Binary to BCD Conversion and BCD to Binary Conversion

Binary to BCD :

① Binary → Decimal.
② Decimal → BCD.

$$(\overset{}{\underset{2^5}{1}}\ \underset{2^4}{1}\ \underset{2^3}{0}\ \underset{2^2}{1}\ \underset{2^1}{0}\ \underset{2^0}{1})_2$$

$$= (1\times 2^5 + 1\times 2^4 + 1\times 2^2 + 1\times 2^0)$$

$$= 32 + 16 + 4 + 1$$

$$= (53)_{10} \longrightarrow \text{Decimal.}$$

$$0101\ 0011 \longrightarrow \text{BCD.}$$

BCD to Binary.

① BCD → Decimal.

② Decimal → Binary.

$$(\underbrace{1001}_{\downarrow\ 9}\ \underbrace{0101}_{\downarrow\ 5})\ \text{BCD}.$$

$$(95)_{10} \longrightarrow \text{Decimal}$$

$$\begin{array}{r|l|c}
2 & 95 & \\
\hline
2 & 47 & 1 \\
\hline
2 & 23 & 1 \\
\hline
2 & 11 & 1 \\
\hline
2 & 5 & 1 \\
\hline
2 & 2 & 1 \\
\hline
2 & 1 & 0 \\
\hline
 & 0 & 1
\end{array} \uparrow$$

$$(1011111)_2.$$

Excess-3 code

- The Excess-3 code is also called as XS-3 code.
- It is non-weighted code used to express decimal numbers.
- The Excess-3 code words are derived from the 8421 BCD code words by adding $(0011)_2$ or $(3)_{10}$ to each code word in 8421.

Decimal → Excess-3.

① Decimal - BCD.
② BCD + 3

① $(5)_{10}$

$$\begin{array}{rl} & 0101 \rightarrow \text{BCD} \\ + & 0011 \rightarrow +3 \\ \hline & 1000 \rightarrow \text{Excess-3} \end{array}$$

② $(246)_{10}$

$$\begin{array}{rl} & 0010\ 0100\ 0110 \rightarrow \text{BCD} \\ + & 0011\ 0011\ 0011 \\ \hline & 0101\ 0111\ 1001 \rightarrow \text{Excess-3} \end{array}$$

Binary to excess-3.

$$\begin{tikzcd}
\text{Binary} \arrow[d] \\
\text{decimal.} \arrow[d] \\
\text{BCD} \\
+3 \arrow[d] \\
\text{excess-3.}
\end{tikzcd}$$

Decimal fractional Value → Binary.

48.25

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

$$\begin{array}{r} 0.25 \\ \times \quad 2 \\ \hline \boxed{0}.50 \end{array} \quad \begin{array}{r} 0.50 \\ \times \quad 2 \\ \hline \boxed{1}.00 \end{array}$$

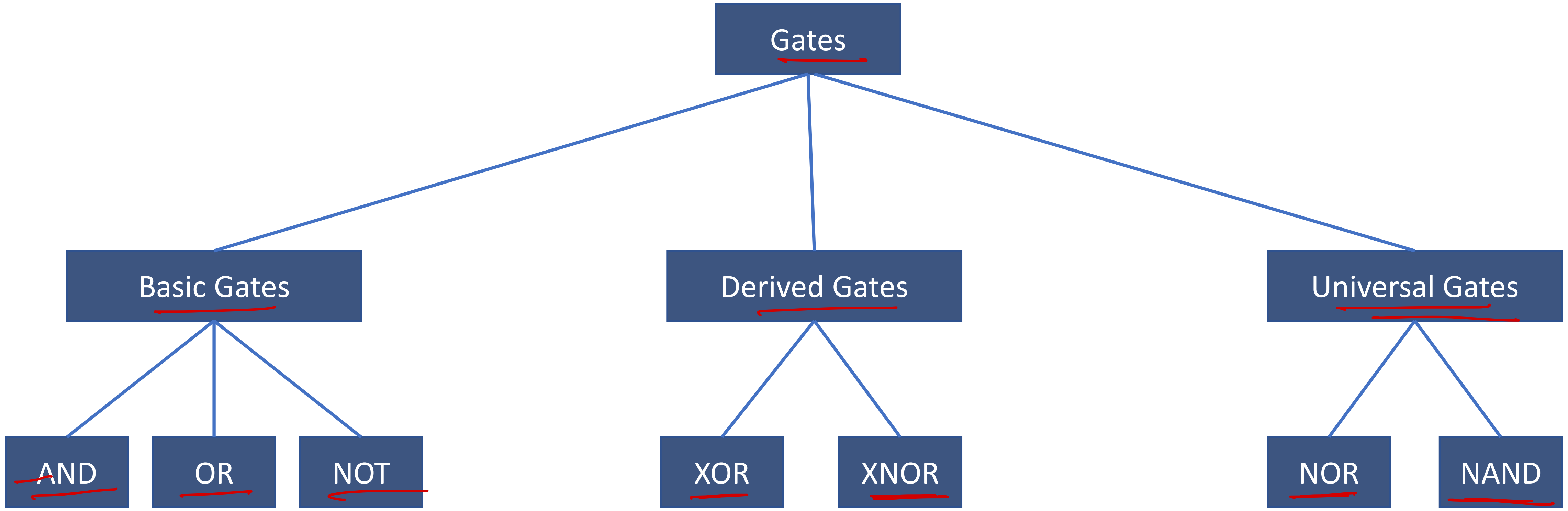
$$\begin{array}{r} 0.00 \\ \times \quad 2 \\ \hline 0.00 \end{array}$$

