UNIT-1

Review of Number systems & codes

There are four types of number systems in digital electronics

① Decimal number system (Base 10)
② Binary number system (Base 2)
③ Octal number system (Base 8)
④ Hexa-decimal number system (Base 16)

Base is also known as radix

If the base of any number system describes the no. of distinct symbols and the highest digit/number in that number system.

Ex: In hexadecimal number system, there are 16 symbols they are

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

* List the first 20 numbers in hexadecimal number system

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, 10, 11, 12, 13

* List the first 15 numbers in Base 12 system

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, 10, 11, 12

* List the first 10 numbers in octal number system

0, 1, 2, 3, 4, 5, 6, 7, 10, 11

Base Conversions:

To Convert $(\ )_{10}$ to $(\ )_r$ perform the successive division of the given decimal number with required base upto the coefficient is less than the base.

Ex: Convert $(48)_{10} = (?)_2$

2	48
	24 - 0
	12 - 0
	6 - 0
	3 - 0
	1 - 1

$\rightarrow (48)_{10} = (110000)_2$

*Convert $(71)_{10} = (?)_2$

2	71
	35 - 1
	17 - 1
	8 - 1
	4 - 0
	2 - 0
	1 - 0

$(71)_{10} = (1000111)_2$

*Convert $(524)_{10} = (?)_2$

2	524
	262 - 0
	131 - 0
	65 - 1
	32 - 1
	16 - 0
	8 - 0
	4 - 0
	2 - 0
	1 - 0

$(524)_{10} = (1000001100)_2$

*Convert $(461)_{10} = (?)_2$

	461
	230 - 1
	115 - 0
	57 - 1
	28 - 1
	14 - 0
	7 - 0
	3 - 1
	1 - 1

$(461)_{10} = (111001101)_2$

*Convert $(10.125)_{10} = (?)_2$

2	10
2	5 - 0
2	2 - 1
	1 - 0

$.125 \times 2 = 0.250$
$.250 \times 2 = 0.500$
$.500 \times 2 = 1.000$

$(10.125)_{10} = (1010.001)_2$

*Convert $(39.225)_{10} = (?)_2$

2 | 39	
2 | 19 - 1	
2 | 9 - 1	
2 | 4 - 1	
2 | 2 - 0	
1 - 0	

$.225 \times 2 = 0.450$
$.450 \times 2 = 0.900$
$.900 \times 2 = 1.800$
$.800 \times 2 = 1.600$

$(39.225)_{10} = (100111.0011)_2$

*Convert $(63.525)_{10} = (?)_2$

2 | 63	
2 | 31 - 1	
2 | 15 - 1	
2 | 7 - 1	
2 | 3 - 1	
1 - 1	

$.525 \times 2 = 1.050$
$.050 \times 2 = 0.100$
$.100 \times 2 = 0.200$
$.200 \times 2 = 0.400$
$.400 \times 2 = 0.800$

$(63.525)_{10} = (111111.1000)_2$

⇒ Conversion of decimal to octal

*Convert $(25)_{10} = (?)_8$

8 | 25	
3 - 1	

$(25)_{10} = (31)_8$

*Convert $(93)_{10} = (?)_8$

8 | 93	
8 | 11 - 5	
1 - 3	

$(93)_{10} = (135)_8$

*Convert $(125)_{10} = (?)_8$

8 | 125	
8 | 15 - 5	
1 - 7	

$(125)_{10} = (175)_8$

*Convert $(154.125)_{10} = (?)_8$

8 | 154	
8 | 19 - 2	
2 - 3	

$.125 \times 8 = 1.000$

$(154.125)_{10} = (232.1)_8$

*Convert $(23.225)_{10} = (?)_8$

8 | 23	
2 - 7	

$.225 \times 8 = 1.800$
$.800 \times 8 = 6.400$
$.400 \times 8 = 3.200$
$.200 \times 8 = 1.600$

$(23.225)_{10} = (27.1631)_8$

⇒ Conversion from decimal to hexa decimal ④

* Convert $(31)_{10} = (?)_{16}$

16 | 31	
1 - F	

$(31)_{10} = (1F)_{16}$

* convert $(49)_{10} = (?)_{16}$

16 | 49	
3 - 1	

$(49)_{10} = (31)_{16}$

* convert $(62)_{10} = (?)_{16}$

16 | 62	
3 - E	

$(62)_{10} = (3E)_{16}$

* Convert $(74)_{10} = (?)_{16}$

16 | 74	
4 - A	

$(74)_{10} = (4A)_{16}$

* Convert $(70.250)_{10} = (?)_{16}$

16 | 70	
4 - 6	

$(70.250)_{10} = (46.4)_{16}$

$\cdot 250 \times 16 = 4.000$

* $(92.150)_{10} = (?)_{16}$

16 | 92	
5 - 6	

$(92.150)_{10} = (5C.2666)_{16}$

$\cdot 150 \times 16 = 2.400$
$\cdot 400 \times 16 = 6.400$
$\cdot 400 \times 16 = 6.400$
$\cdot 400 \times 16 = 6.400$

* Convert $(10.125)_{10} = (?)_4$

4 | 10	
2 - 2	

$(10.125)_{10} = (22.02)_4$

$\cdot 125 \times 4 = 0.500$
$\cdot 500 \times 4 = 2.000$

8 to 10 conversions

Binary to decimal conversion

→ $(\overset{5}{1}\overset{4}{1}\overset{3}{0}\overset{2}{0}\overset{1}{0}\overset{0}{0})_2 = (?)_{10}$

$1\times2^5+1\times2^4+0\times2^3+0\times2^2+0\times2^1+0\times2^0$

$\Rightarrow 32+16$

→ $\boxed{(48)_{10}}$

→ $(\overset{9}{1}\overset{8}{0}\overset{7}{0}\overset{6}{0}\overset{5}{0}\overset{4}{1}\overset{3}{1}\overset{2}{0}\overset{1}{0}\overset{0}{0})_2$

$1\times2^9+0+1\times2^3+1\times2^2$

$512+8+4$

$\boxed{(524)_{10}}$

→ $(1000111)_2$

$1\times2^6+0+1\times2^2+1\times2^1+1\times2^0$

$64+4+2+1$

$68+3$

$\boxed{(71)_{10}}$

→ $(111001101)_2$

$1\times2^8+1\times2^7+1\times2^6+1\times2^3+1\times2^2+1\times2^0$

$256+128+64+8+4+2+1$

$\boxed{(461)_{10}}$

→ $(\overset{3}{1}\overset{2}{0}\overset{1}{1}\overset{0}{0}.\overset{-1}{0}\overset{-2}{0}\overset{-3}{1})_2 = (?)_{10}$

$1\times2^3+1\times2^1+0\times2^{-1}+1\times2^{-3}$

$8+2+\frac{1}{2^3} \Rightarrow 10+0.125$

$\boxed{(10.125)_{10}}$

→ $(\overset{5}{1}\overset{4}{0}\overset{3}{0}\overset{2}{1}\overset{1}{1}\overset{0}{1}.\overset{-1}{0}\overset{-2}{0}\overset{-3}{1}\overset{-4}{1})_2 = (?)_{10}$

$1\times2^5+1\times2^4+1\times2^3+1\times2^2+1\times2^0+1\times2^{-3}+1\times2^{-4}$

$\Rightarrow 32+4+2+1+\frac{1}{2^3}+\frac{1}{2^4}$

$\Rightarrow 39+0.125+0.0625 \Rightarrow (39.1875)_{10} \approx \boxed{(39.225)}$

→ $(\overset{5}{1}\overset{4}{1}\overset{3}{1}\overset{2}{1}\overset{1}{1}\overset{0}{1}.\overset{-1}{1}\overset{-2}{0}\overset{-3}{0}\overset{-4}{0})_2 = (?)_{10}$

$1\times2^5+1\times2^4+1\times2^3+1\times2^2+1\times2^1+1\times2^0+1\times2^{-1}$

$32+16+8+4+2+1+0.5 \Rightarrow \boxed{(63.5)_{10}}$

Octal to decimal conversion:

* $(31)_8 = (?)_{10}$

$8^1 \times 3 + 1 \times 8^0$

$24 + 1$

$\boxed{(25)_{10}}$

* $(135)_8 = (?)_{10}$

$1 \times 8^2 + 3 \times 8^1 + 5 \times 8^0$

$64 + 24 + 5$

$\boxed{(93)_{10}}$

* $(23 \overset{-1}{2}.1)_8 = (?)_{10}$

$2 \times 8^2 + 3 \times 8^1 + 2 \times 8^0 + 1 \times 8^{-1}$

$128 + 24 + 2 \times \frac{1}{8}$

$154 + 0.125$

$\boxed{154.125}$

* $(175)_8 = (?)_{10}$

$1 \times 8^2 + 7 \times 8^1 + 5 \times 8^0$

$64 + 56 + 5$

$\boxed{(125)_{10}}$

* $(27.1631)_8 = (?)_{10}$

$2 \times 8^1 + 7 \times 8^0 + 1 \times 8^{-1} + 6 \times 8^{-2} + 3 \times 8^{-3} + 8^{-4} \times 1$

$16 + 7 + 0.125 + 0.09375 + 0.00585 + 0.00024$

$\boxed{(23.2248)_{10}}$

* hexadecimal to decimal

→ $(1F)_{16} = (?)_{10}$

$1 \times 16^1 + 15 \times 16^0$

$16 + 15 \Rightarrow \boxed{(31)_{10}}$

→ $(3E)_{16} = (?)_{10}$

$3 \times 16^1 + E \times 16^0$

$48 + 14 \times 1$

$\boxed{(62)_{10}}$

→ $(4A)_{16} = (?)_{10}$

$4 \times 16^1 + A \times 16^0$

$64 + 10 \times 1$

$\boxed{(74)_{10}}$

→ $(31)_{16} = (?)_{10}$

$16^1 \times 3 + 16^0 \times 1$

$48 + 1 = \boxed{49}_{10}$

→ $(46.4)_{16}$

$4 \times 16^1 + 6 \times 16^0 + 4 \times \frac{1}{16}$

$64 + 6 + 0.250 \Rightarrow \boxed{70.250}_{10}$

* $(5C.2666)_{16} = (?)_{10}$

$5 \times 16^1 + 12 \times 16^0 + 2 \times 16^{-1} + 6 \times 16^{-2} + 6 \times 16^{-3} + 6 \times 16^{-4}$

$80 + 12 + 0.125 + 0.0234 + 0.00146 + 0.00009$

$\Rightarrow (92.1498)_{10} \approx \boxed{(92.150)_{10}}$

Base 4 to Base 10

* $(22.02)_4$

$2\times4^1 + 2\times4^0 + 0 + 2\times4^{-2}$

$8 + 2 + 0 + 2\times\frac{1}{16}$

$10 + 0.125$

$(10.125)_{10}$

8-8 Conversions:

In binary to octal conversion segment the given binary number as group and each group should contain 3 bit. Replace each group with its octal equivalent.

3-bit binary code ↑ octal

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

$2^3 = 8$

* Convert $(1011010)_2$ to $(?)_8$

Sol

001 | 011 | 010

1 3 2

$(132)_8$

* $(1101011011101)_2$ to $(?)_8$

001 | 101 | 011 | 011 | 101

1 5 3 3 5

$(15335)_8$

* $(1110111111110110.0110111)_2 = (?)_8$. (8)

111 | 011 | 111 | 110 | 110.011 | 011 | 100

7 3 7 6 6.3 3 4

$(73766.334)_8$

Binary to hexadecimal :-

4 bit binary code

2^3	2^2	2^1	2^0	
8	4	2	1	
0	0	0	0	- 0
0	0	0	1	- 1
0	0	1	0	- 2
0	0	1	1	- 3
0	1	0	0	- 4
0	1	0	1	- 5
0	1	1	0	- 6
0	1	1	1	- 7
1	0	0	0	- 8
1	0	0	1	- 9
1	0	1	0	- 10 - A
1	0	1	1	- 11 - B
1	1	0	0	- 12 - C
1	1	0	1	- 13 - D
1	1	1	0	- 14 - E
1	1	1	1	- 15 - F

$2^4 = 16$

* $(11001010)_2 = (?)_{16}$

$\underline{\text{Sol}}$

11 0010 10
C | A

$(CA)_{16}$

* $(1010111010101011)_2 = (?)_{16}$

0010 | 1011 | 1010 | 1011
2 B A B

$(2BAB)_{16}$

* $(10\,11010\,10.10\,10\,10\,111)_2 = (?)_{16}$

0001 | 0110 | 1010 . 1010 | 1011 | 1000
1 6 A A B 8

$(16A.AB8)_{16}$

00010000 00010001

Octal to Binary conversion:

* In octal to Binary conversion each digit of octal number should be replaced with its 3-bit binary equivalent

* $(132)_8$

1 | 3 | 2
001 | 011 | 010

$(001011010)_2$.

* $(16555)_8$

1 | 6 | 5 | 5 | 5
001 | 110 | 101 |

* $(7376\,6.334)_8$

7 | 3 | 7 | 6 | 6 . 3 | 3 | 4
111 | 011 | 111 | 110 | 110 . 011 | 011 | 100

$(111011111110110 . 011011)_2$.

Hexadecimal to binary:

* $(1011)_{16} = (?)_2$

1	0	1	1
0001	0000	0001	0001

$(0001\,0000\,0001\,0001)_2$

* $(CA)_{16} = (?)_2$

C	A
1100	1010

$(11001010)_2$

Octal to hexadecimal Conversion:

* to Convert Octal to hexadecimal convert the octal number to binary and then to hexadecimal.

→ Convert $(145)_8 = (?)_{16}$.

(a) $(145)_8 = (?)_2$

1	4	5
001	100	101

$(001100101)_2$.

(b)

0000	0110	0101
0	6	5

$(065)_{16} = (65)_{16}$

→ $(1372)_8 = (?)_{16}$

(a) $(1372)_8 = (?)_2$.

1	3	7	2
001	011	111	010

$(001011111010)_2$

(b) $(001011111010)_2 = (?)_{16}$.

0010	1111	1010
2	F	A

$(2FA)_{16}$

Hexadecimal to octal

→ $(DAC)_{16} = (?)_8$

(a) $(DAC)_{16} = (?)_2$.

D	A	C
1101	1010	1100

$(110110101100)_2$.

(b) $(110 | 110 | 101 | 100)_2 = (?)_8$

6	6	5	4

$(6654)_8$

→ $(1A8)_{16} = (?)_8$

(a) $(1A8)_{16} = (?)_2$

1	A	8
0001	1010	1000

$(000110101000)_2$

(b)

000	110	101	000
0	6	5	0

$(650)_8$

*Note

10 − r → Successive division

r − 10 → powers multiplication

r − r → if powers relation exists then grouping and use bit code

else

use intermediate (decimal / binary) to convert

Binary addition:-

(2)

→ 8 → 1 0 0 0
7 → 0 1 1 1

15 → 1 1 1 1

Here 2 → 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

→

		carry	1	1	1	
7			0	1	1	1
7			0	1	1	1
14			1	1	1	0

→

	carry	1	1	1	
15		1	1	1	1
15		1	1	1	1
	1	1	1	1	0

→

	carry	1		1	
5		0	1	0	1
5		0	1	0	1
10		1	0	1	0

→

	110	10	1	
	1	0	1	0
	1	1	1	1
	0	1	1	1
1 0	0	0	0	0

→ Binary subtraction:-

8
−4
4

0 10
1 0
3
07

1 15
2 5
7
1 8

6 0 1 1 0
2 0 1 0 0
4 0 1 0 0

8 1 0 0 0
7 0 1 1 1
1 0 0 0 1

(borrows: 0 1 1 2 / 0 1 2 / 0 2)

0 2 0 2
1 0 1 0
0 1 0 1
0 1 0 1

```
        0   1   1   2   2
        0   1   2   2
        0   2   2
*16     1   0   0   0   0
14      0   1   1   1   0
---------------------------
2       0   0   0   1   0
```

```
        0   1   2   0   2
*18     1   0   0   1   0
13      0   1   1   0   1
---------------------------
5       0   0   1   0   1
```

Compliments:

For any base 'r' system there exists two compliments 1. r's compliment

2. (r-1)'s compliment

* To find r's Compliment the following formula is used i.e., $r^n - N$

  where r – Base of number system
  n – no. of integer digits in the given number
  N – The number for which r's compliment to be Calculated.

* To find (r-1)'s compliment the following formula is used i.e., $(r^n - 1) - N$

Compliments in binary number system.

1's compliment

→ Find 1's compliment of 101

Sol:

N = 101
n = 3, r = 2.

Formula: $(r^n - 1) - N$

$(2^3 - 1) - N$

$(8 - 1) - 101$

$7(111) - 101$

$111 - 101$

010

```
 111
 101
-----
 010
```

1's compliment of 101 is 010

→ Find 1's complement of 1010

1's complement of 1010 is 0101

Simply replace 1's with 0 & 0 with 1.

(14)

→ 100011

1's compliment is 011100

→ 2's Complement

2's compliment can be calculated by adding 1 to 1's compliment of number.

* Find the 2's Complement of following number

→ 1101

1'S Compliment = 0010

Add 1 = 1

2'S compliment 0011

→ 1000

1S Compliment = 0111

Add 1 = 1

2'S compliment 1000

→ 10000

1'S Compliment = 01111

Add 1 = 1

10000

→ 10110

1S compliment = 01001

1

01010

By directly 01010

→ 10000

1's Compliment = 01111

1111 1

10000

use not in semexams

* To find 2's compliment directly there are two cases.

(i) if units place have zero then from right to left do not change the number until 1's reached then next numbers are interchanged by 0's as 1 & 1's as 0

(ii) if units place have 1 then not exchange 1 kept it as 1 & then next numbers are interchanged by 0 as 1 & 1 as 0

(i) 1010

2's comp Ans 0110

(ii) 111101

2's comp 000011

1's Complement Subtraction

If "A-B" is the required operation take the 1's complement of B & add it to A. After adding there are two cases

Case(i): Carry generated.

If Carry generated after the addition of A and 1's complement of B End-around the Carry i.e., add Carry to the LSB (least significant bit)

Case(ii): Carry not generated

If Carry not generated after the addition, take the 1's Complement of result again and put a '-' sign before the result.

Ex:

$9 \rightarrow 1001$

$(-)4 \quad 0100$

$5 \quad 0101$

Case(i)

$9 \rightarrow 1001$

1's comp of 4 $\rightarrow 1011$

$(+) \quad 11$

Carry $\leftarrow$ (1) 0100

$+ \quad 1$

$5 \rightarrow 0101$

Case(ii)

$4 \quad 0100$

$(+)9 \quad 0110$

1010

$-0101 \rightarrow$ 1's comp of

* $8 \rightarrow 1000$

$3 \rightarrow 0011$

$(-)$

$5 \rightarrow 0101$

Case(i):

$8 \quad 1000$

$-3 \quad (+)1100$

$5 \quad (1)0100$

carry $+ \quad 1$

0101

Case(ii):

$3 \quad 0011$

8(1's comp) 0111

$1010 \rightarrow$ carry not generated

$-5 \quad -0101$

* 10 → 1 0 1 0
(-) 7 → 0 1 1 1
3 → 0 0 1 1

10 → 1 0 1 0
1's comp of 7 (+) 0 0 0 0
1 0 0 1 0
(+) 1
0 0 1 1

7 → 0 1 1 1
10 → 0 1 0 1
-3 → 1 1 1
1 1 0 0
-0 0 1 1

* 15 → 1 1 1 1
(-) 11 → 1 0 1 1
4 → 0 1 0 0

15 → 1 1 1 1
1's comp of 11 (+) 0 1 0 0
1 0 0 1 1
4 +1
0 0 1 0 0

11 → 1 0 1 1
1's of 15 0 0 0 0
-4 → 1 0 1 1
-0 1 0 0

* 10 11 01
01 01 10

Case (i):
10 11 01
(+) 10 10 01
10 10 11 10
+1
01 01 11 1

01 01 10
(+) 01 00 10
10 10 00
(-01 01 11 1)

2's compliment subtraction:

If "A-B" is the required operation, takes the Two's Compliment of "B" and Add it to 'A'. After addition there are two cases.

