NUMBER SYSTEM AND CODES

INTRODUCTION:-

- The term digital refers to a process that is achieved by using discrete unit.
- In number system there are different symbols and each symbol has an absolute value and also has place value.

RADIX OR BASE:-

The radix or base of a number system is defined as the number of different digits which can occur in each position in the number system.

RADIX POINT :-

The generalized form of a decimal point is known as radix point. In any positional number system the radix point divides the integer and fractional part.

N_r = [Integer part . Fractional part]

↑

Radix point

NUMBER SYSTEM:-

In general a number in a system having base or radix ' r ' can be written as

$$a_n \; a_{n-1} \; a_{n-2} \; \ldots\ldots\ldots\ldots \; a_0 \; . \; a_{-1} \; a_{-2} \; \ldots\ldots\ldots\ldots a_{-m}$$

This will be interpreted as

$$Y = a_n \times r^n + a_{n-1} \times r^{n-1} + a_{n-2} \times r^{n-2} + \ldots\ldots\ldots + a_0 \times r^0 + a_{-1} \times r^{-1} + a_{-2} \times r^{-2} + \ldots\ldots\ldots + a_{-m} \times r^{-m}$$

where Y = value of the entire number

a_n = the value of the n^{th} digit

r = radix

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 NUMBER SYSTEM:-

- The decimal number system contain ten unique symbols 0,1,2,3,4,5,6,7,8 and 9.
- In decimal system 10 symbols are involved, so the base or radix is 10.
- It is a positional weighted system.
- The value attached to the symbol depends on its location with respect to the decimal point.

In general,

$$d_n\ d_{n-1}\ d_{n-2}\ \ldots\ldots\ldots\ d_0\ .\ d_{-1}\ d_{-2}\ \ldots\ldots\ldots d_{-m}$$

is given by

$$(d_n \times 10^n) + (d_{n-1} \times 10^{n-1}) + (d_{n-2} \times 10^{n-2}) + \ldots + (d_0 \times 10^0) + (d_{-1} \times 10^{-1}) + (d_{-2} \times 10^{-2}) + \ldots + (d_{-m} \times 10^{-m})$$

For example:-

$9256.26 = 9 \times 1000 + 2 \times 100 + 5 \times 10 + 6 \times 1 + 2 \times (1/10) + 6 \times (1/100)$

$= 9 \times 10^3 + 2 \times 10^2 + 5 \times 10^1 + 6 \times 10^0 + 2 \times 10^{-1} + 6 \times 10^{-2}$

BINARY NUMBER SYSTEM:-

- The binary number system is a positional weighted system.
- The base or radix of this number system is 2.
- It has two independent symbols.
- The symbols used are 0 and 1.
- A binary digit is called a bit.
- The binary point separates the integer and fraction parts.

In general,

$$d_n\ d_{n-1}\ d_{n-2}\ \ldots\ldots\ldots\ d_0\ .\ d_{-1}\ d_{-2}\ \ldots\ldots\ldots d_{-k}$$

is given by

$$(d_n \times 2^n) + (d_{n-1} \times 2^{n-1}) + (d_{n-2} \times 2^{n-2}) + \ldots + (d_0 \times 2^0) + (d_{-1} \times 2^{-1}) + (d_{-2} \times 2^{-2}) + \ldots + (d_{-k} \times 2^{-k})$$

OCTAL NUMBER SYSTEM:-

- It is also a positional weighted system.
- Its base or radix is 8.
- It has 8 independent symbols 0,1,2,3,4,5,6 and 7.
- Its base $8 = 2^3$, every 3- bit group of binary can be represented by an octal digit.

HEXADECIMAL NUMBER SYSTEM:-

- The hexadecimal number system is a positional weighted system.
- The base or radix of this number system is 16.
- The symbols used are 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E and F
- The base 16 = 24 , every 4 – bit group of binary can be represented by an hexadecimal digit.

CONVERSION FROM ONE NUMBER SYSTEM TO ANOTHER :-

1. BINARY NUMBER SYSTEM:-

(a) Binary to decimal conversion:-

In this method, each binary digit of the number is multiplied by its positional weight and the product terms are added to obtain decimal number.

For example:

(i) Convert $(10101)_2$ to decimal.

Solution :

(Positional weight) $2^4\ 2^3\ 2^2\ 2^1\ 2^0$

Binary number 10101

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

$= 16 + 0 + 4 + 0 + 1$

$= (21)_{10}$

(ii) Convert $(111.101)_2$ to decimal.

Solution:

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

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

$= (7.625)_{10}$

(b) Binary to Octal conversion:-

For conversion binary to octal the binary numbers are divided into groups of 3 bits each, starting at the binary point and proceeding towards left and right.

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

For example:

(i) Convert $(101111010110.110110011)_2$ into octal.

Solution :

Group of 3 bits are	101	111	010	110	.	110	110	011
Convert each group into octal =	5	7	2	6	.	6	6	3

The result is **$(5726.663)_8$**

(ii) Convert $(10101111001.0111)_2$ into octal.

Solution :

Binary number		10	101	111	001	.	011	1
Group of 3 bits are	=	010	101	111	001	.	011	100
Convert each group into octal =		2	5	7	1	.	3	4

The result is **$(2571.34)_8$**

(c) Binary to Hexadecimal conversion:-

For conversion binary to hexadecimal number the binary numbers starting from the binary point, groups are made of 4 bits each, on either side of the binary point.

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

For example:
(i) Convert $(1011011011)_2$ into hexadecimal.

Solution:

Given Binary number 10 1101 1011

Group of 4 bits are 0010 1101 1011

Convert each group into hex = 2 D B

The result is **$(2DB)_{16}$**

(ii) Convert $(01011111011.011111)_2$ into hexadecimal.

Solution:

Given Binary number 010 1111 1011 . 0111 11

Group of 3 bits are = 0010 1111 1011 . 0111 1100

Convert each group into octal = 2 F B . 7 C

The result is **$(2FB.7C)_{16}$**

2. DECIMAL NUMBER SYSTEM:-

(a)Decimal to binary conversion:-

In the conversion the integer number are converted to the desired base using successive division by the base or radix.

For example:
(i) Convert $(52)_{10}$ into binary.

Solution:

Divide the given decimal number successively by 2 read the integer part remainder upwards to get equivalent binary number. Multiply the fraction part by 2. Keep the integer in the product as it is and multiply the new fraction in the product by 2. The process is continued and the integer are read in the products from top to bottom.

2 | 52
2 | 26 — 0
2 | 13 — 0
2 | 6 — 1
2 | 3 — 0
2 | 1 — 1
0 — 1

Result of $(52)_{10}$ is **$(110100)_2$**

(ii) Convert $(105.15)_{10}$ into binary.

Solution:

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

Result of $(105.15)_{10}$ is **$(1101001.001001)_2$**

(b)Decimal to octal conversion:-

To convert the given decimal integer number to octal, successively divide the given number by 8 till the quotient is 0. To convert the given decimal fractions to octal successively multiply the decimal fraction and the subsequent decimal fractions by 8 till the product is 0 or till the required accuracy is obtained.

For example:

(i) Convert $(378.93)_{10}$ into octal.

Solution:

8 | 378		0.93 x 8 = 7.44
8 | 47	— 2	0.44 x 8 = 3.52
8 | 5	— 7	0.52 x 8 = 4.16
0	— 5	0.16 x 8 = 1.28

Result of $(378.93)_{10}$ is **$(572.7341)_8$**

(c)Decimal to hexadecimal conversion:-

The decimal to hexadecimal conversion is same as octal.

For example:

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

Solution:

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

Result of $(2598.675)_{10}$ is **$(A26.ACCC)_{16}$**

3. OCTAL NUMBER SYSTEM:-

(a)Octal to binary conversion:-

To convert a given a octal number to binary, replace each octal digit by its 3- bit binary equivalent.

For example:

Convert $(367.52)_8$ into binary.

Solution:

Given Octal number is	3	6	7	.	5	2
Convert each group octal to binary	= 011	110	111	.	101	010

Result of $(367.52)_8$ is **$(011110111.101010)_2$**

(b)Octal to decimal conversion:-

For conversion octal to decimal number, multiply each digit in the octal number by the weight of its position and add all the product terms

For example: -

Convert $(4057.06)_8$ to decimal

Solution:

$(4057.06)_8 = 4 \times 8^3 + 0 \times 8^2 + 5 \times 8^1 + 7 \times 8^0 + 0 \times 8^{-1} + 6 \times 8^{-2}$

$= 2048 + 0 + 40 + 7 + 0 + 0.0937$

$= (2095.0937)_{10}$

Result is **$(2095.0937)_{10}$**

(c) Octal to hexadecimal conversion:-

For conversion of octal to Hexadecimal, first convert the given octal number to binary and then binary number to hexadecimal.

For example :-

Convert $(756.603)_8$ to hexadecimal.

Solution :-

Given octal no.		7	5	6	.	6	0	3
Convert each octal digit to binary	=	111	101	110	.	110	000	011
Group of 4bits are	=	0001	1110	1110	.	1100	0001	1000
Convert 4 bits group to hex.	=	1	E	E	.	C	1	8

Result is **$(1EE.C18)_{16}$**

(4) HEXADECIMAL NUMBER SYSTEM :-

(a)Hexadecimal to binary conversion:-

For conversion of hexadecimal to binary, replace hexadecimal digit by its 4 bit binary group.

For example:

Convert $(3A9E.B0D)_{16}$ into binary.

Solution:

Given Hexadecimal number is	3	A	9	E	.	B	0	D
Convert each hexadecimal digit to 4 bit binary	= 0011	1010	1001	1110	.	1011	0000	1101

Result of $(3A9E.B0D)_8$ is **$(0011101010011110.101100001101)_2$**

(b)Hexadecimal to decimal conversion:-

For conversion of hexadecimal to decimal, multiply each digit in the hexadecimal number by its position weight and add all those product terms.

For example: -
Convert $(A0F9.0EB)_{16}$ to decimal

Solution:

$(A0F9.0EB)_{16} = (10 \times 16^3)+(0 \times 16^2)+(15 \times 16^1)+(9 \times 16^0)+(0 \times 16^{-1})+(14 \times 16^{-2})+(11 \times 16^{-3})$
$= 40960 + 0 + 240 + 9 + 0 + 0.0546 + 0.0026$
$= (41209.0572)_{10}$

Result is **$(41209.0572)_{10}$**

(c) Hexadecimal to Octal conversion:-

For conversion of hexadecimal to octal, first convert the given hexadecimal number to binary and then binary number to octal.

For example :-
Convert $(B9F.AE)_{16}$ to octal.

Solution :-

Given hexadecimal no.is		B		9		F	.	A		E
Convert each hex. digit to binary	=	1011		1001		1111	.	1010		1110
Group of 3 bits are	=	101	110	011	111		.	101	011	100
Convert 3 bits group to octal.	=	5	6	3	7		.	5	3	4

Result is **$(5637.534)_8$**

BINARY ARITHEMATIC OPERATION :-

1. BINARY ADDITION:-

The binary addition rules are as follows
0 + 0 = 0 ; 0 + 1 = 1 ; 1 + 0 = 1 ; 1 + 1 = 10 , i.e 0 with a carry of 1

For example :-

Add $(100101)_2$ and $(1101111)_2$.
Solution :-

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

Result is **$(10010100)_2$**

2. BINARY SUBTRACTION:-

The binary subtraction rules are as follows
0 - 0 = 0 ; 1 - 1 = 0 ; 1 - 0 = 1 ; 0 - 1 = 1 , with a borrow of 1

For example :-
Substract $(111.111)_2$ from $(1010.01)_2$.
Solution :-

$$\begin{array}{r} 1010.010 \\ - \quad 111.111 \\ \hline 0010.011 \end{array}$$

Result is **$(0010.011)_2$**

3. BINARY MULTIPLICATION:-

The binary multiplication rules are as follows
0 x 0 = 0 ; 1 x 1 = 1 ; 1 x 0 = 0 ; 0 x 1 = 0
For example :-

Multiply $(1101)_2$ by $(110)_2$.
Solution :-

$$\begin{array}{r} 1101 \\ \times \quad 110 \\ \hline 0000 \\ 1101 \\ + \quad 1101 \\ \hline 1001110 \end{array}$$

Result is **$(1001110)_2$**

4. BINARY DIVISION:-

The binary division is very simple and similar to decimal number system. The division by '0' is meaningless. So we have only 2 rules
0 ÷ 1 = 0
1 ÷ 1 = 1
For example :-
Divide $(10110)_2$ by $(110)_2$.

Solution :-

$$\begin{array}{l} 110 \;)\; 101101 \;(\; 111.1 \\ \quad - \;\; 110 \\ \quad\quad 1010 \\ \quad\quad\; 110 \\ \quad\quad\quad 1001 \\ \quad\quad\;\; 110 \\ \quad\quad\quad 110 \\ \quad\quad\quad 110 \\ \quad\quad\quad\; 000 \end{array}$$

Result is **$(111.1)_2$**

1's COMPLEMENT REPRESENTATION :-

The 1's complement of a binary number is obtained by changing each 0 to 1 and each 1 to 0.

For example :-

Find $(1100)_2$ 1's complement.

Solution :-

Given	1	1	0	0
1's complement is	0	0	1	1

Result is **$(0011)_2$**

2's COMPLEMENT REPRESENTATION :-

The 2's complement of a binary number is a binary number which is obtained by adding 1 to the 1's complement of a number i.e.

2's complement = 1's complement + 1

For example :-

Find $(1010)_2$ 2's complement.

Solution :-

Given		1	0	1	0
1's complement is		0	1	0	1
	+				1
2's complement		0	1	1	0

Result is **$(0110)_2$**

SIGNED NUMBER :-

In sign – magnitude form, additional bit called the sign bit is placed in front of the number. If the sign bit is 0, the number is positive. If it is a 1, the number is negative.

For example:-

0 1 0 1 0 0 1 = +41

↑

Sign bit

1 1 0 1 0 0 1 = -41

↑

Sign bit

SUBSTRACTION USING COMPLEMENT METHOD :-

1's COMPLEMENT:-

In 1's complement subtraction, add the 1's complement of subtrahend to the minuend. If there is a carry out, then the carry is added to the LSB. This is called end around carry. If the MSB is 0, the result is positive. If the MSB is 1, the result is negative and is in its 1's complement form. Then take its 1's complement to get the magnitude in binary.

For example:-

Subtract $(10000)_2$ from $(11010)_2$ using 1's complement.

Solution:-

```
  1 1 0 1 0                  1 1 0 1 0                        =  26
- 1 0 0 0 0      =>      + 0 1 1 1 1  (1's complement)        = - 16
                 Carry →   1 0 1 0 0 1                        + 10
                         +           1
                           0 1 0 1 0    = +10
```

Result is **+10**

2's COMPLEMENT:-

In 2's complement subtraction, add the 2's complement of subtrahend to the minuend. If there is a carry out, ignore it. If the MSB is 0, the result is positive. If the MSB is 1, the result is negative and is in its 2's complement form. Then take its 2's complement to get the magnitude in binary.

For example:-

Subtract $(1010100)_2$ from $(1010100)_2$ using 2's complement.

Solution:-

```
  1 0 1 0 1 0 0                  1 0 1 0 1 0 0                    =   84
- 1 0 1 0 1 0 0      =>      + 0 1 0 1 1 0 0  (2's complement)    = - 84
                               1 0 0 0 0 0 0 0 ( Ignore the carry)     0
                          =                0 (result = 0)
```

Hence MSB is 0. The answer is positive. So it is +0000000 = **0**

DIGITAL CODES:-

In practice the digital electronics requires to handle data which may be numeric, alphabets and special characters. This requires the conversion of the incoming data into binary format before it can be processed. There is various possible ways of doing this and this process is called encoding. To achieve the reverse of it, we use decoders.

WEIGHTED AND NON-WEIGHTED CODES:-

There are two types of binary codes

1) Weighted binary codes
2) Non- weighted binary codes

In weighted codes, for each position (or bit) ,there is specific weight attached.

For example, in binary number, each bit is assigned particular weight 2n where 'n' is the bit number for n = 0,1,2,3,4 the weights are 1,2,4,8,16 respectively.

Example :- BCD

Non-weighted codes are codes which are not assigned with any weight to each digit position, i.e., each digit position within the number is not assigned fixed value.

Example:- Excess – 3 (XS -3) code and Gray codes

BINARY CODED DECIMAL (BCD):-

BCD is a weighted code. In weighted codes, each successive digit from right to left represents weights equal to some specified value and to get the equivalent decimal number add the products of the weights by the corresponding binary digit. 8421 is the most common because 8421 BCD is the most natural amongst the other possible codes.

For example:-

$(567)_{10}$ is encoded in various 4 bit codes.

Solution:-

Decimal	→	5	6	7
8421 code	→	0101	0110	0111
6311 code	→	0111	1000	1001
5421 code	→	1000	0100	1010

BCD ADDITION:-

Addition of BCD (8421) is performed by adding two digits of binary, starting from least significant digit. In case if the result is an illegal code (greater than 9) or if there is a carry out of one then add 0110(6) and add the resulting carry to the next most significant.

For example:-

Add 679.6 from 536.8 using BCD addition.