$(110000.010)_2$

Logic Gates

- The basic digital electronic circuit that has one or more inputs and single output is known as **Logic gate**.
- Logic gates are the building blocks of any digital system. We can classify these Logic gates into the following three categories.
  - ✓ Basic gates
  - ✓ Universal gates
  - ✓ Derived gates

Types of GATES



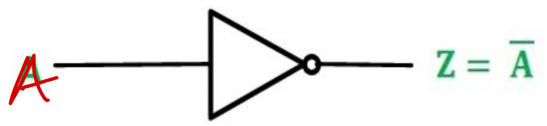
Truth Table

- A truth table shows how a logic circuit's output responds to various combinations of the inputs, using logic 1 for true and logic 0 for false.
- Formula for truth table: $\boxed{2^n = m}$ → i/p . ↳ comb .

$2^2 = 4$.

- Where,
  - n → number of inputs
  - m → combination of inputs

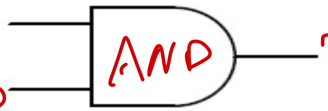
NOT gate 1's comp .



A	Y
0	1
1	0

Logic Gates

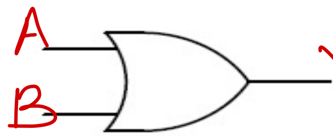
i/p $\begin{cases} A \\ B \end{cases}$



$Y = A \cdot B$ = A AND B.

AND

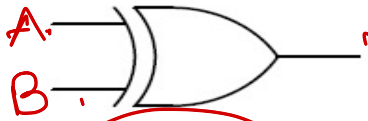
A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1



$Y = A + B$ = A OR B.

OR

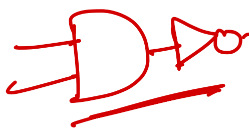
A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

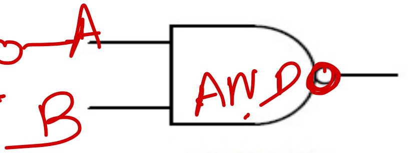


$Y = A \oplus B$.
$= \bar{A}B + A\bar{B}$

XOR

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

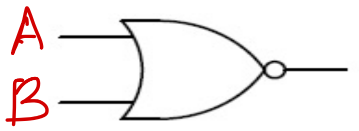




$Y = \overline{A \cdot B}$
$= \bar{A} + \bar{B}$
$= A \uparrow B$

NAND

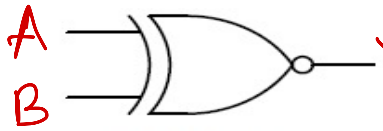
A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0



$Y = \overline{A + B}$
$= \bar{A} \cdot \bar{B}$
$= A \downarrow B$.

