UNIT -1
BOOLEAN ALGEBRA AND LOGIC GATES
TOPIC – 2
BINARY,OCTAL AND HEXADECIMAL NUMBER SYSTEMS AND NUMBER BASE CONVERSION

TYPES OF NUMBER SYSTEM:

There are four types of number systems. They are

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

Decimal	Binary	Octal	Hexadecimal
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10
17	10001	21	11
18	10010	22	12
19	10011	23	13
20	10100	24	14

- Base of the number system such as 2,8,10 or 16 defines the number of unique symbols.
- The first digit in the number system is always zero and last digit in the number system is always base-1.
- The standard notation of using the subscript to indicate the base of the number still applies.$(10)_8$ indicates eight, while $(10)_{16}$ indicates sixteen.

Decimal Number System:

- Base (also called radix) = 10

10 digits { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }

- Digit Position

Integer & fraction

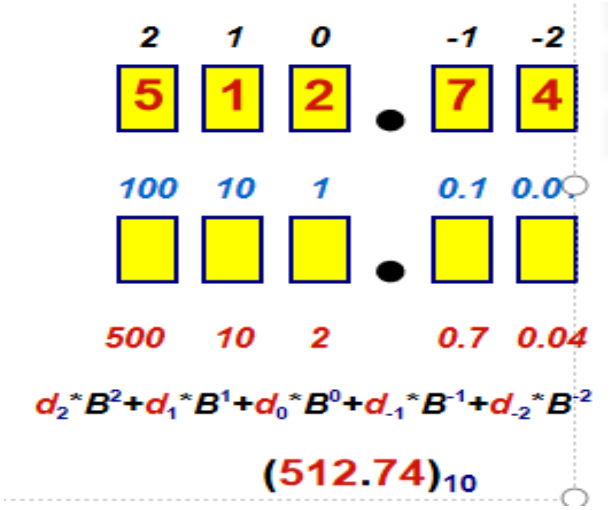
- Digit Weight

Weight = (Base) $^{\text{Position}}$

- Magnitude

Sum of "Digit x Weight"

Formal Notation:



Binary Number System:

- Base = 2

2 digits { 0, 1 }, called binary digits or "bits"

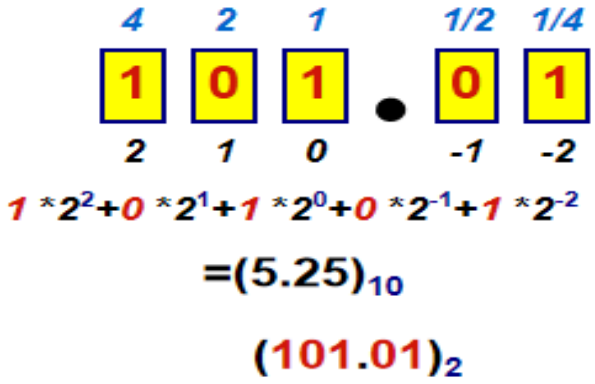
- Weights

Weight = (Base) $^{\text{Position}}$

- Magnitude

Sum of "Bit x Weight"

Formal Notation:



Groups of 4 bits = Nibble

Groups of 8 bits = Byte

Octal Number System:

- Base = 8

8 digits { 0, 1, 2, 3, 4, 5, 6, 7 }

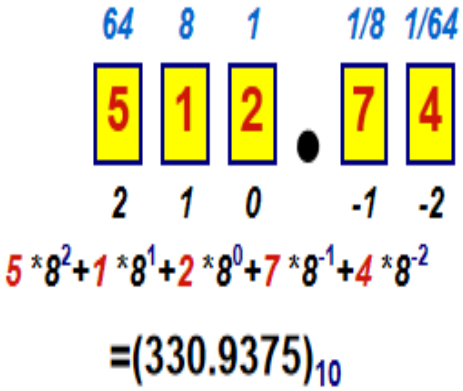
- Weights

Weight = (Base) $^{\text{Position}}$

- Magnitude

Sum of "Digit x Weight"

Formal Notation:



Hexadecimal Number System:

- Base = 16

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

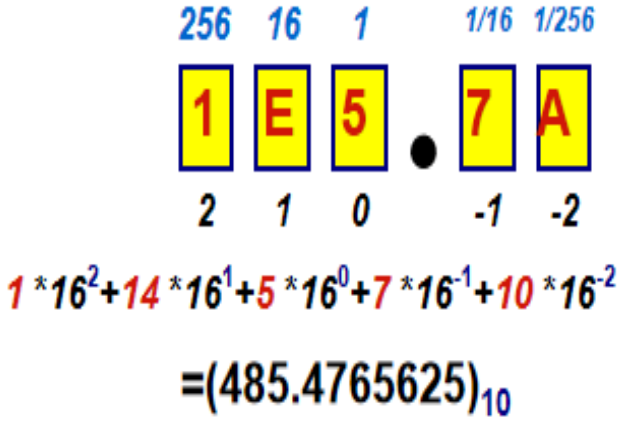
- Weights

Weight = (Base) $^{\text{Position}}$

- Magnitude

Sum of "Digit x Weight"

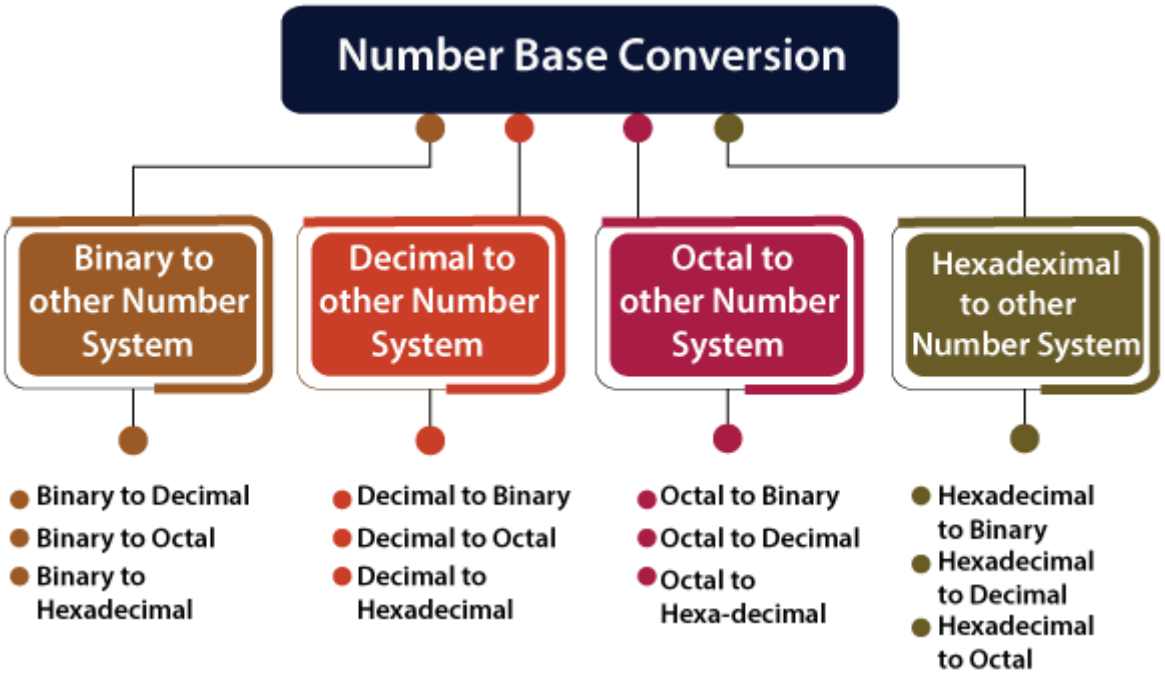
Formal Notation:



UNIT -1
BOOLEAN ALGEBRA AND LOGIC GATES
TOPIC – 3
NUMBER BASE CONVERSION

Number Base conversions:

Human beings use decimal number system while computer uses binary number system. Therefore, it is necessary to convert decimal number system into its equivalent binary.



Decimal (Integer) to Binary Conversion:

- Divide the number by the 'Base' (=2)
- Take the remainder (either 0 or 1) as a coefficient
- Take the quotient and repeat the division

Example: $(13)_{10}$

	Quotient	Remainder	Coefficient
13 / 2 =	6	1	$a_0 = 1$
6 / 2 =	3	0	$a_1 = 0$
3 / 2 =	1	1	$a_2 = 1$
1 / 2 =	0	1	$a_3 = 1$

Answer: $(13)_{10} = (a_3\ a_2\ a_1\ a_0)_2 = (1101)_2$

MSB LSB

Decimal (Fraction) to Binary Conversion:

- Multiply the number by the 'Base' (=2)
- Take the integer (either 0 or 1) as a coefficient
- Take the resultant fraction and repeat the multiplication

Example: $(0.625)_{10}$

		Integer		Fraction	Coefficient
0.625	* 2 =	1	.	25	$a_{-1} = 1$
0.25	* 2 =	0	.	5	$a_{-2} = 0$
0.5	* 2 =	1	.	0	$a_{-3} = 1$

Answer: $(0.625)_{10} = (0.a_{-1}\ a_{-2}\ a_{-3})_2 = (0.101)_2$

MSB LSB

Example:Convert (105.15)10 into binary.

- Successively divide the given decimal integer number by 2 till the quotient is 0.
- To convert Fractional part of decimal ,multiply the number by the 'Base' (2)
- Take the integer as a coefficient
- Take the resultant fraction and repeat the multiplication.
- Combine both integral and fractional part of binary number.

Integer part		Fraction part
2 I 105		0.15 x 2 = 0.30
2 I 52 —	1	0.30 x 2 = 0.60
2 I 26 —	0	0.60 x 2 = 1.20
2 I 13 —	0	0.20 x 2 = 0.40
2 I 6 —	1	0.40 x 2 = 0.80
2 I 3 —	0	0.80 x 2 = 1.60
2 I 1 —	1	
0 —	1	

Result of (105.15)10 is (1101001.001001)2

Decimal to Octal Conversion:

- Successively divide the given decimal integer number by 8 till the quotient is 0.

Example: $(175)_{10}$

	Quotient	Remainder	Coefficient
175 / 8 =	21	7	$a_0 = 7$
21 / 8 =	2	5	$a_1 = 5$
2 / 8 =	0	2	$a_2 = 2$

Answer: $(175)_{10} = (a_2\ a_1\ a_0)_8 = (257)_8$

Decimal(Fractional) to Octal Conversion

- Multiply the number by the 'Base' (8)
- Take the integer as a coefficient
- Take the resultant fraction and repeat the multiplication

Example: $(0.3125)_{10}$

		Integer		Fraction	Coefficient
0.3125	* 8 =	2	.	5	$a_{-1} = 2$
0.5	* 8 =	4	.	0	$a_{-2} = 4$

Answer: $(0.3125)_{10} = (0.a_{-1}\, a_{-2}\, a_{-3})_8 = (0.24)_8$

Decimal to hexadecimal conversion:

- Successively divide the given number by 16 till the quotient is 0.
- To convert the given decimal fractions to octal successively multiply the decimal fraction and the subsequent decimal fractions by 16 till the product is 0 or till the required accuracy is obtained.

Example:Convert $(2598.675)_{10}$ into hexadecimal.

	Decimal	Hex		Hex
16 | 2598				
16 | 162	— 6	6	0.675 x 16 = 10.8	A
16 | 10	— 2	2	0.800 x 16 = 12.8	C
0	— 10	A	0.800 x 16 = 12.8	C
			0.800 x 16 = 12.8	C

$(2598.675)_{10} = (A26.ACCC)_{16}$

UNIT -1
BOOLEAN ALGEBRA AND LOGIC GATES
TOPIC – 4
ANY NUMBER BASE TO DECIMAL AND OTHER BASE CONVERSION

Binary to Decimal Conversion:

- Multiply each bit by 2^n, where n is the "weight" of the bit
- The weight is the position of the bit, starting from 0 on the right
- Add the results.