Solution:-

6 7 9.6	0110 0111 1001 . 0110	(679.6 in BCD)
+ 5 3 6.8	=>+ 0101 0011 0110 . 1000	(536.8 in BCD)
1 2 1 6.4	1011 1010 1111 . 1110	(All are illegal codes)
	+ 0110 +0110 +0110 .+0110	(Add 0110 to each)
	0001 0010 0001 0110 . 0100	
	1 2 1 6 . 4	(corrected sum = 1216.4)

Result is **1216.4**

BCD SUBTRACTION:-

The BCD subtraction is performed by subtracting the digits of each 4 – bit group of the subtrahend from corresponding 4 – bit group of the minuend in the binary starting from the LSD. If there is no borrow from the next higher group[then no correction is required. If there is a borrow from the next group, then 6_{10} (0110) is subtracted from the difference term of this group.

For example:-

Subtract 147.8 from 206.7 using 8421 BCD code.

Solution:-

2 0 6.7	0010 0000 0110 . 0111	(206.7 in BCD)
- 1 4 7.8	=>- 0001 0100 0111 . 1000	(147.8 in BCD)
5 8.9	0000 1011 1110 . 1111	(Borrows are present)
	- 0110 -0110 .- 0110	
	0101 1000 . 1001	
	5 8 . 9	(corrected difference = 58.9)

Result is **$(58.9)_{10}$**

EXCESS THREE(XS-3) CODE:-

The Excess-3 code, also called XS-3, is a non- weighted BCD code. This derives it name from the fact that each binary code word is the corresponding 8421 code word plus 0011(3). It is a sequential code. It is a self complementing code.

XS-3 ADDITION:-

In XS-3 addition, add the XS-3 numbers by adding the 4 bit groups in each column starting from the LSD. If there is no carry out from the addition of any of the 4 bit groups, subtract 0011 from the sum term of those groups. If there is a carry out, add 0011 to the sum term of those groups

For example:-

Add 37 and 28 using XS-3 code.

Solution:-

```
  3  7              0110  1010     ( 37 in XS-3)
+ 2  8      =>   + 0101  1011     ( 28 in XS-3)
  6  5              1011 11010     ( Carry is generated)
                  +    1          ( Propagate carry)
                    1100  0101    ( Add 0110 to correct 0101 and
                  - 0011 +0011    subtract 0011 to correct 1100)
                    1001  1000    ( Corrected sum in XS-3 = 65(10))
```

XS-3 SUBTRACTION:-

To subtract in XS-3 number by subtracting each 4-bit group of the subtrahend from the corresponding 4-bit group of the minuend starting from the LSD. If there is no borrow from the next 4-bit group. add 0011 to the difference term of such groups. If there is a borrow, subtract 0011 from the difference term.

For example :-

. **Subtract 175 from 267 using XS-3 code.**

Solution :-`

```
  267              0101  1010  1010  ( 267 in XS-3)
 -175      =>   - 0100  1010  1000  ( 175 in XS-3)
  092              0000  1111  0010  (Correct 0010 and 0000 by adding 0011 and
                 +0011 -0011 +0011     correct 1111 by subtracting 0011)
                  0011  1100  0101  (Corrected difference in XS-3 = 92(10) )
```

ASCII CODE:-

The American Standard Code for Information Interchange (ASCII) pronounced as 'ASKEE' is widely used alphanumeric code. This is basically a 7 bit code. The number of different bit patterns that can be created with 7 bits is 27 = 128 , the ASCII can be used to encode both the uppercase and lowercase characters of the alphabet (52 symbols) and some special symbols in addition to the 10 decimal digits. It is used extensively for printers and terminals that interface with small computer systems. The table shown below shows the ASCII groups.

The ASCII code

LSBs	MSBs							
	000	001	010	011	100	101	110	111
0000	NUL	DEL	Space	0	@	P	P	
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s

0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	‘	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DLE

EBCDIC CODE:-

The Extended Binary Coded Decimal Interchange Code (EBCDIC) pronounced as ‘eb – si- dik’ is an 8 bit alphanumeric code. Since 28 = 256 bit patterns can be formed with 8 bits. It is used by most large computers to communicate in alphanumeric data. The table shown below shows the EBCDIC code.

The EBCDIC code

LSD (Hex)	**MSD(Hex)**															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	DLE	DS		SP	&							[	]	\	0
1	SOH	DC1	SOS				/		a	j	~		A	J		1
2	STX	DC2	FS	SYN					b	k	s		B	K	S	2
3	ETX	DC3							c	l	t		C	L	T	3
4	PF	RES	BYP	PN					d	m	u		D	M	U	4
5	HT	NL	LF	RS					e	n	v		E	N	V	5
6	LC	BS	EOB	YC					f	o	w		F	O	W	6
7	DEL	IL	PRE	EOT					g	p	x		G	P	X	7
8		CAN							h	q	y		H	Q	Y	8
9		EM							i	r	z		I	R	Z	9
A	SMM	CC	SM		Ø	!	I	:								
B	VT				.	$	,	#								
C	FF	IFS		DC4	<	*	%	@								
D	CR	IGS	ENQ	NAK	(	)	_	‘								
E	SO	IRS	ACK		+	;	>	=								
F	SI	IUS	BEL	SUB	I	‘	?	‘								

GRAY CODE:-

The gray code is a non-weighted code. It is not a BCD code. It is cyclic code because successive words in this differ in one bit position only i.e it is a unit distance code.

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

BINARY- TO – GRAY CONVERSION:-

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

$G_n = B_n$

$G_{n-1} = B_n \oplus B_{n-1}$

.

.

.

.

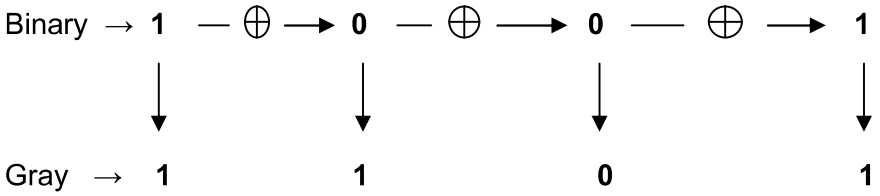
$G_1 = B_2 \oplus B_1$

Where the symbol $\oplus$ stands for Exclusive OR (X-OR)

For example :-

Convert the binary 1001 to the Gray code.

Solution :-`



The gray code is **1101**

GRAY- TO - BINARY CONVERSION:-

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

$B_n = G_n$

$B_{n-1} = B_n \oplus G_{n-1}$

.

.

.

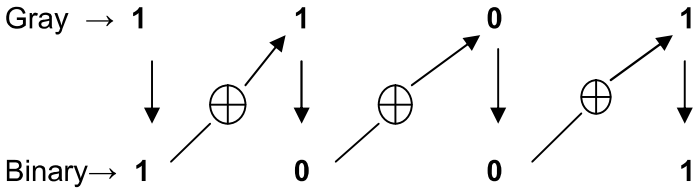
.

$B_1 = B_2 \oplus G_1$

For example :-

Convert the Gray code 1101 to the binary.

Solution :-



The binary code is **1001**

LOGIC GATES

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 the lower 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 the higher of the two voltages level represents the logic 0.

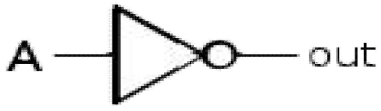
DIFFERENT TYPES OF LOGIC GATES:-

NOT GATE (INVERTER):-

- A NOT gate, also called and 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

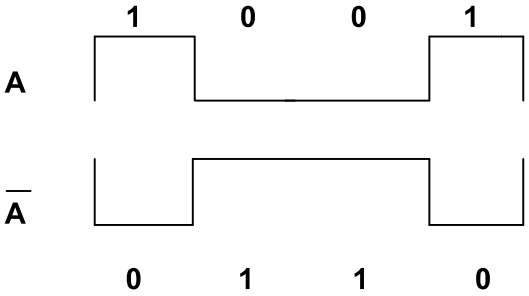
Logic Symbol



Truth table

INPUT A	OUTPUT $\overline{A}$
0	1
1	0

Timing Diagram

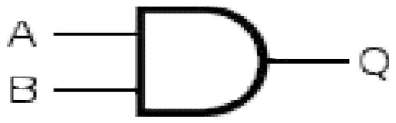


AND GATE:-

- 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

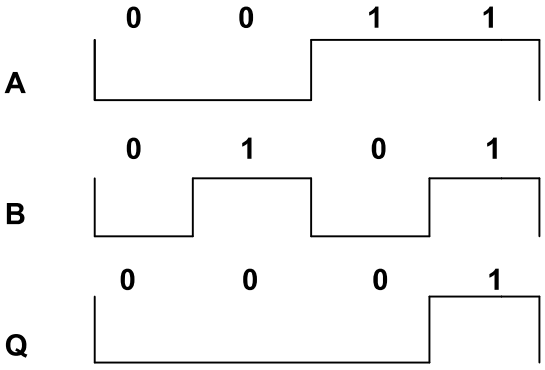
Logic Symbol



Truth Table

		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.

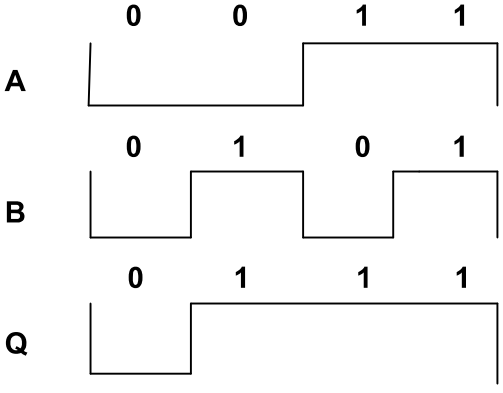
IC No.:- 7432

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



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.

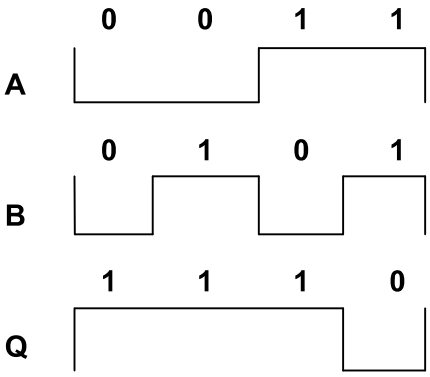
IC No.:- 7400 two input NAND gate
7410 three input NAND gate
7420 four input NAND gate
7430 eight input NAND gate

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.

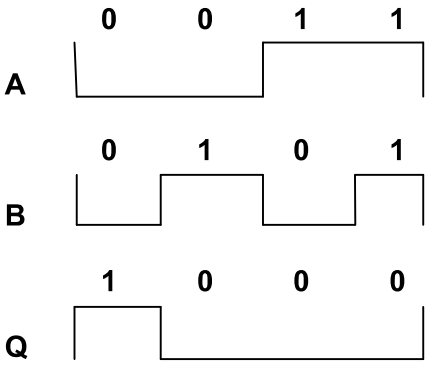
IC No.:- 7402 two input NOR gate
7427 three input NOR gate
7425 four input NOR gate

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

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

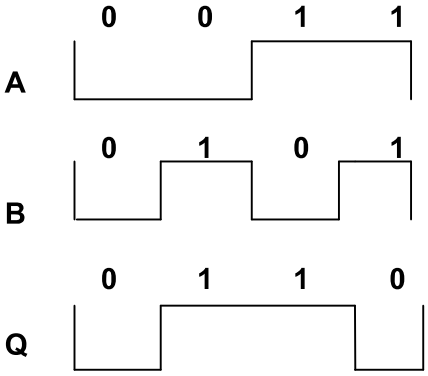
OUTPUT is **Q** = $A \oplus B$

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

Truth Table

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

Timing Diagram

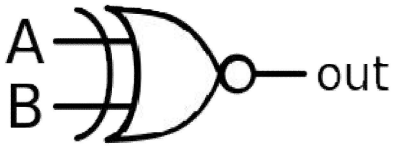


EXCLUSIVE – NOR (X-NOR) GATE:-

- 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

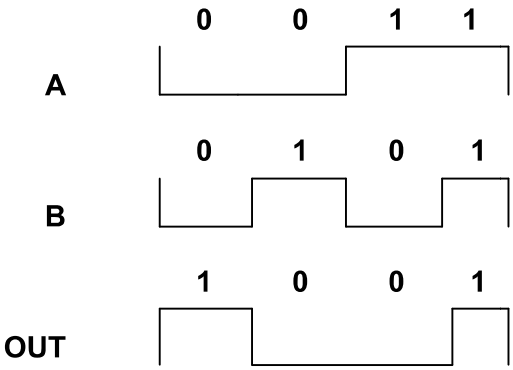


INPUT		OUTPUT
A	B	OUT =A XNOR B
0	0	1
0	1	0
1	0	0
1	1	1

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

= A XNOR B

Timing Diagram

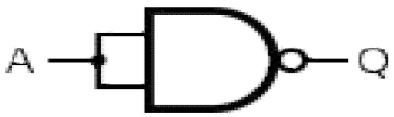


UNIVERSAL GATES:-

There are 3 basic gates AND, OR and NOT, there are two universal gates NAND and NOR, each of which can realize logic circuits single handedly. 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:-

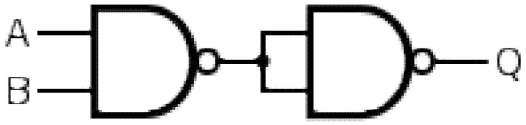
a) **Inverter from NAND gate**



Input = **A**
Output **Q =** $\overline{A}$

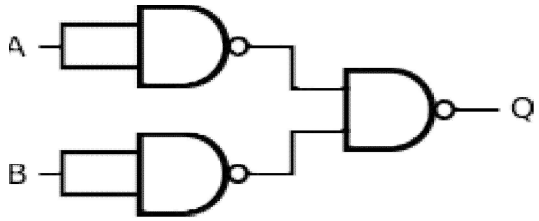
b) **AND gate from NAND gate**

Input s are **A** and **B**
Output **Q = A.B**



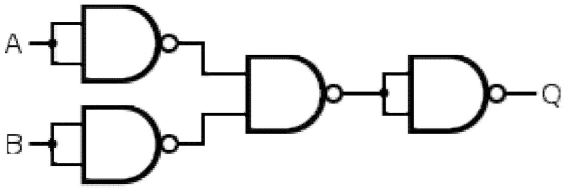
c) **OR gate from NAND gate**

Inputs are **A** and **B**
Output **Q = A+B**



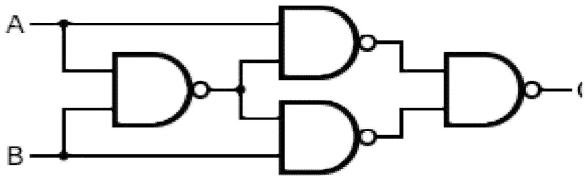
d) NOR gate from NAND gate

Inputs are **A** and **B**
Output **Q = $\overline{A+B}$**



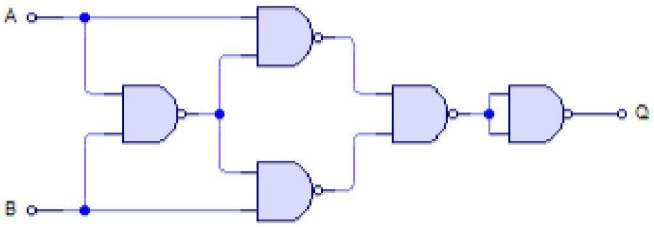
e) EX-OR gate from NAND gate

Inputs are **A** and **B**
Output **Q = A $\overline{B}$ + $\overline{A}$B**



f) EX-NOR gate From NAND gate

Inputs are **A** and **B**
Output **Q = A B + $\overline{A}$ $\overline{B}$**



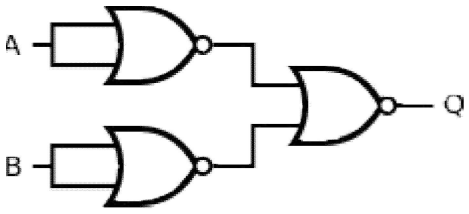
NOR GATE:-

a) Inverter from NOR gate

Input = **A**
Output **Q = $\overline{A}$**

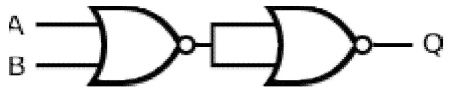
b) AND gate from NOR gate

Input s are **A** and **B**
Output **Q = A.B**



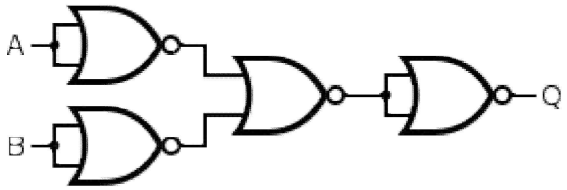
c) OR gate from NOR gate

Inputs are **A** and **B**
Output **Q = A+B**



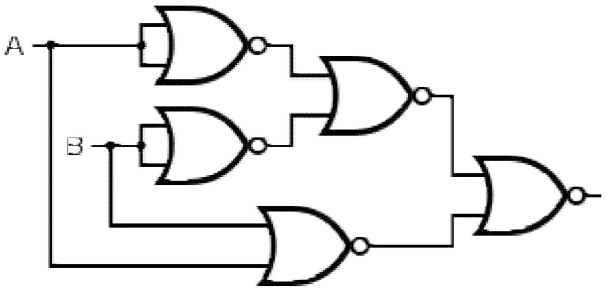
d) NAND gate from NOR gate

Inputs are **A** and **B**
Output **Q = $\overline{A.B}$**



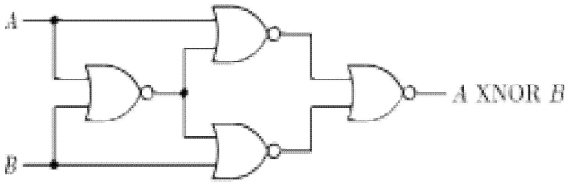
e) EX-OR gate from NOR gate

Inputs are **A** and **B**
Output **Q = A $\overline{B}$ + $\overline{A}$B**



f) EX-NOR gate From NOR gate

Inputs are **A** and **B**
Output **Q = A B + $\overline{A}$ $\overline{B}$**



THRESHOLD LOGIC:-

INTRODUCTION:-

- The threshold element, also called the threshold gate (T-gate) is a much more powerful device than any of the conventional logic gates such as NAND, NOR and others.
- Complex, large Boolean functions can be realized using much fewer threshold gates.
- Frequently a single threshold gate can realize a very complex function which otherwise might require a large number of conventional gates.
- T-gate offers incomparably economical realization; it has not found extensive use with the digital system designers mainly because of the following limitations.
  1. It is very sensitive to parameter variations.
  2. It is difficult to fabricate it in IC form.

