

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

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

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

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

Steps to convert POS to standard POS form:-

Step 1:- Find the missing literals in each sum term if any

Step 2:- OR each sum term having missing literal/s with term/s form by ANDing the literal and it's complement.

Step 3:— Expand the terms by applying distributive law and reorder the literals in the sum terms.

Step 4:— Reduce the expression by omitting repeated sum terms if any

$A \cdot A = A$ (or) $A + A = A$

Eg:— Convert the given expression in standard POS form

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

Sol:—

Step 1:— $f(A,B,C) = A+B \quad B+C \quad A+C$

$A+C$ — literal B is missing

$B+C$ — " A " "

$A+B$ — " C "

Step 2:

original products

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

missing literals & their complements.

Step 3:— Expand: $f(A,B,C) = (A+B+C)(A+B+\bar{C})(B+C+A)(B+C+\bar{A}) + (A+C+B)(A+C+\bar{B})$

Reorder:— $f(A,B,C) = (A+B+C)(A+B+\bar{C})(A+B+C) \cdot (\bar{A}+B+C)(A+B+C)(A+\bar{B}+C)$

Step 4:— Omit repeated terms.

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

repeated sum terms.

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

Eg:- Convert the given expression in standard POS form

$Y = A \cdot (A+B+C)$

Sol:- ① $f(A,B,C) = A \cdot (A+B+C)$

literals B & C are missing

② $f(A,B,C) = (A + B\cdot\bar{B} + C\cdot\bar{C})(A+B+C)$

③ Expand:- $f(A,B,C) = (A+B+C)(A+\bar{B}+C)(A+B+\bar{C})(A+\bar{B}+\bar{C})(A+B+C)$

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

④ Omit repeated sum terms

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

⑤ Expand

G.NAVEEN(EEE)

M-Notations: Minterms and Maxterms:-

→ Each individual term in standard SOP form is called minterm and each individual term in standard POS form is called maxterm.

→ the concept of minterms and max terms allows us to introduce a very convenient shorthand notations to express logical functions.

→ Below table gives the minterms and max terms for a three literal/variable logical function

→ where the number of minterms as well as maxterms is $2^3 = 8$

→ for an n-variable logical function there are 2^n minterms and an equal number of max terms.

Variables			minterms	maxterms
A	B	C	m_i	M_i
0	0	0	$\bar{A}\bar{B}\bar{C} = m_0$	$A+B+C = M_0$
0	0	1	$\bar{A}\bar{B}C = m_1$	$A+B+\bar{C} = M_1$
0	1	0	$\bar{A}B\bar{C} = m_2$	$A+\bar{B}+C = M_2$
0	1	1	$\bar{A}BC = m_3$	$A+\bar{B}+\bar{C} = M_3$
1	0	0	$A\bar{B}\bar{C} = m_4$	$\bar{A}+B+C = M_4$
1	0	1	$A\bar{B}C = m_5$	$\bar{A}+B+\bar{C} = M_5$
1	1	0	$AB\bar{C} = m_6$	$\bar{A}+\bar{B}+C = M_6$
1	1	1	$ABC = m_7$	$\bar{A}+\bar{B}+\bar{C} = M_7$

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

$= m_0 + m_1 + m_3 + m_4 + m_6 + m_7$

$= \Sigma m(0,1,3,4,6,7)$

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

$= M_1 + M_3 + M_6 = \pi M(1,3,6)$

where Σ denotes sum of product, while π denotes product of SUM.

Representation of Truth table for minterms & max terms:-

minterms:

A	B	C	Y	
0	0	0	0	
0	0	1	0	
0	1	0	1	← $\bar{A}B\bar{C}$
0	1	1	1	← $\bar{A}BC$
1	0	0	0	
0	0	1	0	
1	1	0	1	← $AB\bar{C}$
1	1	1	0	

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

$= m_2 + m_3 + m_6$

max terms:-

A	B	C	Y	
0	0	0	1	
0	0	1	1	
0	1	0	0	← $A+\bar{B}C$
0	1	1	1	
1	0	0	1	
1	0	1	0	← $\bar{A}+B+\bar{C}$
1	1	0	1	
1	1	1	1	

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

$= M_2 + M_5$

Complements of canonical formulae:—

A product of sums form derived from a truth table is logically equivalent to a sum of products form derived from truth table (above).

