PNS SCHOOL OF ENGINEERING & TECHNOLOGY

Nishamani Vihar, Marshaghai, Kendrapara

LECTURE NOTES
ON
DIGITAL ELECTRONICS & MICROPROCESSOR

DEPARTMENT OF ELECTRICAL ENGINEERING

5TH SEMESTER

PREPARED BY

MR. ADITYA NARAYAN JENA

LECTURER IN ELECTRONICS & TELECOMMUNICATION

BASIC 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 hasplace value.

1.1 NUMBER SYSTEM:-

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

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

This will be interpreted as

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

where Y = value of the entire number

a_n = the value of the nth

digitr = radix

TYPES OF NUMBER SYSTEM:-

There are four types of number systems. They are

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

➢ 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.

DECIMAL NUMBER SYSTEM:-

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

➢ 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 termsare added to obtain decimal number.

Examples:

(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 thebinary point and proceeding towards left and right.

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

Examples:

(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

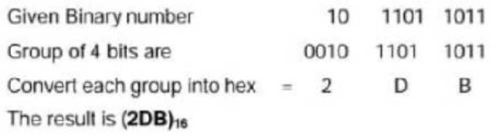
The result is $(5726.663)_8$

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

Example:

(i) Convert $(1011011011)_2$ into hexadecimal.

Solution:

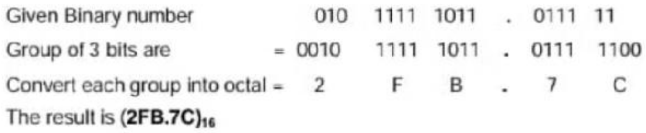


Given Binary number		10	1101	1011
Group of 4 bits are		0010	1101	1011
Convert each group into hex	=	2	D	B

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

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

Solution:



Given Binary number	010	1111	1011	.	0111	11
Group of 3 bits are	= 0010	1111	1011	.	0111	1100
Convert each group into octal =	2	F	B	.	7	C

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

2. DECIMAL NUMBER SYSTEM:-

(a) Decimal to binary conversion:-

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

Example:

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

Solution:

Divide the given decimal number successively by 2 read the integer part remainder upwards to getequivalent binary number. Multiply the fraction part by 2. Keep the integer in the product as it is and multiplythe 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

Answer of $(52)_{10}$ is $(110100)_2$.

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

Solution:

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

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

(b) Decimal to octal conversion:-

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

Example:

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

Solution:

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

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

(c) Decimal to hexadecimal conversion:-

The decimal to hexadecimal conversion is same as octal.

Example:

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

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

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

3. OCTAL NUMBER SYSTEM:-

(a) Octal to binary conversion:-

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

Example:

Convert $(367.52)_8$ into binary.

Solution:

Given Octal number is	6 7. 5 2
Convert each group octalto binary	110 111 . 101 010

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

(b) Octal to decimal conversion:-

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

For example: -

Convert $(4057.06)_8$ to decimal

Solution:

$$
\begin{aligned}
(4057.06)_8 &= 4 \times 8^3 + 0 \times 8^2 + 5 \times 8^1 + 7 \times 8^0 + 0 \times 8^{-1} + 6 \times 8^{-2} \\
&= 2048 + 0 + 40 + 7 + 0 + 0.0937 \\
&= (2095.\ 0937)_{10}
\end{aligned}
$$

Result is $(2095.0937)_{10}$

(c) Octal to hexadecimal conversion:-

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

For example :-

Convert $(756.603)_8$ to hexadecimal.

Solution :-

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

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

(4) HEXADECIMAL NUMBER SYSTEM :-

(a) Hexadecimal to binary conversion:-

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

For example:

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

Solution:

Given Hexadecimal number is 3 A 9 E . B 0 D

Convert each hexadecimal = 0011 1010 1001 1110 . 1011 0000
1101digit to 4 bit binary

Result of $(3A9E.B0D)_8$ is $(0011\ 1010\ 1001\ 1110.1011\ 0000\ 1101)_2$

(b) Hexadecimal to decimal conversion:-

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

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

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

$= 40960 + 0 + 240 + 9 + 0 +0.0546 + 0.0026$
$= (41209.0572)_{10}$

Result is $(41209.0572)_{10}$

(c) Hexadecimal to Octal conversion:-

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

For example :-

Convert $(B9F.AE)_{16}$ to octal.

Solution :-

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

Result is $(5637.534)_8$

ARITHEMATIC OPERATION

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 :-

$$\begin{array}{r} 100101 \\ + \ \underline{1101111} \\ \underline{10010100} \end{array}$$

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 :-

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

Result is $(0010.011)_2$

3. BINARY MULTIPLICATION:-

The binary multiplication rules are as follows0 x 0 = 0 ; 1 x 1 = 1 ; 1 x 0 = 0 ; 0 x 1 = 0

For example :-

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

```
           1 1 0 1
     x      1 1 0
          --------
            0 0 0 0
          1 1 0 1
  +     1 1 0 1
        ----------
        1 0 0 1 1 1 0
        -------------
```

Result is $(1001110)_2$

4. BINARY DIVISION:-

The binary division is very simple and similar to decimal number system. The division by '0' is meaningless.So we have only 2 rules
0 ÷ 1 = 0
1 ÷ 1 = 1

For example :-
Divide $(10110)_2$ by $(110)_2$.

Solution :-

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

Result is $(111.1)_2$

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

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

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.

GRAY CODE:-

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

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

BINARY- TO – GRAY CONVERSION:-

If an n-bit binary number is represented by B_n B_n.1 - - - - - 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

followsG$\oplus$ = B_n

G_{n-1} = B_n B_{n-1}

.

.

.

.

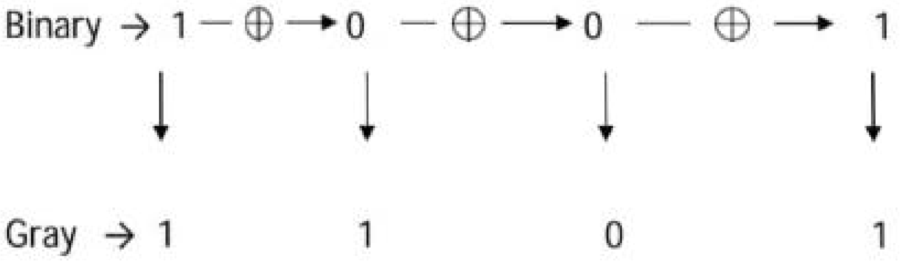
$G_1 = B_2 \oplus B_1$

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

For example

Convert the binary 1001 to the Gray code.

Solution :-`



The gray code is 1101

<u>GRAY- TO - BINARY CONVERSION</u>:-

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 $= G_n$

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

.

.

.

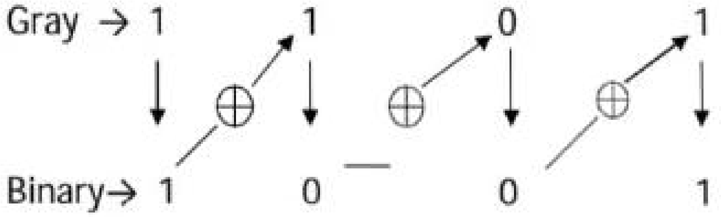
.

$B_1 = B_2 \oplus G$

For example :-

Convert the Gray code 1101 to the binary.

Solution :-



The binary code is 1001

1's COMPLEMENT REPRESENTATION :-

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

For example :-

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

Solution :-

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

Result is $(0011)_2$

2's COMPLEMENT REPRESENTATION :-

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

2's complement = 1's complement + 1

For example :-

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

Solution :-

```
   Given               1    0    1    0
1's complement is      0    1    0    1
                 +   ___________________1

2's complement         0    1    1    0
```

Result is $(0110)_2$

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:-

```
 11010               11010                      =  26
- 1000      =>     + 01111  (1's complement)    = - 16
     0
          Carry →  1 01001                        + 10
                   +     1
                     01010   = +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:-