3. The speed of switching of threshold elements in much lower than that of conventional gates.

THE THRESHOLD ELEMENTS:-

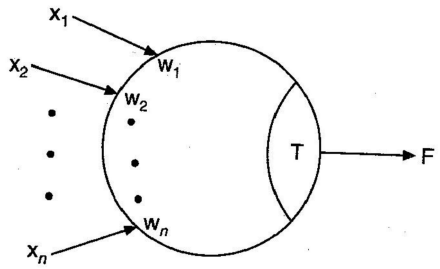
- A threshold element or gate has 'n' binary inputs x_1, x_2, ….., x_n; and a single binary output F. But in addition to those, it has two more parameters.
- Its parameters are a threshold T and weights w_1, w_2, ….,w_n. The weights w_1, w_2, …, w_n are associated with the input variables x_1, x_2, …, x_n.
- The value of the threshold (T) and weights may be real, positive or negative number.
- The symbol of the threshold element is shown in fig.(a).
- It is represented by a circle partitioned into two parts, one part represents the weights and other represents T.
- It is defined as

  $F(x_1, x_2, \ldots\ldots, x_n) = 1$ if and only if $\sum_{i=1}^{n} w_i x_i \geq T$

  otherwise

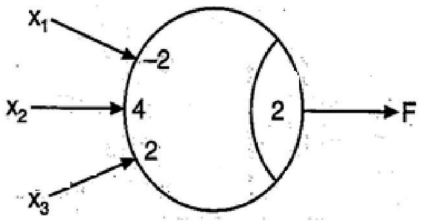
  $F(x_1, x_2, \ldots\ldots, x_n) = 0$

- The sum and product operation are normal arithmetic operations and the sum $\sum_{i=1}^{n} w_i x_i \geq T$ is called the weighted sum of the element or gate.



Example:-

Obtain the minimal Boolean expression from the threshold gate shown in figure.



Solution:-

The threshold gate with three inputs x_1, x_2, x_3 with weights $-2(w_1)$, $4(w_2)$ and $2(w_3)$ respectively. The value of threshold is 2(T). The table shown is the weighted sums and outputs for all input combinations. For this threshold gate, the weighted sum is

$$w = w_1x_1 + w_2x_2 + w_3x_3$$

$$= (-2)x_1 + (4)x_2 + (2)x_3$$

$$= -2x_1 + 4x_2 + 2x_3$$

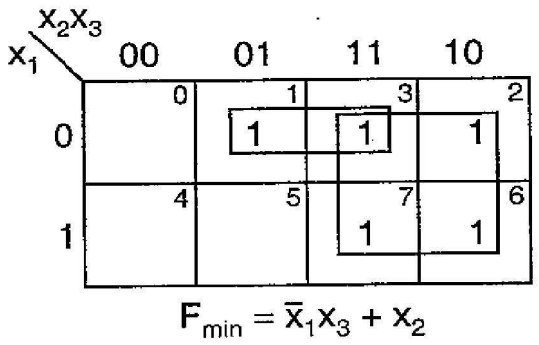
The output F is logic 1 for w≥2 and it is logic 0 for w<2

Input Variables			Weighted Sum	Output
x_1	x_2	x_3	$w = -2x_1 + 4x_2 + 2x_3$	F
0	0	0	0	0
0	0	1	2	1
0	1	0	4	1
0	1	1	6	1
1	0	0	-2	0
1	0	1	0	0
1	1	0	2	1
1	1	1	4	1

From the input – output relation is given in the table, the Boolean expression for the output is

$$F=\sum m\,(1, 2, 3, 6, 7)$$

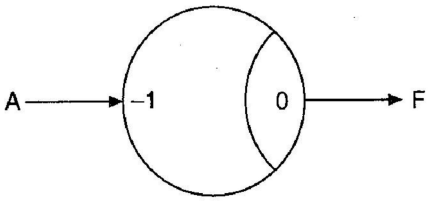
The K-map for F is



$$F_{min} = \bar{x}_1 x_3 + x_2$$

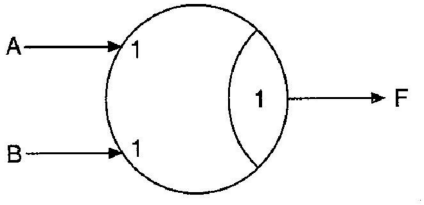
UNIVERSALITY OF A T-GATE:-

- A single T-gate can realize a large number of functions by merely changing either the weights or the threshold or both, which can be done by altering the value of the corresponding resistors.
- Since a threshold gate can realize universal gates, i.e., NAND gates and NOR gates, a threshold gate is also a universal gate.
- Single threshold gate cannot realize by a single T-gate
- Realization of logic gates using T-gates is shown in the below figure.



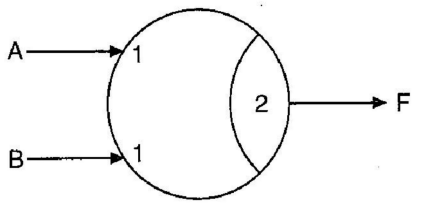
(a) NOT gate $F = \overline{A}$

Input	Weighted sum	Output
A	W = −A	F
0	0	1
1	−1	0



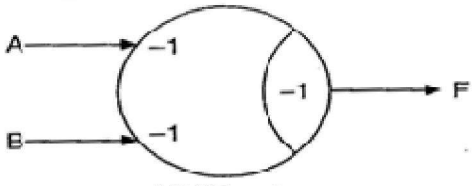
(b) OR gate F = A + B

Inputs		Weighted sum	Output
A	B	w = A + B	F
0	0	0	0
0	1	1	1
1	0	1	1
1	1	2	1



(c) AND gate F = AB

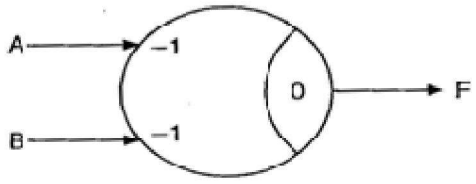
Inputs		Weighted sum	Output
A	B	w = A + B	F
0	0	0	0
0	1	1	0
1	0	1	0
1	1	2	1



NAND gate

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

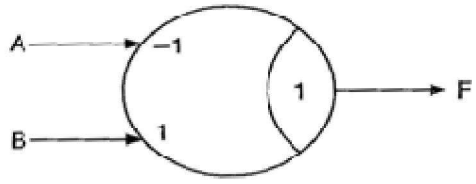
Inputs		Weighted sum	Output
A	B	w = −A − B	F
0	0	0	1
0	1	−1	1
1	0	−1	1
1	1	−2	0



NOR gate

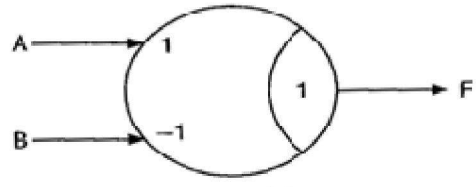
(e) $F = \overline{A} \cdot \overline{B} = \overline{A + B}$

Inputs		Weighted sum	Output
A	B	w = −A − B	F
0	0	0	1
0	1	−1	0
1	0	−1	0
1	1	−2	0



(f) $F = \overline{A}B$

Inputs		Weighted sum	Output
A	B	w = −A + B	F
0	0	0	0
0	1	1	1
1	0	−1	0
1	1	0	0



(g) $F = A\overline{B}$

Inputs		Weighted sum	Output
A	B	w = A − B	F
0	0	0	0
0	1	−1	0
1	0	1	1
1	1	0	0

BOOLEAN ALGEBRA

INTRODUCTION:-

- 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: $\overline{1} = 0$
Axiom 2: 0 . 1 = 0	Axiom 6: 0 + 1 = 1	Axiom 10: $\overline{0} = 1$
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: $\overline{0} = 1$

Law 2: $\overline{1} = 0$

Law 3: if A = 0, then $\overline{A} = 1$

Law 4: if $A = 1$, then $\overline{A} = 0$

Law 5: $\overline{\overline{A}} = 0$ (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	A+(B+C)
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: $A + \bar{A}B = A + B$

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

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

Proof

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

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

10. Consensus Theorem (Included Factor Theorem):-

Theorem 1:

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

Proof

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

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

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

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

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

$= AB + \bar{A}C$

$=$ **RHS**

Theorem 2:

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

Proof

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

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

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

$= ABC + BC + \bar{A}B + AC + BC + \bar{A}BC$

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

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

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

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

$= LHS$

11. Transposition Theorem:-

Theorem:

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

Proof

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

$= A\bar{A} + C\bar{A} + AB + CB$

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

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

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

$= AB + \bar{A}C$

$= LHS$

12. 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 complements.

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

Proof

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

=

A	B	$\overline{A}$	$\overline{B}$	$\overline{A} + \overline{B}$
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 stand proved. This is called the principle of duality.

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

Relations between complement and dual

$f_c(A, B, C, \ldots..) = \overline{f(A, B, C, \ldots..)} = f_d(\overline{A}, \overline{B}, \overline{C}, \ldots)$

$f_d(A, B, C, \ldots..) = f(\overline{A}, \overline{B}, \overline{C}, \ldots) = f_c(\overline{A}, \overline{B}, \overline{C}, \ldots..)$

The first relation states that the complement of a function f(A, B, C, …) can be obtained by complementing all the variables in the dual function $f_d(A, B, C, \ldots..)$.

The second relation states that the dual can be obtained by complementing all the literals in $\overline{f(A, B, C, \ldots.)}$.

DUALS:-

	Given expression	*Dual*
1.	$\overline{0} = 1$	$\overline{1} = 0$
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.	$\overline{A} \cdot (\overline{A} \cdot B) = A \cdot B$	$\overline{A + A} + \overline{B} = A + B$
14.	$\overline{AB} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A}\,\overline{B}$
15.	$(A + B)(\overline{A} + C)(B + C) = (A + B)(\overline{A} + C)$	$AB + \overline{A}C + BC = AB + \overline{A}C$
16.	$A + \overline{B}C = (A + \overline{B})(A + C)$	$A(\overline{B} + C) = A\overline{B} + AC$
17.	$(A+C)(\overline{A}+B) = AB+\overline{A}C$	$AC+\overline{A}B=(A+B)(\overline{A}+C)$
18.	$(A+B)(C+D) = AC + AD + BC + BD$	$(AB+CD) = (A+C)(A+D)(B+C)(B+D)$
19.	$A + B = AB + \overline{A}B + A\overline{B}$	$AB = (A+B)(\overline{A}+B)(A+\overline{B})$
20.	$\overline{A}\overline{B} + \overline{A} + AB = 0$	$\overline{A + B} \cdot \overline{A} \cdot (A + B) = 1$

SUM - OF - PRODUCTS FORM:-

- This is also called disjunctive Canonical Form (DCF) or Expanded Sum of Products Form or Canonical Sum of Products Form.
- In this form, the function is the sum of a number of products terms where each product term contains all variables of the function either in complemented or uncomplemented form.
- This can also be derived from the truth table by finding the sum of all the terms that corresponds to those combinations for which 'f ' assumes the value 1.

  For example

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

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

  $= \bar{A}\bar{B}C + A\bar{B}\bar{C} + A\bar{B}C + ABC$
- The product term which contains all the variables of the functions either in complemented or uncomplemented form is called a minterm.
- The minterm is denoted as mo, m1, m2 … .
- An 'n' variable function can have 2n minterms.
- Another way of representing the function in canonical SOP form is the showing the sum of minterms for which the function equals to 1.

  For example

  $f(A, B, C) = m_1 + m_2 + m_3 + m_5$

  or

  $f(A, B, C) = \sum m(1, 2, 3, 5)$

  where $\sum m$ represents the sum of all the minterms whose decimal codes are given the parenthesis.

PRODUCT- OF - SUMS FORM:-

- This form is also called as Conjunctive Canonical Form (CCF) or Expanded Product - of – Sums Form or Canonical Product Of Sums Form.
- This is by considering the combinations for which f = 0
- Each term is a sum of all the variables.
- The function $f(A, B, C) = (A + B + C \cdot C) + (A + B + C \cdot \bar{C})$

  $= (\bar{A} + \bar{B} + C)(\bar{A} + \bar{B} + \bar{C})(A + B + C)(A + B + \bar{C})$
- The sum term which contains each of the 'n' variables in either complemented or uncomplemented form is called a maxterm.
- Maxterm is represented as $M_0, M_1, M_2, \ldots\ldots$

  Thus CCF of 'f' may be written as

  $f(A, B, C) = M_0 \cdot M_4 \cdot M_6 \cdot M_7$

  or

  $f(A, B, C) = (0, 4, 6, 7)$

  Where represented the product of all maxterms.

CONVERSION BETWEEN CANONICAL FORM:-

The complement of a function expressed as the sum of minterms equals the sum of minterms missing from the original function.

Example:-

$f(A, B, C) = \sum m(0,2,4,6,7)$

This has a complement that can be expressed as

$\overline{f(A, B, C)} = \sum m(1, 3, 5) = m_1 + m_3 + m_5$

If we complement f by De- Morgan's theorem we obtain 'f' in a form.

$f = \overline{(m_1 + m_3 + m_5)} = \bar{m}_1 . \bar{m}_3 . \bar{m}_5$

$= M_1 M_3 M_5 = \prod M(1, 3, 5)$

Example:-

Expand $A(\overline{A} + B)(\overline{A} + B + \overline{C})$ to maxterms and minterms.

Solution:-

In POS form

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

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

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

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

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

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

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

Therefore

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

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

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

$= M_0 \cdot M_1 \cdot M_2 \cdot M_3 \cdot M_4 \cdot M_5$

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

The maxterms M_6 and M_7 are missing in the POS form.

So, the SOP form will contain the minterms 6 and 7

KARNAUGH MAP OR K- MAP:-

- The K- map is a chart or a graph, composed of an arrangement of adjacent cells, each representing a particular combination of variables in sum or product form.
- The K- map is systematic method of simplifying the Boolean expression.

TWO VARIABLE K- MAP:-

A two variable expression can have $2^2 = 4$ possible combinations of the input variables A and B.

Mapping of SOP Expression:-

- The 2 variable K-map has $2^2 = 4$ squares. These squares are called cells.
- A '1' is placed in any square indicates that corresponding minterm is included in the output expression, and a 0 or no entry in any square indicates that the corresponding minterm does not appear in the expression for output.

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

Example:-

Map expression $f = \overline{A}B + A\overline{B}$

Solution:-

The expression minterms is

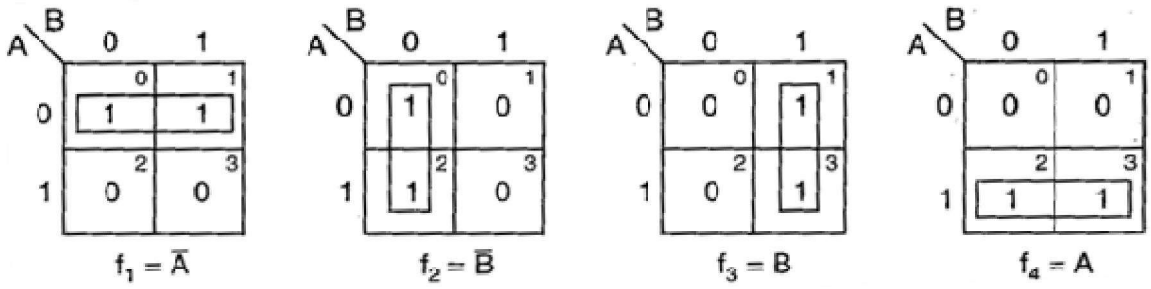
$F = m_1 + m_2 = m(1, 2)$

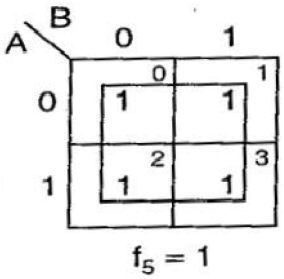
A \ B	0	1
0	0 (0)	1 (1)
1	1 (2)	0 (3)

Minimization of SOP Expression:-

To minimize a Boolean expression given in the SOP form by using K- map, the adjacent squares having 1s, that is minterms adjacent to each other are combined to form larger squares to eliminate some variables.

The possible minterm grouping in a two variable K- map are shown below





- Two minterms, which are adjacent to each other, can be combined to form a bigger square called 2 – square or a pair. This eliminates one variable that is not common to both the minterms.
- Two 2-squares adjacent to each other can be combined to form a 4- square. A 4- square eliminates 2 variables. A 4-square is called a quad.
- Consider only those variables which remain constant throughout the square, and ignore the variables which are varying. The non-complemented variable is the variable remaining constant as 1.The complemented variable is the variable remaining constant as a 0 and the variables are written as a product term.