NOR

A	B	Output
0	0	1
0	1	0
1	0	0
1	1	0



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

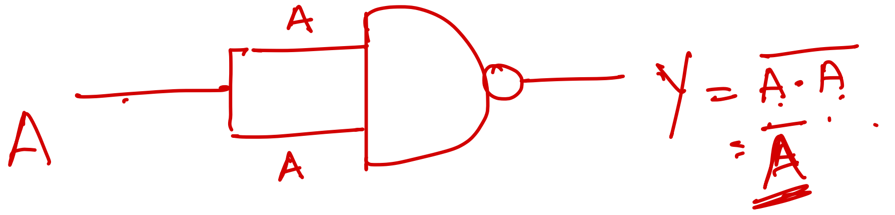
XNOR

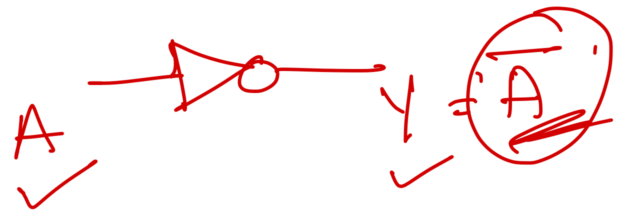
A	B	Output
0	0	1
0	1	0
1	0	0
1	1	1

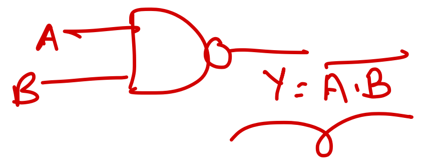
Universal Gates

- A **universal gate** is a logic gate which can implement any Boolean function without the need to use any other type of logic gate
- The NAND gate and NOR gate are universal gates.
- Any logic circuit can be built using NAND gates or NOR gates.

using NAND $\longrightarrow$ design NOT



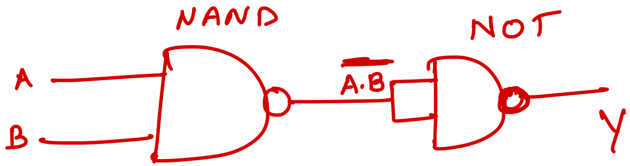




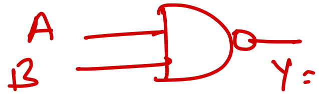
$A \cdot A = A$ $\quad 0 \cdot 0 = 0$

$1 \cdot 1 = 1$

Using NAND → design AND .



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



AND .

$Y = A \cdot B$

Gray code

- It is the non-weighted code and it is not arithmetic codes.
- There are no specific weights assigned to the bit position.
- It has a very special feature that, only one bit will change each time the decimal number is incremented(only one bit changes at a time)
- Gray code is popularly used in the shaft position encoders.

Decimal Number	Gray Code
0	0000
1	0001
2	0011
3	0010
4	0110
5	0111
6	0101

Gray code.

$$\begin{tikzcd} 0 \arrow[r] & 00 \\ 1 \arrow[r] & 01 \\ 2 \arrow[r] & 11 \\ 3 \arrow[r] & 10 \end{tikzcd}$$