Let us write standard SOP and POS from from the truth table's.

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

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

Now by simplifying equation in the POS form we have,

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

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

Converting to standard SOP we have

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

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

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

Rearranging terms we have.

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

∴ we can say that POS & SOP derived from the same truth table are logically equivalent. In terms of minterms and maxterms we can then write.

$f(A,B,C) = m_0 + m_1 + m_3 + m_4 + m_6 + m_7 = M_2 + M_5$

$f(A,B,C) = \Sigma m(0,1,3,4,6,7) = \Pi M(2,5)$

from the above expressions we can easily notice that there is a complementary type of relationship b/w a function expressed in terms of max terms.

Eg:- four variables.

$f(A,B,C,D) = \Sigma m(0,2,4,6,8,10,12,14)$

then $f(A,B,C,D) = \Pi M(1,3,5,7,9,11,13,15)$

G.NAVEEN (EEE)

Eg: In an odd-parity scheme, which of the following words contain an error?

Sol: ① 10110111

Sol: The number of is in the word is even (6). So, this word has an error.

② 11101010

Sol: the number of is in the word is odd (5). ∴ there is no error.

Error-correcting code :-

A code is said to be an error-correcting code if the correct code word can always be deduced from an erroneous word.

For a code to be a single bit error-correcting code, the minimum distance of that code must be three (or) more. It can also detect (but not correct) two-bit errors.

The key to error correction is that it must be possible to detect and locate erroneous digits. If the location of has been determined, then by complementing the erroneous digit, the message can be corrected.

One type of error-correcting code is "Hamming code"

Hamming code :-

This code uses a number of parity bits (depending on the no. of information bits) located at certain positions in the code group

No. of Parity Bits :-

No. of parity bits depend on the number of information bits.

If the No. of information bits is 'x', then the No. of parity bits p is determined by

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

Eg: let $x = 4$ p is found by trail & error

let $p = 2$ then

$$2^p = 2^2 = 4$$

and $x + p + 1 = 4 + 2 + 1 = 7$

$\therefore 2^p \geq x + p + 1$, the relationship in eq will satisfied.

let p^3 $\quad 2^3 \geq x + p + 1$

$8 \geq 4 + 3 + 1 \quad \because 8 \geq 8.$

This value of p satisfies the relationship given in Equation.

$\therefore$ we can say 3 parity bits are required to provide ~~sigh~~ single error correction for four information bits

G. NAVEEN (EEE)

Location of the parity Bits in the code:-

From the above information for a 4-bit information we need 3-Parity bits. ∴ the total Number of bits - 7.

the right most bit is designated bit 1, the next bit is bit 2 - - - -

Bit 7, Bit 6, Bit 5, Bit 4, Bit 3, Bit 2, Bit 1

The Parity Bits are located in the positions that are numbered corresponding to ascending Powers of two (1, 2, 4, 8 - - - -)

∴ for 7-bit code, location for parity bit and information bit are

$D_7, P_6, D_5, P_4, D_3, P_2, P_1$

P_n → Particular Parity bit

D_n → Particular information bit.

Assigning values to parity Bits:-

How to determine 1 (or) 0 value to each parity bit.

Bit designation	D_7	D_6	D_5	P_4	D_3	P_2	P_1
Bit location	7	6	5	4	3	2	1
Binary location number	111	110	101	100	011	010	001
Information bits (D_n)							
Parity bits (P_n)							

Eg:- Encode the binary word 1011 into seven bit even parity Hamming code.

Sol:-

Step 1:- Find the no. of parity bits required

Let p=3, then $2^p = 2^3 = 8$

$x+p+1 = 4+3+1 = 8$.

∴ 3 Parity bits are sufficient.

∴ Total code bits = 4+3 = 7.

Step 2:- Construction of bit location table.

Bit designation	D_7	D_6	D_5	P_4	D_3	P_2	P_1
Bit location	7	6	5	4	3	2	1
Binary location number	111	110	101	100	011	010	001
Information bits	1	0	1		1		
Parity bits				0		0	1

Step 3:- Determine the parity bits

For P_1:- Bit locations 3, 5, and 7 have three 1's and ∴ to have an even parity P_1 must be 1.

For P_2:- Bit locations 3, 6, and 7 have two 1s and ∴ to have an even parity P_2 must be 0.

For P_3:- Bit locations 5, 6 and 7 have two 1's & ∴ to have even parity P_4 must be 0.