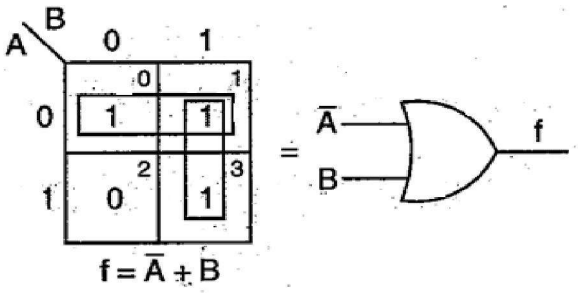
Example:-

Reduce the expression f= $\bar{A}\bar{B} + \bar{A}B + AB$ using mapping.

Solution:-

Expressed in terms of minterms, the given expression is

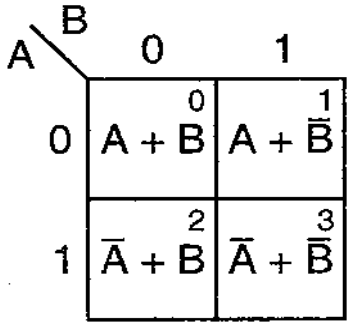
$$f = m_0 + m_1 + m_3 = \sum m\ (0, 1, 3)$$



F = $\bar{A}$ + B

Mapping of POS Expression:-

Each sum term in the standard POS expression is called a Maxterm. A function in two variables (A,B) has 4 possible maxterms, $A + B$, $A + \overline{B}$, $\overline{A} + B$ and $\overline{A} + \overline{B}$. They are represented as M_0, M_1, M_2 and M_3 respectively.



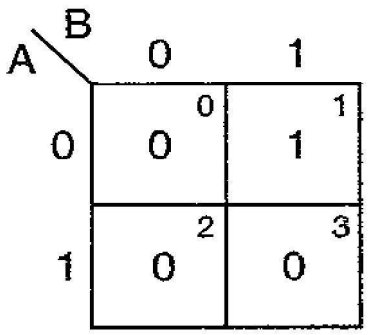
The maxterm of a two variable K-map

Example:-

Plot the expression f= $(A + B)(\overline{A} + B)(\overline{A} + \overline{B})$

Solution:-

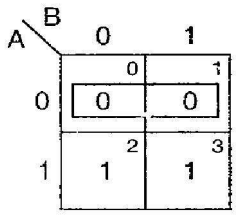
Expression interms of maxterms is f = $\Pi M (0, 2, 3)$



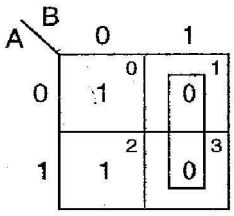
Minimization of POS Expressions:-

In POS form the adjacent 0s are combined into large square as possible. If the squares having complemented variable then the value remain constant as a 1 and the non-complemented variable if its value remains constant as a 0 along the entire square and then their sum term is written.

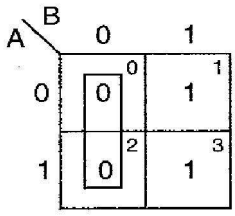
The possible maxterms grouping in a two variable K-map are shown below



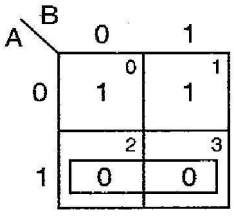
$f_1 = A$



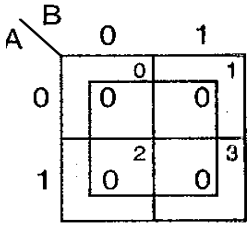
$f_2 = \overline{B}$



$f_3 = B$



$f_4 = \overline{A}$



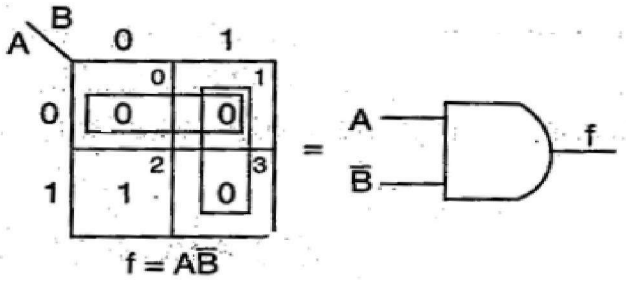
$f_5 = 0$

Example:-

Reduce the expression f = $(A + \bar{B})(\bar{A} + \bar{B})(A + B)$ using mapping

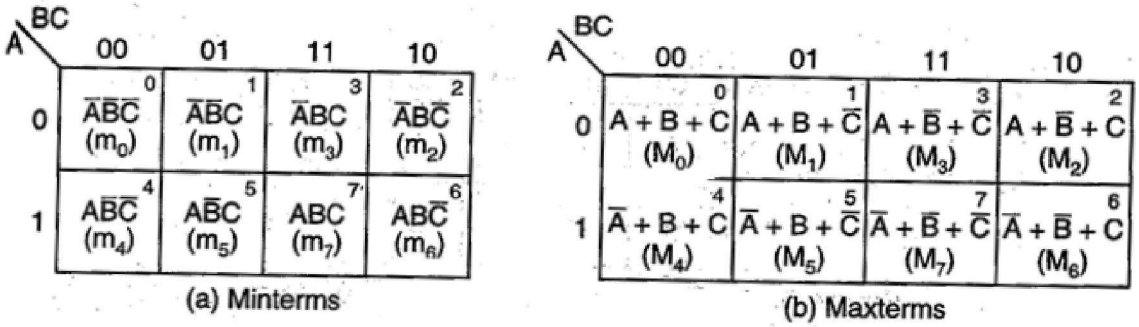
Solution:-

The given expression in terms of maxterms is f = ΠM (0, 1, 3)



THREE VARIABLE K- MAP:-

A function in three variables (A, B, C) can be expressed in SOP and POS form having eight possible combination. A three variable K- map have 8 squares or cells and each square on the map represents a minterm or maxterm is shown in the figure below.

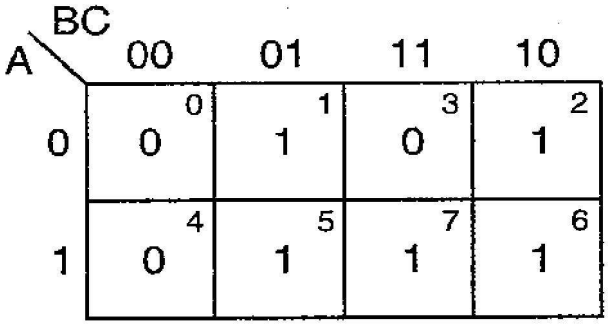


Example:-

Map the expression f = $\bar{A}\bar{B}C + A\bar{B}\bar{C} + \bar{A}B\bar{C} + AB\bar{C} + ABC$

Solution:-

So in the SOP form the expression is f = $\sum$ m (1, 5, 2, 6, 7)

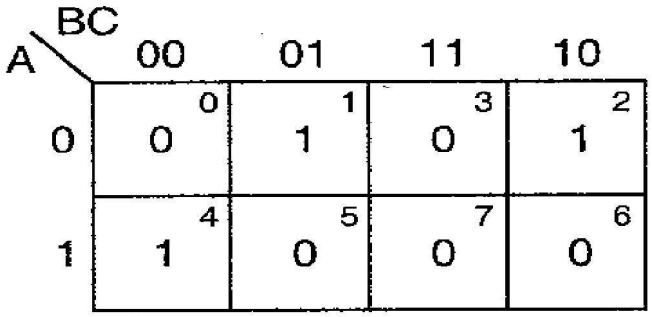


Example:-

Map the expression f = $(A + B + C)(\bar{A} + B + \bar{C})(\bar{A} + \bar{B} + \bar{C})(A + \bar{B} + \bar{C})(\bar{A} + \bar{B} + C)$

Solution:-

So in the POS form the expression is f = Π M (0, 5, 7, 3, 6)

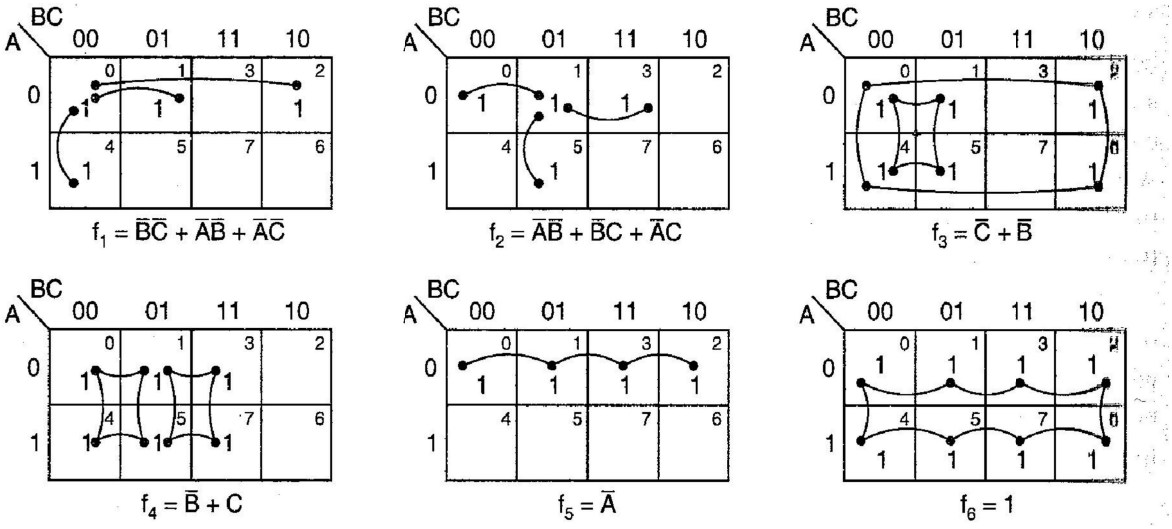


Minimization of SOP and POS Expressions:-

For reducing the Boolean expressions in SOP (POS) form the following steps are given below

- Draw the K-map and place 1s (0s) corresponding to the minterms (maxterms) of the SOP (POS) expression.
- In the map 1s (0s) which are not adjacent to any other 1(0) are the isolated minterms (maxterms). They are to be read as they are because they cannot be combined even into a 2-square.
- For those 1s (0s) which are adjacent to only one other 1(0) make them pairs (2 squares).
- For quads (4- squares) and octet (8 squares) of adjacent 1s (0s) even if they contain some 1s (0s) which have already been combined. They must geometrically form a square or a rectangle.
- For any 1s (0s) that have not been combined yet then combine them into bigger squares if possible.
- Form the minimal expression by summing (multiplying) the product (sum) terms of all the groups.

Some of the possible combinations of minterms in SOP form

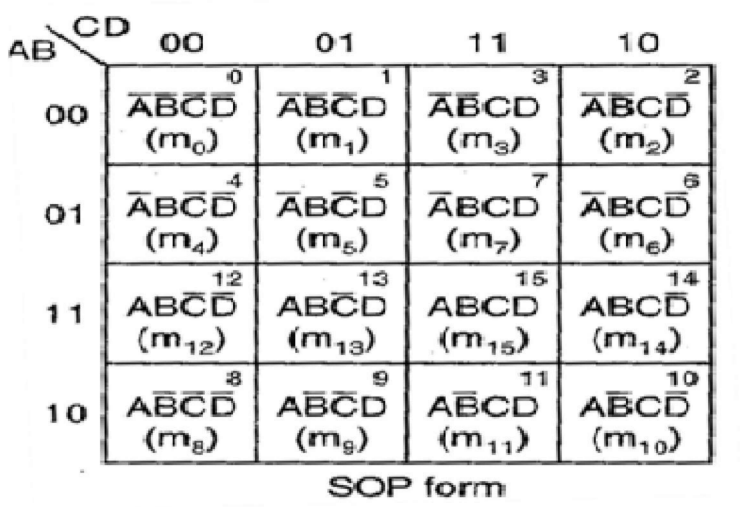


These possible combinations are also for POS but 1s are replaced by 0s.

FOUR VARIABLE K-MAP:-

A four variable (A, B, C, D) expression can have $2^4 = 16$ possible combinations of input variables. A four variable K-map has $2^4 = 16$ squares or cells and each square on the map represents either a minterm or a maxterm as shown in the figure below. The binary number designations of the rows and columns are in the gray code. The binary numbers along the top of the map indicate the conditions of C and D along any column and binary numbers along left side indicate the conditions of A and B along any row. The numbers in the top right corners of the squares indicate the minterm or maxterm desginations.

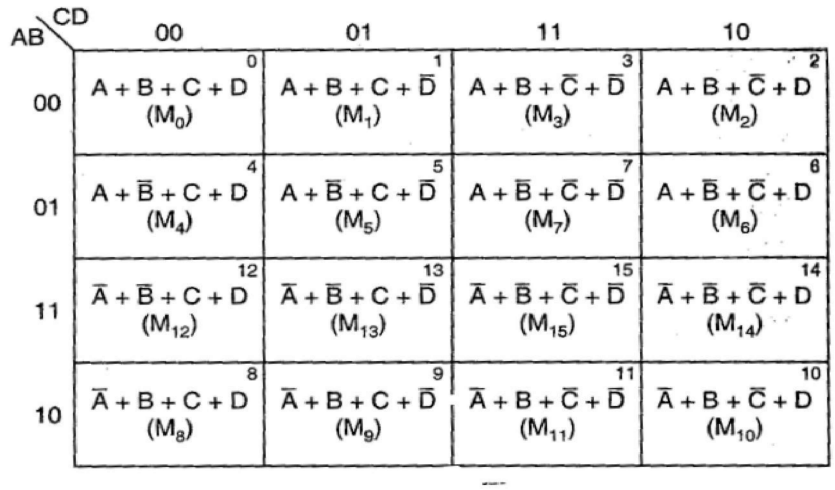
SOP FORM



AB \ CD	00	01	11	10
00	$\bar{A}\bar{B}\bar{C}\bar{D}$ (m_0) 0	$\bar{A}\bar{B}\bar{C}D$ (m_1) 1	$\bar{A}\bar{B}CD$ (m_3) 3	$\bar{A}\bar{B}C\bar{D}$ (m_2) 2
01	$\bar{A}B\bar{C}\bar{D}$ (m_4) 4	$\bar{A}B\bar{C}D$ (m_5) 5	$\bar{A}BCD$ (m_7) 7	$\bar{A}BC\bar{D}$ (m_6) 6
11	$AB\bar{C}\bar{D}$ (m_{12}) 12	$AB\bar{C}D$ (m_{13}) 13	$ABCD$ (m_{15}) 15	$ABC\bar{D}$ (m_{14}) 14
10	$A\bar{B}\bar{C}\bar{D}$ (m_8) 8	$A\bar{B}\bar{C}D$ (m_9) 9	$A\bar{B}CD$ (m_{11}) 11	$A\bar{B}C\bar{D}$ (m_{10}) 10

SOP form

POS FORM



AB \ CD	00	01	11	10
00	$A+B+C+D$ (M_0) 0	$A+B+C+\bar{D}$ (M_1) 1	$A+B+\bar{C}+\bar{D}$ (M_3) 3	$A+B+\bar{C}+D$ (M_2) 2
01	$A+\bar{B}+C+D$ (M_4) 4	$A+\bar{B}+C+\bar{D}$ (M_5) 5	$A+\bar{B}+\bar{C}+\bar{D}$ (M_7) 7	$A+\bar{B}+\bar{C}+D$ (M_6) 6
11	$\bar{A}+\bar{B}+C+D$ (M_{12}) 12	$\bar{A}+\bar{B}+C+\bar{D}$ (M_{13}) 13	$\bar{A}+\bar{B}+\bar{C}+\bar{D}$ (M_{15}) 15	$\bar{A}+\bar{B}+\bar{C}+D$ (M_{14}) 14
10	$\bar{A}+B+C+D$ (M_8) 8	$\bar{A}+B+C+\bar{D}$ (M_9) 9	$\bar{A}+B+\bar{C}+\bar{D}$ (M_{11}) 11	$\bar{A}+B+\bar{C}+D$ (M_{10}) 10

Minimization of SOP and POS Expressions:-

For reducing the Boolean expressions in SOP (POS) form the following steps are given below

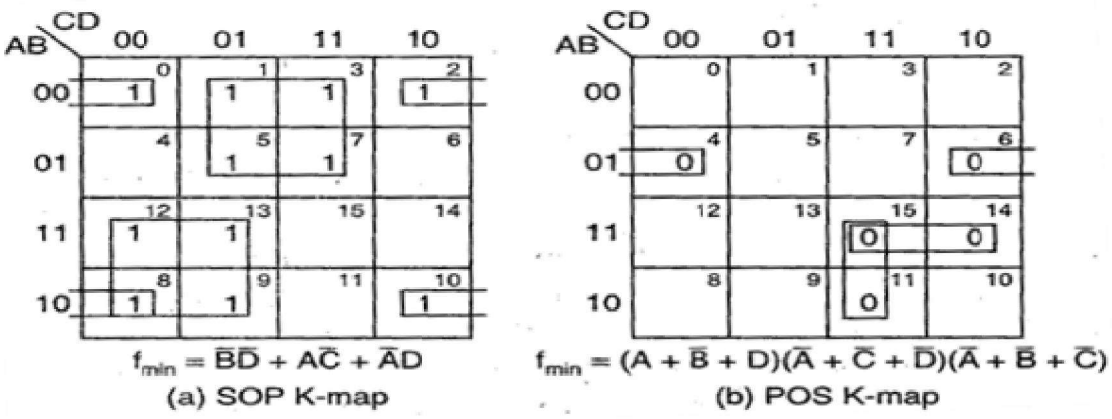
- Draw the K-map and place 1s (0s) corresponding to the minterms (maxterms) of the SOP (POS) expression.
- In the map 1s (0s) which are not adjacent to any other 1(0) are the isolated minterms (maxterms). They are to be read as they are because they cannot be combined even into a 2-square.
- For those 1s (0s) which are adjacent to only one other 1(0) make them pairs (2 squares).
- For quads (4- squares) and octet (8 squares) of adjacent 1s (0s) even if they contain some 1s (0s) which have already been combined. They must geometrically form a square or a rectangle.
- For any 1s (0s) that have not been combined yet then combine them into bigger squares if possible.
- Form the minimal expression by summing (multiplying) the product (sum) terms of all the groups.