3-bit

$0 \rightarrow 0\ 0\ 0$

$1 \rightarrow 0\ 0\ 1$

$2 \rightarrow 0\ 1\ 1$

$3 \rightarrow 0\ 1\ 0$

$4 \rightarrow 1\ 1\ 0$

4-bit
Gray code

$0 \rightarrow 0\ 0\ 0\ 0$

$1 \rightarrow 0\ 0\ 0\ 1$

$2 \rightarrow 0\ 0\ 1\ 1$

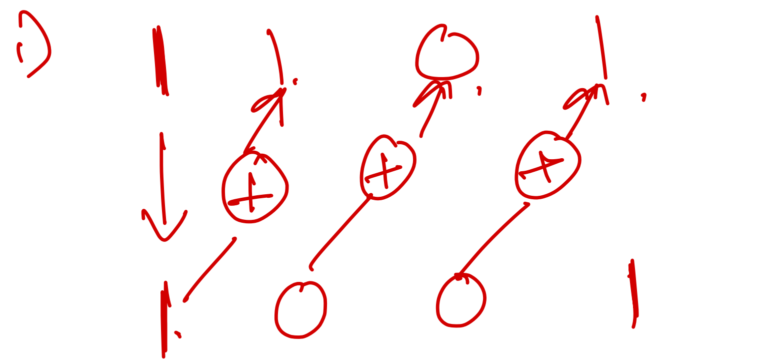
$3 \rightarrow 0\ 0\ 1\ 0$

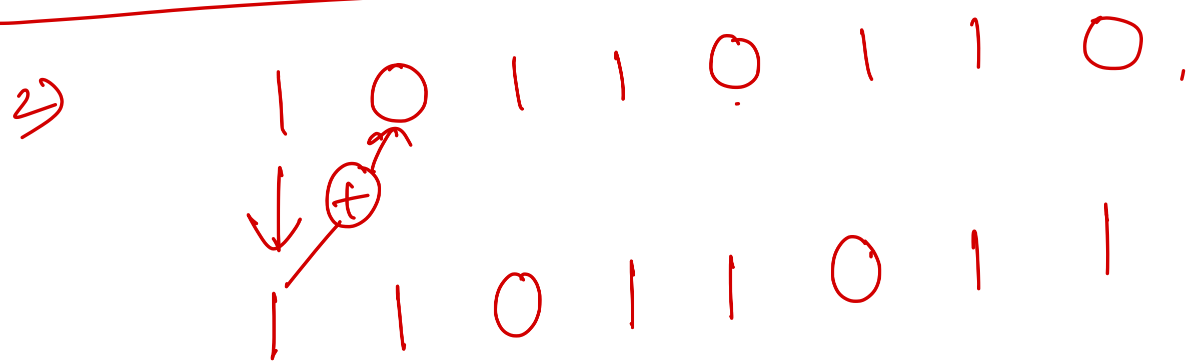
$4 \rightarrow 0\ 1\ 1\ 0$

GRAY TO BINARY CONVERSION and BINARY TO GRAY CONVERSION

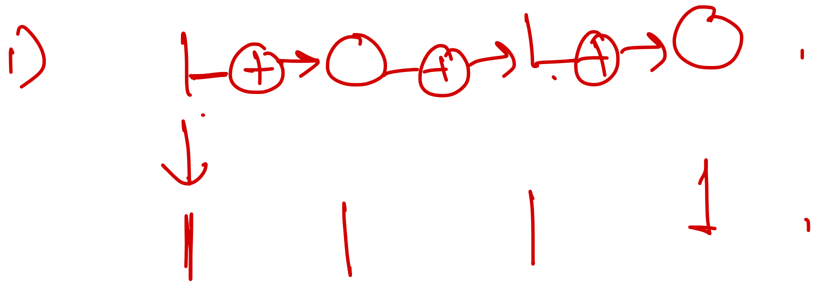
Gray to Binary

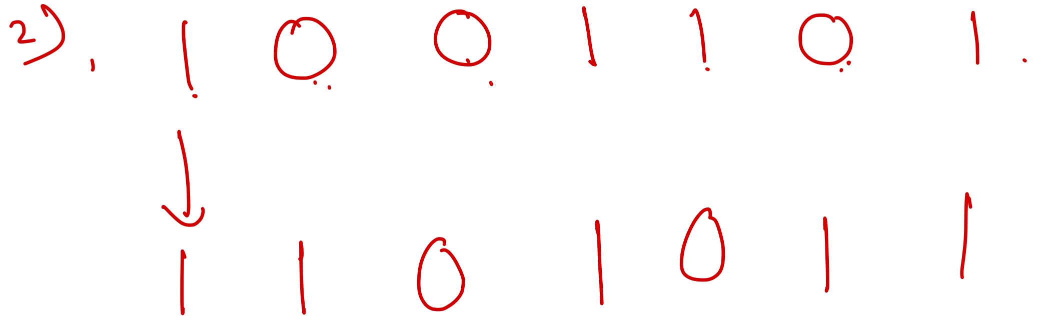
XOR		
0	0	0
0	1	1
1	0	1
1	1	0

1)


2)


Binary to Gray

1)


2)