```
  1010100                  1010100                       =   84
- 1010100      =>       +  0101100 (2's complement)         - 84
                         1 0000000 ( Ignore the carry)        0
                      =
                         0 (result = 0)
```

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

LOGIC GATES

LOGIC GATES:-

- Logic gates are the fundamental building blocks of digital systems.
- There are 3 basic types of gates AND, OR and NOT.
- Logic gates are electronic circuits because they are made up of a number of electronic devices andcomponents.
- Inputs and outputs of logic gates can occur only in 2 levels. These two levels are termed HIGH andLOW, 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 iscalled a truth table.

LEVEL LOGIC:-

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

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

Negative Logic:-

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

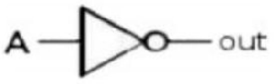
DIFFERENT TYPES OF LOGIC GATES:-

NOT GATE (INVERTER):-

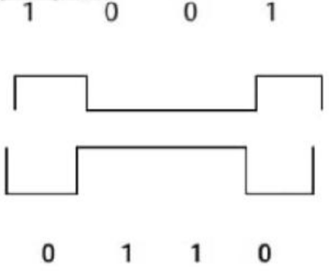
- A NOT gate, also called and inverter, has only one input and one output.
- It is a device whose output is always the complement of its input.
- The output of a NOT gate is the logic 1 state when its input is in logic 0 state and the logic 0 state whenits inputs is in logic 1 state.

IC No. :- 7404

Logic Symbol



Timing Diagram



Truth table

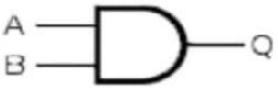
INPUT A	OUTPUT $\overline{A}$
0	1
1	0

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.

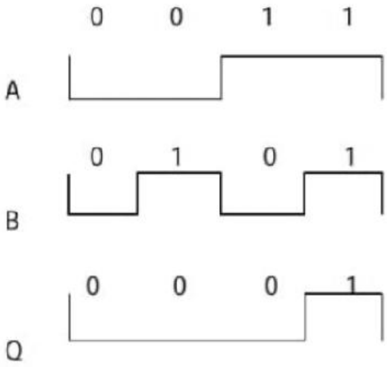
IC No.:- 7408

Logic Symbol



Truth Table

Timing Diagram

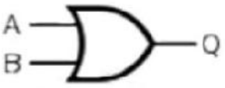


		OUTPUT
A	B	Q=A . B
0	0	0
0	1	0
1	0	0
1	1	1

OR GATE:-

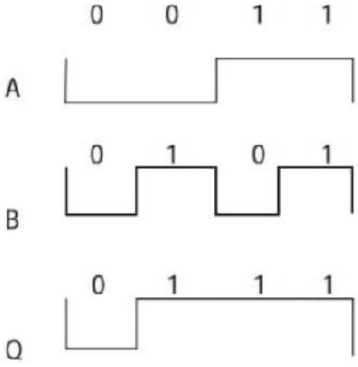
- An OR gate may have two or more inputs but only one output.
- The output is logic 1 state, even if one of its input is in logic 1 state.
- The output is logic 0 state, only when each one of its inputs is in logic state.

IC No.:- 7432



Logic Symbol
Truth Table

Timing Diagram



Truth table:

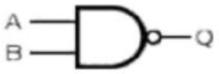
INPUT		OUTPUT
A	B	Q=A + B
0	0	0
0	1	1
1	0	1
1	1	1

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, theoutput is logic 1.

IC No.:- 7400 two input NAND
gate 7410 three input
NAND gate7420 four
input NAND gate 7430
eight input NAND gate

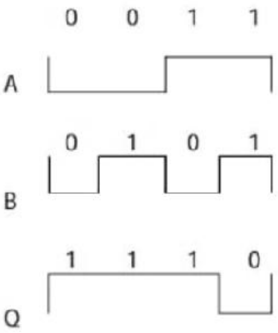
Logic Symbol



Truth Table

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

Timing Diagram



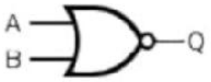
NOR GATE:-

- 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 gate7425 four
input NOR gate

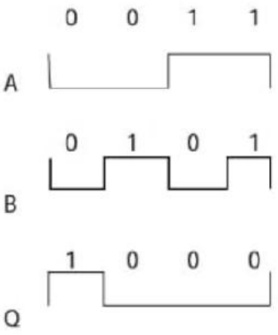
Logic Symbol



Truth Table

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

Timing Diagram

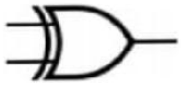


EXCLUSIVE – OR (X-OR) GATE:-

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

IC No.:- 7486

Logic Symbol



Truth Table

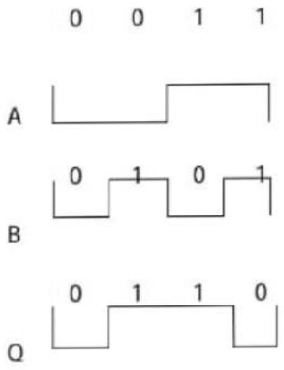
INPUT		OUTPUT
A	B	Q = A⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

INPUTS are A and B

OUTPUT is $Q = A \oplus B$

$= A'B + AB'$

Timing Diagram

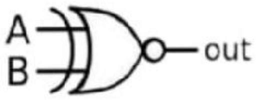


EXCLUSIVE – NOR (X-NOR) GATE:-

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

IC No.:- 74266

Logic Symbol

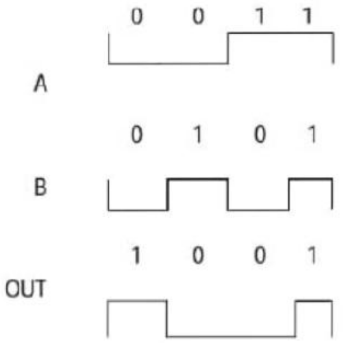


INPUT		OUTPUT
A	B	OUT =A XNOR B
0	0	1
0	1	0
1	0	0
1	1	1

OUT =A B + A′B′

= A XNOR B

Timing Diagram

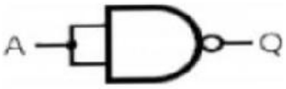


UNIVERSAL GATES:-

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:-

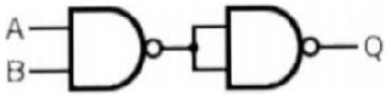
a) Inverter from NAND gate



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

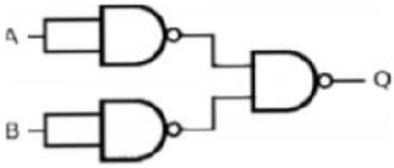
b) AND gate from NAND gate

Input s are A and BOutput Q = A.B



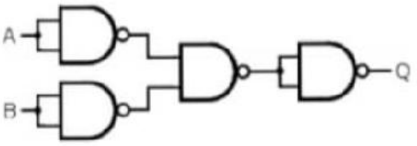
c) OR gate from NAND gate

Inputs are A and BOutput Q = A+B



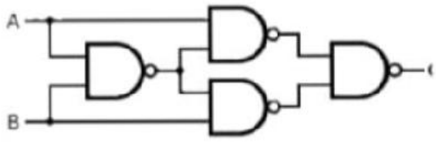
d) NOR gate from NAND gate

Inputs are A and BOutput Q = A+B



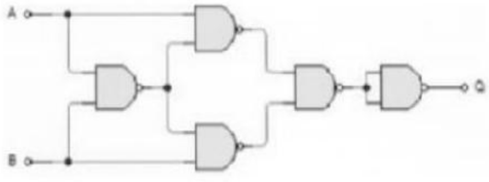
e) EX-OR gate from NAND gate

Inputs are A ard_B Output Q = A′ B +AB′



f) EX-NOR gate From NAND gate

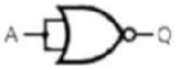
Inputs are A and B
Output Q = A B + A′ B′



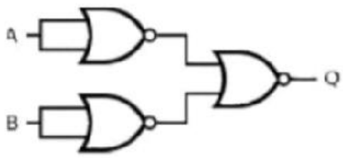
NOR GATE:-

a) Inverter from NOR gate

Input = A
Output Q = A′

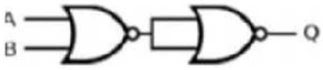


b) AND gate from NOR gateInput s are A and B Output Q = A.B



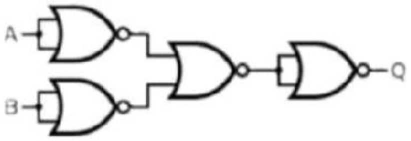
a) OR gate from NOR gate

Inputs are A and BOutput Q = A+B



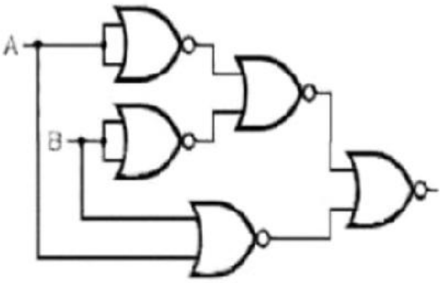
b) NAND gate from NOR gate

Inputs are A and BOutput Q = A.B



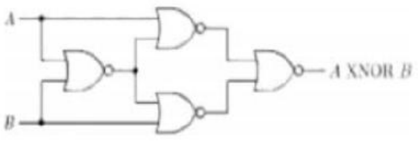
c) EX-OR gate from NOR gate

Inputs are A and_B Output Q = A′B + AB′



d) EX-NOR gate From NOR gate

Inputs are A and B Output Q = A B + A′ 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 ofelements (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 opcration

Axiom 1: 0 . 0 = 0

Axiom 2: 0 . 1 = 0

Axiom 3: 1 . 0 = 0

Axiom 2: 1 . 1 = 1

OR opcration

Axiom 5: 0 + 0 = 0

Axiom 6: 0 + 1 = 1

Axiom 7: 1 + 0 = 1

Axiom 8: 1 + 1 = 1

NOT opcration

Axiom 9: $\bar{1} = 0$

Axiom 10:0 = 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: $\underline{0} = 1$

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

Law 3: if A = 0, then A = 1

Law 4: if $\underline{A}$ = 1,thenA = 0

Law 5: A = 0 (double complementation law)

2. OR Laws:-

The four OR laws are as follows Law 1: A + 0 = 0

Law 2: A + 1 = 1

Law 3: A + A = A Law 4: A +A = 1

3. AND Laws:-

The four AND laws are as follows Law 1: A . 0 = 0

Law 2: A . 1 = 1

Law 3: A . A = A

Law 4: A $.\underline{A}$ = 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

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

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)

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)

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: $A\ (B + C) = AB + AC$

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

$= AA + AC + BA + BC$

$= A + AC + AB + BC$

$= A\ (1+ C + B) + BC$

$= A.\ 1 + BC$ $\qquad (1 +C + B = 1 + B = 1)$

$= A + BC$

$=$ LHS

7. Redundant Literal Rule

(RLR):-Law 1: $A + AB = A$

$+ B$

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

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

$=AA' + AB$
$= 0 + AB$
$= AB$

8. Idempotence Laws:- Idempotence means same value.

Law 1: A. A = A

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

If A = 0, then A + A = 0 + 0 = 0
= AIf A = 1, then A + A = 1 + 1
= 1 = A

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

9. Absorption Laws:-

There are two laws:

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

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

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

2: $A\,(A + B) = A$

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

Law

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

10. De Morgan's Theorem:-

Law 1: (A + B)′ =A′· B′

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 individualcomp~~le~~ments.

Law 2: (A· B)′ = A ′+ B′

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

=

A	B	A	B	A + 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 individualcomplements.

Types of canonical expression:

1. *Sum of Product (SOP)*
2. *Product of Sum (POS)*

SUM - OF - PRODUCTS FORM:-

- This is also called disjunctive Canonical Form (DCF) or Expanded Sum of Products Form or CanonicalSum of Products Form.
- In this form, the function is the sum of a number of products terms where each product term contains allvariables 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 tothose combinations for which 'f ' assumes the value 1.

  For example

  $$\begin{aligned} f(A, B, C) &= AB + BC \\ &= AB\,(C + C') + BC\,(A + A') \\ &= A\,BC + ABC' + ABC + A'BC \end{aligned}$$

- The product term which contains all the variables of the functions either in complemented oruncomplemented 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 forwhich 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 min terms.

PRODUCT- OF - SUMS FORM:-

- This form is also called as Conjunctive Canonical Form (CCF) or Expanded Product - of – Sums Formor 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 $f\,(A, B, C) = (A + \underline{B} + C\cdot C'\underline{)} + (\underline{A} + \underline{B} + C\cdot C')$
-

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

  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.

1.SUM OF PRODUCT (SOP) FORM or MINTERM:

- The SOP expression usually takes the form of two or more variable ANDed together ORed with two or more other variable ANDed together.

Example➔AB′+AC+A′BC
➔AB+CD

STANDARD SOP FORM:

- A standard SOP expression is one in which all the variables is present in each product term in the expression.

- **Example ➔AB′CD+A′B′CD′+ABC′D′**

2.PRODUCT OF SUM (POS)FORM or MAXTERM:

- The POS expression generally takes the form of two or more ORed .Variable with in parentheses ANDed with two or more other variable within parentheses.

Example➔(A+B).(C+D)
➔(X+Y′)(Y+Z)
➔(Y+Z′+X′)(XY+Z)

STANDARD POS FORM:

- A standard POS expression is one in which all the variables is present in each sum term in the expression.

- **Example➔**(A′+B′+C′+D′)(A+B′+C+D)(A+B+C′+D)

GENERAL SOP TO STANDARD SOP:

Example 1:
AB+BC

Solution-AB.1+BC.1
=AB(C+C′)+BC(A+A′)
=ABC+ABC′+ABC+A′BC
=ABC+ABC′+A′BC **[ABC+ABC=ABC Using OR rules A+A=A]**

Example 2:
ABC′+AB+C
Solution:
=ABC′+AB(C+C′)+C(A+A′)(B+B′)
=ABC′+ABC+ABC′+C(AB+AB′+A′B+A′B′)
=ABC′+ABC+ABC′+ABC+AB′C+A′BC+A′B′C
=ABC+ABC′+AB′C+A′BC+A′B′C

GENERAL POS TO STANDARD POS:-

Example 1:
(A+B+C)(A+B)
=(A+B+C)(A+B+CC′)
=(A+B+C).(A+B+C)(A+B+C′)
=(A+B+C).(A+B+C′)

Example 2:
(A′+B+C).A
=(A′+B+C)(A+BB′+CC′)
=(A′+B+C)(A+B+C)(A+B+C′)(A+B′+C)(A+B′+C′)

RULES FOR STANDARD SOP TO STANDARD POS:

- Consider each variables as 1.
- Write the possible combinations.
- Write the left combinations.
- In the left combinations consider each variables as zero &write the sum terms.
- The product of the sum terms is the standard POS.

Example 1→ABC+A′BC+A′B′C+A′B′C′

- Consider each variables as 1 i.e. A=B=C=1
- Possible combinations i.e.111+011+001+000
- Left combinations are

010→A+B′+C
100 →A′+B+C
101→A′+B+C′
110→A′+B′+C
POS →(A+B′+C)(A′+B+C)(A′+B+C′)(A′+B′+C)→Standard POS

Example 2→A′B′C′+A′B′C+A′BC+ABC′+AB′C′

- Consider each variables as 1 i.e. A=B=C=1
- Possible combinations i.e.000+001+011+110+100
- Left combinations are

010→A+B′+C
101→A′+B+C′
111→A′+B′+C′
POS→ (A+B′+C) (A′+B+C′) (A′+B′+C′)→Standard POS

RULES FOR STANDARD POS TO STANDARD SOP:-

- Consider each variables as 0.
- Write the possible combinations.
- Write the left combination.
- In the left combinations consider each variable as 1and write the product terms.
- The sum of the product term is the standard SOP.

Example 1→ (A′+B+C)(A+B+C)(A+B+C′)(A+B′+C)(A+B′+C′)

- Consider each variable as 0 i.e.=B=C=0
- Possible combinations i.e. (1+0+0)(0+0+0)(0+0+1)(0+1+0)(0+1+1)
- Left combinations are

1 0 1→AB′C
1 1 0→ABC′
1 1 1→ABC
SOP→(AB′C)+(ABC′)+(ABC)→Standards SOP

Example 2→ (A+B+C′)(A′+B′+C′)(A′+B′+C):

- Consider each variable as 0 i.e. A=B=C=0
- Possible combinations i.e.(0+0+1)(1+1+1)(1+1+0)
- Left combinations are

0 0 0→A′B′C′
0 1 0→A′BC′
0 1 1→A′BC
1 0 0→AB′C′
1 0 1→AB′C
SOP=(A′B′C)+(A′BC′)+(A′BC)+(AB′C′)+(AB′C)→Standards SOP.

Q.1. Find the canonical SOP (minterm) for the following function.
Y (A.B) =A+B

Solution:
Y (A.B) =A+B
=A.1+B.1
=A (B+B′) +B (A+A′)

=AB+AB′+AB+A′B
=AB+AB′+A′B

Q.2. Y=A+B′C express the function in canonical SOP & canonical POS.

Solution:

Y=A+B′C

=A(B+B′)(C+C′)+B′C(A+A′)

=A(BC+BC′+B′C+B′C′)+AB′C+A′B′C

=ABC+ABC′+AB′C+AB′C′+AB′C′+A′B′C

=ABC+ABC′+AB′C+AB′C′+A′B′C

Y=m_7+m_6+m_5+m_4+m_1

Therefore Y=∑m (1, 4, 5, 6, 7)

- canonical pos (maxterm)

Y=A+B′C

=(A+B′)(A+C)

=(A+B′+CC′)(A+C+BB′)

=(A+B′+C)(A+B′+C′)(A+B+C)(A+B′+C)

=(A+B+C)(A+B′+C)(A+B′+C′) **[(A+B′+C)+ (A+B′+C)= (A+B′+C)]**

Y=$m_0m_2m_3$

Therefore Y=Πm (0, 2, 3)

<u>3.</u> Expand the function A+BC′+ABD′+ABCD to minterm & maxterm.

Solution:-

Y=A+BC′+ABD′+ABCD

=A(B+B′)(C+C′)(D+D′)+BC(A+A′)(D+D′)+ABD′(C+C′)+ABCD

=A[(BC+BC′+B′C+B′C′)(D+D′)]+BC′(AD+AD′+A′D+A′D′)+ABCD′+ABC′D′+ABCD

=A[BCD+BC′D+B′CD+B′C′D+BCD′+BC′D′+B′CD′+B′C′D′]+ABC′D+ABC′D′+A′BC′D+A′BC′D′+ABCD′+ABC′D′+ABCD

=ABCD+ABC′D+AB′CD+AB′C′D+ABCD′+ABC′D′+AB′CD′+AB′C′D′+A′BC′D+A′BC′D′

SOP=∑m(4,5,8,9,10,11,12,13,14,15)

POS=Πm(0,1,2,3,6,7)

Q.4.Expand A(B′+A)B to maxterm & minterm

Solution:

<u>In SOP (maxterm):</u>

Y=A(B′+A)B

=(A+0)(B′+A)(B+0)

=(A+BB′)(B′+A)(B+AA′)

=(A+B)(A+B′)(A+B′)(A+B)(A′+B)

=(A+B)(A+B′)(A′+B′)

Y=Πm(0,1,2) → maxterm(POS)

- The maxterm **m3** is missing in the POS form, so the SOP form will contain only the minterm **m3**.

<u>In SOP (minterm):</u>

Solution:

Y=A(B′+A)B

=ABB′+AAB

=0+AB **[AND rules BB′=0 & AA=A]**

=AB

Y=∑m(3) → minterm(SOP)

Q.5. Expand the following expression to min terms & max terms.

x′ + x(x + y′) (y + z′)

Solution:

F= X′ + X (X + Y′)(Y + Z′)

= X′ + (XX +X Y′)(Y + Z′) [DISTRIBUTIVE LAW]

=X′ + (X +X Y′)(Y + Z′) [AND RULES XX=X]

= X′ + X(1 + Y′)(Y + Z′) [OR RULES 1+Y′=1]

=X′+X(Y+Z′)

= X′ + XY + XZ

= X′ (Y+Y′) (Z+Z′) +XY (Z+Z′) +XZ′ (Y+Y′)

=X′ (YZ+YZ′+Y′Z+Y′Z′) +XYZ+XYZ′+XYZ′+XY′Z′

=X′YZ+X′YZ′+X′Y′Z+X′Y′Z′+XYZ+XYZ′+XYZ′+XY′Z′

=X′YZ+X′YZ′+X′Y′Z+X′Y′Z′+XYZ+XYZ′+XY′Z′

Y= ∑M(0,1,2,3,4,6,7)→MINTERM(SOP)

Y=(5)→MAXTERM(POS)

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.

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

Example:-

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

ABSolution:-

The expression

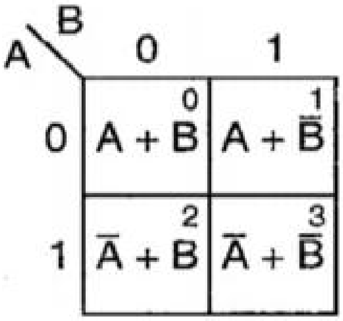
minterms isF = $m_1 + m_2$

= m(1, 2)

		B	
		0	1
A	0	0 ^{0}	1 ^{1}
	1	1 ^{2}	0 ^{3}

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 A + B . They are represented as M_0, M_1, M_2 and M_3 respectively.



A \ B	0	1
0	0 $A + B$	1 $A + \bar{B}$
1	2 $\bar{A} + B$	3 $\bar{A} + \bar{B}$

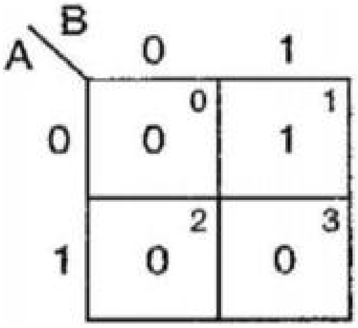
The maxterm of a two variable K-map

Example:-

Plot the expression f= (A + B)(A′ + B)(A′+ B′)

Solution:-

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

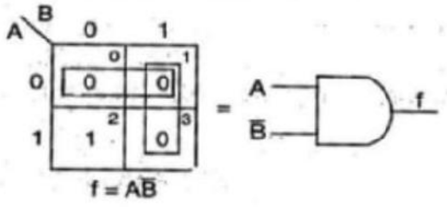


A \ B	0	1
0	0 0	1 1
1	2 0	3 0

Example:-

Reduce the expression f = (A + B) (A + B')(A' +B ') using mappingSolution:-

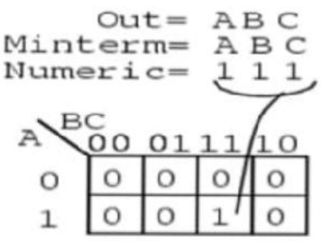
The given expression in terms of maxterms is f = πM (0, 1, 3)



THREE VARIABLE K- MAP:-

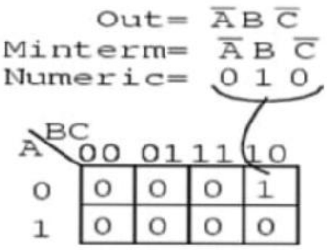
Example 2

- The number of cells in 3 variable K-map is eight, since the number of variables is three.
- The following figure shows **3 variable K-Map**.
- **Examples 1:**



Out= A B C
Minterm= A B C
Numeric= 1 1 1

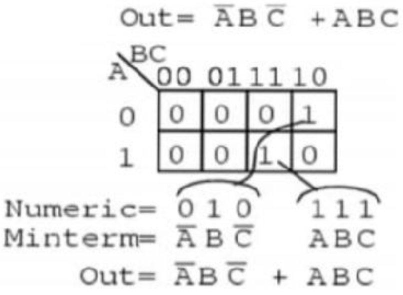
A \ BC	00	01	11	10
0	0	0	0	0
1	0	0	1	0



Out= $\bar{A}B\bar{C}$
Minterm= $\bar{A}B\bar{C}$
Numeric= 0 1 0

A \ BC	00	01	11	10
0	0	0	0	1
1	0	0	0	0

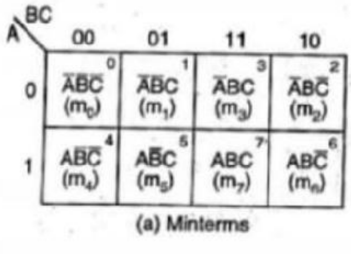
- Out= A B C　　　　Out= $\bar{A}B\bar{C}$



Out= $\bar{A}B\bar{C}$ +ABC

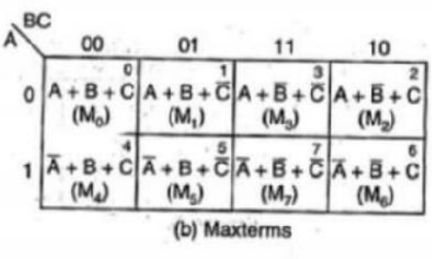
A \ BC	00	01	11	10
0	0	0	0	1
1	0	0	1	0

Numeric= 0 1 0　　1 1 1
Minterm= $\bar{A}B\bar{C}$　　ABC
Out= $\bar{A}B\bar{C}$ + ABC



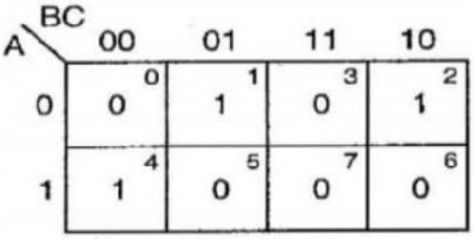
A \ BC	00	01	11	10
0	$\bar{A}\bar{B}\bar{C}$ (m_0) 0	$\bar{A}\bar{B}C$ (m_1) 1	$\bar{A}BC$ (m_3) 3	$\bar{A}B\bar{C}$ (m_2) 2
1	$A\bar{B}\bar{C}$ (m_4) 4	$A\bar{B}C$ (m_5) 5	ABC (m_7) 7	$AB\bar{C}$ (m_6) 6

(a) Minterms



A \ BC	00	01	11	10
0	$A+B+C$ (M_0) 0	$A+B+\bar{C}$ (M_1) 1	$A+\bar{B}+\bar{C}$ (M_3) 3	$A+\bar{B}+C$ (M_2) 2
1	$\bar{A}+B+C$ (M_4) 4	$\bar{A}+B+\bar{C}$ (M_5) 5	$\bar{A}+\bar{B}+\bar{C}$ (M_7) 7	$\bar{A}+\bar{B}+C$ (M_6) 6

(b) Maxterms

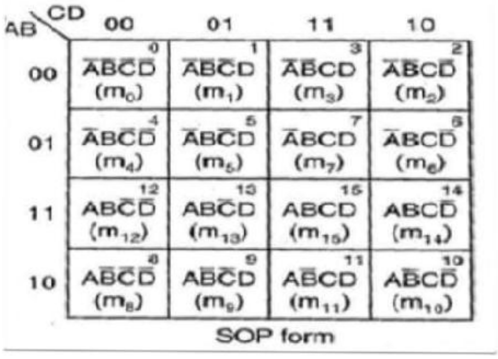


A \ BC	00	01	11	10
0	0 (0)	1 (1)	0 (3)	1 (2)
1	1 (4)	0 (5)	0 (7)	0 (6)

FOUR VARIABLE K-MAP:-

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

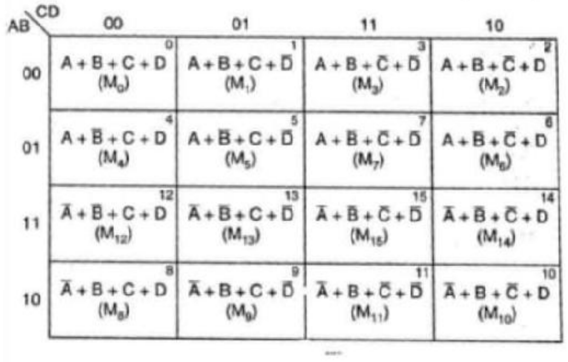
SOP FORM



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

SOP form

POS FORM



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

Minimization of SOP and POS Expressions:-

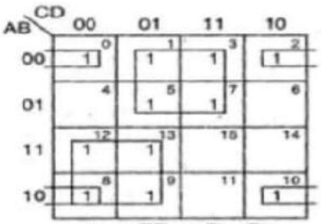
For reducing the Boolean expressions in SOP (POS) form the following steps are given below

- Draw the K-map and place 1s (0s) corresponding to the minterms (maxterms) of the SOP (POS)expression.
- In the map 1s (0s) which are not adjacent to any other 1(0) are the isolated minterms (maxterms). Theyare 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:-

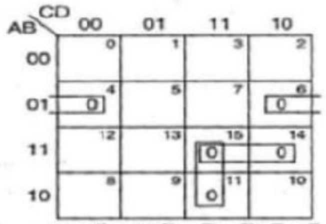
Reduce using mapping the expression f = ∑ m (0, 1, 2, 3, 5, 7, 8, 9, 10, 12, 13)Solution:-

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)



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

(a) SOP K-map



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

(b) POS K-map

The minimal SOP expression is f_{min}= BD + AC + AD

– – –

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 oroptional 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.

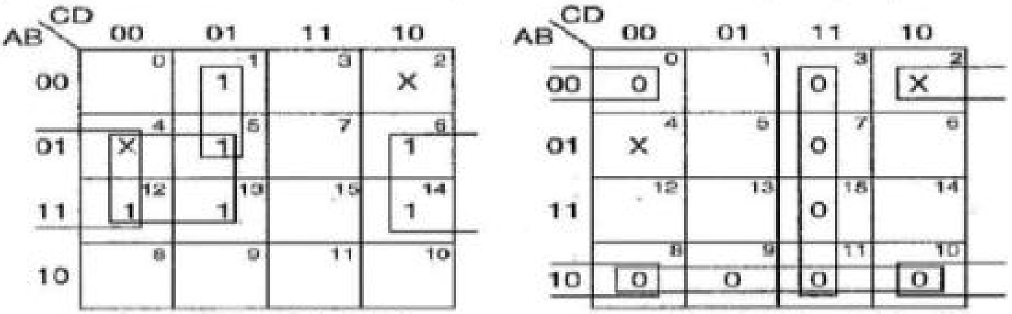
Example:-

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)



$f_{min} = B\overline{C} + B\overline{D} + \overline{A}CD$
(a) SOP K-map

$f_{min} = (B + D)(\overline{A} + B)(\overline{C} + \overline{D})$
(b) POS K-map

The minimal of SOP expression is f_{min} = BC + BD +ACD

– –

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

LECTURE NOTES

ON

DIGITAL ELECTRONICS & MICROPROCESSOR

5TH SEMESTER

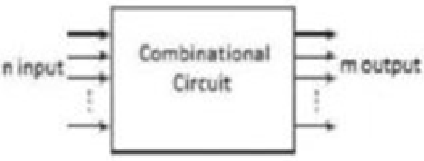
DEPARTMENT OF ELECTRICAL ENGINEERING

Prepared by :
MR. ADITYA NARAYAN JENA
Lecturer in Electronics & Telecommunication Engineering.

PNS SCHOOL OF ENGINEERING & TECHNOLOGY
Nishamani Vihar, Marshaghai, Kendrapara

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.

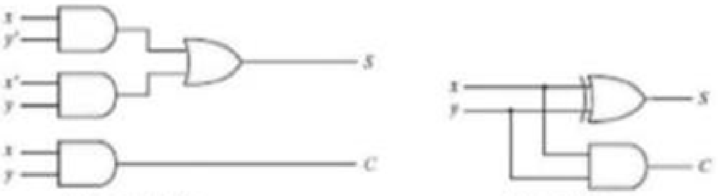
x	y	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Truth Table

- The simplified sum-of-products expressions are

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

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



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

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

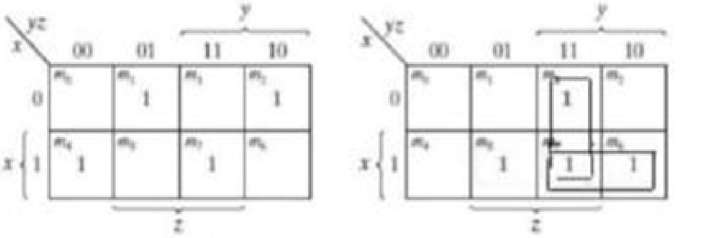
FULL ADDER:-

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

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

Truth Table

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



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

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

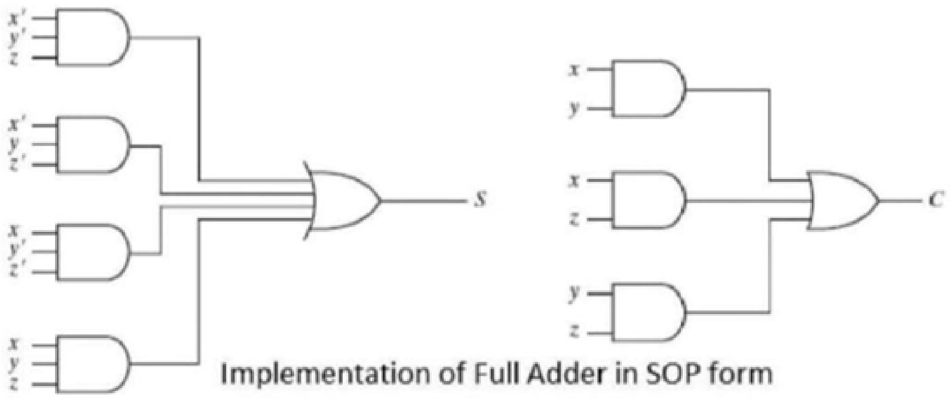
K-Map for full adder

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

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

$$C = xy + xz + yz$$

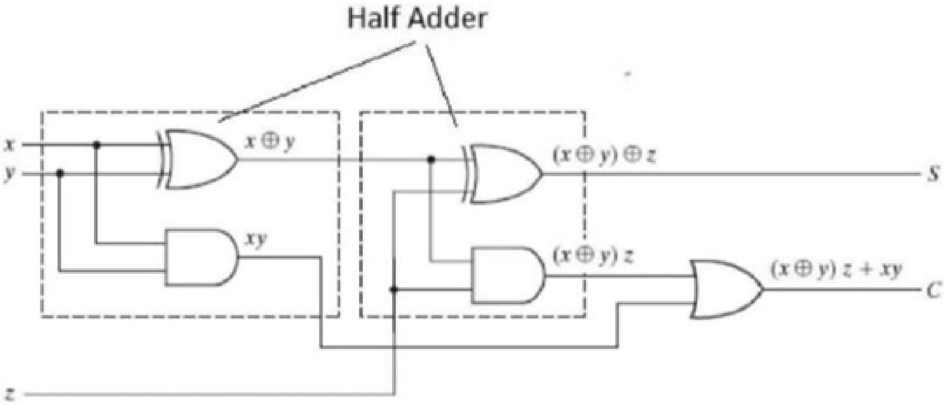
- The logic diagram for the full adder implemented in sum-of-products form is shown in figure.



Implementation of Full Adder in SOP form

Full adder using half adder

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



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

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

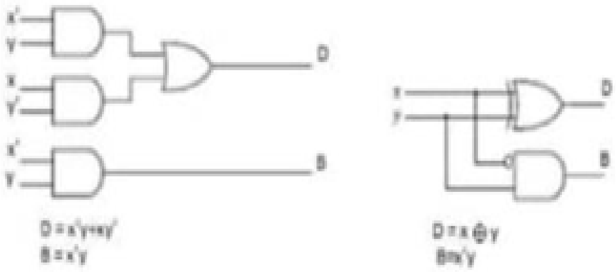
Truth Table

- 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

$$D = x'y + xy' \text{ and } B = x'y$$



- 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:-

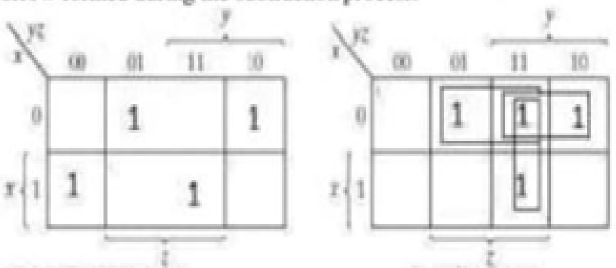
- A full subtractor is a combinational circuit that forms the arithmetic subtraction operation of three bits.
