NILASAILA INSTITUTE OF SCIENCE
&
TECHNOLOGY

SERGARH-756060, BALASORE (ODISHA)

(Approved by AICTE & affiliated to SCTE&VT,Odisha)

3RD

DIGITAL ELECTRONICS & MICROPROCESSOR

PREPARED BY-ER.BISWAJIT PARIDA

CONTENTS

(CHAPTER-1)

BASICS OF DIGITAL ELECTRONICS

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

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

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.
- The value attached to the symbol depends on its location with respect to the decimal point.

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.

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)**

(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 1(

Group of 4 bits are 0010 1101 1(

Convert each group into hex = 2 D

(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 $(\mathbf{2FB.7C})_{16}$

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

Ans 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	

Ans 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

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

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

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

2. 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	

Ans 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:

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

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

= $(2095.0937)_{10}$

Ans 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

Ans 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 bit binary = 0011 1010 1001 1110 . 1011 0000 1101 digit to 4

Ans 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:

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

= $40960 + 0 + 240 + 9 + 0 + 0.0546 + 0.0026$

= $(41209.0572)_{10}$

Ans 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	

Ans 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 :-

$$\begin{array}{rr} & 1\,0\,0\,1\,0\,1 \\ + & \underline{1\,1\,0\,1\,1\,1\,1} \\ & \underline{1\,0\,0\,1\,0\,1\,0\,0} \end{array}$$

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 :-

$$\begin{array}{rr} & 1010.010 \\ - & \underline{\ \ 111.111} \\ & \underline{0010.011} \end{array}$$

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 :-

$$\begin{array}{rr} & 1101 \\ \times & \underline{\ \ 110} \\ & 0000 \\ & 1101 \\ + & \underline{1101\ \ \ \ } \\ & \underline{1001110\ \ } \end{array}$$

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

$0 \div 1 = 0$

$1 \div 1 = 1$

For example :-

Divide $(10110)_2$ by $(110)_2$. Solution

:-

```
110 ) 101101 ( 111.1
  -  110
      1010
      110
       1001
       110
        110
        110
        000
```

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

Ans 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				01	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.

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

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.Theyare also used in shaft encoders, I/O devices, A/D converters and other peripheral equipmen

BINARY- TO – GRAY CONVERSION:-

If an n-bit binary number is represented by B_n B_{n-1} - - - - - B1 and its gray code equivalent by G_n G_{n-1}----------G1, 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

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

.
.
.

$G_1 = B_2 \oplus B_1$

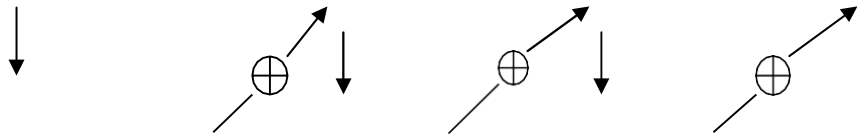
Where the symbol $\oplus$ stands for Exclusive OR (X-OR)

For example :-

Convert the binary 1001 to the Gray code.

Solution :-`

- Shift registers are used for converting serial data to parallel form, so that a serial input can be processed by a parallel system and for converting parallel data to serial form, so that parallel data can be transmitted serially.
- A serial in, parallel out shift register can be used to perform serial-to parallel conversion, and a parallel in, serial out shift register can be used to perform parallel- to –serial conversion.
- A universal shift register can be used to perform both the serial- to – parallel and parallel-to- serial data conversion.
- A bidirectional shift register can be used to reverse the order of data.



Gray → **1** **1** **0** **1**

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}---------- B1, then binary bits are obtained from Gray bits as follows : B_n = G_n

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

.
.
.

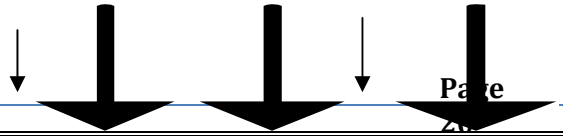
$B_1 = B_2 \oplus G_1$

For example :-

Convert the Gray code 1101 to the binary.

Solution :-

Gray → **1** **1** **0** **1**



Binary→ **1** **0** **0** **1**

The binary code is **100**

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.

DIFFERENT TYPES OF LOGIC GATES:-

AND GATE
OR GATE
NOT GATE
NAND GATE
NOR GATE
EX-OR GATE
EX NOR GATE

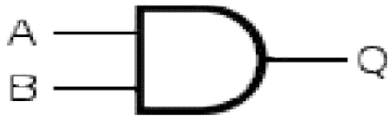
AND GATE:-

- 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

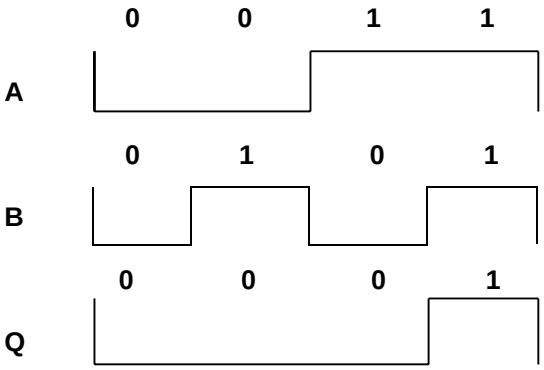
Logic Symbol



Truth Table

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

Timing Diagram



OR GATE:-

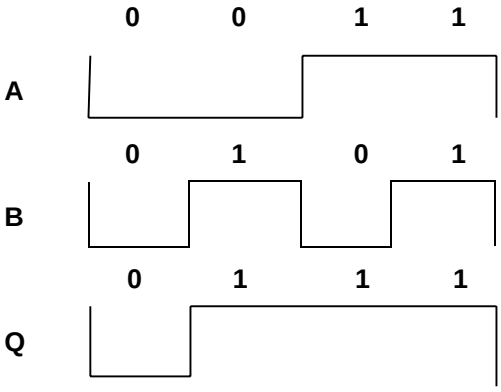
- An OR gate may have two or more inputs but only one output.
- The output is logic 1 state, even if one of its input is in logic 1 state.
- The output is logic 0 state, only when each one of its inputs is in logic state.

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

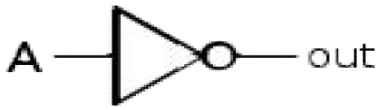


NOT GATE (INVERTER):-

- 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

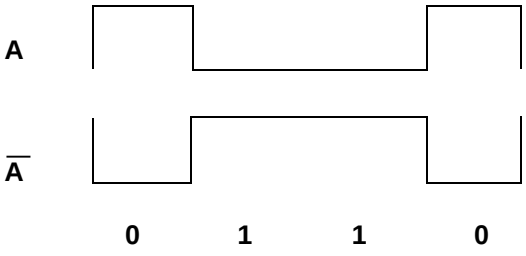


Truth table

INPUT A	OUTPUT A
0	**1**
1	0

Timing Diagram

1 0 0 1



0 1 1 0

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.

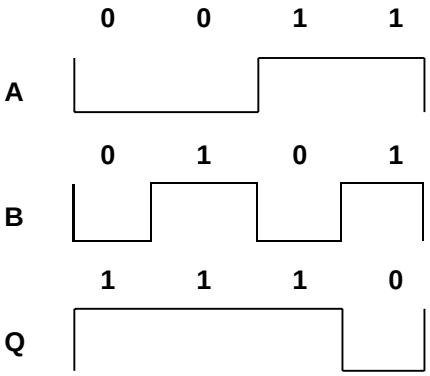
IC No.:- 7400 two input NAND gate 7410
three input NAND gate 7420
four input NAND gate 7430
eight input NAND gate

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.

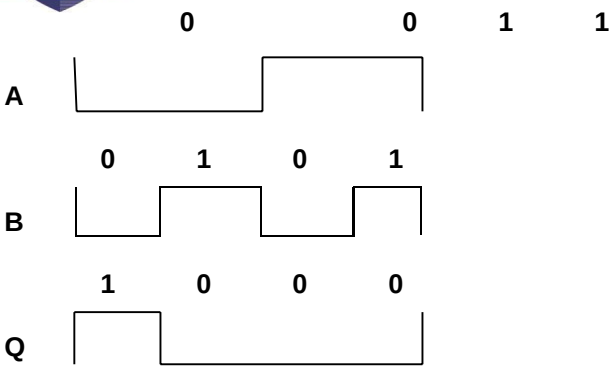
IC No.:- 7402 two input NOR gate 7427
three input NOR gate 7425
four input NOR gate

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

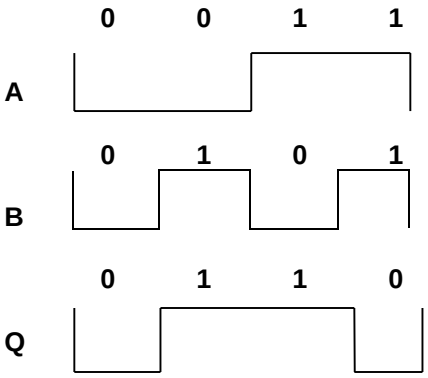
is **Q** = $A \oplus B$

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

Truth Table

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

Timing Diagram

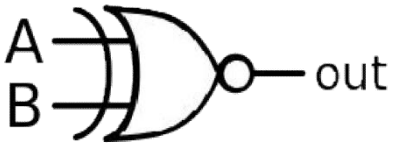


EXCLUSIVE – NOR (X-NOR) GATE:-

- 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.

Logic Symbol

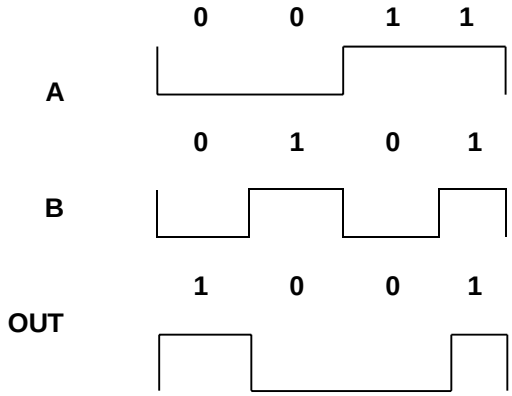


INPUT		OUTPUT
A	**B**	**OUT =A XNOR B**
0	0	1
0	1	0
1	0	0
1	1	1

OUT =A B + A B

= A XNOR B

Timing Diagram

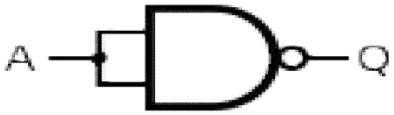


UNIVERSAL GATES:-

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:-

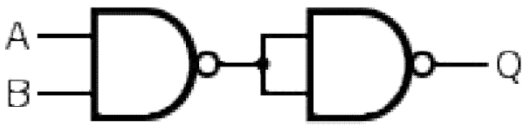
a) Inverter from NAND gate



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

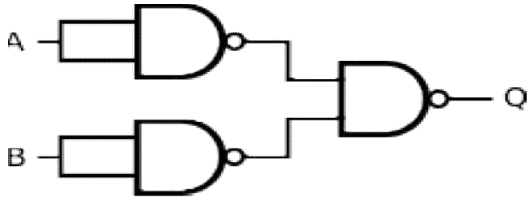
b) AND gate from NAND gate

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



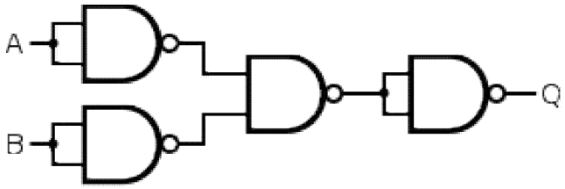
C) OR gate from NAND gate

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



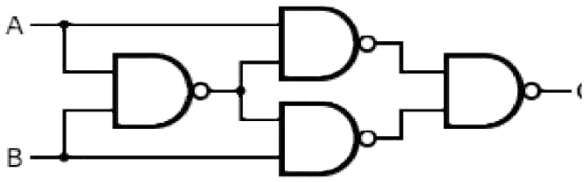
D) NOR gate from NAND gate

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



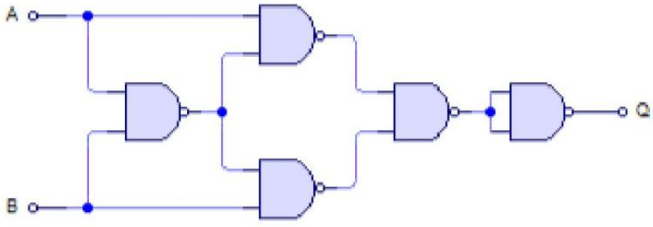
E) EX-OR gate from NAND gate

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



F) EX-NOR gate From NAND gate

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



NOR GATE:-

a) Inverter from NOR gate

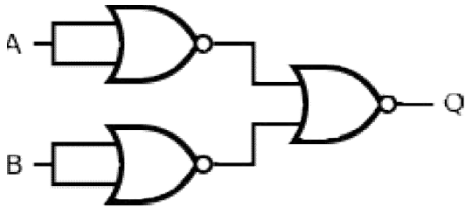
Input = **A**

Output **Q = A**

b) AND gate from NOR gate

Input s are **A** and **B**

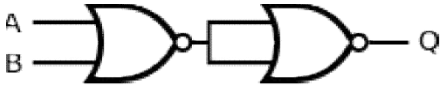
Output **Q = A.B**



C) OR gate from NOR gate

Inputs are **A** and **B**

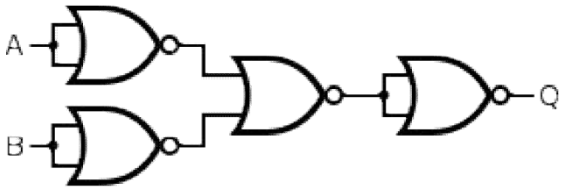
Output **Q = A+B**



D) NAND gate from NOR gate

Inputs are **A** and **B**

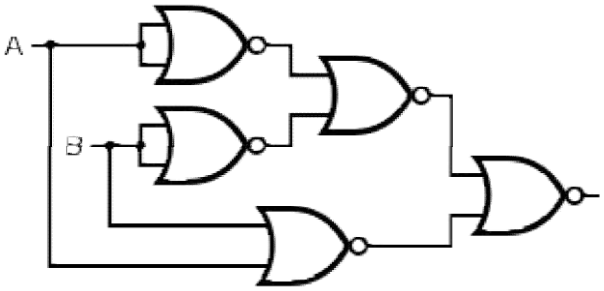
Output **Q = $\overline{A.B}$**



E) X-OR gate from NOR gate

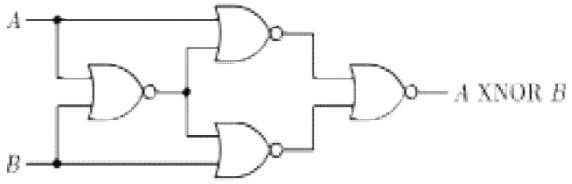
Inputs are **A** and **B**

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



F) EX-NOR gate From NOR gate

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



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: $\bar{1}$ = 0
Axiom 2: 0 . 1 = 0	Axiom 6: 0 + 1 = 1	Axiom 10:0 = 1
Axiom 3: 1 . 0 = 0	Axiom 7: 1 + 0 = 1	
Axiom 2: 1 . 1 = 1	Axiom 8: 1 + 1 = 1	

1. Complementation Laws:-

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

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

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

La3: if A = 0, then $A = \overline{1}$ **Law 4:** if $\underline{A} = 1$,then$A = 0$

Law 5: $\overline{A} = 0$ (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

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

$= 1. (A + B)$

$= A + B$

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 · B = A

Proof

A + A · B = A (1 + B) = A · 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 · A + A · B = A + AB = A(1 + B) = A · 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

2. Consensus Theorem (Included Factor Theorem):- Theorem 1:

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

Proof

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

$= AB + AC + BC(A+A)$

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

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

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

Theorem 2:

$= AB + \bar{A}C$

$=$ **RHS**

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

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

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

$= (AC + BC + AB)(B + C)$

$= ABC + BC + AB + AC + BC + ABC$

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

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

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

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

$= LHS$

3. Transposition Theorem:-

Theorem:

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

Proof

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

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

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

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

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

$= AB + AC$

$= LHS$

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

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

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

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, …..).

The second relation states that the dual can be obtained by complementing all the literals in f (A, B, C, ….).

DUALS:-

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

20. $AB + A + \overline{AB} = 0$ $A + B \cdot A \cdot (\overline{A} + \overline{B}) = 1$

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

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

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

  $= A\bar{B}\bar{C} + \bar{A}\bar{B}C + AB\bar{C} + ABC$
- 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 ∑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 $f(A, B, C) = (A + \bar{B} + C \cdot \bar{C}) + (\bar{A} + \bar{B} + C \cdot C)$

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

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

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 (A + B) (A + B + C) to maxterms and minterms.

Solution:-

In POS form

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

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

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

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

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

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

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

Therefore

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

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

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

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

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

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	$\bar{A}B$	$\bar{A}B$
1	$A\bar{B}$	AB

Example:-

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

Solution:-

The expression minterms is F

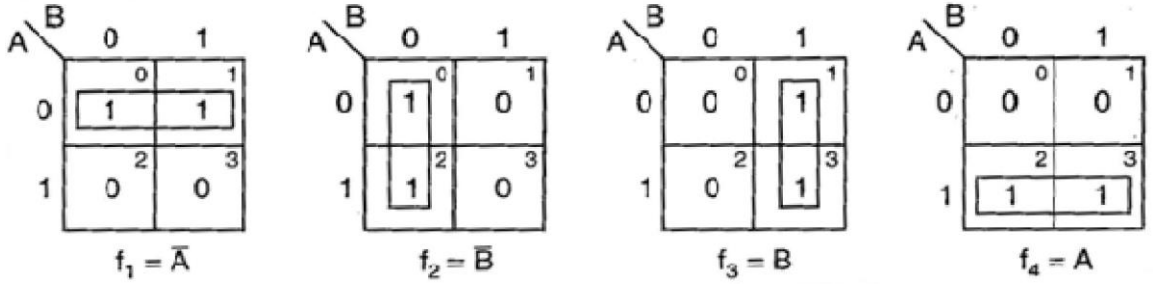
$= m_1 + m_2 = m(1, 2)$

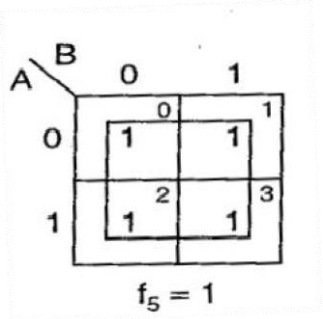
A \ B	0	1
0	0	1
1	2	3

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 somevariables.

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.

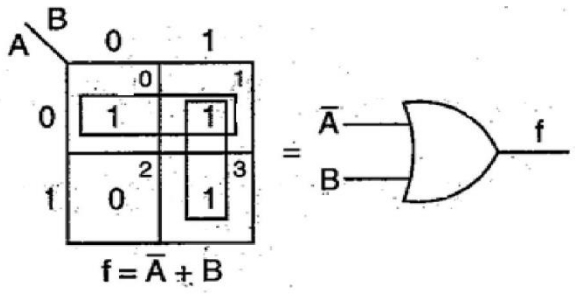
Example:-

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

Solution:-

Expressed in terms of minterms, the given expression is f =

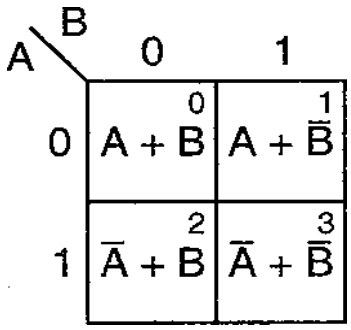
$$m_0 + m_1 + m_3 = \sum m\,(0, 1, 3)$$



F = 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 + B, A + B and $\overline{A} + B$. They are represented as M_0, M_1, M_2 and M_3 respectively.