Step 4:- Enter the parity bits into the table to form a seven bit Hamming code

1 0 1 0 1 0 1

Eg:- Determine the single error-correcting code for the information code 1011 for odd parity.

Sol:- Step 1:- Find the no of parity bits required

let P = 4 $\quad 2^P \geq x+P+1$

$2^4 = 4+5+1$

$16 = 10$

∴ equation satisfied & 4 bits are sufficient.

∴ Total code bit = 5+4 = 9.

Step 2:-

Bit designation	D_9	P_8	D_7	P_6	D_5	P_4	D_3	P_2	P_1
Bit location	9	8	7	6	5	4	3	2	1
Binary location number	1001	1000	0111	0110	0101	0100	0011	0010	0001
Information bits	1		0	1	1		1		
Parity Bits		0				1		1	0

Step 3:-

For P_1:- Bit locations 3, 5, 7, & 9 have three 1's and ∴ to have odd parity P_1 must be 0

For P_2:- Bit locations 3, 6, 7 have two 1's " " " P_2 must be 1

For P_4:- Bit locations 5, 6 & 7 have two 1's " " " P_4 must be 1

For P_8:- Bit P_8 checks bit locations 8 & 9 and must be 0 to have an odd parity.

Step 4:-

Enter the parity bits into the table to form a nine bit Hamming code

1 0 0 1 1 1 1 1 0

G.NAVEEN (EEE)

Detecting and correcting an Error:-

For the detection, each parity bit along with its corresponding group of bits must be checked for proper parity. The correct result of individual parity check is marked by '0'. whereas wrong result is marked by '1'. After all parity checks, binary word is formed taking resulting bit for P_1 as LSB. this word gives bit location where error has occured. If word has all bits '0' then there is no error in the Hamming code.

Eg:- Assume that the even parity Hamming code in example (0110011) is transmitted and that 0100011 is received. the receiver does not know that was transmitted. Determine bit location where error has occurred using received code.

Sol:-

Step 1:- Bit location table

Bit designation	D_7	D_6	D_5	P_4	D_3	P_2	P_1
Bit location	7	6	5	4	3	2	1
Binary location number	111	110	101	100	011	010	001
Received code	0	1	0	0	0	1	1

Step 2:- check for parity bits

for P_1 :- P_1 checks locations 1, 3, 5 & 7

There is one 1 in the group

∴ parity checks for even parity is wrong ⟶ 1 (LSB)

for P_2 :- P_2 checks locations 2, 3, 6, 7

∴ there are two 1's in group

∴ parity check for even parity is correct ⟶ 0

for P_4 :- P_4 checks locations 4, 5, 6, 7 ⟶ (1)

the resultant word is 101 ⟶ 5

this says that the bit in the no. 5 location is in error. It is 0 and should be a 1.

the correct code is 0110011, which agrees with the transmitted code.

Eg:- the hamming code 101101101 is received. correct it if any error. there are four parity bits and odd parity is used.

Sol:

Bit designation	D_9	P_8	D_7	D_6	D_5	P_4	D_3	P_2	P_1
Bit location	9	8	7	6	5	4	3	2	1
Binary location number	1001	1000	0111	0110	0101	0100	0011	0010	0001
Received code.	1	0	1	1	0	1	1	0	1

for P_1 – 1,3,5,7,9 total is -4 odd Parity wrong — 1

P_2 – 2,3,6,7 total is -3 " is correct -0

P_4 – 4,5,6,7 total is -3 " " " 0

P_8 – 8,9 total is -1 " " " 0

∴ the resultant word is 0001.

This says that the bit in the number '1' location is in error. It is '1' and should be a '0'

∴ the correct code is 10110100.