- It consists of three inputs and two outputs. Two of the input variables, denoted by x and y , represent the two significant bits to be subtracted. The third input, z , is subtracted from the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



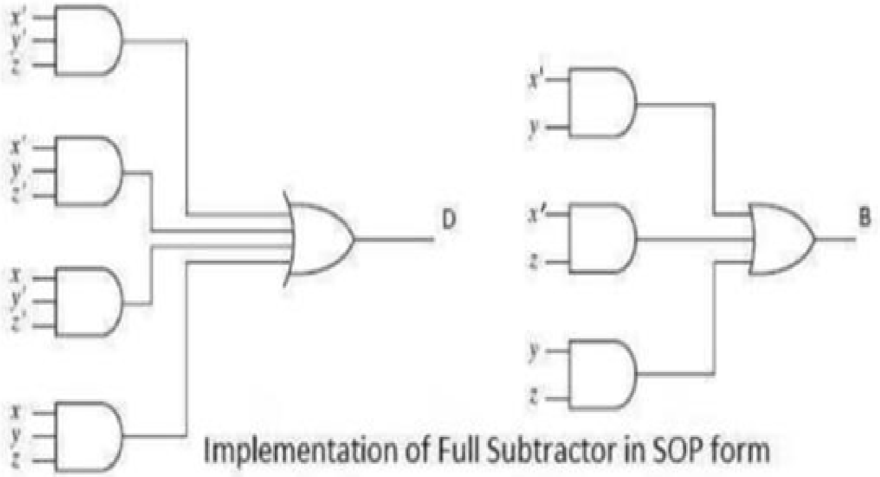
D = x'y'z+x'yz'+xy'z'+xyz

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

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



Implementation of Full Subtractor in SOP form

MAGNITUDE COMPARATOR:-

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

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

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

$x1=A_1B_1+A_1'B_1'$

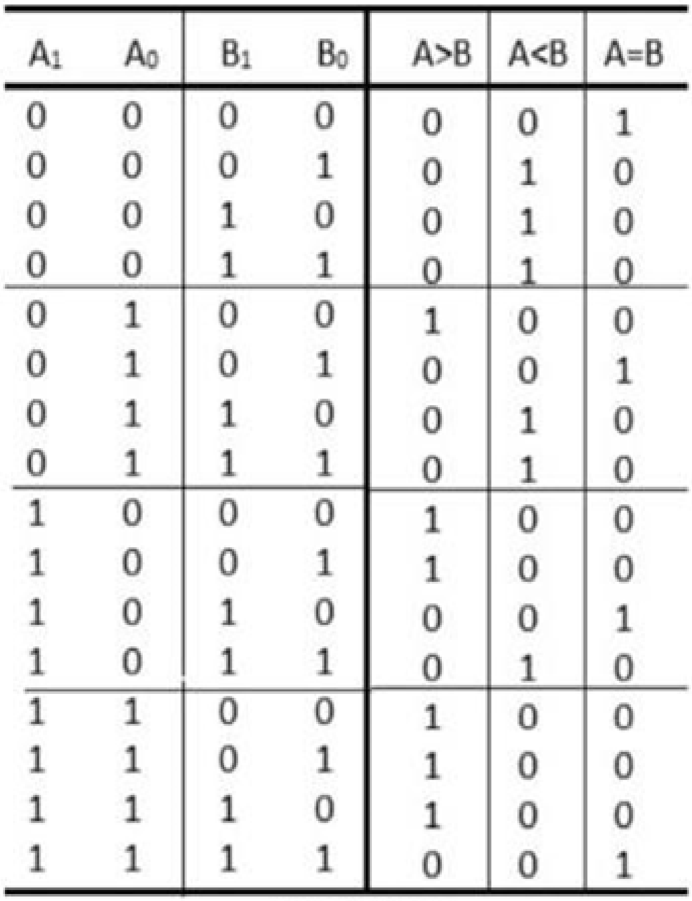
And $x0=A_0B_0+A_0'B_0'$

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

$$(A = B) = x_1x_0$$

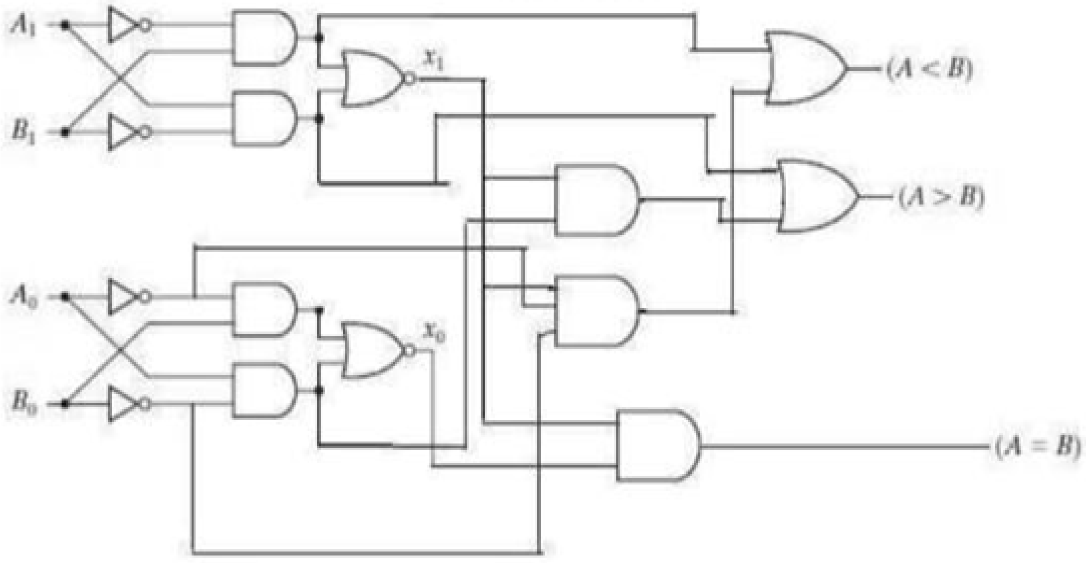
- The binary variable (A = B) is equal to 1 only if all pairs of digits of the two numbers are equal.
- To determine whether A is greater or less than B, we inspect the relative magnitudes of pairs of significant digits, starting from the most significant position. If the two digits of a pair are equal,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

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



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

Truth Table

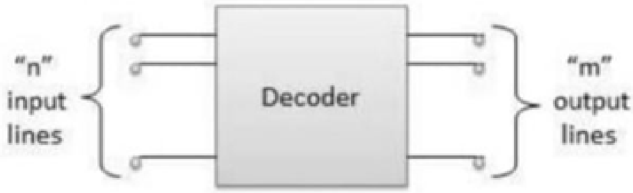


Logic Diagram of 2-bit Magnitude Comparator

Decoder

A decoder is a combinational circuit. It has n input and to a maximum m = 2n outputs. Decoder is identical to a demultiplexer without any data input. It performs operations which are exactly opposite to those of an encoder.

Block diagram



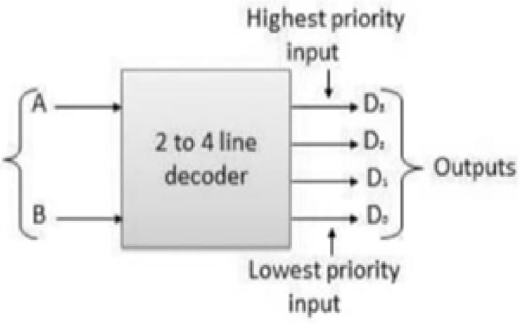
Examples of Decoders are following.

- Code converters
- BCD to seven segment decoders
- Nixie tube decoders
- Relay actuator

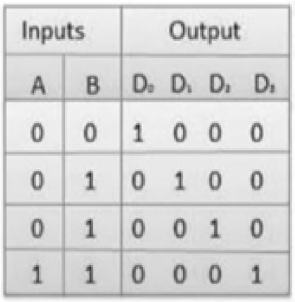
2 to 4 Line Decoder

The block diagram of 2 to 4 line decoder is shown in the fig. A and B are the two inputs where D through D are the four outputs. Truth table explains the operations of a decoder. It shows that each output is 1 for only a specific combination of inputs.

Block diagram

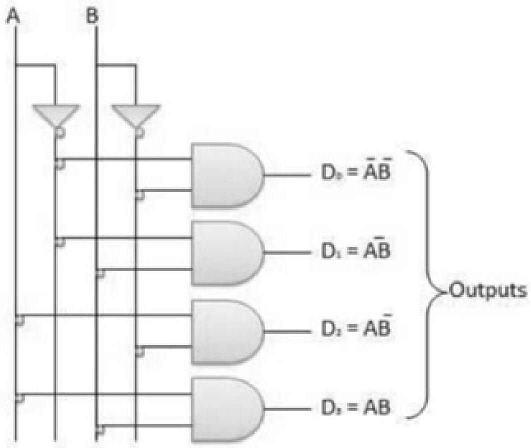


Truth Table



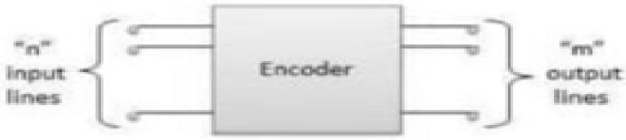
Inputs		Output			
A	B	D_0	D_1	D_2	D_3
0	0	1	0	0	0
0	1	0	1	0	0
0	1	0	0	1	0
1	1	0	0	0	1

Logic Circuit



Encoder

Encoder is a combinational circuit which is designed to perform the inverse operation of the decoder. An encoder has n number of input lines and m number of output lines. An encoder produces an m bit binary code corresponding to the digital input number. The encoder accepts an n input digital word and converts it into an m bit another digital word. Block diagram



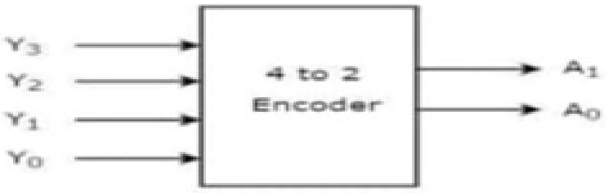
Examples of Encoders are following.

- Priority encoders
- Decimal to BCD encoder
- Octal to binary encoder
- Hexadecimal to binary encoder

4 to 2 Encoder

Let 4 to 2 Encoder has four inputs Y_3, Y_2, Y_1& Y_0 and two outputs A_1& A_0.

The **block diagram** of 4 to 2 Encoder is shown in the following figure.



At any time, only one of these 4 inputs can be '1' in order to get the respective binary code at the output. The **Truth table** of 4 to 2 encoder is shown below.

Inputs				Outputs	
Y_3	Y_2	Y_1	Y_0	A_1	A_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

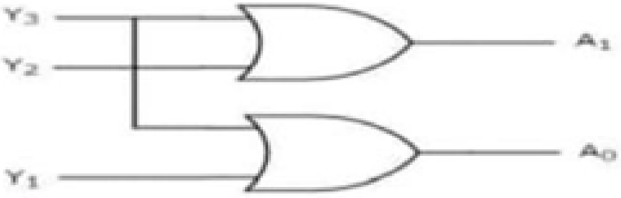
From Truth table, we can write the **Boolean functions** for each output as

$$A1 = Y3 + Y2$$

$$A0 = Y3 + Y1$$

We can implement the above two Boolean functions by using two input OR gates.

The **circuit diagram** of 4 to 2 encoder is shown in the following figure.

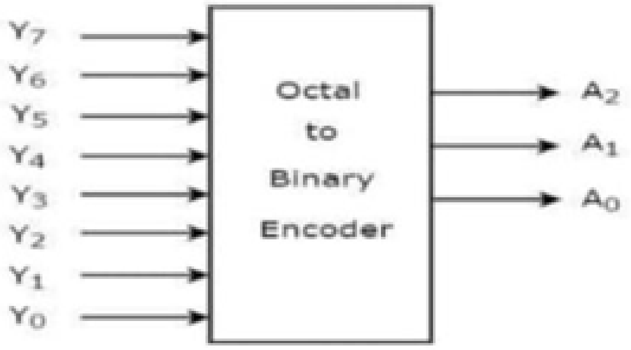


The above circuit diagram contains two OR gates. These OR gates encode the four inputs with two bits

Octal to Binary Encoder

Octal to binary Encoder has eight inputs, Y_7 to Y_0 and three outputs A_2, A_1& A_0. Octal to binary encoder is nothing but 8 to 3 encoder.

The **block diagram** of octal to binary Encoder is shown in the following figure.



At any time, only one of these eight inputs can be '1' in order to get the respective binary code. The **Truth table** of octal to binary encoder is shown below.

Inputs								Outputs		
Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0	A_2	A_1	A_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

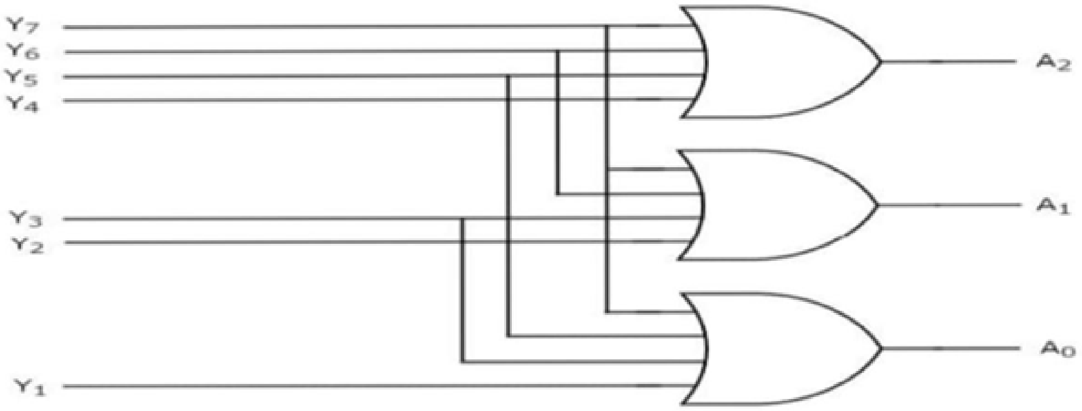
From Truth table, we can write the **Boolean functions** for each output as

$$A2 = Y7 + Y6 + Y5 + Y4$$

$$A1 = Y7 + Y6 + Y3 + Y2$$

$$A0 = Y7 + Y5 + Y3 + Y1$$

We can implement the above Boolean functions by using four input OR gates.



The above circuit diagram contains three 4-input OR gates. These OR gates encode the eight inputs with three bits.

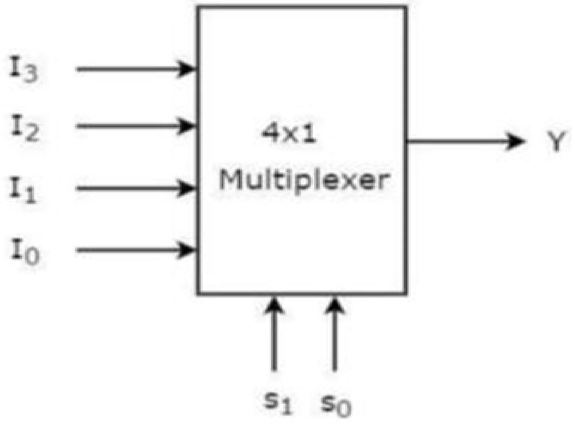
Multiplexer:-

Multiplexer is a combinational circuit that has maximum of 2^n data inputs, „n" selection lines and single output line. One of these data inputs will be connected to the output based on the values of selection lines.

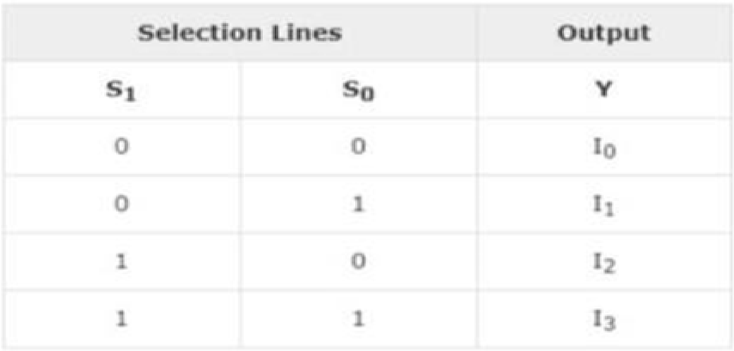
Since there are „n" selection lines, there will be 2^n possible combinations of zeros and ones. So, each combination will select only one data input. Multiplexer is also called as **Mux**.

4x1 Multiplexer

4x1 Multiplexer has four data inputs I_3, I_2, I_1 & I_0, two selection lines s_1 & $s_0$and one output Y. The **block diagram** of 4x1 Multiplexer is shown in the following figure.



One of these 4 inputs will be connected to the output based on the combination of inputs present at these two selection lines. **Truth table** of 4x1 Multiplexer is shown below.

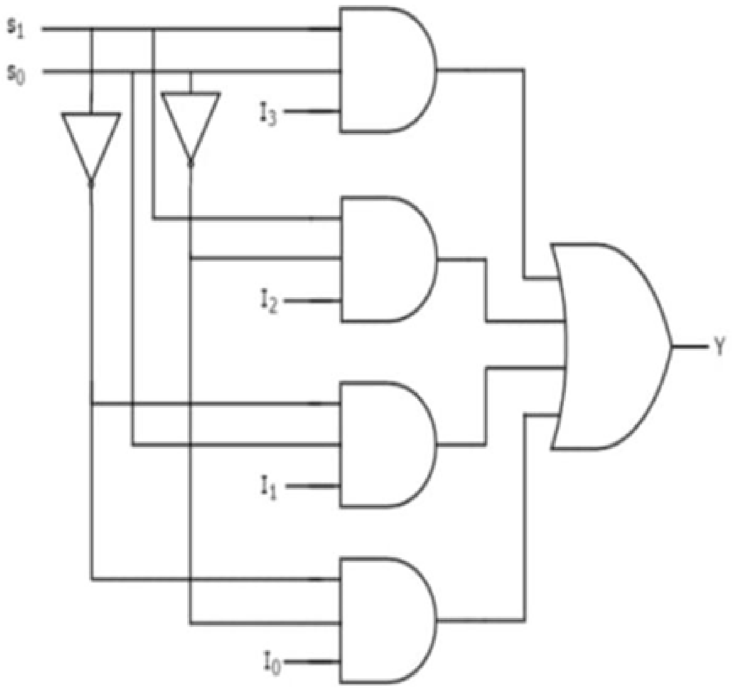


Selection Lines		Output
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

From Truth table, we can directly write the **Boolean function** for output, Y as

$$Y=S1'S0'I0+S1'S0I1+S1S0'I2+S1S0I3Y=S1'S0'I0+S1'S0I1+S1S0'I2+S1S0I3$$

We can implement this Boolean function using Inverters, AND gates & OR gate. The **circuit diagram** of 4x1 multiplexer is shown in the following figure.



Applications of Multiplexer:

Multiplexer are used in various fields where multiple data need to be transmitted using a single line. Following are some of the applications of multiplexers -

1. **Communication system** – Communication system is a set of system that enable communication like transmission system, relay and tributary station, and communication network. The efficiency of communication system can be increased considerably using multiplexer. Multiplexer allow the process of transmitting different type of data such as audio, video at the same time using a single transmission line.

2. **Telephone network –** In telephone network, multiple audio signals are integrated on a single line for transmission with the help of multiplexers. In this way, multiple audio signals can be isolated and eventually, the desire audio signals reach the intended recipients.

3. **Computer memory** - Multiplexers are used to implement huge amount of memory into the computer, at the same time reduces the number of copper lines required to connect the memory to other parts of the computer circuit.

4. **Transmission from the computer system of a satellite** – Multiplexer can be used for the transmission of data signals from the computer system of a satellite or spacecraft to the ground system using the GPS (Global Positioning System) satellites.

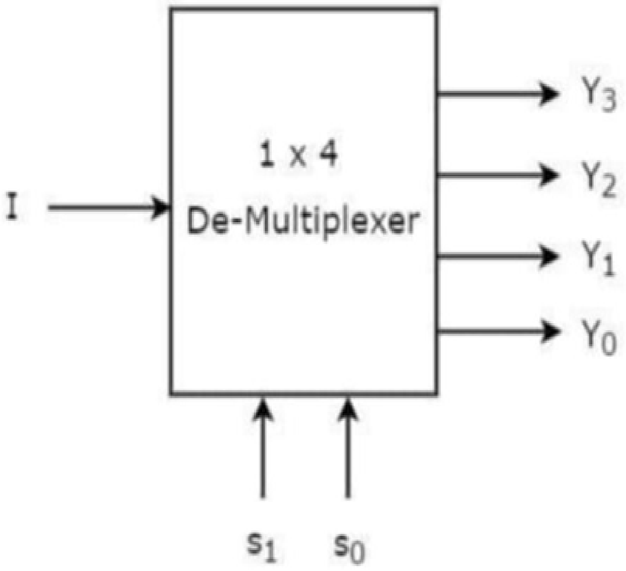
De-Multiplexer

De-Multiplexer is a combinational circuit that performs the reverse operation of Multiplexer. It has single input, „n" selection lines and maximum of 2^noutputs. The input will be connected to one of these outputs based on the values of selection lines.

Since there are „n" selection lines, there will be 2^n possible combinations of zeros and ones. So, each combination can select only one output. De-Multiplexer is also called as **De-Mux**.

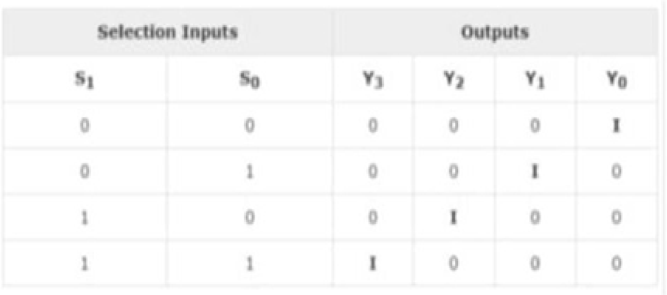
1x4 De-Multiplexer

1x4 De-Multiplexer has one input I, two selection lines, s_1 & s_0 and four outputs Y_3, Y_2, Y_1 &Y_0. The **block diagram** of 1x4 De-Multiplexer is shown in the following figure.



The single input „I" will be connected to one of the four outputs, Y_3 to Y_0 based on the values of selection lines s_1 & s0. The **Truth table** of 1x4 De-Multiplexer is shown below.

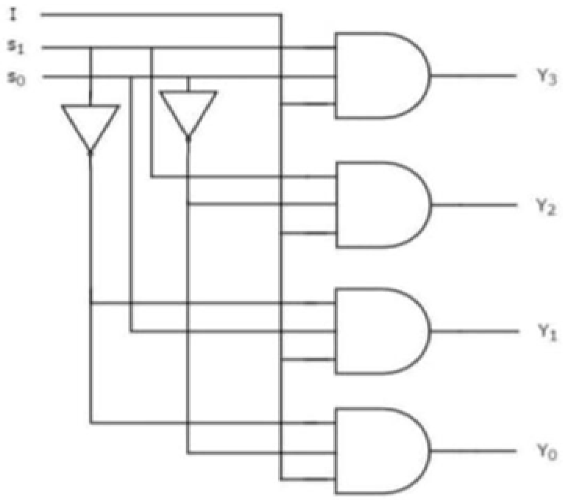
From the above Truth table, we can directly write the **Boolean functions** for each output as



Selection Inputs		Outputs			
S_1	S_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	I
0	1	0	0	I	0
1	0	0	I	0	0
1	1	I	0	0	0

Y3=s1s0I
Y2=s1s0I′
Y1=s1′s0I
Y0=s1′s0I′

We can implement these Boolean functions using Inverters & 3-input AND gates. The **circuit diagram** of 1x4 De-Multiplexer is shown in the following figure.



We can easily understand the operation of the above circuit. Similarly, you can implement 1x8 De-Multiplexer and 1x16 De-Multiplexer by following the same procedure.

Applications of Demultiplexer:

1. Demultiplexer is used to connect a single source to multiple destinations. The main application area of demultiplexer is communication system where multiplexer are used. Most of the communication system are bidirectional i.e. they function in both ways (transmitting and receiving signals). Hence, for most of the applications, the multiplexer and demultiplexer work in sync. Demultiplexer is also used for reconstruction of parallel data and ALU circuits.
2. **Communication System** - Communication system use multiplexer to carry multiple data like audio, video and other form of data using a single line for transmission. This process makes the transmission easier. The demultiplexer receives the output signals of the multiplexer and converts them back to the original form of the data at the receiving end. The multiplexer and demultiplexer work together to carry out the process of transmission and reception of data in communication system.
3. **ALU (Arithmetic Logic Unit)** – In an ALU circuit, the output of ALU can be stored in multiple registers or storage units with the help of demultiplexer. The output of ALU is fed as the data input to the demultiplexer. Each output of demultiplexer is connected to multiple register which can be stored in the registers.
4. **Serial to parallel converter** - A serial to parallel converter is used for reconstructing parallel data from incoming serial data stream. In this technique, serial data from the incoming serial data stream is given as data input to the demultiplexer at the regular intervals. A counter is attaching to the control input of the demultiplexer. This counter directs the data signal to the output of the demultiplexer where these data signals are stored. When all data signals have been stored, the output of the demultiplexer can be retrieved and read out in parallel.

LECTURE NOTES
ON
SEQUENTIAL
LOGIC CIRCUITS(Unit-3)

Prepared by

Er.Aditya Narayan Jena

Dept. of Electronics & Telecommunication Engg.

PNS SCHOOL OF ENGG. & TECH.

Nishamani Vihar,Marshaghai,Kendrapara

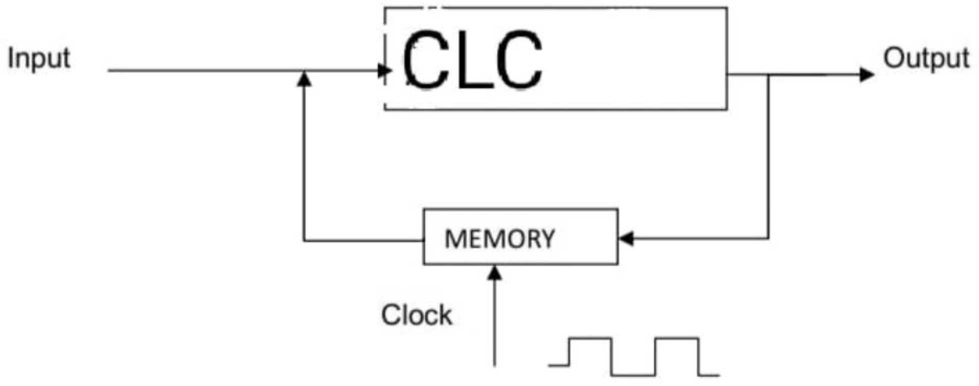
SEQUENTIAL LOGIC CIRCUIT

SEQUENTIAL CIRCUIT:-

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

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

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



[Block diagram of Sequential Logic Circuit]

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

TYPES:-

Sequential logic circuits (SLC) are classified as

(i) Synchronous SLC
(ii) Asynchronous SLC

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

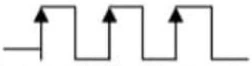
Difference between

combinational logic ckt	sequential logic ckt
1. Depends upon the present input only	1. Depends upon present input as well as past input
2. Combinational circuit is time-independent and it is very fast in operation	2. Sequential circuit is time-dependent and comparatively slower in operation
3. There is no feedback path between output and input	3. There is a feedback path available between output and input
4. The main elementary building block of a combinational circuit is basic logic gate	4. The elementary building block of a sequential circuit is the flip flop
5. The combinational circuits are mainly used for arithmetic and boolean operations	5. The sequential logic circuits are used for data storing
6. The operation of a combinational circuit is very simple	6. The operation of a sequential circuit is very complex
7. The combinational circuit does not need any external triggering so they are clock independent	7. The sequential circuits require external triggering means they are clock dependent

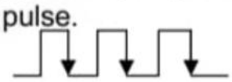
FLIP-FLOP AND LATCH:-

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

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



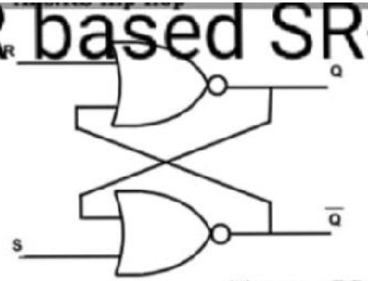
- Negative-edge triggered flip-flops are those in which state transition take place only at negative- going edge of the clock pulse.



Types of FFs:-

- 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

NOR based SR-FF:-



Inputs		Outputs	
R	S	Q	Status
0	0	Last State	No Change
1	0	0	Reset
0	1	1	Set
1	1	Forbidden	Race

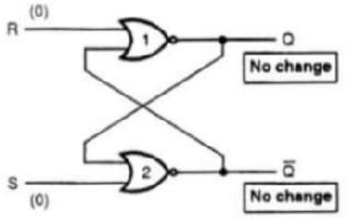
Figures : RS FlipFLop with NOR Gate.

In figure output of one NOR gates drives one of the input of the other NOR gate. The S and R inputs are used to set and reset the flip flop respectively.

Note : For the NOR gate , if any input of the NOR gate is '1' its output will be 0 irrespective of other inputs.

Operation of RS FlipFLop

Case 1: When R = 0 and S = 0 and Q = 0 , Q=1



When S = 0, R = 0 and Q=0 , Q=1 a '0' comes out from the upper NOR gate corresponding to Q = 0.

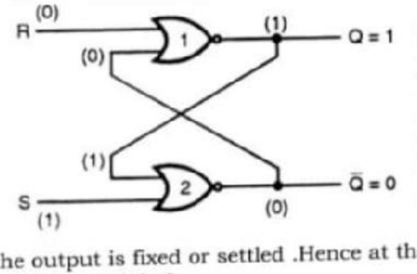
Now the lower NOR gate has both input '0' and hence a '1' comes out from the lower NOR gate corresponding to Q = 1.

Hence when R= 0 , S =0 Flip Flop remain in last state or No change State

Case 2: When R = 0 and S = 1 and Q = 0 , Q=1

When S = 1, R = 0 and Q=0 , Q=1 a '0' comes out from the upper NOR gate corresponding to Q = 0.

Now the lower NOR gate has one input '0' and other input as 1(Q=0 , S=1) ,hence a '0' comes out from the lower NOR gate corresponding to Q = 0.



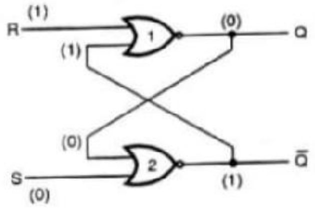
This Q = 0 is fed as input to upper NOR gate making R =0 and Q = 0 , this time a "1" come upper NOR gate.

Now the lower NOR gate has both input as 1(Q=1 , S=1) ,hence a '0' comes out from the lower NOR gate corresponding to Q = 0.

This process repeats till the output is fixed or settled .Hence at the end when R= 0 , S =0 Flip Flop out Q =1 and Q=0.

Hence when R= 0 , S =1 Flip Flop goes in Set state

Case 3: When R = 1 and S = 0 and Q = 1 , Q=0



When S = 0, R = 1 and Q=1, Q=0 a '0' comes out from the upper NOR gate corresponding to Q = 0.

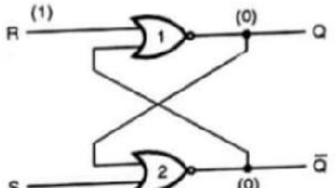
Now the lower NOR gate has both input as '0' (Q=0 , S=0) ,hence a '1' comes out from the lower NOR gate corresponding to Q = 1.

This Q = 1 is fed as input to upper NOR gate making R =1 and Q = 1 , this time a "0" come upper NOR gate.

Now the lower NOR gate has both input as 0(Q=0 , S=0) ,hence a '1' comes out from the lower NOR gate corresponding to Q = 1.

This process repeats till the output is fixed or settled .Hence at the end when R=1 , S =0 Flip Flop out Q =0 and Q=1.

Hence when R=1 , S =0 Flip Flop goes in Reset state

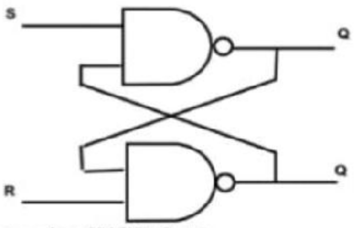


Case 4: When R = 1 and S = 0 and Q = 1 , Q=0

If S = 1 and R = 1 a '0' comes out of both NOR gates giving Q = Q = 1. This is condition is forbidden.

NAND Based SR-FF:-

Inputs		Outputs	
R	S	Q	Status
1	1	Last State	No Change
0	1	0	Reset
1	0	1	Set
0	0	Forbidden	Race

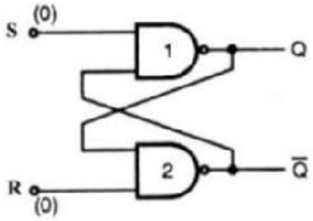


Truth Table **SR Flip-Flop using NAND Gate**

In figure output of one NAND gates drives one of the input of the other NAND gate. The S and R inputs are used to set and reset the flip flop respectively.

Note : For the NAND gate , if any input of the NAND gate is '0' its output will be 1 irrespective of other inputs.

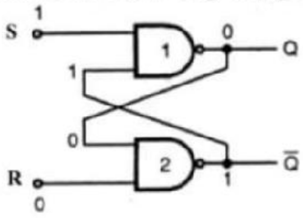
Case 1 : S=0 , R =0 Q =0 , Q=1 (Race Condition)



When S = 1, R = 1 and Q=0 , Q=1 a '1' comes out from the upper NAND gate corresponding to Q = 1.

Now the lower NAND gate has one input '0' and Other input as 1 and hence a '1' comes out from the lower NAND gate corresponding to Q = 1. Hence when R=0 , S =0 Flip Flop both outputs try to become one , this undefined or illegal or Forbidden state. This condition is called as RACE condition.

Case 1 : S=0 , R =1 Q =0 , Q=1 (SET Condition)

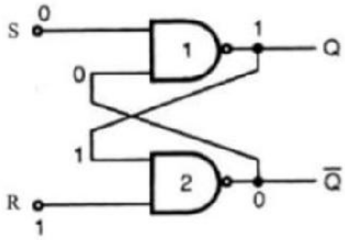


When S = 0, R = 1 and Q=0 , Q=1 a '1' comes out from the upper NAND gate corresponding to Q = 1.

Now the lower NAND gate has both input '1' ,hence a '0' comes out from the lower NAND gate corresponding to Q = 0.

This state remains as its , Hence when R=1 , S = 0 Flip Flop , output Q = 1 and Q = 0 , and the state is called as Set State

Case 2 : S=1 , R =0 Q =1 , Q=0 (RESET Condition)

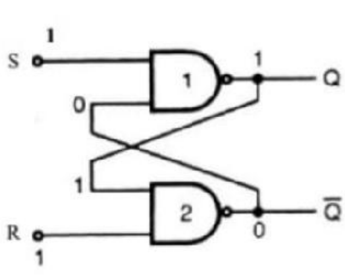


When S = 1, R = 0 and Q=1 , Q=0 a '1' comes out from the upper NAND gate corresponding to Q = 1.

Now the lower NAND gate has both input '1' ,hence a '0' comes out from the lower NAND gate corresponding to Q = 1.This 1 is fed as input to upper gate , now this time upper NAND gate both input as 1 , due to which output is 0.

Now this output is fed as input to lower NAND gate, whose both input are 0 , making output 1.This state remains as its , Hence when R=0 , S=1 Flip Flop , output Q = 0 and Q = 1, and the state is called as Reset State

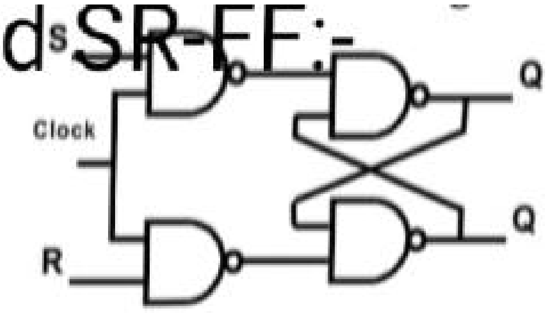
Case 4: When R = 1 and S = 1 and Q = 1 , Q = 0

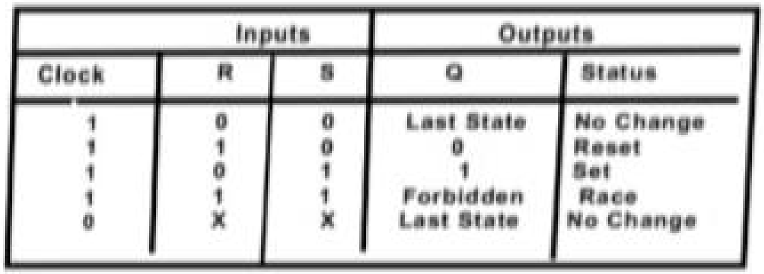


When S = 1, R = 1 and Q=1 , Q=0 a '1' comes out from the upper NAND gate corresponding to Q = 1.

Now the lower NAND gate has both input '1' and hence a '0' comes out from the lower NAND gate corresponding to Q = 0. Hence when R= 0 , S =0 Flip Flop remain in last state or No change State

Clocked SR-FF:-





Inputs			Outputs	
Clock	R	S	Q	Status
1	0	0	Last State	No Change
1	1	0	0	Reset
1	0	1	1	Set
1	1	1	Forbidden	Race
0	X	X	Last State	No Change

Figure Clocked RS Flip-Flop

It is often required to set or reset the memory cell in synchronism with a train of the pulse known as Clock. Such circuit is referred to as **clocked SR (set-reset flip-flop)**

The clock is a square wave signal because the clock drives both NAND and prevents S and R from controlling the latch.

Operation is as Follows

Case 1 : S =1 , R=0 (Set Condition)

- If S = 1 and R = 0 the output of gate A = 0 and B = 1. Now with clock = 1, S = 0, R = 1 and flip-flop set Q = 1 and Q= 0. I.e **Set Condition**

Case 2 : S =0 , R=1 (Reset Condition)

- If S = 0 and R = 1 the output of gate A = 1 and B = 0. Thus with clock = 1, S = 1, R = 0 and flip-flop set Q = 0 and Q = 1. I.e **Reset Condition**

Case 3 : S =1 , R=1 (Illegal/Forbidden Condition)

- With S = 1 and R = 1 the output of both gates will be 0 it is a forbidden condition state or a race condition. **I.e Illegal Condition or Forbidden State**

Case 4 : S =0 , R=0 (Last State Condition)

- When both S = 0, R = 0 and clock = 1 the output A & B gate = 1 which keep the flip-flop in last state. **I.e Last State**

D-flipFlop:-

- D flip flop is actually a slight modification of the above explained clocked SR flip-flop. From the figure you can see that the D input is connected to the S input and the complement of the D input is connected to the R input.

- The D input is passed on to the flip flop when the value of CP is '1'.

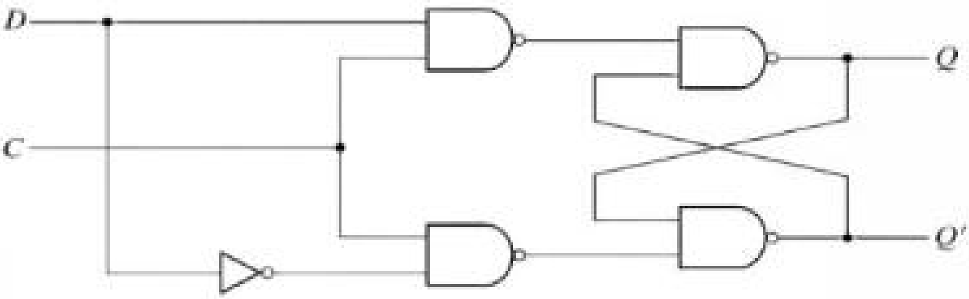