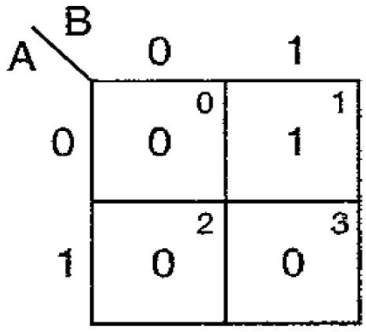
The maxterm of a two variable K-map

Example:-

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

Solution:-

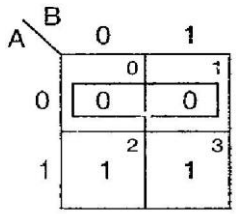
Expression interms of maxterms is f = πM (0, 2, 3)



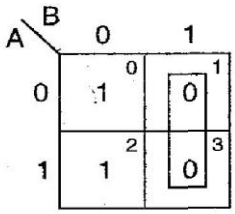
Minimization of POS Expressions:-

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 iswritten.

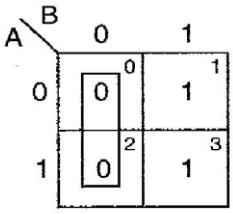
The possible maxterms grouping in a two variable K-map are shown below



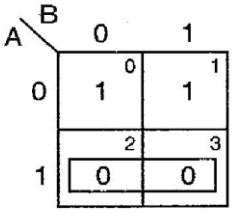
$f_1 = A$



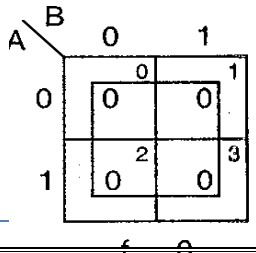
$f_2 = \overline{B}$



$f_3 = B$



$f_4 = \overline{A}$

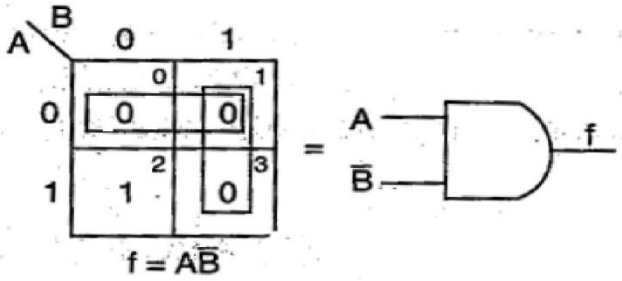


$f_5 = 0$

Example:-

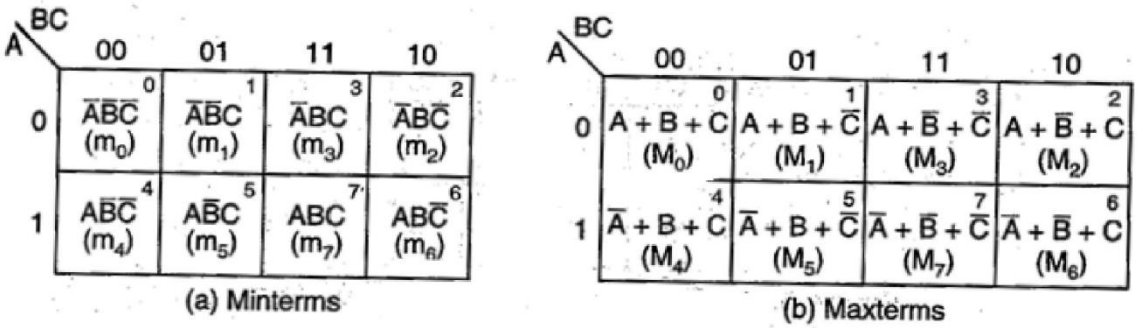
Reduce the expression $\overline{f} = (\overline{A} + B)(A + B)(A + \overline{B})$ using mapping Solution:-

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 minterm or on the map represents a maxterm is shown in the figure below.

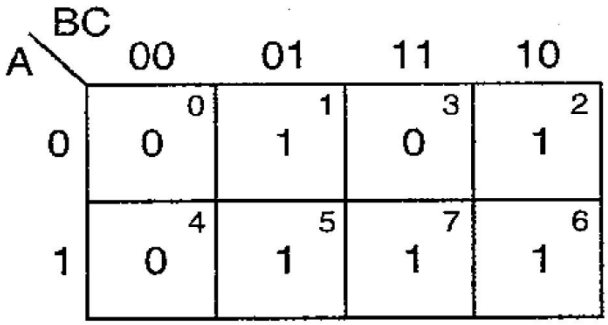


Example:-

Map the expression f = $A\overline{B}\overline{C}+\overline{A}BC + \overline{AB}C + A\overline{B}C + AB\overline{C}$

Solution:-

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

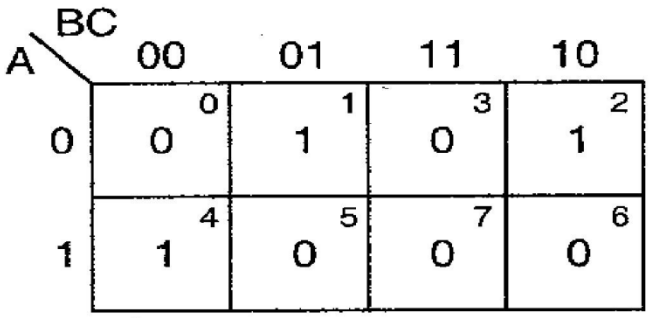


Example:-

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

Solution:-

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

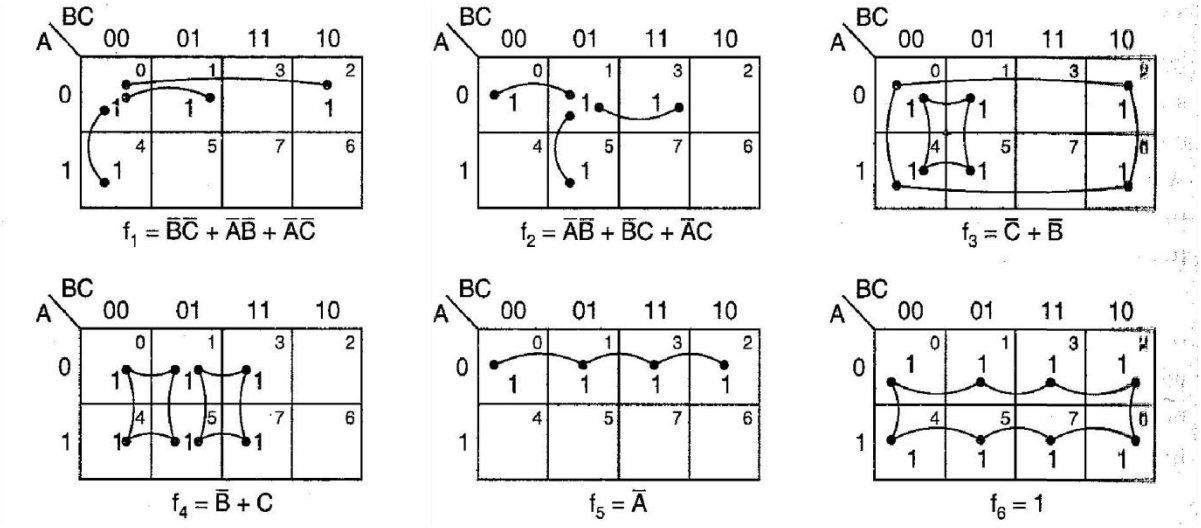


Minimization of SOP and POS Expressions:-

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

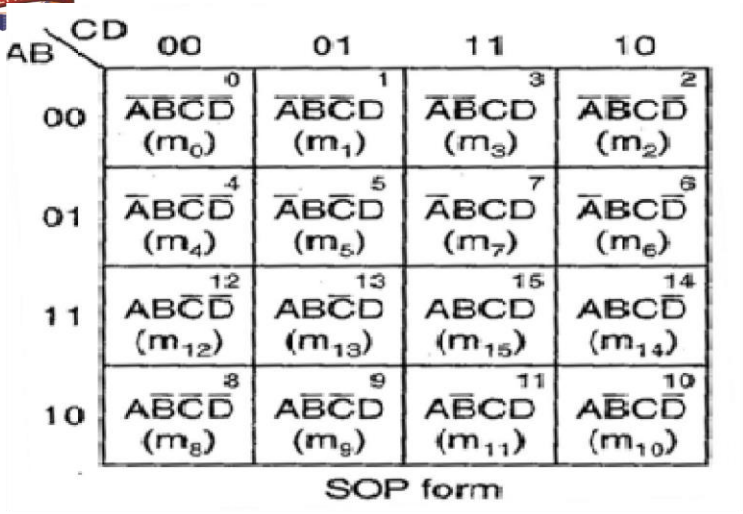


These possible combinations are also for POS but 1s are replaced by 0s.

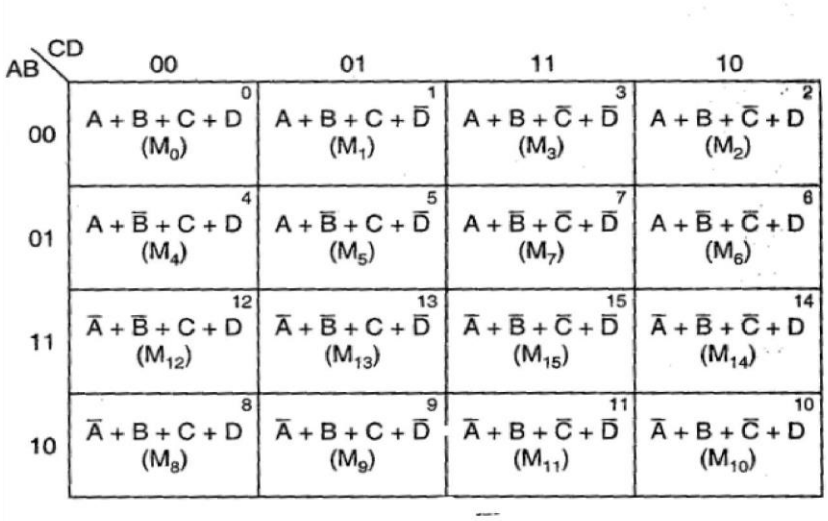
FOUR VARIABLE K-MAP:-

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



POS FORM



Minimization of SOP and POS Expressions:-

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

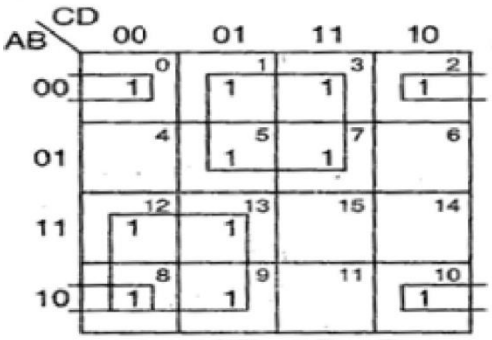
- Draw the K-map and place 1s (0s) corresponding to the minterms (maxterms) of the SOP (POS) expression.
- In the map 1s (0s) which are not adjacent to any other 1(0) are the isolated minterms (maxterms). They are to be read as they are because they cannot be combined even into a 2-square.
- For those 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.

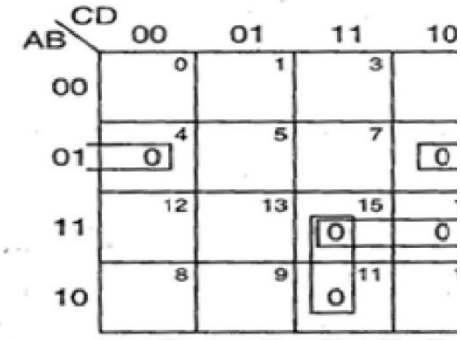
Example:-

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 $f_{min} = B\overline{D} + \overline{A}C + \overline{A}\overline{D}$

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

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.

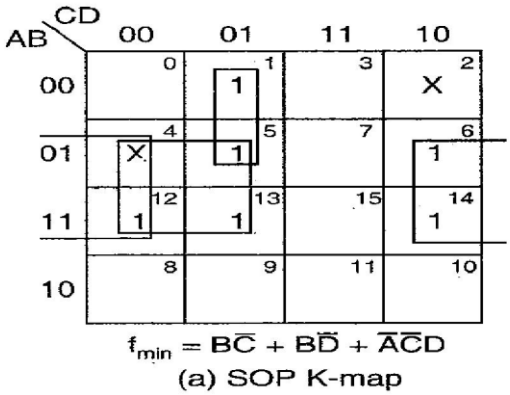
Example:-

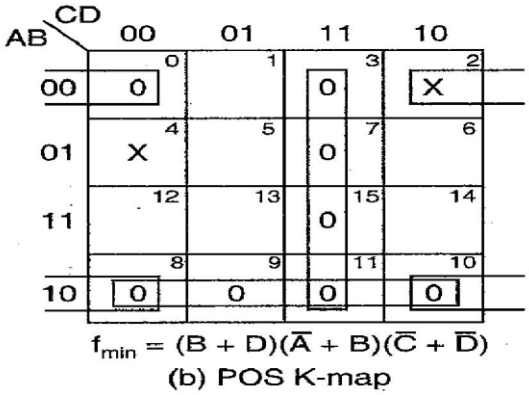
Reduce the expression f = ∑ m(1, 5, 6, 12, 13, 14) + d(2, 4) using K- map.

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 = π M (0, 3, 7, 8, 9, 10, 11,15) + d(2, 4)



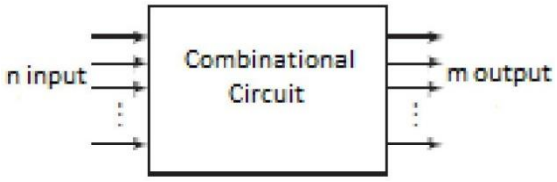


The minimal of SOP expression is $f_{min} = \overline{B}C + \overline{B}\overline{D} + \overline{A}\overline{C}D$

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

(CHAPTER-2)
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

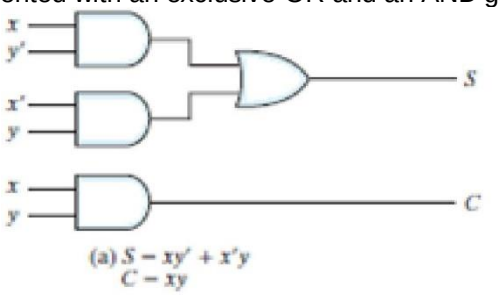
Truth Table

- The simplified sum-of-products expressions are

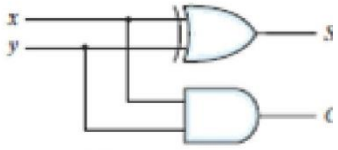
$$S = x'y + xy'$$
$$C = xy$$

- 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) $S = x \oplus y$
$C = xy$

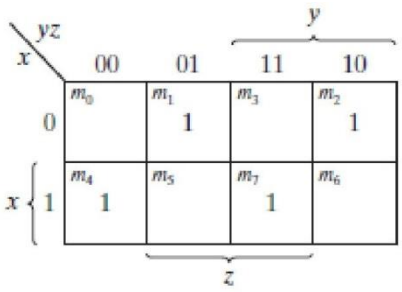
FULL ADDER:-

- 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.
- Two outputs are necessary because the arithmetic sum of three binary digits ranges in value from 0 to 3, and

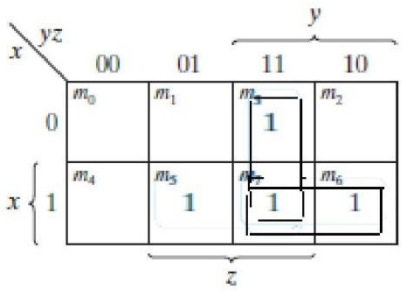
x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Truth Table

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) $C = xy + xz + yz$

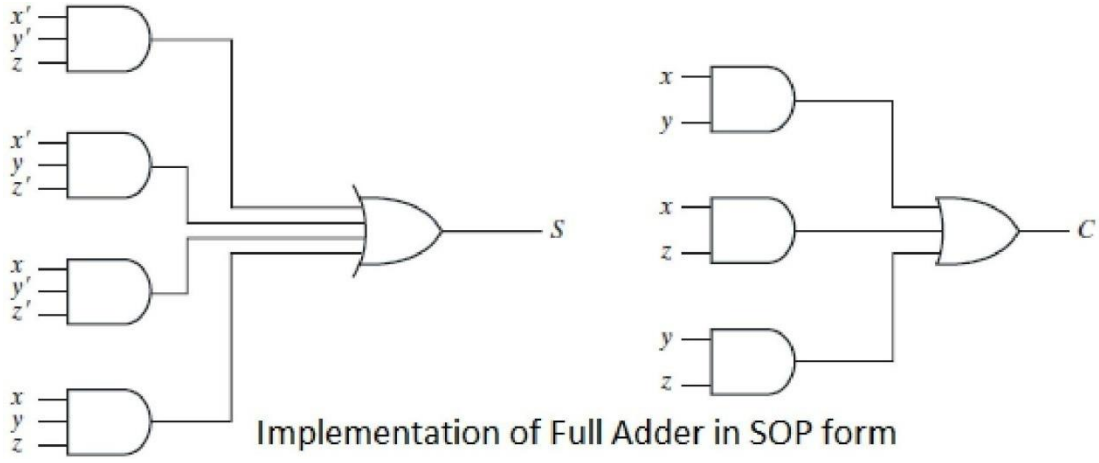
K-Map for full adder

- 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

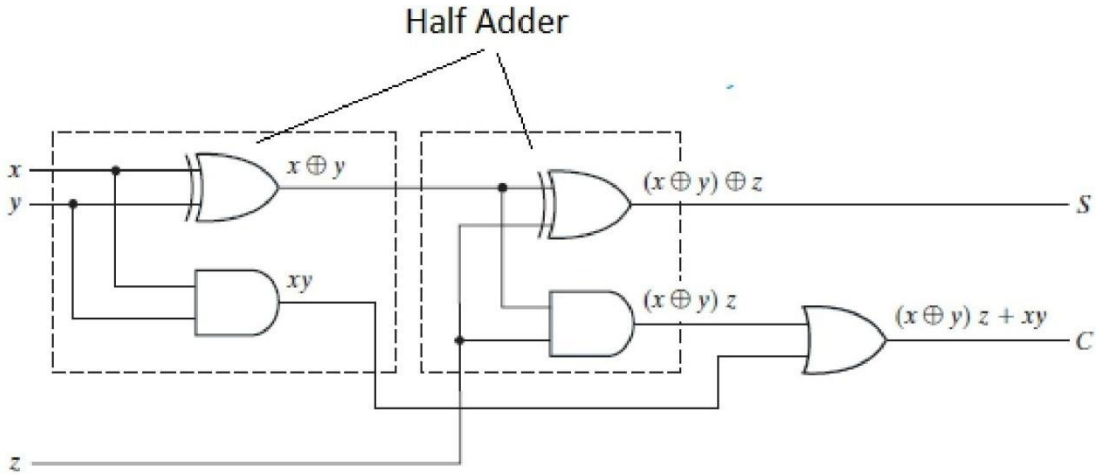
S = x'y'z + x'yz' + xy'z' + xyz

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



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

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

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

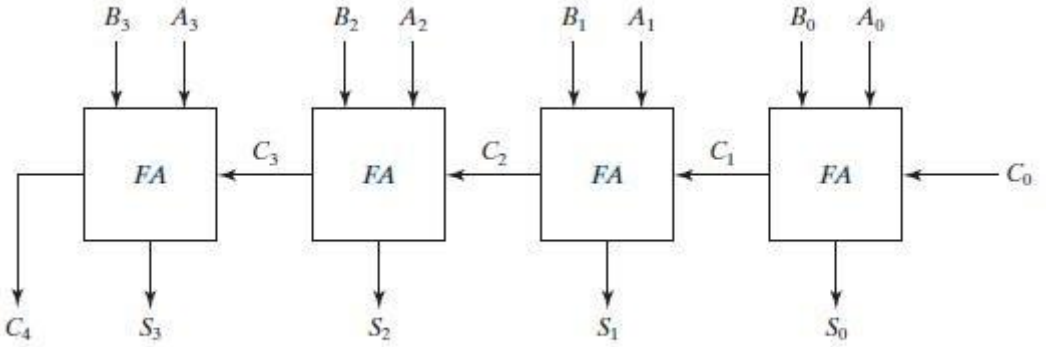
BINARY ADDER:-

- 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

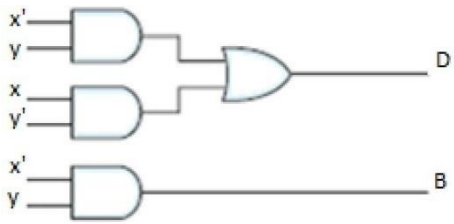
Truth Table

- 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

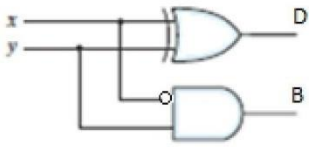
0-0=0;
0-1=1 with borrow 1;
1-0=1;
1-1=0.