Unsigned Number and Signed Number

$\cancel{\pm}5$ $5 \rightarrow \underbrace{0101}$

Unsigned Number

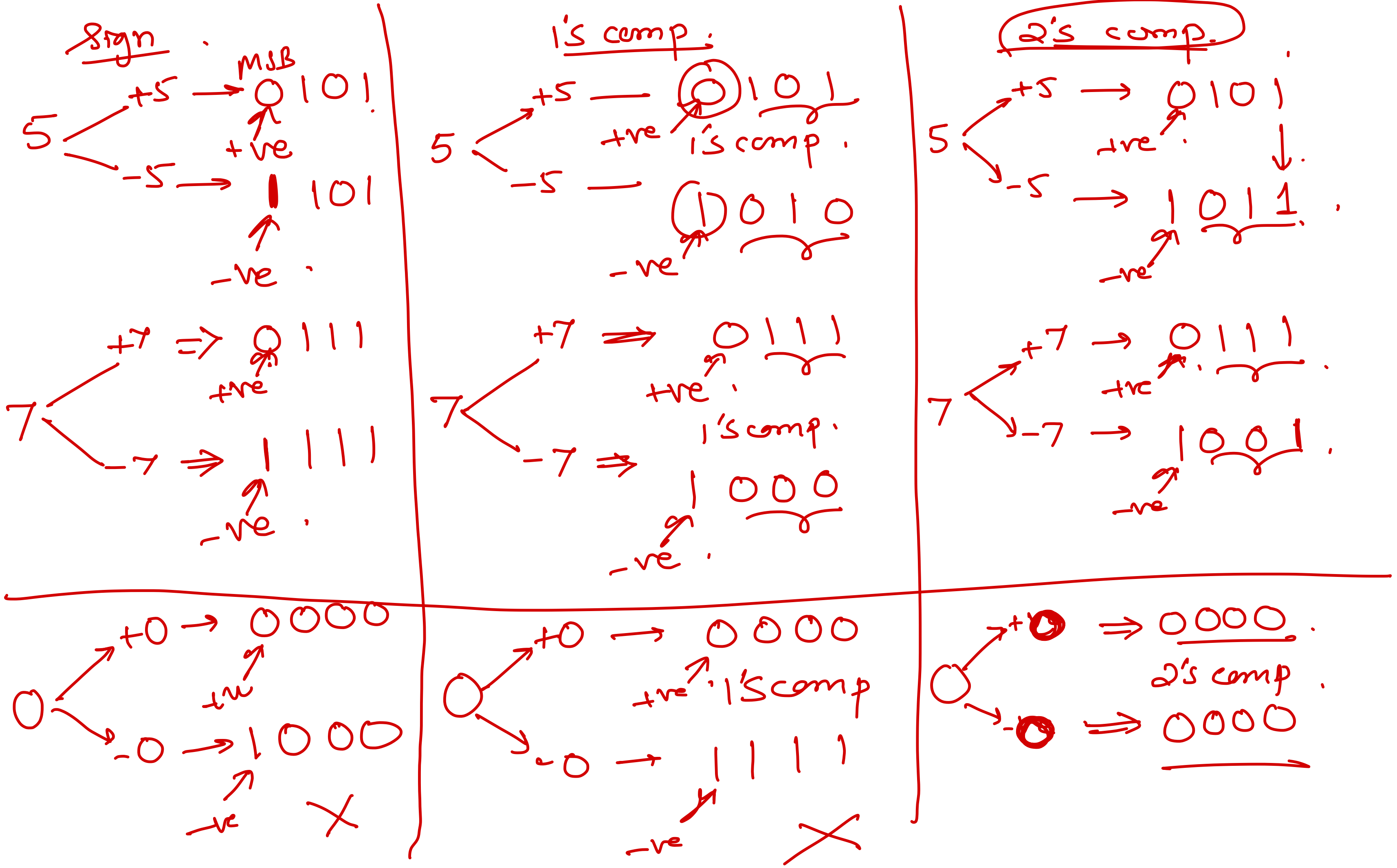
- Unsigned numbers don't have any sign, these can contain only magnitude of the number. So, representation of unsigned binary numbers are all positive numbers only. The range of unsigned binary number is from 0 to (2^n-1).

$2^4-1 = 15$

Signed Numbers

- Signed numbers contain sign flag, this representation distinguish positive and negative numbers. This technique contains both sign bit and magnitude of a number. For example, in representation of negative decimal numbers, we need to put negative symbol in front of given decimal number.
- There are three types of representations for signed binary numbers
  - Sign-Magnitude form
  - 1's complement form
  - 2's complement form

$$\begin{tikzcd} & \text{MSB} & & & \text{LSB} \\ 5 \arrow[r] & 0 & 1 & 0 & 1 \\ & \text{sign} \arrow[u] & & \text{magnitude} \end{tikzcd}$$


Sign.
MSB
5
+5 → 0101
+ve
-5 → 1101
-ve.
7
+7 ⇒ 0111
+ve
-7 ⇒ 1111
-ve.
0
+0 → 0000
+ve
-0 → 1000
-ve
1's comp:
5
+5 — 0101
+ve
1's comp.
-5 — 1010
-ve
7
+7 ⇒ 0111
+ve.
1's comp.
-7 ⇒ 1000
-ve.
0
+0 → 0000
+ve
1's comp
-0 → 1111
-ve
2's comp.
5
+5 → 0101
+ve.
-5 → 1011.
-ve
7
+7 → 0111.
+ve
-7 → 1001.
-ve
0
+0 ⇒ 0000.
2's comp.
-0 ⇒ 0000

Postulate of Boolean Algebra

- Following are the some laws of Boolean:
- Commutative Law:
  - A.B = B.A
  - A+B = B+A
- Associative Law:
  - (A.B).C = A.(B.C)
  - (A+B)+C = A+(B+C)
- Distributive Law:
  - 1 A. (B+C) = AB + AC
- Identity Law
  - A + A = A
- Involution Law
  - $\bar{\bar{A}}$ = A
- AND Law:
  - A.0 = 0
  - A.1 = A
  - A.A = A
  - A. Ā = 0