Case(1):

Carry generated. After adding 'A' & 2's Comp-liment of 'B" if Carry is generated discard the Carry

Case(ii):

Carry is not generated. After adding "A" of 2's compliment of 'B" if Carry is not generated take the 2's Compliment of result again and Put a (-)(minus) sign before it.

Ex:

* 8 → 1 0 0 0 → 1 0 0 0

(-) 5 → 0 1 0 1 → (+) 1 0 1 1

3

1 0 0 1 1

discarded the carry

0 0 1 1

Ans. 3

5 → 0 1 0 1 → 0 1 0 1

(-) 8 → 1 0 0 0 (+) 1 0 0 0

-3

1 1 0 1

Ag 2's (-0 0 1 1) → -3.

* 9 → 1 0 0 1 → 1 0 0 1

(-) 4 → 0 1 0 0 (+) 1 1 0 0

5.

1 0 1 0 1

discard

4 → 0 1 0 0 → 0 1 0 0 → 5

(-) 9 → 1 0 0 1 → 0 1 1 1

-5

1

1 0 1 1

2's (-0 1 0 1) → -5

* 23 → 10111

10 (-) 01010 → 10111

(+) 10110

(1) 01101

discard 01101

10 01010 → 01010

23 (-) 10111

(+) 01001

10001

(-01101)

* 32 16 8 4 2 1

00110 → 00110

11101

(+) 00011

01000

-11000

11110 → 11110

00110

(+) 11001

(1) 11000

discard 11000

9's Complement Subtraction:-

If 'A-B' is the required operation take the 9's complement of B & add it to A. After adding there are two cases

Case1: Carry generated.

If Carry generated after the addition of A and 9's complement of B End around the carry i.e., add carry to LSB

Case2: Carry not generated

If Carry not generated after the addition take the 9's compliment of result again and put a '-' sign before the result.

(19)

* 8 → 8
−5 9's com → 4
12
+1 → end around carry
[3]

9 − 5 = 4

5 → 5
−8 9's com → 1
6
9's of 6 [−3]

9 − 8 = 1, 9 − 6 = 3

* 9 → 9
−4 9's com → (+)5
14
+1
[5]

9 − 4 = 5

4 → 4
−9 9's → (+)0
4
9's of 4 [−5]

9 − 4 = 5

* 25 → 25
−19 9's com → (+)80
105
+1
[6]

99 − 19 = 80

19 → 19
−25 9's comp → (+)74
93
[−6]

99 − 25 = 74
99 − 93 = 6

* 84 → 84
−27 9's com → (+)72
156
+1
[57]

99 − 27 = 72

27 → 27
(−)84 → (+)15
42
9's of 42 [−57]

99 − 84 = 15
99 − 42 = 57

* 459 → 459
(−)362 9's of → (+)637
1096
+1
[097]

999 − 362 = 637

362 → 362
−(459) → (+)540
902
9's of 902 [−97]

999 − 459 = 540
999 − 902 = 97

Scanned with CamScanner

10's Compliment subtraction

If "A - B" is the required operation, takes 10's compliment of B and add it to A. After addition there are two cases:

Case(i): Carry generated

If carry generated after adding A & 10's compliment of B, if carry generated then discard the carry

Case(ii): Carry not generated.

After adding A & 10's compliment of B if carry not generated take 10's compliment of result again and put a (-) minus sign before it.

* 9 → 9
(-) 3 —10's of 3→ 7
(+)
(1) 6 — discard.

10's comp: 9 (-) 3 = 6; 6 + 1 = **7**

3 → 3
\- 9 —10's of 9→ 1
= 4
10's of 4 **-6**

10's comp: 9 - 9 = 0; 0 + 1 = 1

* 8 → 8
(-) 4 —10's of 4→ 6
(+)
(1) **4**

10's comp: 9 (-) 4 = 5; 5 + 1 = 6

4 → 4
\- 8 —10's of 8→ (+) 2
= 6
10's of 6 **-4**

10's compl: 9 (-) 8 = 1; 1 + 1 = 2 | 9 (-) 6 = 3; 3 + 1 = 4

* 56 → 56
\- 43 —10's of 43→ 57
(+)
(1) 13.

10's comp: 99 - 43 = 56; 56 + 1 = 57.

43 → 43
-(56) —10's of 56→ 44
(+)
= 87
10's of 87 → **-13**

10's comp: 99 - 56 = 43; 43 + 1 = 44

99 - 87 = 12; 12 + 1 = 13

* 546 → 546 (1's of 307 → 692, +1 → 693)

- 307 → 1's of → 693

(+) 1

⊛239

999 − 307 = 692, + 1 = 693

307 → 307

−546 → 454 (+1)

761

10's of 761 → −239

999 − 546 = 453, + 1 = 454

999 − 761 = 238, + 1 = 239

Binary codes:

Binary codes are classified into 6 types

1. Weighted Codes
2. Non-Weighted Codes
3. Sequential codes
4. Reflective codes
5. Alphanumeric Codes
6. Error Correcting & detecting codes

Weighted Codes:

In weighted codes each place value is having a specific weight. Based on the weights, the codes are derived.

Ex: 8421, 2421, 5411 etc....

Non-weighted codes.

These are the codes derived from weighted codes and no specific weight for place value

Ex: Gray code, Parity, excess 3 code etc...

Sequential codes:

(4211) → not sequential

In sequential codes the weights are continously increasing from LSB to MSB

Ex: 8421.

Reflective codes.

In these codes zero is the 1's compliment of 9, 1 is the 1's compliment of 8

(P.T.O)

2 is the is compliment of 8 and so on.

Ex: 2421

→ Write the 2421 code

	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	1	1	1
6	1	1	0	0
7	1	1	0	1
8	1	1	1	0
9	1	1	1	1

Alphanumeric codes:

In Alphanumeric codes each alphabet (Both uppercase & lowercase), numbers and symbols are having specific binary representation

Ex: ASCII code

American Standard Code for Information Interchange

Error Correcting & Error detecting Codes

By using some of the binary codes errors can be detected but not corrected they are called error detecting codes

By using some of the binary codes errors can be detected & corrected they are called error correcting code

Ex: Hamming code.

BCD Code: (Binary coded decimal)

In BCD code all the decimal digits from 0 to 9 are coded with binary numbers as shown below

	8	4	2	1
0 – 0	0	0	0	0
1 – 0	0	0	0	1
2 – 0	0	0	1	0
3 – 0	0	0	1	1
4 – 0	0	1	0	0
5 – 0	0	1	0	1
6 – 0	0	1	1	0
7 – 0	0	1	1	1
8 – 1	0	0	0	0
9 – 1	0	0	0	1

BCD addition

In BCD addition, after adding two BCD numbers, the result should be valid BCD number. If it is invalid add correction factor 6 (0110) to the invalid BCD number.

* 5 → 0 1 0 1
3 → (+) 0 0 1 1
8 → 1 0 0 0

* 5 → 0 1 0 1
5 → 0 1 0 1
10 (+) 1 1
1 0 1 0 → Invalid BCD.
+6 → (+) 0 1 1 0
0001 0 0 0 0
1 0

* 8 → 1 0 0 0
+6 → 0 1 1 0
14 1 1 1 0 → Invalid BCD
+6 → (+) 0 1 1 0
0001 0 1 0 0
1 4

* 15 → 0 0 0 1 0 1 0 1
12 → + 0 0 0 1 0 0 1 0
27 0 0 1 0 0 1 1 1
2 7

* 39 → 0 0 1 1 1 0 0 1
(+) 23 → 0 0 1 0 0 0 1 1
62 0 1 0 1 1 1 0 0 → Invalid BCD.
0110 0010 +6 0 1 1 0
6 2 1 1 1
0 1 1 0 0 0 1 0
6 2

* 28 → 0010 1000
  26 → (+) 0010 0110
  54 → 0100 1110 → Invalid BCD

0101 0100 +6
5 4
0110
0101 0100
5 4

* 45 → 0100 0101
  38 → 0011 1000
  83 → 0111 1101 → Invalid BCD

1000 0011 +6
8 3
0110
1000 0011
8 3

* 157 → 0001 0101 0111
  138 → 0001 0011 1000
  295 → 0010 1001 0101

0010 1001 0101 +6
2 9 5
0110
0010 1001 0101
2 9 5

* 867 → 1000 0110 0111
  574 → 0101 0111 0100
  1441 → 1101 1101 1011

0001 0100 0100 0001 +6
0110
1101 1110 0001
+6
0110
1110 0100 0001
+6
0110
0001 0100 0100 0001
1 4 4 1

BCD Subtraction:-

If "(A-B)" is required operation, take the 9's Complement of "B" and add it to A. After adding there are two cases.

Case 1: Invalid BCD

If the sum is invalid BCD number validate it by adding 6 (0110). After addition of 6, the Carry has to be end arounded.

Case 2: Valid BCD.

After addition, if the result is valid BCD, take the 9's complement of result once again & put (-)minus sign before it.

Ex:

① 8 → 1000 ; 4 → 0100

(-)4 —9's of 4→ 0101 ; -8 —9's of 8→ 0001

4 ; -4

0100 ; -0100

(+) 0101 ; 0001

1101 → Invalid BCD ; 0101 → valid BCD

+6 0110 ; 9's of result → 0100

1 ; 1

1011 ; 1

1 ; 1

+ 1 1 ; +

0100 ; 0100

② 9 → 1001 ; 9

-3 —9's of 3→ 0110 ; -3

6 ; 6

0110 ; 3 → 0011

1111 → Invalid BCD ; -9 —9's of 9→ 0000

+6 0110 ; (+) 0011

1 ; -6

1010 ; 9's of result is -0110

1 ; 1

+ 1 ; +

0110 ; 0110

```
* 25   ->   0010 0101                          99
- 13   9's of 13 ->  1000 0110               - 13
-----        (+) --------------                ----
  12             1010 1011                      86
0001 0010   +6        0110                     (27)
                 --------------
                 1011 0001
            +6   0110
                 --------------
                 1 0001 0001                    99
                 |------->  +1              (-) 87
                 --------------                ----
                 [0001 0010]                    12

                                                99
                                                25
                                               ----
                                                74

  13   ->   0001  0011      0001  0011
- 25        0010  0101  +   0111  0100
-----       ----------      ------------
 -12                        1000  0111
              9's of result [- 0001 0010]

* 257  ->   0010  0101  0111
- 143   (+) 0001  0100  0011
------      ----------------
  114                   1010                   999
                                               143
0001 0001 0100                                 ----
                                               856
            0010  0101  0111
          + 1000  0101  0110
            ----------------
            1010  1010  1101
        +6              0110
            ----------------
            1010  1011  0011
        +6  0110
            ----------------
            1011  0001  0011
       +6   0110
            ----------------
          1 0001  0001  0011
          |------------->  +1
            ----------------
            0001  0001  0100
```


143 → 0001 0100 0011

-257 → 0111 0100 0010

+ 1 1 1 1 1

-114 → 1000 1000 0101

999
257
742

999
885
114

9's of 885 is [-0001 0001 0100]

Gray code:

Gray code is a non weighted code, derived from binary code. In gray code there is only one bit change from one number to the immediate next number.

Q: Generate 4-bit gray code from single bit gray code.

Ans

Q - 0
1 - 1

0 - 0
1 - 1

0-00
1-01
1-11
2-10

0-000
1-001
2-011
3-010
4-110
5-111
6-101
7-100

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

Binary to Gray code:

To convert given 4-bit binary to its equivalent 4-bit gray code, the following formula is used.

Let the given 4 bit binary number is $(B_3 B_2 B_1 B_0)$ and its equivalent 4 bit gray code is $(G_3 G_2 G_1 G_0)$ then

$G_3 = B_3$

$G_2 = B_3 \oplus B_2$

$G_1 = B_2 \oplus B_1$

$G_0 = B_1 \oplus B_0$

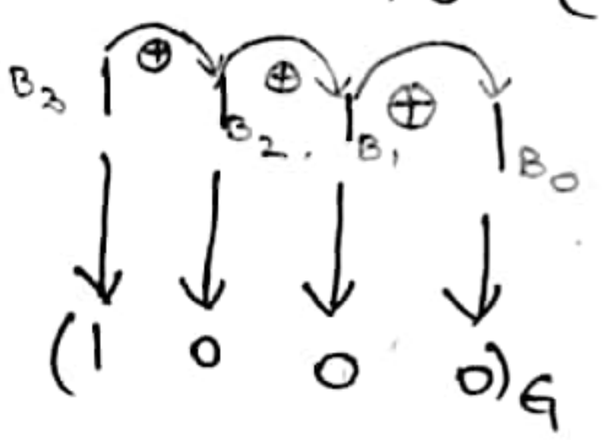
EX-OR truth table

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

$\oplus$

→ Convert $(1111)_B = (?)_G$

Ans:



$(1\ 0\ 0\ 0)_G$

→ $(0111)_B = (?)_G$

Ans:

0 1 1 1

↓ ↓ ↓ ↓

$(0\ 1\ 0\ 0)_G$

→ $(1001)_B = (?)_G$

1 0 0 1

↓ ↓ ↓ ↓

$(1\ 1\ 0\ 1)_G$

→ $(1110)_B = (?)_G$

1 1 1 0

↓ ↓ ↓ ↓

$(1\ 0\ 0\ 1)_G$

→ $(10111\ 0)_B = (?)_G$

1 0 1 1 1 0

↓ ↓ ↓ ↓ ↓ ↓

1 1 1 0 0 1

$(11001)_G$

Gray code to Binary code Conversion:

Let the given 4-bit gray code is $(G_3G_2G_1G_0)$ and its equivalent binary is $(B_3B_2B_1B_0)$ then

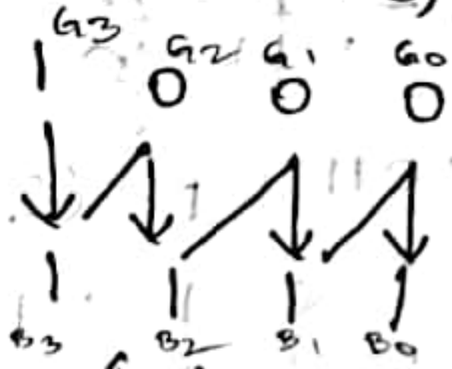
$$B_3 = G_3$$

$$B_2 = G_3 \oplus B_3 \quad B_3 \oplus G_2$$

$$B_1 = B_2 \oplus G_1$$

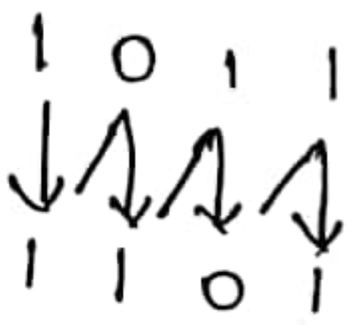
$$B_0 = B_1 \oplus G_0$$

→ Convert $(1000)_G = (?)_B$



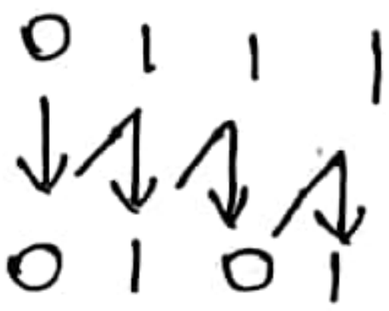
$(1111)_B$

→ $(1011)_G = (?)_B$



$(1101)_B$

→ $(0111)_G = (?)_B$



$(0101)_B$

Excess-3 code:

Excess-3 code is a non weighted code delivered from BCD code for every BCD number 3 is added to get excess 3-code

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

Excess-3 addition:

In excess-3 addition after adding two excess-3 numbers there are two cases

Case(i): No Carry

If there is no Carry after excess-3 addition subtract (0011) 3 from the result To get final result

Case(ii): Carry generated:

After excess-3 addition if Carry generated add 3(0011) to both sum & carry.

Ex:

$$\begin{array}{rcl} 3 & \rightarrow & 0110 \\ (+)\ 4 & \rightarrow & (+)\ 0111 \\ \hline 7 & & 1101 \\ (1010) & & 0011 \text{ (sub-3)} \\ \hline & & 1010 \end{array}$$

2 → 0101
+3 → 0110
---
85 1011
1000 0011 (Sub 3)

1000

6 → 1001
7 → 1010
---
13 0001 0011
0011 0011 Add 3

0100 0110
---
1 3

13 → 0100 0110
12
+

25

0101 1000 (+) 0100 0101
---
2 5 1000 1011
-0011 Sub(3)

1000 1000

Sub 3
0011

0101 1000
2 5

38 → 0110 1011
+25 0101 1000
---
63 (+) 1111

1100 0011
0011 0011 Sub 3
(-) (+) 1 1

1001 0110

45 → 0111 1000
36 → 0110 1001
+
---
81 (+) 1111

1110 0001

1011 0100 0011 Add 3
---
8 1 1110 0100

Sub3 0011

1011 0100 E
---

93 → 1100 0110
48 (+) 0111 1011
---
141 0001 0100
0011 0011 Add (3)

0100 0111 0100 0100 0111 0100

Excess-3 Subtraction:

In excess-3 subtraction if A-B is the required operation then take the 9's or 1's complement compliment of B and addit to A. After addition there are two cases. for excess 3 only

Case 1: Carry generated.

After addition if Carry generated, end-around the carry and add 3 to the sum to get the final answer.

Case 2: Carry not generated.

After addition if Carry is not generated subtract 3 from the sum and once again take the 9's Compliment of the result and put (-) sign before the result.

(or 1's compliment for excess 3 only)

* Excess-3 is a self compliment Code, Because 9's compliment of 1's compliment is same.

Ex:

* $6 \rightarrow 1\,0\,0\,1$ (9's of 3: 6)

$-3 \xrightarrow{\text{9's of 3}} 1\,0\,0\,1$

3

$1\,0\,0\,1\,0$

$0\,1\,1\,0$ (end-around) $+1$

$0\,0\,1\,1\,0$

$0\,0\,1\,1$ Add 3 (0011)

$0\,1\,1\,0$

$3 \rightarrow 0\,1\,1\,0$

$-6 \xrightarrow{\text{9's of 6}} 0\,1\,1\,0$ (+)

-3

$1\,1\,0\,0$

-0110 $0\,0\,1\,1$ Sub 3 (0011)

9's of result $1\,0\,0\,1$

$\hookrightarrow -0\,1\,1\,0$

* 5 → 1 0 0 0   9
−4 9's of 4 → (+) 1 0 0 0   4
1   5

1 0 0 0 0

0100   end around carry.

0 0 0 1
0 0 1 1 Add 3(0011)

0 1 0 0

4 → 0 1 1 1   9
−5 → 0 1 1 1   5
−1   4

1 1̶0 1̶02 0̶2
−0100   0 0 1 1 Sub 3(0011)

1 0 1 1

9's of result (0 1 0 0)

* 26 → 0101 1001 → 0101 1001
−14 9's → 0100 0111 1's comp → (+) 1011 1000

12   0001 0001 0001

0100 0101

(end around carry)

0001 0010
(+) 0011 (0011) Add 3

0001 0101
0011 Add 3

0100 0101
1   2

* 14 → 0100 0111
−26 9's → (+) 1010 0110

−12   1110 1101

99
26
73

0011 (sub 3)

1 1̶1̶0² 0̶2 1010
0011   Sub 3

1011 1010.