- 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:-

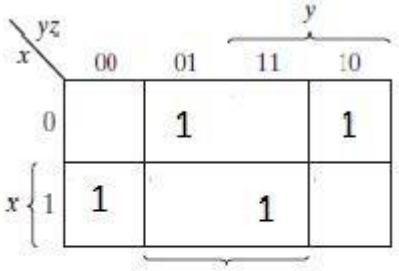
- A full subtractor is a combinational circuit that forms the arithmetic subtraction operation of three bits.
- It consists of three inputs and two outputs. Two of the input variables, denoted by x and y , represent the two significant bits to be subtracted. The third input, z , is subtracted from the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

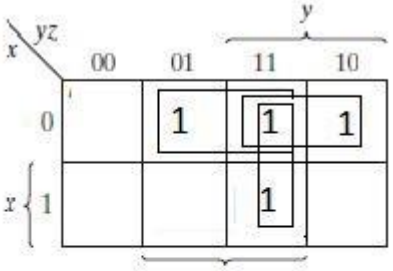
Truth Table

- 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

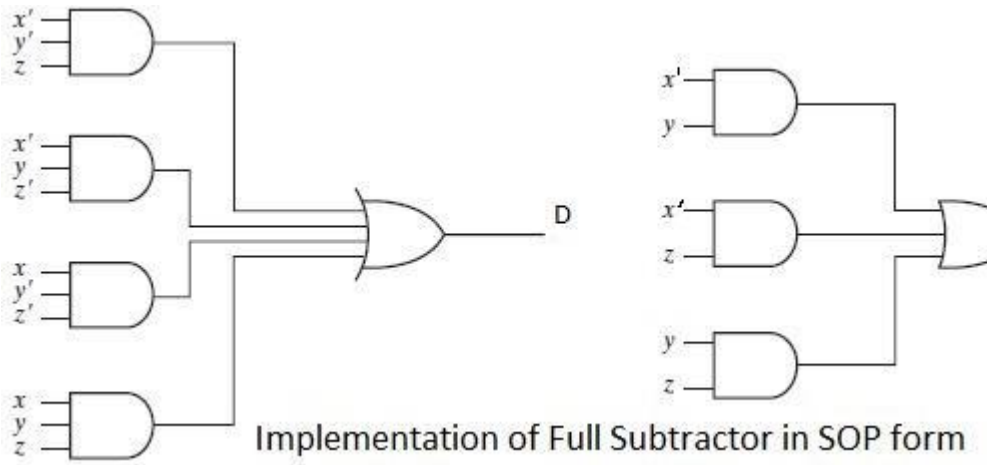
K-Map for full Subtractor

- 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:

$$(A = B) = x_1x_0$$

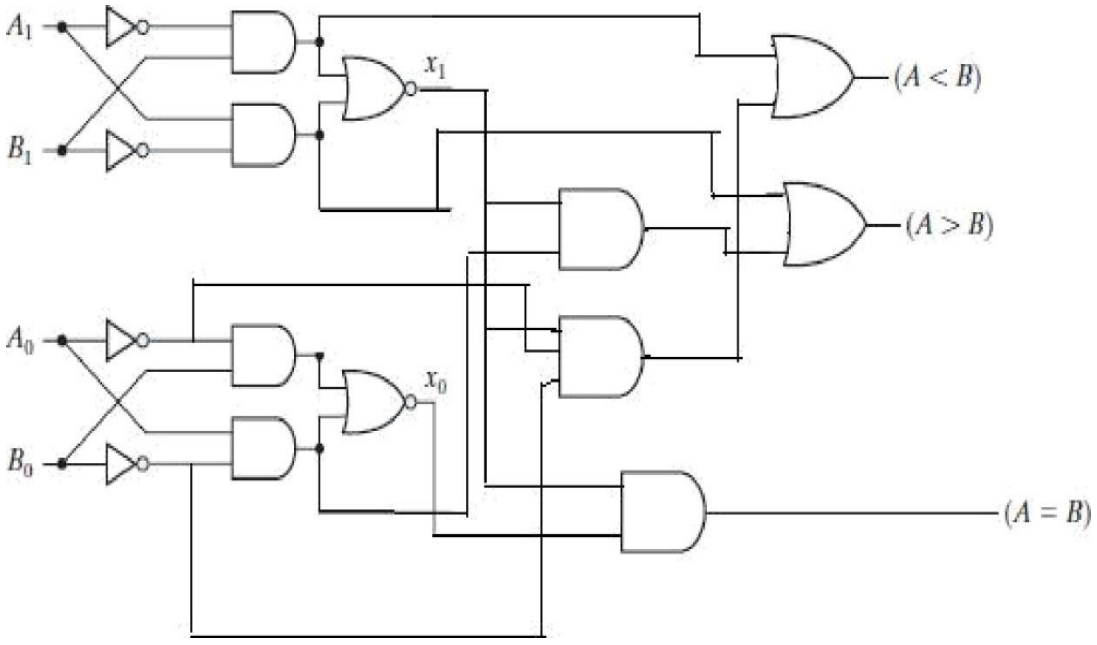
- The binary variable (A = B) is equal to 1 only if all pairs of digits of the two numbers are equal.
- To determine whether A is greater or less than B, we inspect the relative magnitudes of pairs of significant digits, starting from the most significant position. If the two digits of a pair are equal, 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

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

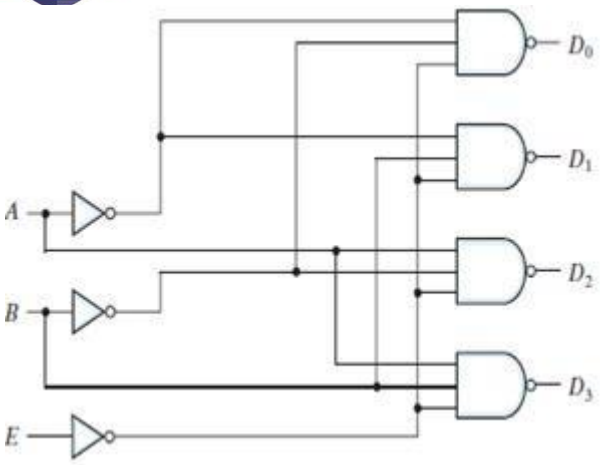
Ai	A	B	Br	A>B	A<B	A=B
0	0	0	0	o	o	o
0	0	0	1	o		o
0	0	1	0	o	¿	
0	0	1	1	0		
0	1	0	0	1	0	0
0	1	0	1	o	o	o
0	1	1	0	o	¿	
0	1	1	1	0		
1	0	0	0		o	o
1	0	0	1		o	i
1	0	1	0	o	i	o
1	0	1	1	o		
1	1	0	0		o	o
1	1	0	1	i	o	o
1	1	1	0	i	o	o
1	1	1	1	o		i

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

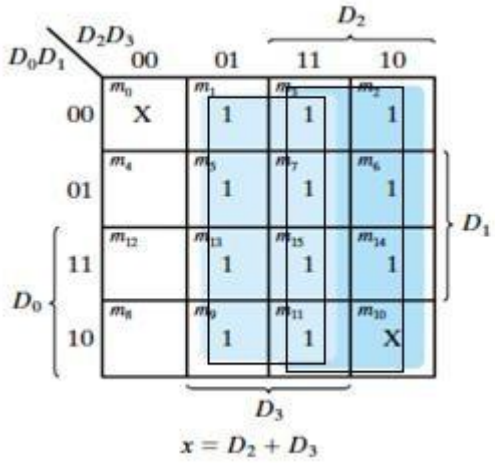
- In addition to the two outputs x and y , the circuit has a third output designated by V ; this is a valid bit indicator that is set to 1 when one or more inputs are equal to 1.

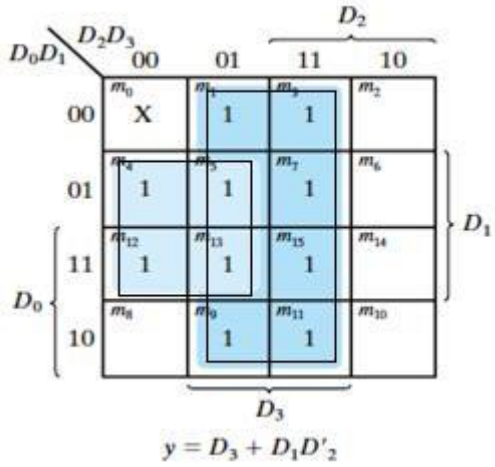
- If all inputs are 0, there is no valid input and V is equal to 0.
- The other two outputs are not inspected when V equals 0 and are specified as don't-care conditions.

- 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.

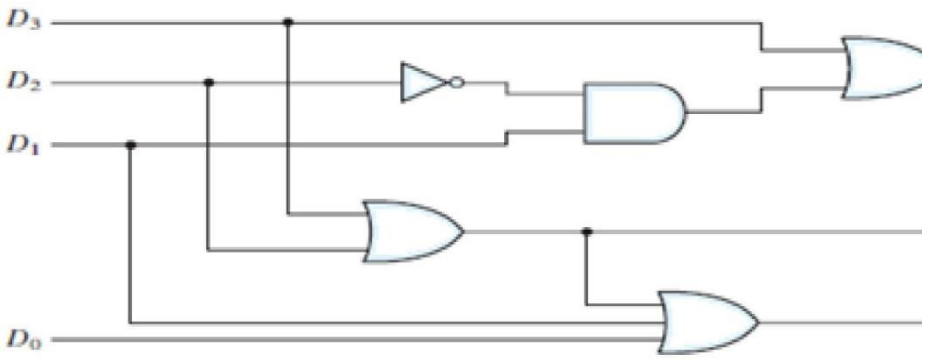




- The maps for simplifying outputs x and y are shown in above Fig.
- The minterms for the two functions are derived from its truth table.
- Although the table has only five rows, when each X in a row is replaced first by 0 and then by 1, we obtain all 16 possible input combinations.
- For example, the 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: $x = D_2 + D_3$

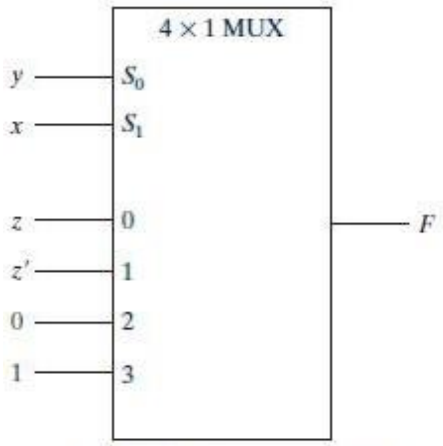
$y = D_3 + D_1 D'_2$

$V = D_0 + D_1 + D_2 + D_3$

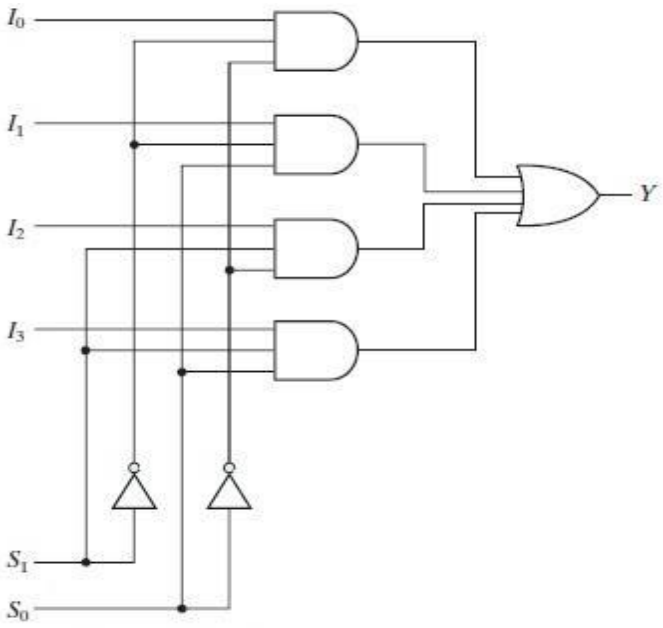


MULTIPLEXER:-

- 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
0	0
0	1
1	0
1	1

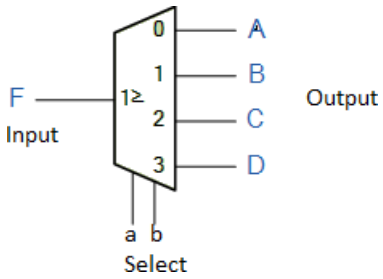
Truth t

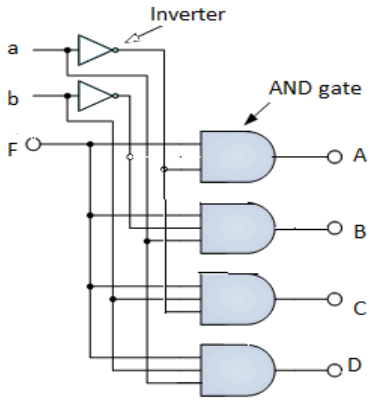
DEMULTIPLEXER:-

- 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.

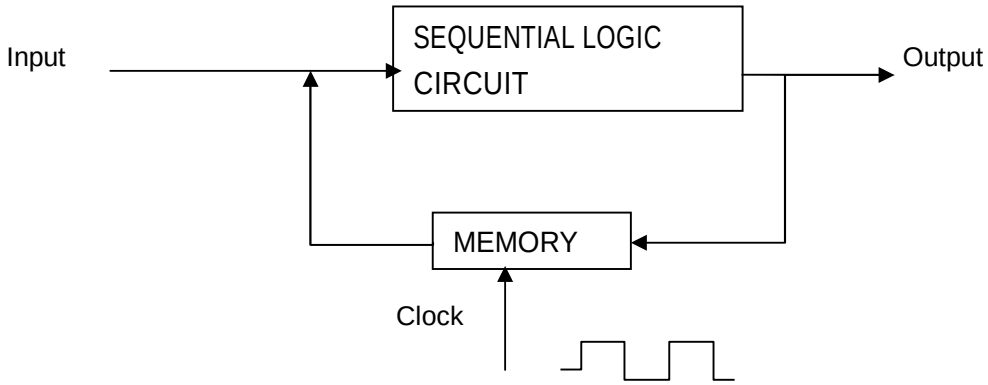
Output Select		Data output Selected
b	a	
0	0	A
0	1	B
1	0	C
1	1	D

Truth Table

(CHAPTER-3)
SEQUENTIAL LOGIC CIRCUITS

SEQUENTIAL CIRCUIT:-

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

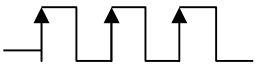
(i) Synchronous SLC
(ii) Asynchronous SLC

- 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.

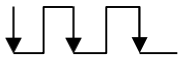
FLIP-FLOP AND LATCH:-

- 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 s

- 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.



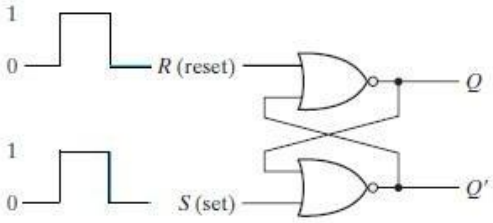
- Negative-edge triggered flip-flops are those in which state transition take place only at negative- going edge of the clock pulse.



- 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.

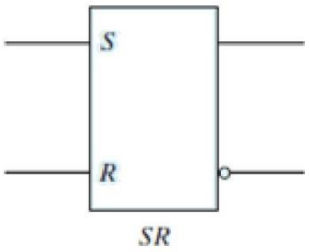


- Under normal conditions, both inputs of the latch remain at 0 unless the state has to be changed. The application of a momentary 1 to the S input causes the latch to go to the set state. The S input must go back to 0 before any other changes take place, in order to avoid the occurrence of an undefined 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'	QNext	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	



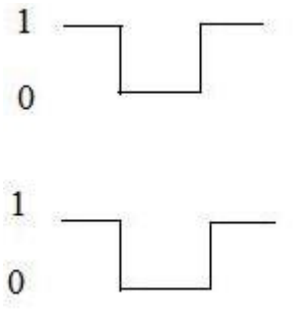
Symbol for SR NOR Latch

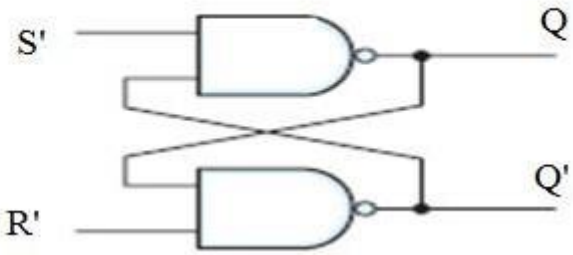
Racing Condition:-

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:-

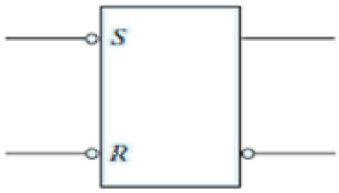
- The below figure represents a SR latch with two cross-coupled NAND gates. The circuit has NAND gates and as we know if any one of the input for a NAND gate is LOW then its output will be HIGH and if both the inputs are HIGH then only the output will be LOW.
- It operates with both inputs normally at 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

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.



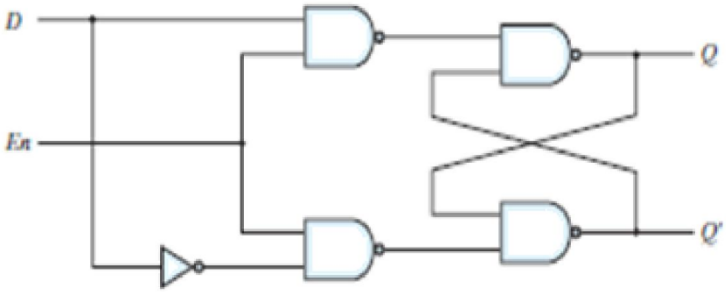
SR

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	Qnext	Q'next
0	0	Race	Race
0	1	0	1 (Reset)
1	0	1	0 (Set)
1	1	Q (No change)	Q' (No change)

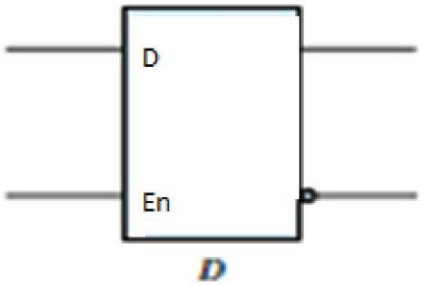
D LATCH:-

- 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)

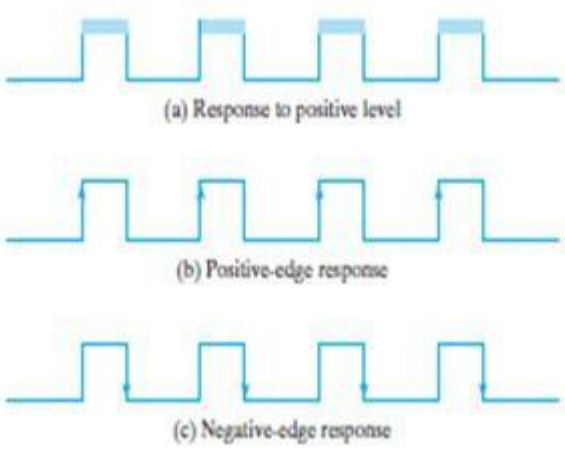
- As long as the enable input is at 0, the cross-coupled SR latch has both inputs at the 1 level and the circuit can't change state regardless of the value of D.
- The below represents the truth table for the D-latch.