$A \cdot B = B \cdot A$

2 $A + (BC) = (A+B) \cdot (A+C)$

$\bar{\bar{A}}$

$\bar{\bar{0}} = 0$

Postulate of Boolean Algebra

- OR Laws:
  - A + 0 = A
  - A + 1 = 1
  - A + A = A
  - A + $\bar{A}$ = 1 $\Rightarrow 0 + 1 = 1 \quad 1 + 0 = 1$
- De-Morgan's Theorem:
  - First theorem states that complement of sum is equal to the product of their individual complements

    $$\overline{A + B} = \bar{A} * \bar{B}$$
  - Second theorem states that complement of products is equal to the sum of their complements.

    $$\overline{A * B} = \bar{A} + \bar{B}$$
- Absorption
  - A + A.B = A
  - A .(A + B) = A

$A + A \cdot B$

$A(1 + B) = A(1) = A$

B	A	A · B	A + A · B
0	0	0	0
0	1	0	1
1	0	0	0
1	1	1	1

Boolean Function

- We can minimize number of literals in Boolean functions by using two methods
  - Boolean Algebraic Method
  - K-map Method

Boolean Algebraic

- Boolean Algebra is a form of mathematical algebra that is used in digital logic in digital electronics.
- The main aim of any logic design is to simplify the logic as much as possible so that the final implementation will become easy.
- Solve using Algebraic method
  - $x + x \cdot y = x$
  - $x \cdot (x + y) = x$
  - $x \cdot y + x \cdot z + y \cdot z = x \cdot y + x \cdot z$
  - Complement of a function x’yz’ + x’y’z

$$x + x \cdot y \Rightarrow x$$
$$x(1+y)$$
$$x(1)$$
$$x$$

- Complement of a function x’yz’ + x’y’z

$\boxed{x' = \overline{x}}$

$$\overline{\underbrace{\overline{x}y\overline{z}}_{A} + \underbrace{\overline{x}\,\overline{y}z}_{B}}.$$

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

Demorgan's.

$$= \left(\overline{\overline{x}y\overline{z}}\right) \cdot \left(\overline{\overline{x}\,\overline{y}z}\right).$$

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

$$= \left(\overline{\overline{x}} + \overline{y} + \overline{\overline{z}}\right) \cdot \left(\overline{\overline{x}} + \overline{\overline{y}} + \overline{z}\right).$$

$\overline{\overline{x}} = x$.

$$= \underbrace{(x + \overline{y} + z) \cdot (x + y + \overline{z})}$$

Algebraic method

$x + x \cdot y = x$

Proof: $x + x \cdot y$

$= x \cdot 1 + x \cdot y$

$= x \cdot (1 + y)$

$= x \cdot 1 = x \; x \cdot (x + y)$

$= x$

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

Proof: $x \cdot (x + y)$ demorgans

$= (x + 0) \cdot (x + y)$

$= x + (0 \cdot y)$

$= x + 0$

$= x$

Algebraic method

- $x \cdot y + x \cdot z + y \cdot z = x \cdot y + x \cdot z$
- Proof: $x \cdot y + x \cdot z + y \cdot z$

$= x \cdot y + x \cdot z + 1 \cdot y \cdot z$

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

$= x \cdot y + x \cdot z + x \cdot y \cdot z + x \cdot y \cdot z$

$= x \cdot y + x \cdot y \cdot z + x \cdot z + x \cdot y \cdot z$

$= x \cdot y \cdot 1 + x \cdot y \cdot z + x \cdot 1 \cdot z + x \cdot y \cdot z$

$= x \cdot y \cdot (1 + z) + x \cdot z \cdot (1 + y)$

$= x \cdot y \cdot 1 + x \cdot z \cdot 1$

$= x \cdot y + x \cdot z$

Standard Form

- There are two standard forms
  - SOP
  - POS

Sum of Product(SOP)

- The sum of product form is represented by using basic logic gates: AND gate and OR gate. The SOP form implementation will have AND gates at its input side and as the output of the function is the sum of all product terms, it has an OR gate at its output side.
- It is formed by adding (OR operation) the product terms.
- These product terms are also called as ‘min-terms’.
- **Examples**
  - AB + BA

$(A.B)$

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

$(A.\bar{B})$

$(\bar{A}B)$

OR

$$\underbrace{\boxed{AB} + \boxed{\bar{A}\bar{B}} + \boxed{A\bar{B}} + \boxed{\bar{A}B}}$$

Standard Form

Product of Sum(POS)

- The product of sums form can be represented by using basic logic gates like AND gate and OR gates. The POS form implementation will have the OR gate at its input side and as the output of the function is product of all sum terms, it has AND gate at its output side.
- It is formed by multiplying(AND operation) the sum terms.
- These sum terms are also called as ‘max-terms’.
- **Examples**
  - (A+B) * (A + B + C) * (C +D)

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

K-Map

- Karnaugh introduced a simplification of Boolean functions in an easy way.
- This method is known as Karnaugh map method or K-map method.
- It is a pictorial representation of graphical method, which consists of 2^n cells for 'n' variables. The adjacent cells are differed only in single bit position.
- K-map uses Gray code
- 2- bits

A \ B	0	1
0	0	1
1	2	3

A	B
0	0
0	1
1	0
1	1

- 3-bits

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

← Gray code.