- When CP is HIGH, the flip flop moves to the SET state. If it is '0', the flip flop switches to the CLEAR state.

- As long as the clock input C = 0, the SR latch has both inputs equal to 0 and it can't change its state regardless of the value of D

- When C is 1, the latch is placed in the set or reset state based on the value of D.

  If D = 1, the Q output goes to 1.

  If D = 0, the Q output goes to 0.



(a) Logic diagram

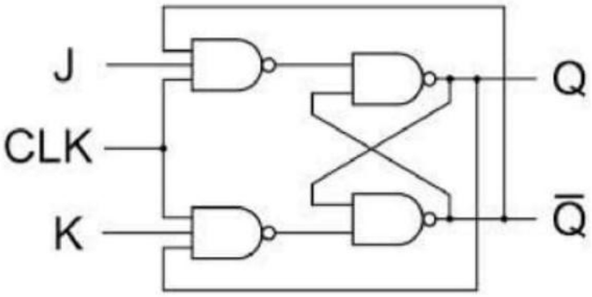
Truth Table:-

C	D	Q_{n+1}
0	0	NC
0	1	NC
1	0	0
1	1	1

JK FLIP-FLOP:-

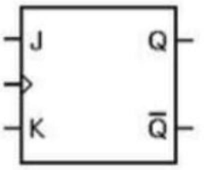
- The JK flip-flop can be constructed by using basic SR latch and a clock. In this case the outputs Q and Q' are returned back and connected to the inputs of NAND gates.
- This simple JK flip Flop is the most widely used of all the flip-flop designs and is considered to be a universal flip-flop circuit.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

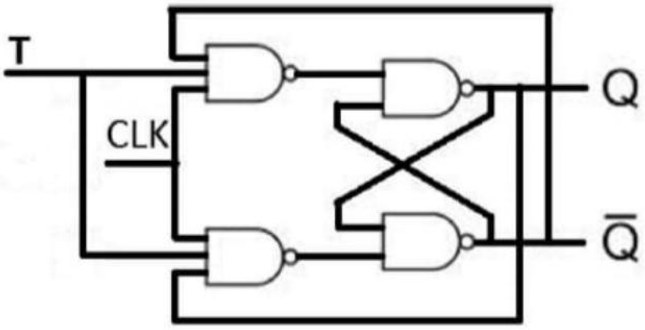
Truth Table of JK-FF:-

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.

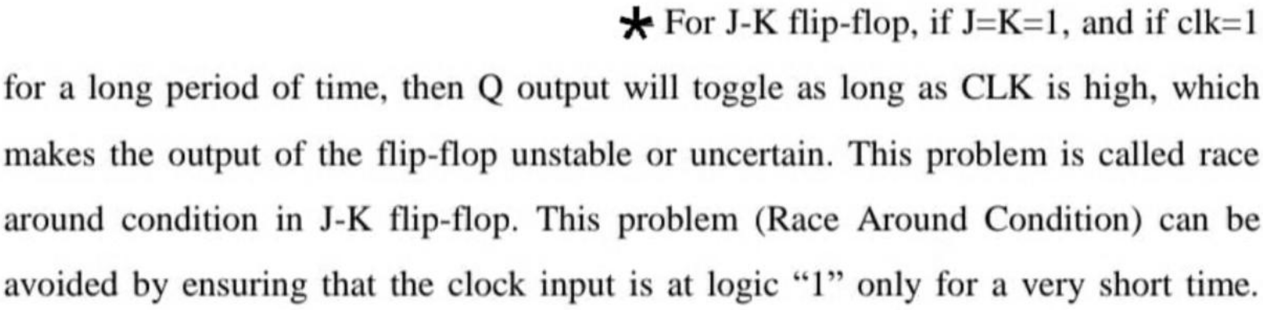


- Hence its truth table is same as that of JK flip-flop when J=K= 0 and J=K=1.So its truth table is as follows.

Truth Table:-

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

Race around condition in JK-FF:-



* For J-K flip-flop, if J=K=1, and if clk=1 for a long period of time, then Q output will toggle as long as CLK is high, which makes the output of the flip-flop unstable or uncertain. This problem is called race around condition in J-K flip-flop. This problem (Race Around Condition) can be avoided by ensuring that the clock input is at logic "1" only for a very short time.

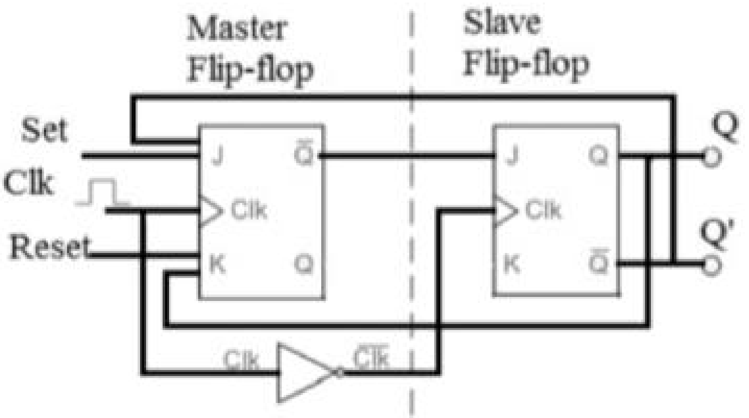
*It can be avoided by using Master-Slave JK-FF

*Also by using $tp<\Delta t$;where tp=width of clock pulse and Δt is the Propagation delay.

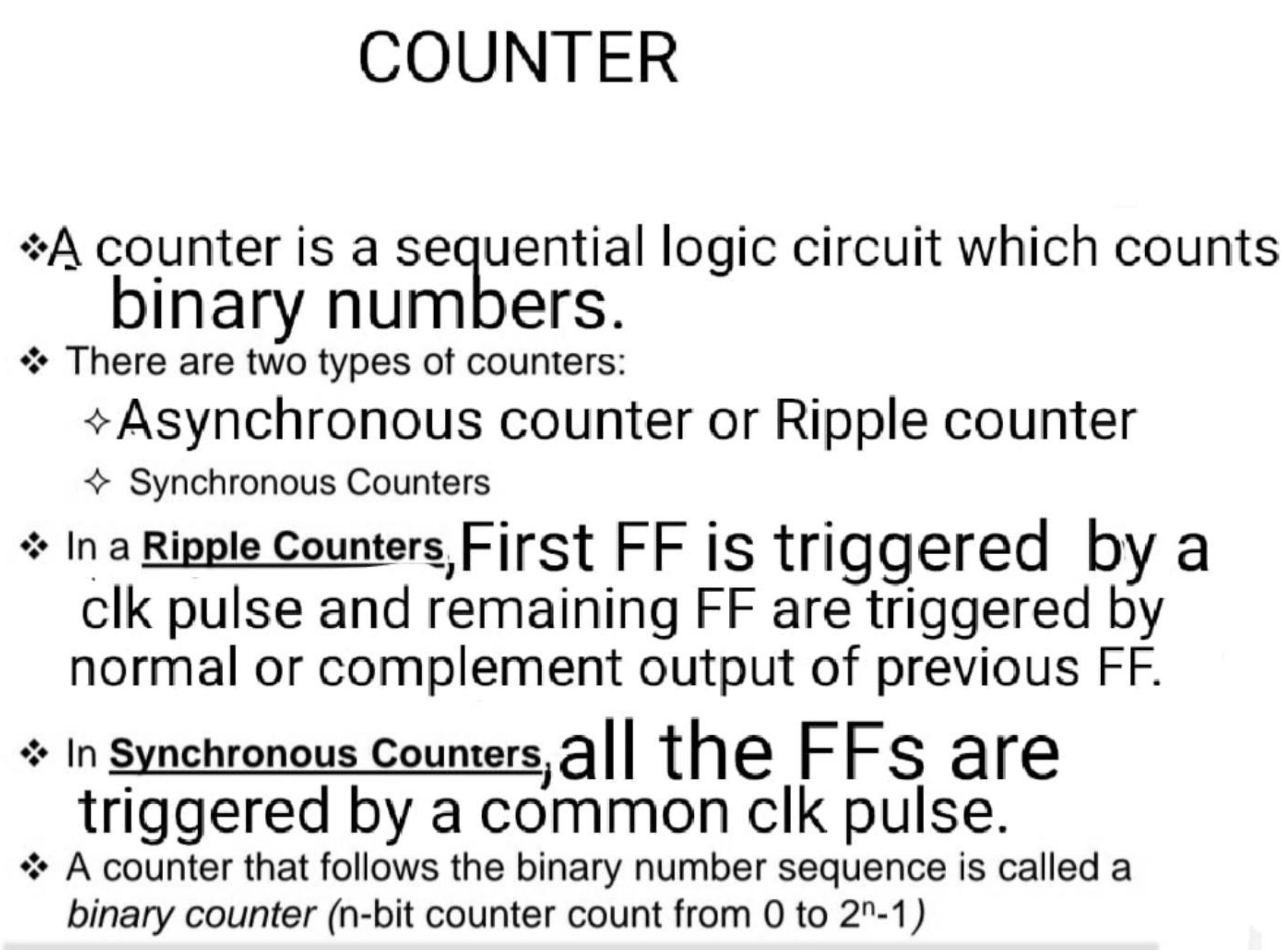
MASTER-SLAVE JK FLIP-FLOP:-

- The Master-Slave Flip-Flop is basically two gated SR flip-flops connected together in a seriesconfiguration 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 theoutputs 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 toggleof the JK flip flop as shown below.

The Master-Slave JK Flip Flop

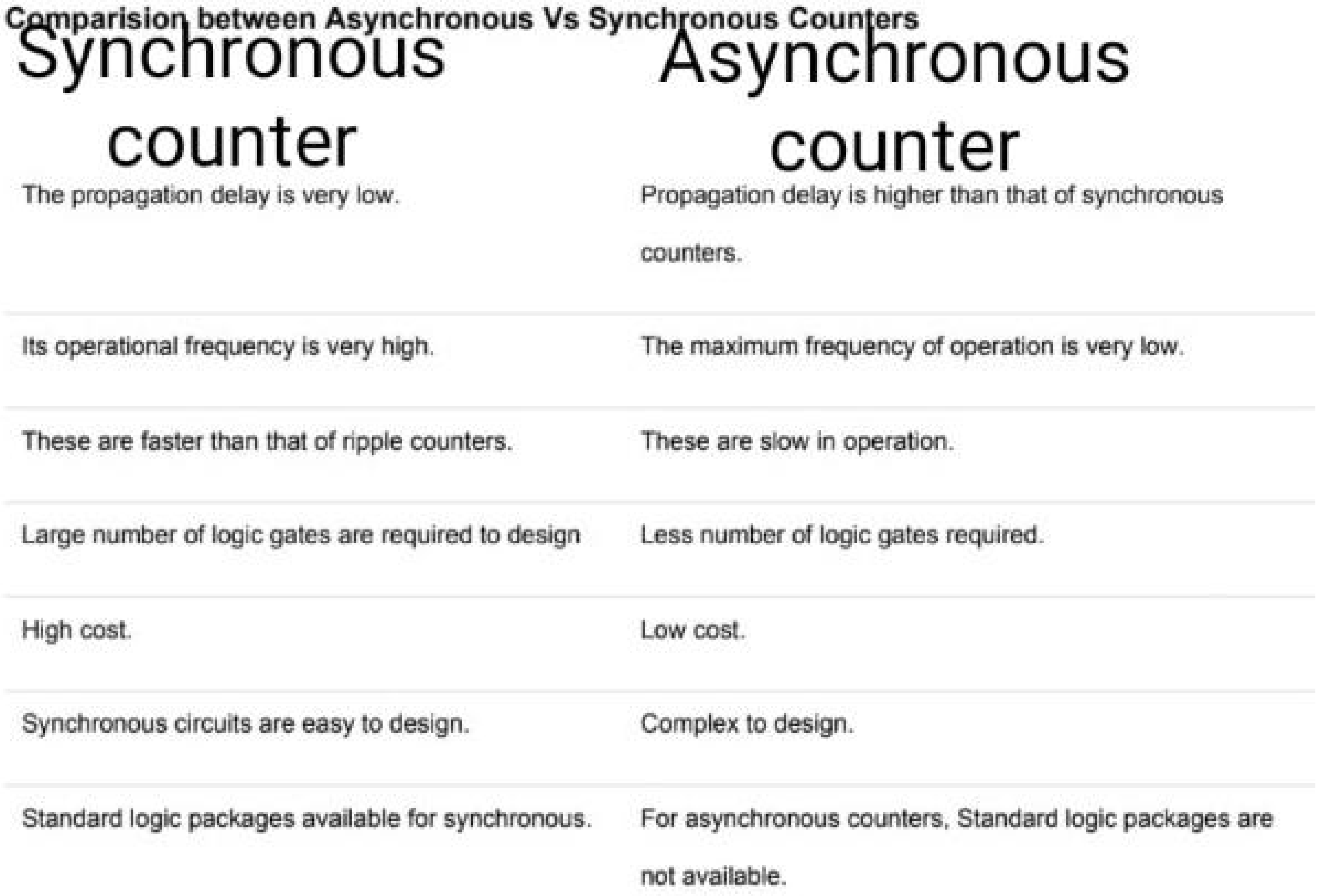


- The input signals J and K are connected to the gated "master" SR flip flop which "locks" the inputcondition while the clock (Clk) input is "HIGH" at logic level "1".
- As the clock input of the "slave" flip flop is the inverse (complement) of the "master" clock input, the"slave" SR flip flop does not toggle.
- The outputs from the "master" flip flop are only "seen" by the gated "slave" flip flop when the clock inputgoes "LOW" to logic level "0".
- When the clock is "LOW", the outputs from the "master" flip flop are latched and any additionalchanges 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.



COUNTER

- A counter is a sequential logic circuit which counts binary numbers.
- There are two types of counters:
  - Asynchronous counter or Ripple counter
  - Synchronous Counters
- In a **Ripple Counters**, First FF is triggered by a clk pulse and remaining FF are triggered by normal or complement output of previous FF.
- In **Synchronous Counters**, all the FFs are triggered by a common clk pulse.
- A counter that follows the binary number sequence is called a *binary counter* (n-bit counter count from 0 to 2^n-1)



Comparision between Asynchronous Vs Synchronous Counters

Synchronous counter	Asynchronous counter
The propagation delay is very low.	Propagation delay is higher than that of synchronous counters.
Its operational frequency is very high.	The maximum frequency of operation is very low.
These are faster than that of ripple counters.	These are slow in operation.
Large number of logic gates are required to design	Less number of logic gates required.
High cost.	Low cost.
Synchronous circuits are easy to design.	Complex to design.
Standard logic packages available for synchronous.	For asynchronous counters, Standard logic packages are not available.

Applications of counters :-

Counter found their applications in many digital electronic devices. Some of their applications are listed below.

1- Frequency counters

2- Digital clocks

3- Analog to digital convertors.

4- With some changes in their design, counters can be used as frequency divider circuits. The frequency divider circuit is that which divides the input frequency exactly by '2'.

5- In time measurement. That means calculating time in timers such as electronic devices like ovens and washing machines.

6- design digital triangular wave generator by using counters.

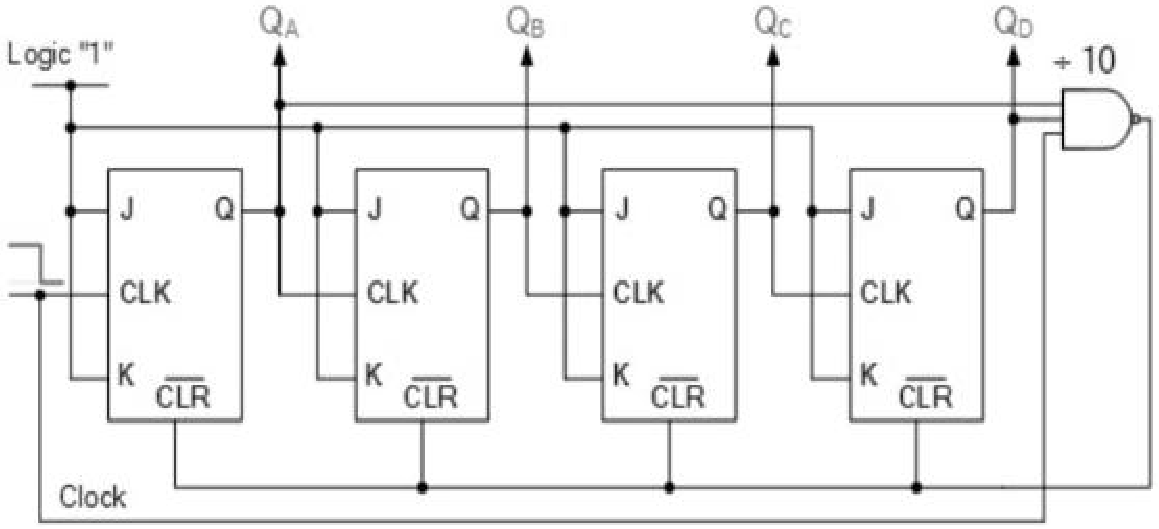
Modulus of a Counter:-

*The no. of states a counter can count is called its Modulus.

For eg;A MOD-5 Counter can count only 5 states.

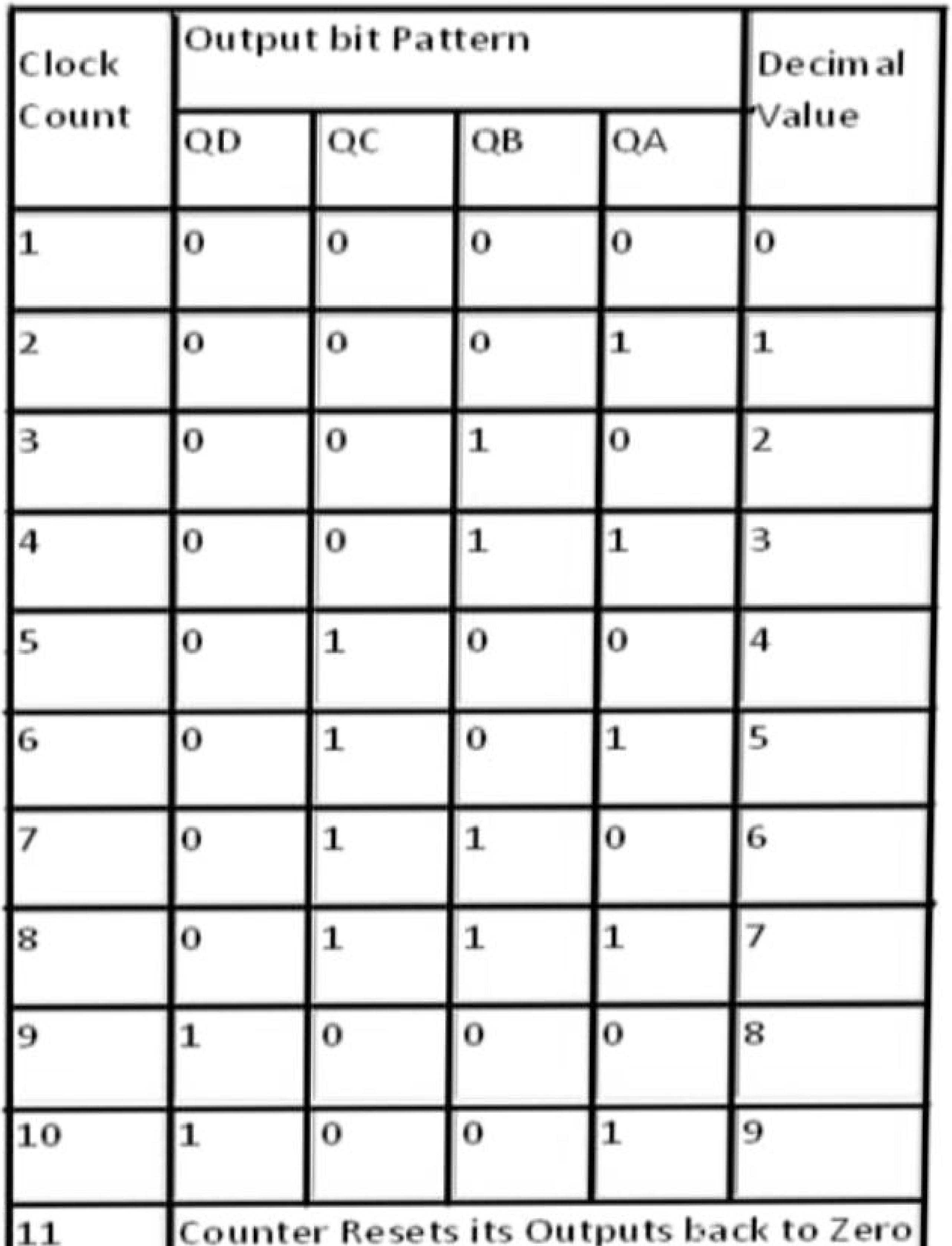
- Modulus counters are used in digital computers.
- A binary modulo-8 counter with three flip-flops, i.e., three stages, will produce an output pulse, i.e., display an output one-digit, after eight input pulses have been counted, i.e., entered or applied. This assumes that the counter started in the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

h 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				

4-bit Ripple Counter:-

Ans. 4-bit binary asynchronous or serial or ripple counter

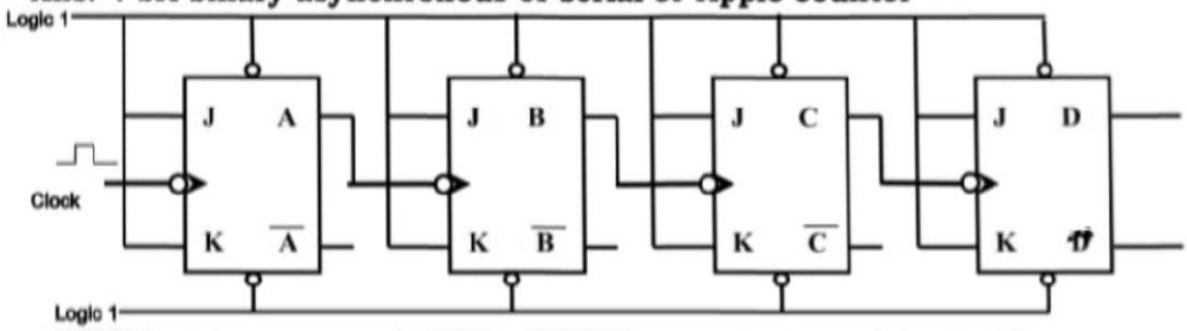


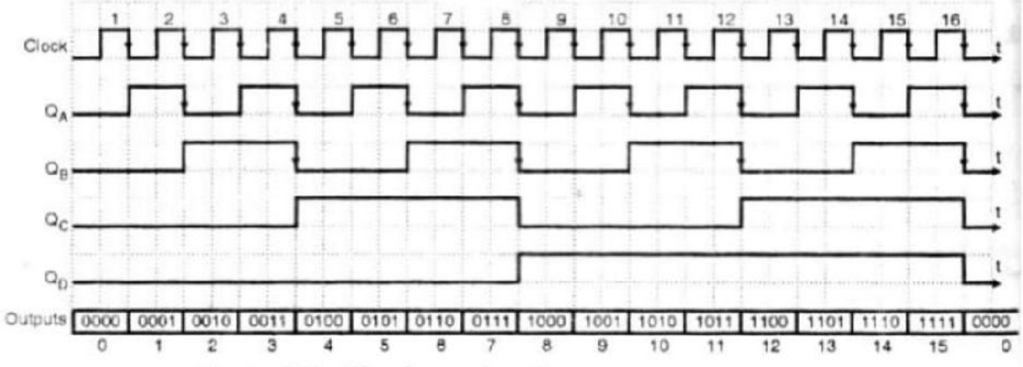
Fig. Shows a 4bit binary asynchronous counter.

It uses four negative edge triggered JK flip-flop. All the flip-flop will operate in the toggle mode because there J and K input are tied to Vcc. The clock pulse are applied to the flip-flop A. The output of flip-flop A drives clock input of flip-flop B, the output of flip-flop B drives clock input of flip-flop C and the output of flip-flop C drives clock input of flip-flop D. Since all flip-flop are negative edge triggered flip-flop they require a transition of 1 to 0 at their clock input to toggle or change the state.

Clock	D	C	B	A	Count
0(Intially)	0	0	0	0	1
1	0	0	0	1	2
2	0	0	1	0	3
3	0	0	1	1	4
4	0	1	0	0	5
5	0	1	0	1	6
6	0	1	1	0	7
7	0	1	1	1	8
8	1	0	0	0	9
9	1	0	0	1	10
10	1	0	1	0	11
11	1	0	1	1	12
12	1	1	0	0	13
13	1	1	0	1	14
14	1	1	1	0	15
15	1	1	1	1	16
16	0	0	0	0	17(0)

Truth Table for 4 bit Aysnchronous Counter

Timing diagram:-



Timing diagram

Operation of Counter

Initially all the flip-flop are cleared by using a common low clear signal. Therefore

DCBA = 0000

On the first clock pulse A flip-flop will toggle from 0 to 1 this will not trigger B flip-flop because it requires a change in 1 to 0 in A. Therefore B remain in last state and since B does change its state also C and D remain in last state. Hence on the first clock pulse we get output as,

DCBA = 0001

On the second clock pulse A flip-flop again toggles from 1 to 0. This now triggers B flip-flop, Now B FlipFlop toggles from 0 to 1. This will not affect C flip-flop because C flip-flop requires a change of 1 to 0 in B flip-flop. Therefore C remains 0 and so is D flip-flop. Hence on 2nd clock pulse we get

DCBA = 0010.

On the 3rd clock pulse A flip-flop changes from 0 to 1, B flip-flop remains at 1 and C and D flip-flop remain at 0 therefore,

DCBA = 0011.

On 4th clock pulse A flip-flop changes from 1 to 0 therefore B flip-flop now changes from 1 to 0. Now C flip-flop is triggered which will change from 0 to 1 but this will not effect D because it requires a change from 1 to 0 in C flip-flop. Hence D remains 0. Therefore or 4th clock pulse we get,

DCBA = 0100.

Thus it is observed that A flip-flop toggles with every clock pulse it receives. B flip-flop toggles whenever A flip-flop changes from 1 to 0.

C flip-flop toggles whenever B flip-flop changes from 1 to 0 and D flip-flop toggles whenever C changes from 1 to 0.

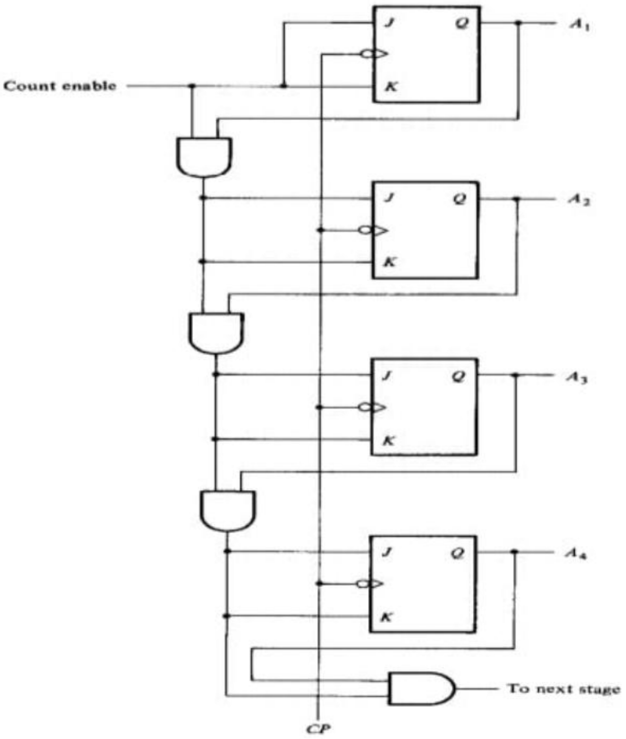
Hence on 15th clock pulse we get DCBA = 1111. On the next clock pulse A flip-flop changes from 1 to 0, B flip-flop change from 1 to 0. Therefore C flip-flop changes from 1 to 0. Hence D changes from 1 to 0, therefore all flip-flop are cleared again & we get,

DCBA = 0000

Thus this counter can count from 0 to 15 i.e. totally 16 count (or states). The number of discrete states through which the counter can progress on the application of pulse is given by 2^n where n= number of flip-flop used into the counter. If we connect 5 flip-flop the counter will progress through 00000 to 11111 i.e. 32 counts (0 to 31).

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.

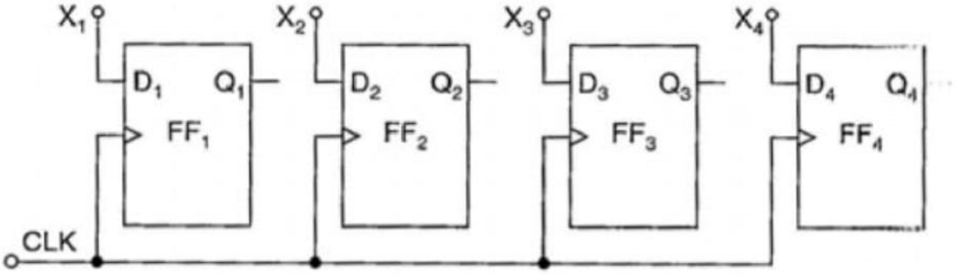
REGISTERS

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.

- The binary word to be stored is applied to the data terminals.
- 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 .

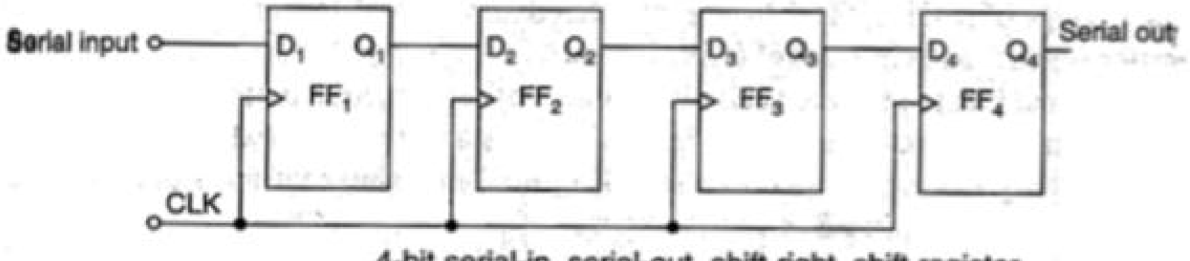
  This circuit is too primitive to be of any use.

Types of Shift Registers:-

- 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:-

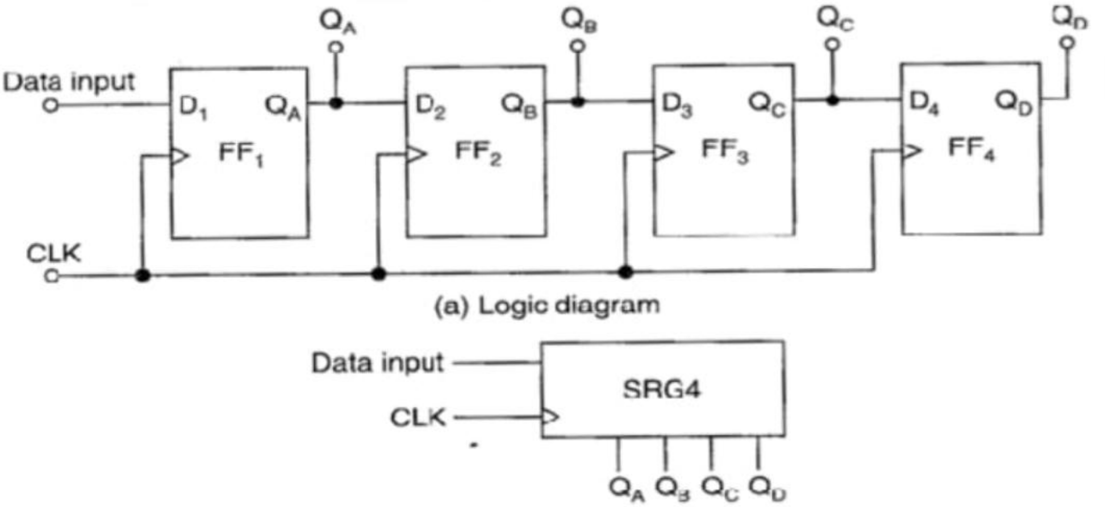
- This type of shift register accepts data serially, i.e., one bit at a time and also outputs data serially.
- The logic diagram of a four bit serial in, serial out shift register is shown in below figure:
- In 4 stages i.e. with 4 FFs, the register can store upto 4 bits of data.
- Serial data is applied 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 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.



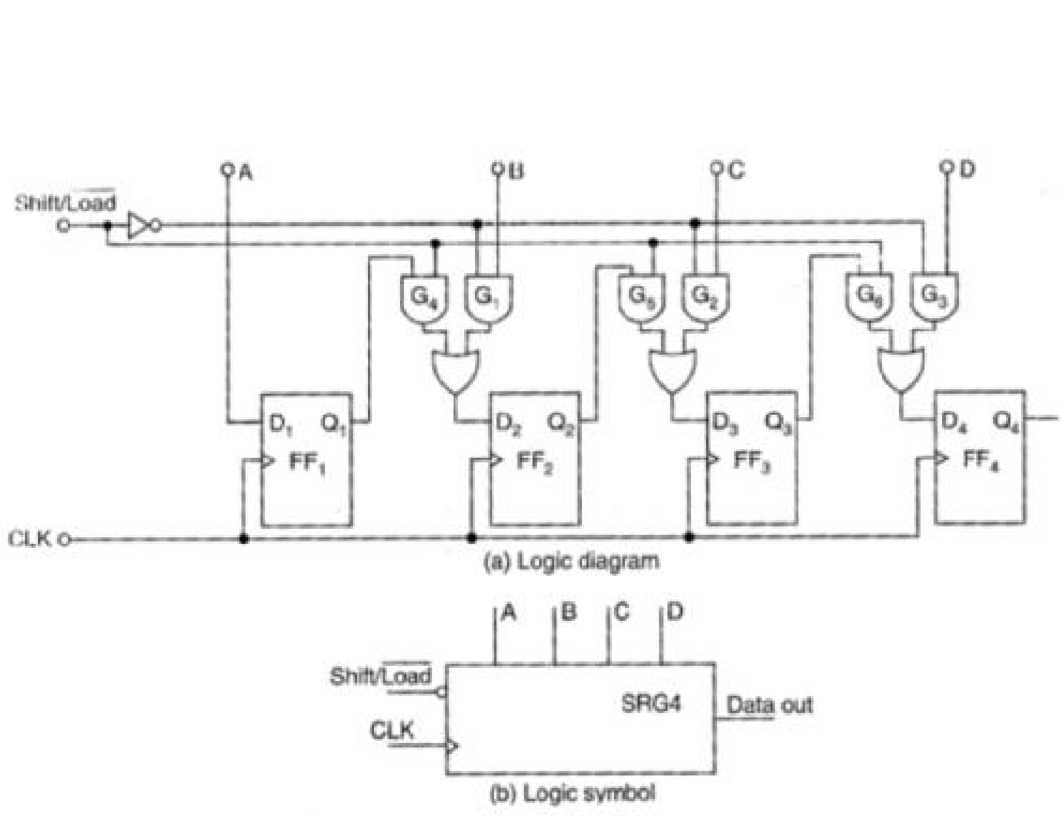
(a) Logic diagram

(b) Logic symbol

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 /$\overline{\text{LOAD}}$ 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 /$\overline{\text{LOAD}}$ 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 /$\overline{\text{LOAD}}$ input.



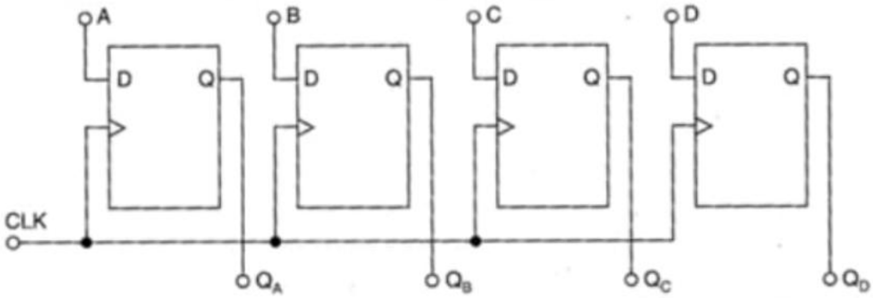
(a) Logic diagram

(b) Logic symbol

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.



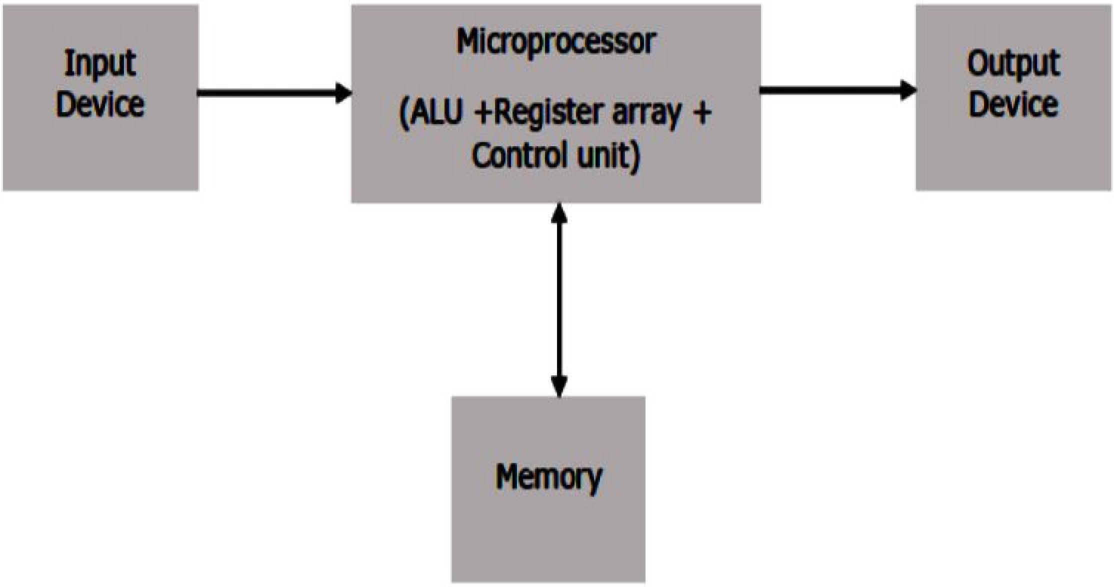
Logic diagram of a 4 – bit parallel in, parallel out shift register

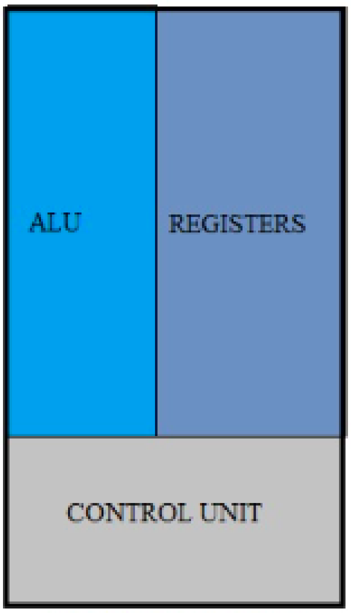
UNIT-4:

8085 MICROPROCESSOR

- The processor of a micro-computer is called microprocessor.
- It is a programable logical device which takes data as input and process the data according to the instruction given and gives output .
- Micro computers is a very small computer assembled to do a specific task.

BLOCK DIAGRAM OF A MICROPROCESSOR COMPUTER





MICROPROCESSOR

ALU:- Arithmetic and logical unit

- It performs all kinds of arithmetic and logical operation according to the instruction and data given to it .

Register Array:-

- Register stores data
- Register array means group of registers
- It stores input data and result for temporary period to be used by the ALU .

Control Unit :-

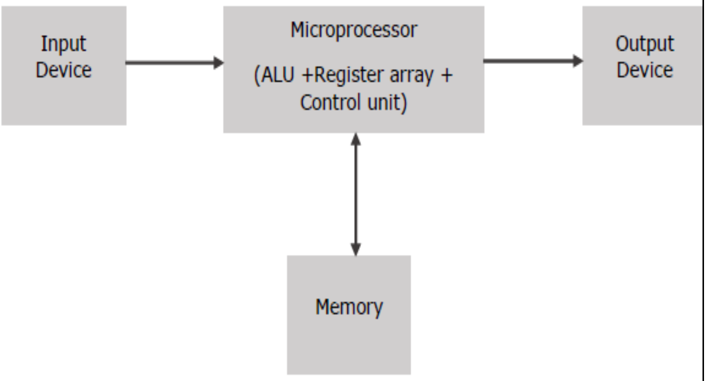
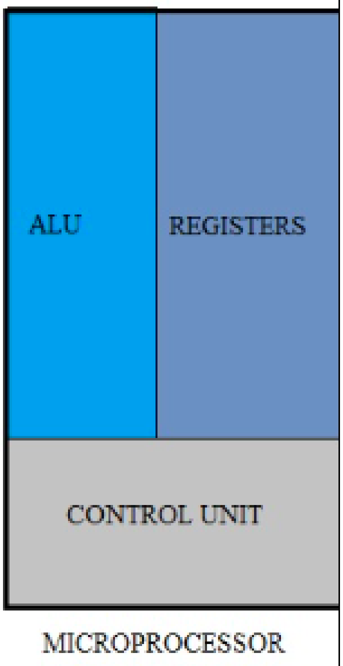
- It gives the control signal to the ALU and register array so that all the works are done in the proper co-ordination

Application of Microprocessor:-

- Domestic Application

- Industrial Application
- Automobiles Application
- Packing Industries application
- In robotics

Difference between microcomputer and microprocessor

Micro-Computer	Microprocessor
• Microcomputer is a very small computer assembled to do a specific task .	• The processor of a microcomputer is called microprocessor
 Input Device Microprocessor (ALU +Register array + Control unit) Output Device Memory	
• Micro-computer is a system	• Microprocessor is a component of the system

Evolution of microprocessor:-

- In 1971, Intel corporation of USA developed first microprocessor called "**Intel 4004**".
- It was unit processor that means at a time 4 bits of data could be handled by the processor.

- In the year 1971 again another 4 bit processor was developed named "**Intel 4040**"
- In 1972 first 8 bit processor was developed by Intel corporation named "**Intel 8008".** It was using P-MOS technology. So the speed was very low.
- In 1973 another 8 bit processor was developed by Intel corporation named "**Intel 8080".** It was faster because N-MOS technology was used.
- In 1975 one very successful 8-bit processor was developed by Intel corporation named "**Intel 8085".** It was using N-MOS technology and single power supply.
- 1978, first 16 bit processor was developed named "**Intel 8086**"
- In subsequent years many other companies took to develop microprocessor, such as zilogs, Motorola ,Celeron, Fairchild etc.
- The other 16 bit processors are **80186, 80286, 80386, 80486** etc. 280, 2800, PI, PII, PIII, PIV etc
- After successful development of 16 bit processor 32 bit processor was developed by many companies.
- Latest in market 64 bit processor are using this i-series such as **i3, i5, i7, i9.**

Word Length :-

- The data handling capacity of any processor is called word length of processor.
- Example:- Intel 8085 is a 8 bit processor. So its wordlength is = 8 bit
- Intel 4004 is a 4 bit processor so its wordlength is = 4 bit.

Intel 8085:-

Special features of Intel 8085:-

- It is a 8 bit processor.
- It uses N-MOS technology.
- It has 40 pins.
- It is a DIP chip (Dual in-line package).
- It uses frequency of 3mHz.
- It has a clock duration of 330 nsec.

$$\frac{1}{3mhz} = \frac{1}{3*10^2} = \frac{10^{-6}}{3} = \frac{1}{3} * 10^{-6}$$

$= 0.33 \times 10^{-6}$ sec

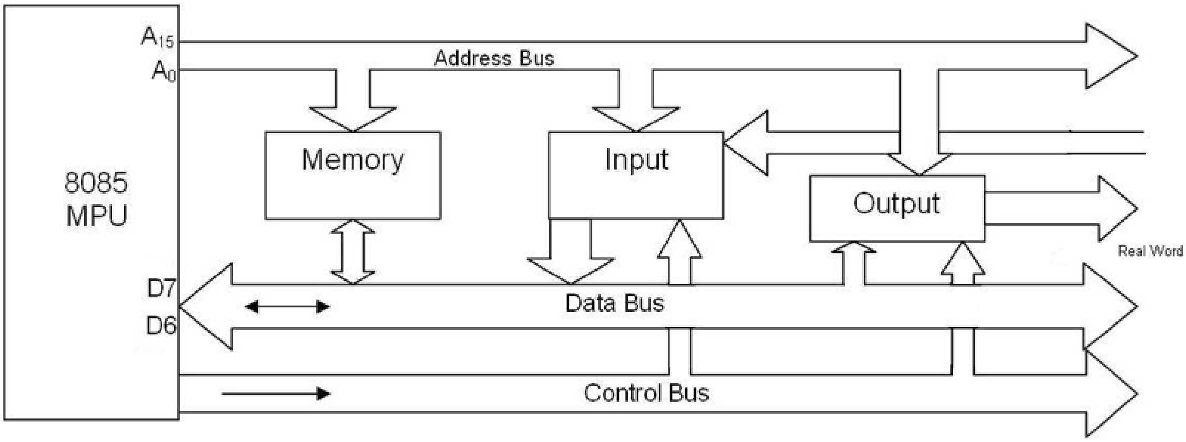
= 330 Nano second

- It uses +5v power supply.

BUS ARCHITECTURE OF INTEL 8085

BUS:-

- In a processor bus means no. of parallel line used for the transportation of address, data and control signals
- This transportation is done from the processor to the I/O device and memories
- There are 3 kinds of buses in Intel 8085.
  - Address bus
  - Data bus
  - Control bus



a) Address bus:-

- It is used to carry the address of a particulars location in memory or I/O device.
- In intel 8085, the address bus is of 16 bit circle
- It can allocate 64KB of memory location.

  => $2^{16} = 2^{10} * 2^{6}$ $(2^{10} = 1\text{KB})$

  => 64 x 1 KB = 64 KB.
- The size of address bus gives the information of memory allocation capacity.

- The address line of address bus starts with A_0 and finishes with A_{15} .
- The address bell is unidirectional.

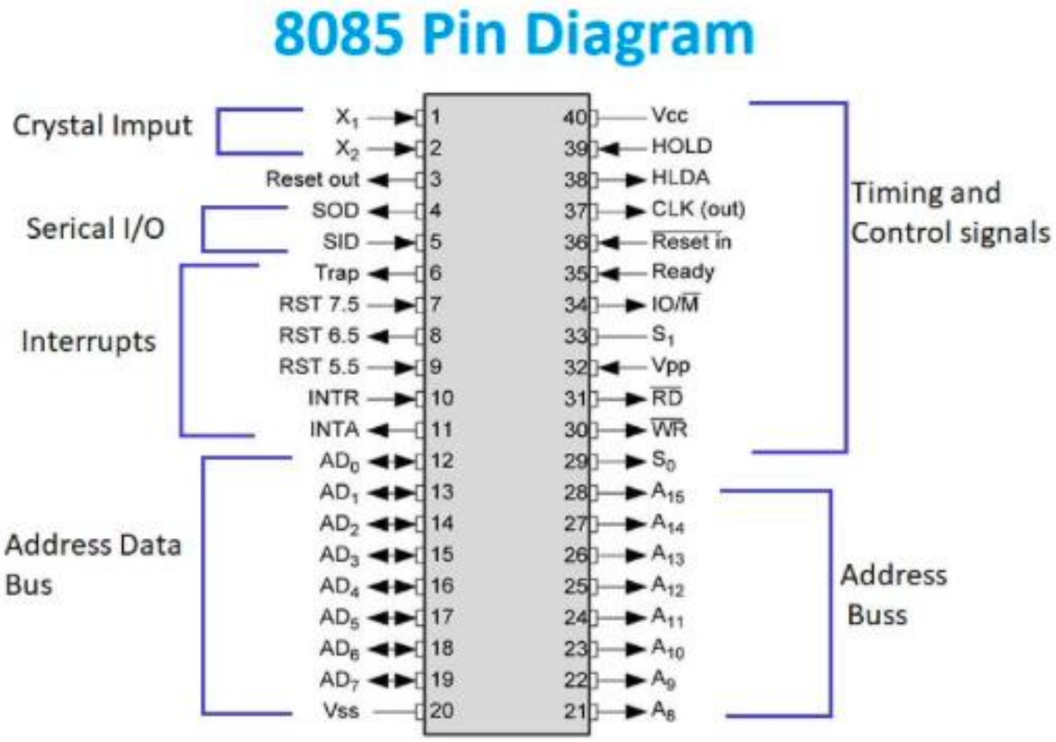
b) Data bus:-

- The data bus carries the data from processes to I/O devices and memory
- In Intel 8085 the data bus is 8 bit wide i.e. D_0 to D_7.
- The data bus size gives the information of data handling capacity or word length of the processor
- The data bus is always bi-directional.

c) Control bus:-

- This is also unidirectional
- It carries the control signals for data address bus.

PIN CONFIGURATION OF INTEL 8085



Multiplexing:-

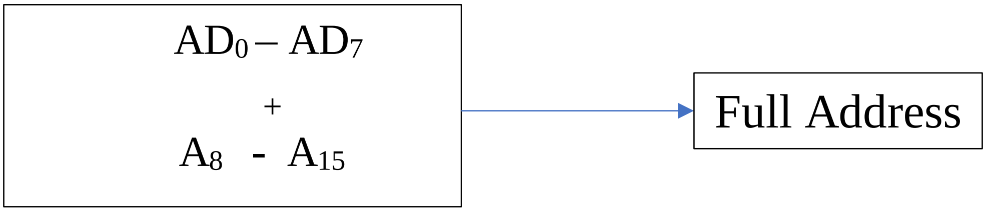
- The address bus size is 16 bit wide i.e. A_0 to A_{15} and data bus size is 8 bot wide i.e. D_0 to D_7.
- The total address bus can be divided into 2 groups i.e. lower order address bus A_0 to A_7 higher order address bus A_8 to A_{15}
- In multiplying the lower order address bus is mixed with data bus to AD bus.
- This is done to reduce the number of pins in the chip.

$AD_0 – AD_7$:-

- These pins are bidirectional pins
- These are address and data bus lines
- Through these pins lower order address as well as data go.

$A_8 – A_{15}$:-

- These are output pins
- These pins carry higher order address
- The lower order address goes through AD bus and higher order address goes through $A_8 – A_{15}$, they combine and make the full address.



ALE (Address latch Enable) :-

- This is an output pin
- When this plan goes high, the lower order address is mixed with higher order address and finally the data goes from the AD bus to the memory location.

IO/ $\overline{M}$

- This is an output pin

- When this pin goes high the processor is commuting with IO device using its buses.
- But when this pin goes low processor is communicating with memory.

$\overline{RD}$

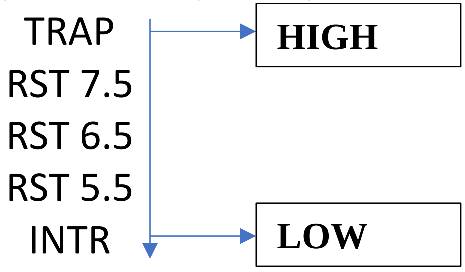
- This is a output pin.
- This is an active low pin/signal.
- When these pin. Goes low the processor reads the data from IO device or memory

$\overline{WR}$

- It is an output pin
- This is an active low pin/signal
- When these pin goes low the processor writes the data into the I/O device or memory.

TRAP, RST 7.5, 6.5, 5.5, INTR:-

- Interruption means disturbing the processor for a short duration.
- In Intel 8085 there are 5 interrupt lines, they are TRAP, RST 7.5, RST 6.5, RST 5.5 and INTR
- All these are active high pins
- TRAP has the highest priority and INTR priority



- These are all input pins
- The interrupt which can be avoided are called maskable interrupts those are RST 7.5, RST 6.5, RST 5.5 AND INTR
- Those interrupts which cannot be avoided are called non-maskable interrupt, i.e. TRAP is the non – maskable interrupt.

$\overline{\text{INTA}}$;- (Interrupt acknowledgement)

- It is an output pin.
- It is an active low signal
- When this pin goes low, the processor tells that it has received and interrupt request

HOLD :-

- This is an input pin, when this is high, the external devices request to the processor for the use of busses (address bus and data bus)

HLDA:-

- It stands for HOLD Acknowledgement
- When this pin goes high processor tells that, it has received the whole request and the control over the busses is given to the device as soon as the current work is complete.

$\overline{\textit{RESET IN}}$

- This is an input and active low pin
- It resets the PC 20 and it also resets interrupt enable lines and HLDA flip flops.

RESET OUT:-

- It is an output and active high pin
- When this goes high the CPU is in reset condition

X_2 & X_2:-

- These are external terminals connected to the crystal oscillator to produce a suitable clock for the operation of microprocessor.

SOD:-

Serial Output Data

- The 7th bit of the accumulator is placed on SOD line

SID:-

- It is a data line for serial input
- the data on this line is loaded into the 7th bit of the accumulator.

READY:-

- It is an input pin and It is used by the processor to sense whether the peripheral is ready or not for the data transfer.

S_0 & S_1:-

- These are status signals to let the users know that a processor is doing what kind of work.

S_1	S_0	
0	0	HLT
0	1	WRITE
1	0	READ
1	1	FETCH

$+V_{CC}$:-

- 5V Supply

Clk (out):-

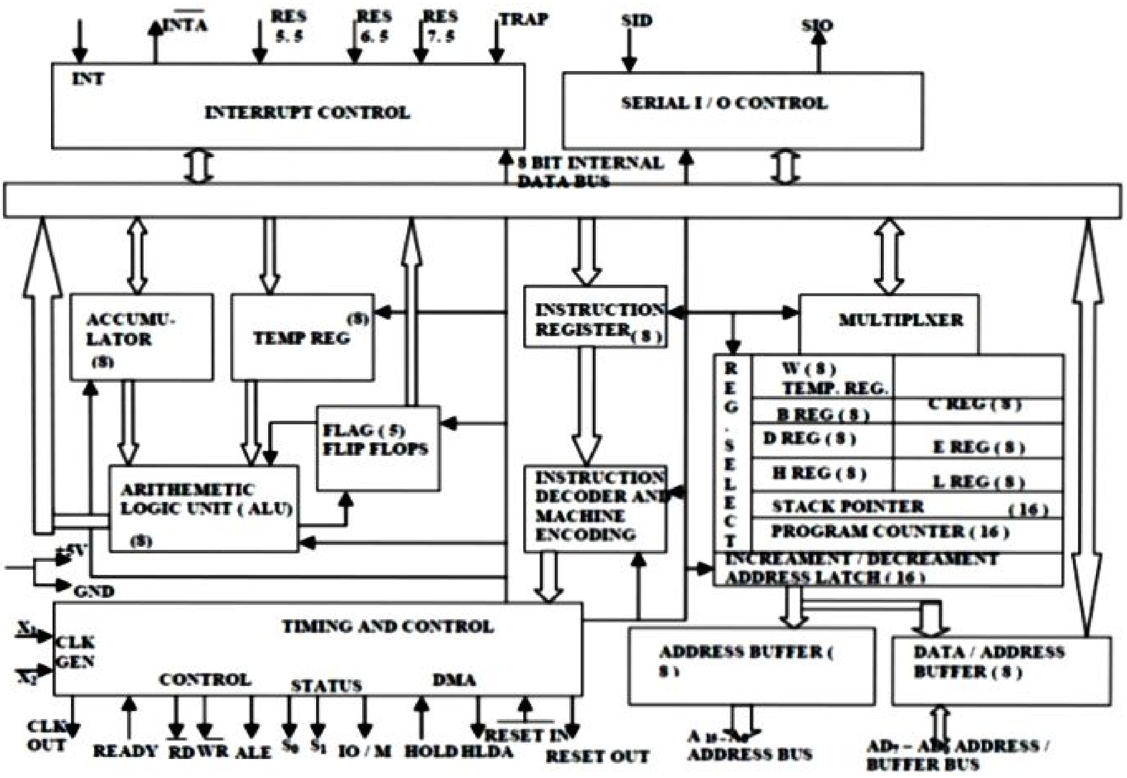
- Clock signal

V_{SS}:-

- Ground

. 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 2^{16} = 64 KB of memory. The internal architecture of 8085 is shown :



Internal Architecture of 8085

Arithmetic and Logic Unit

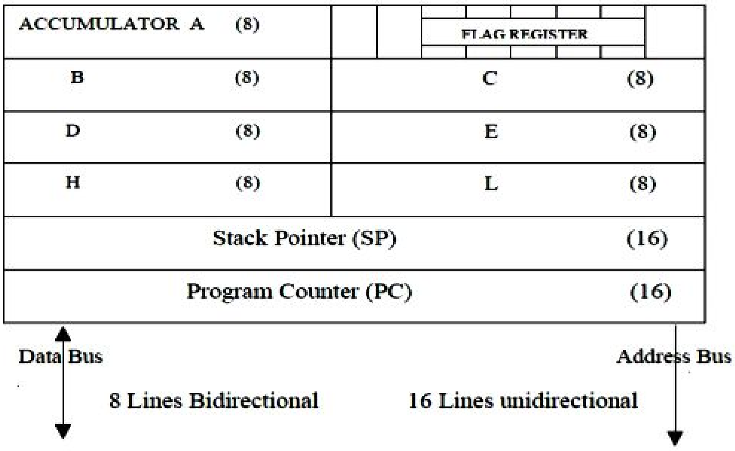
The ALU performs the actual numerical and logical operations such as Addition (ADD), Subtraction (SUB), AND, OR etc. It uses data from memory and from Accumulator to perform operations. The results of the arithmetic and logical operations are stored in the accumulator.

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. They are briefly described as follows.

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.



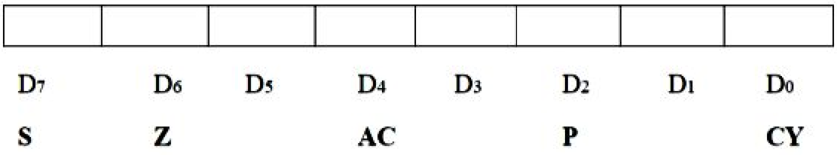
Register organisation

Accumulator

The accumulator is an 8-bit register that is a part of ALU. This register is used to store 8-bit data and to perform arithmetic and logical operations. The result of an operation is stored in the accumulator. The accumulator is also identified as register A.

Flag register

The ALU includes five flip-flops, which are set or reset after an operation according to data condition of the result in the accumulator and other registers. They are called Zero (Z), Carry (CY), Sign (S), Parity (P) and Auxiliary Carry (AC) flags. Their bit positions in the flag register are shown in Fig. 4. The microprocessor uses these flags to test data conditions.