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:-

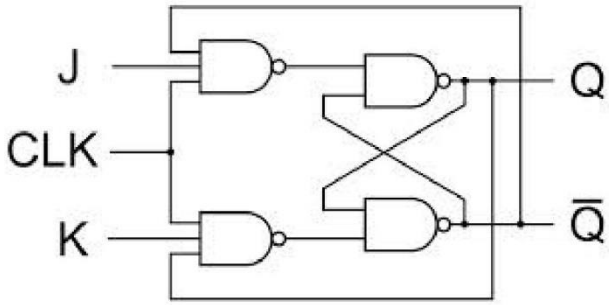
- The state of a latch or flip-flop is switched by a change in the control input. This momentary change is called a trigger, and the transition it causes is said to trigger the flip-flop.
- Flip-flop circuits are constructed in such a way as to make them operate properly when they are part of a sequential 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:-

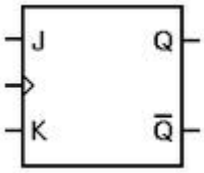
- The JK flip-flop can be constructed by using basic SR latch and a clock. In this case the outputs Q and Q' are returned back and connected to the inputs of NAND gates.
- This simple JK flip Flop is the most widely used of all the flip-flop designs and is considered to be a universal flip-flop circuit.
- 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.

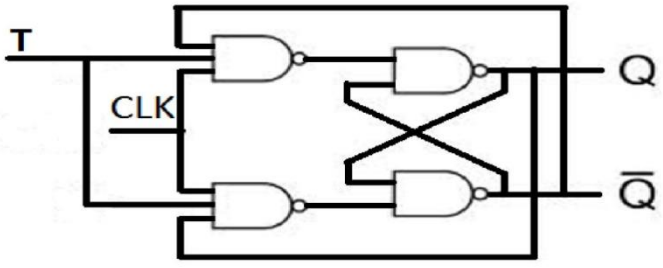
(Truth table for JK flip-flop)

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	Qnext	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

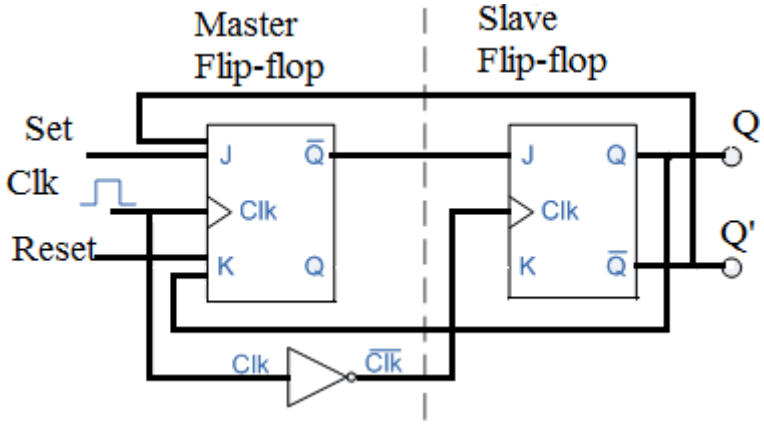
- The characteristic table of T flip-flop has only two conditions: When T = 0, the clock edge does not change the state; when T = 1, the clock edge complements the state of the flip-flop and the characteristic equation is

MASTER-SLAVE JK FLIP-FLOP:-

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

- 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.

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.

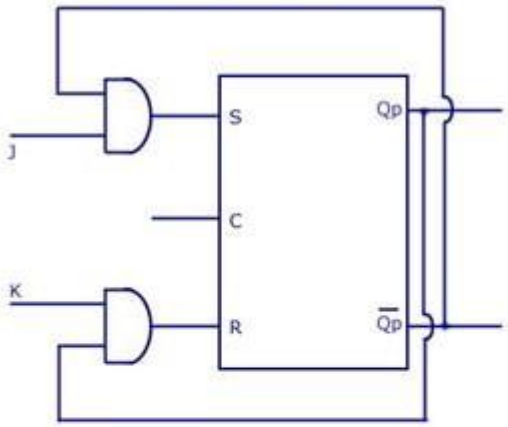
For two inputs J and K, there will be eight possible combinations. For each combination of J, K and Qp, the corresponding Qp+1 states are found. Qp+1 simply suggests the future values to be obtained by the JK flip flop after the value of Qp. The table is then completed by writing the values of S and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

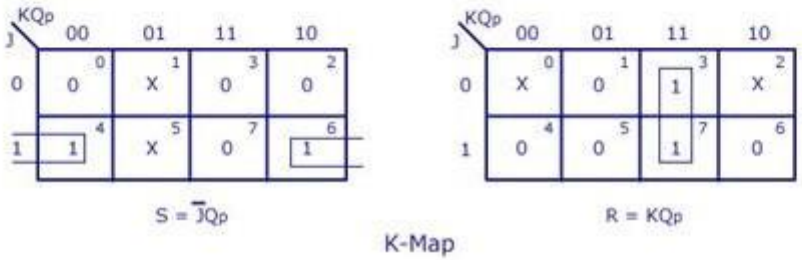
S-R Flip Flop to J-K Flip Flop

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





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.

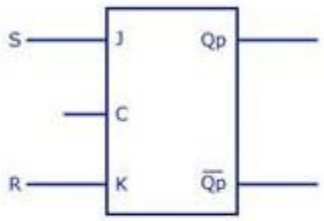
- For two inputs, S and R, eight combinations are made. For each combination, the corresponding Qp+1 outputs are found out.
- The outputs for the combinations of S=1 and R=1 are not permitted for an SR flip flop. Thus the outputs are considered invalid and the J and K values are taken as “don’t cares”.

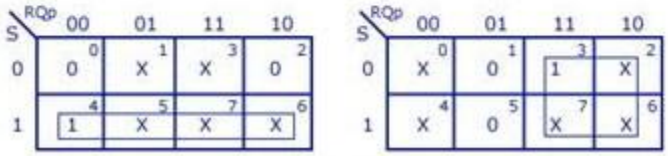
J-K Flip Flop to S-R Flip Flop

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

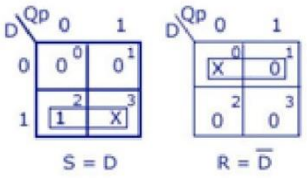
- S and R are the actual inputs of the flip flop and D is the external input of the flip flop.
- The four combinations, the logic diagram, conversion table, and the K-map for S and R in terms of D and Qp are shown below.

S-R Flip Flop to D Flip Flop

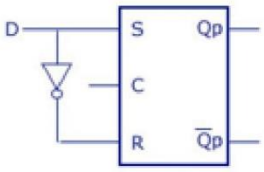
Conversion Table

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

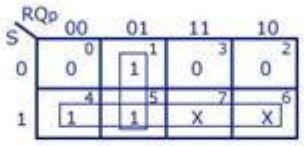
- D is the actual input of the flip flop and S and R are the external inputs. Eight possible combinations are achieved from the external inputs S, R and Qp.
- But, since the combination of S=1 and R=1 are invalid, the values of Qp+1 and D are considered as “don’t cares”.
- The logic diagram showing the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

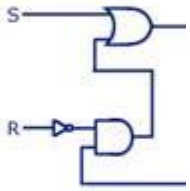
S-R Inputs S	R	Outputs Qp	Qp+1	D Input
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1
1	1	Invalid		Don't care
1	1	Invalid		Don't care

K-map



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

Logi



JK Flip Flop to T Flip Flop:-

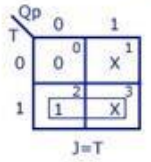
- J and K are the actual inputs of the flip flop and T is taken as the external input for conversion
- Four combinations are produced with T and Qp. J and K are expressed in terms of T and Qp.
    - The conversion table, K-maps, and the logic diagram are given below.

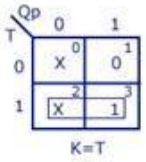
J-K Flip Flop to T Flip Flop

Conversion Table

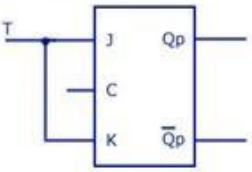
T Input	Outputs Qp	Qp+1	J-K Inputs J	K
0	0	0	0	X
0	1	1	X	0
1	0	1	1	X
1	1	0	X	1

K-maps





Logic Diagram



D Flip Flop to JK Flip Flop:-

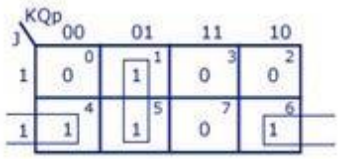
- In this conversion, D is the actual input to the flip flop and J and K are the external inputs.
- J, K and Qp make eight possible combinations, as shown in the conversion table below. D is expressed in terms of J, K and Qp.
- The conversion table, the K-map for D in terms of J, K and Qp and the logic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

D Flip Flop to J-K Flip Flop

Conversion Table

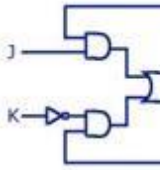
J-K Input J	K	Outputs Qp	Qp+1	D Input
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	0	0

K-map



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

L



JK Flip Flop to D Flip Flop:-

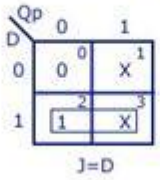
- D is the external input and J and K are the actual inputs of the flip flop. D and Qp make four combinations. J and K are expressed in terms of D and Qp.
- The four combination conversion table, the K-maps for J and K in terms of D and Qp.

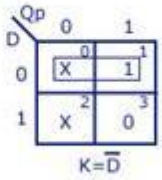
J-K Flip Flop to D Flip Flop

Conversion Table

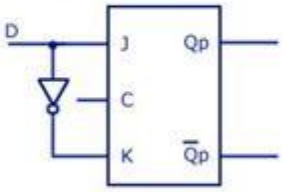
D Input	Outputs Qp	Qp+1	J-K Inputs J	K
0	0	0	0	X
0	1	0	X	1
1	0	1	1	X
1	1	0	X	0

K-maps





Logic Diagram

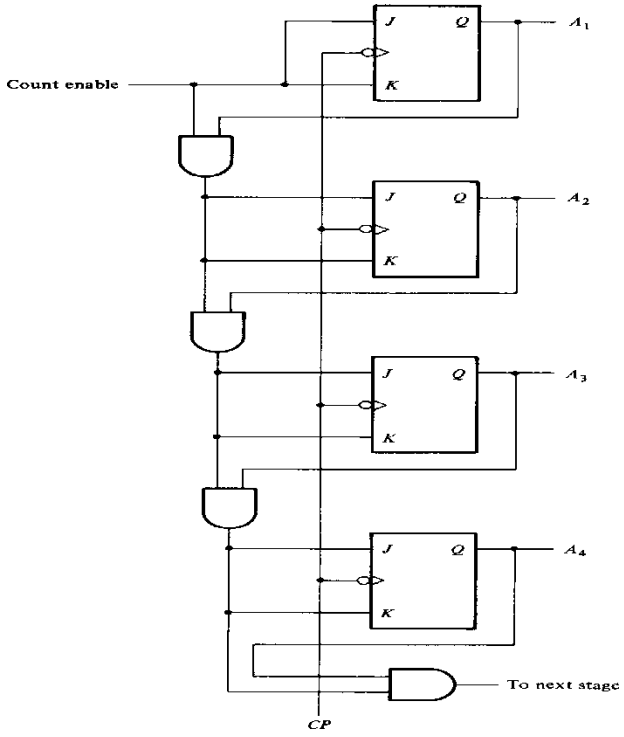


COUNTER

- A counter is a device which stores (and sometimes displays) the number of times a particular event or process has occurred. In electronics, counters can be implemented quite easily using register-type circuits.
- There are different types of counters, viz.
    - Asynchronous (ripple) counter
    - Synchronous counter
    - Decade counter
    - Up/down counter
    - Ring counter
    - Johnson counter
    - Cascaded counter
    - Modulus counter.

Synchronous counter

- A 4-bit synchronous counter using JK flip-flops is shown in the figure.
- In synchronous counters, the clock inputs of all the flip-flops are connected together and are triggered by the input pulses. Thus, all the flip-flops change state simultaneously (in parallel).

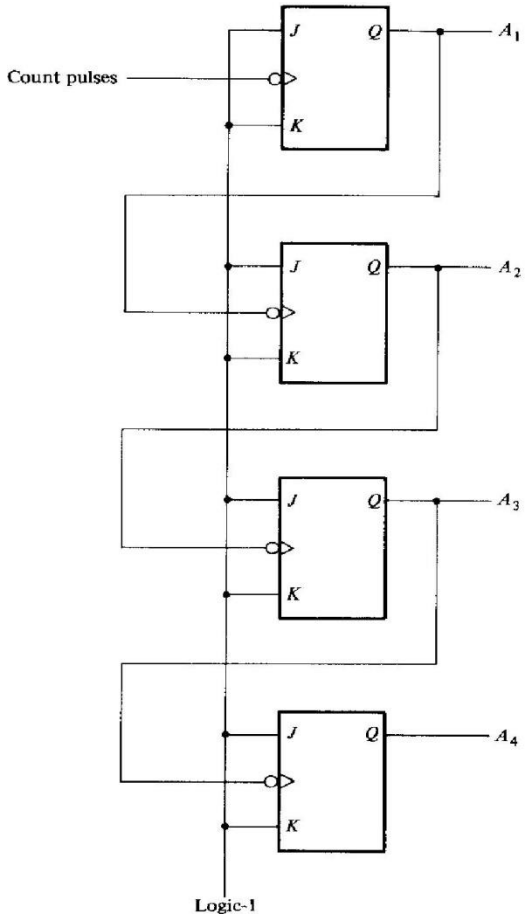


- The circuit below is a 4-bit synchronous counter.
- The J and K inputs of FF0 are connected to HIGH. FF1 has its J and K inputs connected to the output of FF0, and the J and K inputs of FF2 are connected to the output of an AND gate that is fed by the outputs of FF0 and FF1.
- A simple way of implementing the logic for each bit of an ascending counter (which is what is depicted in the image to the right) is for each bit to toggle when all of the less significant bits are at a logic high state.
- For example, bit 1 toggles when bit 0 is logic high; bit 2 toggles when both bit 1 and bit 0 are logic high; bit 3 toggles when bit 2, bit 1 and bit 0 are all high; and so on.

- Synchronous counters can also be implemented with hardware finite state machines, which are more complex but allow for smoother, more stable transitions.

Asynchronous Counter

- An asynchronous (ripple) counter is a single d-type flip-flop, with its J (data) input fed from its own inverted output.
- This circuit can store one bit, and hence can count from zero to one before it overflows (starts over from 0).



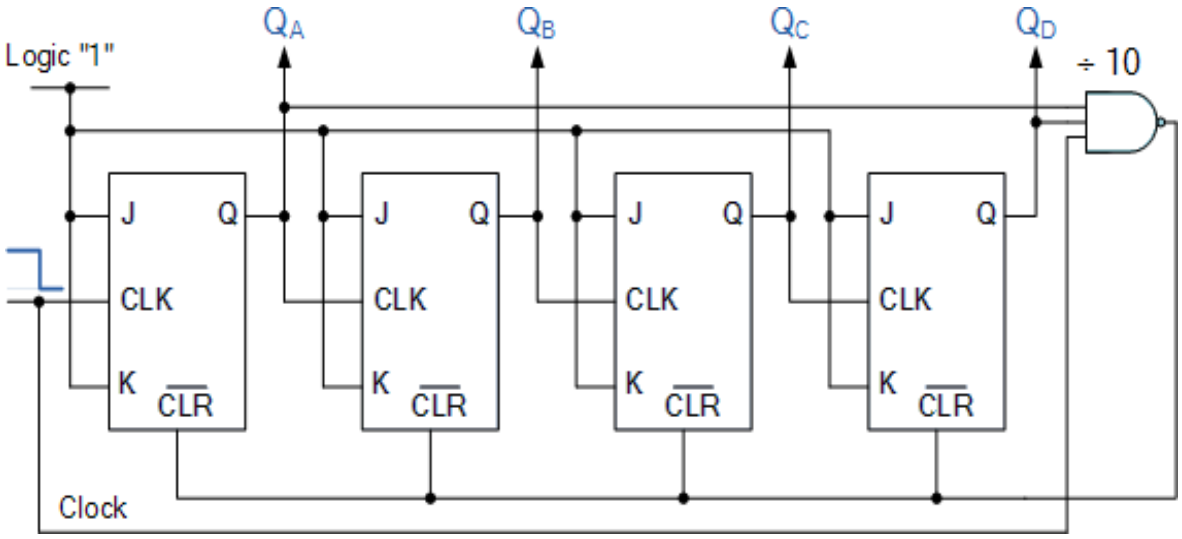
- This counter will increment once for every clock cycle and takes two clock cycles to overflow, so every cycle it will alternate between a transition from 0 to 1 and a transition from 1 to 0.
- This creates a new clock with a 50% duty cycle at exactly half the frequency of the input clock.
- If this output is then used as the clock signal for a similarly arranged D flip-flop, remembering to invert the output to the input, one will get another 1 bit counter that counts half as fast. These together yield a two-bit counter.
- Additional flip-flops can be added, by always inverting the output to its own input, and using the output from the previous flip-flop as the clock signal. The result is called a ripple counter, which can count to 2^n – 1, where n is the number of bits (flip-flop stages) in the counter.
- Ripple counters suffer from unstable outputs as the overflows "ripple" from stage to stage, but they find application as dividers for clock signals.

Modulus Counter

- A modulus counter is that which produces an output pulse after a certain number of input pulses is applied.
- In modulus counter the total count possible is based on the number of stages, i.e., digit positions.

- Modulus counters are used in digital computers.
- A binary modulo-8 counter with three flip-flops, i.e., three stages, will produce an output pulse, i.e., display an output one-digit, after eight input pulses have been counted, i.e., entered or applied. This assumes that the counter started in the zero-condition.

Asynchronous Decade Counter



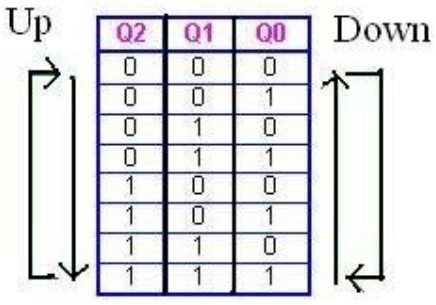
- A decade counter can count from BCD “0” to BCD “9”.
- A decade counter requires resetting to zero when the output count reaches the decimal value of 10, ie. when DCBA = 1010 and this condition is fed back to the reset input.
- A counter with a count sequence from binary “0000” (BCD = “0”) through to “1001” (BCD = “9”) is generally referred to as a BCD binary-coded-decimal counter becauseits ten state sequence is that of a BCD code but binary decade counters are more common.
- This type of asynchronous counter counts upwards on each leading edge of the input clock signal starting from 0000 until it reaches an output 1001 (decimal 9).
- Both outputs Q_A and Q_D are now equal to logic “1” and the output from the NAND gate changes state from logic “1” to a logic “0” level and whose output is also connected to the CLEAR (CLR) inputs of all the J-K Flip-flops.
- This signal causes all of the Q outputs to be reset back to binary 0000 on the count of 10. Once QA and QD are both equal to logic “0” the output of the NAND gate returns back to a logic level “1” and the counter restarts again from 0000. We now have a decade or Modulo-10 counter.

