Unit -3
Number System

The technique to represent and work with numbers is called **Number System**.

Types of Number System:

There are various types of number system. The four most common number system types are:

1. Decimal number system (Base- 10)
2. Binary number system (Base- 2)
3. Octal number system (Base-8)
4. Hexadecimal number system (Base- 16)
5.

Decimal Number System-

Decimal number system is a **base 10** number system having 10 digits from 0 to 9. This means that any numerical quantity can be represented using these 10 digits. Decimal number system is also a **positional value system**. This means that the value of digits will depend on its position. Let us take an example to understand this.

Say we have three numbers – 734, 971 and 207. The value of 7 in all three numbers is different–

In 734, value of 7 is 7 hundreds or 700 or 7×100 or 7×10^2

In 971, value of 7 is 7 tens or 70 or 7×10 or 7×10^1

In 207, value 0f 7 is 7 units or 7 or 7×1 or 7×10^0

The weightage of each position can be represented as follows –

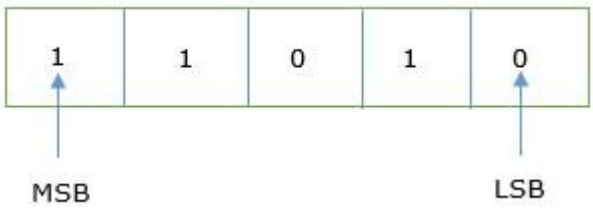
10^5	10^4	10^3	10^2	10^1	10^0

Binary Number System-

The number system having just these two digits – 0 and 1 – is called **Binary Number System**. Each binary digit is also called a **bit**. Binary number system is also positional value system, where each digit has a value expressed in powers of 2, as displayed here.

2^5	2^4	2^3	2^2	2^1	2^0

In any binary number, the rightmost digit is called **least significant bit (LSB)** and leftmost digit is called **most significant bit (MSB)**.



And decimal equivalent of this number is sum of product of each digit with its positional value.

$11010_2 = 1\times2^4 + 1\times2^3 + 0\times2^2 + 1\times2^1 + 0\times2^0$

$= 16 + 8 + 0 + 2 + 0$

$= 26_{10}$

Octal Number System-

Octal number system has eight digits – 0, 1, 2, 3, 4, 5, 6 and 7. Octal number system is also a positional value system with where each digit has its value expressed in powers of 8, as shown here –

8^5	8^4	8^3	8^2	8^1	8^0

Decimal equivalent of any octal number is sum of product of each digit with its positional value.

$726_8 = 7\times8^2 + 2\times8^1 + 6\times8^0$

$= 448 + 16 + 6$

$= 470_{10}$

Hexadecimal Number System-

Hexadecimal number system has 16 symbols – 0 to 9 and A to F where A is equal to 10, B is equal to 11 and so on till F. Hexadecimal number system is also a positional value system with where each digit has its value expressed in powers of 16, as shown here −

16^5	16^4	16^3	16^2	16^1	16^0

Decimal equivalent of any hexadecimal number is sum of product of each digit with its positional value.

$27FB_{16} = 2\times16^3 + 7\times16^2 + 15\times16^1 + 10\times16^0$

$= 8192 + 1792 + 240 + 10$

$= 10234_{10}$

Number System Conversion

Number system conversions deal with the operations to change the base of the numbers.
Now let us learn, conversion from one base to another.

Decimal to Other Bases-

Converting decimal number to other base numbers is easy. We have to divide the decimal number by the converted value of the new base.

Decimal to Binary Number:

Suppose if we have to convert a decimal to binary, then divide the decimal number by 2.

Example 1. Convert $(25)_{10}$ to binary number.

Solution: Let us create a table based on this question.

Operation	Output	Remainder
25 ÷ 2	12	1(MSB)
12 ÷ 2`	6	0
6 ÷ 2	3	0
3 ÷ 2	1	1
1 ÷ 2	0	1(LSB)

Therefore, from the above table, we can write,

$(25)_{10} = (11001)_2$

Decimal to Octal Number:

To convert decimal to octal number we have to divide the given original number by 8 such that base 2 changes base 8. Let us understand with the help of an example.

Example 2: Convert 128_{10} to octal number.
Solution: Let us represent the conversion in tabular form.