Example 1: $(101011)_2 = (?)_{10} = (43)_{10}$

$(101011)_2 = (1 \times 2^5) + (0 \times 2^4) + (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0)$

$= 32+0+8+0+2+1$

$= (43)_{10}$

Example 2: Convert $(111.101)_2$

$(111.101)_2 = (1 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) + (1 \times 2^{-1}) + (0 \times 2^{-2}) + (1 \times 2^{-3}) =$

$= 4+ 2+ 1 + 0.5 + 0 + 0.125$

$= (7.625)_{10}$

Octal to Decimal Conversion:

- Multiply each bit by 8^n, where n is the "weight" of the bit
- The weight is the position of the bit, starting from 0 on the right
- Add the results

Example 1: $(724)_8 \Rightarrow 7 \times 8^2 + 2 \times 8^1 + 4 \times 8^0$

$= 448+16+4$

$= (468)_{10}$

Hexadecimal to Decimal Conversion:

- Multiply each bit by 16^n, where n is the "weight" of the bit
- The weight is the position of the bit, starting from 0 on the right
- Add the results

Example 1: $(ABC)_{16} \Rightarrow A \times 16^2 + B \times 16^1 + C \times 16^0$

$= 10 \times 256 + 11 \times 16 + 12 \times 1$

$= 2560+176+12$

$= (2748)_{10}$

Example 2: Convert $(31.D2)_{16}$ in to Decimal $\Rightarrow (3 \times 16^1) + (1 \times 16^0) + (D \times 16^{-1}) + (2 \times 16^{-2})$

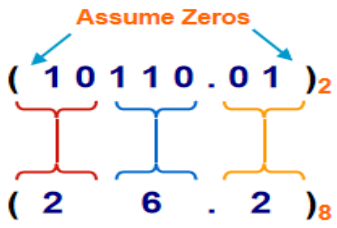
$= 48+1+0.8125+0.0078125$

$= (49.8203125)_{10}$

Binary – Octal Conversion:

- $8 = 2^3$
- Each group of 3 bits represents an octal digit

Example:



Works both ways (*Binary* to *Octal* & *Octal* to *Binary*)

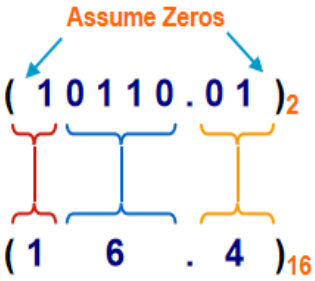
Octal	Binary
0	0 0 0
1	0 0 1
2	0 1 0
3	0 1 1
4	1 0 0
5	1 0 1
6	1 1 0
7	1 1 1

Binary − Hexadecimal Conversion:

- $16 = 2^4$
- Each group of 4 bits represents a hexadecimal digit

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

Example:

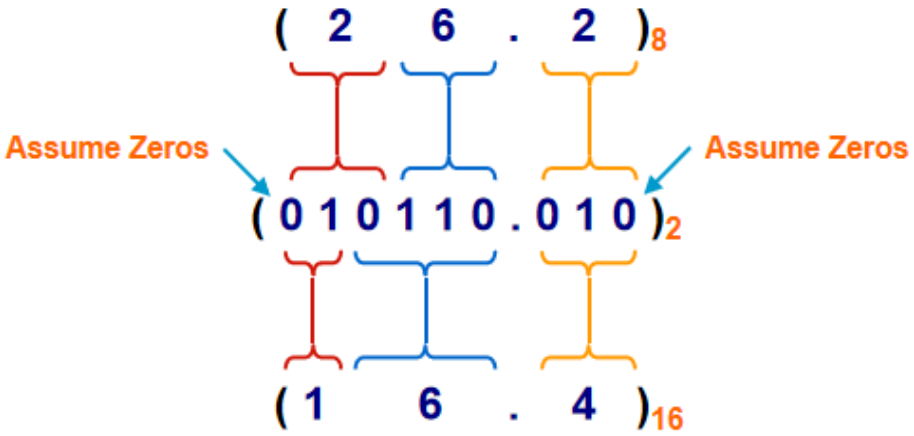


Works both ways (*Binary* to *Hex* & *Hex* to *Binary*)

Octal − Hexadecimal Conversion:

- Convert to Binary as an intermediate step

Example:



Works both ways (*Octal* to *Hex* & *Hex* to *Octal*)

The Power of 2:

n	2^n
0	$2^0=1$
1	$2^1=2$
2	$2^2=4$
3	$2^3=8$
4	$2^4=16$
5	$2^5=32$
6	$2^6=64$
7	$2^7=128$

n	2^n	
8	$2^8=256$	
9	$2^9=512$	
10	$2^{10}=1024$	Kilo
11	$2^{11}=2048$	
12	$2^{12}=4096$	
20	$2^{20}=1M$	Mega
30	$2^{30}=1G$	Giga
40	$2^{40}=1T$	Tera

UNIT -1
BOOLEAN ALGEBRA AND LOGIC GATES
TOPIC – 5
BINARY ARITHMETIC

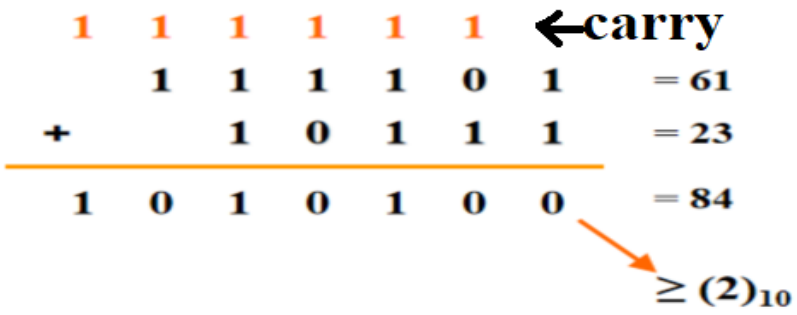
- Binary arithmetic is an essential part of various digital systems.
- You can add, subtract, multiply, and divide binary numbers using various methods.
- These operations are much easier than decimal number arithmetic operations because the binary system has only two digits: 0 and 1.

Binary Addition:

- Binary addition is much easier than the decimal addition
- The four rules of binary addition are:
  1. 0 + 0 = 0
  2. 0 + 1 = 1
  3. 1 + 0 = 1
  4. 1 + 1 =10
- When we give the input for x = 0 and y = 0, then the output is equal to 0.
- When x = 0 or 1 and y = 1 or 0, then x+y = 1.
- But when both x and y are equal to 1, then their sum equals to 0, but the carryout r will equal to 1, which means basically 1 + 1 = 10 in binary addition, where 1 is carry forwarded to the next digit.

Example:

$$(111101)_2+(10111)_2=(1010100)_2$$



Binary Subtraction:

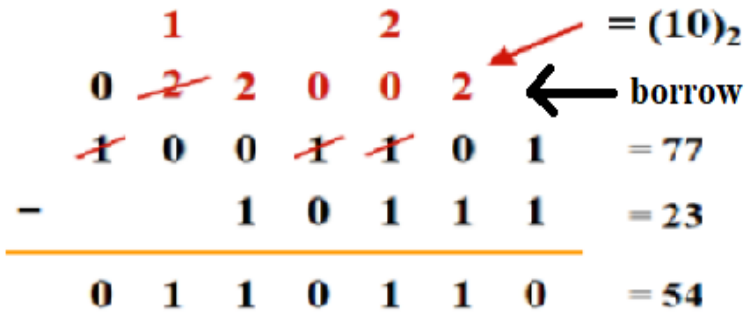
- Binary subtraction is very easy as the addition, subtraction of one bit from another.
- Subtraction and borrowing are the two steps involved in binary subtraction.

The rules for binary subtraction are:

1. 0 - 0 = 0
2. 1 - 0 = 1
3. 1 - 1 = 0
4. 0 - 1 = 1 (borrow of 1)

- In rule 4, borrow of 1 is performed as 0 has a lower value than one, so we cannot subtract from it, so we borrow one from the next column.
- However, borrowing from the next higher-order column is sometimes essential when subtracting.
- One is borrowed from the next higher-order column, which leaves 0 in that column and creates a$(10)_2$, i.e., two being subtracted.

Example:$(1001101)_2$-$(10111)_2$=$(0110110)_2$



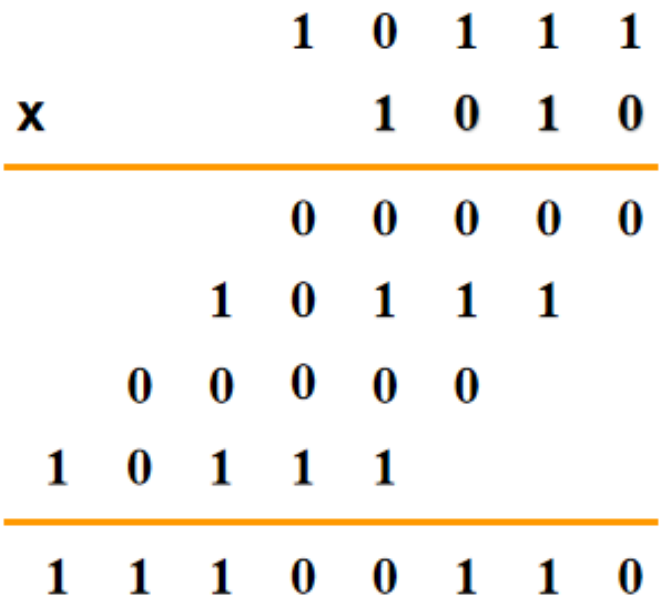
Binary Multiplication:

- Binary multiplication is similar to decimal multiplication.
- However, as there are only 2 bits, 0 and 1, It is much simpler than decimal multiplication because there are only two possible results of multiplying two bits.

There are four rules of binary multiplication.

1. **0 x 0 = 0**
2. **0 x 1 = 0**
3. **1 x 0 = 0**
4. **1 x 1 = 1**

Example:$(10111)_2$X $(1010)_2$=$(0110110)_2$



UNIT -1
BOOLEAN ALGEBRA AND LOGIC GATES
TOPIC – 6
COMPLEMENTS

- In digital computers, to simplify the subtraction operation & for logical manipulation complements are used.
- There are two types of complements used in Digital system.
    1. Diminished radix complement or **(r-1)'s** complement
    2. Radix complement or **r's** complement

Diminished Radix Complement - (r-1)'s Complement:

- The diminished radix complements are called by the radix-1.
- The diminished radix complement of a decimal number number system is 9's complement and in binary number system it is 1's complement.
- Given a number N in base r having n digits,the (r–1)'s complement of N is defined as: $(r^n – 1) – N$.

9's Complement:

- Subtract each digit of given decimal number from 9 we obtain 9's complement.

Example for 6-digit decimal numbers:
9's complement is $(r^n – 1)–N = (10^6–1)–N = 999999–N$

Example:
Find the 9's complement of $(547600)_{10}$

$$\begin{array}{r} 999999 \\ -\ 547600 \\ \hline 452399 \end{array}$$

9′s Complement of $(547600)_{10}$ is 453299.

1's Complement:

- The 1's complement of a binary number is obtained by change each 0 to 1 and each 1 to 0 or subtract each bit from Binary "1".

Example for 7-digit binary numbers:
- 1's complement is $(r^n – 1) – N = (2^7–1)–N = 1111111– N$.