Decade Counter Truth Table

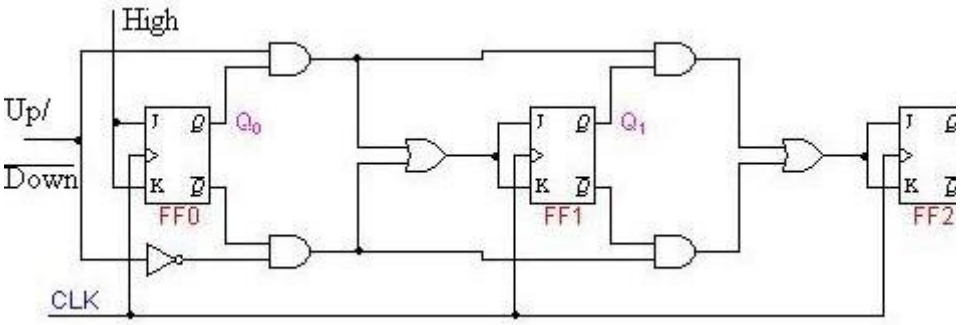
Clock Count	Output bit Pattern				Decimal Value
	QD	QC	QB	QA	
1	0	0	0	0	0
2	0	0	0	1	1
3	0	0	1	0	2
4	0	0	1	1	3
5	0	1	0	0	4
6	0	1	0	1	5
7	0	1	1	0	6
8	0	1	1	1	7
9	1	0	0	0	8
10	1	0	0	1	9
11	Counter Resets its Outputs back to Zero				

Up/Down Counter

- In a synchronous up-down binary counter the flip-flop in the lowest-order position is complemented with every pulse.
- A flip-flop in any other position is complemented with a pulse, provided all the lower-order pulse equal to 0.
- Up/Down counter is used to control the direction of the counter through a certain sequence.



- From the sequence table we can observe that:
  - For both the UP and DOWN sequences, Q_0 toggles on each clock pulse.
  - For the UP sequence, Q_1 changes state on the next clock pulse when Q_0=1.
  - For the DOWN sequence, Q_1 changes state on the next clock pulse when Q_0=0.
  - For the UP sequence, Q_2 changes state on the next clock pulse when Q_0=Q_1=1.
  - For the DOWN sequence, Q_2 changes state on the next clock pulse when Q_0=Q_1=0.



- These characteristics are implemented with the AND, OR & NOT logic connected as shown in the logic diagram above**.**

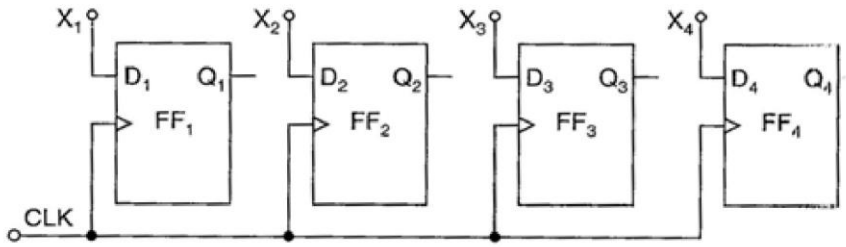
REGISTERS

INTRODUCTION:-

- The sequential circuits known as register are very important logical block in most of the digital systems.
- Registers are used for storage and transfer of binary information in a digital system.
- A register is mostly used for the purpose of storing and shifting binary data entered into it from an external source and has no characteristics internal sequence of states.
- The storage capacity of a register is defined as the number of bits of digital data, it can store or retain.
- These registers are normally used for temporary storage of data.

BUFFER REGISTER:-

- These are the simplest registers and are used for simply storing a binary word.
- These may be controlled by Controlled Buffer Register.
- D flip – flops are used for constructing a buffer register or other flip- flop can be used.
- The figure shown below is a 4- bit buffer register.



Logic diagram of a 4-bit buffer register.

- When the clock pulse is applied, the output word becomes the same as the word applied at the input terminals, i.e. the input word is loaded into the register by the application of clock pulse.
- When the positive clock edge arrives, the stored word becomes:

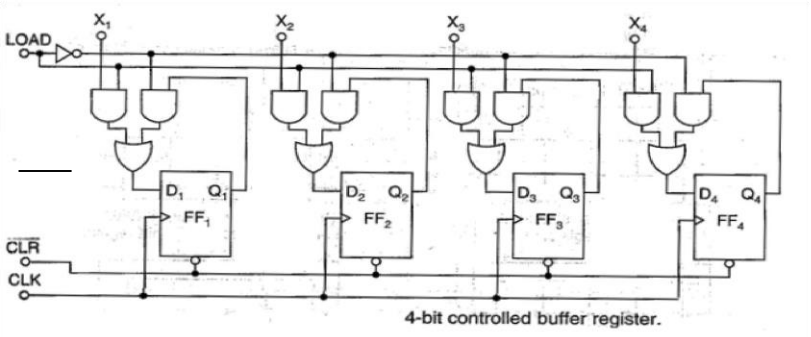
  Q4 Q3 Q2 Q1= X4 X3 X2 X1

  or Q = X .

  This circuit is too primitive to be of any use.

CONTROLLED BUFFER REGISTER:-

- The figure shows a controlled buffer register.



4-bit controlled buffer register.

- If CLR goes LOW, all the flip-flops are RESET and the output becomes, Q = 0000.
- When CLR is HIGH, the register is ready for action

- LOAD is control input.
- When LOAD is HIGH, the data bits X can reach the D inputs of FFs.

- At the positive going edge of the next clock pulse, the register is loaded, i.e.

  Q4 Q3 Q2 Q1= X4 X3 X2 X1

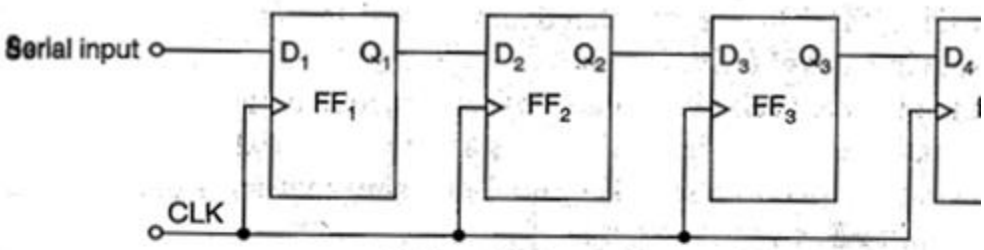
  or Q = X .
- When LOAD is LOW, the X bits cannot reach the FFs. At the same time the inverted signal LOAD is HIGH. This forces each flip-flop output to feedback to its data input.
- Therefore data is circulated or retained as each clock pulse arrives.
- In other words the content register remains unchanged in spite of the clock pulses.
- Longer buffer registers can built by adding more FFs.

CONTROLLED BUFFER REGISTER:-

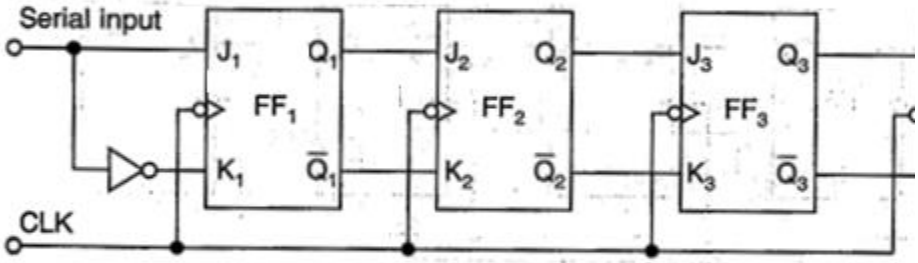
- A number of FFs connected together such that data may be shifted into and shifted out of them is called a shift register.
- Data may be shifted into or out of the register either in serial form or in parallel form.
- There are four basic types of shift registers
  1. Serial in, serial out
  2. Serial in, parallel out
  3. Parallel in, serial out
  4. Parallel in , parallel out

SERIAL IN, SERIAL OUT SHIFT REGISTER:-

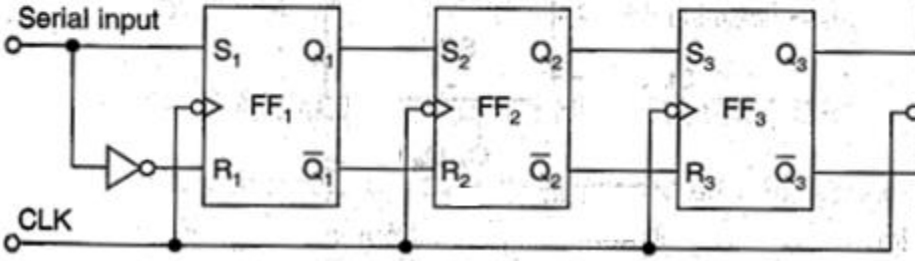
- This type of shift register accepts data serially, i.e., one bit at a time and also outputs data serially.
- The logic diagram of a four bit serial in, serial out shift register is shown in below figure:
- In 4 stages i.e. with 4 FFs, the register can store upto 4 bits of data.
- Serial data is applied at the D input of the first FF. The Q output of the first FF is connected to the D input of the second FF, the output of the second FF is connected to the D input of the third FF and the Q output of the third FF is connected to the D input of the fourth FF. The data is outputted from the Q terminal of the last FF.
- When a serial data is transferred to a register, each new bit is clocked into the first FF at the positive going edge of each clock pulse.
- The bit that is previously stored by the first FF is transferred to the second FF.
- The bit that is stored by the second FF is transferred to the third FF, and so on.
- The bit that was stored by the last FF is shifted out.
- A shift register can also be constructed using J-K FFs or S-R FFs as shown in the figure below.



4-bit serial-in, serial-out, shift-right, shift reg



(a) Using J-K FFs

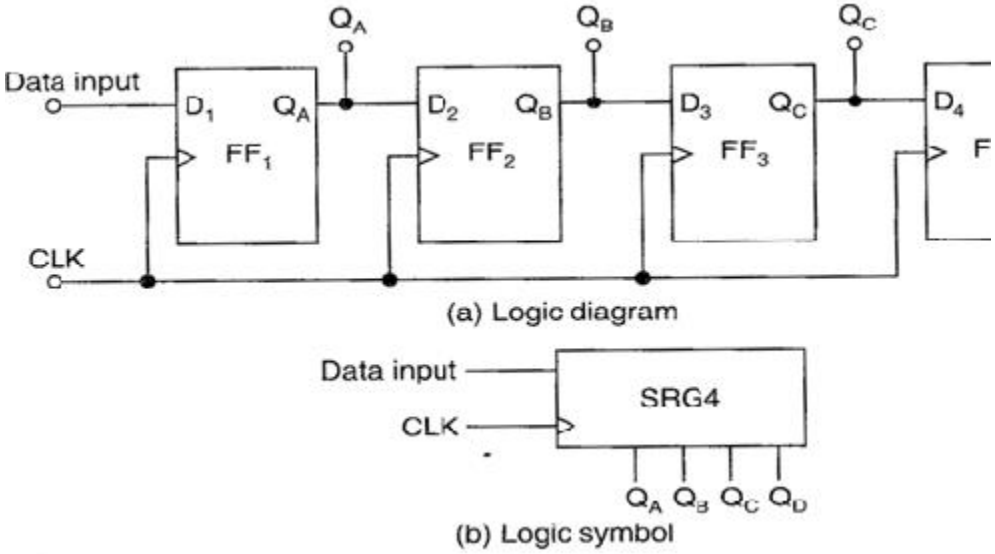


(b) Using S-R FFs

A 4-bit serial-in, serial-out, shift register.

SERIAL IN, PARALLEL OUT SHIFT REGISTER:-

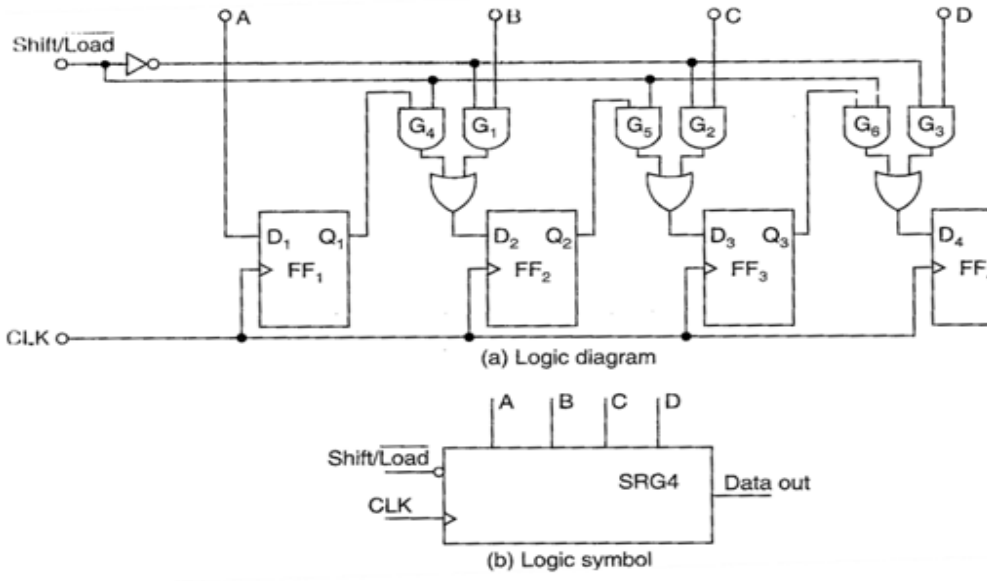
- In this type of register, the data bits are entered into the register serially, but the data stored in the register serially, but the stored in the register is shifted out in the parallel form.
- When the data bits are stored once, each bits appears on its respective output line and all bits are available simultaneously, rather than bit – by – bit basis as in the serial output.
- The serial in, parallel out shift register can be used as a serial in, serial out shift register if the output is taken from the Q terminal of the last FF.
- The logic diagram and logic symbol of a 4 bit serial in, parallel out shift register is given below.



A 4- bit serial in, parallel out shift register

PARALLEL IN, SERIAL OUT SHIFT REGISTER:-

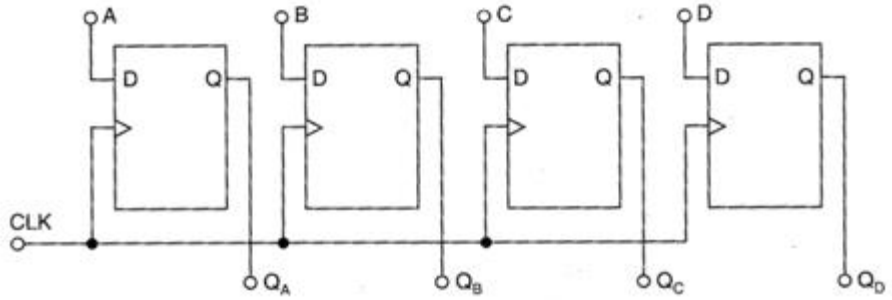
- For parallel in, serial out shift register the data bits are entered simultaneously into their respective stages on parallel lines, rather than on bit by bit basis on one line as with serial data inputs, but the data bits are transferred out of the register serially, i.e., on a bit by bit basis over a single line.
- The logic diagram and logic symbol of 4 bit parallel in, serial out shift register using D FFs is shown below.
- There are four data lines A, B, C and D through which the data is entered into the register in parallel form.
- The signal Shift /LOAD allows
    1. The data to be entered in parallel form into the register and
    2. The data to be shifted out serially from terminal Q_4.
- When Shift /LOAD line is HIGH, gates G1, G2, and G3 are disabled, but gates G4, G5 and G6 are enabled allowing the data bits to shift right from one stage to next.
- When Shift /LOAD line is LOW, gates G4, G5 and G6 are disabled, whereas gates G1, G2 and G3 are enabled allowing the data input to appear at the D inputs of the respective FFs.
- When clock pulse is applied, these data bits are shifted to the Q output terminals of the FFs and therefore the data is inputted in one step.
- The OR gate allows either the normal shifting operation or the parallel data entry depending on which AND gates are enabled by the level on the Shift /LOAD input.



A 4- bit parallel in, serial out shift register

PARALLEL IN, PARALLEL OUT SHIFT REGISTER:-

- In a parallel in, parallel out shift register, the data entered into the register in parallel form and also the data taken out of the register in parallel form. Immediately following the simultaneous entry of all data bits appear on the parallel outputs.
- The figure shown below is a 4 bit parallel in parallel out shift register using D FFs.
- Data applied to the D input terminals of the FFs.
- When a clock pulse is applied at the positive edge of that pulse, the D inputs are shifted into the Q outputs of the FFs.
- The register now stores the data.
- The stored data is available instantaneously for shifting out in parallel form.



Logic diagram of a 4 – bit parallel in, parallel out shift register

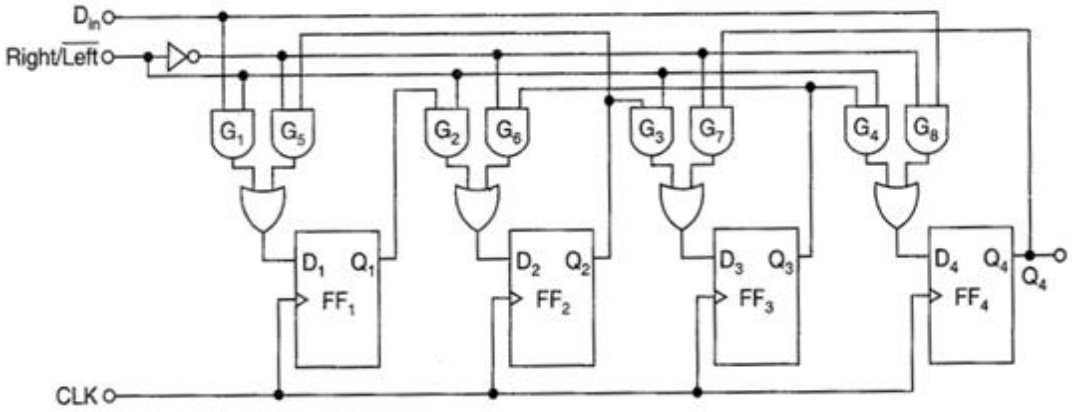
BIDIRECTIONAL SHIFT REGISTER:-

- In bidirectional shift register is one in which the data bits can be shifted from left to right or from right to left.
- The figure shown below the logic diagram of a 4 bit serial in, serial out, bidirectional (shift-left, shift- right) shift register.
- Right /Left is the mode signal. When Right /Left is a 1, the logic circuit works as a shift right shift register. When
- Right /Left is a 0, the logic circuit works as a shift right shift register.

- The bidirectional is achieved by using the mode signal and two AND gates and one OR gate for each stage.

- A HIGH on the Right/Left control input enables the AND gates G_1, G_2, G_3 and G_4 and disables the AND gates G_5, G_6, G_7 and G_8 and the state of Q output of each FF is passed through the gate to the D input of the following FF. When clock pulse occurs, the data bits are effectively shifted one place to the right.
- A LOW Right/Left control input enables the AND gates G_5, G_6, G_7 and G_8 and disables the AND gates G_1, G_2, G_3 and G_4 and the Q output of each FF is passed to the D input of the preceding FF. When clock pulse occurs the data bits are then effectively shifted one place to the left.

  So, the circuit works as a bidirectional shift register



Logic diagram of 4- bit bidirectional shift register

.

UNIVERSAL SHIFT REGISTERS:-

- The register which has both shifts and parallel load capabilities, it is referred as a universal shift register. So, universal shift register is a bidirectional register, whose input can be either in serial form or in parallel form and whose output also can be either in serial form or parallel form.
- The universal shift register can be realized using multiplexers.
- The figure shows the logic diagram of a 4 bit universal shift register that has all the capabilities of a general shift register.

