



## Exp. No. (1)

### Logic Gate and Boolean Algebra

#### Object

To study the logical function and to get familiar with Boolean algebra.

#### Theory

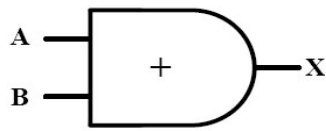
A number system is nothing more than a code representing quantity. For each circuit quantity, there is an assigned symbol for the quantity. After we memorize the code we can count and this lead to arithmetic and higher form of mathematics. So, the binary number system is a code that uses only two basic symbols. These symbols can be any two distinct characters like A and B or the customary 0 and 1.

Gate:- In logic circuit, this is a device with one output and two or more inputs, designed in such a way that there is an output for certain combinations of input signals.

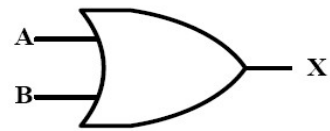
In digital logic there are three basic elements:

1. The OR gate: It is a device whose output is a logic (1) if either or both inputs are a logic (1). The OR gate is shown by the symbol in Fig.(1) with two inputs (A and B).





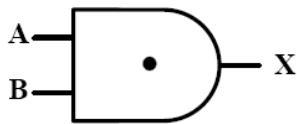
$$X = A + B$$



$$X = A + B$$

**Fig. (1)**  
**The OR Gate.**

2. The AND gate: It is a device whose output is a logic (1) if both of it's inputs are logic (1). This gate is shown by the symbol in Fig(2) with two inputs marked A and B and one output marked X.



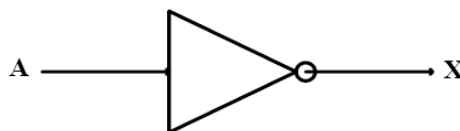
$$X = A \cdot B$$



$$X = A \cdot B$$

**Fig. (2)**  
**The AND Gate.**

3. The Not gate (Inverter): Is another basic kind of a digital circuit, else called a complementing circuit which is most simple element of digital logic. The inverter is different from AND and OR gates, in that it has only a single input. As a result, it dose not perform any decision making function that is dependent on a combination of inputs. Instead the inverter simply convert logic (1) at it's input to logic (0) at it's output and conversely, a logic (0) to logic (1). The inverter can be represented by the symbol shown in Fig. (3).



$$X = \bar{A}$$

**Fig. (3)**  
**The NOT Gate.**



***Notes:***

- The AND function may not be constructed of OR gates only.
- The OR function may not be constructed of AND gates only.
- The NOT function may not be constructed of AND, OR gates.

Two types of logic gates found often in digital circuit are the NAND (NOTAND) and NOR (NOT-OR). As their name imply, a NAND gate is an inverted AND function, each LOW (0) output of AND function is made HIGH (1). And NOR gate is equivalent of an inverted OR functions and will yield logic (1) output if both inputs are logic (0). NAND and NOR gates are shown in Fig. (4).

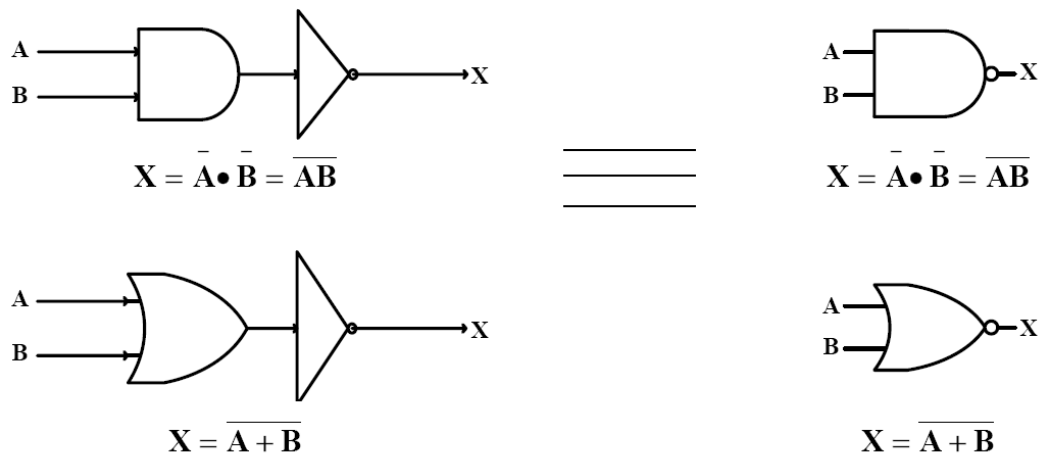


Fig. (4)  
A. NAND Gate.  
B. NOR Gate.

**Boolean algebra:** is a name given to mean of mathematically manipulation of Boolean expression for the purpose of putting them in their simplest form.