Flag register

For example, after an addition of two numbers, if the result in the accumulator is larger than 8-bit, the flip-flop uses to indicate a carry by setting CY flag to 1. When an arithmetic operation results in zero, Z flag is set to 1. The S flag is just a copy of the bit D7 of the accumulator. A negative number has a 1 in bit D7 and a positive number has a 0 in 2's complement representation. The AC flag is set to 1, when a carry result from bit D3 and passes to bit D4. The P flag is set to 1, when the result in accumulator contains even number of 1s.

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.

INSTRUCTION SET AND EXECUTION IN 8085

Based on the design of the ALU and decoding unit, the microprocessor manufacturer provides instruction set for every microprocessor. The instruction set consists of both machine code and 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 memory location to another either conditionally or unconditionally.

V. Machine control operations: Instruction in this group control execution of other instructions and control operations 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.

Examples of one byte instructions

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

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

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 for specifying operands are called addressing modes. The 8085 has the following five types of addressing:

I. Immediate addressing
II. Memory direct addressing
III. Register direct addressing
IV. Indirect addressing
V. Implicit addressing

Immediate Addressing:

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

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 is given 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

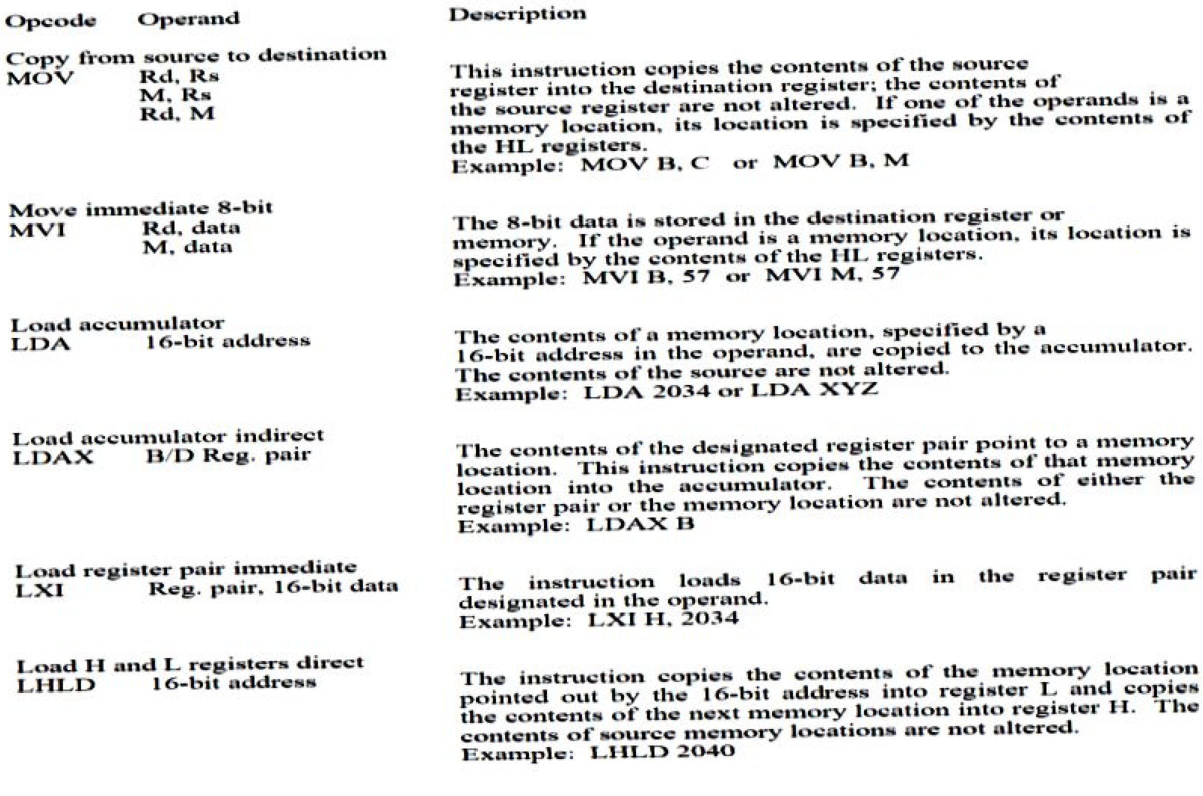
In this addressing mode the data itself specifies the data to be operated upon.

Ex: CMA

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

INSTRUCTION SET OF 8085

Data Transfer Instructions:



Opcode	Operand	Description
Copy from source to destination		
MOV	Rd, Rs M, Rs Rd, M	This instruction copies the contents of the source register into the destination register; the contents of the source register are not altered. If one of the operands is a memory location, its location is specified by the contents of the HL registers. Example: MOV B, C or MOV B, M
Move immediate 8-bit		
MVI	Rd, data M, data	The 8-bit data is stored in the destination register or memory. If the operand is a memory location, its location is specified by the contents of the HL registers. Example: MVI B, 57 or MVI M, 57
Load accumulator		
LDA	16-bit address	The contents of a memory location, specified by a 16-bit address in the operand, are copied to the accumulator. The contents of the source are not altered. Example: LDA 2034 or LDA XYZ
Load accumulator indirect		
LDAX	B/D Reg. pair	The contents of the designated register pair point to a memory location. This instruction copies the contents of that memory location into the accumulator. The contents of either the register pair or the memory location are not altered. Example: LDAX B
Load register pair immediate		
LXI	Reg. pair, 16-bit data	The instruction loads 16-bit data in the register pair designated in the operand. Example: LXI H, 2034
Load H and L registers direct		
LHLD	16-bit address	The instruction copies the contents of the memory location pointed out by the 16-bit address into register L and copies the contents of the next memory location into register H. The contents of source memory locations are not altered. Example: LHLD 2040

Store accumulator direct

STA 16-bit address

The contents of the accumulator are copied into the memory location specified by the operand. This is a 3-byte instruction, the second byte specifies the low-order address and the third byte specifies the high-order address.
Example: STA 4350 or STA XYZ

Store accumulator indirect

STAX Reg. pair

The contents of the accumulator are copied into the memory location specified by the contents of the operand (register pair). The contents of the accumulator are not altered.
Example: STAX B

Store H and L registers direct

SHLD 16-bit address

The contents of register L are stored into the memory location specified by the 16-bit address in the operand and the contents of H register are stored into the next memory location by incrementing the operand. The contents of registers HL are not altered. This is a 3-byte instruction, the second byte specifies the low-order address and the third byte specifies the high-order address.
Example: SHLD 2470

Exchange H and L with D and E

XCHG none

The contents of register H are exchanged with the contents of register D, and the contents of register L are exchanged with the contents of register E.
Example: XCHG

Copy H and L registers to the stack pointer

SPHL none

The instruction loads the contents of the H and L registers into the stack pointer register, the contents of the H register provide the high-order address and the contents of the L register provide the low-order address. The contents of the H and L registers are not altered.
Example: SPHL

Exchange H and L with top of stack

XTHL none

The contents of the L register are exchanged with the stack location pointed out by the contents of the stack pointer register. The contents of the H register are exchanged with the next stack location (SP+1); however, the contents of the stack pointer register are not altered.
Example: XTHL

Push register pair onto stack

PUSH Reg. pair

The contents of the register pair designated in the operand are copied onto the stack in the following sequence. The stack pointer register is decremented and the contents of the high-order register (B, D, H, A) are copied into that location. The stack pointer register is decremented again and the contents of the low-order register (C, E, L, flags) are copied to that location.

Example: PUSH B or PUSH A

Pop off stack to register pair

POP Reg. pair

The contents of the memory location pointed out by the stack pointer register are copied to the low-order register (C, E, L, status flags) of the operand. The stack pointer is incremented by 1 and the contents of that memory location are copied to the high-order register (B, D, H, A) of the operand. The stack pointer register is again incremented by 1.

Example: POP H or POP A

Output data from accumulator to a port with 8-bit address

OUT 8-bit port address

The contents of the accumulator are copied into the I/O port specified by the operand.

Example: OUT 87

Input data to accumulator from a port with 8-bit address

IN 8-bit port address

The contents of the input port designated in the operand are read and loaded into the accumulator.

Example: IN 82

Arithmetic Instructions:

Opcode Operand Description

Add register or memory to accumulator

ADD R
M

The contents of the operand (register or memory) are added to the contents of the accumulator and the result is stored in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the addition.

Example: ADD B or ADD M

Add register to accumulator with carry

ADC R
M

The contents of the operand (register or memory) and the Carry flag are added to the contents of the accumulator and the result is stored in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the addition.

Example: ADC B or ADC M

Add immediate to accumulator

ADI 8-bit data

The 8-bit data (operand) is added to the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the addition.

Example: ADI 45

Add immediate to accumulator with carry

ACI 8-bit data

The 8-bit data (operand) and the Carry flag are added to the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the addition.

Example: ACI 45

Add register pair to H and L registers

DAD Reg. pair

The 16-bit contents of the specified register pair are added to the contents of the HL register and the sum is stored in the HL register. The contents of the source register pair are not altered. If the result is larger than 16 bits, the CY flag is set. No other flags are affected.

Example: DAD H

Subtract register or memory from accumulator

SUB R
M

The contents of the operand (register or memory) are subtracted from the contents of the accumulator, and the result is stored in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the subtraction.

Example: SUB B or SUB M

Subtract source and borrow from accumulator

SBB R
M

The contents of the operand (register or memory) and the Borrow flag are subtracted from the contents of the accumulator and the result is placed in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the subtraction.

Example: SBB B or SBB M

Subtract immediate from accumulator

SUI 8-bit data

The 8-bit data (operand) is subtracted from the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the subtraction.

Example: SUI 45

Subtract immediate from accumulator with borrow

SBI 8-bit data

The 8-bit data (operand) and the Borrow flag are subtracted from the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the subtracion.

Example: SBI 45

Increment register or memory by 1

INR R
M

The contents of the designated register or memory) are incremented by 1 and the result is stored in the same place. If the operand is a memory location, its location is specified by the contents of the HL registers.

Example: INR B or INR M

Increment register pair by 1

INX R

The contents of the designated register pair are incremented by 1 and the result is stored in the same place.

Example: INX H

Decrement register or memory by 1
DCR R
M

The contents of the designated register or memory are decremented by 1 and the result is stored in the same place. If the operand is a memory location, its location is specified by the contents of the HL registers.
Example: DCR B or DCR M

Decrement register pair by 1
DCX R

The contents of the designated register pair are decremented by 1 and the result is stored in the same place.
Example: DCX H

Decimal adjust accumulator
DAA none

The contents of the accumulator are changed from a binary value to two 4-bit binary coded decimal (BCD) digits. This is the only instruction that uses the auxiliary flag to perform the binary to BCD conversion, and the conversion procedure is described below. S, Z, AC, P, CY flags are altered to reflect the results of the operation.

If the value of the low-order 4-bits in the accumulator is greater than 9 or if AC flag is set, the instruction adds 6 to the low-order four bits.

If the value of the high-order 4-bits in the accumulator is greater than 9 or if the Carry flag is set, the instruction adds 6 to the high-order four bits.

Example: DAA

BRANCHING INSTRUCTIONS

Opcode Operand Description

Jump unconditionally
JMP 16-bit address

The program sequence is transferred to the memory location specified by the 16-bit address given in the operand.
Example: JMP 2034 or JMP XYZ

Jump conditionally

Operand: 16-bit address

The program sequence is transferred to the memory location specified by the 16-bit address given in the operand based on the specified flag of the PSW as described below.
Example: JZ 2034 or JZ XYZ

Opcode	Description	Flag Status
JC	Jump on Carry	CY = 1
JNC	Jump on no Carry	CY = 0
JP	Jump on positive	S = 0
JM	Jump on minus	S = 1
JZ	Jump on zero	Z = 1
JNZ	Jump on no zero	Z = 0
JPE	Jump on parity even	P = 1
JPO	Jump on parity odd	P = 0

Unconditional subroutine call

CALL 16-bit address

The program sequence is transferred to the memory location specified by the 16-bit address given in the operand. Before the transfer, the address of the next instruction after CALL (the contents of the program counter) is pushed onto the stack.
Example: CALL 2034 or CALL XYZ

Call conditionally

Operand: 16-bit address

The program sequence is transferred to the memory location specified by the 16-bit address given in the operand based on the specified flag of the PSW as described below. Before the transfer, the address of the next instruction after the call (the contents of the program counter) is pushed onto the stack.
Example: CZ 2034 or CZ XYZ

Opcode	Description	Flag Status
CC	Call on Carry	CY = 1
CNC	Call on no Carry	CY = 0
CP	Call on positive	S = 0
CM	Call on minus	S = 1
CZ	Call on zero	Z = 1
CNZ	Call on no zero	Z = 0
CPE	Call on parity even	P = 1
CPO	Call on parity odd	P = 0

Return from subroutine unconditionally

RET none

The program sequence is transferred from the subroutine to the calling program. The two bytes from the top of the stack are copied into the program counter, and program execution begins at the new address.
Example: RET

Return from subroutine conditionally

Operand: none

The program sequence is transferred from the subroutine to the calling program based on the specified flag of the PSW as described below. The two bytes from the top of the stack are copied into the program counter, and program execution begins at the new address.
Example: RZ

Opcode	Description	Flag Status
RC	Return on Carry	CY = 1
RNC	Return on no Carry	CY = 0
RP	Return on positive	S = 0
RM	Return on minus	S = 1
RZ	Return on zero	Z = 1
RNZ	Return on no zero	Z = 0
RPE	Return on parity even	P = 1
RPO	Return on parity odd	P = 0

Load program counter with HL contents

PCHL none

The contents of registers H and L are copied into the program counter. The contents of H are placed as the high-order byte and the contents of L as the low-order byte.
Example: PCHL

Restart

RST 0-7

The RST instruction is equivalent to a 1-byte call instruction to one of eight memory locations depending upon the number. The instructions are generally used in conjunction with interrupts and inserted using external hardware. However these can be used as software instructions in a program to transfer program execution to one of the eight locations. The addresses are:

Instruction	Restart Address
RST 0	0000H
RST 1	0008H
RST 2	0010H
RST 3	0018H
RST 4	0020H
RST 5	0028H
RST 6	0030H
RST 7	0038H

The 8085 has four additional interrupts and these interrupts generate RST instructions internally and thus do not require any external hardware. These instructions and their Restart addresses are:

Interrupt	Restart Address
TRAP	0024H
RST 5.5	002CH
RST 6.5	0034H
RST 7.5	003CH

LOGICAL INSTRUCTIONS

Opcode Operand Description

Compare register or memory with accumulator

CMP R
M

The contents of the operand (register or memory) are compared with the contents of the accumulator. Both contents are preserved . The result of the comparison is shown by setting the flags of the PSW as follows:
if (A) < (reg/mem): carry flag is set, s=1
if (A) = (reg/mem): zero flag is set, s=0
if (A) > (reg/mem): carry and zero flags are reset, s=0
Example: CMP B or CMP M

Compare immediate with accumulator

CPI 8-bit data

The second byte (8-bit data) is compared with the contents of the accumulator. The values being compared remain unchanged. The result of the comparison is shown by setting the flags of the PSW as follows:
if (A) < data: carry flag is set, s=1
if (A) = data: zero flag is set, s=0
if (A) > data: carry and zero flags are reset, s=0
Example: CPI 89

Logical AND register or memory with accumulator

ANA R
M

The contents of the accumulator are logically ANDed with the contents of the operand (register or memory), and the result is placed in the accumulator. If the operand is a memory location, its address is specified by the contents of HL registers. S, Z, P are modified to reflect the result of the operation. CY is reset. AC is set.
Example: ANA B or ANA M

Logical AND immediate with accumulator

ANI 8-bit data

The contents of the accumulator are logically ANDed with the 8-bit data (operand) and the result is placed in the accumulator. S, Z, P are modified to reflect the result of the operation. CY is reset. AC is set.
Example: ANI 86

Exclusive OR register or memory with accumulator

XRA R
M

The contents of the accumulator are Exclusive ORed with the contents of the operand (register or memory), and the result is placed in the accumulator. If the operand is a memory location, its address is specified by the contents of HL registers. S, Z, P are modified to reflect the result of the operation. CY and AC are reset.

Example: XRA B or XRA M

Exclusive OR immediate with accumulator

XRI 8-bit data

The contents of the accumulator are Exclusive ORed with the 8-bit data (operand) and the result is placed in the accumulator. S, Z, P are modified to reflect the result of the operation. CY and AC are reset.

Example: XRI 86

Logical OR register or memory with accumulaotr

ORA R
M

The contents of the accumulator are logically ORed with the contents of the operand (register or memory), and the result is placed in the accumulator. If the operand is a memory location, its address is specified by the contents of HL registers. S, Z, P are modified to reflect the result of the operation. CY and AC are reset.

Example: ORA B or ORA M

Logical OR immediate with accumulator

ORI 8-bit data

The contents of the accumulator are logically ORed with the 8-bit data (operand) and the result is placed in the accumulator. S, Z, P are modified to reflect the result of the operation. CY and AC are reset.

Example: ORI 86

Rotate accumulator left

RLC none

Each binary bit of the accumulator is rotated left by one position. Bit D7 is placed in the position of D0 as well as in the Carry flag. CY is modified according to bit D7. S, Z, P, AC are not affected.

Example: RLC

Rotate accumulator right

RRC none

Each binary bit of the accumulator is rotated right by one position. Bit D0 is placed in the position of D7 as well as in the Carry flag. CY is modified according to bit D0. S, Z, P, AC are not affected.

Example: RRC

Rotate accumulator left through carry

RAL none

Each binary bit of the accumulator is rotated left by one position through the Carry flag. Bit D7 is placed in the Carry flag, and the Carry flag is placed in the least significant position D0. CY is modified according to bit D7. S, Z, P, AC are not affected.
Example: RAL

Rotate accumulator right through carry

RAR none

Each binary bit of the accumulator is rotated right by one position through the Carry flag. Bit D0 is placed in the Carry flag, and the Carry flag is placed in the most significant position D7. CY is modified according to bit D0. S, Z, P, AC are not affected.
Example: RAR

Complement accumulator

CMA none

The contents of the accumulator are complemented. No flags are affected.
Example: CMA

Complement carry

CMC none

The Carry flag is complemented. No other flags are affected.
Example: CMC

Set Carry

STC none

The Carry flag is set to 1. No other flags are affected.
Example: STC

CONTROL INSTRUCTIONS

Opcode Operand Description

No operation

NOP none

No operation is performed. The instruction is fetched and decoded. However no operation is executed.
Example: NOP

Halt and enter wait state

HLT none

The CPU finishes executing the current instruction and halts any further execution. An interrupt or reset is necessary to exit from the halt state.
Example: HLT

Disable interrupts

DI none

The interrupt enable flip-flop is reset and all the interrupts except the TRAP are disabled. No flags are affected.
Example: DI

Enable interrupts

EI none

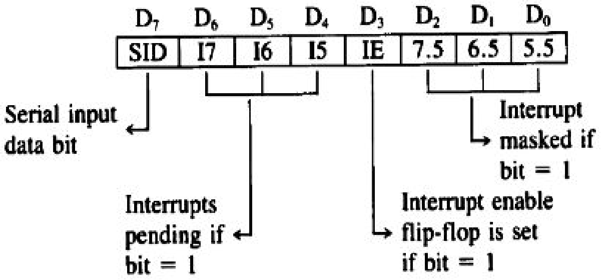
The interrupt enable flip-flop is set and all interrupts are enabled. No flags are affected. After a system reset or the acknowledgement of an interrupt, the interrupt enable flip-flop is reset, thus disabling the interrupts. This instruction is necessary to reenable the interrupts (except TRAP).
Example: EI

Read interrupt mask

RIM none