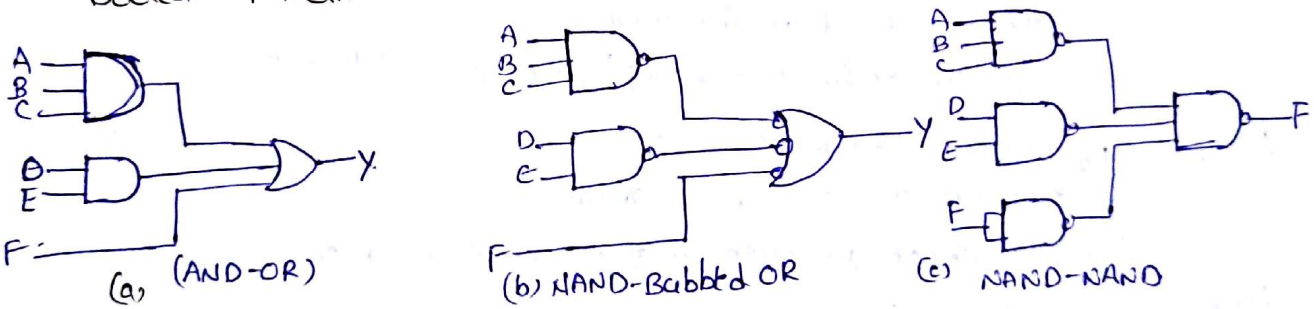
NAND-NAND Implementation:-

The implementation of Boolean function with NAND-NAND logic requires that the function be simplified in the sum of product form.

The relationship b/w AND-OR logic and NAND-NAND logic is Explained using following.

Consider a Boolean function $Y = ABC + DE + F$

The Boolean function can be implemented using AND-OR logic as shown in fig(a)



(a) (AND-OR) (b) NAND-Bubbled OR (c) NAND-NAND

Fig(b) shows the AND gates are replaced by NAND gates and the OR gate is replaced by a bubbled OR gate. The implementation is shown. It is equivalent to implementation shown in fig(a), because two bubbled-on the same line represent double inversion (Complementation) which is equivalent to having no bubble on the line. In case of single variable, F, the complemented variable is again complemented by bubble to produce the normal value of F.

→ Fig (c), the output NAND gate is redrawn with the conventional symbol. The NAND gate with same inputs gives complemented result; ∴ $\bar{F}$ is replaced by NAND gate with F input to its both inputs. Thus all the 3 implementations of Boolean function are Equivalent

From the above Example we can summarize the rules for obtaining the NAND-NAND logic diagram from a Boolean function as follows.

1. Simplify the given Boolean function and express it in Sum of product form (SOP)
2. Draw a NAND gate for each product term of the function that has two or more literals. The inputs to each NAND gate are the literals of the term. This constitutes a group of first-level gates.
3. If Boolean function includes any single literal (or) literals draw NAND gate for each single literal and connect corresponding literal as an i/p to the NAND gate.
4. Draw a single NAND gate in the second level, with inputs coming from outputs of first level gates.

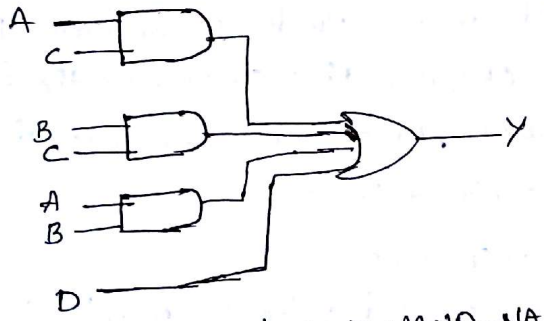
Eg:- Implement the following Boolean function with NAND-NAND logic.

$$Y = AC + ABC + \bar{A}BC + AB + D$$

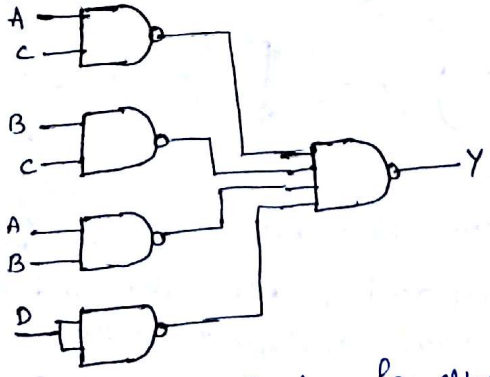
Sol:- Step 1:- Simplify the given Boolean function

$$Y = AC + ABC + \bar{A}BC + AB + D$$
$$= AC + BC(A + \bar{A}) + AB + D$$
$$= AC + BC + AB + D$$

Step 2:- Implement using AND-OR logic.



Step 3:- Convert AND-OR logic to NAND-NAND logic.