Operation	Output	Remainder
128÷8	16	0(MSB)
16÷8	2	0
2÷8	0	2(LSB)

Therefore, the equivalent octal number is 200_8

Decimal to Hexadecimal:

Again in decimal to hex conversion, we have to divide the given decimal number by 16.
Example 3: Convert 128_{10} to hex.
Solution: As per the method, we can create a table;

Operation	Output	Remainder
128÷16	8	0(MSB)
8÷16	0	8(LSB)

Therefore, the equivalent hexadecimal number is 80_{16}

Other Base System to Decimal Conversion-

Binary to Decimal:

In this conversion, binary number to a decimal number, we use multiplication method, in such a way that, if a number with base n has to be converted into a number with base 10, then each digit of the given number is multiplied from MSB to LSB with reducing the power of the base.
Let us understand this conversion with the help of an example.
Example 1. Convert $(1101)_2$ into decimal number.
Solution: Given a binary number $(1101)_2$.
Now, multiplying each digit from MSB to LSB with reducing the power of the base number 2.
$1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$
$= 8 + 4 + 0 + 1$
$= 13$
Therefore, $(1101)_2 = (13)_{10}$

Octal to Decimal:

To convert octal to decimal, we multiply the digits of octal number with decreasing power of the base number 8, starting from MSB to LSB and then add them all together.
Example 2: Convert 22_8 to decimal number.
Solution: Given, 22_8
$2 \times 8^1 + 2 \times 8^0$
$= 16 + 2$
$= 18$
Therefore, $22_8 = 18_{10}$

Hexadecimal to Decimal:

To convert hexadecimalto decimal, we multiply the digits of hexadeciaml number with decreasing power of the base number 16, starting from MSB to LSB and then add them all together.
Example 3: Convert 121_{16} to decimal number.

Solution: $1 \times 16^2 + 2 \times 16^1 + 1 \times 16^0$
$= 16 \times 16 + 2 \times 16 + 1 \times 1$
$= 289$
Therefore, $121_{16} = 289_{10}$

Hexadecimal to Binary:

To convert hexadecimal numbers to binary and vice versa is easy, you just have to memorize the table given below.

Hexadecimal Number	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

You can easily solve the problems based on hexadecimal and binary conversions with the help of this table. Let us take an example.
Example: Convert $(89)_{16}$ into a binary number.
Solution: From the table, we can get the binary value of 8 and 9, hexadecimal base numbers.
8 = 1000 and 9 = 1001
Therefore, $(89)_{16} = (10001001)_2$

Octal to Binary :

To convert octal to binary number we can simply use the table. Just like having a table for hexadecimal and its equivalent binary, in the same way, we have a table for octal and its equivalent binary number.

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

Example: Convert $(214)_8$ into a binary number.
Solution: From the table, we know,
2 → 010
1 → 001
4 → 100
Therefore,$(214)_8 = (010001100)_2$

Unit-3

Representation of Numbers:

We can make the binary numbers into the following two groups − **Unsigned numbers** and **Signed numbers**.

Unsigned Numbers:

Unsigned numbers contain only magnitude of the number. They don't have any sign. That means all unsigned binary numbers are positive.

As in decimal number system, the placing of positive sign in front of the number is optional for representing positive numbers. Therefore, all positive numbers including zero can be treated as unsigned numbers if positive sign is not assigned in front of the number.

Signed Numbers:

Signed numbers contain both sign and magnitude of the number. Generally, the sign is placed in front of number. So, we have to consider the positive sign for positive numbers and negative sign for negative numbers. Therefore, all numbers can be treated as signed numbers if the corresponding sign is assigned in front of the number.

If sign bit is zero, which indicates the binary number is positive. Similarly, if sign bit is one, which indicates the binary number is negative.

Representation of Un-Signed Binary Numbers-

The bits present in the un-signed binary number holds the **magnitude** of a number. That means, if the un-signed binary number contains **'N'** bits, then all **N** bits represent the magnitude of the number, since it doesn't have any sign bit.

Example

Consider the decimal number 108. The binary equivalent of this number is 1101100. This is the representation of unsigned binary number.

$108_{10} = 1101100_2$

It is having 7 bits. These 7 bits represent the magnitude of the number 108.

Representation of Signed Binary Numbers

The Most Significant Bit *MSB* of signed binary numbers is used to indicate the sign of the numbers. Hence, it is also called as **sign bit**.

The positive sign is represented by placing '0' in the sign bit.

Similarly, the negative sign is represented by placing '1' in the sign bit.

If the signed binary number contains 'N' bits, then $N-1$ bits only represent the magnitude of the number since one bit *MSB* is reserved for representing sign of the number.

There are three **types of representations** for signed binary numbers

- **Sign-Magnitude form**
- **1's complement form**
- **2's complement form**

Representation of a positive number in all these 3 forms is same. But, only the representation of negative number will differ in each form.

Example

Consider the **positive decimal number +108**. The binary equivalent of magnitude of this number is 1101100. These 7 bits represent the magnitude of the number 108. Since it is positive number, consider the sign bit as zero, which is placed on left most side of magnitude.

$$+108_{10} = 01101100_2$$

Therefore, the **signed binary representation** of positive decimal number +108 is $\mathbf{01101100}$. So, the same representation is valid in sign-magnitude form, 1's complement form and 2's complement form for positive decimal number +108.

Sign-Magnitude form:

In sign-magnitude form, the MSB is used for representing **sign** of the number and the remaining bits represent the **magnitude** of the number. So, just include sign bit at the left most side of unsigned binary number. This representation is similar to the signed decimal numbers representation.