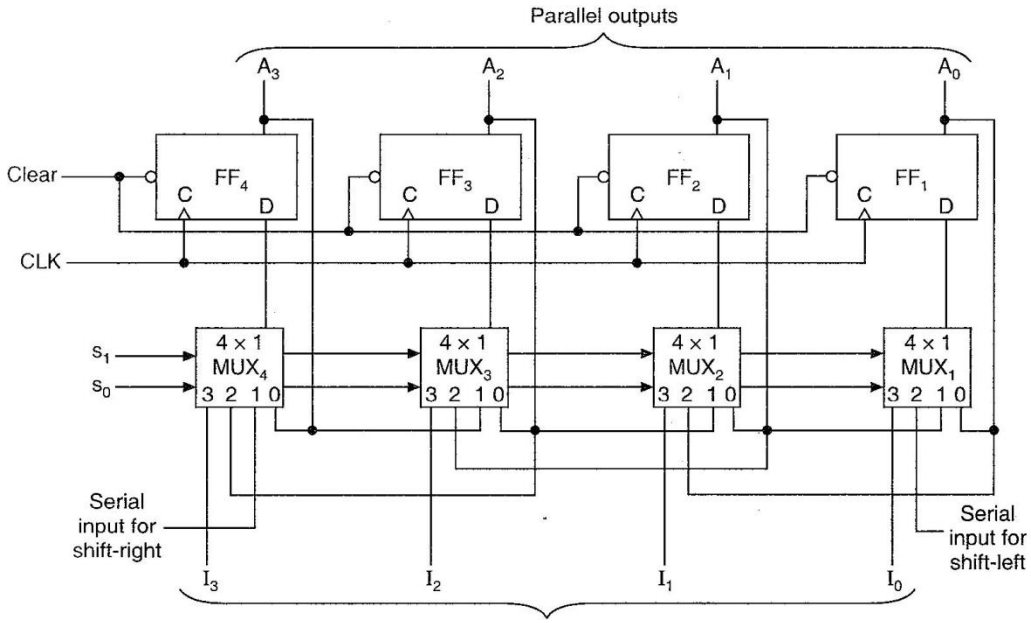


Fig- (a) 4 bit universal shift register

- It consists of four D flip- flops and four multiplexers.
- The four multiplexers have two common selection inputs S_1 and S_0.
- Input 0 in each multiplexer is selected when S_1S_0 = 00, input 1 is selected when S_1S_0 = 01, and input 2 is selected when S_1S_0 = 10 and input 3 is selected when S_1S_0= 11.
- The selection inputs control the mode of operation of the register is according to the function entries shown in the table.
- When S_1S_0 = 00 the present value of the register is applied to the D inputs **o**f flip-flops. This condition forms a path from the output of each FF into the input of the same FF.
- The next clock edge transfers into each FF the binary value it held previously, and no change of state occurs.
- When S_1S_0 = 01, terminal 1 of the multiplexer inputs have a path of the D inputs of the flip- flops. This causes a shift right operation, with serial input transferred into FF_4.
- When S_1S_0 = 10 a shift left operation results with the other serial input going into the FF_1.
- Finally when S_1S_0 = 11, the binary information on the parallel input lines is transferred into the register simultaneously during the next clock edge.

Functional table for the register of fig – a:

Mode control		
S_1	S_0	Register operation
0	0	No change
0	1	Shift right
1	0	Shift left
1	1	Parallel load

APPLICATIONS OF SHIFT REGISTERS:-

2. Time delays:

- In digital systems, it is necessary to delay the transfer of data until the operation of the other data have been completed, or to synchronize the arrival of data at processed a subsystem where it is with other data.
- A shift register can be used to delay the arrival of serial data by a specific number of clock pulses, since the number of stages corresponds to the number of clock pulses required to shift each bit completely through the register.
- The total time delay can be controlled by adjusting the clock frequency and by the number of stages in the register.
- In practice, the clock frequency is fixed and the total delay can be adjusted only by controlling the number of stages through which the data is passed.

3. Serial / Parallel data conversion:

- Transfer of data in parallel form is much faster than that in serial form.
- Similarly the processing of data is much faster when all the data bits are available simultaneously. Thus in digital systems in which speed is important so to operate on data parallel form is used.
- When large data is to be transmitted over long distances, transmitting data on parallel lines is costly and impracticable.
- It is convenient and economical to transmit data in serial form, since serial data transmission requires only one line.

- Shift registers are used for converting serial data to parallel form, so that a serial input can be processed by a parallel system and for converting parallel data to serial form, so that parallel data can be transmitted serially.
- A serial in, parallel out shift register can be used to perform serial-to parallel conversion, and a parallel in, serial out shift register can be used to perform parallel- to –serial conversion.
- A universal shift register can be used to perform both the serial- to – parallel and parallel-to- serial data conversion.
- A bidirectional shift register can be used to reverse the order of data.

(CHAPTER-4)

8085 MICROPROCESSOR

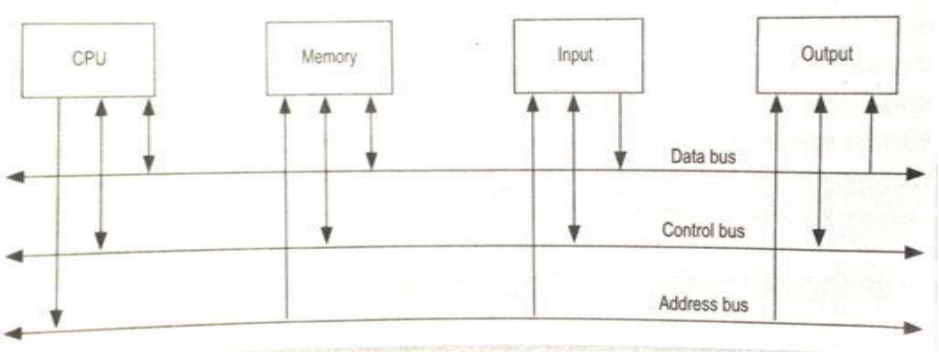
History of microprocessor:-

- The invention of the transistor in 1947 was a significant development in the world of technology.
- It could perform the function of a large component used in a computer in the early years.
- Shockley, Brattain and Bardeen are credited with this invention and were awarded the Nobel prize for the same.
- Soon it was found that the function this large component was easily performed by a group of transistors arranged on single platform.
- This platform, known as the integrated chip (IC), turned out to be a very crucial achievement and brought along a revolution in the use of computers.
- The use of microprocessors was limited to task-based operations specifically required for company projects such as the automobile sector.

- The 16 bit microprocessors started becoming a commercial sell-out in the 1980s with the first popular one being the TMS9900 of Texas Instruments.

INTRODUCTION TO MICROPROCESSOR AND MICROCOMPUTER

- A microprocessor is a programmable electronics chip that has computing and decision making capabilities similar to central processing unit of a computer.
- Any microprocessor-based systems having limited number of resources are called microcomputers.
- Nowadays, microprocessor can be seen in almost all types of electronics devices like mobile phones, printers washing machines etc.
- Microprocessors are also used in advanced applications like radars, satellites and flights.
- Due to the rapid advancements in electronic industry and large scale integration of devices results in a significant cost reduction and increase application of microprocessors and their derivatives.



(Fig.1 Microprocessor-based system)

Bit : A bit is a single binary digit.
Word : A word refers to the basic data size or bit size that can be processed by the arithmetic and logic unit of the processor. A 16-bit binary number is called a word in a 16-bit processor.
Bus : A bus is a group of wires/lines that carry similar information.•
System Bus : The system bus is a group of wires/lines used for communication between the microprocessor .
Memory Word : The number of bits that can be stored in a register or memory element is called a memory word.
Address Bus : It carries the address, which is a unique binary pattern used to identify• a memory location or an I/O port . For example, an eight bit address bus has eight lines and thus it can address 28 = 256 different locations. The locations in hexadecimal format can be written as 00H – FFH.
Data Bus : The data bus is used to transfer data between memory and processor or between I/O device and processor.For example, an 8-bit processor will generally have an 8-bit data bus and a 16-bit processor will have 16-bit data bus.
Control Bus : The control bus carry control signals, which consists of signals for selection of memory or I/O device from the given address, direction of data transfer and synchronization of data transfer in case of slow devices.

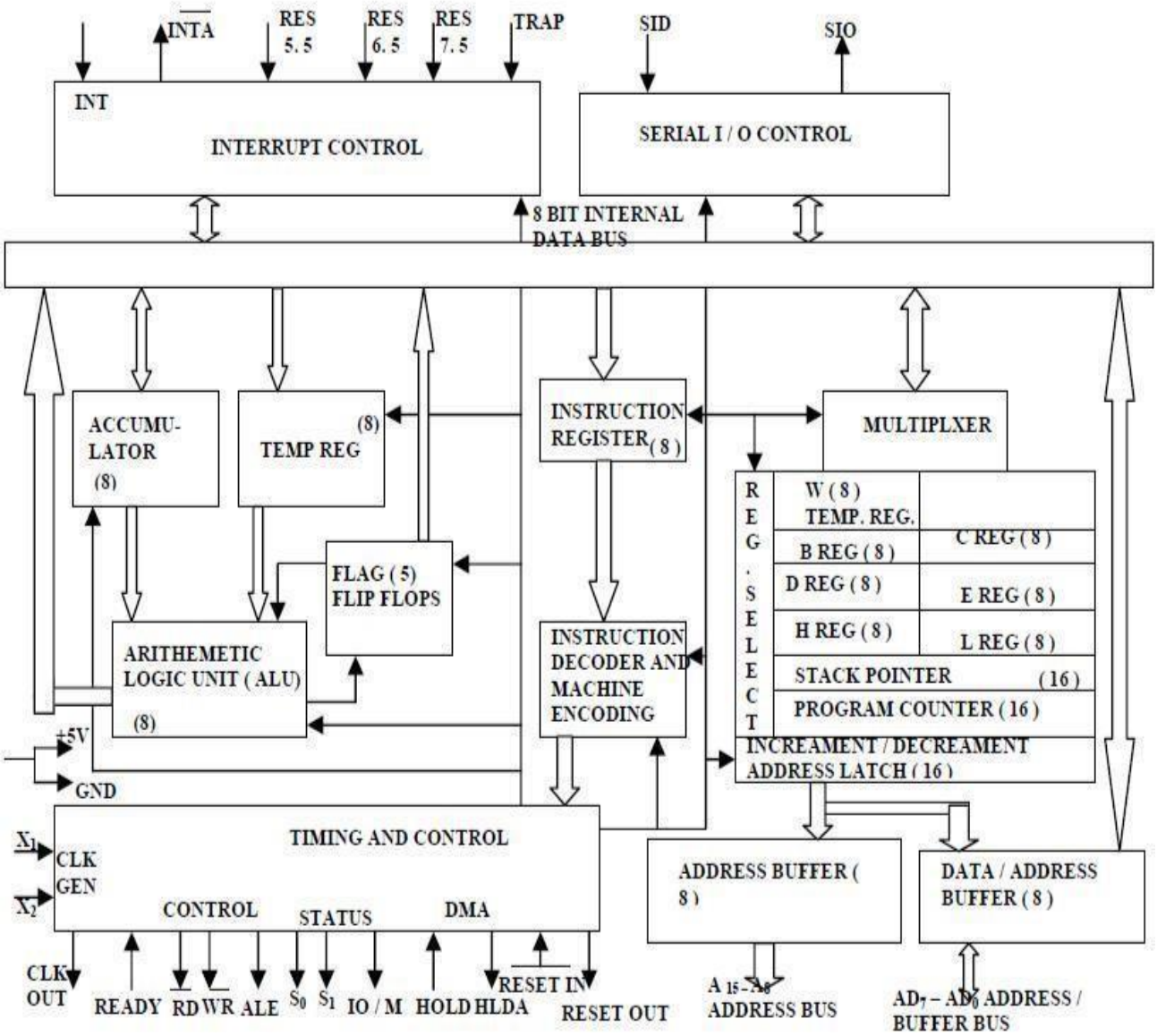
- A typical microprocessor consists of arithmetic and logic unit (ALU) in association with control unit to process the instruction execution.
- Almost all the microprocessors are based on the principle of store-program concept.
- In storeprogram concept, programs or instructions are sequentially stored in the memory locations that are to be executed.
- To do any task using a microprocessor, it is to be programmed by the user.
- So the programmer must have idea about its internal resources, features and supported instructions.
- Each microprocessor has a set of instructions, a list which is provided by the microprocessor manufacturer.
- The instruction set of a microprocessor is provided in two forms: binary machine code and mnemonics.
- Microprocessor communicates and operates in binary numbers 0 and 1.
- The set of instructions in the form of binary patterns is called a machine language and it is difficult for us to understand.
- Therefore, the binary patterns are given abbreviated names, called mnemonics, which forms the assembly language.
- The conversion of assembly-level language into binary machine-level language is done by using an application called assembler

8085 MICROPROCESSOR ARCHITECTURE

- The 8085 microprocessor is an 8-bit processor available as a 40-pin IC package and uses +5 V for power.
- It can run at a maximum frequency of 3 MHz. Its data bus width is 8-bit and address bus width is 16-bit, thus it can address 216 = 64 KB of memory.

- The internal architecture of 8085 is shown is fig

- The ALU performs the actual numerical and logical operations such as Addition (ADD), Subtraction (SUB), AND, OR etc.
- It uses data from memory and from Accumulator to perform operations.
- The results of the arithmetic and logical operations are stored in the accumulator.

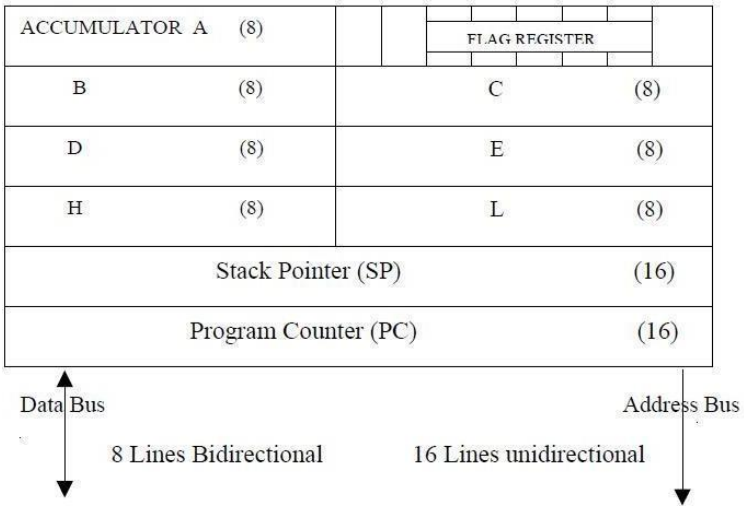


(Internal Architecture of 8085 Microprocessor)

REGISTERS

- The 8085 includes six registers, one accumulator and one flag register, as shown in Fig. 3.
- In addition, it has two 16-bit registers: stack pointer and program counter.
- The 8085 has six general-purpose registers to store 8-bit data; these are identified as B, C, D, E, H and L. they can be combined as register pairs - BC, DE and HL to perform some

- 16-bit operations. The programmer can use these registers to store or copy data into the register by using data copy instructions.



(Register organization)

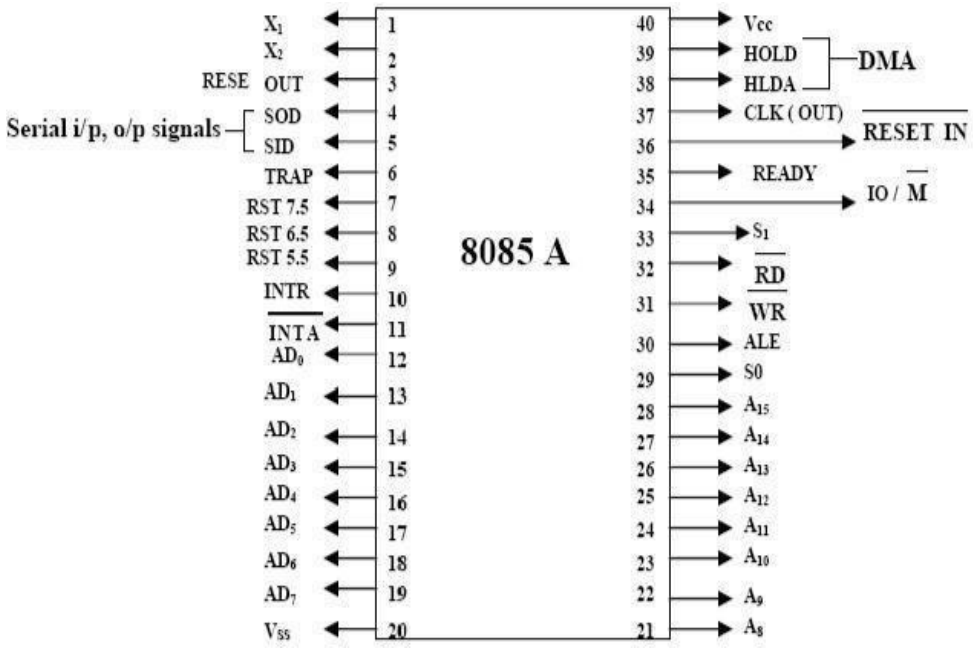
8085 PIN DESCRIPTION

Properties:

- It is a 8-bit microprocessor
- Manufactured with N-MOS technology
- 40 pin IC package
- It has 16-bit address bus and thus has 2^{16} = 64 KB addressing capability.
- Operate with 3 MHz single-phase clock
- +5 V single power supply

The logic pin layout and signal groups of the 8085nmicroprocessor are shown in Fig. 6. All the signals are classified intosix groups:

- Address bus
- Data bus
- Control & status signals
- Power supply and frequency signals
- Externally initiated signals
- Serial I/O signals



(8085 microprocessor pin layout and signal groups)

Address and Data Buses:

- A8 – A15 (output, 3-state): Most significant eight bits of memory addresses and the eight bits of the I/Oaddresses. These lines enter into tri-state high impedance state during HOLD and HALT modes.
- AD0 – AD7 (input/output, 3-state): Lower significant bits of memory addresses and the eight bits of the I/Oaddresses during first clock cycle.
- Behaves as data bus during third and fourth clock cycle. These lines enter into tri-state high impedance state during HOLD andHALT modes.

Control & Status Signals:

- ALE: Address latch enable
- RD : Read control signal.
- WR : Write control signal.
- IO/M , S1 and S0 : Status signals

PowerSupply & Clock Frequency:

- Vcc: +5 V power supply
- Vss: Ground reference
- X1, X2: A crystal having frequency of 6 MHz is connected at these two pins
- CLK: Clock output

Externally Initiated and Interrupt Signals:

- **RESET IN** : When the signal on this pin is low, the PC is set to 0, the buses are tri-stated and the processor is reset.
- **RESET OUT**: This signal indicates that the processor is being reset. The signal can be used to reset other devices.
- **READY**: When this signal is low, the processor waits for an integral number of clock cycles until it goes high.
- **HOLD**: This signal indicates that a peripheral like DMA (direct memory access) controller is requesting the use of address and data bus.
- **HLDA**: This signal acknowledges the HOLD request.
- **INTR**: Interrupt request is a general-purpose interrupt.
- **INTA** : This is used to acknowledge an interrupt.
- **RST 7.5, RST 6.5, RST 5,5 – restart interrupt**: These are vectored interrupts and have highest priority than INTR interrupt.
- **TRAP**: This is a non-maskable interrupt and has the highest priority.

Serial I/O Signals:

- SID: Serial input signal. Bit on this line is loaded to D7 bit of register A using RIM instruction.
- SOD: Serial output signal. Output SOD is set or reset by using SIM instruction.

Program Counter (PC)

- This 16-bit register deals with sequencing the execution of instructions.
- This register is a memory pointer.
- The microprocessor uses this register to sequence the execution of the instructions.
- The function of the program counter is to point to the memory address from which the next byte is to be fetched.
- When a byte is being fetched, the program counter is automatically incremented by one to point to the next memory location.

Stack Pointer (SP)

- The stack pointer is also a 16-bit register, used as a memory pointer.
- It points to a memory location in R/W memory, called stack.
- The beginning of the stack is defined by loading 16-bit address in the stack pointer.

Instruction Register/Decoder

- It is an 8-bit register that temporarily stores the current instruction of a program.
- Latest instruction sent here from memory prior to execution.
- Decoder then takes instruction and decodes or interprets the instruction.
- Decoded instruction then passed to next stage.

Control Unit

Generates signals on data bus, address bus and control bus within microprocessor to carry out the instruction, which has been decoded. Typical buses and their timing are described as follows:

- **Data Bus**: Data bus carries data in binary form between microprocessor and other external units such as memory.

It is used to transmit data i.e. information, results of arithmetic etc between memory and the microprocessor. Data bus is bidirectional in nature. The data bus width of 8085 microprocessor is 8-bit i.e. 2^8 combination of binary digits and are typically identified as D0 – D7. Thus size of the data bus determines what arithmetic can be done. If only 8-bit wide then largest number is 11111111 (255 in decimal). Therefore, larger numbers have to be broken down into chunks of 255. This slows microprocessor.