This is a multipurpose instruction used to read the status of interrupts 7.5, 6.5, 5.5 and read serial data input bit. The instruction loads eight bits in the accumulator with the following interpretations.

Example: RIM



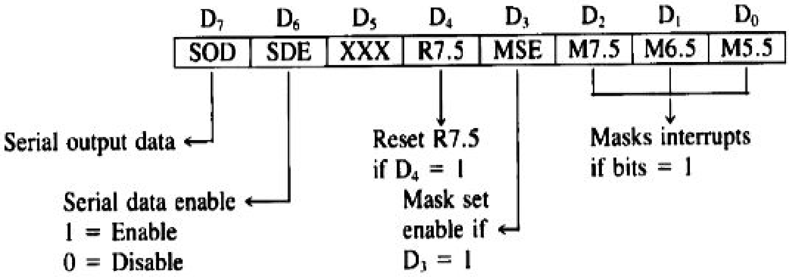
D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
SID I7 I6 I5 IE 7.5 6.5 5.5
Serial input data bit
Interrupts pending if bit = 1
Interrupt enable flip-flop is set if bit = 1
Interrupt masked if bit = 1

Set interrupt mask

SIM none

This is a multipurpose instruction and used to implement the 8085 interrupts 7.5, 6.5, 5.5, and serial data output. The instruction interprets the accumulator contents as follows.

Example: SIM



D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
SOD SDE XXX R7.5 MSE M7.5 M6.5 M5.5
Serial output data
Serial data enable
1 = Enable
0 = Disable
Reset R7.5 if D_4 = 1
Mask set enable if D_3 = 1
Masks interrupts if bits = 1

- □ SOD—Serial Output Data: Bit D_7 of the accumulator is latched into the SOD output line and made available to a serial peripheral if bit D_6 = 1.
- □ SDE—Serial Data Enable: If this bit = 1, it enables the serial output. To implement serial output, this bit needs to be enabled.
- □ XXX—Don't Care
- □ R7.5—Reset RST 7.5: If this bit = 1, RST 7.5 flip-flop is reset. This is an additional control to reset RST 7.5.
- □ MSE—Mask Set Enable: If this bit is high, it enables the functions of bits D_2, D_1, D_0. This is a master control over all the interrupt masking bits. If this bit is low, bits D_2, D_1, and D_0 do not have any effect on the masks.

□ M7.5—D_2	=	0, RST 7.5 is enabled.
	=	1, RST 7.5 is masked or disabled.
□ M6.5—D_1	=	0, RST 6.5 is enabled.
	=	1, RST 6.5 is masked or disabled.
□ M5.5—D_0	=	0, RST 5.5 is enabled.
	=	1, RST 5.5 is masked or disabled.

Instruction cycle of 8085 Microprocessor

Time required to execute and fetch an entire instruction is called instruction cycle. It consists:

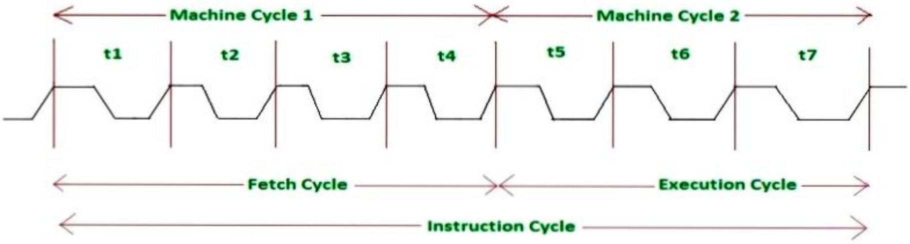
Fetch cycle – The next instruction is fetched by the address stored in program counter (PC) and then stored in the instruction register.

Decode instruction – Decoder interprets the encoded instruction from instruction register.

Execution cycle – consists memory read (MR), memory write (MW), input output read (IOR) and input output write (IOW)

The time required by the microprocessor to complete an operation of accessing memory or input/output devices is called **machine cycle.** One time period of frequency of microprocessor is called **t-state**. A t-state is measured from the falling edge of one clock pulse to the falling edge of the next clock pulse.

Fetch cycle takes four t-states and execution cycle takes three t-states.

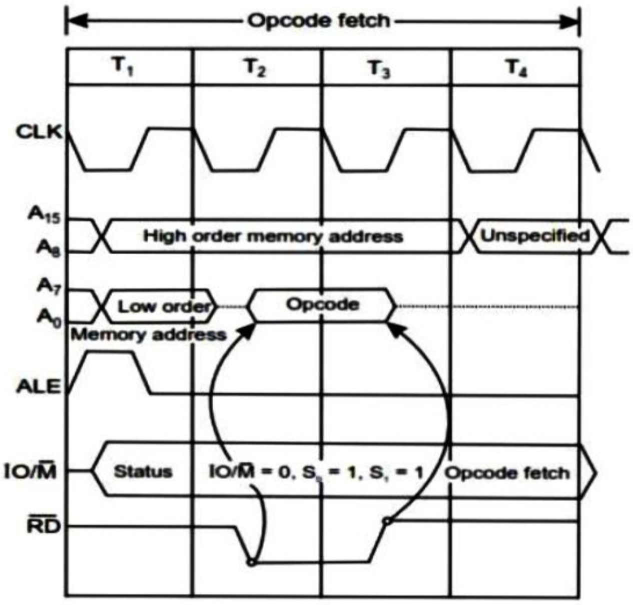


Instruction cycle in 8085 microprocessor

Timing diagram

Timing Diagram is a graphical representation. It represents the execution time taken by each instruction in a graphical format. The execution time is represented in T-states.

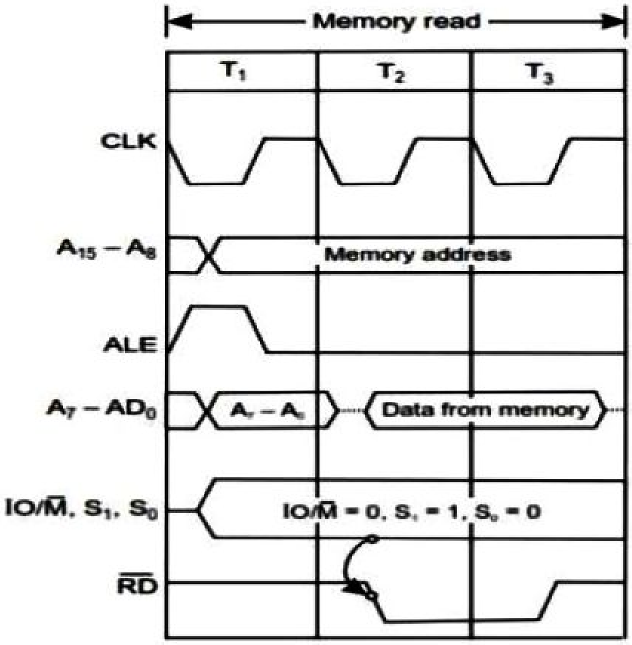
Opcode fetch cycle



- Each instruction of the processor has one byte opcode.
- The opcodes are stored in memory. So, the processor executes the opcode fetch machine cycle to fetch the opcode from memory.
- Hence, every instruction starts with opcode fetch machine cycle.
- The time taken by the processor to execute the opcode fetch cycle is 4T.
- In this time, the first, 3 T-states are used for fetching the opcode from memory and the remaining T-states are used for internal operations by the processor.

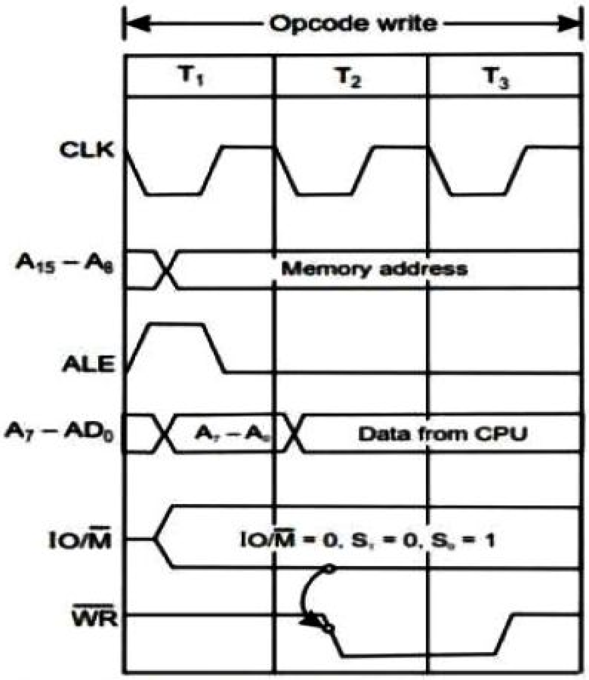
Memory read cycle

- The memory read machine cycle is executed by the processor to read a data byte from memory.
- The processor takes 3T states to execute this cycle.
- The instructions which have more than one byte word size will use the machine cycle after the opcode fetch machine cycle.



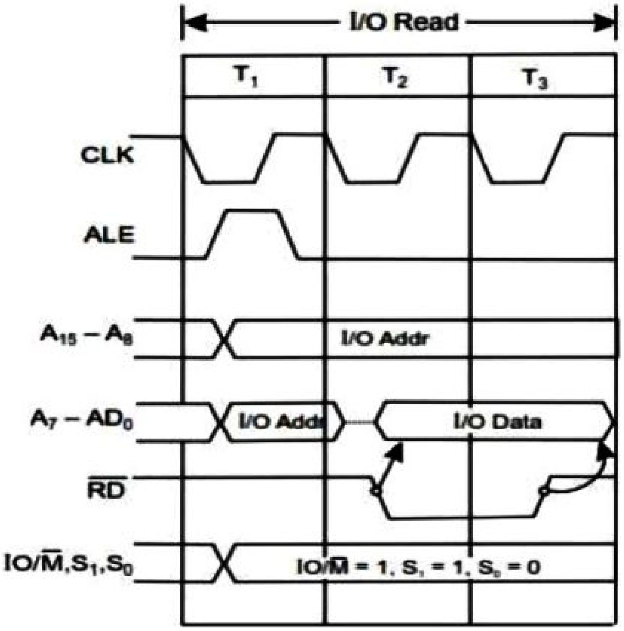
Memory write cycle

- The memory write machine cycle is executed by the processor to write a data byte in a memory location.
- The processor takes, 3T states to execute this machine cycle.



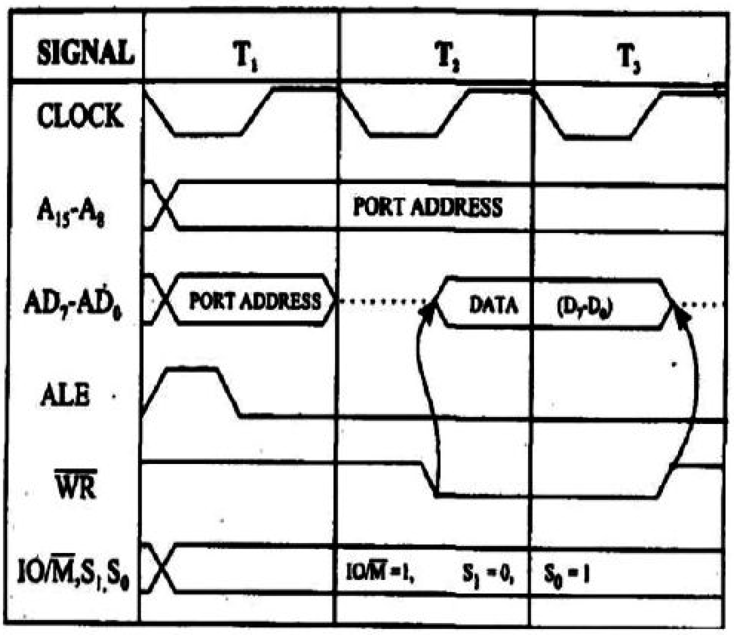
I/O read cycle

- The I/O Read cycle is executed by the processor to read a data byte from I/O port or from the peripheral, which is I/O, mapped in the system.
- The processor takes 3T states to execute this machine cycle.
- The IN instruction uses this machine cycle during the execution.



I/O write cycle

- The I/O Read cycle is executed by the processor to write a data byte from system to I/O port or peripheral, which is I/O mapped.
- The processor takes 3T states to execute this machine cycle.
- The OUT instruction uses this machine cycle during the execution.

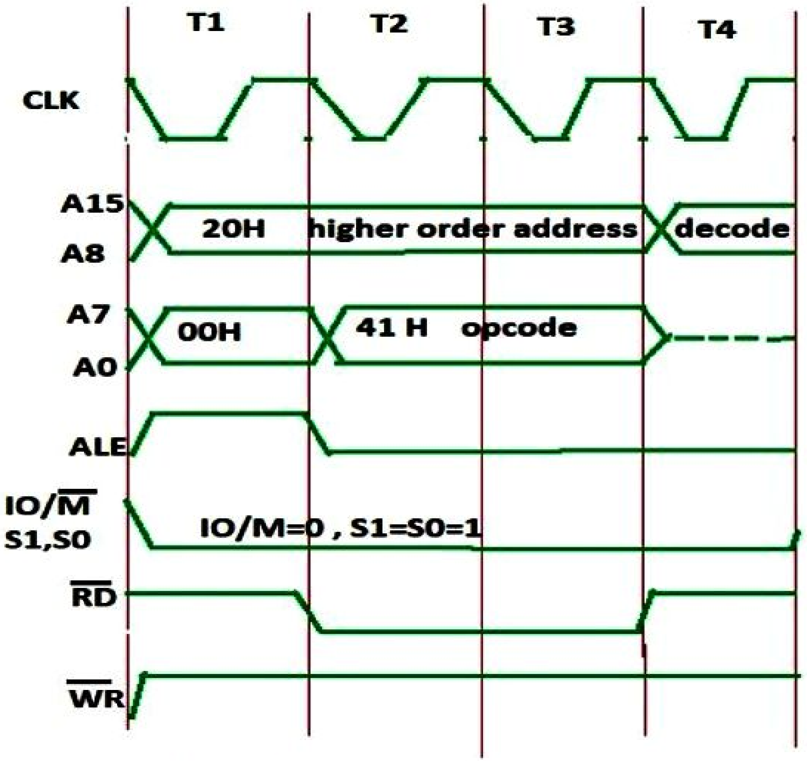


Example-1

The instruction MOV B, C is of 1 byte; therefore, the complete instruction will be stored in a single memory address.

2000 MOV B,C

Only opcode fetching is required for this instruction and thus we need 4 T states for the timing diagram. For the opcode fetch the IO/M (low active) = 0, S1 = 1 and S0 = 1.



In Opcode fetch (t1-t4 T-states):

1. 00 – lower bit of address where opcode is stored, i.e., 00 2.

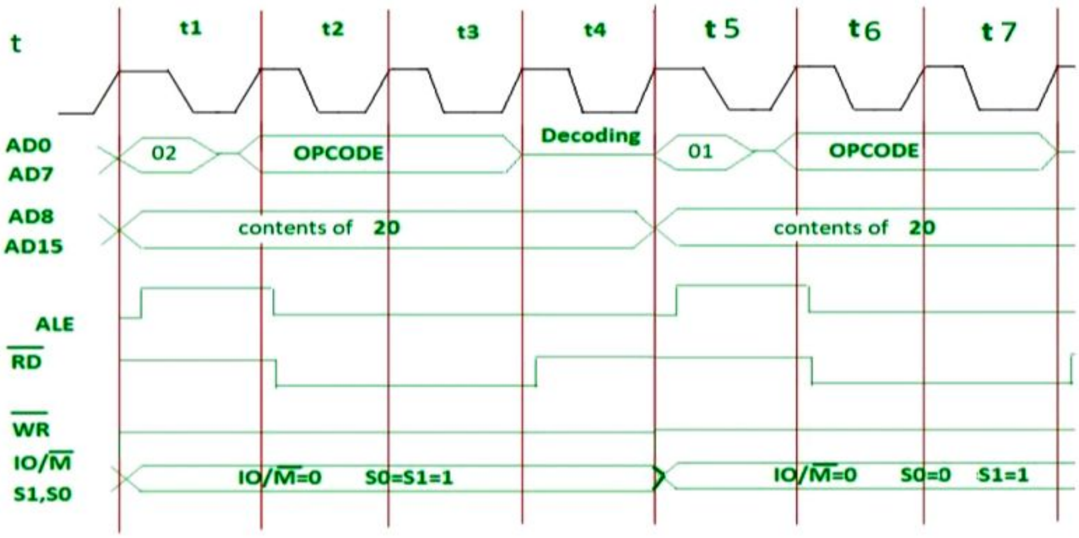
20 – higher bit of address where opcode is stored, i.e., 20.

3. ALE – provides signal for multiplexed address and data bus. Only in t1 it used as address bus to fetch lower bit of address otherwise it will be used as data bus.
4. RD (low active) – signal is 1 in t1 & t4 as no data is read by microprocessor. Signal is 0 in t2 & t3 because here the data is read by microprocessor.
5. WR (low active) – signal is 1 throughout, no data is written by microprocessor.
6. IO/M (low active) – signal is 1 in throughout because the operation is performing on memory.
7. S0 and S1 – both are 1 in case of opcode fetching.

Example-2

MVI B, 45
2000: Opcode
2001: 45

- The opcode fetch will be same in all the instructions.
- Only the read instruction of the opcode needs to be added in the successive T states.
- For the opcode fetch the IO/M (low active) = 0, S1 = 1 and S0 = 1. Also, 4 T states will be required to fetch the opcode from memory.
- For the opcode read the IO/M (low active) = 0, S1 = 1 and S0 = 0. Also, only 3 T states will be required to read data from memory.



In Opcode fetch (t1-t4 T-states) –

1. 00 – lower bit of address where opcode is stored.
2. 20 – higher bit of address where opcode is stored.
3. ALE – Provides signal for multiplexed address and data bus. Only in t1 it used as address bus to fetch lower bit of address otherwise it will be used as data bus.
4. RD (low active) – Signal is 1 in t1 & t4, no data is read by microprocessor. Signal is 0 in t2 & t3, data is read by microprocessor.
5. WR (low active) – Signal is 1 throughout, no data is written by microprocessor.
6. IO/M (low active) – Signal is 0 in throughout, operation is performing on memory.
7. S0 and S1 – Signal is 1 in t1 to t4 states, as to fetch the opcode from the memory.

In Opcode read (t5-t7 T-states) –

1. 01 – lower bit of address where data is stored.
2. 320 – higher bit of address where data is stored.
3. ALE – Provides signal for multiplexed address and data bus. Only in t5 it used as address bus to fetch lower bit of address otherwise it will be used as data bus.
4. RD (low active) – Signal is 1 in t5 as no data is read by microprocessor. Signal is 0 in t6 & t7 as data is read by microprocessor.
5. WR (low active) – Signal is 1 throughout, no data is written by microprocessor.
6. IO/M (low active) – Signal is 0 in throughout, operation is performing on memory.
7. S0 – Signal is 0 in throughout, operation is performing on memory to read data 45.
8. S1 – Signal is 1 throughout, operation is performing on memory to read data 45.

Intel8085: Basic Programs:

Ex1: Place 05 in register B

Mnemonics	Operands	Memory Address	Machine Code	Comments
MVI	B, 05	2000 H	06, 05	Get 05 in register B
HLT		2002 H	76	Stop

Ex2: Get 05 in register A; then move it to register B

Mnemonics	Operands	Memory Address	Machine Code	Comments
MVI	A, 05	2000 H	3E, 05	Get 05 in register A
MOV	B, A	2002 H	47	Content of register A is moved to register B
HLT		2003 H	76	Stop

Ex3: Load the content of memory location 3550H directly to accumulator, then transfer to register B. The content of memory location 3550H is 05

Mnemonics	Operands	Memory Address	Machine Code
LDA	3550H	2000 H	3A, 50, 35
MOV	B, A	2003 H	47
HLT		2004 H	76

Ex4: Move the content of memory location 3550H to register C. The content of the memory location 3550H is 08

Mnemonics	Operands	Memory Address	Machine Code
LXI	H, 3550H	2000 H	21, 50, 35
MOV	C, M	2003 H	4E
HLT		2004 H	76

Ex5: Place the content of the memory location FC50H in register B and that of FC51H in register C. The content of FC50H and FC51H are 11H and 12H respectively

Mnemonics	Operands	Memory Address	Machine Code
LXI	H, FC50H	2500H	21, 50, FC
MOV	B, M	2503H	46
INX	H	2504H	23
MOV	C, M	2505H	4E
HLT		2506H	76

The register B and C will contain 11H and 12H

DATA:

FC50 – 11H

FC51 – 12H

Ex6: Place 05 in the accumulator. Increment it by one and store the result in the memory location 3600H

Mnemonics	Operands	Memory Address	Machine Code
MVI	A, 05	2500 H	3E, 05
INR	A	2502 H	3C
STA	3600 H	2503 H	32, 00, 36
HLT		2506 H	76

INR A increases the content of accumulator from 05 to 06

RESULT:

3600 H – 06 H

Addition of Two 8-bit Numbers: Sum 8-bit

Add 49 H and 56 H. **Method#1**

The 1st number 49 H is in the memory location 2501 H.

The 2nd number 56 H is in the memory location 2502 H.

The result to be stored in the memory location 2503 H.

Mnemonics	Operands	Memory Address	Machine Code
LDA	2501 H	2000 H	3A, 01, 25
MOV	B, A	2003 H	47
LDA	2502 H	2004 H	3A, 02, 25
ADD	B	2007 H	80
STA	2503 H	2008 H	32, 03, 25
HLT		200B H	76

DATA

2501 – 49 H

2502 – 56 H

By Method#2

Mnemonics	Operands	Memory Address	Machine Code
LXI	H, 2501 H	2000 H	21, 01, 25
MOV	A, M	2003 H	7E
INX	H	2004 H	23
ADD	M	2005 H	86
STA	2503 H	2006 H	32, 03, 25
HLT		2009 H	76

DATA

2501 – 49 H

2502 – 56 H

Addition of Two 8-bit Numbers: Sum 16-bit

Add 98 H and 9A H.   Method#1

The 1st number 98 H is in the memory location 2501 H.

The 2nd number 9A H is in the memory location 2502 H.

The result to be stored in the memory location 2503 H & 2504 H.

	Mnemonics	Operands	Memory Address	Machine Code
	MVI	C, 00H	2000 H	0E, 00
	LDA	2501 H	2002 H	3A, 01, 25
	MOV	B, A	2005 H	47
	LDA	2502 H	2006 H	3A, 02, 25
	ADD	B	2009 H	80
	JNC	AHEAD	200A H	D2, 0E, 20
	INR	C	200D H	0C
AHEAD:	STA	2503 H	200E H	32, 03, 25
	MOV	A, C	2011 H	79
	STA	2504 H	2012 H	32, 04, 25
	HLT		2015 H	76

DATA

2501 – 98 H

2502 – 9A H

Result

2503 – 32 H

2504 – 01 H

Method#2

	Mnemonics	Operands	Memory Address	Machine Code
	MVI	C, 00H	2000 H	0E, 00
	LXI	H, 2501 H	2002 H	21, 01, 25
	MOV	A, M	2005 H	7E
	INX	H	2006 H	23
	ADD	M	2007 H	86
	JNC	AHEAD	2008 H	D2, 0C, 20
	INR	C	200B H	0C
AHEAD:	STA	2503 H	200C H	32, 03, 25
	MOV	A, C	200F H	79
	STA	2504 H	2010 H	32, 04, 25
	HLT		2013 H	76

DATA

2501 – 98 H

2502 – 9A H

Result

2503 – 32 H

2504 – 01 H

Decimal Addition of Two 8-bit Numbers:

Add 84 D and 75 D.

The 1st number 84 D is in the memory location 2501 H.

The 2nd number 75 D is in the memory location 2502 H.

The result to be stored in the memory location 2503 H & 2504 H.

	Mnemonics	Operands	Memory Address	Machine Code
	MVI	C, 00H	2000 H	0E, 00
	LXI	H, 2501 H	2002 H	21, 01, 25
	MOV	A, M	2005 H	7E
	INX	H	2006 H	23
	ADD	M	2007 H	86
	DAA		2008 H	27
	JNC	AHEAD	2009 H	D2, 0D, 20
	INR	C	200C H	0C
AHEAD:	STA	2503 H	200D H	32, 03, 25
	MOV	A, C	2010 H	79
	STA	2504 H	2011 H	32, 04, 25
	HLT		2014 H	76

DATA
2501 – 84 D
2502 – 75 D

Result
2503 – 59 D
2504 – 01

Subtraction of Two 8-bit Numbers:

Subtract 32 H from 49 H. 49 H – 32 H Method#1
The 1st number 49 H is in the memory location 2501 H.
The 2nd number 32 H is in the memory location 2502 H.
The result to be stored in the memory location 2503 H & 2504 H.

	Mnemonics	Operands	Memory Address	Machine Code
	MVI	C, 00H	2000 H	0E, 00
	LDA	2501 H	2002 H	3A, 01, 25
	MOV	B, A	2005 H	47
	LDA	2502 H	2006 H	3A, 02, 25
	SUB	B	2009 H	90
	JNC	AHEAD	200A H	D2, 0E, 20
	INR	C	200D H	0C
AHEAD:	STA	2503 H	200E H	32, 03, 25
	MOV	C, A	2011 H	79
	STA	2504 H	2012 H	32, 04, 25
	HLT		2015 H	76

DATA
2501 – 49 H
2502 – 32 H

Result
2503 – 17 H
2504 – 00 H

Method#2

	Mnemonics	Operands	Memory Address	Machine Code
	MVI	C, 00H	2000 H	0E, 00
	LXI	H, 2501 H	2002 H	21, 01, 25
	MOV	A, M	2005 H	7E
	INX	H	2006 H	23
	SUB	M	2007 H	96
	JNC	AHEAD	2008 H	D2, 0C, 20
	INR	C	200B H	0C
AHEAD:	STA	2503 H	200C H	32, 03, 25
	MOV	C, A	200F H	79
	STA	2504 H	2010 H	32, 04, 25
	HLT		2013 H	76

DATA
2501 – 32 H
2502 – 49 H

Result
2503 – F9 H
2504 – 01 H

Decimal Subtraction of Two 8-bit Numbers:

Subtract 38 D from 96 D. 96 – 38
The 1st number 96 D is in the memory location 2501 H.
The 2nd number 32 D is in the memory location 2502 H.
The result to be stored in the memory location 2503 H.

In decimal subtraction, the number which is to be subtracted is converted into 10's complement

Mnemonics	Operands	Memory Address	Machine Code
LXI	H, 2502 H	2000 H	21, 02, 25
MVI	A, 99 H	2003 H	3E, 99
SUB	M	2005 H	96
INR	A	2006 H	3C
DCX	H	2007 H	2B
ADD	M	2008 H	86
DAA		2009 H	27
STA	2503 H	200A H	32, 03, 25
HLT		200D H	76

DATA
2501 – 96 D
2502 – 38 D

Result
2503 – 58 D

Find One's complement of an 8-bit number

96 H = 1001 0110
One's complement = 0110 1001 = (69 H)
The number is placed in the memory location 2501 H
The result is stored in the memory location 2502 H

Mnemonics	Operands	Memory Address	Machine Code
LDA	2501 H	2000 H	3A, 01, 25
CMA		2003 H	2F
STA	2502 H	2004 H	32, 02, 25
HLT		2007 H	76

DATA
2501 – 96 H

Result
2502 – 69 H

Find Two's complement of an 8-bit number:

96 H = 1001 0110
One's complement = 0110 1001 = (69 H)
Add 01 in 1's complement, after addition = 6A H
The number is placed in the memory location 2501 H
The result is stored in the memory location 2502 H

Mnemonics	Operands	Memory Address	Machine Code
LDA	2501 H	2000 H	3A, 01, 25
CMA		2003 H	2F
INR	A	2004	3C
STA	2502 H	2005 H	32, 02, 25
HLT		2008 H	76

DATA
2501 – 96 H

Result
2502 – 6A H

Find One's complement of a 16-bit number:

5485 H = 0101 0100 1000 0101
One's complement = 1010 1011 0111 1010= (AB7A H)
The number is placed in the memory location 2501 H & 2502 H
The result is stored in the memory location 2503 H & 2504 H

Mnemonics	Operands	Memory Address	Machine Code
LXI	H, 2501 H	2000 H	21, 01, 25
MOV	A, M	2003 H	7E
CMA		2004 H	2F
STA	2503 H	2005 H	32, 03, 25
INX	H	2008 H	23
MOV	A, M	2009 H	7E
CMA		200A H	2F
STA	2504 H	200B H	32, 04, 25
HLT		200E H	76

DATA
2501 – 85 H
2502 – 54 H

Result
2503 – 7A H
2504 – AB H

UNIT-5 Interfacing and support chips

8255 Programable Peripheral Interface (PPI)

- PPI 8255 is a general purpose programmable I/O device designed to interface the CPU with its outside world such as ADC, DAC, keyboard etc.
- We can program it according to the given condition. It can be used with almost any microprocessor.
- It consists of three 8-bit bidirectional I/O ports (24 I/O lines) which can be configured as per the requirement.

Ports of 8255A

8255A has three ports, i.e., PORT A, PORT B, and PORT C.

- Port A (PA0-PA7) contains one 8-bit output latch/buffer and one 8-bit input buffer.
- Port B (PB0-PB7) is similar to PORT A.
- Port C can be split into two parts, i.e., PORT C lower (PC0-PC3) and PORT C upper (PC7-PC4) by the control word.

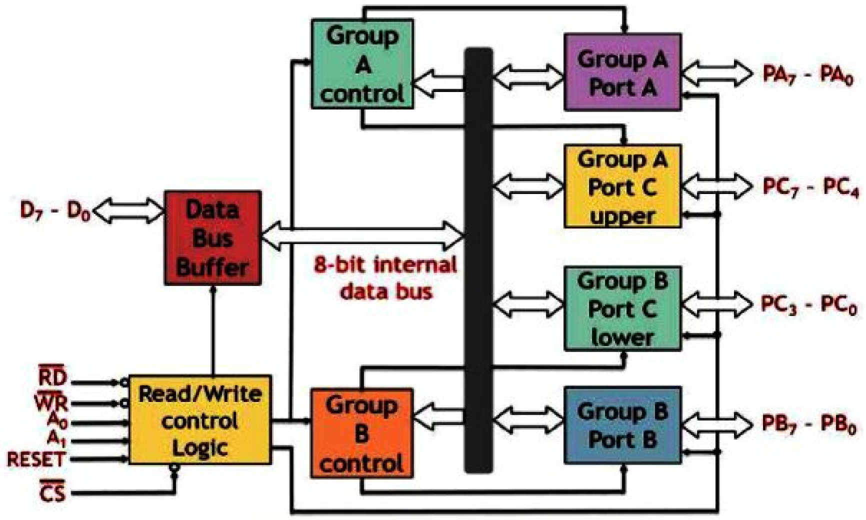
These three ports are further divided into two groups, i.e., Group A includes PORT A and upper PORT C. Group B includes PORT B and lower PORT C. These two groups can be programmed in three different modes, i.e., the first mode is named as mode 0, the second mode is named as Mode 1 and the third mode is named as Mode 2.

Features of 8255A

The prominent features of 8255A are as follows − □ It consists of 3 8-bit I/O ports i.e., PA, PB, and PC.

- Address/data bus must be externally demultiplexed.
- It is TTL compatible.
- It has improved DC driving capability.

Architecture of 8255 PPI:



Architecture of 8255 PPI

The figure above represents the architectural representation of 8255 PPI:

Let us understand the operation performed by each unit separately.

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:

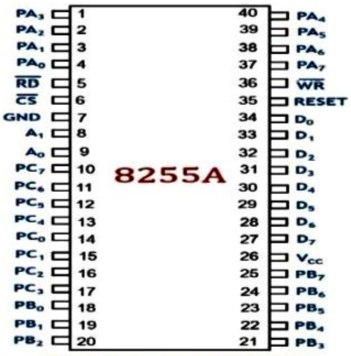
- This unit manages the internal operations of the system. This unit holds the ability to control the transfer of data and control or status words both internally and externally.
- Whenever there exists a need for data fetch then it accepts the address provided by the processor through the bus and immediately 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.
- **Group A** the access of the **port A** and higher order bits of **port C**. While **group B** controls port B with the lower order bits of **port C.**

Pin Diagram of 8255 PPI

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



Pin Diagram of 8255 PPI

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.

PA7-PA0:

These are eight **port A** lines that acts as either latched output or buffered input lines depending upon the control word loaded into the control word register.

PC7-PC4:

- Upper nibble of port C lines. They may act as either output latches or input buffers lines. This port also can be used for generation of handshake lines in mode 1 or mode 2.

PC3-PC0:

These are the lower port C lines, other details are the same as PC7-PC4 lines.

PB0-PB7:

These are the eight port B lines which are used as latched output lines or buffered input lines in the same way as port A.

D0-D7: These are the data bus lines those carry data or control word to/from the microprocessor.

VCC:

It is a supply voltage. The 8255 requires +5V supply to operate.

GND:

It is the ground reference. If there is excessive power supply then it passed to ground.

MODES OF OPERATION:

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

1. Bit set/reset mode
2. I/O mode

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 I/ O mode is sub-classified into 3 modes. So, let us now discuss the 3 modes here.

Mode 0: Input/output mode:

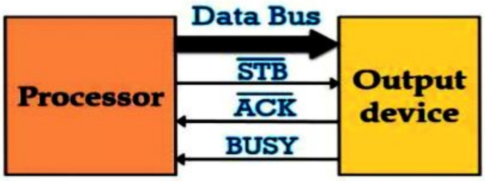
This mode is the simple input output mode of 8255 which allows the programming of each port as either input or output port. The input/output feature of mode 0 includes:

- 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.

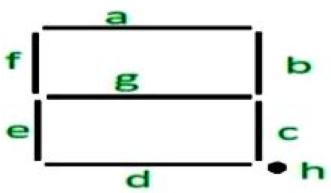
Mode 2: Bidirectional I/O port with handshaking:

- 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.

Seven segment LED display

A seven-segment LED is a kind of LED (Light Emitting Diode) consisting of 7 small LEDs it usually comes with the microprocessor's as we commonly need to interface them with microprocessors like 8085.

Structure of Seven Segments LED:



- It can be used to represent numbers from 0 to 8 with a decimal point.
- We have eight segments in a Seven Segment LED display consisting of 7 segments which include '.'.
- The seven segments are denoted as "a, b, c, d, e, f, g, h" respectively, and '.' is represented by "h".

Interfacing Seven Segment Display with 8085:

We will see a program to Interfacing Seven Segment Display with 8085 using 8255.

Note logic needed for activation –

Common Anode – 0 will make an LED glow.

Common Cathode – 1 will make an LED glow.

Common Anode Method:

Here we are using a common anode display therefore 0 logic is needed to activate the segment. Suppose to display number 9 at the seven-segment display, therefore the segments F, G, B, A, C, and D have to be activated.

The instructions to execute it is given as,