Example

Consider the **negative decimal number -108**. The magnitude of this number is 108. We know the unsigned binary representation of 108 is 1101100. It is having 7 bits. All these bits represent the magnitude.

Since the given number is negative, consider the sign bit as one, which is placed on left most side of magnitude.

$$-108_{10} = 11101100_2$$

Therefore, the sign-magnitude representation of -108 is **11101100**.

1's complement form:

The 1's complement of a number is obtained by **complementing all the bits** of signed binary number. So, 1's complement of positive number gives a negative number. Similarly, 1's complement of negative number gives a positive number.

That means, if you perform two times 1's complement of a binary number including sign bit, then you will get the original signed binary number.

Example

Consider the **negative decimal number -108**. The magnitude of this number is 108. We know the signed binary representation of 108 is 01101100.

It is having 8 bits. The MSB of this number is zero, which indicates positive number. Complement of zero is one and vice-versa. So, replace zeros by ones and ones by zeros in order to get the negative number.

$$-108_{10} = 10010011_2$$

Therefore, the **1's complement of 108_{10}** is **10010011_2**.

2's complement form:

The 2's complement of a binary number is obtained by **adding one to the 1's complement** of signed binary number. So, 2's complement of positive number gives a negative number. Similarly, 2's complement of negative number gives a positive number.

That means, if you perform two times 2's complement of a binary number including sign bit, then you will get the original signed binary number.

Example

Consider the **negative decimal number -108**.

We know the 1's complement of $(\mathbf{108})_{10}$ is $(\mathbf{10010011})_2$

2's compliment of 108_{10} = 1's compliment of 108_{10} + 1.

= 10010011 + 1

= 10010100

Therefore, **the 2's complement of 108_{10}is10010100_2.**

Complements

Complements are used in the digital computers to simplify the subtraction operation and for the logical manipulations.
For each radix-r system (radix r represents base of number system) there are two types of complements.

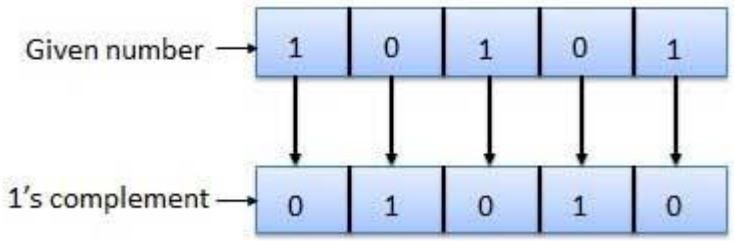
1. r's complement
2. (r-1)'s complement

Binary System complements:

As the binary system has base r = 2. So the two types of complements for the binary system are 2's complement and 1's complement.

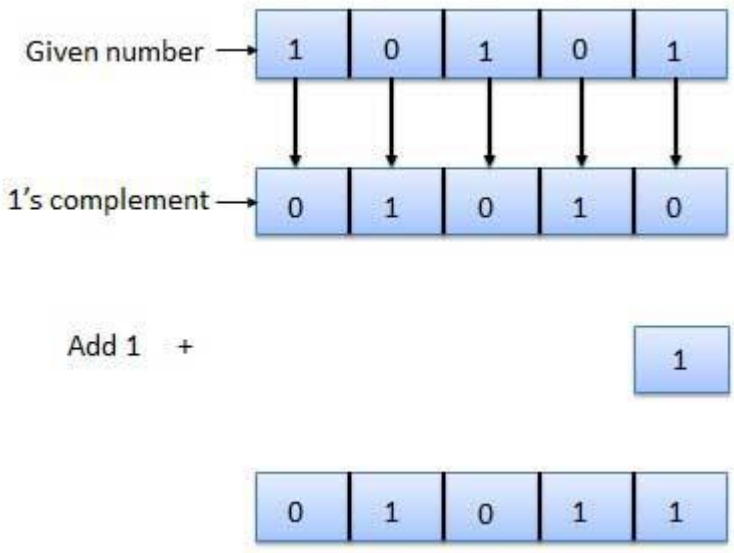
1's complement-

The 1's complement of a number is found by changing all 1's to 0's and all 0's to 1's. This is called as taking complement or 1's complement. Example of 1's Complement is as follows.



2's complement-

The 2's complement of binary number is obtained by adding 1 to the Least Significant Bit (LSB) of 1's complement of the number.
2's complement = 1's complement + 1
Example of 2's Complement is as follows.



Binary Arithmetic

Binary arithmetic is essential part of all the digital computers and many other digital system.

Binary Addition

It is a key for binary subtraction, multiplication, division. There are four rules of binary addition.

Case	A + B	Sum	Carry
1	0 + 0	0	0
2	0 + 1	1	0
3	1 + 0	1	0
4	1 + 1	0	1

In fourth case, a binary addition is creating a sum of (1 + 1 = 10) i.e. 0 is written in the given column and a carry of 1 over to the next column.