Example 1:
Find the 1's complement of $(1011000)_2$

$$\begin{array}{r} 1111111 \\ -\ 1011000 \\ \hline 0100111 \end{array}$$

1's complent of 1011000 is 0100111

- **Example 2:** Find the 1's complement of binary number $(10110000)_2$

  1's complement of binary number $(10110000)_2$ $(01001111)_2$

[**Note:**Change each 0 to 1 and each 1 to 0 or subtract each bit from Binary "1".]

Radix Complement:

- The r's complement of an n-digit number N in base r is defined as $r^n - N$.
- Comparing with the (r-1) 's complement, we note that the r's complement is obtained by adding 1 to the (r-1) 's complement, since $r^n - N = [(r^n - 1) - N] + 1$.
- The 10's complement of a decimal number is obtained by adding 1 to the 9's complement of a number

Example 1: Base-10

Find the 10's complement of $(012398)_{10}$

$$\begin{array}{r} 999999 \\ 012398 \\ \hline 987601 \\ 1 \\ \hline 987602 \end{array} \quad \begin{array}{l} - \\ \\ \\ + \\ \\ \end{array}$$

10's complement of 012398 is 987602

- **Example 2:** Find the 10's complement of $(246700)_{10}$

The 10's complement of 246700 is 753300

2's Complement :

- Obtain 1's complement of given number and then add 1 to the least significant bit(L.S.B)
- Toggle all bits and add '1' from the right

Example 1: Base-2:

The 2's complement of 10110000 is 01010000

Number: 1 0 1 1 0 0 0 0

1's Comp.: 0 1 0 0 1 1 1 1

2's Comp.: 0 1 0 1 0 0 0 0

$$\begin{array}{r} 1\ 0\ 1\ 1\ 0\ 0\ 0\ 0 \\ 0\ 1\ 0\ 0\ 1\ 1\ 1\ 1 \\ + \qquad\qquad\qquad 1 \\ \hline 0\ 1\ 0\ 1\ 0\ 0\ 0\ 0 \end{array}$$

Example 2: Find the 2's complement of 0110111

The 2's complement of 0110111 is 1001001.

[**Note:**Change each 0 to 1 and each 1 to 0 or subtract each bit from Binary "1",add binary 1 to the least significant bit of 1's Complement.]

UNIT -1
BOOLEAN ALGEBRA AND LOGIC GATES
TOPIC – 7
SUBTRACTION USING COMPLEMENTS AND SIGNED BINARY NUMBERS

Subtraction Using Complements:

The subtraction of two n-digit unsigned numbers M – N in base r can be done as follows:

Subtraction Using r's Complement: -

Subtraction using r's complement (Radix's complement) is a method used in digital systems to perform subtraction by using addition.

- In r's complement subtraction, add r's complement of subtrahend to the minuend.
- If there is a carry out, discard the carry out, and the remaining digits represent the final result.
- If there is no carry ,the result is negative and we need to take the r's complement of this result to get the final answer. The answer will have a negative sign.

1. Add the minuend M to the r's complement of the subtrahend N. Mathematically, $M + (r^n - N) = M - N + r^n$.
2. If $M \geqq N$, the sum will produce and end carry r^n, which can be discarded; what is left is the result $M - N$.
3. If $M < N$, the sum does not produce an end carry and is equal to $r^n - (N - M)$, which is the r's complement of $(N - M)$. To obtain the answer in a familiar form, take the r's complement of the sum and place a negative sign in front.

Example 1

Using 10's complement, subtract 72532 – 3250.

	M =	72532
10's complement of	N =	+96750
	Sum =	169282
	Discard end carry 10^5 =	−100000
	Answer =	69282

[Note: Ignore the end carry]

Example 2

Using 10's complement, subtract 3250 – 72532.

	M =	03250
10's complement of	N =	+27468
	Sum =	30718

There is no end carry.

Therefore, the answer is – (10's complement of 30718) = – 69282.

Example 3

Given the two binary numbers X = 1010100 and Y = 1000011, perform the subtraction (a) X – Y ; and (b) Y – X, by using 2's complement.

(a) X-Y

(a)	X =	1010100
	2's complement of Y =	+0111101
	Sum =	10010001
	Discard end carry 2^7 =	−10000000
	Answer. X − Y =	0010001

(b) Y – X

(b)	Y =	1000011
	2's complement of X =	+0101100
	Sum =	1101111

There is no end carry. Therefore, the answer is Y – X = – (2's complement of 1101111) = – 0010001.

Subtraction Using r-1's Complement: -

Subtraction using r-1's complement (Radix-minus-one's complement) is a method used in digital systems to perform subtraction by using addition.

- In r-1's complement subtraction, add r-1's complement of subtrahend to the minuend.
- If there is a carry out, add this carry out to the result obtained in the previous step. This is known as an end-around carry.
- If there is no carry ,the result is negative and we need to take the r-1's complement of this result to get the final answer. The answer will have a negative sign.

Example 4

Using 9's complement, subtract 72532 – 3250.

	M =	72532
9's complement of	N =	+96749
	Sum =	169281
	Add the carry 1 =	+ 1
	Answer =	69282

Example 5

Using 9's complement, subtract 3250 – 72532

	M =	03250
9's complement of	N =	+ 27467
	Sum =	30717

There is no end carry.

Therefore, the answer is – (9's complement of 30717) = – 69282.

Example 6

Given the two binary numbers X = 1010100 and Y = 1000011, perform the subtraction X – Y ; and (b) Y – X, by using 1's complement.

(a) $X - Y = 1010100 - 1000011$

X =	1010100
1's complement of Y =	± 0111100
Sum =	10010000
End-around carry =	+ 1
Answer. $X - Y$ =	0010001

(b) $Y - X = 1000011 - 1010100$

Y =	1000011
1's complement of X =	+ 0101011
Sum =	1101110

There is no end carry, Therefore, the answer is Y – X = – (1's complement of 1101110)

= – 0010001.

Signed Binary Numbers:

Signed binary numbers are used to represent both positive and negative integers in binary form. There are several methods for representing signed binary numbers, with the most common ones being **Sign-Magnitude**, **One's Complement**, and **Two's Complement**.

Sign-Magnitude Representation:

In sign-magnitude representation, the most significant bit (MSB) is used to represent the sign of the number, while the remaining bits represent the magnitude (absolute value) of the number.

- **MSB = 0**: The number is positive.
- **MSB = 1**: The number is negative

Example:

- +5 in 4-bit sign-magnitude: **0101**
- −5 in 4-bit sign-magnitude: **1101**

2. One's Complement Representation:

In one's complement, the negative of a number is represented by inverting all the bits of its positive counterpart (i.e., changing all 0s to 1s and all 1s to 0s).

Example:

- +5 in 4-bit one's complement: 0101
- −5 in 4-bit one's complement: 1010

3. Two's Complement Representation:

Two's complement is the most widely used method for representing signed numbers in binary. To obtain the two's complement of a number, take the one's complement of the number and add 1 to it.

Example:

- +5 in 4-bit two's complement: 0101
- −5 in 4-bit two's complement: 1011

[Note:Add 1 to the 1's complement of -5: 1010 + 1 = 1011]

Example:

Representation of -9 in Sign-Magnitude, One's Complement, and Two's Complement form.

Signed-magnitude representation:	10001001
Signed-1's-complement representation:	11110110
Signed-2's-complement representation:	11110111

Signed Binary Numbers

Decimal	Signed-2's Complement	Signed-1's Complement	Signed Magnitude
+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	—	1111	1000
−1	1111	1110	1001
−2	1110	1101	1010
−3	1101	1100	1011
−4	1100	1011	1100
−5	1011	1010	1101
−6	1010	1001	1110
−7	1001	1000	1111
−8	1000	—	—

UNIT -1
BOOLEAN ALGEBRA AND LOGIC GATES
TOPIC – 8
(BCD CODES)

BINARY CODED DECIMAL CODE(BCD):

- **BCD (Binary-Coded Decimal)** is a method of encoding decimal numbers (0-9) using binary digits. Each decimal digit is represented by a fixed number of four binary bits
- The binary combinations 1010 through 1111 are not used and have no meaning in BCD.

Binary-Coded Decimal (BCD)

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

- A number with k decimal digits will require 4k bits in BCD.
- Decimal 396 is represented in BCD with 12 bits as 0011 1001 0110, with each group of 4 bits representing one decimal digit.

Example 1:

Consider decimal 185 and its corresponding value in BCD and binary:

$$(185)_{10} = (0001\ 1000\ 0101)_{BCD} = (10111001)_2$$

BCD (Binary-Coded Decimal) addition :

- Addition of BCD involves adding two BCD numbers just like binary numbers, with some additional steps to ensure that the result is a valid BCD digit (i.e., within the range 0000 to 1001, or 0 to 9 in decimal).
- If the result exceeds 9 (1001 in binary), a correction is required to make it a valid BCD digit.

Steps for BCD Addition:

1. Add the BCD numbers as if they were binary numbers.
2. Check each 4-bit group (nibble) in the result:

   - If the 4-bit sum is less than or equal to 9 (1001 in binary), no correction is needed.
   - If the 4-bit sum is greater than 9, add 6 (0110 in binary) to the 4-bit sum to correct it.

3.Handle the carry:

   - If adding 6 causes a carry out of the nibble, this carry must be added to the next higher nibble.

Example 2:

4	0100	4	0100	8	1000
+5	+0101	+8	+1000	+9	+1001
9	1001	12	1100	17	10001
			+0110		+0110
			10010		10111

Example 3:

Consider the addition of 184 + 576 = 760 in BCD:

BCD	1	1		
	0001	1000	0100	184
	+0101	0111	0110	+576
Binary sum	0111	10000	1010	
Add 6		0110	0110	
BCD sum	0111	0110	0000	760

Other BCD Codes:

- In addition to the standard **BCD (Binary-Coded Decimal)**, which represents each decimal digit (0-9) as a 4-bit binary number, there are several other types of BCD codes used in digital systems.

Four Different Binary Codes for the Decimal Digits

Decimal Digit	BCD 8421	2421	Excess-3	8, 4, −2, −1
0	0000	0000	0011	0000
1	0001	0001	0100	0111
2	0010	0010	0101	0110
3	0011	0011	0110	0101
4	0100	0100	0111	0100
5	0101	1011	1000	1011
6	0110	1100	1001	1010
7	0111	1101	1010	1001
8	1000	1110	1011	1000
9	1001	1111	1100	1111
	1010	0101	0000	0001
Unused	1011	0110	0001	0010
bit	1100	0111	0010	0011
combi-	1101	1000	1101	1100
nations	1110	1001	1110	1101
	1111	1010	1111	1110

1. 8421 BCD Code (Natural BCD)

- This is the standard BCD code and is sometimes referred to as **8421 BCD** because each bit represents a power of 2.
- Each decimal digit is encoded into a 4-bit binary number.

Example:

- Decimal 7 in 8421 BCD: 0111
- Decimal 3 in 8421 BCD: 0011

2. 2421 BCD Code

- In this code, the weights of the bits are 2, 4, 2, and 1, respectively.
- The 2421 code is a self-complementary code, meaning that the complement of a number is automatically generated by inverting the bits.

Example:

- Decimal 5 in 2421 BCD: 1011
- Decimal 3 in 2421 BCD: 0011

3. Excess-3 Code (XS-3):

- Excess-3 is a non-weighted code and is obtained by adding 3 to each decimal digit and then converting the result to binary.
- This code is used in some digital systems because it helps in error detection and simplifies certain types of arithmetic operations.