Example:-

Reduce using mapping the expression f = ∑ m (0, 1, 2, 3, 5, 7, 8, 9, 10, 12, 13)

Solution:-

The given expression in POS form is f = Π M (4, 6, 11, 14, 15) and in SOP form f = ∑ m (0, 1, 2, 3, 5, 7, 8, 9, 10, 12, 13)



The minimal SOP expression is $f_{min} = \overline{B}\overline{D} + A\overline{C} + \overline{A}D$

The minimal POS expression is $f_{min} = (A + \overline{B} + D)(\overline{A} + \overline{C} + D)(A + B + C)$

DON'T CARE COMBINATIONS:-

The combinations for which the values of the expression are not specified are called don't care combinations or optional combinations and such expression stand incompletely specified. The output is a don't care for these invalid combinations. The don't care terms are denoted by d or X. During the process of designing using SOP maps, each don't care is treated as 1 to reduce the map otherwise it is treated as 0 and left alone. During the process of designing using POS maps, each don't care is treated as 0 to reduce the map otherwise it is treated as 1 and left alone.

A standard SOP expression with don't cares can be converted into standard POS form by keeping the don't cares as they are, and the missing minterms of the SOP form are written as the maxterms of the POS form. Similarly, to convert a standard POS expression with don't cares can be converted into standard SOP form by keeping the don't cares as they are, and the missing maxterms of the POS form are written as the minterms of the SOP form.

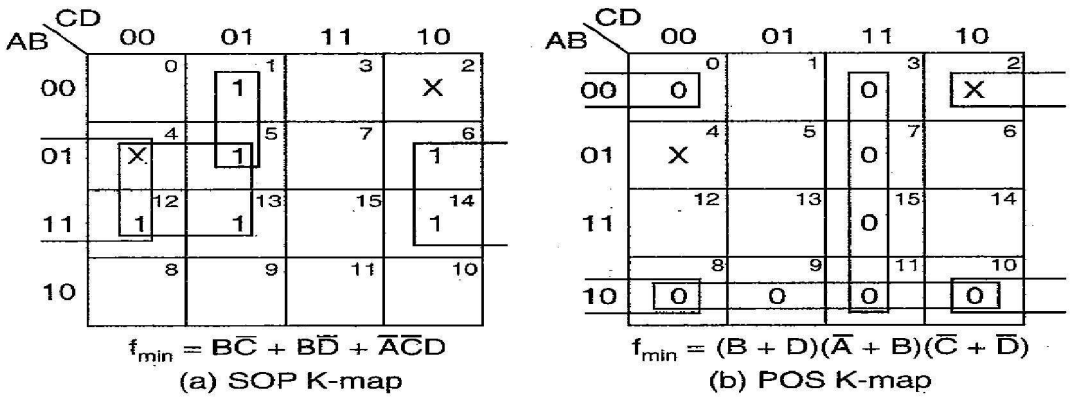
Example:-

Reduce the expression f = ∑ m(1, 5, 6, 12, 13, 14) + d(2, 4) using K- map.

Solution:-

The given expression in SOP form is f = ∑ m (1, 5, 6, 12, 13, 14) + d(2, 4)

The given expression in POS form is f = ΠM (0, 3, 7, 8, 9, 10, 11,15) + d(2, 4)



The minimal of SOP expression is $f_{min} = B\overline{C} + B\overline{D} + \overline{A}C\overline{D}$

The minimal of POS expression is $f_{min} = (B + D)(\overline{A} + B)(\overline{C} + \overline{D})$

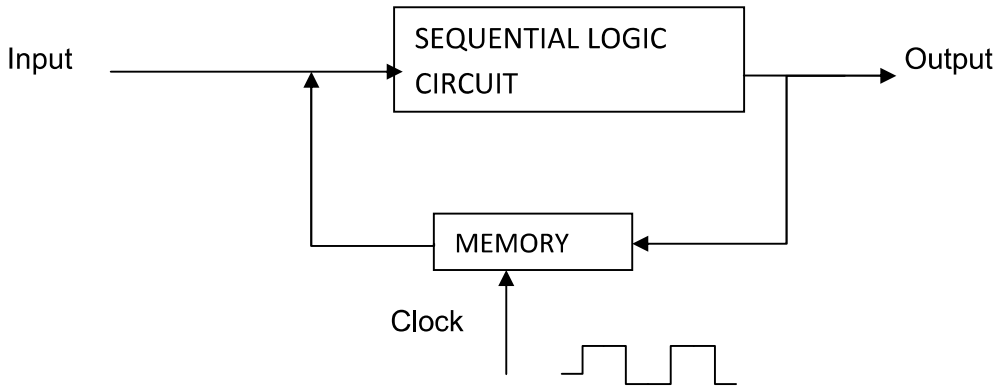
SEQUENTIAL LOGIC CIRCUIT

SEQUENTIAL CIRCUIT:-

- It is a circuit whose output depends upon the present input, previous output and the sequence in which the inputs are applied.

HOW THE SEQUENTIAL CIRCUIT IS DIFFERENT FROM COMBINATIONAL CIRCUIT? :-

- In combinational circuit output depends upon present input at any instant of time and do not use memory. Hence previous input does not have any effect on the circuit. But sequential circuit has memory and depends upon present input and previous output.
- Sequential circuits are slower than combinational circuits and these sequential circuits are harder to design.



[Block diagram of Sequential Logic Circuit]

- The data stored by the memory element at any given instant of time is called the present state of sequential circuit.

TYPES:-

Sequential logic circuits (SLC) are classified as

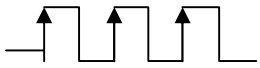
(i) Synchronous SLC
(ii) Asynchronous SLC

- The SLC that are controlled by clock are called synchronous SLC and those which are not controlled by a clock are asynchronous SLC.
- Clock:- A recurring pulse is called a clock.

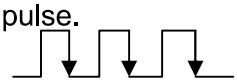
FLIP-FLOP AND LATCH:-

- A flip-flop or latch is a circuit that has two stable states and can be used to store information.
- A flip-flop is a binary storage device capable of storing one bit of information. In a stable state, the output of a flip-flop is either 0 or 1.
- Latch is a non-clocked flip-flop and it is the building block for the flip-flop.
- A storage element in digital circuit can maintain a binary state indefinitely until directed by an input signal to switch state.
- Storage element that operate with signal level are called latches and those operate with clock transition are called as flip-flops.

- The circuit can be made to change state by signals applied to one or more control inputs and will have one or two outputs.
- A flip-flop is called so because its output either flips or flops meaning to switch back and forth.
- A flip-flop is also called a bi-stable multi-vibrator as it has two stable states. The input signals which command the flip-flop to change state are called excitations.
- Flip-flops are storage devices and can store 1 or 0.
- Flip-flops using the clock signal are called clocked flip-flops. Control signals are effective only if they are applied in synchronization with the clock signal.
- Clock-signals may be positive-edge triggered or negative-edge triggered.
- Positive-edge triggered flip-flops are those in which state transitions take place only at positive- going edge of the clock pulse.



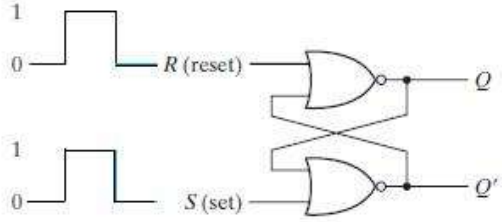
- Negative-edge triggered flip-flops are those in which state transition take place only at negative- going edge of the clock pulse.



- Some common type of flip-flops include
  a) SR (set-reset) F-F
  b) D (data or delay) F-F
  c) T (toggle) F-F and
  d) JK F-F

SR latch:-

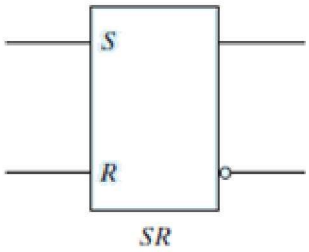
- The SR latch is a circuit with two cross-coupled NOR gates or two cross-coupled NAND gates.
- It has two outputs labeled Q and Q'. Two inputs are there labeled S for set and R foe reset.
- The latch has two useful states. When Q=0 and Q'=1 the condition is called reset state and when Q=1 and Q'=0 the condition is called set state.
- Normally Q and Q' are complement of each other.
- The figure represents a SR latch with two cross-coupled NOR gates. The circuit has NOR gates and as we know if any one of the input for a NOR gate is HIGH then its output will be LOW and if both the inputs are LOW then only the output will be HIGH.



- Under normal conditions, both inputs of the latch remain at 0 unless the state has to be changed. The application of a momentary 1 to the S input causes the latch to go to the set state. The S input must go back to 0 before any other changes take place, in order to avoid the occurrence of an undefined next state that results from the forbidden input condition.
- The first condition (S = 1, R = 0) is the action that must be taken by input S to bring the circuit to the set state. Removing the active input from S leaves the circuit in the same state. After both inputs return to 0, it is then possible to shift to the reset state by momentary applying a 1 to the R input. The 1 can then be removed from R, whereupon the circuit remains in the reset state. When both inputs S and R are equal to 0, the latch can be in either the set or the reset state, depending on which input was most recently a 1.

- If a 1 is applied to both the S and R inputs of the latch, both outputs go to 0. This action produces an undefined next state, because the state that results from the input transitions depends on the order in which they return to 0. It also violates the requirement that outputs be the complement of each other. In normal operation, this condition is avoided by making sure that 1's are not applied to both inputs simultaneously.
- Truth table for SR latch designed with NOR gates is shown below.

Input		Output				Comment
S	R	Q	Q'	Q_{Next}	Q'_{Next}	
0	0	0	1	0	1	No change
0	0	1	0	1	0	
0	1	0	1	0	1	Reset
0	1	1	0	0	1	
1	0	0	1	1	0	Set
1	0	1	0	1	0	
1	1	0	1	X	X	Prohibited state
1	1	1	0	X	X	



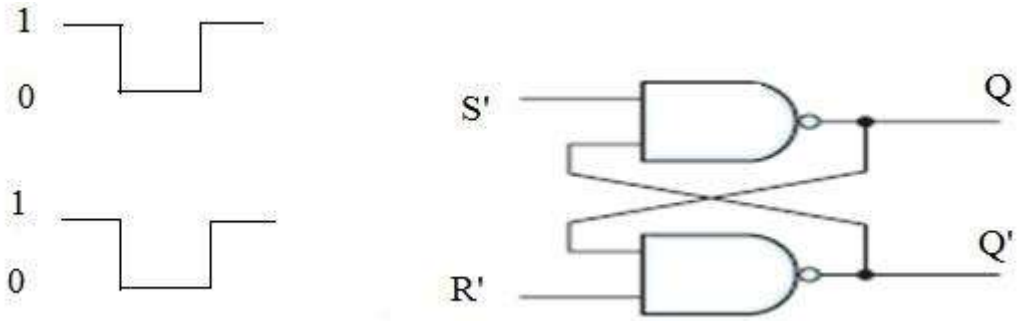
Symbol for SR NOR Latch

Racing Condition:-

In case of a SR latch when S=R=1 input is given both the output will try to become 0. This is called Racing condition.

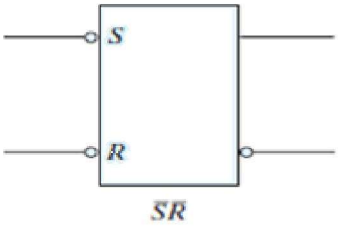
SR latch using NAND gate:-

- The below figure represents a SR latch with two cross-coupled NAND gates. The circuit has NAND gates and as we know if any one of the input for a NAND gate is LOW then its output will be HIGH and if both the inputs are HIGH then only the output will be LOW.
- It operates with both inputs normally at 1, unless the state of the latch has to be changed. The application of 0 to the S input causes output Q to go to 1, putting the latch in the set state. When the S input goes back to 1, the circuit remains in the set state. After both inputs go back to 1, we are allowed to change the state of the latch by placing a 0 in the R input. This action causes the circuit to go to the reset state and stay there even after both inputs return to 1.



- The condition that is forbidden for the NAND latch is both inputs being equal to 0 at the same time, an input combination that should be avoided.

In comparing the NAND with the NOR latch, note that the input signals for the NAND require the complement of those values used for the NOR latch. Because the NAND latch requires a 0 signal to change its state, it is sometimes referred to as an S'R' latch. The primes (or, sometimes, bars over the letters) designate the fact that the inputs must be in their complement form to activate the circuit.



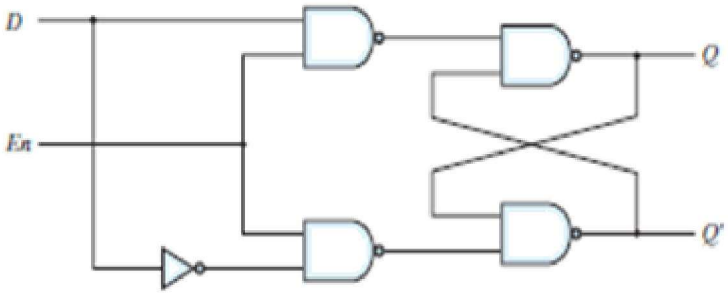
SR

The above represents the symbol for inverted SR latch or SR latch using NAND gate.
Truth table for SR latch using NAND gate or Inverted SR latch

S	R	Q_{next}	Q'_{next}
0	0	Race	Race
0	1	0	1 (Reset)
1	0	1	0 (Set)
1	1	Q (No change)	Q' (No change)

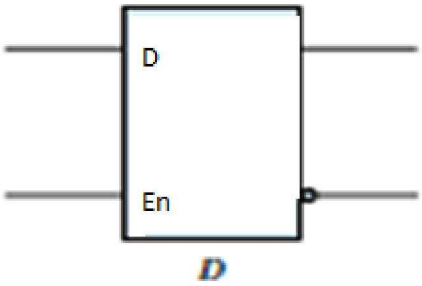
D LATCH:-

- One way to eliminate the undesirable condition of the indeterminate state in the SR latch is to ensure that inputs S and R are never equal to 1 at the same time.



(a) Logic diagram

- This is done in the D latch. This latch has only two inputs: D (data) and En (enable).
- The D input goes directly to the S input, and its complement is applied to the R input.



D

(Symbol for D-Latch)

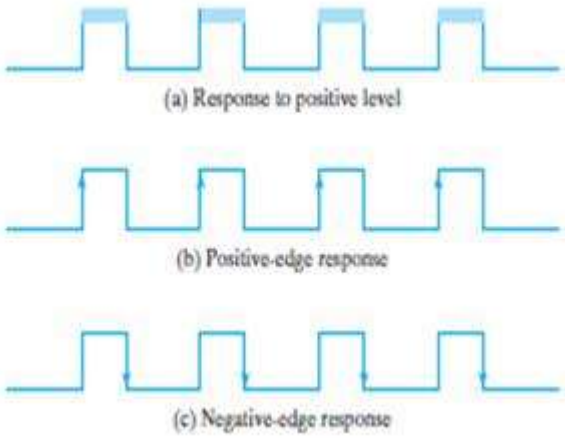
- As long as the enable input is at 0, the cross-coupled SR latch has both inputs at the 1 level and the circuit can't change state regardless of the value of D.
- The below represents the truth table for the D-latch.

En	D	Next State of Q
0	X	No change
1	0	Q=0;Reset State
1	1	Q=1;Set State

- The D input is sampled when En = 1. If D = 1, the Q output goes to 1, placing the circuit in the set state. If D = 0, output Q goes to 0, placing the circuit in the reset state. This situation provides a path from input D to the output, and for this reason, the circuit is often called a TRANSPARENT latch.

TRIGGERING METHODS:-

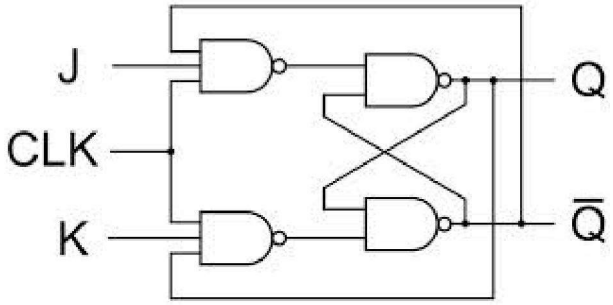
- The state of a latch or flip-flop is switched by a change in the control input. This momentary change is called a trigger, and the transition it causes is said to trigger the flip-flop.
- Flip-flop circuits are constructed in such a way as to make them operate properly when they are part of a sequential circuit that employs a common clock.
- The problem with the latch is that it responds to a change in the level of a clock pulse. For proper operation of a flip-flop it should be triggered only during a signal transition.
- This can be accomplished by eliminating the feedback path that is inherent in the operation of the sequential circuit using latches. A clock pulse goes through two transitions: from 0 to 1 and the return from 1 to 0.
- A ways that a latch can be modified to form a flip-flop is to produce a flip-flop that triggers only during a signal transition (from 0 to 1 or from 1 to 0) of the synchronizing signal (clock) and is disabled during the rest of the clock pulse.