9's of result (−0100 0101)
1   2

* 2 5 5 → 0101 1000 1000 → 0101 1000 1000
(−) 1 6 3 → 0100 1001 0110 → 1011 0110 1001

0 9 2

(+) 1 0000 1111 0001

→ 1

0000 1111 0010

0011 1100 0101

(0011) Add 3

0000 1111 0010

(+) 0011 0011 0011

0010 0101

Add 3

0011

0000 1111 0101

(Add 3) 0011 0011

0011 1100 0101

0 9 2.

→ 1 6 3 → 0100 1001 0110
(−) 2 5 5 1's → 1010 0111 0111

−0 9 2

(+)

1111 0000 1101

(−) 0011 + 0011 (+) 0011

1100 0011 1010

1's of result

(−0011 1100 0101)

* 2 5 5 → 0110 1011 1000 (over 3 3 8 8 1 5 with corrections 2, 17)
− 1 9 6 → 1011 0011 0110

1 8 9

(+) 1 0001 1110 1110

→ 1

0001 1101 1111

(+) 0011 (−) 0011 (−) 0011 Add 3

0100 1011 1100

1 8 9

999
196
803

0100 1011 1100

196 → 0100 1100 1001

-385 → 1001 0100 0111

-189 → 1111 0001 0000

0100 1011 → (-) 0011 (+) 1100 (+) 0001

1100 → 1011 0100 0011

⇓

-(0100 1011 1100)

9's of result: 1 8 9

Signed Binary numbers:

In signed binary numbers, one sign bit is allocated for representation of +ve numbers and -ve numbers. For +ve numbers sign bit is 0. For -ve numbers the sign bit is 1.

For Ex:

sign bit

0 0001 = +1

1 0001 = -1

* The signed binary numbers can be represented in three ways.
  (1) sign magnitude representation
  (2) 1's complement "
  (3) 2's Complement "

Note:
***
***In all three representations, +ve numbers are having unique representation, where as -ve numbers having different represen-tations

Ex:

	+5	-5
sign magnitude	0 0000101	1 0000101
1's Complement	0 0000101	1 1111010
2's Complement	0 0000101	1 1111011

Ex: using 2's Complement

+7 = 00000111

+5 = 00000101

-5 = 11111011

-7 = 11111001

+2 = 00000010

-2 = 11111110

+12 = 00001100

-12 = 11110100

+7 → 00000111
+5 → 00000101 (+)
+12 → 00001100

+7 → 00000111
-5 → 11111011 (+)
+2 → discard 1 00000010

-7 → 11111001
+5 → 00000101 (+)
-2 → 11111110

-7 → 11111001
-5 → 11111011
-12 (+) 11110100
discard (*)

some above question using 1's compliment

* Using 1's Complement.

+7 → 00000111
+5 → 00000101
-7 → 11111000
-5 → 11111010
+2 → 00000010
-2 → 11111101
+12 → 00001100
-12 → 11110011

+7 → 00000111
+5 → 00000101
(+) 00001100

+7 → 00000111
-5 → 11111010
(+) 100000001
+1
00000010

-7 → 11111000
+5 → 00000101
(+) 11111101
-2

-7 → 11111000
-5 → 11111010
(+) 111110011
+1
111100 11

⇒ ERROR DETECTING CODES:

Parity bit is an extra bit which is added to the original bit. Parity is of two types 1. Even Parity
2. odd Parity.

Even Parity: For even Parity the no. of 1's in the information should be even including Parity bit.

Odd Parity: For odd Parity the no. of 1's in information is odd including Parity bit.

Ex: Generate Even & odd Parity for given message.

Ans → 0 0 1 0 1 1 0

0 0 1 0 1 1 0 (1) → even Parity
0 0 1 0 1 1 0 (0) → odd parity.

Block Parity: In block Parity the Parity is taken for both rows & columns for group of messages.

Ex:

Even parity

0 0 1 0 1 0	0
0 0 1 0 1 0	0
0 0 1 0 1 1	1
1 0 1 0 1 0	1
1 0 0 0 0 1	

Error detection & correction codes:

In the error detection & correction codes Hamming code is the best example. By using hamming code, error can be detected and it can be corrected.

Hamming code generation:

To generate hamming code for 'm' no. of message bits the following formula is used

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

where p = no. of parity bits

Generate Hamming code for 1011.

Ans:

1 0 1 1

m_1 m_2 m_3 m_4

$m = 4$

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

$2^p \geq 4+p+1$ [$\because m=4$]

if p = 0

$2^0 \geq 4+0+1$

$1 \geq 5$

if p = 1

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

$2 \geq 6$

if p = 2

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

$4 \geq 7$

if p = 3

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

$8 \geq 8$

$m = 4$

$p = 3$

Total = 7

(4)

1	2	3	4	5	6	7
P_1	P_2	m_1	P_3	m_2	m_3	m_4
		1		0	1	1

0 01-1
0 10-2
0 11-3
1 00-4
1 01-5
1 10-6
1 11-7

1,3,5,7 0 1 0 1

2,3,6,7 1 1 0 1

4,5,6,7 0 0 1 1

(0 11 00 11)

0010!

1	2	3	4	5	6	7
P_1	P_2	m_1	P_3	m_2	m_3	m_4
		0		0	1	0

0010
1001
1010
1110

1,3,5,7 0 0 0 0

2,3,6,7 1 0 1 0

4,5,6,7 1 0 1 0

(0 1 0 1 0 1 0 1 0)

1001!

1	2	3	4	5	6	7
P_1	P_2	m_1	P_3	m_2	m_3	m_4
		1		0	0	1

1,3,5,7 0 1 0 1

2,3,6,7 0 1 0 1

4,5,6,7 1 0 0 1

(0 0 1 1 0 0 1)

1010:

	1	2	3	4	5	6	7
	P_1	P_2	m_1	P_3	m_2	m_3	m_4
			1		0	1	0
1,3,5,7	1		1		0		0
2,3,6,7		0	1			1	0
4,5,6,7				1	0	1	0

(1 0 1 1 0 1 0)

1110:

	1	2	3	4	5	6	7
	P_1	P_2	m_1	P_3	m_2	m_3	m_4
			1		1	1	0
1,3,5,7	0		1		1		0
2,3,6,7		0	1			1	0
4,5,6,7				0	1	1	0

(0 0 1 0 1 1 0)

Error detection and correction:

→ original: 0101010 ⇒ can 0100010

	1	2	3	4	5	6	7
	0	1	0	0	0	1	0
c_1 → 1,3,5,7	0		0		0		0 → 0
c_2 → 2,3,6,7		1	0			1	0 → 0
c_3 → 4,5,6,7				0	0	1	0 → 1

c_3 c_2 c_1

1 0 0 ⇒ 4

∴ there is an error in 4th position.

the correct information is 0101010

$\Rightarrow$ 0101010 $\xRightarrow{\text{error}}$ 0101000

1	2	3	4	5	6	7	
0	1	0	1	0	0	0	
$C_1 \rightarrow 1,3,5,7 \rightarrow$	0		0		0		0 → 0
$C_2 \rightarrow 2,3,6,7 \rightarrow$		1	0			0	0 → 1
$C_3 \rightarrow 4,5,6,7 \rightarrow$				1	0	0	0 → 1

C_3 C_2 C_1

1 1 0 → 6

∴ There is an error in 6th position.
The correct information is 0101010

$\Rightarrow$ 0101010 $\xRightarrow{\text{error}}$ 0001010

1	2	3	4	5	6	7.
0	0	0	1	0	1	0
$C_1 \rightarrow 1,3,5,7 \rightarrow$	0	0		0		0 → 0
$C_2 \rightarrow 2,3,6,7 \rightarrow$		0	0		1	0 → 1
$C_3 \rightarrow 4,5,6,7 \rightarrow$				1	0	1 0 → 0

C_3 C_2 C_1

0 1 0 → 2

∴ There is an error in 2nd position, the Correct information is 0101010.

$\Rightarrow$ 0101010

1	2	3	4	5	6	7.
0	1	0	1	0	1	0
$C_1 \rightarrow 1,3,5,7$	0		0		0	0 → 0
$C_2 \rightarrow 2,3,6,7$		1	0			1 0 → 0
$C_3 \rightarrow 4,5,6,7$				1	0	1 0 → 0

C_3 C_2 C_1

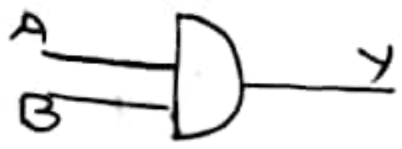
0 0 0

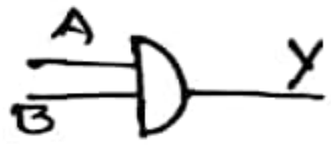
No error.

Logic Gates:

(1) Basic gates → AND, OR, NOT
(2) Universal gates → NAND, NOR
(3) Special gates → EX-OR, EX-NOR

AND Gate





A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

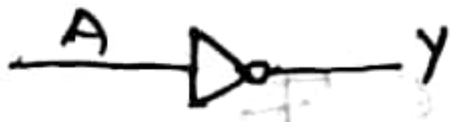
$Y = A \cdot B$
$= AB$

OR Gate:

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

$Y = A + B$

NOT Gate:

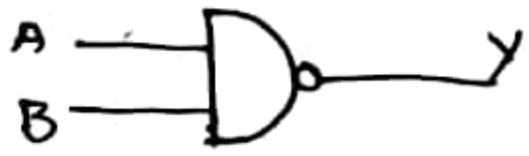


A	Y
0	1
1	0

$Y = \overline{A} = A'$

NAND Gate :

NAND Gate is • AND gate followed by NOT Gate



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

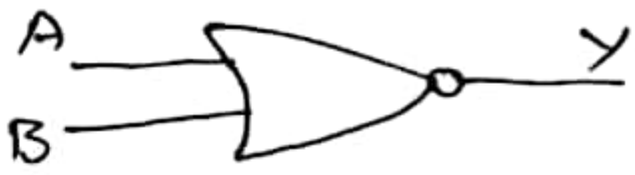
$= \overline{AB}$

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

Any boolean function can be purely implemented with this two gates (NAND, NOR)

NOR Gate :

NOR Gate is OR Gate followed by NOT gate.



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

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

Special gates:

→ EX-OR: (inequality gate)

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

$Y = A \oplus B$

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

Important:

Property: $A \oplus 0 = A$

$A \oplus 1 = \bar{A}$

Ex-NOR (equallity gate)

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

$Y = A \odot B$

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

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

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.

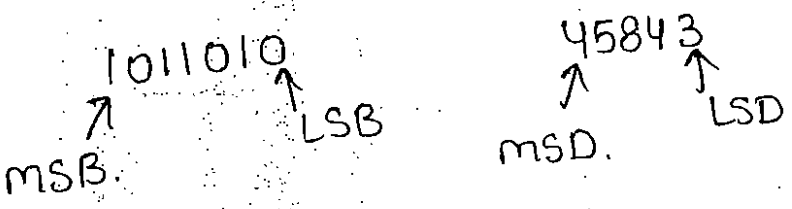
Radix - NO of symbols presented in a respective number system is called radix.

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.

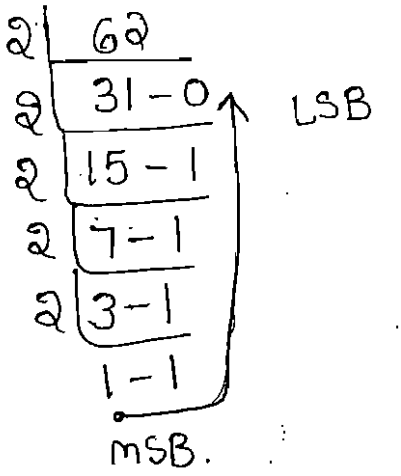
→ Integer part conversion.

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.

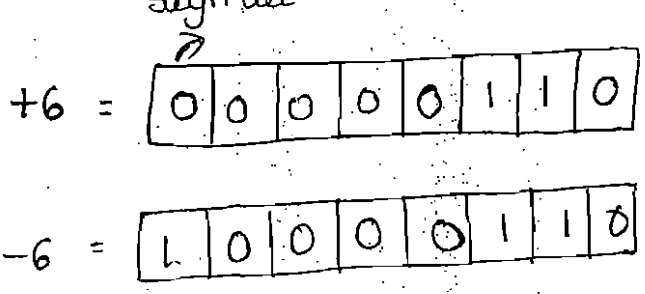
→ 2's complement representation.

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.

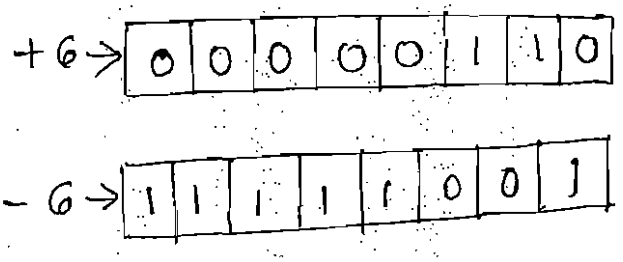
In eight bit representation, the MSB indicates the sign. and remaining seven bits represent the magnitude.

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.

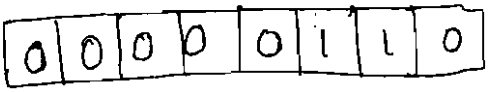


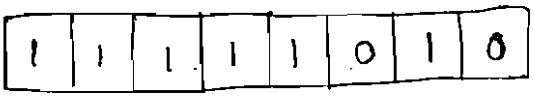
2's Complement Representation :-

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)

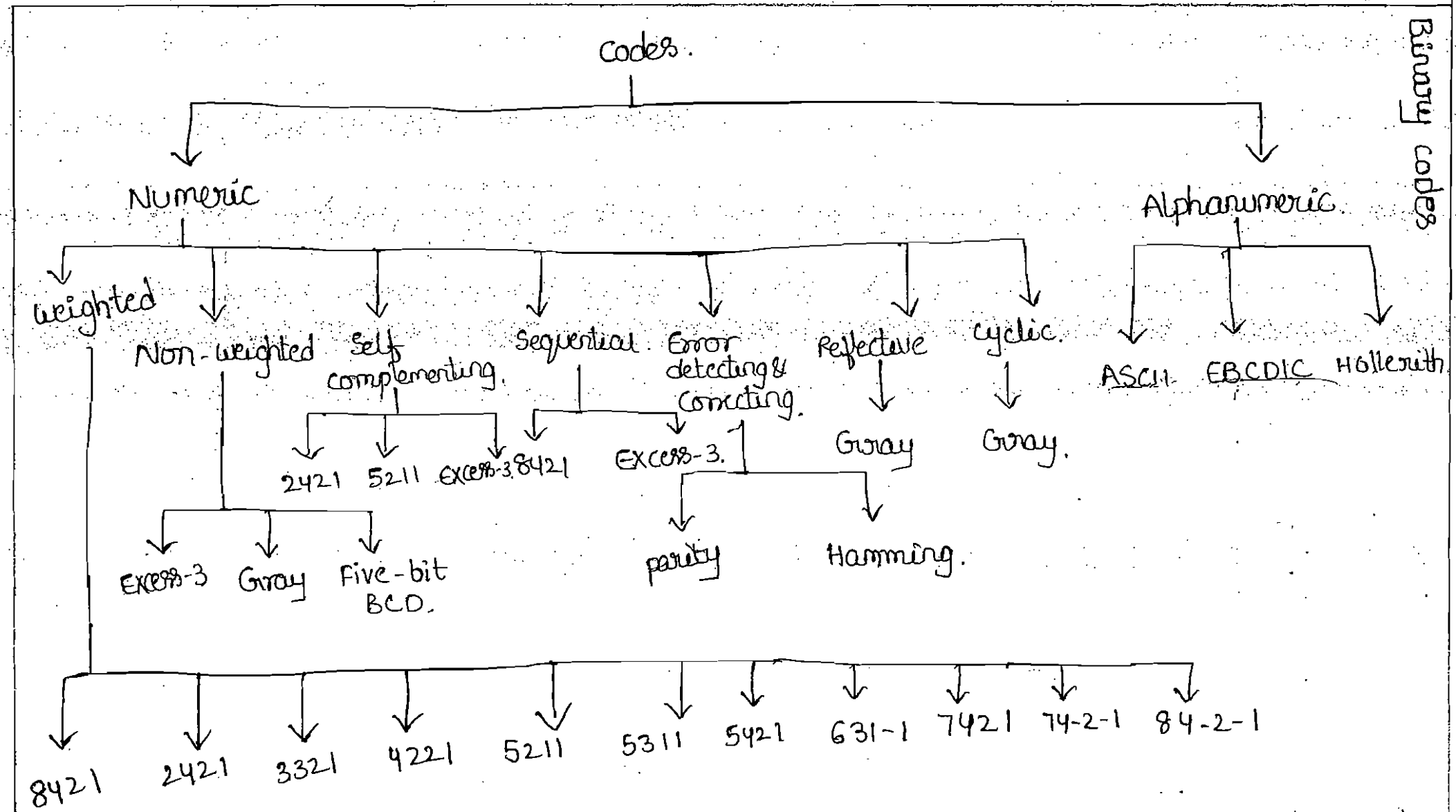
→ find the 2's complement of the number by adding 1 to 1's complement of the number.

+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 :-

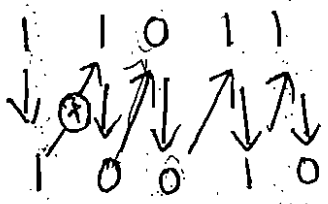
→ The MSB of the binary number is the same as the MSB of the Gray code

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

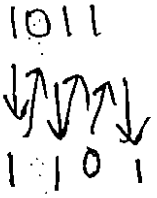
→ Add the 2nd bit of the binary to the 3rd bit of the Gray; the 3rd bit of the binary to the 4th bit of the Gray code.

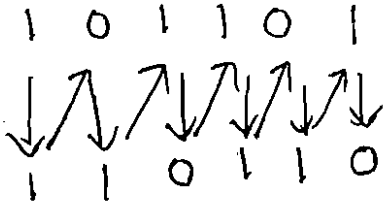
→ convert Gray to Binary.

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:-

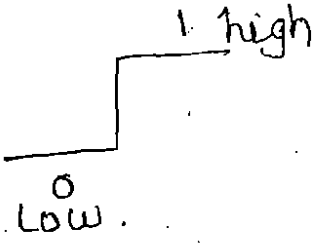
The logic Gates are the fundamental blocks of digital systems. The logic Gate is ability to make decisions (output). logic gate are electronic circuits because they are made up of a number of electronic devices and components.

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

Positive Logic:-

high → 1

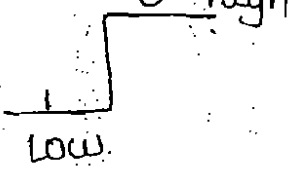
Low → 0.



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.

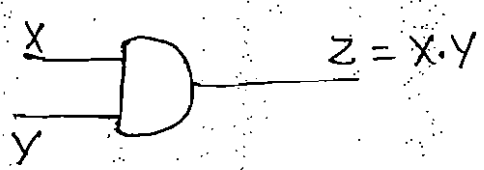
4. NAND Gate
5. NOR Gate

} 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

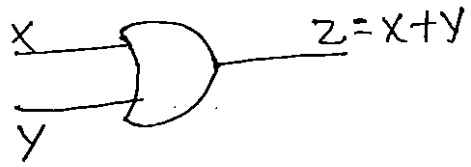
X	Y	Z=X.Y
0	0	0
0	1	0
1	0	0
1	1	1

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.