Example:

- Decimal 2 in Excess-3: 2+3=5, BCD: 0101
- Decimal 7 in Excess-3: 7+3=10, BCD: 1010

4.Gray Code:

- Gray code is a binary numeral system where two successive values differ in only one bit, which minimizes errors during transitions.
- Gray code sequences are often cyclic, means that the sequence wraps around from the last number back to the first.
- Sequence of Gray code is derived by reflecting and prefixing the existing sequence. This method ensures that each Gray code in the sequence differs from the previous one by only one bit.

Reflective Code Method:

- Start with the base Gray code sequence (e.g., 1-bit Gray code).
- Reflect the sequence: Create a new sequence by reversing the original sequence.
- Prefix the original sequence with 0 and the reflected sequence with 1.
- Combine the two sequences to form the Gray code for the next bit length.

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

UNIT -1
BOOLEAN ALGEBRA AND LOGIC GATES
TOPIC – 9
(GRAY CODE AND ERROR DETECTION CODES)

Gray Code/(Reflective Code):

Gray code is a binary numeral system where two successive values differ in only one bit. It is useful in error correction, digital communications, and reducing errors in analog-to-digital conversion.

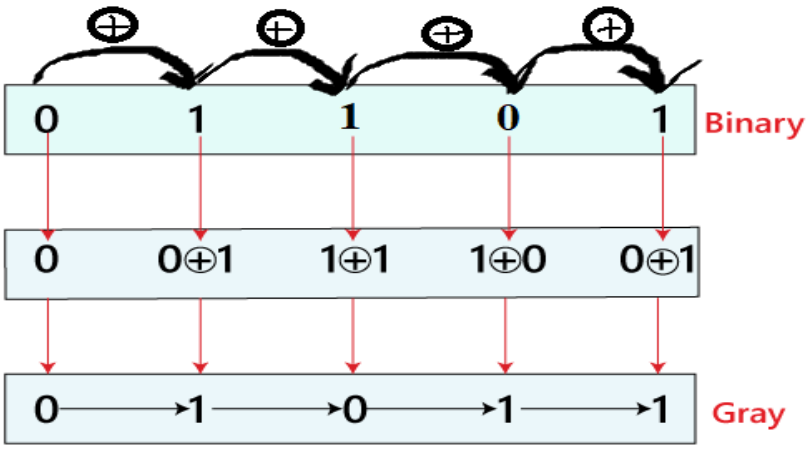
Binary to Gray Code Conversion:

To convert a binary number to Gray code, follow these steps:

1. The most significant bit (MSB) of the Gray code is the same as the MSB of the binary number.
2. Each subsequent Gray code bit is obtained by XORing the corresponding binary bit with the previous binary bit.
3. It means that if both the bits are different, the result will be one(1) else the result will be 0.

Example 1:

Find the gray code of the binary number 01101



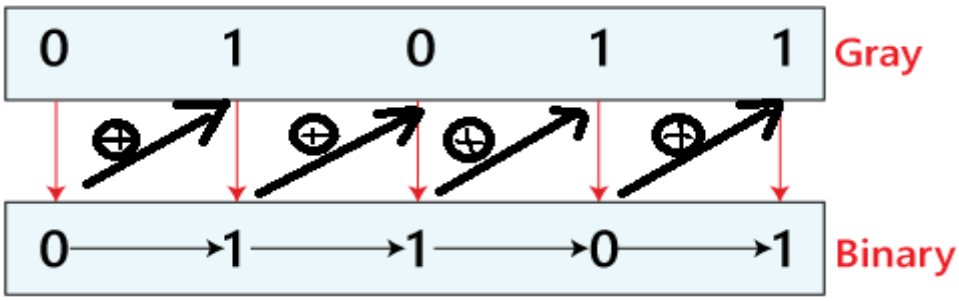
So, the Gray code equivalent of binary $(01101)_2$ is 01011.

Gray Code to Binary Conversion:

To convert a Gray code number back to binary, follow these steps:

1. The MSB of the binary number is the same as the MSB of the Gray code.
2. Each subsequent binary bit is obtained by XORing the previous binary bit with the corresponding Gray code bit.

Example 2:Convert Gray Code 01011 in to Binary



So, the Binary equivalent of Gray code 01011 is $(01101)_2$.

Error Detecting Codes

- Error detecting codes are essential in digital communication systems to ensure data integrity.
- They help us to identify errors that might occur during data transmission over noisy channels or storage in unreliable memory.
- These codes add extra bits to the original data to detect any discrepancies that indicate the presence of an error.

Parity Bit

- The parity bit method is an error-detection technique used in digital communication and storage to ensure data integrity.
- It works by adding an extra bit, called the parity bit, to a string of binary data.
- This bit helps identify whether an error has occurred during the transmission or storage of the data.The parity bit is one of the simplest error-detecting codes.
- A single bit is added to the data to make the number of 1's in the binary sequence either even (even parity) or odd (odd parity).

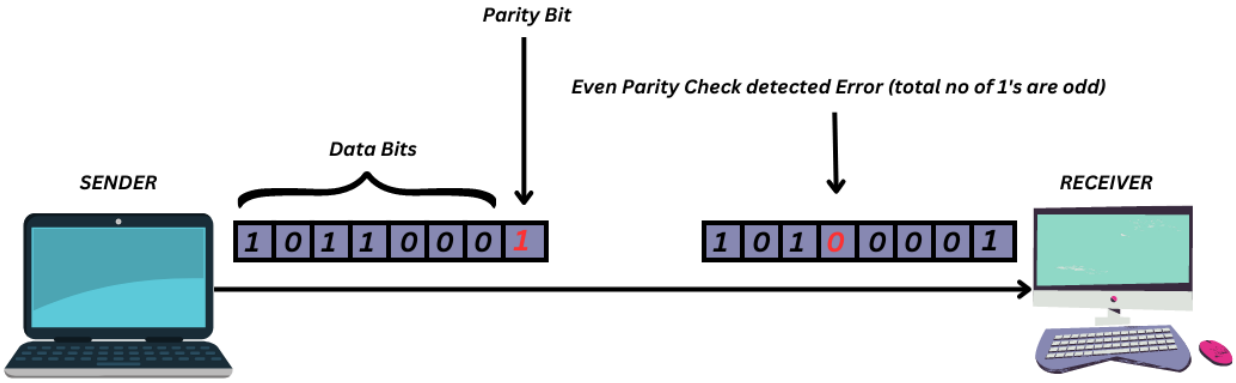
4-bit Message	Even Parity Bit	Message with Even Parity	Odd Parity Bit	Message with Odd Parity
0000	0	00000	1	00001
0001	1	00011	0	00010
0010	1	00101	0	00100
0011	0	00110	1	00111
0100	1	01001	0	01000
0101	0	01010	1	01011
0110	0	01100	1	01101
0111	1	01111	0	01110
1000	1	10001	0	10000
1001	0	10010	1	10011
1010	0	10100	1	10101
1011	1	10111	0	10110
1100	0	11000	1	11001
1101	1	11011	0	11010
1110	1	11101	0	11100
1111	0	11110	1	11111

- **Even Parity**: The parity bit is set to 1 if the number of 1s in the data is odd, making the total count of 1s even.
- **Odd Parity**: The parity bit is set to 1 if the number of 1s in the data is even, making the total count of 1s odd.

Example :

Message A: 10001001**1** (even parity)

Message B: 10001001**0** (odd parity)



Error Detection:

- During transmission, if a single bit error occurs, the parity of the received data will not match the expected parity (even or odd), indicating an error..
- Hence, if any error occurs, the parity check circuit will detect it at the receiver's end.
- However, the parity bit method can only detect an odd number of bit errors. If an even number of bits are flipped, the parity will still match, leading to undetected errors.

Limitations :

- Only errors in a single bit would be identified and we cannot determine the exact location of error where it is occurred.
- If the number of bits in even parity check increase or decrease (data changed) but remained to be even then it won't be able to detect the error as the number of bits are still even and same goes for odd parity check.

UNIT -1
BOOLEAN ALGEBRA AND LOGIC GATES
TOPIC – 10
(ERROR DETECTION AND CORRECTION CODES)

Error detection and correction codes(Hamming Code):

- Hamming code is an error-detection and error-correction method that is used in digital communication and data storage.
- It is designed to detect and correct single-bit errors in data, ensuring that the data transmitted or stored is accurate.
- Hamming code uses a block parity mechanism. The data is divided into blocks, and parity is added to each block.
- Hamming code can correct single-bit errors and detect the presence of two-bit errors in a data block.

Hamming Code Structure:

- For a message of **d** data bits, **p** parity bits are added to form a Hamming code.
- The number of parity bits **p** needed is determined by the condition: $\mathbf{2^p \geq d + p + 1}$
- The total length of the Hamming code is **n=d+p**, where **n** is the total number of bits (data bits + parity bits).

Example 1:

- If we wanted to transmit 7 data bits, the formula would be $2^4 \geq 7 + 4 + 1$, so 4 parity bits are required.

Positions of Parity and Data Bits:

- Parity bits are placed at positions that are powers of 2: 1, 2, 4, 8, etc.(2^0,2^1,2^2,2^3,etc)
- Data bits are placed in the remaining positions.

7-Bit Hamming Code:

- The 7-bit Hamming code is an error-detection and correction code that encodes 4 data bits into 7 bits by adding 3 parity bits.
- Bits 1, 2, 4 are the parity bits.
- Bits 3, 5, 6, 7 are the data bits.

Data/Parity	111 Data 7	110 Data 6	101 Data 5	100 Parity 4	011 Data 3	010 Parity 2	001 Parity 1
	D1	**D2**	D3	P4	D4	P2	P1

- The most significant bit of data is on the left and least significant on the right.
- Find the total block length with both data bits and parity bits is 7(111).

Calculation of Parity Bits:

- Each parity bit $\mathbf{P_i}$ is calculated based on specific positions in the codeword.
- $\mathbf{P_1}$ checks positions 1, 3, 5, 7.
- $\mathbf{P_2}$ checks positions 2, 3, 6, 7.
- $\mathbf{P_3}$ checks positions 4, 5, 6, 7.

- The parity bit is set to ensure that the total number of 1s in the positions it checks is even.
- Set the parity bit equal to 1 if all the values of its data bits are odd; set it to 0 if the total number is even.

Encoding Process of 7-Bit Hamming Code:

Example 1 :

Encode the decimal value $(12)_{10}$ (binary 1100) in Hamming code.

Data/Parity	Data 7	Data 6	Data 5	Parity 4	Data 3	Parity 2	Parity 1
	1	1	0	X	0	X	X

Calculate Parity Bits:

- P_1 checks bits 1, 3, 5, 7 → P_1=?,0,0,1 =P_1=1 (since position 7 contain 1,odd number).
- P_2 checks bits 2, 3, 6, 7 → P_2=?,0,1,1=P_2=0 (since positions 6, 7 contain 1s, even number).
- P_4 checks bits 4, 5, 6, 7 → P_4=?,0,1,1= P_4=0 (since positions 6, 7 contain 1s, even number)

Data/Parity	Data 7	Data 6	Data 5	Parity 4	Data 3	Parity 2	Parity 1
	1	1	0	0	0	0	1

Final Hamming Code:

- The encoded 7-bit Hamming code is 1100001.

Error Detection and Correction:

- When a Hamming code is received, the receiver recalculates the parity bits and compares them to the received parity bits.
- If there's a mismatch, the binary sum of the positions of the incorrect parity bits indicates the position of the error.
- This allows for the correction of a single-bit error.