JK FLIP-FLOP:-

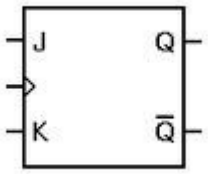
- The JK flip-flop can be constructed by using basic SR latch and a clock. In this case the outputs Q and Q' are returned back and connected to the inputs of NAND gates.
- This simple JK flip Flop is the most widely used of all the flip-flop designs and is considered to be a universal flip-flop circuit.
- The sequential operation of the JK flip flop is exactly the same as for the previous SR flip-flop with the same "Set" and "Reset" inputs.
- The difference this time is that the "JK flip flop" has no invalid or forbidden input states of the SR Latch even when S and R are both at logic "1".

(The below diagram shows the circuit diagram of a JK flip-flop)



- The JK flip flop is basically a gated SR Flip-flop with the addition of a clock input circuitry that prevents the illegal or invalid output condition that can occur when both inputs S and R are equal to logic level "1".
- Due to this additional clocked input, a JK flip-flop has four possible input combinations, "logic 1", "logic 0", "no change" and "toggle".

- The symbol for a JK flip flop is similar to that of an SR bistable latch except the clock input.



(The above diagram shows the symbol of a JK flip-flop.)

- Both the S and the R inputs of the SR bi-stable have now been replaced by two inputs called the J and K inputs, respectively after its inventor Jack and Kilby. Then this equates to: J = S and K = R.
- The two 2-input NAND gates of the gated SR bi-stable have now been replaced by two 3-input NAND gates with the third input of each gate connected to the outputs at Q and Q'.
- This cross coupling of the SR flip-flop allows the previously invalid condition of S = "1" and R = "1" state to be used to produce a "toggle action" as the two inputs are now interlocked.
- If the circuit is now "SET" the J input is inhibited by the "0" status of Q' through the lower NAND gate. If the circuit is "RESET" the K input is inhibited by the "0" status of Q through the upper NAND gate. As Q and Q' are always different we can use them to control the input.

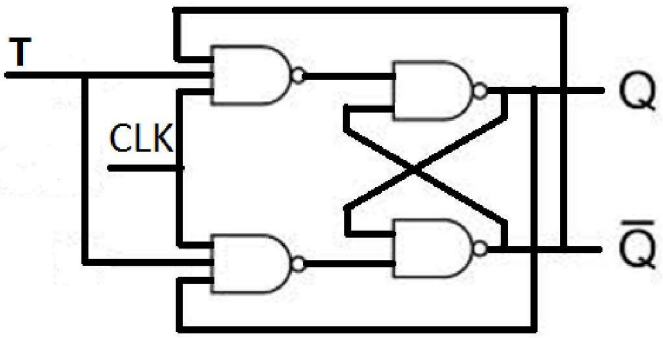
(Truth table for JK flip-flop)

Input		Output		Comment
J	K	Q	Q_{next}	
0	0	0	0	No change
0	0	1	1	
0	1	0	0	Reset
0	1	1	0	
1	0	0	1	Set
1	0	1	1	
1	1	0	1	Toggle
1	1	1	0	

- When both inputs J and K are equal to logic "1", the JK flip flop toggles.

T FLIP-FLOP:-

- Toggle flip-flop or commonly known as T flip-flop.
- This flip-flop has the similar operation as that of the JK flip-flop with both the inputs J and K are shorted i.e. both are given the common input.



- Hence its truth table is same as that of JK flip-flop when J=K= 0 and J=K=1.So its truth table is as follows.

T	Q	Q_{next}	Comment
0	0	0	No change
	1	1	
1	0	1	Toggles
	1	0	

CHARACTERISTIC TABLE:-

- A characteristic table defines the logical properties of a flip-flop by describing its operation in tabular form.
- The next state is defined as a function of the inputs and the present state.
- Q (t) refers to the present state and Q (t + 1) is the next.
- Thus, Q (t) denotes the state of the flip-flop immediately before the clock edge, and Q(t + 1) denotes the state that results from the clock transition.
- The characteristic table for the JK flip-flop shows that the next state is equal to the present state when inputs J and K are both equal to 0. This condition can be expressed as Q (t + 1) = Q (t), indicating that the clock produces no change of state.

Characteristic Table Of JK Flip-Flop

J	K	Q(t+1)
0	0	Q(t) No change
0	1	0 Reset
1	0	1 Set
1	1	Q'(t) Complement

- When K = 1 and J = 0, the clock resets the flip-flop and Q(t + 1) = 0. With J = 1 and K = 0, the flip-flop sets and Q(t + 1) = 1. When both J and K are equal to 1, the next state changes to the complement of the present state, a transition that can be expressed as Q(t + 1) = Q'(t).
- The characteristic equation for JK flip-flop is represented as

$$Q(t+1)= JQ' + K'Q$$

Characteristic Table of D Flip-Flop

D	Q(t+1)
0	0
1	1

- The next state of a D flip-flop is dependent only on the D input and is independent of the present state.
- This can be expressed as Q (t + 1) = D. It means that the next-state value is equal to the value of D. Note that the D flip-flop does not have a "no-change" condition and its characteristic equation is written as Q(t+1)=D.

Characteristic Table of T Flip-Flop

T	Q(t+1)
0	Q(t) No change
1	Q'(t) Complement

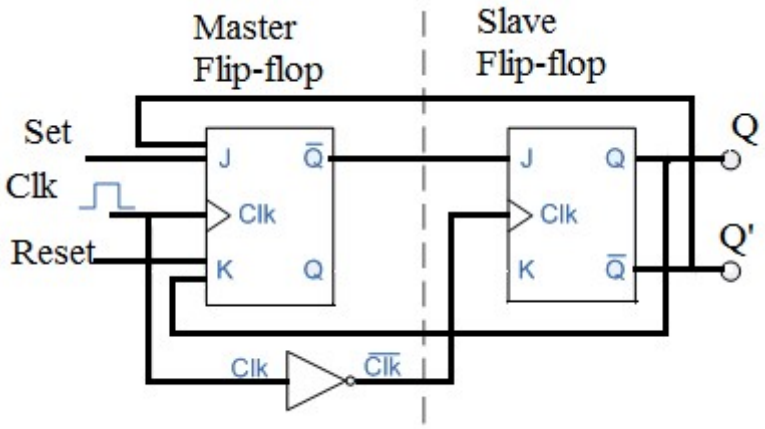
- The characteristic table of T flip-flop has only two conditions: When T = 0, the clock edge does not change the state; when T = 1, the clock edge complements the state of the flip-flop and the characteristic equation is

$$Q(t+1) = T \oplus Q = T'Q + TQ'$$

MASTER-SLAVE JK FLIP-FLOP:-

- The Master-Slave Flip-Flop is basically two gated SR flip-flops connected together in a series configuration with the slave having an inverted clock pulse.
- The outputs from Q and Q' from the "Slave" flip-flop are fed back to the inputs of the "Master" with the outputs of the "Master" flip flop being connected to the two inputs of the "Slave" flip flop.
- This feedback configuration from the slave's output to the master's input gives the characteristic toggle of the JK flip flop as shown below.

The Master-Slave JK Flip Flop



- The input signals J and K are connected to the gated "master" SR flip flop which "locks" the input condition while the clock (Clk) input is "HIGH" at logic level "1".
- As the clock input of the "slave" flip flop is the inverse (complement) of the "master" clock input, the "slave" SR flip flop does not toggle.
- The outputs from the "master" flip flop are only "seen" by the gated "slave" flip flop when the clock input goes "LOW" to logic level "0".
- When the clock is "LOW", the outputs from the "master" flip flop are latched and any additional changes to its inputs are ignored.
- The gated "slave" flip flop now responds to the state of its inputs passed over by the "master" section.
- Then on the "Low-to-High" transition of the clock pulse the inputs of the "master" flip flop are fed through to the gated inputs of the "slave" flip flop and on the "High-to-Low" transition the same inputs are reflected on the output of the "slave" making this type of flip flop edge or pulse-triggered.
- Then, the circuit accepts input data when the clock signal is "HIGH", and passes the data to the output on the falling-edge of the clock signal.
- In other words, the Master-Slave JK Flip flop is a "Synchronous" device as it only passes data with the timing of the clock signal.

FLIP-FLOP CONVERSIONS:-

SR Flip Flop to JK Flip Flop

For this J and K will be given as external inputs to S and R. As shown in the logic diagram below, S and R will be the outputs of the combinational circuit.

The truth tables for the flip flop conversion are given below. The present state is represented by Qp and Qp+1 is the next state to be obtained when the J and K inputs are applied.

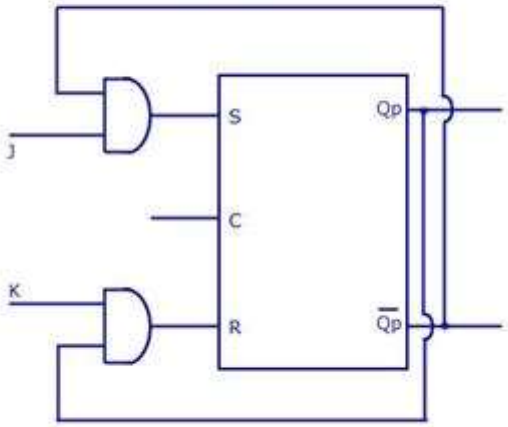
For two inputs J and K, there will be eight possible combinations. For each combination of J, K and Qp, the corresponding Qp+1 states are found. Qp+1 simply suggests the future values to be obtained by the JK flip flop after the value of Qp. The table is then completed by writing the values of S and R required to get each Qp+1 from the corresponding Qp. That is, the values of S and R that are required to change the state of the flip flop from Qp to Qp+1 are written.

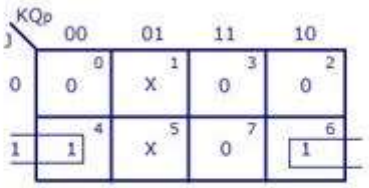
S-R Flip Flop to J-K Flip Flop

Conversion Table

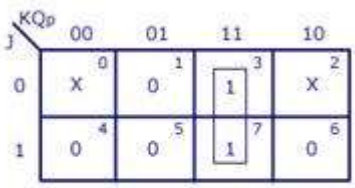
J-K Inputs		Outputs		S-R Inputs	
J	K	Qp	Qp+1	S	R
0	0	0	0	0	X
0	0	1	1	X	0
0	1	0	0	0	X
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	1	X	0
1	1	0	1	1	0
1	1	1	0	0	1

Logic Diagram





$S = \overline{J}Qp$



$R = KQp$

K-Map

JK Flip Flop to SR Flip Flop

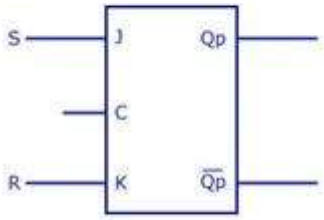
- This will be the reverse process of the above explained conversion. S and R will be the external inputs to J and K. J and K will be the outputs of the combinational circuit. Thus, the values of J and K have to be obtained in terms of S, R and Qp.
- A conversion table is to be written using S, R, Qp, Qp+1, J and K.
- For two inputs, S and R, eight combinations are made. For each combination, the corresponding Qp+1 outputs are found out.
- The outputs for the combinations of S=1 and R=1 are not permitted for an SR flip flop. Thus the outputs are considered invalid and the J and K values are taken as “don’t cares”.

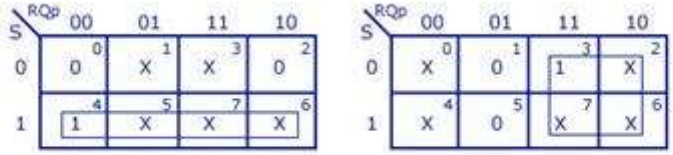
J-K Flip Flop to S-R Flip Flop

Conversion Table

S-R Inputs S	R	Outputs Qp	Qp+1	J-K Inputs J	K
0	0	0	0	0	X
0	0	1	1	X	0
0	1	0	0	0	X
0	1	1	0	X	1
1	0	0	1	1	X
1	0	1	1	X	0
1	1	Invalid		Dont care	
1	1	Invalid		Dont care	

Logic Diagram





SR Flip Flop to D Flip Flop

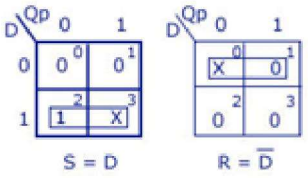
- S and R are the actual inputs of the flip flop and D is the external input of the flip flop.
- The four combinations, the logic diagram, conversion table, and the K-map for S and R in terms of D and Qp are shown below.

S-R Flip Flop to D Flip Flop

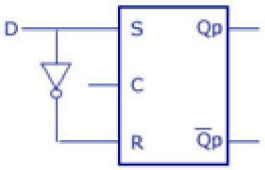
Conversion Table

D Input	Outputs Qp	Qp+1	S-R Inputs S	R
0	0	0	0	X
0	1	0	0	1
1	0	1	1	0
1	1	1	X	0

K-maps



Logic Diagram



D Flip Flop to SR Flip Flop

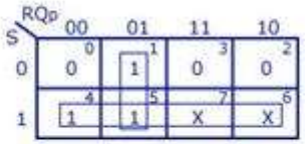
- D is the actual input of the flip flop and S and R are the external inputs. Eight possible combinations are achieved from the external inputs S, R and Qp.
- But, since the combination of S=1 and R=1 are invalid, the values of Qp+1 and D are considered as “don’t cares”.
- The logic diagram showing the conversion from D to SR, and the K-map for D in terms of S, R and Qp are shown below.

D Flip Flop to S-R Flip Flop

Conversion Table

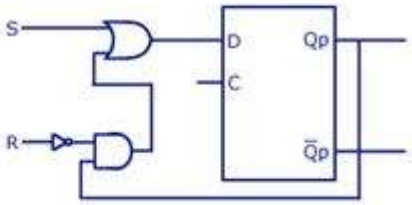
S-R Inputs S	R	Outputs Qp	Qp+1	D Input
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1
1	1	Invalid		Don't care
1	1	Invalid		Don't care

K-map



$D = S + \bar{R}Q_n$

Logic Diagram



JK Flip Flop to T Flip Flop:-

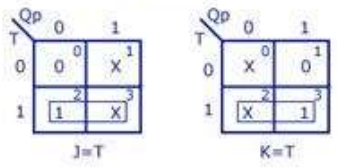
- J and K are the actual inputs of the flip flop and T is taken as the external input for conversion
- Four combinations are produced with T and Qp. J and K are expressed in terms of T and Qp.
- The conversion table, K-maps, and the logic diagram are given below.

J-K Flip Flop to T Flip Flop

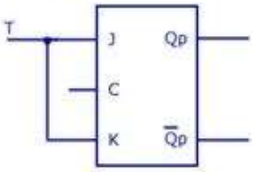
Conversion Table

T Input	Outputs Qp	Qp+1	J-K Inputs J	K
0	0	0	0	X
0	1	1	X	0
1	0	1	1	X
1	1	0	X	1

K-maps



Logic Diagram



D Flip Flop to JK Flip Flop:-

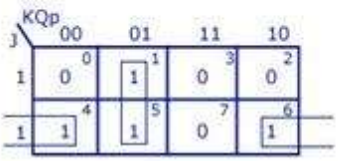
- In this conversion, D is the actual input to the flip flop and J and K are the external inputs.
- J, K and Qp make eight possible combinations, as shown in the conversion table below. D is expressed in terms of J, K and Qp.
- The conversion table, the K-map for D in terms of J, K and Qp and the logic diagram showing the conversion from D to JK are given in the figure below.

D Flip Flop to J-K Flip Flop

Conversion Table

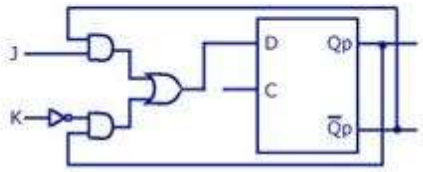
J-K Input J	K	Outputs Qp	Qp+1	D Input
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	0	0

K-map



$D = J\overline{Qp} + \overline{K}Qp$

Logic Diagram



JK Flip Flop to D Flip Flop:-

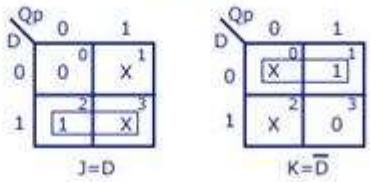
- D is the external input and J and K are the actual inputs of the flip flop. D and Qp make four combinations. J and K are expressed in terms of D and Qp.
- The four combination conversion table, the K-maps for J and K in terms of D and Qp.

J-K Flip Flop to D Flip Flop

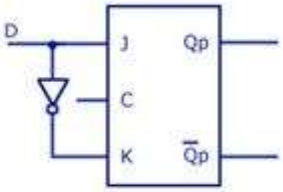
Conversion Table

D Input	Outputs Qp	Qp+1	J-K Inputs J	K
0	0	0	0	X
0	1	0	X	1
1	0	1	1	X
1	1	0	X	0

K-maps



Logic Diagram



COMBINATIONAL LOGIC CIRCUIT

- A combinational circuit consists of logic gates whose outputs at any time are determined from only the present combination of inputs.
- A combinational circuit performs an operation that can be specified logically by a set of Boolean functions.
- It consists of an interconnection of logic gates. Combinational logic gates react to the values of the signals at their inputs and produce the value of the output signal, transforming binary information from the given input data to a required output data.
- A block diagram of a combinational circuit is shown in the below figure.
- The n input binary variables come from an external source; the m output variables are produced by the internal combinational logic circuit and go to an external destination.
- Each input and output variable exists physically as an analog signal whose values are interpreted to be a binary signal that represents logic 1and logic 0.