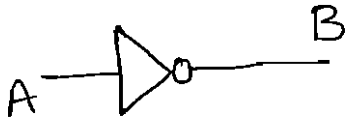
X	Y	Z=X+Y
0	0	0
0	1	1
1	0	1
1	1	1

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



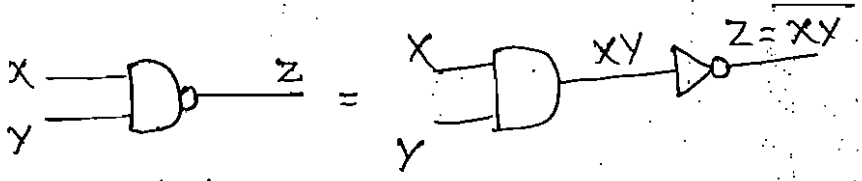
Symbol

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.

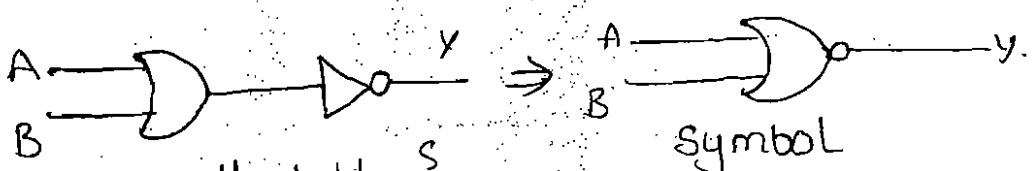
X	Y	$Z = \overline{XY}$
0	0	1
0	1	1
1	0	1
1	1	0



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

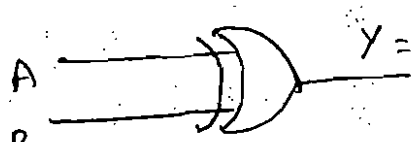
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

$Y = \overline{A+B}$

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



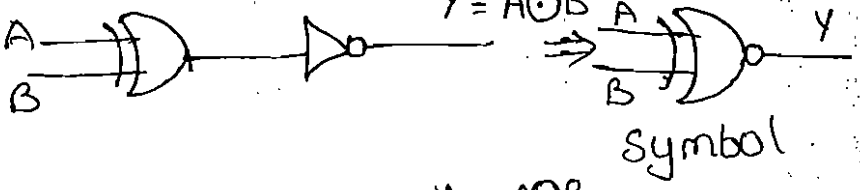
Symbol.

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 $(B \cdot \bar{B})$.

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

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

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

$$= M_1, M_2, M_3$$

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

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

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

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

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

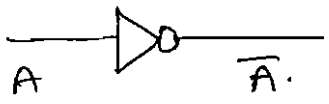
$$(000)\ (001)\ (010)\ (011)\ (100)\ (101)$$

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

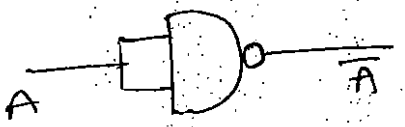
$$= \Pi M(0,1,2,3,4,5).$$

NAND - NAND Realization :-

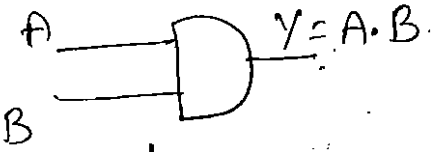
1) NOT Gate

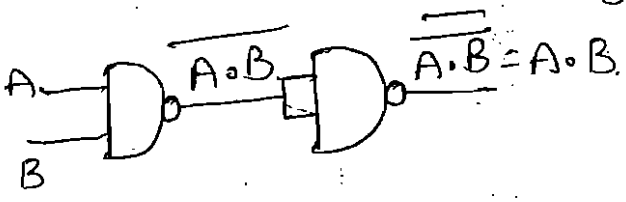


Using NAND gate.



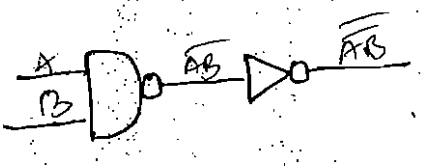
2) AND Gate.

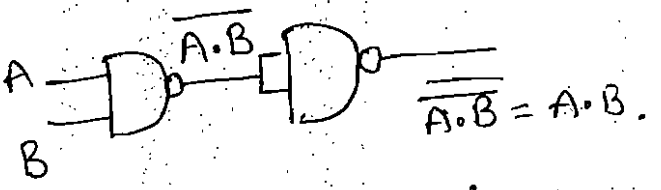




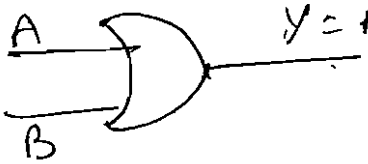
Step :- 1 add bubble on output

Step :- 2 add one NOT gate

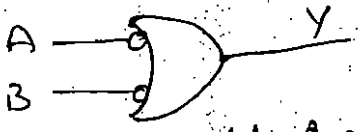




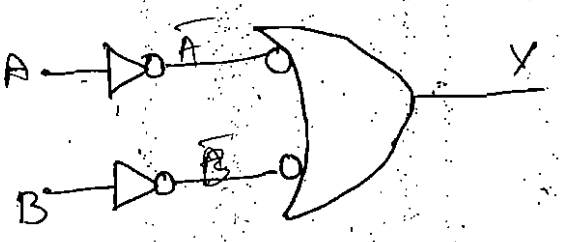
3) OR Gate

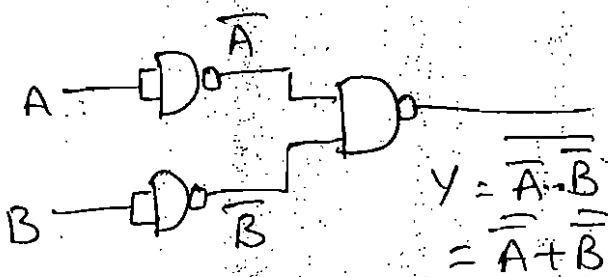


Step :- 1 add bubble on input



step 2 :- add not gate



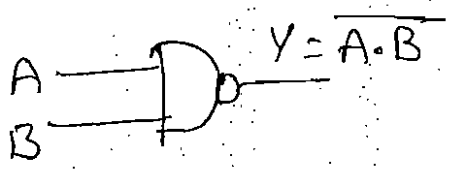


$Y = \overline{\overline{A} \cdot \overline{B}}$

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

$= A + B.$

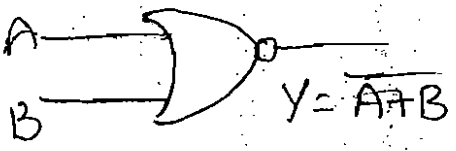
4) NAND gate

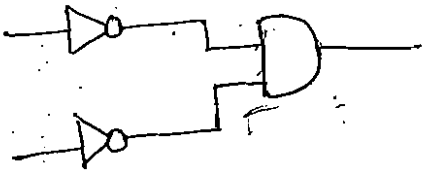


5) NOR Gate

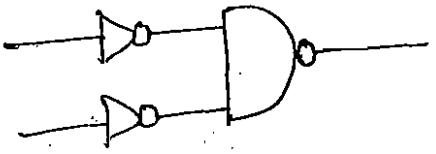
$Y = \overline{A+B}$.

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

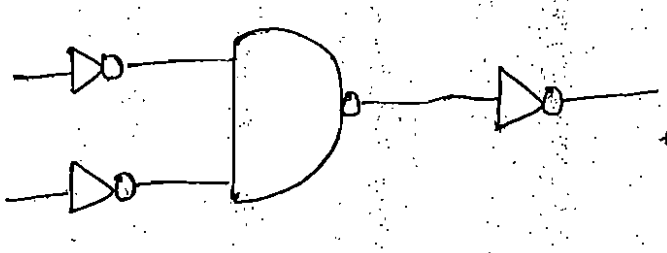




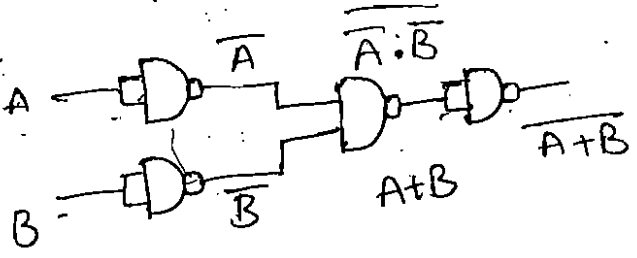
Step:-1



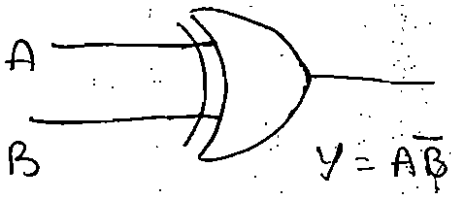
Step :- 2

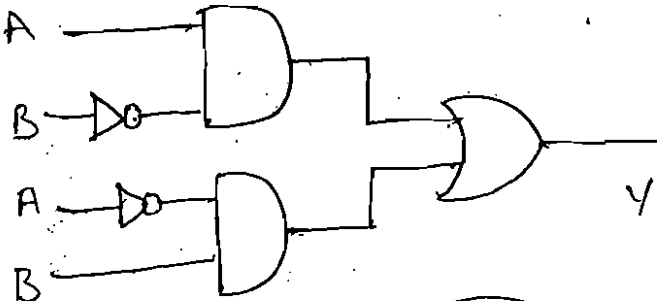


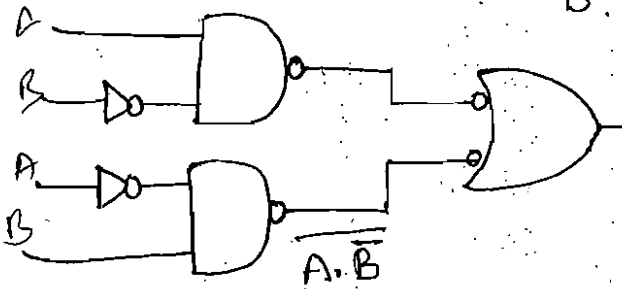
Step 3 :-

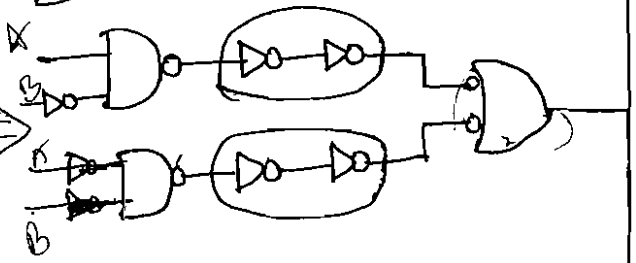


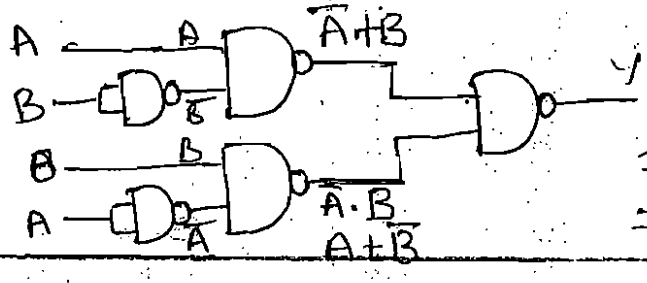
6) EX-OR Gate









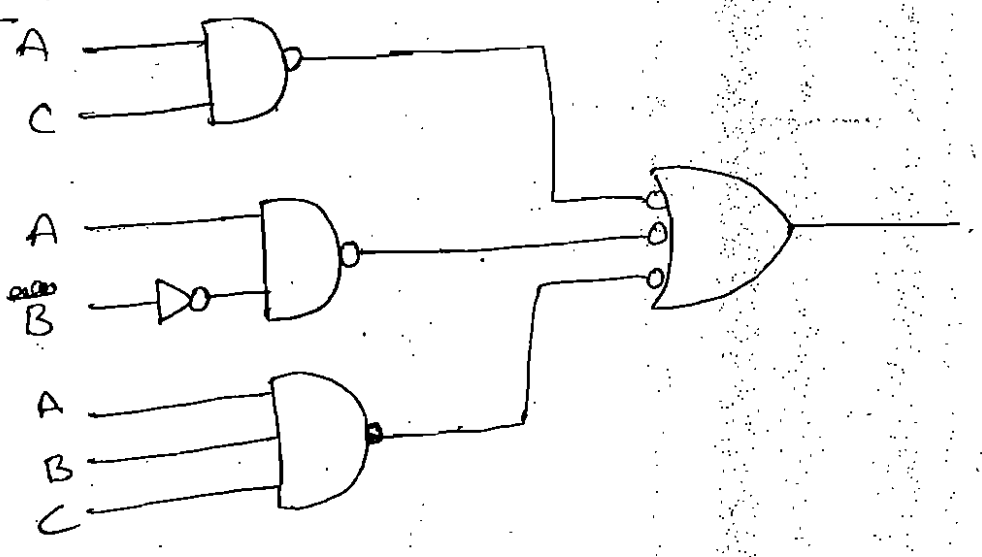


$Y = \overline{(\overline{A}+B)(A+\overline{B})}$

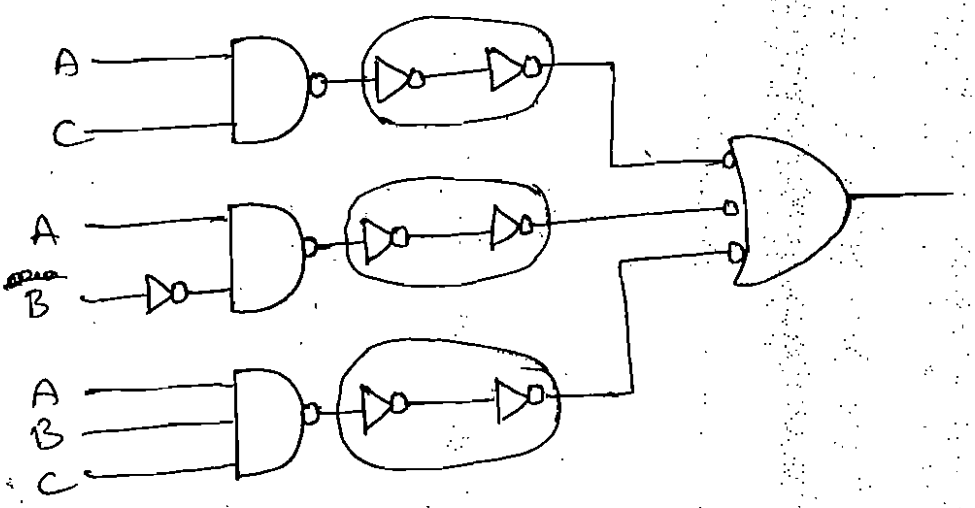
$= \overline{(\overline{A}+B)} + \overline{(A+\overline{B})}$

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

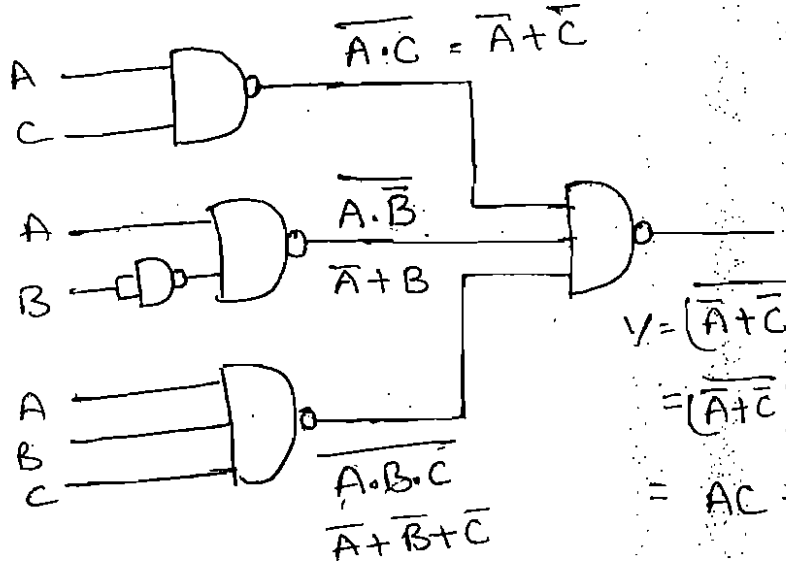
Step 1:-



Step 2:-



Step 3:-



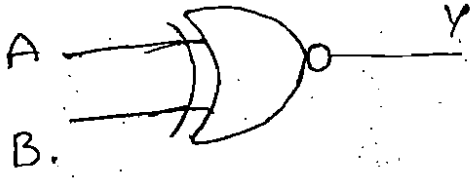
$$Y = \overline{(\overline{A}+\overline{C})(\overline{A}+B)(\overline{A}+\overline{B}+\overline{C})}$$

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

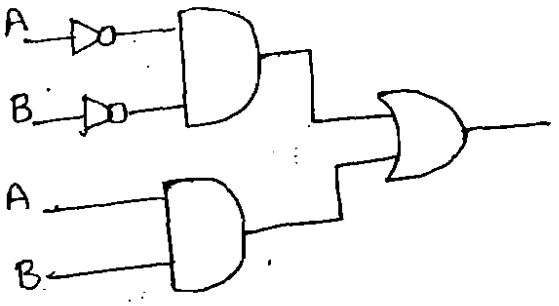
$$= AC + A\overline{B} + \overline{\overline{A}} \cdot \overline{\overline{B}} \cdot \overline{\overline{C}}$$

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

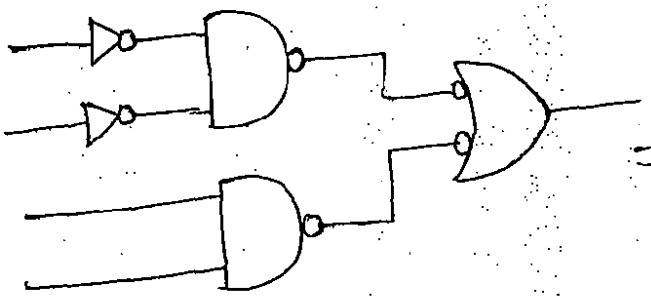
7) EX-NOR Gate :-

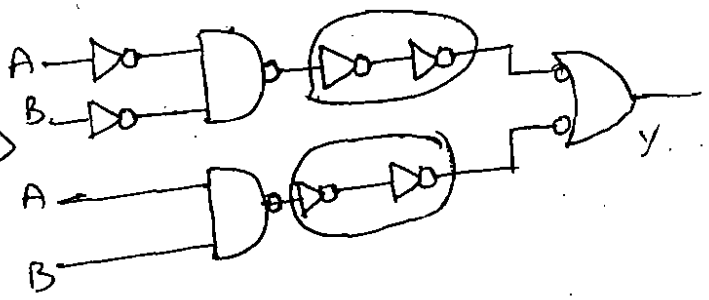


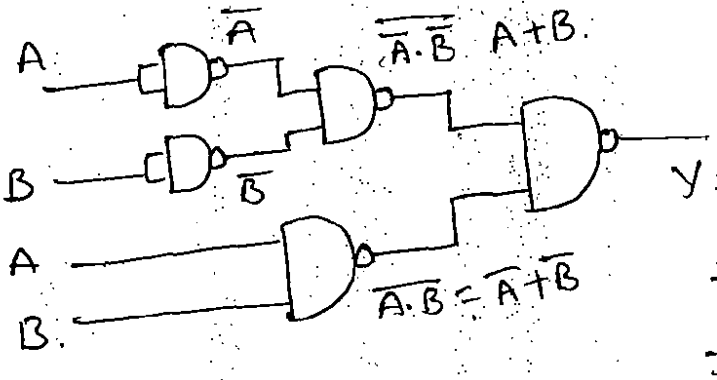
$Y = \overline{A}B + A\overline{B}$.