Example – Addition

0011010 + 001100 = 00100110

```
  1 1          carry
  0 0 1 1 0 1 0  = 26(10)
+ 0 0 0 1 1 0 0  = 12(10)
-----------------
  0 1 0 0 1 1 0  = 38(10)
```

Binary Subtraction

Subtraction and Borrow, these two words will be used very frequently for the binary subtraction. There are four rules of binary subtraction.

Case	A - B	Subtract	Borrow
1	0 - 0	0	0
2	1 - 0	1	0
3	1 - 1	0	0
4	0 - 1	0	1

Example – Subtraction

0011010 - 001100 = 00001110

```
    1 1        borrow
  0 0 1 1 0 1 0  = 26(10)
- 0 0 0 1 1 0 0  = 12(10)
-----------------
  0 0 0 1 1 1 0  = 14(10)
```

Binary Multiplication

Binary multiplication is similar to decimal multiplication. It is simpler than decimal multiplication because only 0s and 1s are involved. There are four rules of binary multiplication.

Case	A x B	Multiplication
1	0 x 0	0
2	0 x 1	0
3	1 x 0	0
4	1 x 1	1

Example – Multiplication

Example:

0011010 x 001100 = 100111000

$$
\begin{array}{rl}
0\,0\,1\,1\,0\,1\,0 & = 26_{10} \\
\times\, 0\,0\,0\,1\,1\,0\,0 & = 12_{10} \\
\hline
0\,0\,0\,0\,0\,0\,0 & \\
0\,0\,0\,0\,0\,0\,0 & \\
0\,0\,1\,1\,0\,1\,0 & \\
0\,0\,1\,1\,0\,1\,0 & \\
\hline
0\,1\,0\,0\,1\,1\,1\,0\,0\,0 & = 312_{10}
\end{array}
$$

Binary Division

Binary division is similar to decimal division. The long hand division method is used for this purpose. In decimal division it is seen how many times the divisor goes into the dividend, but in binary division there are only two possibilities o and 1 i.e. if the divisor goes into the dividend, the quotient becomes 1, if it does not the quotient becomes zero. The divisor is then subtracted from dividend. The next bit of the dividend is copied in the remainder of the subtraction and again seen if the divisor goes into the dividend. This process is continued till all the bits of the dividend are considered.

Example – Division

101010 / 000110 = 000111

$$
\begin{array}{rl}
 & 1\,1\,1 \quad = 7_{10} \\
000110\ \overline{)\ 1\,0\,1\,0\,1\,0} & = 42_{10} \\
 -\,1\,1\,0 & = 6_{10} \\
\hline
 1\,0\,0\,1 & \\
 -\,1\,1\,0 & \\
\hline
 1\,1\,0 & \\
 -\,1\,1\,0 & \\
\hline
 0 &
\end{array}
$$

Unit-3

Binary Code

In the coding, when numbers, letters or words are represented by a specific group of symbols, it is said that the number, letter or word is being encoded. The group of symbols is called as a code. The digital data is represented, stored and transmitted as group of binary bits. This group is also called as **Binary Code**. The binary code is represented by the number as well as alphanumeric letter.

Advantages of Binary Code

Following is the list of advantages that binary code offers.

- Binary codes are suitable for the computer applications.
- Binary codes are suitable for the digital communications.
- Binary codes make the analysis and designing of digital circuits if we use the binary codes.
- Since only 0 & 1 are being used, implementation becomes easy.

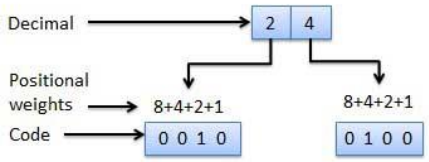
Classification of binary codes

The codes are broadly categorized into following categories.

- Weighted Codes
- Non-Weighted Codes
- Binary Coded Decimal Code
- Alphanumeric Codes

Weighted Codes:

Weighted binary codes are those binary codes which obey the positional weight principle. Each position of the number represents a specific weight. Several systems of the codes are used to express the decimal digits 0 through 9. In these codes each decimal digit is represented by a group of four bits.



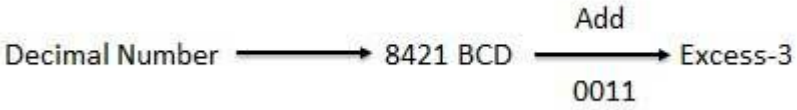
Non-Weighted Codes:

In this type of binary codes, the positional weights are not assigned.