BINARY ADDER–SUBTRACTOR:-

- Digital computers perform a variety of information-processing tasks. Among the functions encountered are the various arithmetic operations.
- The most basic arithmetic operation is the addition of two binary digits. This simple addition consists of four possible elementary operations: 0 + 0 = 0, 0 + 1 = 1, 1 + 0 = 1, and 1 + 1 = 10.
- The first three operations produce a sum of one digit, but when both augend and addend bits are equal to 1; the binary sum consists of two digits. The higher significant bit of this result is called a carry.
- When the augend and addend numbers contain more significant digits, the carry obtained from the addition of two bits is added to the next higher order pair of significant bits.
- A combinational circuit that performs the addition of two bits is called a half adder.
- One that performs the addition of three bits (two significant bits and a previous carry) is a full adder. The names of the circuits stem from the fact that two half adders can be employed to implement a full adder.

HALF ADDER:-

- This circuit needs two binary inputs and two binary outputs.
- The input variables designate the augend and addend bits; the output variables produce the sum and carry. Symbols x and y are assigned to the two inputs and S (for sum) and C (for carry) to the outputs.
- The truth table for the half adder is listed in the below table.
- The C output is 1 only when both inputs are 1. The S output represents the least significant bit of the sum.
- The simplified Boolean functions for the two outputs can be obtained directly from the truth table.

x	y	D	B
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

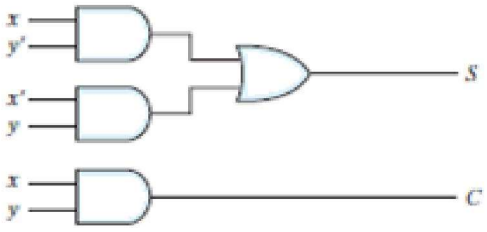
Truth Table

- The simplified sum-of-products expressions are

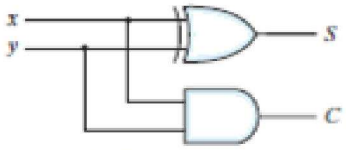
$$S = x'y + xy'$$

$$C = xy$$

- The logic diagram of the half adder implemented in sum of products is shown in the below figure. It can be also implemented with an exclusive-OR and an AND gate.



(a) $S = xy' + x'y$
$C = xy$



(b) $S = x \oplus y$
$C = xy$

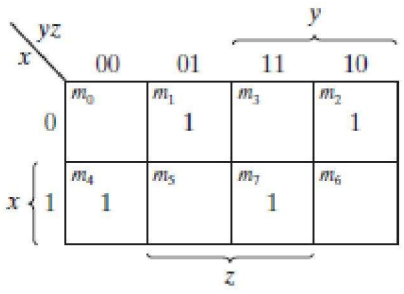
FULL ADDER:-

- A full adder is a combinational circuit that forms the arithmetic sum of three bits.
- It consists of three inputs and two outputs. Two of the input variables, denoted by x and y , represent the two significant bits to be added. The third input, z , represents the carry from the previous lower significant position.

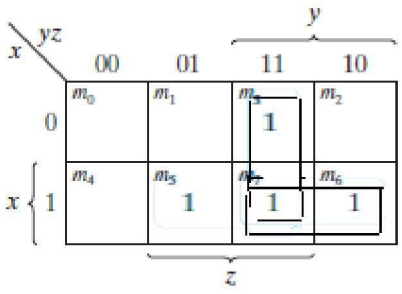
x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Truth Table

- Two outputs are necessary because the arithmetic sum of three binary digits ranges in value from 0 to 3, and binary representation of 2 or 3 needs two bits. The two outputs are designated by the symbols S for sum and C for carry.



(a) $S = x'y'z + x'yz' + xy'z' + xyz$



(b) $C = xy + xz + yz$

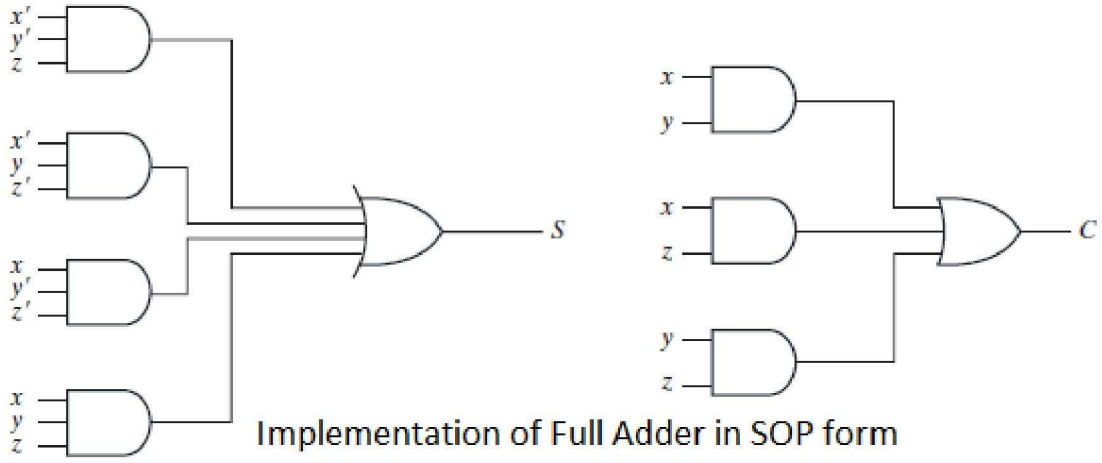
K-Map for full adder

- The binary variable S gives the value of the least significant bit of the sum. The binary variable C gives the output carry formed by adding the input carry and the bits of the words.
- The eight rows under the input variables designate all possible combinations of the three variables. The output variables are determined from the arithmetic sum of the input bits. When all input bits are 0, the output is 0.
- The S output is equal to 1 when only one input is equal to 1 or when all three inputs are equal to 1. The C output has a carry of 1 if two or three inputs are equal to 1.
- The simplified expressions are

S = x'y'z + x'yz' + xy'z' + xyz

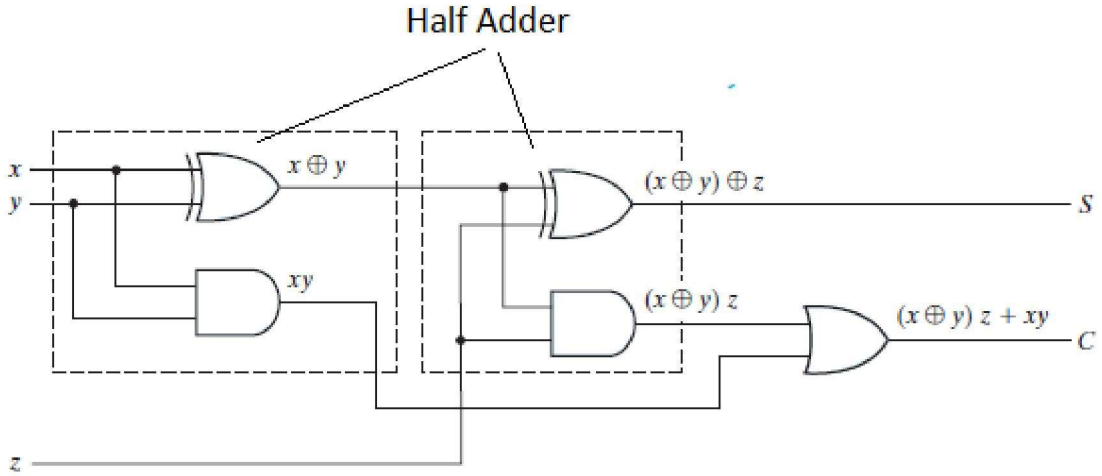
C = xy + xz + yz

- The logic diagram for the full adder implemented in sum-of-products form is shown in figure.



Implementation of Full Adder in SOP form

- It can also be implemented with two half adders and one OR gate as shown in the figure.



Implementation of Full Adder using Two Half Adders and an OR gate

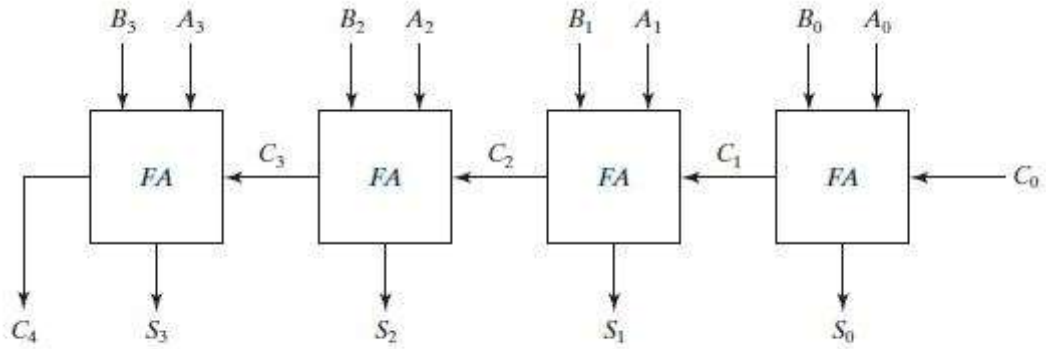
- A full adder is a combinational circuit that forms the arithmetic sum of three bits.

BINARY ADDER:-

- A binary adder is a digital circuit that produces the arithmetic sum of two binary numbers.
- It can be constructed with full adders connected in cascade, with the output carry from each full adder connected to the input carry of the next full adder in the chain.
- Addition of n-bit numbers requires a chain of n full adders or a chain of one-half adder and n-1 full adders. In the former case, the input carry to the least significant position is fixed at 0.
- The interconnection of four full-adder (FA) circuits to provide a four-bit binary ripple carry adder is shown in the figure.
- The augend bits of A and the addend bits of B are designated by subscript numbers from right to left, with subscript 0 denoting the least significant bit.
- The carries are connected in a chain through the full adders. The input carry to the adder is C0, and it ripples through the full adders to the output carry C4. The S outputs generate the required sum bits.
- An n -bit adder requires n full adders, with each output carry connected to the input carry of the next higher order full adder.
- Consider the two binary numbers A = 1011 and B = 0011. Their sum S = 1110 is formed with the four-bit adder as follows:

Subscript *i*:	3	2	1	0	
Input carry	0	1	1	0	C_i
Augend	1	0	1	1	A_i
Addend	0	0	1	1	B_i
Sum	1	1	1	0	S_i
Output carry	0	0	1	1	C_{i+1}

- The bits are added with full adders, starting from the least significant position (subscript 0), to form the sum bit and carry bit. The input carry C_0 in the least significant position must be 0.
- The value of C_{i+1} in a given significant position is the output carry of the full adder. This value is transferred into the input carry of the full adder that adds the bits one higher significant position to the left.
- The sum bits are thus generated starting from the rightmost position and are available as soon as the corresponding previous carry bit is generated. All the carries must be generated for the correct sum bits to appear at the outputs.



Four Bit Binary Adder

HALF SUBTRACTOR:-

- This circuit needs two binary inputs and two binary outputs.
- Symbols x and y are assigned to the two inputs and D (for difference) and B (for borrow) to the outputs.
- The truth table for the half subtractor is listed in the below table.

x	y	D	B
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

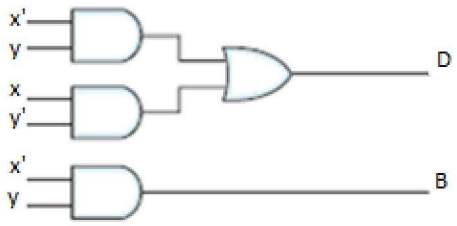
Truth Table

- The B output is 1 only when the inputs are 0 and 1. The D output represents the least significant bit of the subtraction.
- The subtraction operation is done by using the following rules as

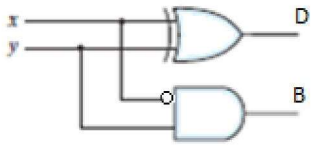
  0-0=0;
  0-1=1 with borrow 1;
  1-0=1;
  1-1=0.

- The simplified Boolean functions for the two outputs can be obtained directly from the truth table. The simplified sum-of-products expressions are

  D = x'y + xy' and B =x'y



D = x'y+xy'
B = x'y



D = x ⊕ y
B=x'y

- The logic diagram of the half adder implemented in sum of products is shown in the figure. It can be also implemented with an exclusive-OR and an AND gate with one inverted input.

FULL SUBTRACTOR:-

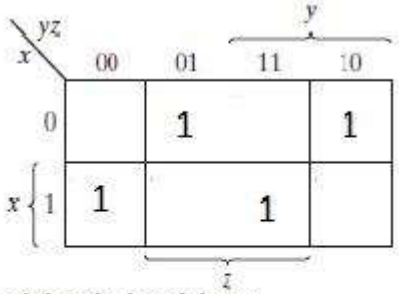
- A full subtractor is a combinational circuit that forms the arithmetic subtraction operation of three bits.
- It consists of three inputs and two outputs. Two of the input variables, denoted by x and y , represent the two significant bits to be subtracted. The third input, z , is subtracted from the result 0f the first subtraction.

x	y	z	D	B
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

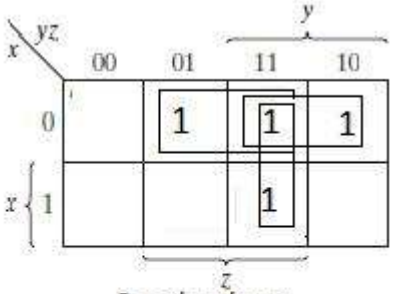
Truth Table

- Two outputs are necessary because the arithmetic subtraction of three binary digits ranges in value from 0 to 3, and binary representation of 2 or 3 needs two bits. The two outputs are designated by the symbols D for difference and B for borrow.

- The binary variable D gives the value of the least significant bit of the difference. The binary variable B gives the output borrow formed during the subtraction process.



D = x'y'z+x'yz'+xy'z'+xyz



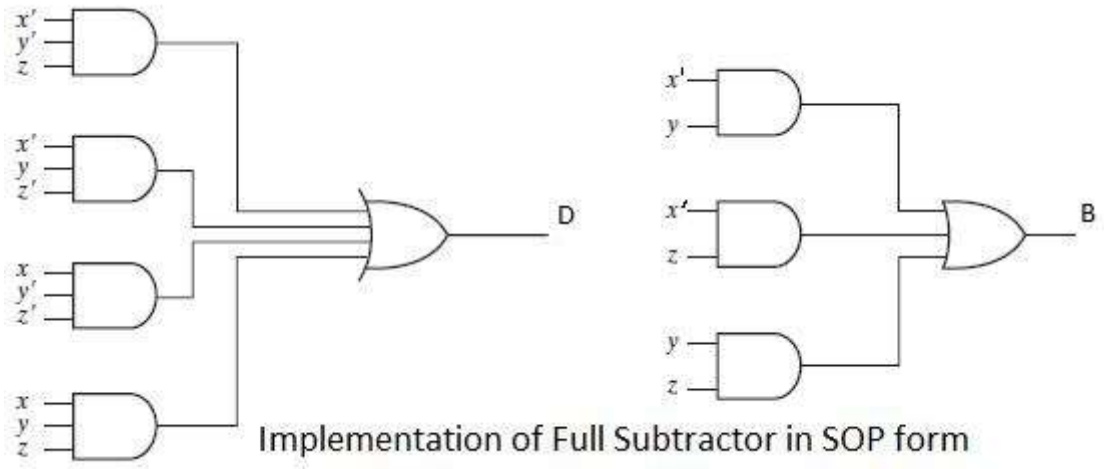
B = x'z+x'y+yz

K-Map for full Subtractor

- The eight rows under the input variables designate all possible combinations of the three variables. The output variables are determined from the arithmetic subtraction of the input bits.
- The difference D becomes 1 when any one of the input is 1or all three inputs are equal to1 and the borrow B is 1 when the input combination is (0 0 1) or (0 1 0) or (0 1 1) or (1 1 1).
- The simplified expressions are

$$D = x'y'z + x'yz' + xy'z' + xyz$$
$$B = x'z + x'y + yz$$

- The logic diagram for the full adder implemented in sum-of-products form is shown in figure.