Example of Error Correction:

Suppose the code 1100001 is received as 1000001:

Recalculate Parity Bits(P_1,P_2,P_4):

1. C_1= 1 $\oplus$0 $\oplus$0 $\oplus$1= 0 (No error in P1's positions)
2. C_2= 0 $\oplus$0 $\oplus$0 $\oplus$1= 1 (Error in P2's positions)
3. C_3= 0 $\oplus$0 $\oplus$0 $\oplus$1= 1 (Error in P4's positions)

- The resulting error position is $\mathbf{C_3\ C_2\ C_1 = (110)_2 = (6)_{10}}$, indicating an error in bit 6.
- Correcting bit 6 will fix the error, resulting in the correct code 1100001

Error Location:

- The incorrect parity bits are P_2 and P_4, which corresponds to the binary sum $\mathbf{(110)_2}$ or position 6.
- Flip the bit at position 6, correcting the code to 1100001, which is the correct original code.

UNIT -1
BOOLEAN ALGEBRA AND LOGIC GATES
TOPIC – 11
(BOOLEAN ALGEBRA)

- Switching circuits are also called logic circuits, gates circuits and digital circuits.
- Switching algebra is also called Boolean algebra.
- Boolean algebra is a system of mathematical logic. It is an algebraic system consisting of the set of elements (0,1), two binary operators called OR and AND and unary operator called NOT.
- It is the basic mathematical tool in the analysis and synthesis of switching circuits.
- It is a way to express logic functions algebraically.
- Any complex logic can be expressed by a Boolean function.
- The Boolean algebra is governed by certain well developed rules and laws.

AXIOMS AND LAWS OF BOOLEAN ALGEBRA:-

Axioms or postulates of Boolean algebra are set of logical expressions that are accepted without proof and upon which we can build a set of useful theorems. Actually, axioms are nothing more than the definitions of the three basic logic operations AND, OR and INVERTER. Each axiom can be interpreted as the outcome of an operation performed by a logic gate.

AND operation	**OR operation**	**NOT operation**
Axiom 1: 0 . 0 = 0	Axiom 5: 0 + 0 = 0	Axiom 9: 1 = 0
Axiom 2: 0 . 1 = 0	Axiom 6: 0 + 1 = 1	Axiom 10:0 = 1
Axiom 3: 1 . 0 = 0	Axiom 7: 1 + 0 = 1	
Axiom 2: 1 . 1 = 1	Axiom 8: 1 + 1 = 1	

1. Complementation Laws:-

The term complement simply means to invert, i.e. to changes 0s to 1s and 1s to 0s. The five laws of complementation are as follows:

Law 1: ~ 0= 1
Law 2: ~1 = 0
Law 3: if A = 0, then ~A = 1
Law 4: if A = 1,then ~A = 0

Law 5: $\overline{\overline{A}} = A$ (double complementation law)

2.OR Laws:-

The four OR laws are as follows

Law 1: A + 0 = 0(Null law)
Law 2: A + 1 = 1(Identity law)
Law 3: A + A = A
Law 4: $A + \overline{A} = 1$

3. AND Laws:-

The four AND laws are as follows

Law 1: A . 0 = 0(Null law)

Law 2: A . 1 = 1(Identity law)

Law 3: A . A = A

Law 4: A .$\overline{A}$ = 0

4. Commutative Laws:-

Commutative laws allow change in position of AND or OR variables. There are two commutative laws.

Law 1: A + B = B + A

Proof:

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

=

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

Law 2: A . B = B . A

Proof:

A	B	A . B
0	0	0
0	1	0
1	0	0
1	1	1

=

B	A	B. A
0	0	0
0	1	0
1	0	0
1	1	1

This law can be extended to any number of variables.

For example

A.B. C = B. C. A = C. A. B = B. A. C

5. Associative Laws:-

The associative laws allow grouping of variables. There are 2 associative laws.

Law 1: (A + B) + C = A + (B + C)

Proof

A	B	C	A+B	(A+B)+C
0	0	0	0	0
0	0	1	0	1
0	1	0	1	1
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

A	B	C	B+C	A+(B+C)
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	1
1	0	0	0	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

=

Law 2: (A .B) C = A (B .C)

Proof

A	B	C	AB	(AB)C
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	0	0
1	1	0	1	0
1	1	1	1	1

=

A	B	C	B.C	A(B.C)
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	0
1	0	0	0	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

This law can be extended to any number of variables. For example

A(BCD) = (ABC)D = (AB) (CD)

6. Distributive Laws:-

The distributive laws allow factoring or multiplying out of expressions. There are two distributive laws.

Law 1: A (B + C) = AB + AC

Proof

A	B	C	B+C	A(B+C)
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	0	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

A	B	C	AB	AC	AB+AC
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	0	1	1
1	1	0	1	0	1
1	1	1	1	1	1

Law 2: A + BC = (A+B) (A+C)

Proof RHS = (A+B) (A+C)

= AA + AC + BA + BC

= A + AC + AB + BC

= A (1+ C + B) + BC

= A. 1 + BC (1 +C + B = 1 + B = 1)
= A + BC
= **LHS**

7. Redundant Literal Rule (RLR):-

Law 1: $A + \overline{A}B = A + B$

Proof $A + \overline{A}B = (A + \overline{A})(A + B)$
$= 1. (A + B)$
$= A + B$

Law 2: $A(\overline{A} + B) = AB$

Proof $A(\overline{A} + B) = A\overline{A} + AB$
$= 0 + AB$
$= AB$

8. Idempotence Laws:-

Idempotence means same value.

Law 1: A**.** A = A

Proof

If A = 0, then A**.** A = 0**.** 0 =0 = A

If A = 1, then A**.** A = 1**.** 1 = 1 = A

This law states that AND of a variable with itself is equal to that variable only.

Law 2: A + A = A

Proof

If A = 0, then A + A = 0 + 0 = 0 = A

If A = 1, then A + A = 1 + 1 = 1 = A

This law states that OR of a variable with itself is equal to that variable only.

9. Absorption Laws:-

There are two laws:

Law 1: $A + A \cdot B = A$

Proof

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

A	B	AB	A+AB
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

A	B	A+B	A(A+B)
0	0	0	0
0	1	1	0
1	0	1	1
1	1	1	1

Law 2: $A(A + B) = A$

Proof

$A(A + B) = A \cdot A + A \cdot B = A + AB = A(1 + B) = A \cdot 1 = A$

10. De Morgan's Theorem:-

De Morgan's theorem represents two laws in Boolean algebra.

Law 1: $\overline{A + B} = \bar{A} \cdot \bar{B}$

Proof

A	B	A + B	$\overline{A + B}$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

A	B	$\bar{A}$	$\bar{B}$	$\bar{A} . \bar{B}$
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	0

This law states that the complement of a sum of variables is equal to the product of their individual complentements.

Law 2: $\overline{A \cdot B} = \bar{A} + \bar{B}$

Proof

A	B	A· B	$\overline{A . B}$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

=

A	B	$\bar{A}$	$\bar{B}$	$\bar{A} + \bar{B}$
0	0	1	1	1
0	1	1	0	1
1	0	0	1	1
1	1	0	0	0

This law states that the complement of a product of variables is equal to the sum of their individual complements.

DUALITY:-

The implication of the duality concept is that once a theorem or statement is proved, the dual also thus standproved. This is called the principle of duality.

$[f(A, B, C, \ldots, 0, 1, +, \cdot)]_d = f(A, B, C, \ldots, 1, 0, \cdot, +)$

A Boolean expression E(A,B,C,…), its **dual** is the expression E′(A,B,C,…) obtained by:

- Replacing all AND (**.**) operators with OR (**+**) operators.
- Replacing all OR (**+**) operators with AND (**.**) operators.
- Replacing all occurrences of 1 (true) with 0 (false).
- Replacing all occurrences of 0 (false) with 1 (true).

Given expression	*Dual*
1. $0 = 1$	$1 = 0$
2. $0 \cdot 1 = 0$	$1 + 0 = 1$
3. $0 \cdot 0 = 0$	$1 + 1 = 1$
4. $1 \cdot 1 = 1$	$0 + 0 = 0$
5. $A \cdot 0 = 0$	$A + 1 = 1$
6. $A \cdot 1 = A$	$A + 0 = A$
7. $A \cdot A = A$	$A + A = A$
8. $A \cdot \overline{A} = 0$	$A + \overline{A} = 1$
9. $A \cdot B = B \cdot A$	$A + B = B + A$
10. $A \cdot (B \cdot C) = (A \cdot B) \cdot C$	$A + (B + C) = (A + B) + C$
11. $A \cdot (B + C) = AB + AC$	$A + BC = (A + B) (A + C)$
12. $A(A + B) = A$	$A + AB = A$
13. $A \cdot (A \cdot B) = A \cdot B$	$A + A + B = A + B$
14. $AB = A + B$	$A + B = A B$
15. $(A + B) (A + C) (B + C) = (A + B)(A + C)$	$AB + AC + BC = AB + AC$
16. $A + BC = (A + B)(A + C)$	$A(B + C) = A B + A C$
17. $(A+C)(A+B) = AB+AC$	$AC+AB=(A+B) (A+C)$
18. $(A+B)(C+D) = AC + AD + BC + BD$	$(AB+CD) = (A+C)(A+D)(B+C)(B+D)$
19. $A + B = AB +$ $AB + AB$	$AB =(A+B) (A+B)$ $(A+B)$
20. $AB + A + AB = 0$	$(A + B) \cdot A \cdot (A + B) = 1$

UNIT -1
BOOLEAN ALGEBRA AND LOGIC GATES
TOPIC – 12
(LOGIC GATES)

- Logic gates are the fundamental building blocks of digital systems.
- There are 3 basic types of gates AND, OR and NOT.
- Logic gates 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 2 levels. These two levels are termed HIGH and LOW, or TRUE and FALSE, or ON and OFF or simply 1 and 0.
- The table which lists all the possible combinations of input variables and the corresponding outputs is called a **Truth table**.

LEVEL LOGIC:-

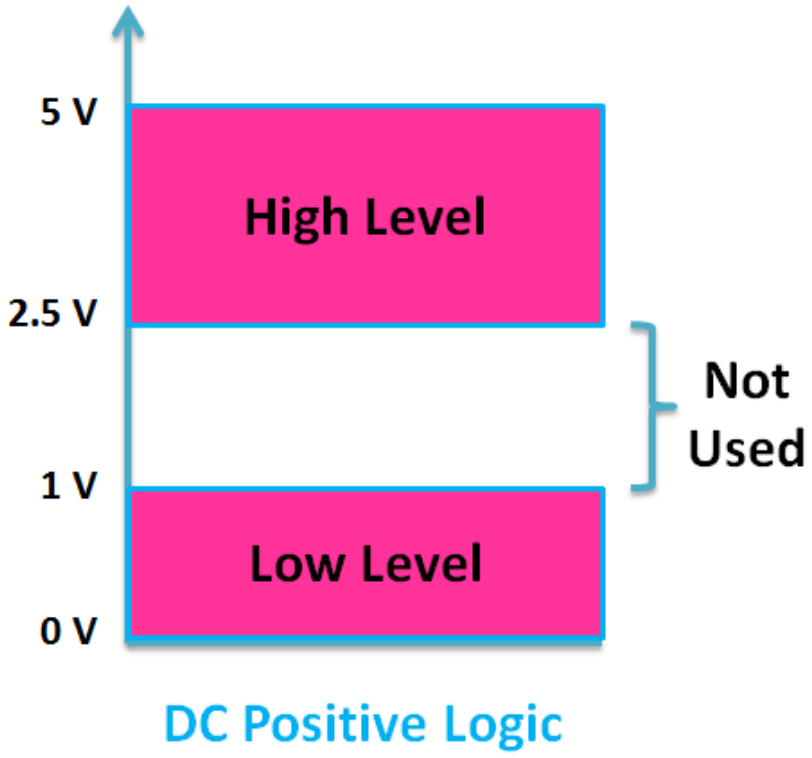
- A logic in which the voltage levels represents logic 1 and logic 0. Level logic may be positive or negative logic.