The examples of non-weighted codes are **Excess-3 code** and **Gray code.**

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 adding $(0011)_2$ or (3)10 to each code word in 8421. The excess-3 codes are obtained as follows −



Example

Decimal	BCD 8 4 2 1	Excess-3 BCD + 0011
0	0 0 0 0	0 0 1 1
1	0 0 0 1	0 1 0 0
2	0 0 1 0	0 1 0 1
3	0 0 1 1	0 1 1 0
4	0 1 0 0	0 1 1 1
5	0 1 0 1	1 0 0 0
6	0 1 1 0	1 0 0 1
7	0 1 1 1	1 0 1 0
8	1 0 0 0	1 0 1 1
9	1 0 0 1	1 1 0 0

Gray Code:

It is the non-weighted code and it is not arithmetic codes. That means 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 as shown in fig. As only one bit changes at a time, the gray code is

called as a unit distance code. The gray code is a cyclic code. Gray code cannot be used for arithmetic operation.

Decimal	BCD	Gray
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 1
3	0 0 1 1	0 0 1 0
4	0 1 0 0	0 1 1 0
5	0 1 0 1	0 1 1 1
6	0 1 1 0	0 1 0 1
7	0 1 1 1	0 1 0 0
8	1 0 0 0	1 1 0 0
9	1 0 0 1	1 1 0 1

Binary Coded Decimal (BCD) code-

In this code each decimal digit is represented by a 4-bit binary number. BCD is a way to express each of the decimal digits with a binary code. In the BCD, with four bits we can represent sixteen numbers (0000 to 1111). But in BCD code only first ten of these are used (0000 to 1001). The remaining six code combinations i.e. 1010 to 1111 are invalid in BCD.

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

Alphanumeric Codes-

A binary digit or bit can represent only two symbols as it has only two states '0' or '1'. But this is not enough for communication between two computers because there we need many more symbols for communication. These symbols are required to represent 26 alphabets with capital and small letters, numbers from 0 to 9, punctuation marks and other symbols.

The alphanumeric codes are the codes that represent numbers and alphabetic characters. Mostly such codes also represent other characters such as symbol and various instructions necessary for conveying information. An alphanumeric code should at least represent 10 digits and 26 letters of alphabet i.e. total 36 items. The following two alphanumeric codes are very commonly used for the data representation.

- American Standard Code for Information Interchange (ASCII).
- Extended Binary Coded Decimal Interchange Code (EBCDIC).

ASCII code:

ASCII stands for American Standard Code for Information Interchange. Several American computer manufacturer have adopted ASCII as their computer's internal code. This code is popular in data communications, is used almost exclusively to represent data internally in microcomputers, and frequently found in larger computers produced by some vendors.

American National Standards Institute (ANSI) published ASCII standard in 1963. However, the standard lacked lowercase letters and ANSI subsequently revised ASCII in 1967. Later revisions in 1968, 1977, and finally in 1986 brought it in its present form. Today, ASCII is one of the most popular and widely supported character encoding standards

ASCII is of two types

- **ASCII-7**
- **ASCII-8**

ASCII-7 is a 7-bit code that can characterize 128 distinctive characters. Computers the usage of 8-bit byte (group of 8 bits for 1 byte) and the 7-bit ASCII both set the 8th bit (leftmost bit) of every byte as zero or use it as a parity bit.

ASCII-8 is an extended version of ASCII-7. It is an 8-bit code that can signify 256 distinctive characters. It adds the extra bit to the left of the seventh bit (leftmost bit) of ASCII-7 codes.

EBCDIC code:

The major problem with BCD code is that it can represent only 64 different characters. Hence , researches extended BCD code from a 6-bit code to an 8-bit code. The added two bits are used as extra zone bits, expanding the zone to 4 bits. The resulting code is referred to as the ***Extended Binary-Coded Decimal Interchange Code (EBCDIC)*.**
In the code, it is possible to represent 256 distinctive characters.In addition to the number personality requirements , this additionally allows a large variety of printable characters and numerous non-printable manage characters.
Since **EBCDIC** is an 8-bit code, we can divide it without difficulty into 4-bits corporations and use one hexadecimal digit for representing every of these. Hence, computer authorities use hexadecimal quantity machine as shortcut notation for reminiscence dump in computers that use EBCDIC for interior illustration of characters.
EBCDIC is used primarily in large IBM computers.

Boolean Function

A Boolean function is an algebraic form of Boolean expression. A Boolean function of n-variables is represented by f(x1, x2, x3….xn). By using Boolean laws and theorems, we can simplify the Boolean functions of digital circuits.
There are different ways of representing a Boolean function is shown below.

- Sum-of-Products (SOP) Form
- Product-of-sums (POS) form

.

Sum of Product (SOP) Form

The sum-of-products (SOP) form is a method (or form) of simplifying the Boolean expressions of logic gates. In this SOP form of Boolean function representation, the variables are operated by AND (product) to form a product term and all these product terms are ORed (summed or added) together to get the final function.
A sum-of-products form can be formed by adding (or summing) two or more product terms using a Boolean addition operation. Here the product terms are defined by using the AND operation and the sum term is defined by using OR operation.
The sum-of-products form is also called as Disjunctive Normal Form as the product terms are ORed together and Disjunction operation is logical OR. Sum-of-products form is also called as Standard SOP.
Examples
AB + ABC + CDE
SOP form can be obtained by

- Writing an AND term for each input combination, which produces HIGH output.
- Writing the input variables if the value is 1, and write the complement of the variable if its value is 0.
- OR the AND terms to obtain the output function.