Step 1 :-







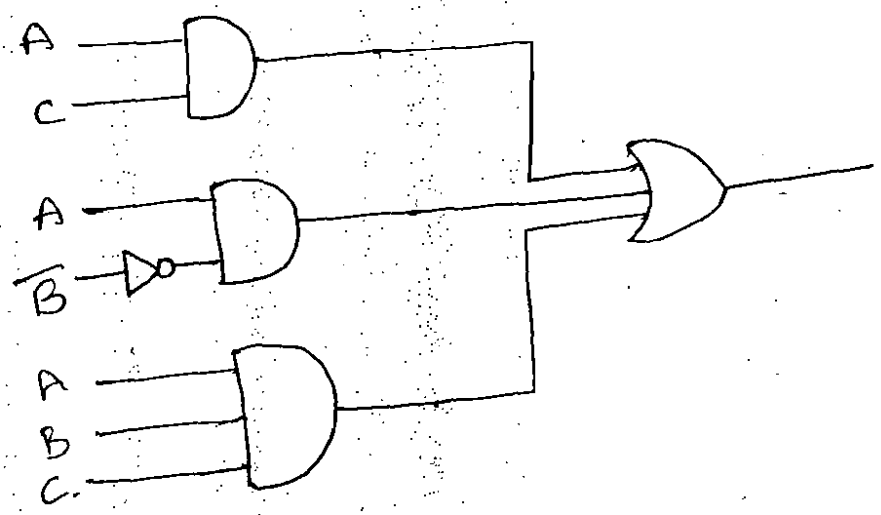
$Y = \overline{(A+B)(\overline{A}+\overline{B})}$

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

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

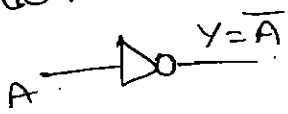
* Implement a given Boolean Expression by using NAND gate.

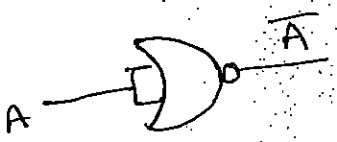
$Y = AC + A\overline{B} + ABC$.



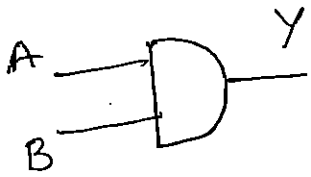
NOR - NOR Realization :-

1) NOT Gate

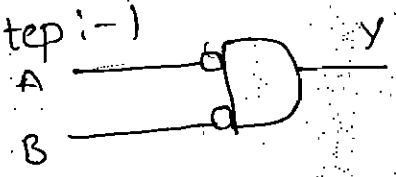




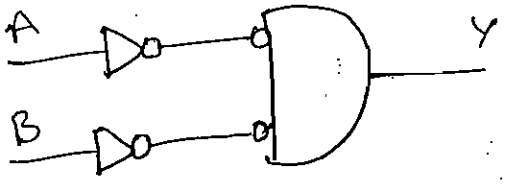
2) AND Gate.



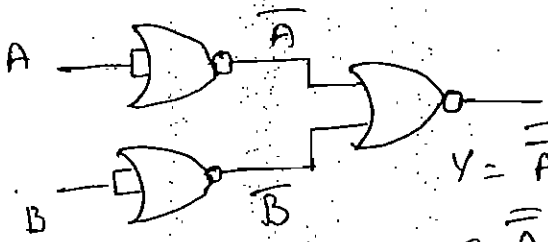
Step :-1



Step 2



Step 3

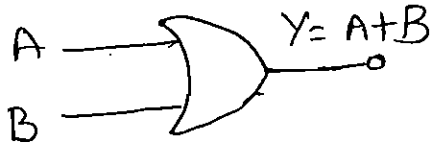


$Y = \overline{\bar{A} + \bar{B}}$

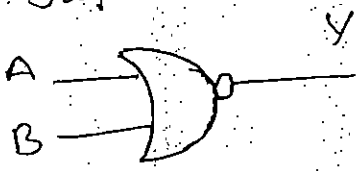
$= \bar{\bar{A}} \cdot \bar{\bar{B}}$

$= A \cdot B.$

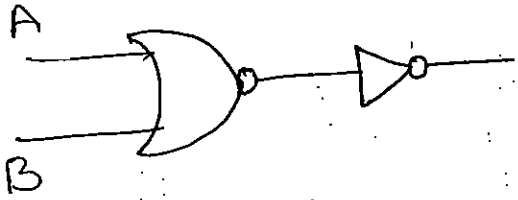
3) OR Gate

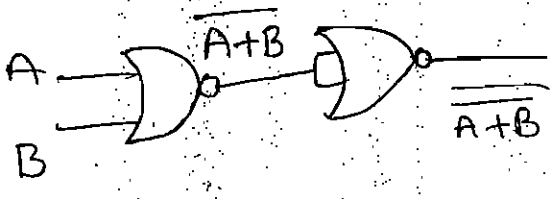


Step :-1



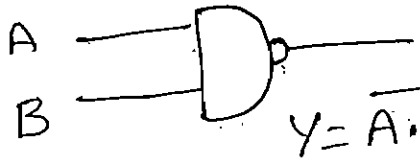
Step :- 2





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

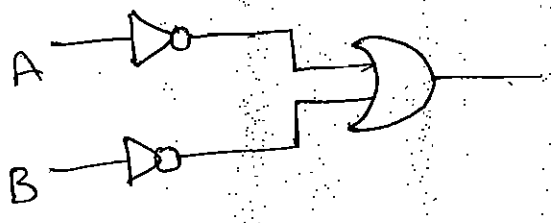
4) NAND Gate



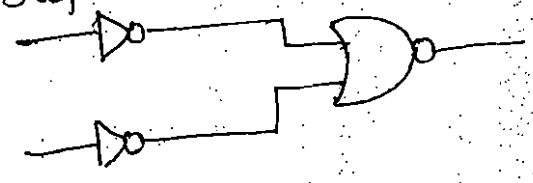
$Y = \overline{A \cdot B}$

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

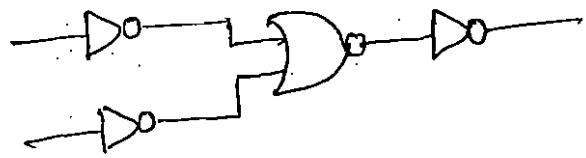
Step :- 1



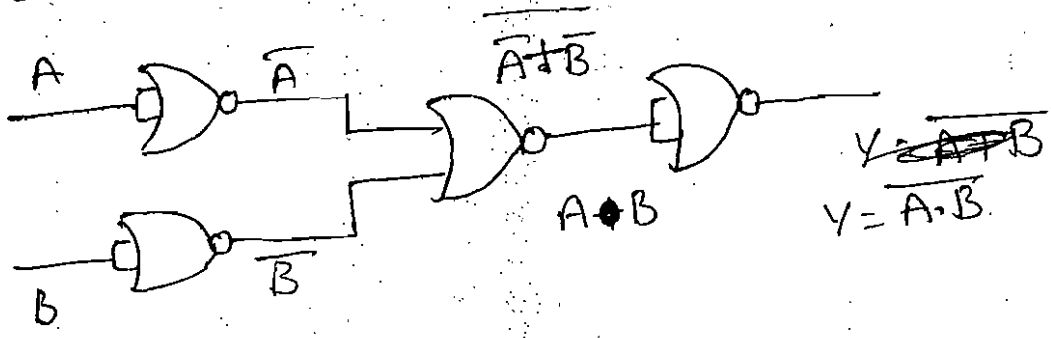
Step 2:-



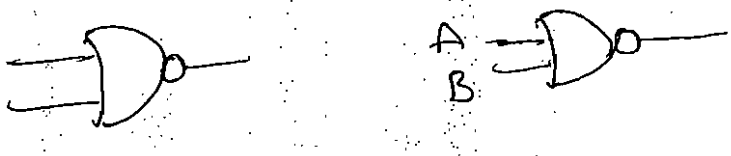
Step 3



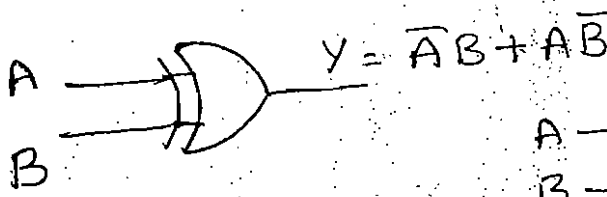
Step 4 :-



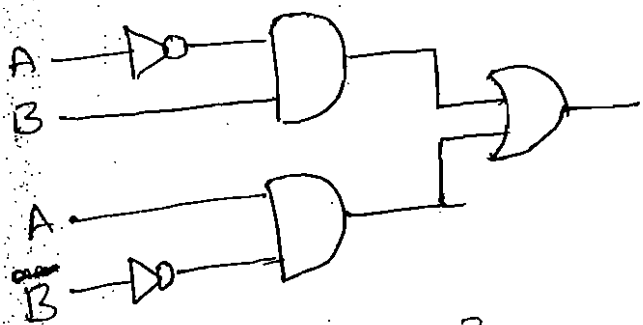
5) NOR Gate :-



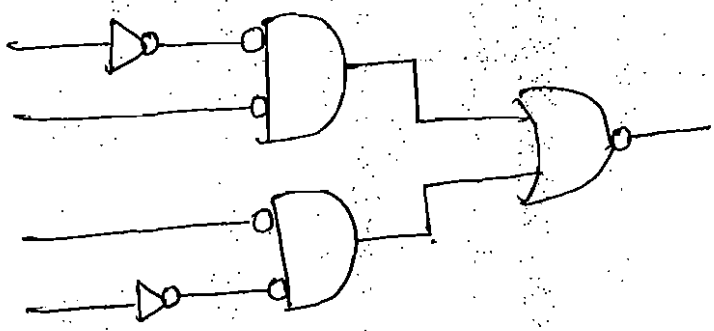
6) EX-OR Gate :-



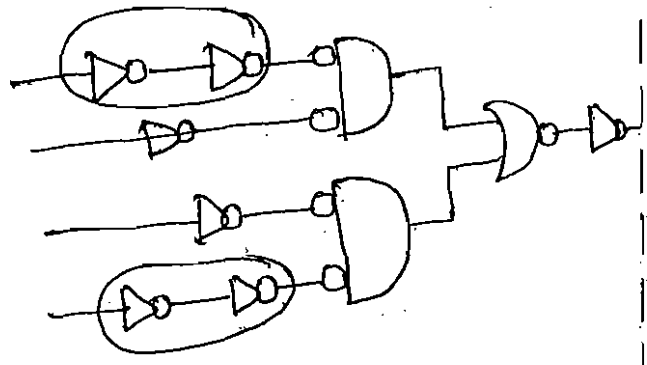
Step :- 1



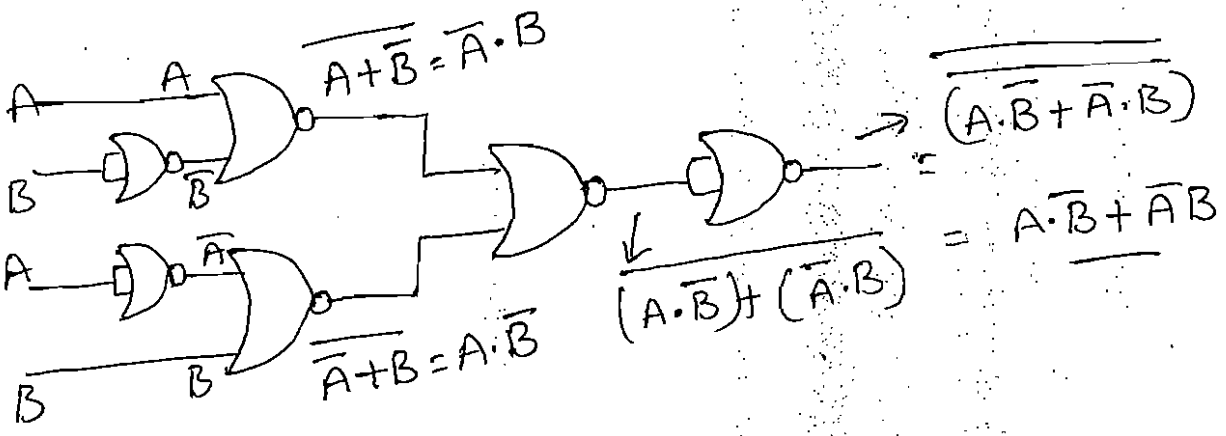
Step :- 2



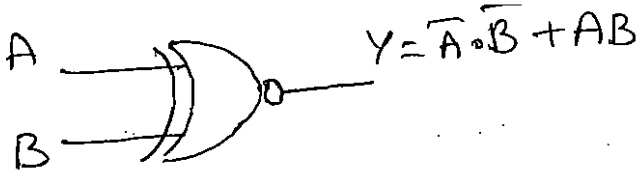
Step :- 3



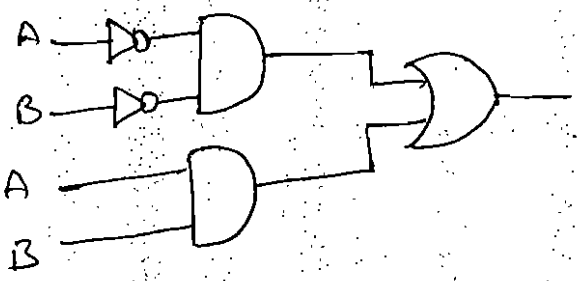
Step 4 :-



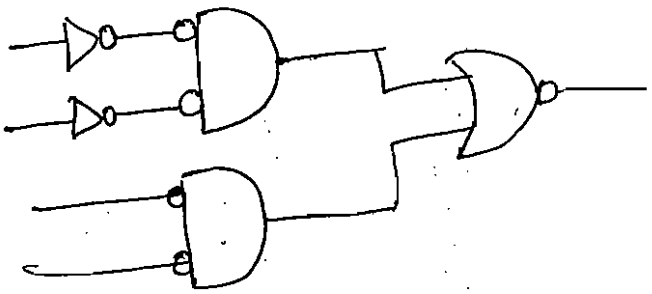
7) EX-NOR gate :-



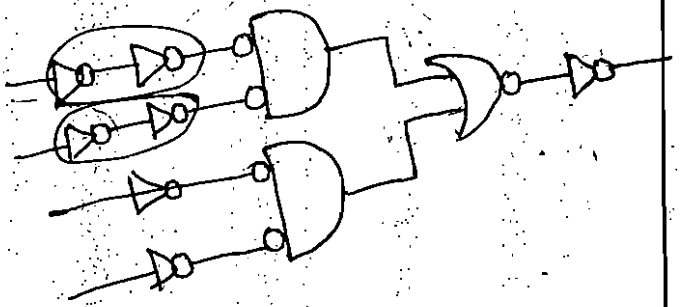
Step 1 :-



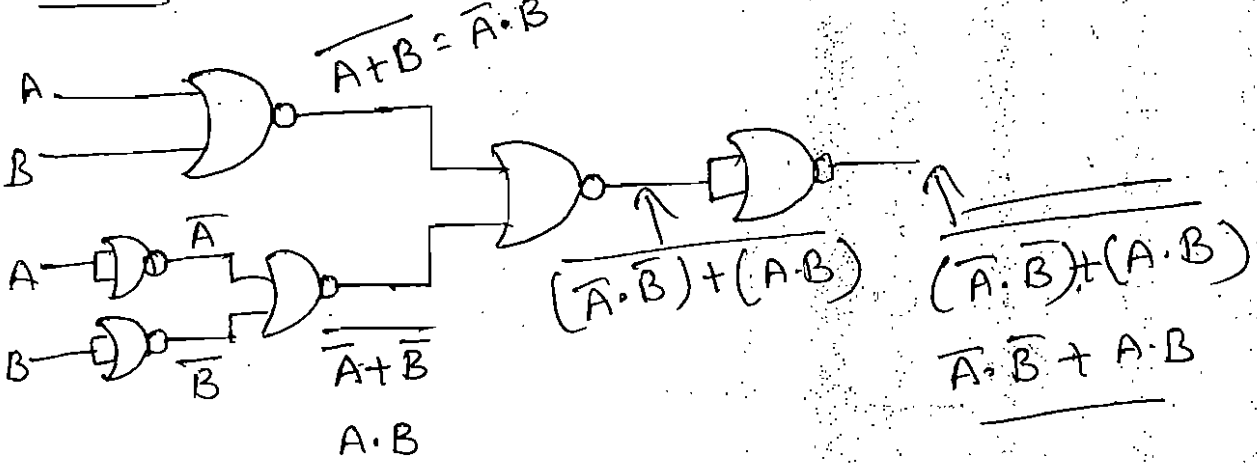
Step 2 :-



Step 3 :-

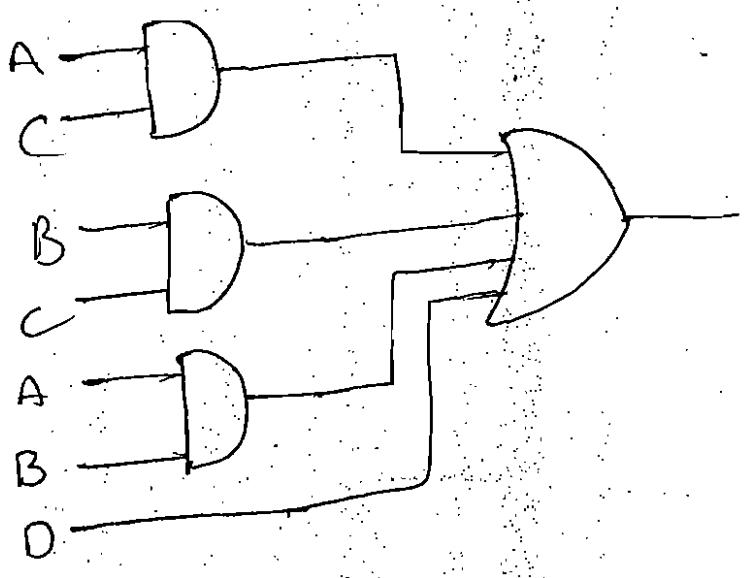


Step 4 :-

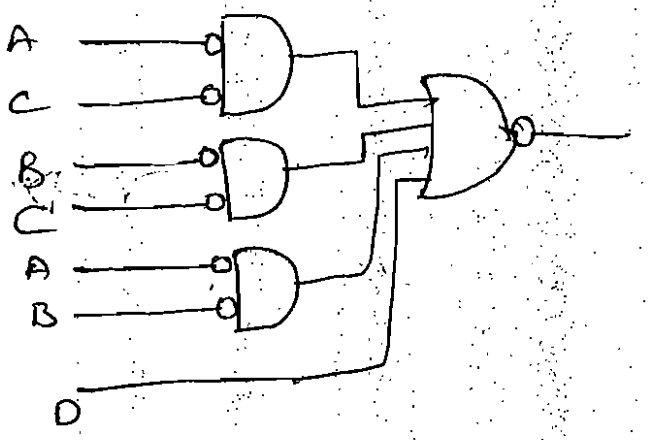


* $Y = AC + BC + AB + D$, implement Boolean Expression by using NOR Gate.