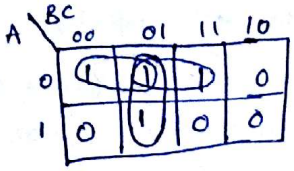


Eg:- Implement the following Boolean function with NAND-NAND logic

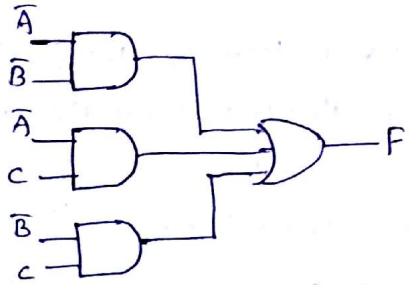
$$F = (A, B, C) = \Sigma m(0, 1, 3, 5)$$

Sol:- Step:1 Simplify the given Boolean function

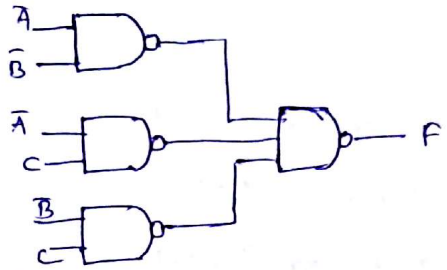
$$F = \bar{A}\bar{B} + \bar{A}C + \bar{B}C$$



Step2:- Implement Boolean function with AND-OR logic.



Step3:- Convert AND-OR logic to NAND-NAND logic



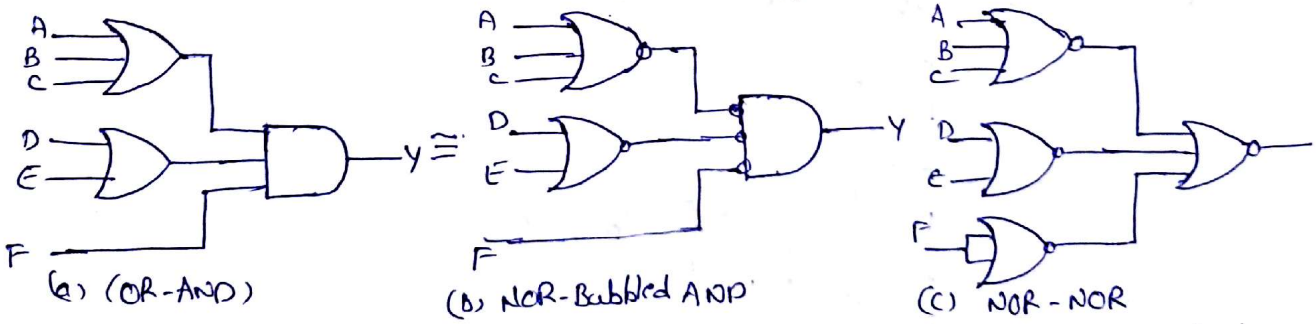
NOR-NOR Implementation:—

The NOR function is a dual of the NAND function. For this reason, the implementation procedures and rules for NOR-NOR logic are the duals of the corresponding procedures and rules developed for NAND-NAND logic.

The Implementation of a Boolean function with NOR-NOR logic requires that the function be simplified in the product of sum form (POS). In POS we implement all sum terms using OR gates. this constitutes the first level. In second level all sum terms are logically ANDed using AND gate. the relationship b/w OR-AND logic & NOR-NOR is explained by using.

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

The function can be implemented using OR-AND logic, as shown in fig (a),



(a) (OR-AND) (b) NOR-Bubbled AND (c) NOR-NOR

→ fig (b) shows the OR gates are replaced by NOR gates and the AND gate is replaced by a bubbled AND gate. fig(a) is equivalent to fig (b).

→ Fig (c). the o/p NOR gate is redrawn with the conventional symbol. the NOR gate with same i/p's gives complemented result. ∴ $\bar{F}$ is replaced by NOR gate with F i/p to its both inputs.

* → Summarise the rules for obtaining the NOR-NOR logic diagram from a Boolean function as follows.