Product of Sums (POS) Form

The product of sums form is a method (or form) of simplifying the Boolean expressions of logic gates. In this POS form, all the variables are ORed, i.e. written as sums to form sum terms.
All these sum terms are ANDed (multiplied) together to get the product-of-sum form. This form is exactly opposite to the SOP form. So this can also be said as "Dual of SOP form".
Here the sum terms are defined by using the OR operation and the product term is defined by using AND operation. When two or more sum terms are multiplied by a Boolean OR operation, the resultant output expression will be in the form of product-of-sums form or POS form.
The product-of-sums form is also called as Conjunctive Normal Form as the sum terms are ANDed together and Conjunction operation is logical AND. Product-of-sums form is also called as Standard POS.

Examples

(A+B) * (A + B + C) * (C +D)

POS form can be obtained by

- Writing an OR term for each input combination, which produces LOW output.
- Writing the input variables if the value is 0, and write the complement of the variable if its value is 1.
- AND the OR terms to obtain the output function.

Difference between SOP and POS :

S.NO.	SOP	POS
1.	A way of representing boolean expressions as sum of product terms.	A way of representing boolean expressions as product of sum terms.
2.	SOP uses minterms. Minterm is product of boolean variables either in normal form or complemented form.	POS uses maxterms. Maxterm is sum of boolean variables either in normal form or complemented form.
3.	It is sum of minterms. Minterms are represented as 'm'	It is product of maxterms. Maxterms are represented as 'M'
4.	SOP is formed by considering all the minterms, whose output is HIGH(1)	POS is formed by considering all the maxterms, whose output is LOW(0)
5.	While writing minterms for SOP, input with value 1 is considered as the variable itself and input with value 0 is considered as complement of the input.	While writing maxterms for POS, input with value 1 is considered as the complement and input with value 0 is considered as the variable itself.

Unit-3

Logic Gates

Logic gates are used to carry out logical operations on single or multiple binary inputs and give one binary output. In simple terms, logic gates are the electronic circuits in a digital system.
These are important electronic circuit that are mainly based on the Boolean function.
To obtain the required circuit we can join any number of logic gates together. We can design many special circuits like FLIPFLOPS, MULTIPLEXERS, REGISTERS, and COUNTERS etc.

Types of Logic Gates

There are following several basic logic gates used in performing operations in digital systems.

- AND Gate
- OR Gate
- NOT Gate
- NAND Gate
- NOR Gate
- XOR Gate
- X-NOR

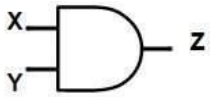
AND Gate :

It takes 2-inputs and gives only a single output. The output of an AND gate is only 1 if and only if all the inputs are 1.

The Boolean expression of two-input AND gate is

Z = X.Y, where X and Y are 2-inputs, Z is the output and **.** represents **AND** operation.

The graphical symbol of a two-input AND gate is given as



AND gate symbol

The truth table of a two-input AND gate is given as

Input-X	Input-Y	Output Z = X.Y
0	0	0
0	1	0
1	0	0
1	1	1

Truth table of AND gate

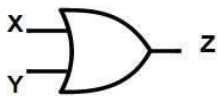
OR Gate :

It takes 2-inputs and gives only a single output. The output of an OR gate is only if one or more inputs are1.

The Boolean expression of two-input OR gate is

Z = X+Y, where X and Y are 2-inputs , Z is the output and **+** represents **OR** operation

The graphical symbol of a two-input OR gate is given as



OR gate symbol

The truth table of a two-input OR gate is given as

Input-X	Input-Y	Output Z = X+Y
0	0	0
0	1	1
1	0	1
1	1	1

Truth table of OR gate

NOT Gate:

NOT gate is also known as **Inverter**. It has one input and one output .
The output of NOT gate is only 1 when input is 0 and output is 0 only when input is 1.

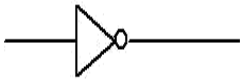
The Boolean expression of NOT gate is
$Z = \overline{X}$, where X is input, Z is the output and —— represents NOT operation.
The graphical symbol of NOT gate is given as

NOT gate symbol

The truth table of NOT gate is given as

Input-X	Output $Z = \overline{X}$
0	1
1	0

Truth table of NOT gate



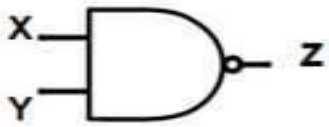
- The three gates (OR, AND and NOT), when connected in various combinations, give us basic logic gates such as NAND, NOR gates, which are the universal building blocks of digital circuits.Therefore ,NAND and NOR gate is also called as **UNIVERSAL GATE**.