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

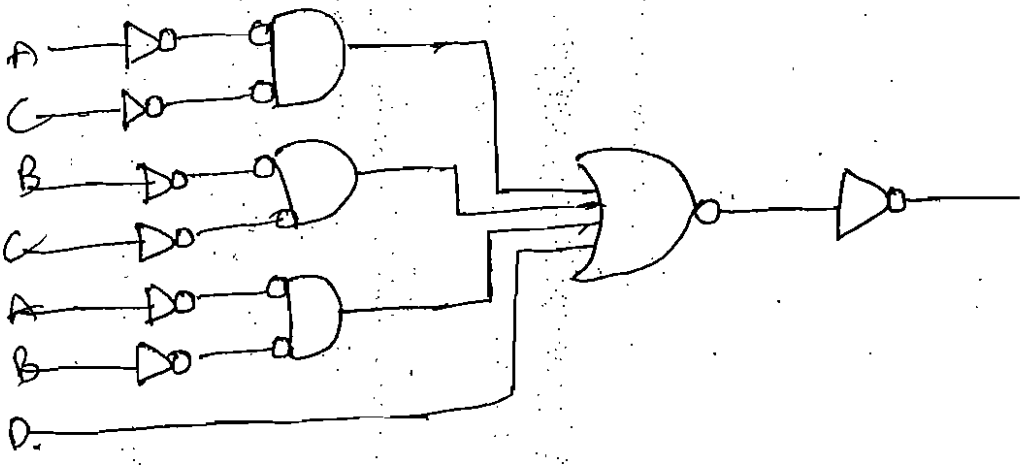


Step :- 1

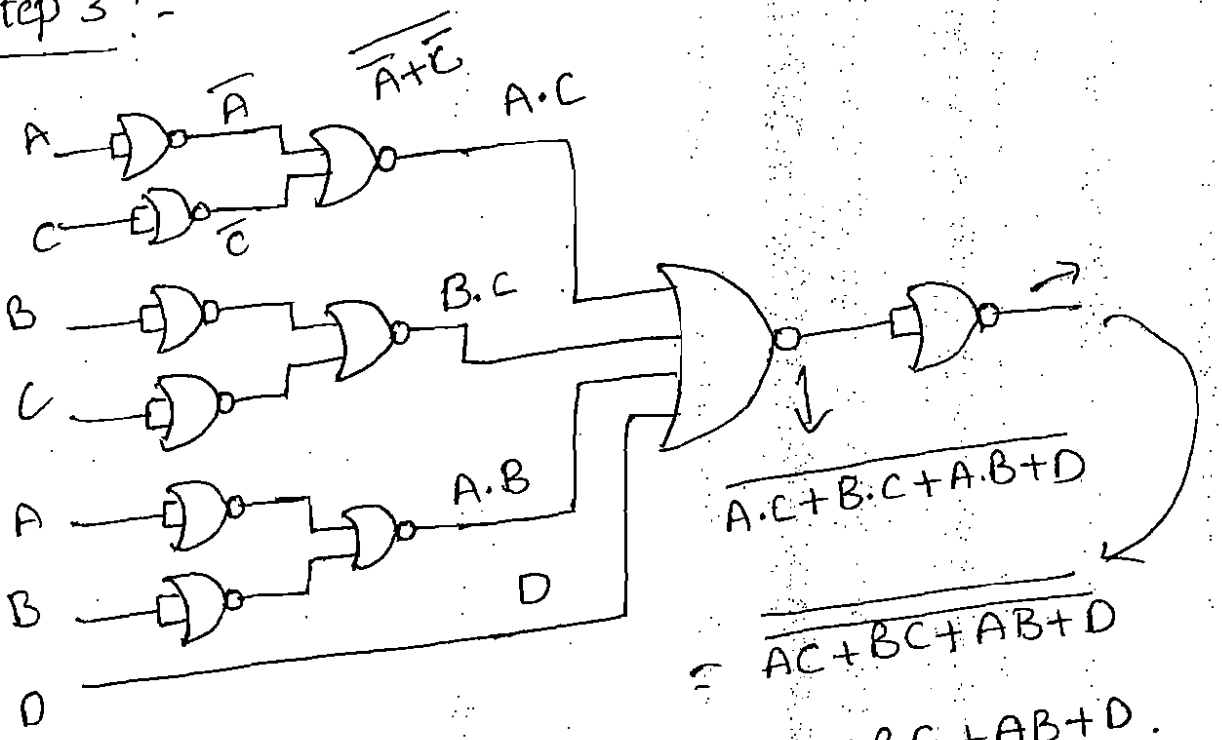


Step :- 2

Step :- 2



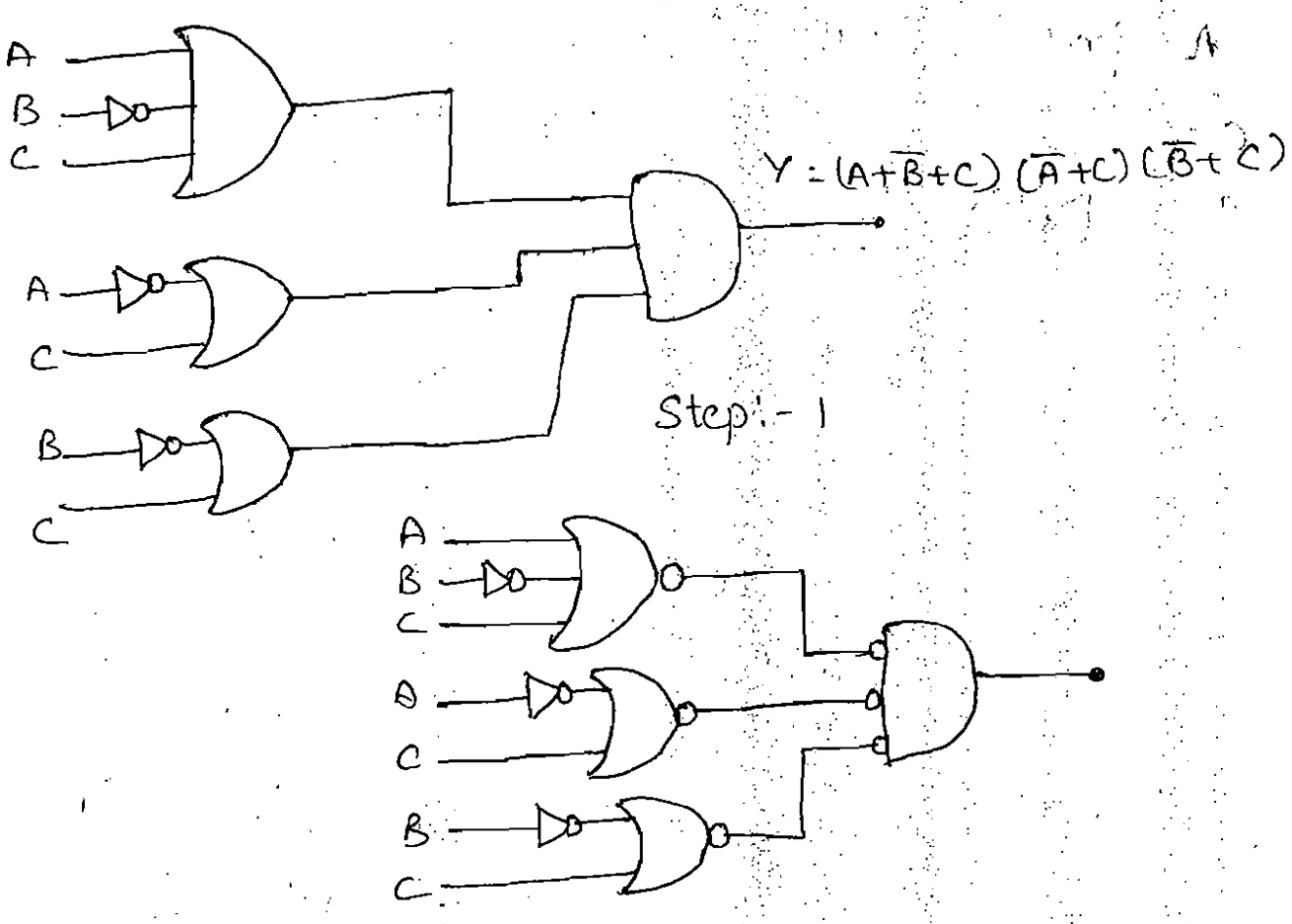
Step 3 :-



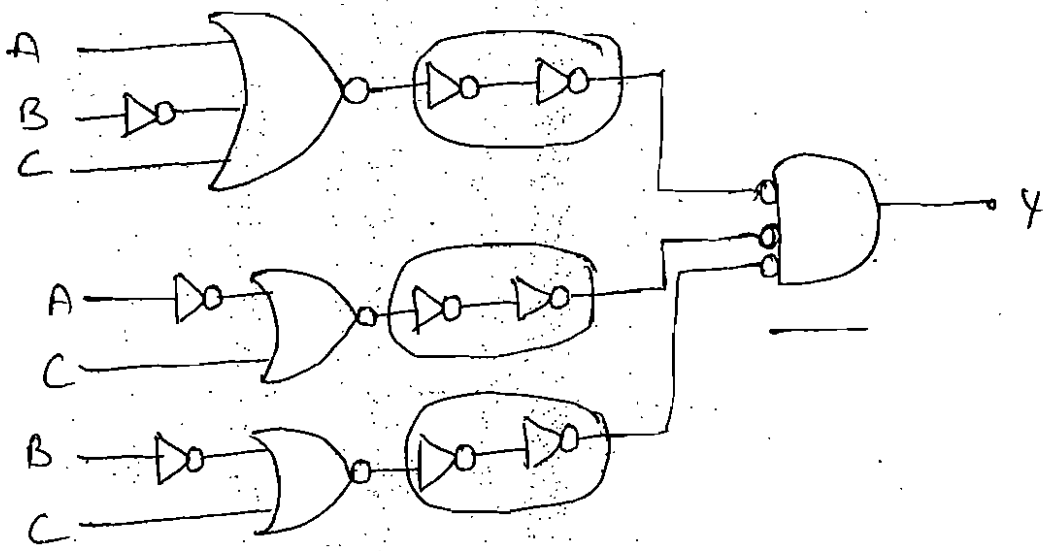
$= \overline{\overline{AC+BC+AB+D}}$

$= AC+BC+AB+D.$

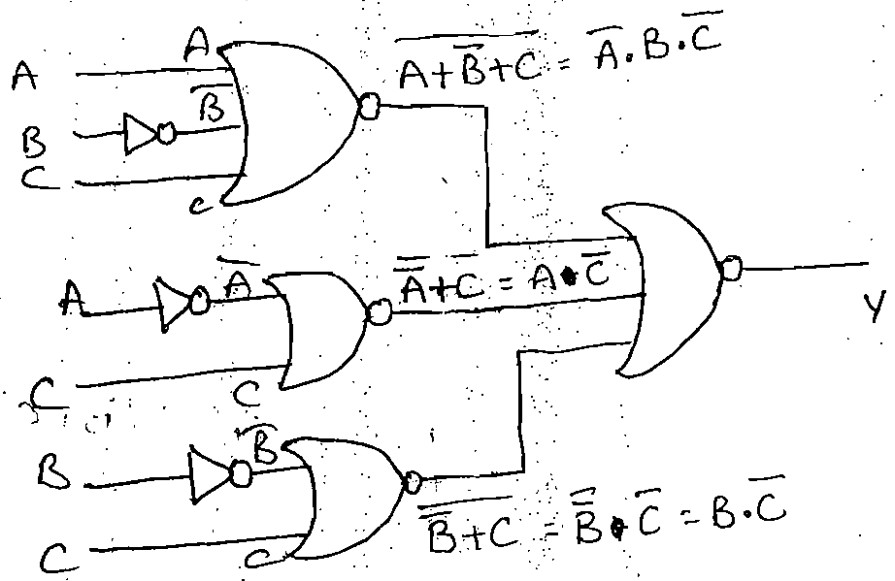
* implement the Boolean Expression by using NOR Gate.
$(A+\bar{B}+C)\ (\bar{A}+C)\ (\bar{B}+C)$.



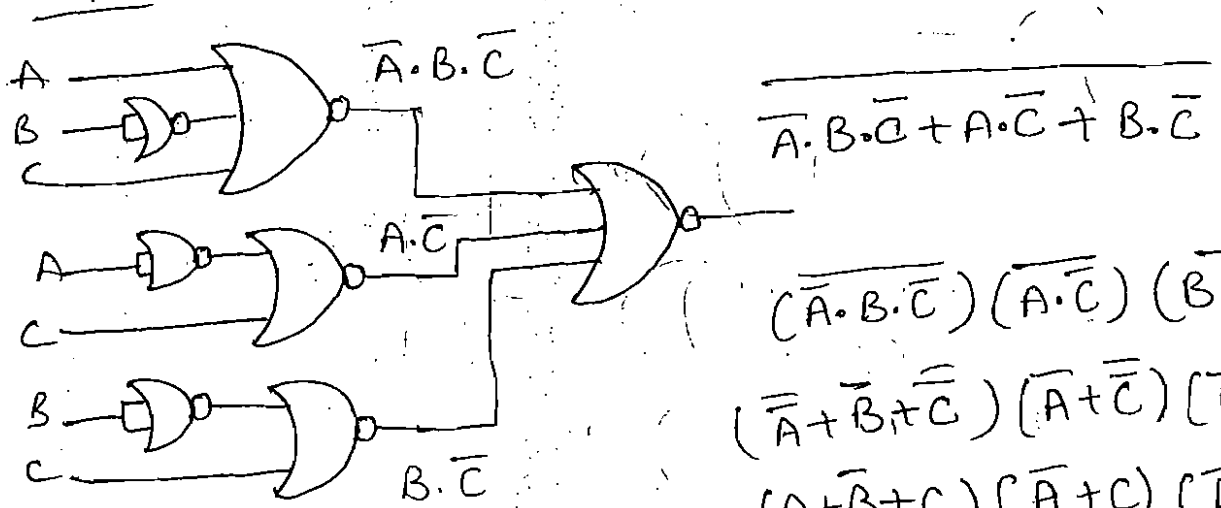
Step 2 :-



Step 3 :-



Step 4 :-



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

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

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

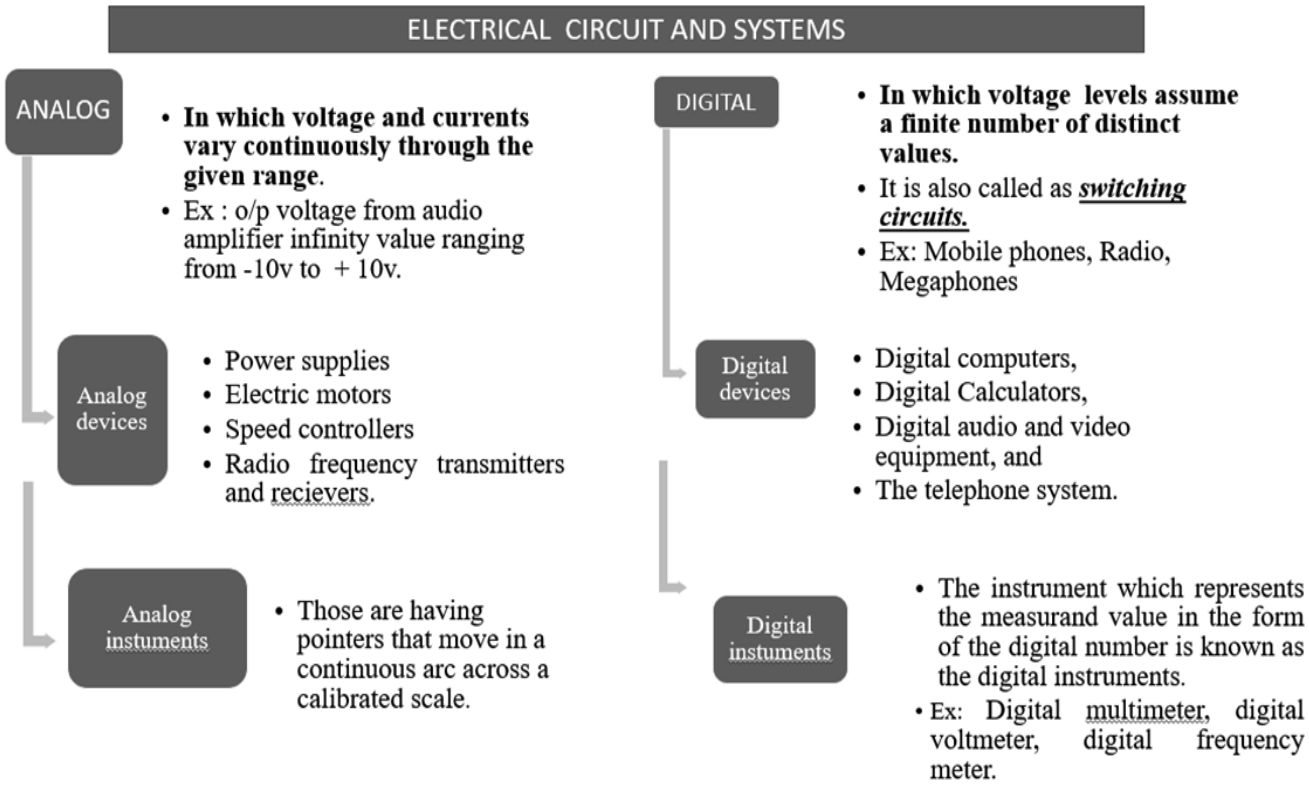
$$= (A+\bar{B}+C)(\bar{A}+C)(\bar{B}+C)$$

MODULE 1

Number systems& Binary codes:

- Number systems: Number Systems, Radix conversions, complement of numbers.
- Binary codes: Binary codes, Weighted and non-Weighted codes, BCD code, gray code, excess 3codes - Error detecting code, Error Correcting code, Hamming Code

INTRODUCTION



DIGITAL CIRCUIT:

- Digital circuit is one in which the voltage levels assume a finite number of distinct values.
- Each voltage level in a practical digital system can actually be a narrow band or range of voltages.
- Also called as switching circuits, the voltage levels in a digital circuit are assumed to be switched from one value to another value instantaneously, that is the transition time is assumed to be zero.

A) COMBINATIONAL SWITCHING CIRCUITS:
   - The output depends only on the present inputs.
   - They have no memory.

B) SEQUENTIAL SWITCHING CIRCUITS
   - The output depends on the present inputs as well as the present state of the circuit, i.e., on the past values also.
   - These are combinational circuits with memory.

❖ SEQUENTIAL SWITCHING CIRCUITS:

a) SYNCHRONOUS SEQUENTIAL CIRCUITS.: Digital sequential circuits in which the feedback to the input for next output generation is governed by clock signals.
b) ASYNCHRONOUS SEQUENTIAL CIRCUITS: Digital sequential circuits in which the feedback to the input for next output generation is not governed by clock signals.

DIGITAL CIRCUIT is also called as Binary signals or Logic signals.

- The digital signals are represented by two voltage bands, one band which is near a reference value (generally 0), and the other band lies near the supply voltage.

- This is similar to the values, '0' and '1' or 'false' and 'true' of the Boolean domain.
- This means that at any particular time, a digital signal can represent only one binary digit.
- The manner in which a logic circuit responds to an input as referred to as the circuit logic.

Application:

- Thermometer, photocopies, landline telephones, audiotape recorders, television, computers, laptops, mobile phones, wristwatches, wall clocks, are all becoming digital nowadays.
- It increases the accuracy of the message as well as makes it easy to read.