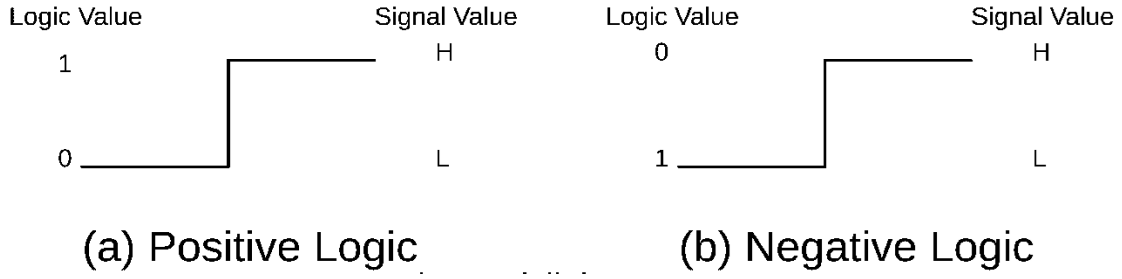
Positive Logic:-

- A positive logic system is the one in which the higher of the two voltage levels represents the logic 1 and thelower of the two voltages level represents the logic 0.

Negative Logic:-

- A negative logic system is the one in which the lower of the two voltage levels represents the logic 1 and thehigher of the two voltages level represents the logic 0.





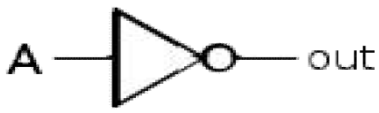
DIFFERENT TYPES OF LOGIC GATES:-

NOT GATE (INVERTER):-

- A NOT gate, also called as an inverter, has only one input and one output.
- It is a device whose output is always the complement of its input.
- The output of a NOT gate is the logic 1 state when its input is in logic 0 state and the logic 0 state when its inputs is in logic 1 state.

IC No. :- 7404

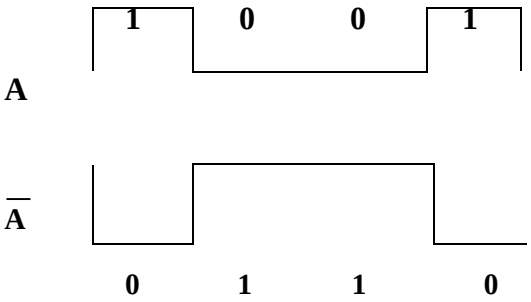
Logic Symbol



Truth table

z	OUTPUT A
0	**1**
1	0

Timing Diagram

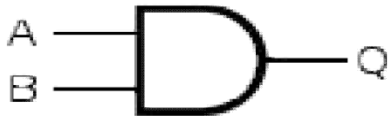


AND GATE:-

- An AND gate has two or more inputs but only one output.
- The output is logic 1 state only when each one of its inputs is at logic 1 state.
- The output is logic 0 state even if one of its inputs is at logic 0 state.

IC No.:- 7408

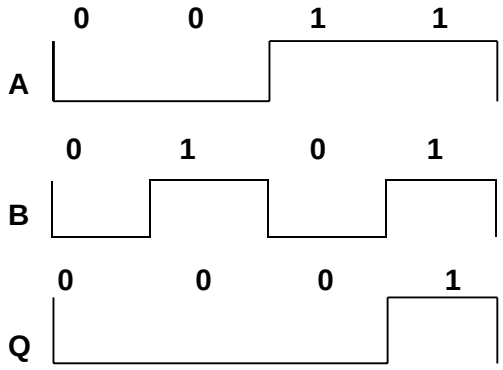
Logic Symbol



Truth Table

INPUT		OUTPUT
A	B	Q=A . B
0	0	0
0	1	0
1	0	0
1	1	1

Timing Diagram



OR GATE:-

- An OR gate may have two or more inputs but only one output.
- The output is logic 1 state, even if one of its input is in logic 1 state.
- The output is logic 0 state, only when each one of its inputs is in logic state.

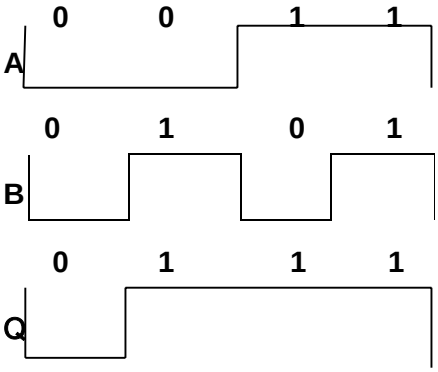
IC No.:- 7432

Logic Symbol

Truth Table

INPUT		OUTPUT
A	B	Q=A + B
0	0	0
0	1	1
1	0	1
1	1	1

Timing Diagram



UNIVERSAL GATES:-

There are two universal gates NAND and NOR, each of which can realize logic circuits. The NAND and NOR gates are called universal building blocks. Both NAND and NOR gates can perform all logic functions i.e. AND, OR, NOT, EXOR and EXNOR.

NAND GATE:-

- NAND gate is a combination of an AND gate and a NOT gate.
- The output is logic 0 when each of the input is logic 1 and for any other combination of inputs, the output is logic 1.

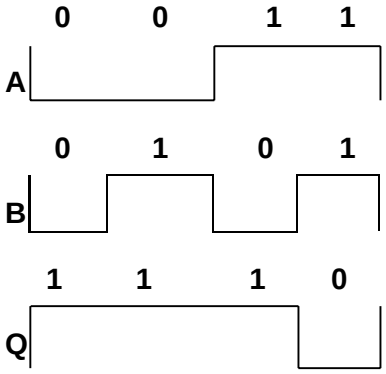
IC No.:- 7400 two input NAND gate
7410 three input NAND gate
7420 four input NAND gate
7430 eight input NAND gate

Logic Symbol

Truth Table

INPUT		OUTPUT
A	B	$Q=\overline{A \cdot B}$
0	0	1
0	1	1
1	0	1
1	1	0

Timing Diagram



NOR GATE:-

- NOR gate is a combination of an OR gate and a NOT gate.
- The output is logic 1, only when each one of its input is logic 0 and for any other combination of inputs, the output is a logic 0 level.

IC No.:- 7402 two input NOR gate

7427 three input NOR gate

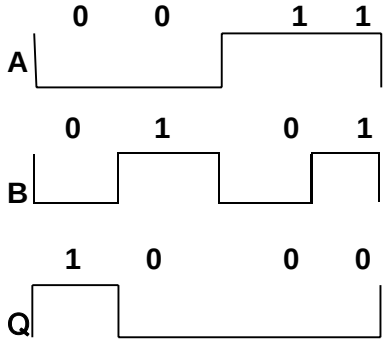
7425 four input NOR gate

Logic Symbol

Truth Table

INPUT		OUTPUT
A	B	$Q=\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

Timing Diagram



EXCLUSIVE – OR (X-OR) GATE:-

- An X-OR gate is a two input, one output logic circuit.
- The output is logic 1 when one and only one of its two inputs is logic 1. When both the inputs is logic 0 or when both the inputs is logic 1, the output is logic 0.

IC No.:- 7486

Logic Symbol

Truth Table

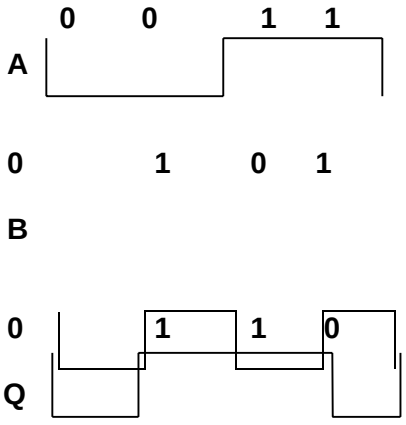
INPUT		OUTPUT
A	B	$Q = A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

INPUTS are **A** and **B**

OUTPUT is **Q** = $A \oplus B$

$= A^{l}B+AB^{l}$

Timing Diagram

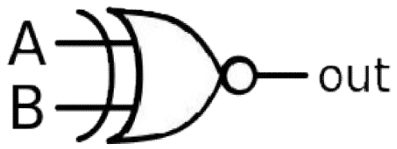


EXCLUSIVE – NOR (X-NOR) GATE:-

- An X-NOR gate is the combination of an X-OR gate and a NOT gate.
- An X-NOR gate is a two input, one output logic circuit.
- The output is logic 1 only when both the inputs are logic 0 or when both the inputs is 1.
- The output is logic 0 when one of the inputs is logic 0 and other is 1.

IC No.:- 74266

Logic Symbol

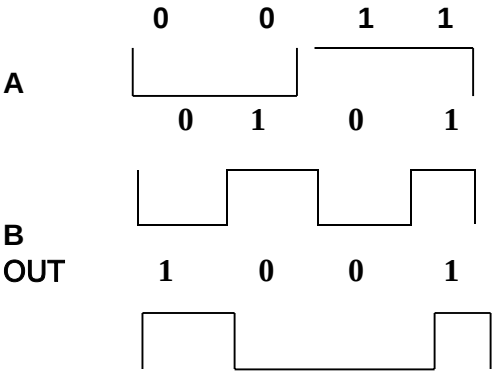


$$Y=A \odot B=A'B'+AB$$

Truth Table

INPUT		OUTPUT
A	B	$Y=A \odot B$
0	0	1
0	1	0
1	0	0
1	1	1

Timing Diagram



Binary Logic:

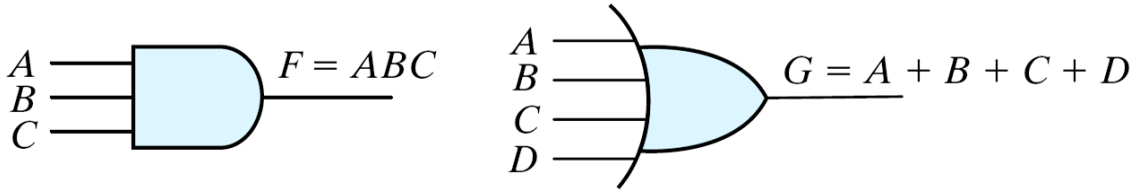
Binary logic is used to describe in mathematical way to manipulate and process the binary information.It is particularly used to analyze the design of digital system.

Binary logic consists of Binary variables and logical operations.

AND and OR gates may have more than two inputs.The three input AND gate give logic 1 out put if all three inputs are logic 1.Output give logic 0 if any input is logic 0.

The four input OR gate responds with logic-1 if any of it's input is logic-1and gives the output zero if all inputs are zero.

Binary Logic is known as Boolean Algebra,which is used to describe the operations of complex networks of digital circuits.