NAND Gate :

NAND gate is the combination of AND and NOT gate. It takes 2-inputs and gives only a single output. The output of NAND gate is only 1 if any of the inputs are 0 and it will be 0 only when all inputs are 1.

The Boolean expression of two-input NAND gate is
$Z = \overline{X.Y}$, where X and Y are 2-inputs , Z is the output

The graphical symbol of a two-input NAND gate is given a



NAND gate symbol

The truth table of a two-input NAND gate is given as

Input-X	Input-Y	Output $Z = \overline{X.Y}$
0	0	1
0	1	1
1	0	1
1	1	0

Truth table of NAND gate

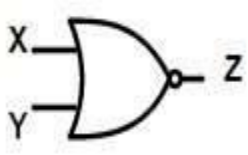
NOR Gate :

NOR gate is the combination of OR and NOT gate. It takes 2-inputs and gives only a single output. The output of NOR gate is only 1 when all inputs are 0 and it will be 0 only if any of the inputs are 1.

The Boolean expression of two-input NOR gate is

$Z = \overline{X+Y}$, where X and Y are 2-inputs , Z is the output

The graphical symbol of a two-input NOR gate is given a



NOR gate symbol

The truth table of a two-input NOR gate is given as

Input-X	Input-Y	Output $Z=\overline{X+Y}$
0	0	1
0	1	0
1	0	0
1	1	0

Truth table of NOR gate

X-OR Gate :

X-OR gate is a special type of gate. It can be used in the half adder, full adder and subtractor. It takes 2-inputs and gives only a single output .The output of X-OR gate is only 1 if both input are different otherwise the output will be 0.

The Boolean expression of two-input X-OR gate is **Z** $= X \oplus Y = \overline{X}.Y + X.\overline{Y}$, where X and Y are 2-inputs , Z is the output

The graphical symbol of a two-input X-OR gate is given as

x

z

y

XOR gate symbol

The truth table of a two-input X-OR gate is given as

Input-X	Input-Y	Output Z $= X \oplus Y$
0	0	0
0	1	1
1	0	1
1	1	0

Truth table of X-OR gate

X-NOR Gate :

X-NOR gate is a special type of gate. It can be used in the half adder, full adder and subtractor. It takes 2-inputs and gives only a single output .The output of X-NOR gate is only 1 if both input are same i.e both 0 or both 1, otherwise the output will be 0.
The Boolean expression of two-input X-OR gate is

$Z = \overline{X \oplus Y} = X \cdot Y + \overline{X} \cdot \overline{Y}$ X and Y are 2-inputs , Z is the output

The graphical symbol of a two-input X=OR gate is given as

X-NOR gate symbol

The truth table of a two-input X-NOR gate is given as

Input-X	Input-Y	Output $Z = \overline{X \oplus Y}$
0	0	1
0	1	0
1	0	0
1	1	1

Truth table of X-NOR gate

Karnaugh Map(K-Map)

To simplified the Boolean functions using Boolean postulates and theorems. It is a time consuming process and we have to re-write the simplified expressions after each step.To overcome this difficulty, **Karnaugh** introduced a method for simplification of Boolean functions in an easy way. This method is known as **Karnaugh Map method or K-Map** method.

In many digital circuits and practical problems we need to find expression with minimum variables. We can minimize Boolean expressions of 2, 3, 4 variables very easily using K-map without using any Boolean algebra theorems. K-map can take two forms Sum of Product (SOP) and Product of Sum (POS) according to the need of problem.

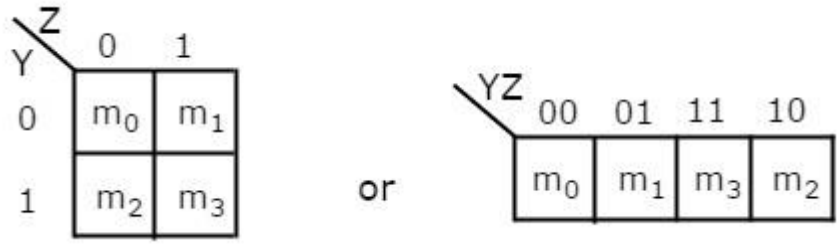
K-map is a graphical method but it gives more information than TRUTH TABLE. We fill grid of K-map with 0's and 1's then solve it by making groups.

K-Maps for 2 to 4 Variables

K-Map method is most suitable for minimizing Boolean functions of 2 variables to 4 variables. Now, let us discuss about the K-Maps for 2 to 4 variables one by one.

2 Variable K-Map

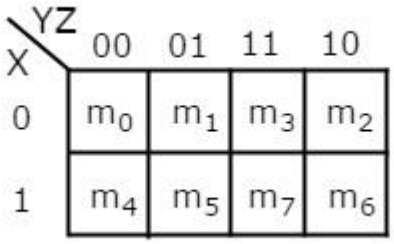
The number of cells in 2 variable K-map is four, since the number of variables is two. The following figure shows **2 variable K-Map**.