- *Address Bus*: The address bus carries addresses and is one way bus from microprocessor to the memory or other devices. 8085 microprocessor contain 16-bit address bus and are generally identified as A0 - A15. The higher order address lines (A8 – A15) are unidirectional and the lower order lines (A0 – A7) are multiplexed (time- shared) with the eight data bits (D0 – D7) and hence, they are bidirectional.
- ***Control Bus*:** Control bus are various lines which have specific functions for coordinating and controlling microprocessor operations. The control bus carries control signals partly unidirectional and partly bidirectional. The following control and status signals are used by 8085 processor:
  I. **ALE (output):** Address Latch Enable is a pulse that is provided when an address appears on the AD0 – AD7 lines, after which it becomes 0.
  II. **RD (active low output):** The Read signal indicates that data are being read from the selected I/O or memorydevice and that they are available on the data bus.
  III. **$\overline{WR}$ (active low output):** The Write signal indicates that data on the data bus are to be written into aselectedmemory or I/O location.
  IV. **IO/$\overline{M}$ (output):** It is a signal that distinguished between a memory operation and an I/O operation. When IO/$\overline{M}$ = 0 it is a memory operation and IO/$\overline{M}$ = 1 it is an I/O operation.
  V. **S1 and S0 (output):** These are status signals used to specify the type of operation being performed;they are listed in Table 1.

Table 1 Status signals and associated operations

S1	S0	States
0	0	Halt
0	1	Write
1	0	Read
1	1	Fetch

The schematic representation of the 8085 bus structure is as shown in Fig. 5. The microprocessor performs primarily fouroperations:

INSTRUCTION SET AND EXECUTION IN 8085

- Based on the design of the ALU provides and decoding unit, the microprocessor manufacturer microprocessor.
- The instruction set for every machine code and instruction set consists of both mnemonics.

- An instruction is a binary pattern designed inside a microprocessor to perform a specific function.
- The entire group of instructions that a microprocessor supports is called instruction set. Microprocessor instructions can be classified based on the parameters such functionality, length and operand addressing.

Classification based on functionality:

I. Data transfer operations: This group of instructions copies data from source to destination. The content of the source is not altered.

II. Arithmetic operations: Instructions of this group perform operations like addition, subtraction, increment & decrement. One of the data used in arithmetic operation is stored in accumulator and the result is also stored in accumulator.

III. Logical operations: Logical operations include AND, OR, EXOR, NOT. The operations like AND, OR and EXOR uses two operands, one is stored in accumulator and other can be any register or memory location. The result is stored in accumulator. NOT operation requires single operand, which is stored in accumulator.

IV. Branching operations: Instructions in this group can be used to transfer program sequence from one memorylocation to another either conditionally or unconditionally.

V. Machine control operations: Instruction in this group control execution of other instructions and controloperations like interrupt, halt etc.

Classification based on length:

I. One-byte instructions: Instruction having one byte in machine code. Examples are depicted in Table 2.

I. Two-byte instructions: Instruction having two byte in machine code. Examples are depicted in Table 3

II. Three-byte instructions: Instruction having three byte in machine code. Examples are depicted in Table 4.

Table 2 Examples of one byte instructions

Opcode	Operand	Machine code/Hex code
MOV	A, B	78
ADD	M	86

Table 3 Examples of two byte instructions

Opcode	Operand	Machine code/Hex code	Byte description
MVI	A, 7FH	3E	First byte
		7F	Second byte
ADI	0FH	C6	First byte
		0F	Second byte

Table 4 Examples of three byte instructions

Opcode	Operand	Machine code/Hex code	Byte description
JMP	9050H	C3	First byte
		50	Second byte
		90	Third byte
LDA	8850H	3A	First byte
		50	Second byte
		88	Third byte

Addressing Modes in Instructions:

The process of specifying the data to be operated on by the instruction is called addressing. The various formats forspecifying operands are called addressing modes. The 8085 has the following five types of addressing:

1. Immediate addressing
2. Memory direct addressing
3. Register direct addressing
4. Indirect addressing
5. Implicit addressing

Immediate Addressing:

In this mode, the operand given in the instruction - a byte or word – transfers to the destination register or memorylocation.

Ex: MVI A, 9AH

- The operand is a part of the instruction.
- The operand is stored in the register mentioned in the instruction.

Memory Direct Addressing:

Memory direct addressing moves a byte or word between a memory location and register. The memory location address isgiven in the instruction.

Ex: LDA 850FH

This instruction is used to load the content of memory address 850FH in the accumulator.

Register Direct Addressing:

Register direct addressing transfer a copy of a byte or word from source register to

destination register. Ex: MOV B, C

It copies the content of register C to register B.

Indirect Addressing:

Indirect addressing transfers a byte or word between a register and a

memory location. Ex: MOV A, M

Here the data is in the memory location pointed to by the contents of HL pair. The data is moved to the accumulator.

Implicit Addressing

In this addressing mode the data itself specifies the data to be

operated upon. Ex: CMA

The instruction complements the content of the accumulator. No specific data or operand is mentioned in the instruction.

INSTRUCTION EXECUTION AND TIMING DIAGRAM:

Each instruction in 8085 microprocessor consists of two part- operation code (opcode) and operand. The opcode is a command such as ADD and the operand is an object to be operated on, such as a byte or the content of a register.

Instruction Cycle: The time taken by the processor to complete the execution of an instruction. An instruction cycle consists of one to six machine cycles.

Machine Cycle: The time required to complete one operation; accessing either the memory or I/O device. A machine cycle consists of three to six T-states.

T-State: Time corresponding to one clock period. It is the basic unit to calculate execution of instructions or programs in a processor.

To execute a program, 8085 performs various operations as:

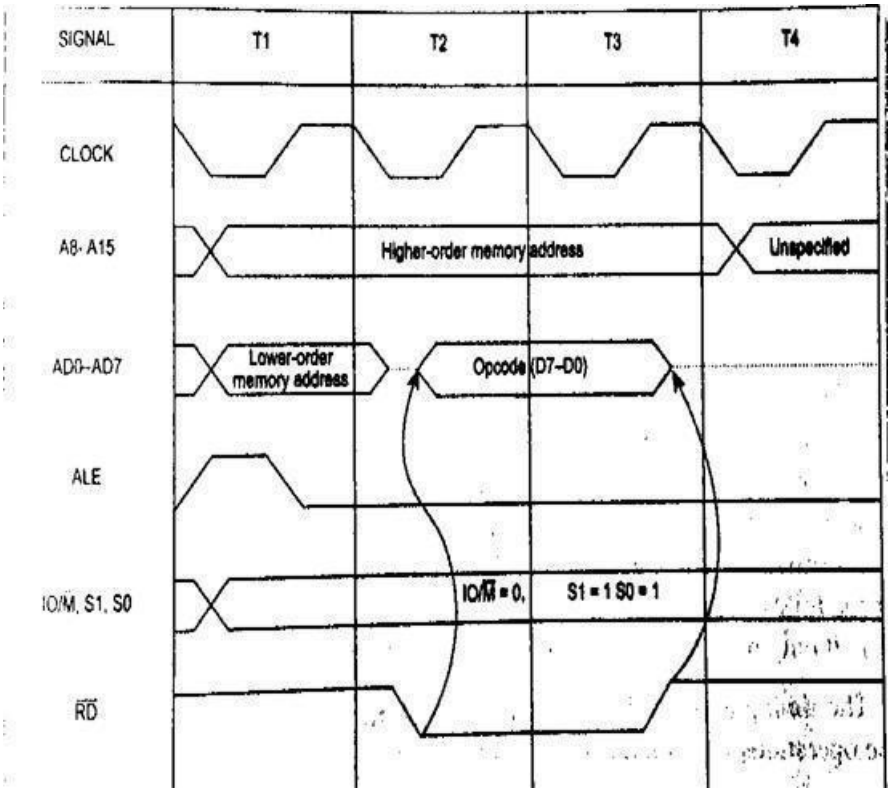
- Opcode fetch
- Operand fetch
- Memory read/write
- I/O read/write

External communication functions are:

- Memory read/write
- I/O read/write
- Interrupt request acknowledge

Opcode Fetch Machine Cycle:

It is the first step in the execution of any instruction. The timing diagram of this cycle is given in Fig.



The following points explain the various operations that take place and the signals that are changed during the execution ofopcode fetch machine cycle:

T1 clock cycle

i. The content of PC is placed in the address bus; AD0 - AD7 lines contains lower bit address and A8 – A15contains higher bit address.

ii. IO/M signal is low indicating that a memory location is being accessed. S1 and S0 also changed to the levels asindicated in Table 1.

ii. ALE is high, indicates that multiplexed AD0 – AD7 act as lower order bus.

T2 clock cycle

i. Multiplexed address bus is now changed to data bus.

ii. The RD signal is made low by the processor. This signal makes the memory device load the data bus with thecontents of the location addressed by the processor.

T3 clock cycle

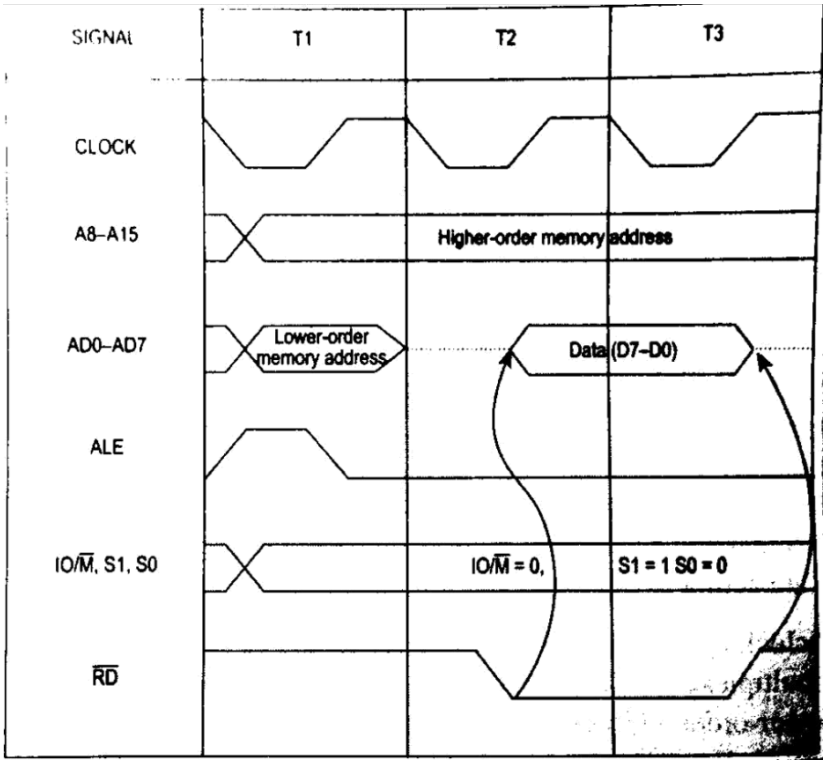
i. The opcode available on the data bus is read by the processor and moved to the instruction register.
ii. The $\overline{RD}$ signal is deactivated by making it logic 1.

T4 clock cycle

i. The processor decode the instruction in the instruction register and generate the necessary control signals to execute the instruction. Based on the instruction further operations such as fetching, writing into memory etc takes place.

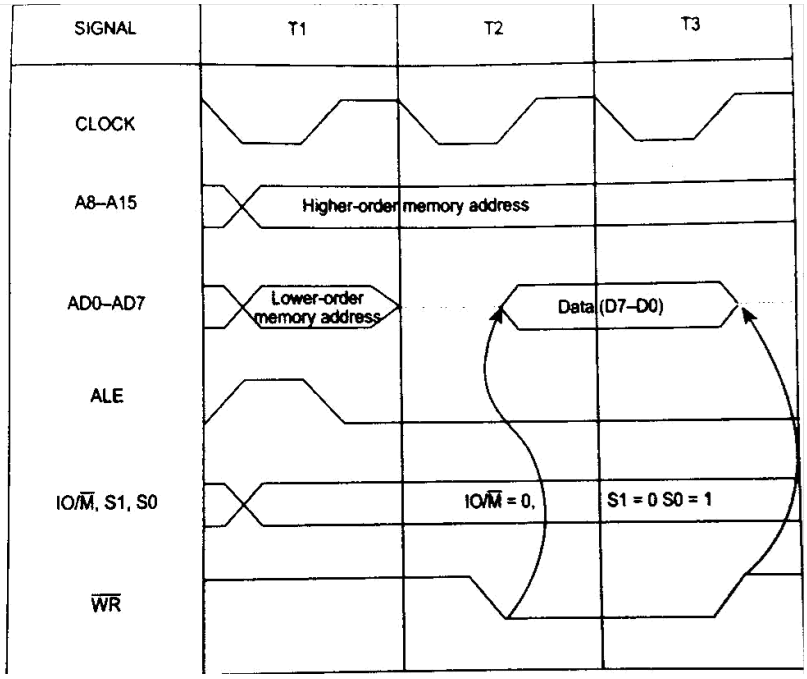
Memory Read Machine Cycle:

The memory read cycle is executed by the processor to read a data byte from memory. The machine cycle is exactly same to opcode fetch except:
a) It has three T-states
b) The S0 signal is set to 0.
The timing diagram of this cycle is given in Fig.



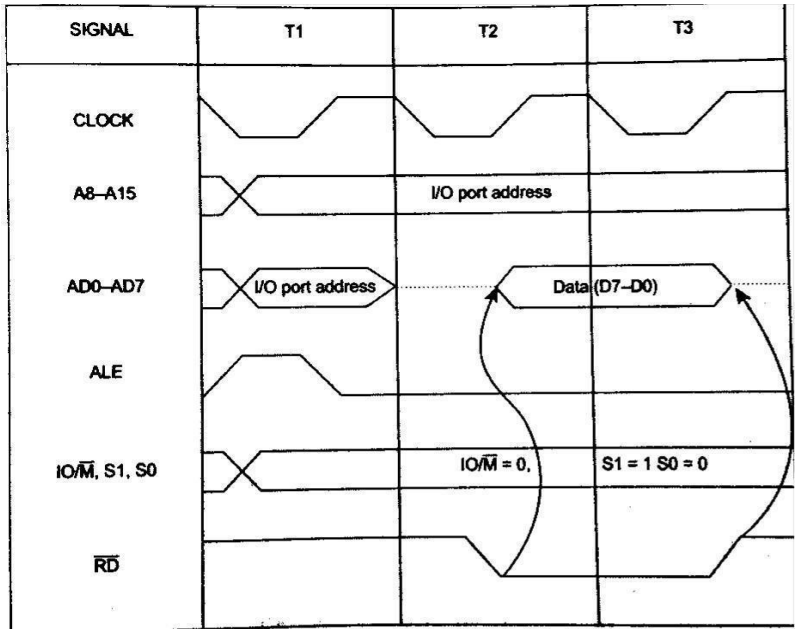
Memory Write Machine Cycle:

The memory write cycle is executed by the processor to write a data byte in a memory location. The processor takes three T-states and WR signal is made low. The timing diagram of this cycle is given in Fig.



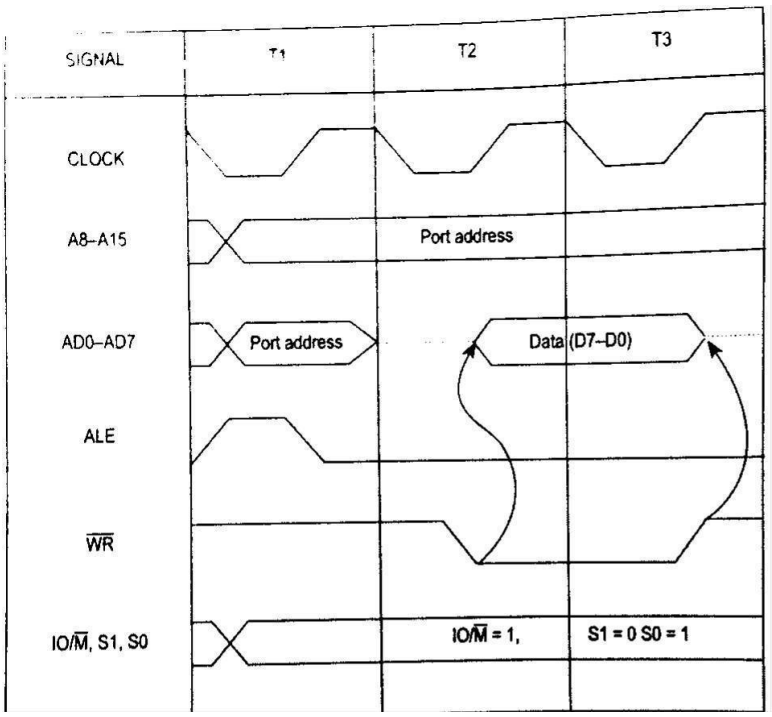
I/O Read Cycle:

The I/O read cycle is executed by the processor to read a data byte from I/O port or from peripheral, which is I/O mapped in the system. The 8-bit port address is placed both in the lower and higher order address bus. The processor takes three T- states to execute this machine cycle. The timing diagram of this cycle is given in Fig.



I/O Write Cycle:

The I/O write cycle is executed by the processor to write a data byte to I/O port or to a peripheral, which is I/O mapped in the system. The processor takes three T-states to execute this machine cycle. The timing diagram of this cycle is given in Fig.



INTERRUPTS

Interrupt Structure:

Interrupt is the mechanism by which the processor is made to transfer control from its current program execution to another program having higher priority. The interrupt signal may be given to the processor by any external peripheral device.

The program or the routine that is executed upon interrupt is called interrupt service routine (ISR). After execution of ISR, the processor must return to the interrupted program. Key features in the interrupt structure of any microprocessor are as follows:

i. Number and types of interrupt signals available.
ii. The address of the memory where the ISR is located for a particular interrupt signal. This address is calledinterrupt vector address (IVA).
iii. Masking and unmasking feature of the interrupt signals.
iv. Priority among the interrupts.
v. Timing of the interrupt signals.
vi. Handling and storing of information about the interrupt program (status information).

Types of Interrupts:

Interrupts are classified based on their maskability, IVA and source. They are classified as:

i. Vectored and Non-Vectored Interrupts
   - Vectored interrupts require the IVA to be supplied by the external device that gives the interruptsignal. This technique is vectoring, is implemented in number of ways.
   - Non-vectored interrupts have fixed IVA for ISRs of different interrupt signals.

ii. Maskable and Non-Maskable Interrupts
   - Maskable interrupts are interrupts that can be blocked. Masking can be done by software or hardwaremeans.
   - Non-maskable interrupts are interrupts that are always recognized; the corresponding ISRs areexecuted.

iii. Software and Hardware Interrupts
   - Software interrupts are special instructions, after execution transfer the control to predefined ISR.
   - Hardware interrupts are signals given to the processor, for recognition as an interrupt and execution ofthe corresponding ISR.

Interrupt Handling Procedure:

The following sequence of operations takes place when an interrupt signal is recognized:

i. Save the PC content and information about current state (flags, registers etc) in the stack.
ii. Load PC with the beginning address of an ISR and start to execute it.
iii. Finish ISR when the return instruction is executed.
iv. Return to the point in the interrupted program where execution was interrupted.

Interrupt Sources and Vector Addresses in 8085:

Software Interrupts:

8085 instruction set includes eight software interrupt instructions called Restart (RST) instructions. These are one byte instructions that make the processor execute a subroutine at predefined locations. Instructions and their vector addressesare given in Table 6.

Table 6 Software interrupts and their vector addresses

Instruction	Machine hex code	Interrupt Vector Address
RST 0	C7	0000H
RST 1	CF	0008H
RST 2	D7	0010H
RST 3	DF	0018H
RST 4	E7	0020H
RST 5	EF	0028H
RST 6	F7	0030H
RST 7	FF	0032H

The software interrupts can be treated as CALL instructions with default call locations. The concept of priority does not apply to software interrupts as they are inserted into the program as instructions by the programmer and executed by the processor when the respective program lines are read.

Hardware Interrupts and Priorities:

8085 have five hardware interrupts – INTR, RST 5.5, RST 6.5, RST 7.5 and TRAP. Their IVA and priorities are given in Table 7.