(a) Three-input AND gate (b) Four-input OR gate

Applications of Logic Gates:

The following are some of the applications of Logic gates:

- Build complex systems that can be used in different fields.
- Construct multiplexers, adders and multipliers.
- Perform several parallel logical operations
- Used in the circuit of automated machine manufacturing industry

UNIT -1
BOOLEAN ALGEBRA AND LOGIC GATES
TOPIC – 13
(OTHER LOGIC OPERATIONS)

Other Logic operations:

- In a system with N input signals, there are 2^{2^N} possible logic functions. So, a two-input logic system has 16 possible logic functions, a three input logic system has 256 possible logic functions, etc.
- The extended truth table below shows the 16 possible logic functions of two variables.
- Many of them have common names (AND, OR, NAND, etc.), and in fact only four possible functions (F2, F4, F11, and F13) are not associated with common relationships.
- Here, x and y are the variables or registers in which the data is stored and F0, F1, ….., F15 are the outputs that occur after performing these logic micro-operations.

x y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0 0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0 1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1 0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1 1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Boolean Expressions for the 16 Functions of Two Variables

Boolean functions	Operator symbol	Name	Comments
$F_0 = 0$		Null	Binary constant 0
$F_1 = xy$	$x \cdot y$	AND	x and y
$F_2 = xy'$	x/y	Inhibition	x but not y
$F_3 = x$		Transfer	x
$F_4 = x'y$	y/x	Inhibition	y but not x
$F_5 = y$		Transfer	y
$F_6 = xy' + x'y$	$x \oplus y$	Exclusive-OR	x or y but not both
$F_7 = x + y$	$x + y$	OR	x or y
$F_8 = (x + y)'$	$x \downarrow y$	NOR	Not-OR
$F_9 = xy + x'y'$	$x \odot y$	Equivalence	x equals y
$F_{10} = y'$	y'	Complement	Not y
$F_{11} = x + y'$	$x \subset y$	Implication	If y then x
$F_{12} = x'$	x'	Complement	Not x
$F_{13} = x' + y$	$x \supset y$	Implication	If x then y
$F_{14} = (xy)'$	$x \uparrow y$	NAND	Not-AND
$F_{15} = 1$		Identity	Binary constant 1

F0 =Logic 0

X	Y	F_0
0	0	0
0	1	0
1	0	0
1	1	0

F1 =X.Y

X	Y	F_1
0	0	0
0	1	0
1	0	0
1	1	1

F2 =X.Y'

X	Y	F_2
0	0	0
0	1	0
1	0	1
1	1	0

F3 =XY'+XY
=X(Y'+Y)
=X(1)
=X

X	Y	F_3
0	0	0
0	1	0
1	0	1
1	1	1

F4 =X'.Y

X	Y	F_4
0	0	0
0	1	1
1	0	0
1	1	0

F5 =X'Y+XY
=Y(X'+X)
=Y(1)
=Y

X	Y	F_5
0	0	0
0	1	1
1	0	0
1	1	1

F6 =X'Y+XY'

X	Y	F_6
0	0	0
0	1	1
1	0	1
1	1	0

F7 =X'Y+XY'+XY
=Y(X'+X)+XY'
=Y+XY'
=(X+Y)(Y'+Y)
=(X+Y)

X	Y	F_7
0	0	0
0	1	1
1	0	1
1	1	1

F8 **=X'.Y'**
=(X+Y)'

X	Y	F_8
0	0	1
0	1	0
1	0	0
1	1	0

F9 **=X'Y'+XY**
=X $\odot$Y
=$\overline{X \oplus Y}$

X	Y	F_9
0	0	1
0	1	0
1	0	0
1	1	1

F10 **=X'Y'+XY'**
=Y'(X'+X)
=Y'(1)
=Y'

X	Y	F_{10}
0	0	1
0	1	0
1	0	1
1	1	0

F11 **=X'Y'+XY'+XY**
=Y'(X'+X)+XY
=Y'+XY
=(X+Y')(Y+Y')
=(X+Y')(1)
=(X+Y')

X	Y	F_{11}
0	0	1
0	1	0
1	0	1
1	1	1

F12 **=X'Y'+X'Y**
=X'(Y'+Y)
=X'(1)
=X'

X	Y	F_{12}
0	0	1
0	1	1
1	0	0
1	1	0

F13 **=X'Y'+X'Y+XY**
=X'(Y'+Y)+XY
=X'(1)+XY
=X'+XY
=(X'+X)(X'+Y)
=(1)(X'+Y)=(X'+Y)

X	Y	F_{13}
0	0	1
0	1	1
1	0	0
1	1	1

F14 **=X'Y'+X'Y+XY'**
=X'(Y'+Y)+XY'
=X'(1)+XY
=X'+XY'
=(X'+X)(X'+Y')
=(1)(X'+Y')
=(X'+Y')
=(XY)'

X	Y	F_{14}
0	0	1
0	1	1
1	0	1
1	1	0

F15 **=X'Y'+X'Y+XY'+XY**
=X'(Y'+Y)+X(Y'+Y)
=X'(1)+X(1)
=X'+X
=Logic 1

X	Y	F_{15}
0	0	1
0	1	1
1	0	1
1	1	1

UNIT -1
BOOLEAN ALGEBRA AND LOGIC GATES
TOPIC – 14
(MINIMIZATION OF BOOLEAN FUNCTIONS)

The process of simplifying the algebraic expression of a boolean function is called minimization. Minimization is important since it reduces the cost and complexity of the associated circuit.

Procedure of Simplifying Boolean Expression using Boolean Algebra

Step 1 − Multiply all the variables required to remove parentheses '()'.

Step 2 − Find all the identical terms in the expression. Only one of those terms will be retained and all other are dropped.

For example −ABC+ABC+ABC=ABC

Step 3 − Find a variable and its negation in the same term. This term will be also dropped.

For example −AB$\overline{B}$C=A.0.C=AC

Step 4 − Identify pairs of terms which are identical except for one variable that may be missing in one of the terms. In this case, the larger term will be dropped.

For example −ABC+AB=AB(C+1)=AB.1=AB

Step 5 − Identify those pairs of terms that have the same variables with one or more variables complemented. If a variable in one term of such a pair is complemented and in the second term it is not complemented. Then, such terms are combined into a single term with that variable dropped.

For example −AB$\overline{C}$+ABC=AB($\overline{C}$ + C)=AB.1=AB

Example 1:

F1 =X'Y'Z+X'YZ+XY'
=X'Z(Y'+Y)+XY'
=X'Z+XY'

Example 2:

F2 =A+A'B
=(A+A')(A+B)
=1.(A+B)
=A+B

Example 3:

F3 = A+AB
=A(1+B)
=A.1
=A

Example 4:

Simplify the expression: P'(P + QR) + Q'R.
=P'P+P'QR+Q'R
=0+R(P'Q+Q')
=R((Q+Q')(P'+Q'))
=R(P'+Q')
=P'R+RQ'

Example 5:

Find the simplified term: A' (M' + A') (M + M'A)
=A'(M'+A')((M+M')(M+A))
=A'(M'+A')(M+A)
=A'(M'M+M'A+MA'+A'A)
=A'(0+M'A+MA'+0)
=A'AM+MA'A'=0+MA'=MA'

Example 6:
F = ABC'D' + ABC'D + AB'C'D + ABCD + AB'CD + ABCD' + AB'CD'
F = ABC' (D' + D) + AB'C'D + ACD (B + B') + ACD' (B + B')
= ABC' + AB'C'D + ACD + ACD'
= ABC' + AB'C'D + AC (D + D')
= ABC' + AB'C'D + AC
= A (BC' + C) + AB'C'D
= A (B +C) + AB'C'D
= AB + AC + AB'C'D
= AB + AC + AC'D
= AB + AC + AD

Example 7: Simplify the Boolean function F=AB+ BC + B′C.
Solution. F = AB + BC + B′C
= AB + C(B + B′)
= AB + C

Example 8: Simplify the Boolean function F= A + A′B.
Solution. F = A+ A′B
= (A + A′) (A + B)
= A + B

Example 9: Simplify the Boolean function F= A′B′C + A′BC + AB′.
Solution. F = A′B′C + A′BC + AB′
= A′C (B′+B) + AB′
= A′C + AB′

Example 10: Simplify the Boolean function F = AB + (AC)′ + AB′C(AB + C).
Solution. F = AB + (AC)′ + AB′C(AB + C)
= AB + A′ + C′+ AB′C.AB + AB′C.C
= AB + A′ + C′ + 0 + AB′C :(B.B′ = 0 and C.C = C)
= ABC + ABC′ + A′ + C′ + AB′C :(AB.1=AB(C+C'))
= AC(B + B′) + C′(AB + 1) + A′ :(AB(C + C′) = ABC + ABC′)
= AC + C′+A′ : (B + B′ = 1 and AB + 1 = 1)
= AC + (AC)′ = 1

Example 11:F1=X'Y+XY'+XY
=Y(X'+X)+XY'
=Y+XY'
=(X+Y)(Y'+Y)
=(X+Y)

Example 12:F(W, X, Y, Z) = W'·X'·Y'·Z' + W'·X'·Y'·Z + W·X'·Y'
= W'·X'·Y'·(Z' + Z) + W·X'·Y'
= W'·X'·Y' + W·X'·Y'
= (W' + W)·X'·Y'
= X'·Y'

UNIT -1
BOOLEAN ALGEBRA AND LOGIC GATES
TOPIC – 15
(STANDARD & CANONICAL FORMS)

The various ways to represent a Boolean Function is:
1.Sum Of Products form(SOP)
2.Product Of Sum form(POS)
3.Standard Sum Of Products form(SSOP)
4.Standard Product Of Sum form(SPOS)

Sum Of Product (SOP):

Sum of Product is the abbreviated form of SOP. Sum of product form is a form of expression in Boolean algebra in which different product terms of inputs are being summed together. This product is not arithmetical multiply but it is Boolean logical AND and the Sum is Boolean logical OR.

To understand better about SOP, we need to know about min term.

Min Term

Minterm is a product term that is true for a minimum number of combinations of inputs. That is true for only one combination of inputs.

Since AND gate also gives True only when all of its inputs are true so we can say min terms are AND of input combinations like in the table given below.

A	B	C	Minterm	Designation
0	0	0	A'B'C'	m_0
0	0	1	A'B'C	m_1
0	1	0	A'BC'	m_2
0	1	1	A'BC	m_3
1	0	0	AB'C'	m_4
1	0	1	AB'C	m_5
1	1	0	ABC'	m_6
1	1	1	ABC	m_7

3 inputs have 8 different combinations. Each combination has a min terms denoted by small m and its decimal combination number written in subscript. Each of these minterms will be only true for the specific input combination.

Types of Sum Of Product (SOP) Forms

There are few different forms of Sum of Product.

- Canonical SOP Form
- Non-Canonical SOP Form
- Minimal SOP Form

Canonical SOP Form

This is the standard form of Sum of Product. It is formed by ORing the minterms of the function for which the output is true. This is also known as Sum of Min terms or Canonical disjunctive normal form (CDNF). It is just a fancy name. “canonical” means “standardized” and “disjunctive” means “Logical OR union”.

Canonical SOP expression is represented by summation sign $\sum$ and minterms in the braces for which the output is true.

For example, a functions truth table is given below.:

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

For this function the canonical SOP expression is

$\mathbf{F = \sum(m_1, m_2, m_3, m_5)}$

Which means that the function is true for the min terms **{1, 2, 3, 5}**.

By expanding the summation we get.

$\mathbf{F = m_1 + m_2 + m_3 + m_5}$

Now putting min terms in the expression

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

Canonical form contains all inputs either complemented or non-complemented in its product terms.

Non-Canonical SOP Form

As the name suggests, this form is the non-standardized form of SOP expressions. The product terms are not the min terms but they are simplified. Let’s take the above function in canonical form as an example.

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

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

$$\mathbf{F = \bar{A}\bar{B}C + \bar{A}B(1) + A\bar{B}C}$$

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

This expression is still in Sum of Product form but it is non-canonical or non-standardized form.