- There is only one possibility of grouping 4 adjacent min terms.
- The possible combinations of grouping 2 adjacent min terms are {(m_0, m_1), (m_2, m_3), (m_0, m_2) and (m_1, m_3)}.

3 Variable K-Map

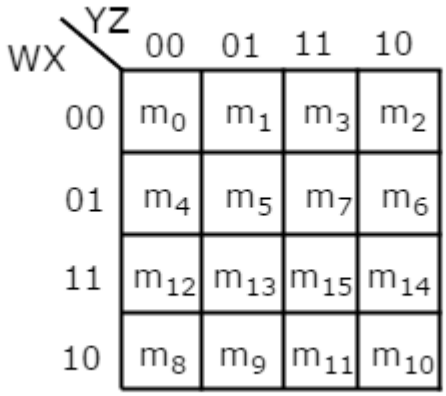
The number of cells in 3 variable K-map is eight, since the number of variables is three. The following figure shows **3 variable K-Map**.



- There is only one possibility of grouping 8 adjacent min terms.
- The possible combinations of grouping 4 adjacent min terms are {(m_0, m_1, m_3, m_2), (m_4, m_5, m_7, m_6), (m_0, m_1, m_4, m_5), (m_1, m_3, m_5, m_7), (m_3, m_2, m_7, m_6) and (m_2, m_0, m_6, m_4)}.
- The possible combinations of grouping 2 adjacent min terms are {(m_0, m_1), (m_1, m_3), (m_3, m_2), (m_2, m_0), (m_4, m_5), (m_5, m_7), (m_7, m_6), (m_6, m_4), (m_0, m_4), (m_1, m_5), (m_3, m_7) and (m_2, m_6)}.
- If x=0, then 3 variable K-map becomes 2 variable K-map.

4 Variable K-Map

The number of cells in 4 variable K-map is sixteen, since the number of variables is four. The following figure shows **4 variable K-Map**.



- There is only one possibility of grouping 16 adjacent min terms.
- Let R_1, R_2, R_3 and R_4 represents the min terms of first row, second row, third row and fourth row respectively. Similarly, C_1, C_2, C_3 and C_4 represents the min terms of first column, second column, third column and fourth column respectively. The possible combinations of grouping 8 adjacent min terms are {(R_1, R_2), (R_2, R_3), (R_3, R_4), (R_4, R_1), (C_1, C_2), (C_2, C_3), (C_3, C_4), (C_4, C_1)}.
- If w=0, then 4 variable K-map becomes 3 variable K-map.

Minimization of Boolean Functions using K-Maps

If we consider the combination of inputs for which the Boolean function is '1', then we will get the Boolean function, which is in **standard sum of products** form after simplifying the K-map.

Similarly, if we consider the combination of inputs for which the Boolean function is '0', then we will get the Boolean function, which is in **standard product of sums** form after simplifying the K-map.

Boolean Algebra

Boolean Algebra is used to analyze and simplify the digital (logic) circuits. It uses only the binary numbers i.e. 0 and 1. It is also called as **Binary Algebra** or **logical Algebra**. Boolean algebra was invented by **George Boole** in 1854.

Rule in Boolean Algebra:

Following are the important rules used in Boolean algebra.

- Variable used can have only two values. Binary 1 for HIGH and Binary 0 for LOW.
- Complement of a variable is represented by an overbar (-). Thus, complement of variable B is represented as $\overline{B}$. Thus if B = 0 then $\overline{B}$ = 1 and B = 1 then $\overline{B}$ = 0.
- ORing of the variables is represented by a plus (+) sign between them. For example ORing of A, B, C is represented as A + B + C.
- Logical ANDing of the two or more variable is represented by writing a dot between them such as A.B.C. Sometime the dot may be omitted like ABC.

Boolean Laws:

There are six types of Boolean Laws.

Commutative law

Any binary operation which satisfies the following expression is referred to as commutative operation.

(i) $A.B = B.A$ (ii) $A + B = B + A$

Commutative law states that changing the sequence of the variables does not have any effect on the output of a logic circuit.

Associative law

This law states that the order in which the logic operations are performed is irrelevant as their effect is the same.

(i) $(A.B).C = A.(B.C)$ (ii) $(A + B) + C = A + (B + C)$

Distributive law

Distributive law states the following condition.

$A.(B + C) = A.B + A.C$

AND law

These laws use the AND operation. Therefore they are called as **AND** laws.

(i) $A.0 = 0$ (ii) $A.1 = A$

(iii) $A.A = A$ (iv) $A.\overline{A} = 0$

OR law

These laws use the OR operation. Therefore they are called as **OR** laws.

(i) $A + 0 = A$ (ii) $A + 1 = 1$

(iii) $A + A = A$ (iv) $A + \overline{A} = 1$

INVERSION law

This law uses the NOT operation. The inversion law states that double inversion of a variable results in the original variable itself.

$\overline{\overline{A}} = A$