Advantages of Digital system or signals:

- Because of the digital nature, the signals in the digital systems can travel significantly faster over digital lines as compared to the Analog signals
- As compared to Analog signals, digital signals can transfer more data.
- The digital systems are less expensive, more reliable, easy to manipulate, and more flexible as compared to the Analog system.
- A digital system can be made compatible with other digital systems to which is not possible in the Analog system.

1. THE DECIMAL SYSTEM

The decimal number system comprises digits from 0-9 that are 0, 1, 2, 3, 4, 5, 6, 7, 8 & 9. The base or radix of the decimal number system is 10 because the total number of digits available in the decimal number system is 10. All the other digits can be expressed with the help of these 10 digit numbers.

number **345** represents:

$$= 3 * 10^2 + 4 * 10^1 + 5 * 10^0$$
$$= 3 * 100 + 4 * 10 + 5$$
$$= 300 + 40 + 5$$
$$= 34$$

the value **123.456** means:

$$= 1 * 10^2 + 2 * 10^1 + 3 * 10^0 + 4 * 10^{-1} + 5 * 10^{-2} + 6 * 10^{-3}$$
$$= 100 + 20 + 3 + 0.4 + 0.05 + 0.006$$

2. THE BINARY SYSTEM

Binary number system can be said to be the simplest one in the number system. It uses only two digits (0 and 1) to represent a number. Thus, as the 'bi' in its name suggests, the system uses 2 as a base. The entire number system can be represented through the binary system. For example, fractions, real numbers, as well as large numbers, can be represented through binary numbers.

BINARY TO DECIMAL CONVERSION

The binary numbering system works just like the decimal numbering system, with two exceptions:

- binary only allows the digits 0 and 1 (rather than 0–9), and
- binary uses powers of two rather than powers of ten.

Therefore, it is very easy to convert a binary number to decimal. For each "1" in the binary string, add 2n where "n" is the bit position in the binary string (0 to n–1 for n bit binary string).

For example, the binary value 1010_2 represents the decimal 10 which can be obtained through the procedure shown in the table 1:

Table 1

Binary No.	1	0	1	0
Bit Position (n)	3^{rd}	2nd	1^{st}	0th
Weight Factor (2n)	2^3	2^2	2^1	2^0
bit * 2n	$1*2^3$	$0*2^2$	$1*2^1$	$0*2^0$
Decimal Value	8	0	2	0

Decimal Number 8 + 0 + 2 + 0 = 10

All the steps in above procedure can be summarized in short as

$1*2^3 + 0*2^2 + 1*2^1 + 0*2^0 = 8 + 0 + 2 + 0 = 1010$

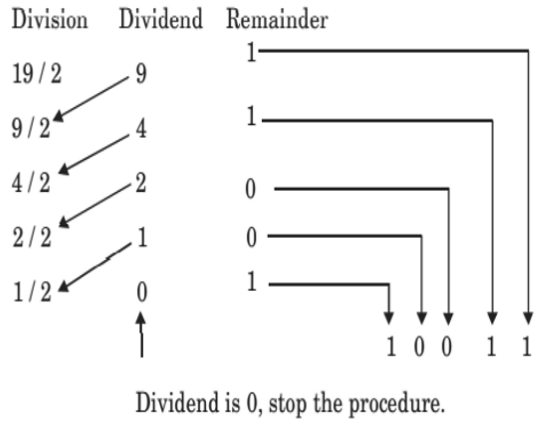
i.e.,

1. Multiply each digit of the binary number by its positional weight and then add up the result.

2. If any digit is 0, its positional weight is not to be taken into account.

DECIMAL TO BINARY CONVERSION

let us find out binary of 19_{10} (decimal 19).



1. The right most bit in a binary number is bit position zero.
2. Each bit to the left is given the next successive bit number.

- An eight-bit binary value uses bits zero through seven:
- X7 X6 X5 X4 X3 X2 X1 X0
- A 16-bit binary value uses bit positions zero through fifteen:

X15 X14 X13 X12 X11 X10 X9 X8 X7 X6 X5 X4 X3 X2 X1 X0

- Bit zero is usually referred to as the ***low order bit.***

or

Least significant bit (LSB).

- The left-most bit is typically called the ***high order bit***.

or

Most significant bit (msb)

3. OCTAL NUMBERING SYSTEM:

- The octal number system uses base 8 instead of base 10 or base 2.
- This is sometimes convenient since many computer operations are based on bytes (8 bits). In octal, we have 8 digits at our disposal, 0–7.

DECIMAL OCTAL

0 0

1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	10
9	11
10	12
11	13
12	14
13	15
14	16
15	17
16	20

Octal to Decimal,

Converting octal to decimal is just like converting binary to decimal, except instead of powers of 2, we use powers of 8.

To convert 172 in octal to decimal:

$$\begin{array}{rl} & \begin{array}{ccc} 1 & 7 & 2 \\ 8^2 & 8^1 & 8^0 \end{array} \\ \text{Weight} & = 1*8^2 + 7*8^1 + 2*8^0 \\ & = 1*64 + 7*8 + 2*1 \\ & = 122_{10} \end{array}$$

Decimal to Octal Conversion,

Converting decimal to octal is just like converting decimal to binary, except instead of dividing by 2, we divide by 8.

To convert 122 to octal:

$$\begin{aligned} 122/8 &= 15 \text{ remainder } 2 \\ 15/8 &= 1 \text{ remainder } 7 \\ 1/8 &= 0 \text{ remainder } 1 \\ &= 172_8 \end{aligned}$$

Octal to binary

Convert $(145056)_8$ to binary.

To convert from octal to binary and vice versa we will need this conversion table.

value $(145056)_8$ can be converted to binary as $(001\ 100\ 101\ .\ 101\ 110)_2$

OCTAL SYMBOL	BINARY CODE
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Binary to Octal:

We can use the same table to convert a binary number to octal number. And for that, we first have to group the binary number into a group of three bits and write the octal equivalent of it.

Convert the binary number $(11001111)_2$ to octal
The three bit group of binary numbers can be written as 011,001,111 because we have to add a zero before each number to complete the in the form of three binary digits. Therefore, the octal numbers will be 3, 1, 7 i.e., $(317)_8$

3. HEXADECIMAL NUMBERING SYSTEM

- Hexadecimal uses a base 16 numbering system. This means that we have 16 symbols to use for digits. Consequently, we must invent new digits beyond 9.
- The digits used in hex are the letters A, B, C, D, E, and F.

Hexa decimal to Decimal

Converting hex to decimal is just like converting binary to decimal, except instead of powers of 2, we use powers of 16.

To convert 15E in hex to decimal:

$$\begin{aligned} & 1 \quad 5 \quad E \\ & 16^2 \quad 16^1 \quad 16^0 \\ \text{Weight} &= 1*16^2 + 5*16^1 + 14*16^0 \\ &= 1*256 + 5*16 + 14*1 \\ &= 350_{10} \end{aligned}$$

Decimal to Hex Conversion

Converting decimal to hex is just like converting decimal to binary, except instead of dividing by 2, we divide by 16. To convert 350 to hex:

$$\begin{aligned} 350/16 &= 21 \text{ remainder } 14 = E \\ 21/16 &= 1 \text{ remainder } 5 \\ 1/16 &= 0 \text{ remainder } 1 \end{aligned}$$

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

Hexa to Octal

A group of 4-bits represent a hexadecimal digit and a group of 3-bits represent an octal digit.

1. Convert the given hexadecimal number into binary.
2. Starting from right make groups of 3-bits and designate each group an octal digit.

. *Convert $(1A3)_{16}$ into octal.*

1. Converting hex to binary

$$(1\ A\ 3)_{16} = \underbrace{0001}_{1}\ \underbrace{1010}_{A}\ \underbrace{0011}_{3}$$

2. Grouping of 3-bits

$$(1A3)_{16} = \underset{0}{\underbrace{000}}\ \underset{6}{\underbrace{110}}\ \underset{4}{\underbrace{100}}\ \underset{3}{\underbrace{011}}$$

so $(1A3)_{16} = (0643)_8 \equiv (643)_8$

Octal to Hex Conversion

1. Convert the given octal number into binary.
2. Starting from right make groups of 4-bits and designate each group as a Hexadecimal digit.

Convert $(76)_8$ into hexadecimal.

Solution. 1. Converting octal to binary

$$(76)_8 = \underbrace{111}_{7}\ \underbrace{110}_{6}$$

2. Grouping of 4-bits

$$(76)_8 = \underset{3}{\underbrace{11}}\ \underset{E}{\underbrace{1110}} \equiv \underset{3}{\underbrace{0011}}\ \underset{E}{\underbrace{1110}}$$

$\therefore$ $(76)_8 = (3E)_{16}$

THE BINARY ARITHMETIC OPERATIONS

- Binary arithmetic's are simpler than decimal because they involve only two digits (bits) 1 and 0.
- Addition, subtraction, multiplication and division.

Binary Addition

Augend	*Addend*	*Sum*	*Carry*	*Result*
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	0	1	10

Binary subtraction

Minuend	*Subtrahend*	*Difference*	*Borrow*
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

(i) Add 1010 and 0011 (ii) Add 0101 and 1111

```
  1 1 1   ← Carry
  0 1 0 1
+ 1 1 1 1
1 0 1 0 0
↑
Carry
```

```
  1 1 1   ← Carry
  0 1 0 1
+ 1 1 1 1
1 0 1 0 0
↑
Carry
```

(i) Subtract 0100 from 1011

1				←Borrow
1	0	1	1	←Minuend
−0	1	0	0	←Subtrahend
0	1	1	1	←Difference
↑	↑	↑	↑	
C_3	C_2	C_1	C_0	

(ii) *Subtract 0110 from 1001*

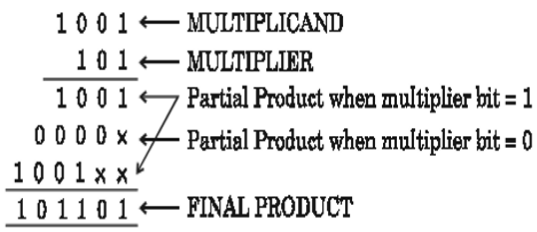
1	1			←Borrow
1	0	0	1	←Minuend
−0	1	1	0	←Subtrahend
0	0	1	1	←Difference
↑	↑	↑	↑	
C_3	C_2	C_1	C_0	

The rules are still the same as in decimal, except that the borrow in a given significant position adds 2 to a minuend digit.

Binary multiplication

Multiplicand	Multiplier	product
0	0	0
1	0	0
1	1	1
0	1	0

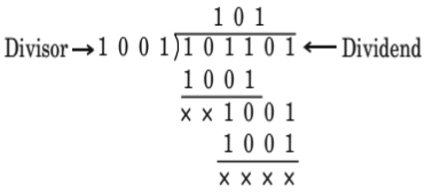
(i) Multiply 1001 with 101



Binary division.

Divisor	dividend	Q
X	Y	IF X< Y, Q = 1
X	Y	IF X>Y, Q = 0

Binary division is also similar to decimal division



NEGATIVE NUMBERS AND THEIR ARITHMETIC

So far, we have discussed straight forward number representation which are nothing but positive number. The negative numbers have got two representation

- **Complement representation**. In digital computers to simplify the subtraction operation & for logical manipulation complements are used. There are two types of complements used in each radix system.
  - The radix complement or r's complement
  - The diminished radix complements or (r-1)'s complement

r's Complement and (r – 1)'s Complement

The r's and (r – 1)'s complements are generalized representation of the complements. r stands for radix or base of the number system; thus, r's complement is referred as radix complement and (r – 1)'s complement is referred as diminished radix complement. Examples of r's complements are 2's complement and 10's complement. Examples of (r – 1)'s complement are 1's complement and 9's

In a base-r system, the r's and $(r-1)$'s complement of the number N having n digits, can be defined as:

$$\boxed{(r-1)\text{'s complement of } N = (r^n - 1) - N}$$

and

$$\boxed{\begin{aligned} r\text{'s complement of } N &= r^n - N \\ &= (r-1)\text{'s complement of } N + 1 \end{aligned}}$$

The $(r-1)$'s complement can also be obtained by subtracting each digit of N from $r-1$. Using the above methodology we can also define the 7's and 8's complement for octal system and 15's and 16's complement for hexadecimal system.

- **Sign magnitude representation** : Representation of signed no's binary arithmetic in computers: Two ways of representation of signed no's Sign Magnitude for Complemented form, Two complimented forms: 1's compliment form, & 2's compliment form

  1's and 2's Complement: These are the complements used for binary numbers. Their representation are very important as digital systems work on binary numbers only.

1's Complement

bit	Actual binary	complement

1's complement of a binary number is obtained simply by replacing each 1 by 0 and each 0 by 1. Alternately, 1's complement of a binary can be obtained by subtracting each bit from 1.

1's Complement	1	0
	0	1

EX: Find 1's complement of (i) 011001 (ii) 00100111

Sol: (i) Replace each 1 by 0 and each 0 by 1

0 1 1 0 0 1

↓ ↓ ↓ ↓ ↓ ↓

1 0 0 1 1 0

So, 1's complement of 011001 is 100110.

2's Complement: 2's complement of a binary number can be obtained by adding 1 to its 1's complement.

EX: Find 2's complement of (i) 011001 (ii) 010110016

Solution.

(i)

0 1 1 0 0 1	← Number	
1 0 0 1 1 0	← 1's complement	
+ 1	← Add 1 to 1's complement	
1 0 0 1 1 1	← 2's complement	

(ii)

0 1 0 1 1 0 0	← Number
1 0 1 0 0 1 1	← 1's complement
+ 1	← Add 1 to 1's complement
1 0 1 0 1 0 0	← 2's complement

Subtraction Using 1's and 2's Complement

Before using any complement method for subtraction equate the length of both minuend and subtrahend by introducing leading zeros.

1's complement subtraction following are the rules for subtraction using 1's complement.

1. To do the subtraction (M-S), represent the M&S in equal no. of digits.

2. Add 1's complement of subtrahend to minuend.

3. If a carry is produced by addition, then add this carry to the LSB of result. This is called as end around carry (EAC).

4. If carry is generated from MSB in step 2 then result is positive. If no carry generated result is negative, and is in 1's complement form.

EX: Perform binary subtraction for $(23)_{10}$-$(11)_{10}$

Sol: M= 23, 10111

S= 11, 1011

Step 1: represent the M&S in equal no. of digits. 10111=23

01011=11

Step 2: 1's complement of subtrahend (01011) = 10100

Add 1's complement of subtrahend to minuend 10111= M

\+ 10100=1'S Comp of S

Carry 101011

Step3: If a carry is produced by addition, then add this carry to the LSB of result.

01011

\+ 1

01100 =12

2's complement Subtraction:

Method of 2's complement is similar to 1's complement subtraction except the end around carry (EAC). The rules are listed below:

1. To do the subtraction (M-S), represent the M&S in equal no. of digits.
2. Take 2's complement of subtrahend. Add 2's complement of subtrahend to minuend.

3. If a carry is produced, then discard the carry and the result is positive. If no carry is produced result is negative and is in 2's compliment form.

EX: Perform binary subtraction for $(22)_{10}$-$(12)_{10}$ Using 2's complement

Sol: M= 22, 10110

S= 12, 1100

Step 1: represent the M&S in equal no. of digits. 10110=22

01100=12

Step 2: 2's complement of subtrahend (01100) = 10100

Add 2's complement of subtrahend to minuend 10110= M

\+ 10100=1'S Comp of S

Carry 101010

(Neglected)

Step 3: If a carry is produced, then discard the carry and the result is positive = (01010) = $(10)_{10}$

Signed Binary Representation

Untill now we have discussed representation of unsigned (or positive) numbers, except one or two places. In computer systems sign (+ve or –ve) of a number should also be represented by binary bits.

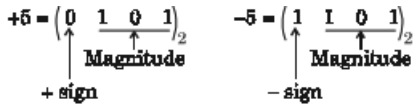
The accepted convention is to use 1 for negative sign and 0 for positive sign. In signed representation MSB of the given binary string represents the sign of the number, in all types of representation. We have two types of signed representation:

1. Signed Magnitude Representation

2. Signed Complement Representation

sign magnitude representation.

In a signed-magnitude representation, the MSB represent the sign and rest of the bits represent the magnitude. e.g.,



Note that positive number is represented similar to unsigned number. From the example it is also evident that out of 4-bits, only 3-bits are used to represent the magnitude.

What is sign magnitude of +5 and -7?

Sign bit	Actual binary number
+ is (0)	X
- Is (1)	X

Sol: actual number 5 is 0101 in binary number system. But to represent signed number in computer it has to represent in 8-bit binary number then

5 → 0000101 → 8 bit binary

+5 → 00000101 → signed magnitude for positive

-5 → 10000101 → signed magnitude for negative

Complement of signed magnitude representation

In a signed-complement representation the positive numbers are represented in true binary form with MSB as 0. Whereas the negative numbers are represented by taking appropriate complement of equivalent positive number, including the sign bit. Both 1's and 2's complements can be used for this purpose e.g.,

$+5 = (0101)_2$

$-5 = (1010)_2$ ←in 1's complement

$= (1011)_2$ ←in 2's complement

What is sign magnitude of +5 and -7 in 1's complement and 2's complement form

Sign bit	*1's complement* Of actualnumber	*2's complement* Of actual number
+ is (0)	X	X +1
- Is (1)	X	X+1

+5 → 00000101 → signed magnitude number

→ 11111010 → 1's complement of signed magnitude number

+1

→ 11111011 → 2's complement of signed magnitude number

9's and 10's Complement

9's and 10's complements are the methods used for the representation of decimal numbers. They are identical to the 1's and 2's complements used for binary numbers.

9's complement: 9's complement of a decimal number is defined as (10n – 1) – N, where n is no. of digits and N is given decimal numbers. Alternately, 9's complement of a decimal number can be obtained by subtracting each digit from 9.

9's complement of $N = (10^n – 1) – N$

EX: Find out the 9's complement of $(36)_{10}$.

Sol: By using $(10^n–1) – N$; n = 2. So, $(10^n–1) – N = (100 – 1) – 36=63$

By subtracting each digit from 9

$$\begin{array}{r} 9\ 9 \\ -3\ 6 \\ \hline 6\ 3 \end{array}$$

So, 9's complement of 36 is 63.

10's complement: 10's complement of a decimal number is defined as $10^n – N$.10's complement of $N = 10^n – N$ (or)

$10^n – N = (10^n – 1) – N + 1$= 9's complement of N + 1. Thus, 10's complement of a decimal number can also be obtained by adding 1 to its 9's complement.

EX: Find out the 10's complement of $(36)_{10}$.

Solution. By adding 1 to 9's complement

$$\begin{aligned} \text{9's complement of 36} &= 99 - 36 \\ &= 63 \\ \text{Hence, 10's complement of 36} &= 63 + 1 \\ &= 64 \end{aligned}$$

CODES

Coding and encoding is the process of assigning a group of binary digits, commonly referred to as 'bits', to represent, identify, or relate to a multivalued items of information. In short, a code is a symbolic representation of an information transform. The bit combination is referred to as 'CODEWORDS'.