1. Simplify the given Boolean function and express it in POS form.
2. Draw a NOR gate for each sum term of the function that has two or more literals. the inputs to each NOR gate are the literals of the term. This constitute a group of first level gates.
3. If Boolean function includes any single literal (or literals, draw NOR gate for each single literal and connect corresponding literal as an input to the NOR gate.
4. Draw a single NOR gate in the second level, with inputs coming from outputs of first level gates

Eg:- Implement the following Boolean function with NOR-NOR logic

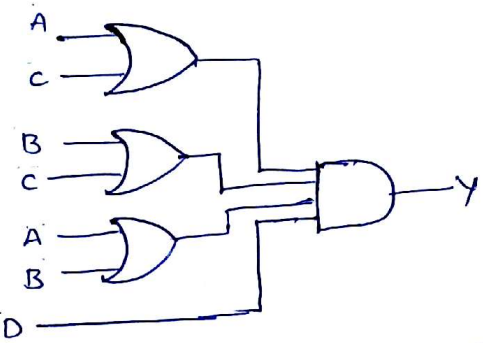
$Y = AC + BC + AB + D$ $\quad \because AB = A+B$

Sol:-

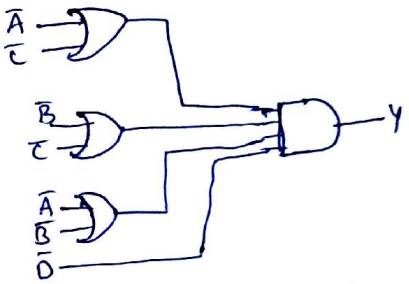
Step:1:- Boolean function is in POS form

$Y = (A+C)(B+C)(A+B)D$ (or) $\bar{Y} = (\bar{A}+\bar{C})(\bar{B}+\bar{C})(\bar{A}+\bar{B})\cdot\bar{D}$ (using Dual. ~~Demorgan's~~ theorem)

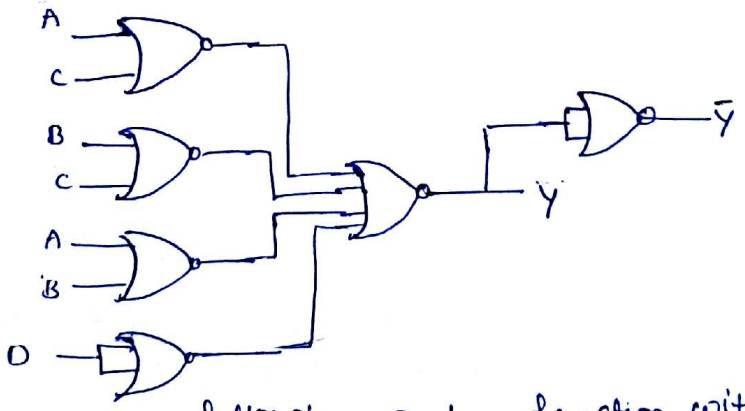
Step 2:- Implement Boolean function with OR-AND logic



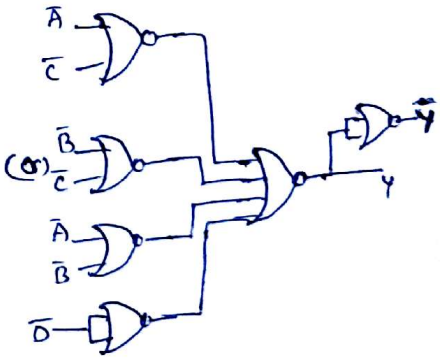
(or)



Step 3:- Convert OR-AND logic to NOR-NOR logic



(or)



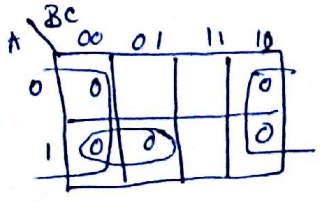
Eg:- Implement the following Boolean function with NOR-NOR logic

$F = (A,B,C) = \pi M(0,2,4,5,6)$

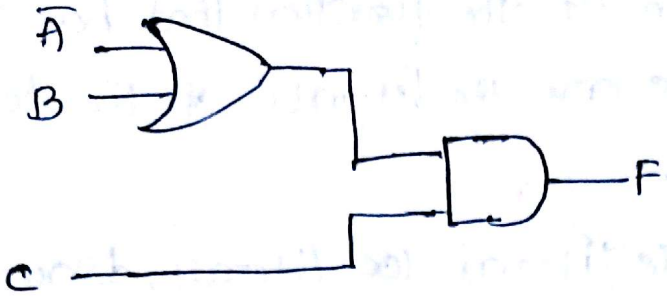
Sol:-

Step 1:-

$F = (A+B)C$.



Step 2:- Implement Boolean function with OR-AND log



Step 3:- convert OR-AND logic to NOR-NOR logic