4-bit

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

K-Map

Karnaugh Map Simplification Rules-

- To minimize the given Boolean function,
- We draw a K-Map according to the number of variables it contains.
- We fill the K-Map with 0's and 1's according to its function.
- Then, we minimize the function in accordance with the following rules.

Rule-1:

- We can either group 0's with 0's or 1's with 1's but we can not group 0's and 1's together.
- X representing don't care can be grouped with 0's as well as 1's.

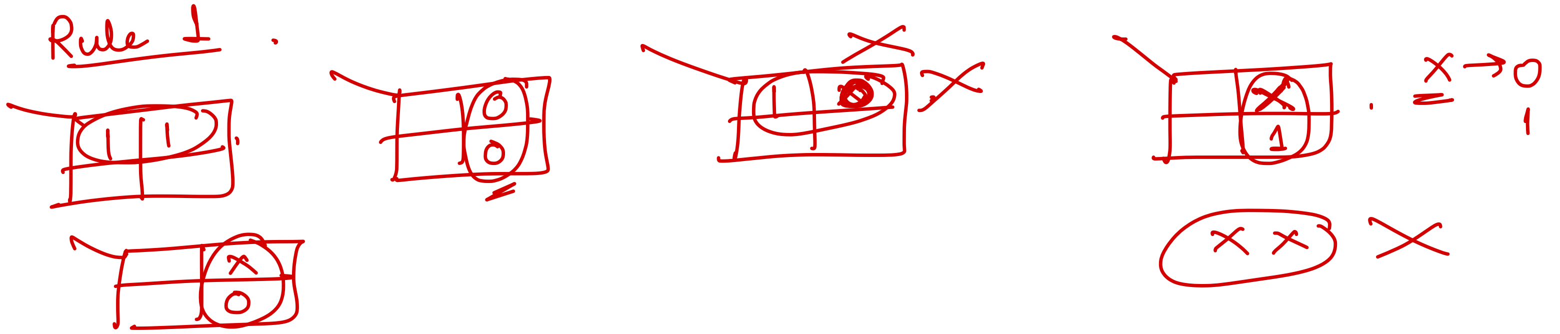
Rule-02:

- Groups may overlap each other.

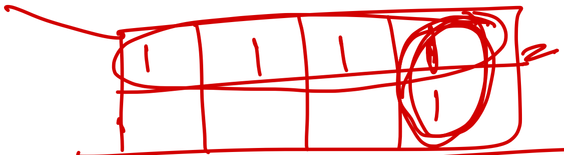
Rule-03:

- We can only create a group whose number of cells can be represented in the power of 2.
- In other words, a group can only contain 2^n i.e. 1, 2, 4, 8, 16 and so on number of cells.

Rule 1



Rule 2



Rules 3

1, 2, 4, 8, 16.

K-Map

Rule-4:

- Groups can be only either horizontal or vertical.
- We can not create groups of diagonal or any other shape.

Rule-5:

- Each group should be as large as possible.

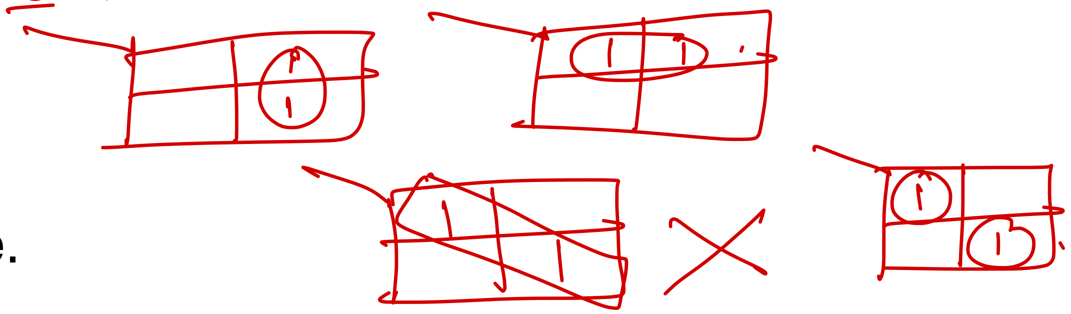
Rule-6:

- Opposite grouping and corner grouping are allowed.