Table 7 Hardware interrupts of 8085

Interrupt	Interrupt vector address	Maskable or non-maskable	Edge or level Triggered	priority
TRAP	0024H	Non-makable	Level	1
RST 7.5	003CH	Maskable	Rising edge	2
RST 6.5	0034H	Maskable	Level	3
RST 5.5	002CH	Maskable	Level	4
INTR	Decided by hardware	Maskable	Level	5

8255 PPI

8255 is a programmable I/O device that acts as interface between peripheral devices and the microprocessor for parallel data transfer. 8255 PPI (programmable peripheral interface) is programmed in way so as to have transfer of data in different conditions according to the need of the system.
In 8255, 24 pins are assigned to the I/O ports. Basically it has three, 8-bit ports that are used for simple or interrupt I/O operations.

The three ports are **Port A, Port B and Port C** and as each port has 8 lines, but the 8 bits of port C is divided into 2 groups of 4-bit each. These are given as port C lower i.e., $PC_3 – PC_0$ and port C upper i.e., $PC_7 – PC_4$. And are arranged in group of 12 pins each thus designated as Group A and Group B.
The two modes in which 8255 can be programmed are as follows:

1. Bit set/reset mode
2. I/O mode
   The bits of port C gets set or reset in the BSR mode. The other mode of 8255 i.e., I/O mode is further classified into:
   Mode 0: Simple input/output
   Mode 1: Input output with handshaking
   Mode 2: Bidirectional I/O handshaking

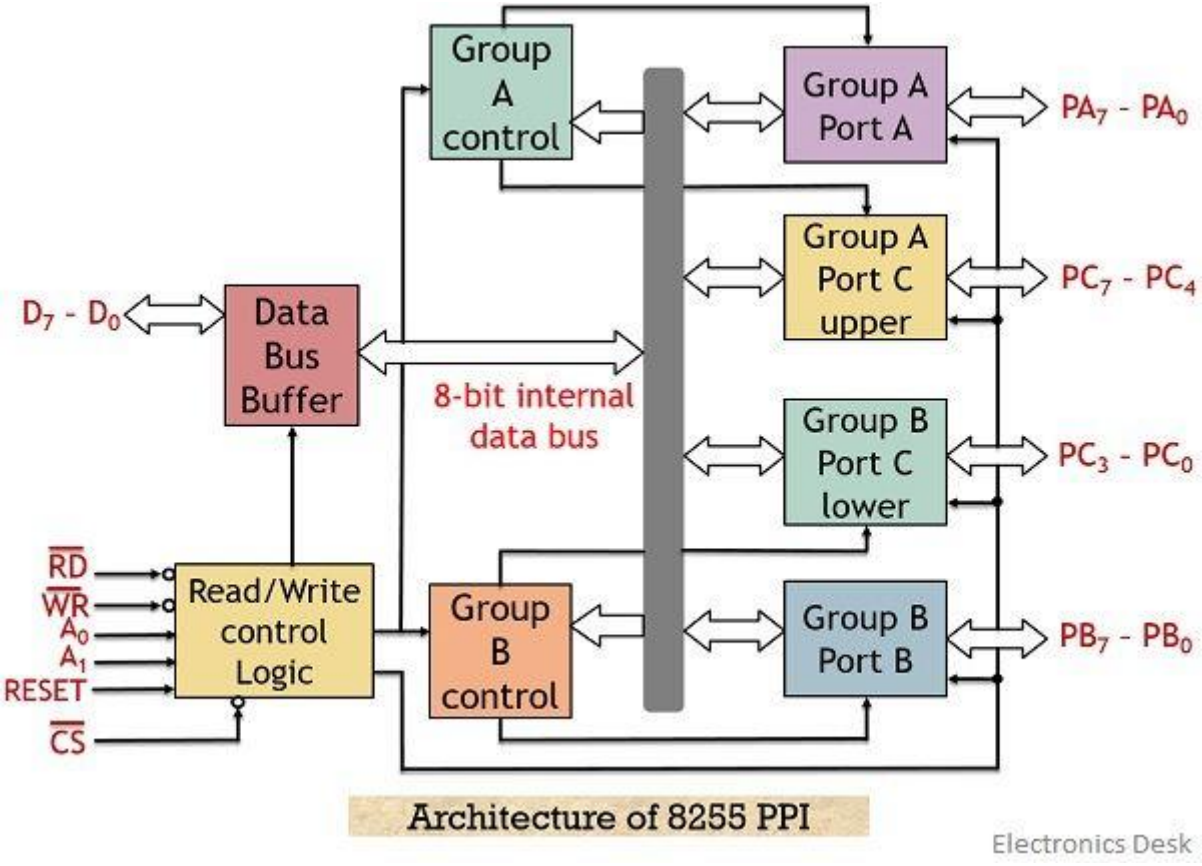
Mode 1 and Mode 2 both are same but the only difference is mode 1 does not support bidirectional handshaking.

This means if 8255 is programmed to mode 1 input, then it will particularly be connected to an input device and performs the input handshaking with the processor.

But if it is programmed to mode 2 then due to bidirectional nature, the PPI will perform both input and output operation with the processor according to the command received.

Architecture of 8255 PPI

The figure below represents the architectural representation of 8255 PPI:



Data bus buffer: It is used to connect the internal bus of 8255 with the system bus so as to establish proper interfacing between the two. The data bus buffer allows the read/write operation to be performed from/to the CPU.
The buffer allows the passing of data from ports or control register to CPU in case of write operation and from CPU to ports or status register in case of read operation.

Read/ Write control logic: This unit manages the internal operations of the system. This unit holds the ability to control the transfer of data and control or status words both internally and externally.
Whenever there exists a need for data fetch then it accepts the address provided by the processor through the bus and immediately generates command to the 2 control groups for the particular operation.

Group A and Group B control: These two groups are handled by the CPU and functions according to the command generated by the CPU. The CPU sends control words to the group A and group B control and they in turn sends the appropriate command to their respective port.

As we have discussed that group A has the access of the port A and higher order bits of port C.While groupB controls port B with the lower order bits of port C.

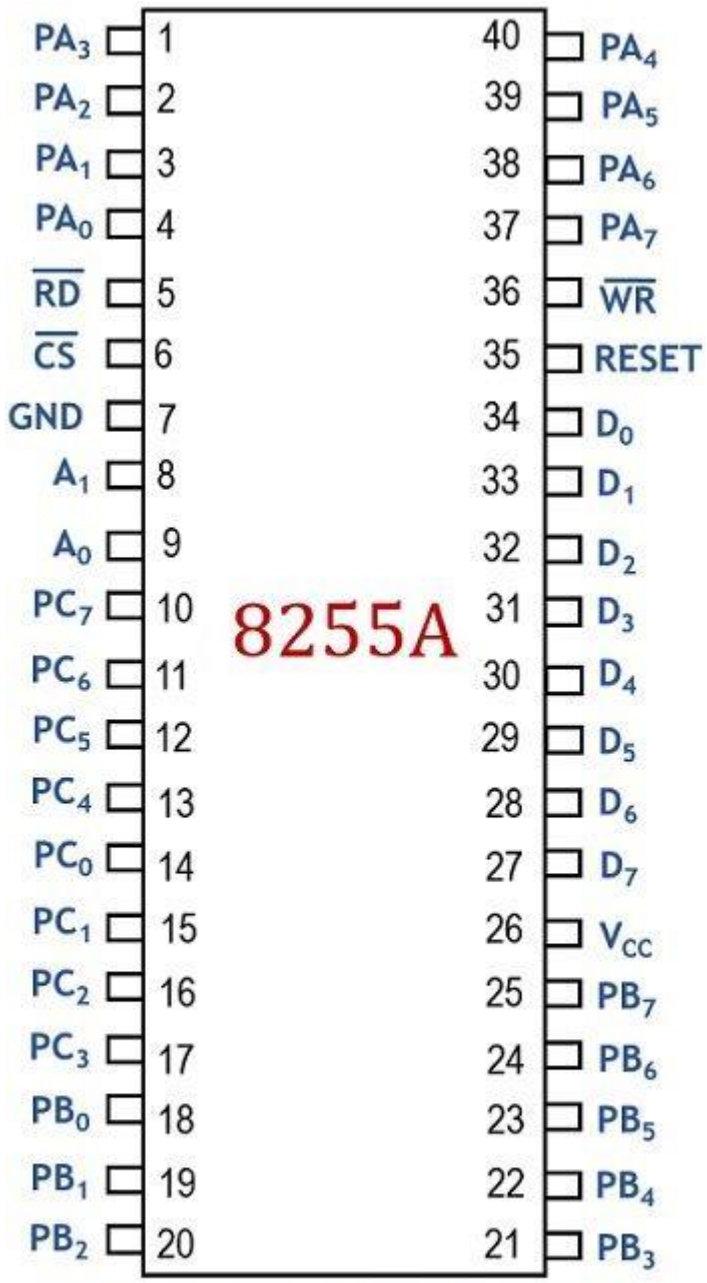
CS: It stands for chip select. A low signal at this pin shows the enabling of communication between the 8255 and the processor. More specifically we can say that the data transfer operation gets enabled by an active low signal at this pin.
RD – It is the signal used for read operation. A low signal at this pin shows that CPU is performing read operation at the ports or status word. Or we can say that 8255 is providing data or information to the CPU through data buffer.
WR – It shows write operation. A low signal at this pin allows the CPU to perform write operation over the ports or control register of 8255 using the data bus buffer.
A_0 and A_1: These are basically used to select the desired port among all the ports of the 8255 and it do so by forming conjunction with RD and WR. It forms connection with the LSB of the address bus.
The table below shows the operation of the control signals:

A_1	A_0	RD'	WR'	CS'	Input/Output Operation
0	0	0	1	0	Port A - Data Bus
0	1	0	1	0	Port B - Data Bus
1	0	0	1	0	Port C - Data Bus
0	0	1	0	0	Data Bus - Port A
0	1	1	0	0	Data Bus - Port B
1	0	1	0	0	Data Bus - Port C
1	1	1	0	0	Data Bus - Control register

Reset: It is an active high signal that shows the resetting of the PPI. A high signal at this pin clears the control registers and the ports are set in the input mode.
Initializing the ports to input mode is done to prevent circuit breakdown. As in case of reset condition, if the ports are initialized to output mode then there exist chances of destruction of 8255 along with the processor.

Pin Diagram of 8255 PPI

The figure below represents the 40 pin configuration of 8255 programmable peripheral interface:



Pin Diagram of 8255 PPI

Electronics Desk

As we can see that pin number 27 to 34 is allotted to data bus. These are tri-state data bus lines of bidirectional nature that connects 8255 with the processor.These are used to allows flow of data and control or status word between the 8255 and the processor.

The external voltage required to drive the circuit is provided at pin number 26 i.e., V_{CC}. Also pin number 7 is the ground connection of the circuit.
Out of 40 pins, 24 pins are allotted to I/O ports. Rest of the pins are allotted to the signals discussed above.

Modes of Operation

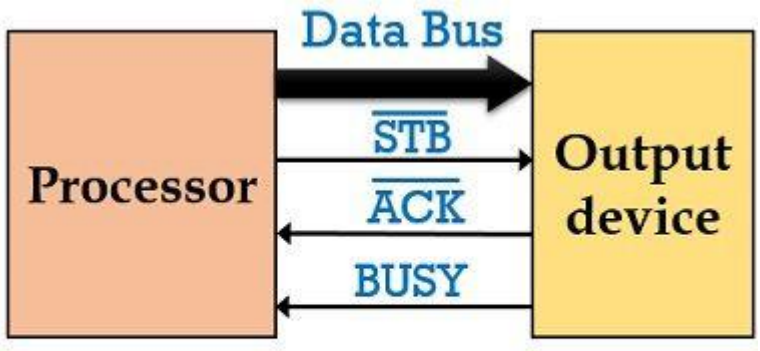
As we have already discussed that 8255 has two modes of operation. These are as follows:

Bit Set-Reset mode: When port C is utilized for control or status operation, then by sending an OUT instruction, each individual bit of port C can be set or reset.
I/O mode: As we know that the I/ O mode is sub-classified into 3 modes. So, let us now discuss the 3 modes here.
Mode 0: Input/Output mode
This mode is the simple input output mode of 8255 which allows the programming of each port as either input or output port. The input/output feature of mode 0 includes:

- It does not support handshaking or interrupt capability.
- The input ports are buffered while outputs are latched.

Mode 1: Input/Output with handshaking
Mode 1 of 8255 supports handshaking with the ports programmed as either input or output mode. We know that it is not necessary that all the time the data is transferred between two devices operating at same speed. So, handshaking signals are used to synchronize the data transfer between two devices that operates at different speeds.

The figure below shows the data transferring between CPU and an output device having different operating speeds:



Data Transfer using handshaking signals

- Here STB signal is used to inform the output device that data is available on the data bus by the processor.
- Here port A and port B can be separately configured as either input or output port.
- Both the port utilizes 3-3 lines of port C for handshaking signals. The rest two lines operates as input/output port.
- It supports interrupt logic.
- The data at the input or output ports are latched.

Mode 2: Bidirectional I/O port with handshaking

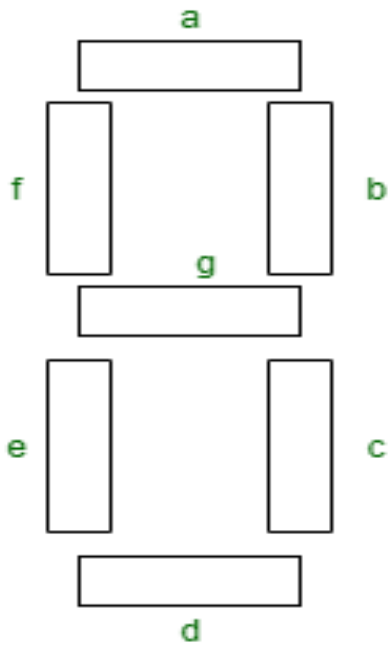
In this mode, the ports can be utilized for the bidirectional flow of information by handshaking signals.The pins of group A can be programmed to acts as bidirectional data bus and the port C upper (PC_7 – PC_4) are used by the handshaking signal. The rest 4 lower port C bits are utilized for I/O operations.
As the data bus exhibits bidirectional nature thus when the peripheral device request for a data input only then the processor load the data in the data bus. Port B can be programmed in mode 0 and 1. And in mode 1 the lower bits of port C of group B are used for handshaking signals.

So, from the above discussion it is clear that how interfacing of peripheral devices is performed with the processor.

Seven Segment Displays

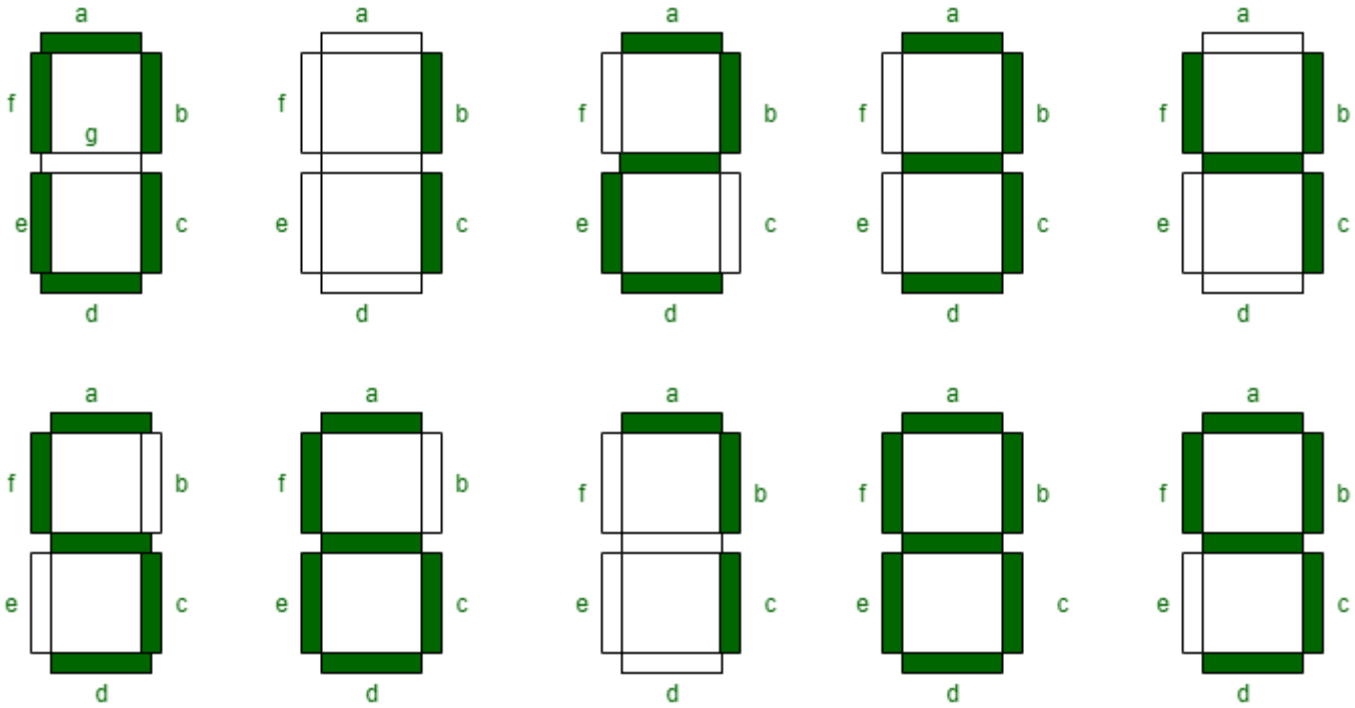
Light Emitting Diode (LED) is the most widely used semiconductor which emits either visible light or invisible infrared light when forward biased. Remote controls generate invisible light. A Light emitting diode (LED) is an optical electrical energy into light energy when voltage is applied.

Seven Segment Displays:
Seven segment displays are the output display device that provide a way to display information in the form of image or text or decimal numbers which is an alternative to the more complex dot matrix displays. It is widely used in digital clocks, basic calculators, electronic meters, and other electronic devices that display numerical information. It consists of seven segments of light emitting diodes (LEDs) which is assembled like numerical 8.



Working of Seven Segment Displays:

The number 8 is displayed when the power is given to all the segments and if you disconnect the power for ‘g’, then it displays number 0. In a seven segment display, power (or voltage) at different pins can be applied at the same time, so we can form combinations of display numerical from 0 to 9. Since seven segment displays can not form alphabet like X and Z, so it can not be used for alphabet and it can be used only for displaying decimal numerical magnitudes. However, seven segment displays can form alphabets A, B, C, D, E, and F, so they can also used for representing hexadecimal digits.



We can produce a truth table for each decimal digit

Decimal Digit	Individual Segments Illuminated						
	a	b	c	d	e	f	g
0	1	1	1	1	1	1	0
1	0	1	1	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1
5	1	0	1	1	0	1	1
6	1	0	1	1	1	1	1
7	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1
9	1	1	1	1	0	1	1

Therefore, Boolean expressions for each decimal digit which requires respective light emitting diodes (LEDs) are ON or OFF. The number of segments used by digit: 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9 are 6, 2, 5, 5, 4, 5, 6, 3, 7, and 6 respectively. Seven segment displays must be controlled by other external devices where different types of microcontrollers are useful to communicate with these external devices, like switches, keypads, and memory.

Types of Seven Segment Displays:
According to the type of application, there are two types of configurations of seven segment displays: common anode display and common cathode display.

1. In common cathode seven segment displays, all the cathode connections of LED segments are connected together to logic 0 or ground. We use logic 1 through a current limiting resistor to forward bias the individual anode terminals a to g.
2. Whereas all the anode connections of the LED segments are connected together to logic 1 in common anode seven segment display. We use logic 0 through a current limiting resistor to the cathode of a particular segment a to g.

Common anode seven segment displays are more popular than cathode seven segment displays, because logic circuits can sink more current than they can source and it is the same as connecting LEDs in reverse.

Applications of Seven Segment Displays:
Common applications of seven segment displays are in:
1. Digital clocks
2. Clock radios
3. Calculators
4. Wrist watchers
5. Speedo meters

6. Motor-vehicle odometers
7. Radio frequency indicators