Implementation of Full Subtractor in SOP form

MAGNITUDE COMPARATOR:-

- A magnitude comparator is a combinational circuit that compares two numbers A and B and determines their relative magnitudes.
- The following description is about a 2-bit magnitude comparator circuit.
- The outcome of the comparison is specified by three binary variables that indicate whether A < B, A = B, or A > B.
- Consider two numbers, A and B, with two digits each. Now writing the coefficients of the numbers in descending order of significance:

$$A = A_1\ A_0$$
$$B = B_1\ B_0$$

- The two numbers are equal if all pairs of significant digits are equal i.e. if and only if A1 = B1, and A0 = B0.
- When the numbers are binary, the digits are either 1 or 0, and the equality of each pair of bits can be expressed logically with an exclusive-NOR function as

$$x1=A_1B_1+A_1'B_1'$$

And $x0=A_0B_0+A_0'B_0'$

- The equality of the two numbers A and B is displayed in a combinational circuit by an output binary variable that we designate by the symbol (A = B).
- This binary variable is equal to 1 if the input numbers, A and B , are equal, and is equal to 0 otherwise.
- For equality to exist, all xi variables must be equal to 1, a condition that dictates an AND operation of all variables:

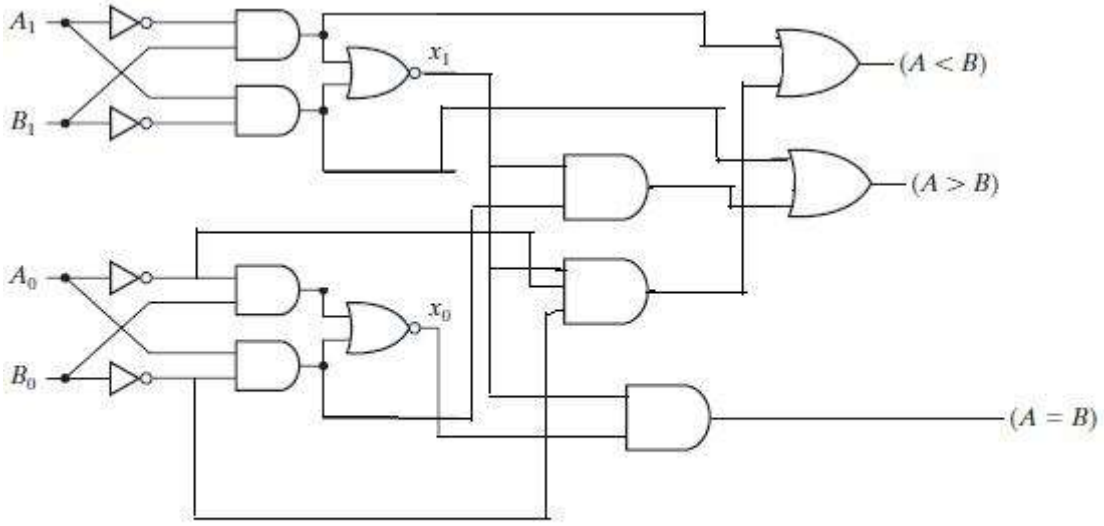
$$(A = B) = x_1x_0$$

- The binary variable (A = B) is equal to 1 only if all pairs of digits of the two numbers are equal.
- To determine whether A is greater or less than B, we inspect the relative magnitudes of pairs of significant digits, starting from the most significant position. If the two digits of a pair are equal, we compare the next lower significant pair of digits. If the corresponding digit of A is 1 and that of B is 0, we conclude that A > B. If the corresponding digit of A is 0 and that of B is 1, we have A < B. The sequential comparison can be expressed logically by the two Boolean functions

$$(A > B) = A_1B_1'+x_1A_0B'_0$$
$$(A < B) = A_1'B_1+x_1A_0'B_0'$$

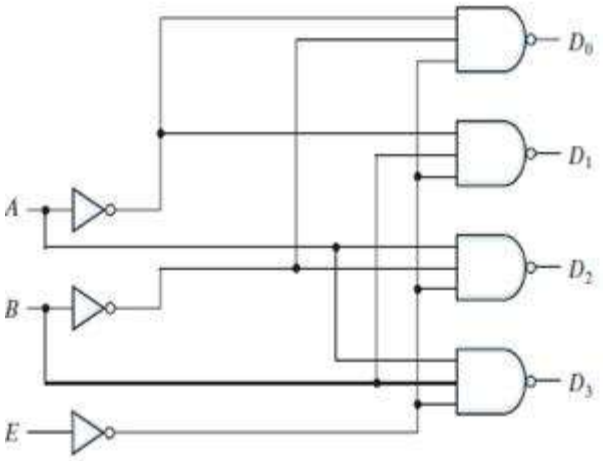
A_1	A_0	B_1	B_0	A>B	A<B	A=B
0	0	0	0	0	0	1
0	0	0	1	0	1	0
0	0	1	0	0	1	0
0	0	1	1	0	1	0
0	1	0	0	1	0	0
0	1	0	1	0	0	1
0	1	1	0	0	1	0
0	1	1	1	0	1	0
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	0	1
1	0	1	1	0	1	0
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	0	1

Truth Table



Logic Diagram of 2-bit Magnitude Comparator

DECODER:-



E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	1

- A decoder is a combinational circuit that converts binary information from n input lines to a maximum of 2^n unique output lines.
- If the n -bit coded information has unused combinations, the decoder may have fewer than 2n outputs.
- The decoders presented here are called n -to- m -line decoders, where m … 2n.
- Their purpose is to generate the 2n (or fewer) minterms of n input variables.
- Each combination of inputs will assert a unique output. The name decoder is also used in conjunction with other code converters, such as a BCD-to-seven-segment decoder.
- Consider the three-to-eight-line decoder circuit of three inputs are decoded into eight outputs, each representing one of the minterms of the three input variables.
- The three inverters provide the complement of the inputs, and each one of the eight AND gates generates one of the minterms.
- The input variables represent a binary number, and the outputs represent the eight digits of a number in the octal number system.
- However, a three-to-eight-line decoder can be used for decoding any three-bit code to provide eight outputs, one for each element of the code.
- A two-to-four-line decoder with an enable input constructed with NAND gates is shown in Fig.
- The circuit operates with complemented outputs and a complement enable input. The decoder is enabled when E is equal to 0 (i.e., active-low enable). As indicated by the truth table, only one output can be equal to 0 at any given time; all other outputs are equal to 1.
- The output whose value is equal to 0 represents the minterm selected by inputs A and B.
- The circuit is disabled when E is equal to 1, regardless of the values of the other two inputs.
- When the circuit is disabled, none of the outputs are equal to 0 and none of the minterms are selected.
- In general, a decoder may operate with complemented or un-complemented outputs.
- The enable input may be activated with a 0 or with a 1 signal.
- Some decoders have two or more enable inputs that must satisfy a given logic condition in order to enable the circuit.
- A decoder with enable input can function as a demultiplexer— a circuit that receives information from a single line and directs it to one of 2n possible output lines.
- The selection of a specific output is controlled by the bit combination of n selection lines.
- The decoder of Fig. can function as a one-to-four-line demultiplexer when E is taken as a data input line and A and B are taken as the selection inputs.
- The single input variable E has a path to all four outputs, but the input information is directed to only one of the output lines, as specified by the binary combination of the two selection lines A and B .
- This feature can be verified from the truth table of the circuit.
- For example, if the selection lines AB = 10, output D_2 will be the same as the input value E, while all other outputs are maintained at 1.
- Since decoder and demultiplexer operations are obtained from the same circuit, a decoder with an enable input is referred to as a decoder – demultiplexer.
- A application of this decoder is binary-to-octal conversion.

ENCODER:-

- An encoder is a digital circuit that performs the inverse operation of a decoder.
- An encoder has 2n (or fewer) input lines and n output lines.
- The output lines, as an aggregate, generate the binary code corresponding to the input value.

Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

- The above Encoder has eight inputs (one for each of the octal digits) and three outputs that generate the corresponding binary number.
- It is assumed that only one input has a value of 1 at any given time.
- The encoder can be implemented with OR gates whose inputs are determined directly from the truth table.
- Output z is equal to 1 when the input octal digit is 1, 3, 5, or 7.
- Output y is 1 for octal digits 2, 3, 6, or 7, and output x is 1 for digits 4, 5, 6, or 7.
- These conditions can be expressed by the following Boolean output functions:

$$z = D_1 + D_3 + D_5 + D_7$$
$$y = D_2 + D_3 + D_6 + D_7$$
$$x = D_4 + D_5 + D_6 + D_7$$

- The encoder can be implemented with three OR gates.
- The encoder defined above has the limitation that only one input can be active at any given time.
- If two inputs are active simultaneously, the output produces an undefined combination.
- To resolve this ambiguity, encoder circuits must establish an input priority to ensure that only one input is encoded which is done in the Priority Encoder .

PRIORITY ENCODER:-

- A priority encoder is an encoder circuit that includes the priority function.
- The operation of the priority encoder is such that if two or more inputs are equal to 1 at the same time, the input having the highest priority will take precedence.

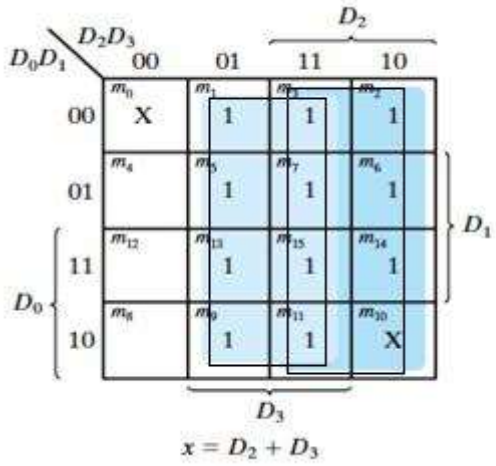
Inputs				Outputs		
D_0	D_1	D_2	D_3	x	y	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

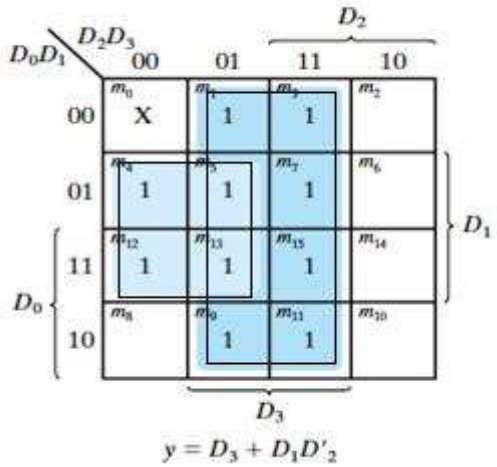
- In addition to the two outputs x and y , the circuit has a third output designated by V ; this is a valid bit indicator that is set to 1 when one or more inputs are equal to 1.

- If all inputs are 0, there is no valid input and V is equal to 0.
- The other two outputs are not inspected when V equals 0 and are specified as don't-care conditions.
- Here X 's in output columns represent don't-care conditions, the X 's in the input columns are useful for representing a truth table in condensed form.

Inputs				Outputs		
D_0	D_1	D_2	D_3	x	y	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

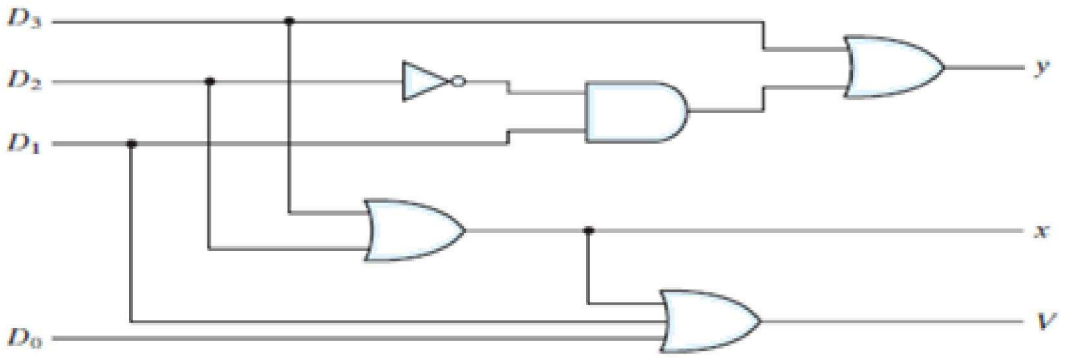
- Higher the subscript number, the higher the priority of the input.
- Input D3 has the highest priority, so, regardless of the values of the other inputs, when this input is 1, the output for xy is 11 (binary 3).
- If D2 = 1, provided that D3 = 0, regardless of the values of the other two lower priority inputs the output is 10.
- The output for D1 is generated only if higher priority inputs are 0, and so on down the priority levels.





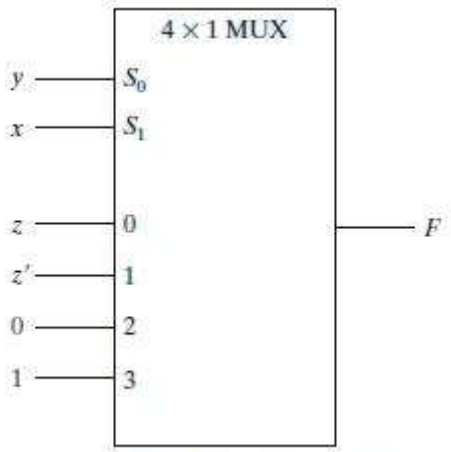
- The maps for simplifying outputs x and y are shown in above Fig.
- The minterms for the two functions are derived from its truth table.
- Although the table has only five rows, when each X in a row is replaced first by 0 and then by 1, we obtain all 16 possible input combinations.
- For example, the fourth row in the table, with inputs XX10, represents the four minterms 0010, 0110, 1010, and 1110. The simplified Boolean expressions for the priority encoder are obtained from the maps.
- The condition for output V is an OR function of all the input variables.
- The priority encoder is implemented according to the following Boolean functions:

$$x = D_2 + D_3$$
$$y = D_3 + D_1 D'_2$$
$$V = D_0 + D_1 + D_2 + D_3$$

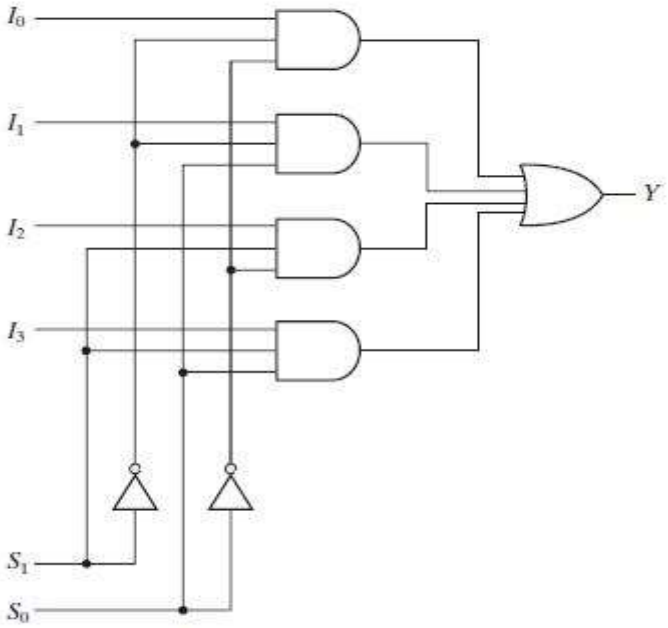


MULTIPLEXER:-

- A multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line.
- The selection of a particular input line is controlled by a set of selection lines.
- Normally, there are 2^n input lines and n selection lines whose bit combinations determine which input is selected.
- A four-to-one-line multiplexer is shown in the below figure. Each of the four inputs, I_0 through I_3, is applied to one input of an AND gate.
- Selection lines S_1 and S_0 are decoded to select a particular AND gate. The outputs of the AND gates are applied to a single OR gate that provides the one-line output.
- The function table lists the input that is passed to the output for each combination of the binary selection values.
- To demonstrate the operation of the circuit, consider the case when
  S_1S_0= 10.
- The AND gate associated with input I_2 has two of its inputs equal to 1 and the third input connected to I_2.
- The other three AND gates have at least one input equal to 0, which makes their outputs equal to 0. The output of the OR gate is now equal to the value of I_2, providing a path from the selected input to the output.
- A multiplexer is also called a data selector, since it selects one of many inputs and steers the binary information to the output line.



(b) Multiplexer implementation



Logic diagram

S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

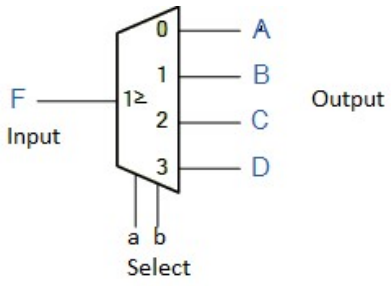
Truth table

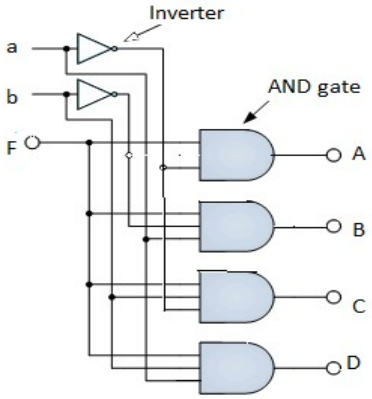
DEMULTIPLEXER:-

- The data distributor, known more commonly as a Demultiplexer or “Demux” for short, is the exact opposite of the Multiplexer.
- The demultiplexer takes one single input data line and then switches it to any one of a number of individual output lines one at a time. The demultiplexer converts a serial data signal at the input to a parallel data at its output lines as shown below.
- The Boolean expression for this 1-to-4 demultiplexer above with outputs A to D and data select lines a, b is given as:

$$F = (ab)'A + a'bB + ab'C + abD$$

- The function of the demultiplexer is to switch one common data input line to any one of the 4 output data lines A to D in our example above. As with the multiplexer the individual solid state switches are selected by the binary input address code on the output select pins “a” and “b” as shown.





Logic Diagram

- Unlike multiplexers which convert data from a single data line to multiple lines and demultiplexers which convert multiple lines to a single data line, there are devices available which convert data to and from multiple lines and in the next tutorial about combinational logic devices.
- Standard demultiplexer IC packages available are the TTL 74LS138 1 to 8-output demultiplexer, the TTL 74LS139 Dual 1-to-4 output demultiplexer or the CMOS CD4514 1-to-16 output demultiplexer.

Output Select		Data output Selected
b	a	
0	0	A
0	1	B
1	0	C
1	1	D

Truth Table