In a broad sense we can classify the codes into five groups:

(i) Weighted Binary codes (ii) Non-weighted codes (iii) sequential codes(iv) Error–detecting codes (v) Error–correcting codes (vi) Alphanumeric codes

i) Weighted Binary Codes

In weighted binary codes, each position of a number represents a specific weight. The bits are multiplied by the weights indicated; and the sum of these weighted bits gives the equivalent decimal digit.

a) **Straight Binary coding:** is a method of representing a decimal number by its binary equivalent. A straight binary code representing decimal 0 through 7

Decimal	*Three bit straight Binary Code*	*Weights MOI* 2^2	2^1	2^0	*Sum*
0	000	0	0	0	0
1	001	0	0	1	1
2	010	0	2	0	2
3	011	0	2	1	3
4	100	4	0	0	4
5	101	4	0	1	5
6	110	4	2	0	6
7	111	4	2	1	7

b) **Binary Codes Decimal Codes (BCD codes).** In BCD codes, individual decimal digits are coded in binary notation and are operated upon singly. Thus, binary codes representing 0 to 9 decimal digits are allowed. Therefore, all BCD codes have at least four bits (∵ min. no. of bits required to encode to decimal digits = 4) For example, decimal 364 in BCD

3 → 0011

$6 \rightarrow 0110$

$4 \rightarrow 0100$

$364 \rightarrow 0011\ 0110\ 0100$

However, we should realize that with 4 bits, total 16 combinations are possible (0000, 0001, ..., 1111) but only 10 are used (0 to 9). The remaining 6 combinations are invalid and commonly referred to as 'UNUSED CODES'

ii) Non weighted codes

Non weighted codes are codes that are not positionally weighted. That is, each position within the binary number is not assigned a fixed value. Ex: Excess-3 code, Gray code.

Excess-3 Code

Excess-3 is a non-weighted code used to express decimal numbers. The code derives its name from the fact that each binary code is the corresponding 8421 code plus 0011(3).

$[643]_{10}$ into XS3 code

Decimal	**6**	**4**	**3**
Add 3 to each	**3**	**3**	**3**
Sum	**9**	**7**	**6**

Converting the sum into BCD code we have

9	**7**	**6**
↓	↓	↓
1001	**0111**	**0110**

Hence, XS3 for $[643]_{10}$ = 1001 0111 0110

Gray Code

The Gray code belongs to a class of codes called minimum change codes, in which only one bit in the code changes when moving from one code to the next. The Gray code is non-weighted code, as the position of bit does not contain any weight. The Gray code is a reflective digital code which has the special property that any two subsequent numbers codes differ by only one bit. This is also called a unit-distance code. In digital Gray code has got a special place.

Gray codes*

Decimal Digit	*Three bit Gray code*	*Four bit Gray code*	*Decimal Digit*	*Three bit Gray code*	*Four bit Gray code*
0	0 0 0	0 0 0 0	8	–	1 1 0 0
1	0 0 1	0 0 0 1	9	–	1 1 0 1
2	0 1 1	0 0 1 1	10	–	1 1 1 1
3	0 1 0	0 0 1 0	11	–	1 1 1 0
4	1 1 0	0 1 1 0	12	–	1 0 1 0
5	1 1 1	0 1 1 1	13	–	1 0 1 1
6	1 0 1	0 1 0 1	14	–	1 0 0 1
7	1 0 0	0 1 0 0	15	–	1 0 0 0

Binary to Gray conversion:

N bit binary no is rep by $B_n\ B_{n-1}$ ---------- B_1
Gray code equivalent is by $G_n\ G_{n-1}$ ---------- G_1

B_n,, G_n are the MSB's then the gray code bits are obtaind from the binary code as

$G_n=B_n$	$G_{n-1}=B_n \oplus B_{n-1}$	$G_{n-2}=B_{n-1} \oplus B_{n-}$	----------	$G_1=B_2 \oplus B1$	

$\oplus$ →EX-or symbol

Procedure: ex-or the bits of the binary no with those of the binary no shifted one position to the right . The LSB of the shifted no. is discarded & the MSB of the gray code no.is the same as the MSB of the original binaryno.

EX: 10001 $\oplus$ $\oplus$ $\oplus$

(a). Binary : 1 →0 →0 →1

Gray : 1 1 0 1

(b). Binary: 1 0 0 1
Shifted binary: 1 0 0 (1)

1 1 0 1→gray

Gray to Binary Conversion:

If an n bit gray no. is rep by $G_n\ G_{n-1}$----------G_1

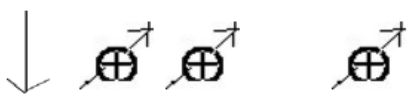
its binary equivalent by $B_n\ B_{n-1}$----------B_1 then the binary bits are obtained from gray bits as

$B_n= G_n$	$B_{n-1}=B_n \oplus G_{n-1}$	$B_{n-2}= \oplus G_{n-2}$	----------	$B1 =B_2 \oplus G_1$

To convert no. in any system into given no. first convert it into binary & then binary to gray. To convert gray no into binary no & convert binary no into require no system.

Ex:10110010(gray) = 11011100_2= DC_{16}=334_8=220_{10}
EX:1101

Gray: 1 1 0 1



Binary:1 0 0 1

Ex: $3A7_{16}$=0011,1010,0111_2=1001110100(gray)
527_8=101,011,011_2=111110110(gray)
652_{10}=1010001100_2= 1111001010(gray)

XS-3 gray code:

In a normal gray code , the bit patterns for 0(0000) & 9(1101) do not have a unit distance between them i.e, they differ in more than one position.In xs-3 gray code , each decimal digit is encoded with gray code patter of the decimal digit that is greater by 3. It has a unit distance between the patterns for 0 & 9.

XS-3 gray code for decimal digits 0 through 9

Decimal digit	Xs-3 gray code	Decimal digit	Xs-3 gray code
0	0010	5	1100
1	0110	6	1101
2	0111	7	1111
3	0101	8	1110
4	0100	9	1010

iii) Sequential Codes

A code is said to be sequential when two subsequent codes, seen as numbers in binary representation, differ by one. This greatly aids mathematical manipulation of data. The 8421 and

Excess-3 codes are sequential, whereas the 2421 and 5211 codes are not.

Binary coded decimal (bcd) and its arithmetic:

The BCD is a group of four binary bits that represent a decimal digit. In this representation each digit of a decimal number is replaced by a 4-bit binary number (i.e., a nibble). Since a decimal digit is a number from 0 to 9, a nibble representing a number greater than 9 is invalid BCD. For example (1010)2 is invalid BCD as it represents a number greater than 9.

Decimal Number	*Binary Representation*	*BCD Representation*
0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1
2	0 0 1 0	0 0 1 0
3	0 0 1 1	0 0 1 1
4	0 1 0 0	0 1 0 0
5	0 1 0 1	0 1 0 1
6	0 1 1 0	0 1 1 0
7	0 1 1 1	0 1 1 1
8	1 0 0 0	1 0 0 0
9	1 0 0 1	1 0 0 1
10	1 0 1 0	0 0 0 1 0 0 0 0
11	1 0 1 1	0 0 0 1 0 0 0 1
12	1 1 0 0	0 0 0 1 0 0 1 0
13	1 1 0 1	0 0 0 1 0 0 1 1
14	1 1 1 0	0 0 0 1 0 1 0 0
15	1 1 1 1	0 0 0 1 0 1 0 1

BCD Addition: In many application it is required to add two BCD numbers. But the adder circuits used are simple binary adders, which does not take care of peculiarity of BCD representation. Thus one must verify the result for valid BCD by using following rules:

1. If Nibble (i.e., group of 4-bits) is less than or equal to 9, it is a valid BCD number.

2. If Nibble is greater than 9, it is invalid. Add 6 (0110) to the nibble, to make it valid. Or If a carry was generated from the nibble during the addition, it is invalid. Add 6 (0110) to the nibble, to make it valid.

3. If a carry is generated when 6 is added, add this carry to next nibble.

EX: Add the BCD numbers i)1000 and 0101 and ii) 00011001 and 00011000

Solution.

```
  1 0 0 0  →  8
+ 0 1 0 1  → + 5
  1 1 0 1     13
```

Since, $(1101)_2 > (9)_{10}$ add 6 (0110) to it

So,

```
  1 1 0 1
  0 1 1 0
1 0 0 1 1
1   3
```

So, result = 00010011

(ii)

```
      1                ← Carry generated from nibble
0 0 0 1   1 0 0 1 →  19
0 0 0 1   1 0 0 0 → +18
0 0 1 1   0 0 0 1    37
```

Since, a carry is generated from right most nibble we must add 6 (0110) to it.

So,

```
0 0 1 1   0 0 0 1
          0 1 1 0
0 0 1 1   0 1 1 1 → (37)16
```

So, result = 00110111

BCD Subtraction:

The best way to cary out the BCD subtraction is to use complements. e. Although any of the two complements can be used, we prefer 10's complement for subtraction. Following are the steps to be followed for BCD subtraction using 10's complement:

1. Add the 10's complement of subtrahend to minuend.
2. Apply the rules of BCD addition to verify that result of addition is valid BCD.
3. Apply the rules of 10's complement on the result obtained in step 2, to declare the final result i.e., to declare the result of subtraction.

Ex: Subtract 61 from 68 using BCD.

Solution. To illustrate the process first we perform the subtraction using 10's comple ment in decimal system. After that we go for BCD subtraction.

we have, D = 68 – 61

So, 10's complement of 61 = 99 – 61 + 1 = 39

So, 10's complement of 61 = 99 – 61 + 1 = 39

So,

```
  6 8
+ 3 9
-----
1 0 7
↑
Carry
```

In 10's complement if an end carry is produced then it is discarded and result is declared positive. So,

D = +07

by using BCD

1.

```
                1          ← Carry generated
                             from nibble
BCD of 68 =   0 1 1 0 1 0 0 0
BCD of 39 = + 0 0 1 1 1 0 0 1
-------------------------------
              1 0 1 0 0 0 0 1
```

2. Check for valid BCD– since a carry is generated from right most nibble, we must add 6 (0110) to it. Since the left most nibble is greater than 9, we must add 6(0110) to it.

Thus,

```
  1 0 1 0   0 0 0 1
+ 0 1 1 0   0 1 1 0
```

(a) 8421 BCD code, sometimes referred to as the Natural Binary Coded Decimal Code (NBCD);

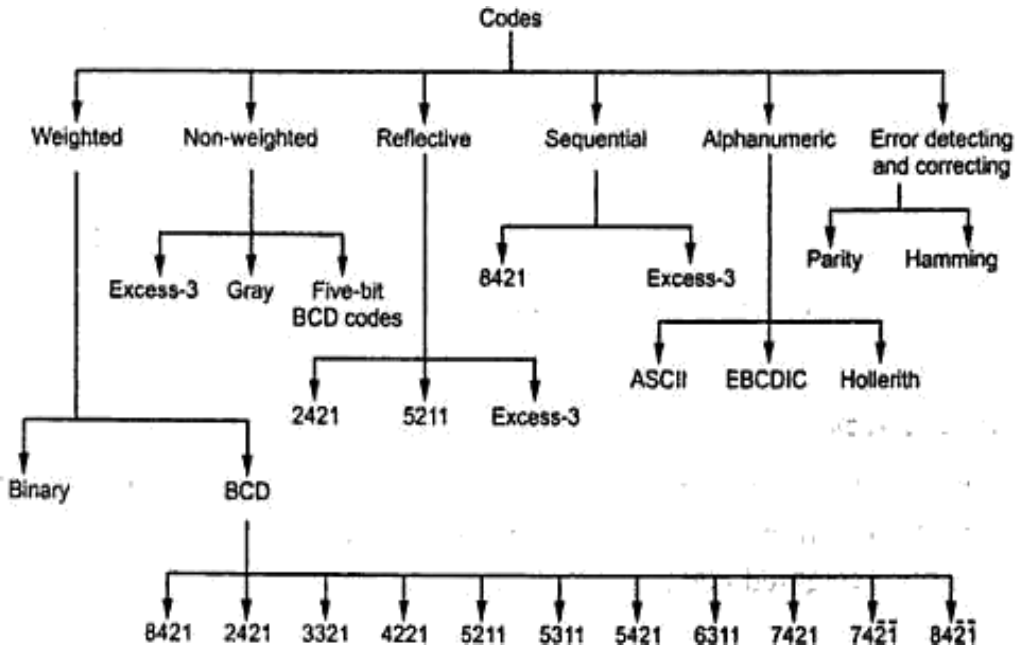
(b)* Excess-3 code (XS3); adding 3 to BCD gives the Excess -3 code.

(c)** 84 –2 –1 code (+8, +4, –2, –1);

(d) 2 4 2 1 code

Table BCD codes

Decimal Digit	*8421 (NBCD)*	*Excess-3 code (XS3)*	*84–2–1 code*	*2421 code*
0	0000	0011	0000	0000
1	0001	0100	0111	0001
2	0010	0101	0110	0010
3	0011	0110	0101	0011
4	0100	0111	0100	0100
5	0101	1000	1011	1011
6	0110	1001	1010	1100
7	0111	1010	1001	1101
8	1000	1011	1000	1110
9	1001	1100	1111	1111



Binary codes block diagram

Error – Detecting codes: When binary data is transmitted & processed,it is susceptible to noise that can alter or distort its contents. The 1‘s may get changed to 0‘s & 1‘s .because digital systems must be accurate to the digit, error can pose a problem. Several schemes have been devised to detect the occurrence of a single bit error in a binary word, so that whenever such an error occurs the concerned binary word can be corrected & retransmitted.

Parity: The simplest techniques for detecting errors is that of adding an extra bit known as parity bit to each word being transmitted.Two types of parity: Oddparity, evenparity forodd parity, the parity bit is set to a ‗0‘ or a ‗1‘ at the transmitter such that the total no. of 1 bit in the word including the parity bit is an odd no.For even parity, the parity bit is set to a ‗0‘ or a ‗1‘ at the transmitter such that the parity bit is an even no.

Decimal	8421 code		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	0100	1		0
6	0110	1		0
7	0111	0		1
8	1000	0		1
9	1001	1		0

When the digit data is received . a parity checking circuit generates an error signal if the total no of 1‘s is even in an odd parity system or odd in an even parity system. This parity check can always detect a single bit error but cannot detect 2 or more errors with in the same word.Odd parity is used more often than even parity does not detect the situation. Where all 0‘s are created by a short ckt or some other fault condition.

Ex: Even parity scheme
(a) 10101010 (b) 11110110 (c)10111001
Ans:

(a) No. of 1‘s in the word is even is 4 so there is no error
(b) No. of 1‘s in the word is even is 6 so there is no error
(c) No. of 1‘s in the word is odd is 5 so there is error

Ex: odd parity
(a)10110111 (b) 10011010 (c)11101010

Ans:

(a) No. of 1‘s in the word is even is 6 so word has error
(b) No. of 1‘s in the word is even is 4 so word has error
(c) No. of 1‘s in the word is odd is 5 so there is no error

Checksums:

Simple parity can‘t detect two errors within the same word. To overcome this, use a sort of 2 dimensional parity. As each word is transmitted, it is added to the sum of the previously transmitted words, and the sum retained at the transmitter end. At the end of transmission, the sum called the check sum. Up to that time sent to the receiver. The receiver can check its sum with the transmitted sum. If the two sums are the same, then no errors were detected at the receiver end. If there is an error, the receiving location can ask for retransmission of the entire data, used in teleprocessing systems.

Block parity:

Block of data shown is create the row & column parity bits for the data using odd parity. The parity bit 0 or 1 is added column wise & row wise such that the total no. of 1‘s in each column & row including the data bits & parity bit is odd as

Data	Parity bit
10110	0
10001	1
10101	0
00010	0
11000	1
00000	1
11010	0

data
10110
10001
10101
00010
11000
00000
11010

Error –Correcting Codes:

A code is said to be an error –correcting code, if the 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. The minimum distance of that code is the smallest no. of bits by which any two code words must differ. A code with minimum distance of 3 can't only correct single bit errors but also detect (can't correct) two bit errors, The key to error correction is that it must be possible to detect & locate erroneous that it must be possible to detect & locate erroneous digits. If the location of an error has been determined. Then by complementing the erroneous digit, the message can be corrected , error correcting , code is the Hamming code , In this , to each group of m information or message or data bits, K parity checking bits denoted by P1,P2,----------pk located at positions 2^{k-1} from left are added to form an (m+k) bit code word.

To correct the error, k parity checks are performed on selected digits of each code word, & the position of the error bit is located by forming an error word, & the error bit is then complemented. The k bit error word is generated by putting a 0 or a 1 in the 2^{k-1}th position depending upon whether the check for parity involving the parity bit P_k is satisfied or not.Error positions & their corresponding values :

Error Position	For 15 bit code $C_4 C_3 C_2 C_1$	For 12 bit code $C_4 C_3 C_2 C_1$	For 7 bit code $C_3 C_2 C_1$
0	0000	0000	0 0 0
1	0001	0001	0 0 1
2	0010	0010	0 1 0
3	0011	0011	0 1 1
4	0100	0100	1 0 0
5	0101	0101	1 0 1
6	0 11 0	0 11 0	1 1 0
7	0 11 1	0 11 1	1 1 1
8	1 00 0	1 00 0	
9	1 00 1	1 00 1	
10	1 01 0	1 01 0	
11	1 01 1	1 01 1	
12	1 10 0	1 10 0	
13	1 10 1		
14	1 11 0		
15	1 11 1		

7- bit Hamming code:

To transmit four data bits, 3 parity bits located at positions 2^0 21&2^2 from left are added to make a 7 bit codeword which is then transmitted.

The word format

P_1	P_2	D_3	P_4	D_5	D_6	D_7

D—Data bits P-
Parity bits

Decimal Digit	For BCD P1P2D3P4D5D6D7	For Excess-3 P1P2D3P4D5D6D7
0	0 0 0 0 00 0	1 0 0 0 0 1 1
1	1 1 0 1 00 1	1 0 0 1 1 0 0
2	0 1 0 1 01 1	0 1 0 0 1 0 1
3	1 0 0 0 01 1	1 1 0 0 1 1 0
4	1 0 0 1 10 0	0 0 0 1 1 1 1
5	0 1 0 0 10 1	1 1 1 0 0 0 0
6	1 1 0 0 11 0	0 0 1 1 0 0 1
7	0 0 0 1 11 1	1 0 1 1 0 1 0

8	**1 1 1 0 0 0 0**	**0 1 1 0 0 1 1**
9	**0 0 1 1 0 0 1**	**0 1 1 1 1 0 0**

Ex: Encode the data bits 1101 into the 7 bit even parity Hamming Code
The bit pattern is
P1P2D3P4D5D6D7

1 1 0 1

Bits 1,3,5,7 (P_1 111) must have even parity, so P_1 =1
Bits 2, 3, 6, 7(P_2 101) must have even parity, so P_2 =0
Bits 4,5,6,7 (P_4 101)must have even parity, so P_4 =0
The final code is 1010101

EX: Code word is 1001001

Bits 1,3,5,7 (C_1 1001) →no error →put a 0 in the 1‘s position→C1=0
Bits 2, 3, 6, 7(C_2 0001)) → error →put a 1 in the 2‘s position→C2=1
Bits 4,5,6,7 (C_4 1001)) →no error →put a 0 in the 4‘s position→C3=0

15-bit Hamming Code: It transmit 11 data bits, 4 parity bits located 2^0 2^1 2^2 2^3 Word format is

P_1	P_2	D_3	P_4	D_5	D_6	D_7	P_8	D_9	D10	D11	D12	D13	D14	D15

12-Bit Hamming Code:It transmit 8 data bits, 4 parity bits located at position 2^0 2^1 2^2 2^3 Word format is

P_1	P_2	D_3	P_4	D_5	D_6	D_7	P_8	D_9	D10	D11	D12

Alphanumeric Codes:

These codes are used to encode the characteristics of alphabet in addition to the decimal digits. It is used for transmitting data between computers & its I/O device such as printers, keyboards & video display terminals.Popular modern alphanumeric codes are ASCII code & EBCDIC code.