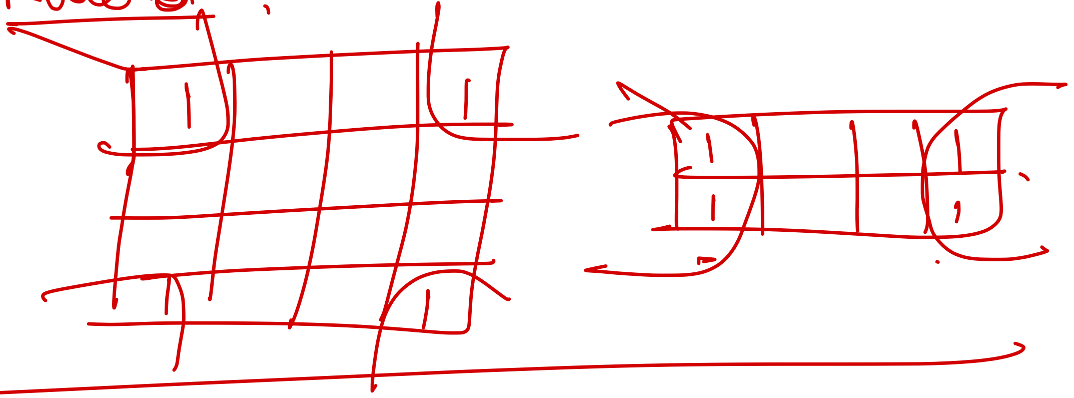
Rule-7:

- There should be as few groups as possible.

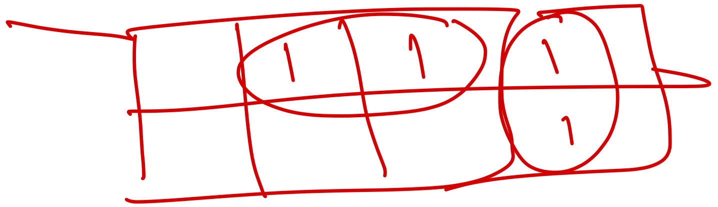
Rule 4



Rule 6.



Rule 7



K-Map

- Simplify the Boolean function F= x'y'z + x'yz +xy'
- Simplify the Boolean function F(A,B,C) =$\Sigma(1,5,6,7)$
- Simplify the Boolean function F(A,B,C) =$\Sigma(0,1,3,4,5)$
- Simplify the Boolean function F(A,B,C,D) =$\Sigma(0,2,4,6,8,9,10)$
- Don't care condition
  - F(A,B,C) = $\Sigma(0,4,5)$ are minterms (2,3,6) are don't care

K-Map.

- Simplify the Boolean function F= x'y'z + x'yz +xy'

SOP.

$\bar{x} = 1$

$\bar{x} = 0$

$$F(xyz) = \bar{x}\bar{y}z + \bar{x}yz + x\bar{y}$$

$$= \bar{x}\bar{y}z + \bar{x}yz + x\bar{y}(z+\bar{z})$$

$x = 1$

$\bar{x} = 0$

$$= \underset{001}{\bar{x}\bar{y}z} + \underset{011}{\bar{x}yz} + \underset{101}{x\bar{y}z} + \underset{100}{x\bar{y}\bar{z}}$$

$$= m_1 + m_3 + m_5 + m_4$$

$$= \Sigma(1, 3, 4, 5)$$

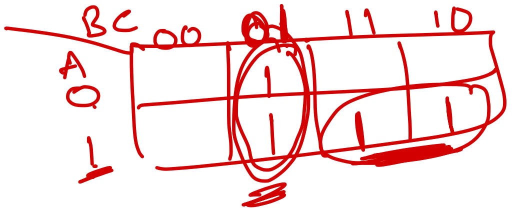
x \ yz	00	01	11	10
0		1	1	
1	1	1		

$= x\bar{y} + \bar{x}z.$

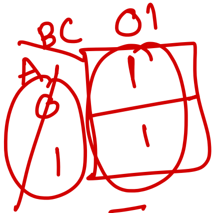
x \ yz	01	11
0	1	1

$= \bar{x}z.$

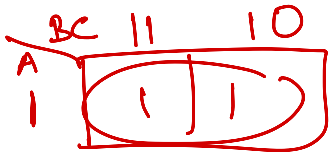
- Simplify the Boolean function F(A,B,C) =Σ(1,5,6,7)



$$= \overline{B}C + AB$$



$$\overline{B}C$$



$$= AB$$