Minimal SOP Form

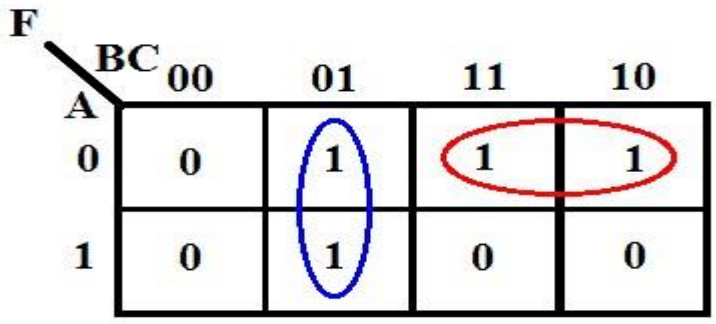
This form is the most simplified SOP expression of a function. It is also a form of non-canonical form. Minimal SOP form can be made using Boolean algebraic theorems but it is very easily made using Karnaugh map (K-map).

Minimal SOP form is preferred because it uses the minimum number of gates and input lines. it is commercially beneficial because of its compact size, fast speed, and low fabrication cost.

Let's take an example of the function given above in canonical form.

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

Its **K-map** is given below:



According to the K-map, the output expression will be

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

This is the most simplified & optimized expression for the said function. This expression requires only two 2-input AND gates & one 2-input OR gate. However, the canonical form needs four 3-input AND gates & one 4-input OR gate, which is relatively more costly than minimal form implementation.

Conversion from Minimal SOP to Canonical SOP Form

Conversion from minimal or any sort of non-canonical form to canonical form is very simple.

As we know canonical form has min terms & min terms consists of all inputs either complemented or non-complemented. So we will multiply every term of minimal SOP with the sum of missing input's complemented and non-complemented form. Example of conversion for the above function in minimal SOP form is given below.

Minimal SOP form

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

The term $\bar{A}B$ is missing input C. So we will multiply $\bar{A}B$ with $(C+\bar{C})$ because $(C+\bar{C} = 1)$. The term $\bar{B}C$ is missing input **A**. so it will be multiplied with $(A+\bar{A})$

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

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

Now, this expression is in canonical form.

Conversion from Canonical SOP to Canonical POS

Standard SOP expression can be converted into standard POS (product of sum) expression. For example, the function given above is in canonical SOP form

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

$$F = \sum (m_1, m_2, m_3, m_5)$$

The remaining terms of this function are maxterms for which output is false. These max terms are M_0,M_4,M_6,M_7. These Max terms will be used in POS expression as the product of these max terms. The Symbol of Product is $\prod$.

$$F = \prod (M_0, M_4, M_6, M_7)$$

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

The Max terms are the complement of minterms. Which is why **M_0=(A+B+C).**

Conversion from Canonical SOP to Minimal SOP

Canonical SOP can be converted to minimal SOP. It can be converted using Karnaugh map or Boolean algebraic theorems. The K-map method is very easy and its example has been done above in the minimal SOP form.

Product of Sum :

Product of Sum abbreviated for POS.

The product of Sum form is a form in which products of different sum terms of inputs are taken. These are not arithmetic product and sum but they are logical Boolean AND and OR respectively.

To better understand about Product of Sum, we need to know about Max term.

Max Term

Maxterm means the term or expression that is true for a maximum number of input combinations or that is false for only one combination of inputs.

Since OR gate also gives false for only one input combination. So Maxterm is OR of either complemented or non-complemented inputs.

Max terms for 3 input variables are given below:

A	B	C	Maxterm	Designation
0	0	0	A+B+C	M_0
0	0	1	A+B+C'	M_1
0	1	0	A+B'+C	M_2
0	1	1	A+B'+C'	M_3
1	0	0	A'+B+C	M_4
1	0	1	A'+B+C'	M_5
1	1	0	A'+B'+C	M_6
1	1	1	A'+B'+C'	M_7

3 inputs have 8 different combinations so it will have 8 maxterms. Maxterms are denoted by capital M and decimal combination number In the subscript as shown in the table given above.

In maxterm, each input is complemented because Maxterm gives '0' only when the mentioned combination is applied and Maxterm is complement of minterm.

M_3 = $\bar{m}_3$

M_3 = $(A\overline{B}\overline{C})'$

M_3 = $\overline{A} + B + C$ DE Morgan's law

Which is why for A=0 Max term consist A & for A=1 Max term consist $\overline{A}$.

Types of Product Of Sum Forms:

There are different types of Product of Sum forms.

- Canonical POS Form
- Non – Canonical Form
- Minimal POS Form

Canonical POS Form

It is also known as Product of Max term or Canonical conjunctive normal form (CCNF). Canonical means standard and conjunctive means intersection.

In this form, Maxterms are AND together for which output is false.

Canonical POS expression is represented by $\prod$ and Maxterms for which output is false in brackets as shown in the example given below.

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

F = ∏ (M_0, M_4, M_6, M_7)

Expanding the product

F = $M_0.M_4.M_6.M_7$

Putting Max terms

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

The canonical form contains all inputs either complemented or non-complemented in its each Sum term.

Non – Canonical Form:

The product of sum expression that is not in standard form is called non-canonical form.

Let's take the above-given function as an example.

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

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

Same but inverted terms eliminates from two Max terms and form a single term to prove it here is an example.

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

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

= 0+AB+AC+$\bar{A}$B+$\bar{A}$C+B+BC+C

= A(B+C)+$\bar{A}$(B+C)+B(1+C)+C

= (B+C)(A+$\bar{A}$)+B(1)+C

= (B+C)(0)+B+C

= B+C The expression achieved is still in Product of Sum form but it is non-canonical form.

Minimal POS Form

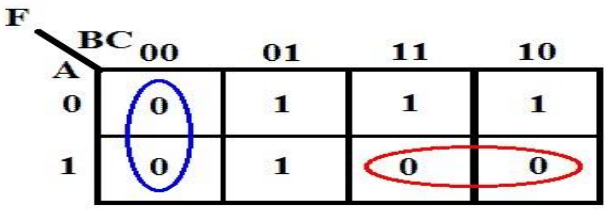
A canonical Product of Sum expression can be converted into Minimal Product of sum form by using Karnaugh map (K-map). Another method for converting canonical into minimal is by using Boolean algebraic theorems.

The use of K-map is very easy that is why K-map is preferred. For minimal POS expression, 0's in K-map are combined into groups.

Minimal POS form uses less number of inputs and logic gates during its implementation, that's why they are being preferred over canonical form for their compact,fast and low-cost implementation.

Let's take the above-given function as example

K-map of the function:



Minimal expression using K-map

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

The achieved expression is the minimal product of sum form. It is still Product of Sum expression But it needs only 2 inputs two OR gates and a single 2 input AND gate. However, the canonical form needs 4 OR gates of 3 inputs and 1 AND gate of 4 inputs.

Conversion from Minimal POS to Canonical form POS

As we know the canonical form of POS has max terms and max terms contains every input either complemented or non-complemented. So we will add every sum term with the product of complemented and non-complemented missing input. Example of its conversion is given below.

Minimal POS form

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

$(\bar{A}+\bar{B})$ term is missing C input so we will add $(C\bar{C})$ with it. (B+C) term is missing A input so we will add $(A\bar{A})$ with it.

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

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

This expression is now in canonical form.

Conversion From Canonical POS to SOP

The product of Sum expression can be converted into Sum of Product form only if the expression is in canonical form. Canonical POS and canonical SOP are inter-convertible i.e. they can be converted into one another. Example of POS to SOP conversion is given below.

POS canonical form

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

In canonical form each sum term is a max term so it can also be written as:

$$F = \prod(M_0,M_4,M_6,M_7)$$

The remaining combinations of inputs are minterms of the function for which its output is true. To convert it into SOP expression first we will change the symbol to summation (∑) and use the remaining minterm.

$$F = \sum (m_1,m_2,m_3,m_5)$$

Now we will expand the summation sign to form canonical SOP expression.

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

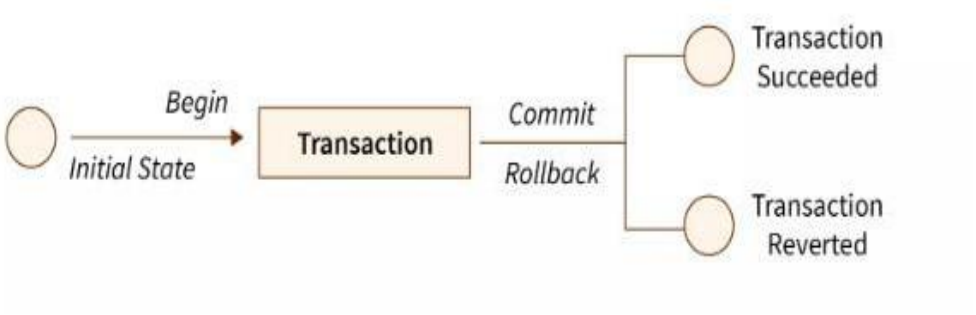
Min terms are complement of Max terms for the same combination of inputs.

DATABASE MANAGEMENT SYSTEMS
Transaction Control Language (TCL) Commands

TCL stands for Transaction Control Language in SQL. Transaction Control Language (TCL) is a set of special commands that deal with the transactions within the database. Basically, they are used to manage transactions within the database. TCL commands ensure the integrity and consistency of data by allowing to control the behavior of transactions.

A transaction is a unit of work that is performed against a database in SQL. In other words, a transaction is a single, indivisible database action.

In SQL, each transaction begins with a particular set of task and ends only when all the tasks in the set is completed successfully. However, if any (or a single) task fails, the transaction is said to fail.



TCL Commands – Save point Commit and Roll back:

COMMIT command:

COMMIT command is used to permanently save any transaction into the database.
When we use any DML command like INSERT, UPDATE or DELETE, the changes made by these commands are not permanent, until the current session is closed, the changes made by these commands can be rolled back.
To avoid that, we use the COMMIT command to mark the changes as permanent.

SYNTAX : COMMIT;

ROLLBACK command :

This command restores the database to last commited state.
If we have used the UPDATE command to make some changes into the database or inserted a record, and realise that those changes were not required, then we can use the ROLLBACK command to rollback those changes, if they were not commited using the COMMIT command.

SYNTAX : ROLLBACK;

It is also used with SAVEPOINT command to jump to a savepoint in an ongoing transaction. The savepoints are like checkpoints, they temporarily save a transaction up to where the transaction can be rolled back

SYNTAX: ROLLBACK TO savepoint_name;

SAVEPOINT command:

SAVEPOINT command is used to temporarily save a transaction so that you can rollback to that point whenever required.

SYNTAX: SAVEPOINT savepoint_name;

In short, using this command we can name the different states of our data in any table and then rollback to that state using the ROLLBACK command whenever required

Example : Using Savepoint and Rollback:

Following is the table class,

id	name
1	Abhi
2	Adam
4	Alex

MySQL, set the autocommit option as shown below

SET AUTOCOMMIT=0;

Let us use some SQL queries on the above table and see the results.

- INSERT INTO class VALUES(5, 'Rahul');
- COMMIT;
- UPDATE class SET name = 'Abhijit' WHERE id = '5';
- SAVEPOINT A;
- INSERT INTO class VALUES(6, 'Chris');
- SAVEPOINT B;
- INSERT INTO class VALUES(7, 'Bravo');
- SAVEPOINT C
- SELECT * FROM class;

The resultant table will look like,

id	Name
1	Abhi
2	Adam

4	Alex
5	Abhijit
6	Chris
7	Bravo

Now let's use the ROLLBACK command to roll back the state of data to the savepoint B.

- ROLLBACK TO B;
- SELECT * FROM class;

The output now – current table would look like

id	Name
1	Abhi
2	Adam
4	Alex
5	Abhijit
6	Chris

Now let's again use the ROLLBACK the state of data to the savepoint A

- ROLLBACK TO A;
- SELECT * FROM class;

Now the table will look like,

id	name
1	Abhi
2	Adam
4	Alex
5	Abhijit