```
      MVI       A,99
OUT 00
```

- First, we are storing the 99H in the accumulator i.e., 10010000 by using MVI instruction.
- By OUT instruction we are sending the data stored in the accumulator to the port 00H.

Common Cathode Method:

Here we are using common cathode 1 logic is needed to activate the signal. Suppose to display number 9 at the seven-segment display, therefore the segments F, G, B, A, C, and D have to be activated.

The instructions to execute it is given as,

```
       MVI       A,6F
OUT 00
```

- First, we are storing the 6FH in the accumulator i.e., 01101111 by using MVI instruction.
- By OUT instruction we are sending the data stored in the accumulator to the port 00H.

Traffic light controller

The traffic lights are interfaced to Microprocessor system through buffer and ports of programmable peripheral Interface 8255. So the traffic lights can be automatically switched ON/OFF in desired sequence. The Interface board has been designed to work with parallel port of Microprocessor system.

Working Program

Design of a microprocessor system to control traffic lights. The traffic should be controlled in the following manner.

1) Allow traffic from W to E and E to W transition for 20 seconds.
2) Give transition period of 5 seconds (Yellow bulbs ON)
3) Allow traffic from N to 5 and 5 to N for 20 seconds
4) Give transition period of 5 seconds (Yellow bulbs ON) 5) Repeat the process.

Source Program:

```
          MVI A, 80H:          Initialize 8255, port A and port B
           OUT 83H (CR):       in output mode
START: MVI A, 09H
          OUT    80H    (PA):  Send data on PA to glow R1 and R2
MVI A, 24H
          OUT 81H (PB):                  Send data on PB to glow G3 and G4
          MVI C, 28H:          Load multiplier count (40ıo) for delay
          CALL DELAY:                    Call delay subroutine
          MVI A, 12H

         OUT (81H) PA:         Send data on Port A to glow Y1 and Y2
         OUT (81H) PB:         Send data on port B to glow Y3 and Y4
         MVI C, 0AH:           Load multiplier count (10ıo) for delay
         CALL: DELAY:                    Call delay subroutine
         MVI A, 24H
         OUT (80H) PA:  MVI    Send data on port A to glow G1 and G2
         A, 09H
         OUT (81H) PB:         Send data on port B to glow R3 and R4
         MVI C, 28H:           Load multiplier count (40ıo) for delay
         CALL DELAY:  MVI                Call delay subroutine
         A, 12H
         OUT PA:               Send data on port A to glow Y1 and Y2
         OUT PB:               Send data on port B to glow Y3 and Y4
         MVI C, 0AH:           Load multiplier count (10ıo) for delay
         CALL DELAY:                     Call delay subroutine
         JMP START
Delay Subroutine:
DELAY:  LXI D, Count:                     Load count to give 0.5 sec delay
BACK:  DCX D:   MOV A, D                 Decrement counter

         ORA E:                          Check whether count is 0
         JNZ BACK:             If not zero, repeat
         DCR C:                Check if multiplier zero, otherwise repeat
         JNZ DELAY
         RET:                  Return to main program
```

Square wave generator

- With 00H as i/p to DAC, analog o/p is -5V, and with FFH as i/p, analog o/p is +5V.
- I/P 00H and FFH at regular intervals generate square wave. □ The frequency can be varied by varying the time delay.

Algorithm

Initialize the control word of 8255 to operate in I/O mode for port A and B & C to operate in o/p mode.

Program

```
        MVI A,80
        OUT CWR      initialize the control word
LOOP: MVI A,00

        OUT PA
        CALL DELAY
        MVI A,FF
        OUT PA
        CALL DELAY
        JMP LOOP
DELAY: MVI C,85
BACK: DCR C
        JNZ BACK
        RET
```