There are three law of Boolean algebra:

1. The commutative law: States that, when inputs to a logic symbol are AND ed or OR ed, the order in which they are written doesn't affect the answer results, therefore:

$$\mathbf{AB = BA \quad A + B = B + A}$$

2. The associative law: State that, the grouping of AND ed function as they written doesn't affect the value of the output, nor dose the grouping of OR ed function, therefore:

$$\mathbf{(AB) C = A (BC) \quad (A + B) + C = A + (B + C)}$$

3. Distributive law: Is a law provides a powerful tool for manipulation of Boolean expressions. It states that Boolean expression, as in (Conventional) algebra can be (Factored) or simplified through (expression), there are:

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

**Identities**

$$A + 1 = 1$$

$$A + A = A$$

$$A \cdot 0 = 0$$

$$A + 0 = A$$

$$A + \bar{A} = 1$$

$$A \cdot A = A$$

$$\bar{\bar{A}} = A$$

$$A \cdot 1 = A$$

$$A \cdot \bar{A} = 0$$

**Truth Table**

A truth table is a list showing all input-output combination of logic circuit of a logic circuit. The number of horizontal rows in table equals  $(2^n)$ , where "n" is the number of input variable to the logic circuit. To include all possible input combination, we normally list the combination in a binary number progression.

**Example (1):**

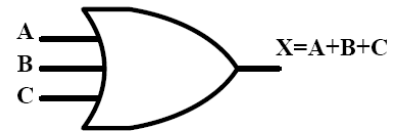
Describe the truth table for a two-input OR gate and a three-input OR gate.



Two input OR gate.  
n = 2

Therefore the horizontal rows =  
 $2^2 = 4$

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



Three input OR gate.  
n = 3

Therefore the horizontal rows =  
 $2^3 = 8$

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

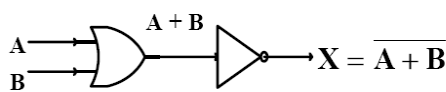
## De Morgan's Theorems

Much of today logic implementation is based upon a set of rules called De Morgan's Theorems. Basically these theorems demonstrate that, any logic function can be formed from AND-NOT gates (NAND) or from OR-NOT gates (NOR).

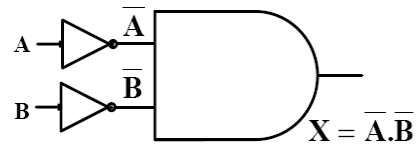
These rules may be summarized in the state:

1. The complement of a sum equals the product of the complements.

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



A



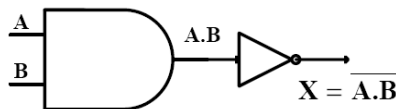
B

Fig. (5)

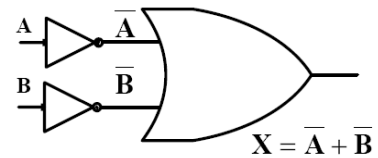
1<sup>st</sup> Law of De Morgan's Theorems

2. The complement of the product equals the sum of the complements.

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



A



B

Fig. (6)

2<sup>nd</sup> Law of De Morgan's Theorems



## Procedure

1. Connect the circuit shown in Fig. (1).
2. Using different states of the inputs, find the truth table of the circuit.
3. Connect the circuit in the figures and for each connection repeat step (2).
  - a) Fig. (2), (3).
  - b) Fig. (4 A and B).
  - c) Fig. (5 A and B).
  - d) Fig. (6 A and B).
  - e) Fig. (7 A and B) and write the Boolean expression for the output.
4. Draw and implement the logic circuit for  $X = \bar{A}B + A\bar{B}$  using basic logic gates. Then describe its truth table, and try to draw the logic circuit for the same function using NAND gate only.

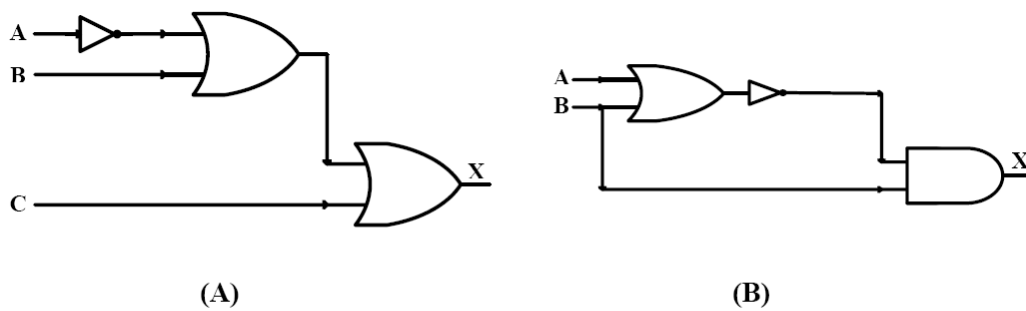


Fig (7)

## Discussion

1. Prepare the implementation of three inputs AND gate by using NOR gate only. Then show how NAND gate can be used for build the logic circuit for  $X = A + BC$ .
2. Can you suggest why all logic function in a digital system are often generated with single gate type.
3. Draw the logical circuit (F) by using only NAND gate  $F = AB + D(B + C)$ , and describe it's truth table.
4. Prove:

.....

1.  $A + \overline{AB} = A + B$ .

2.  $(A + B)(\overline{A} + C) = AC + \overline{AB} + BC$ .

3.  $\overline{A.B.C.D} = \overline{A} + \overline{B} + \overline{C} + \overline{D}$ .

4.  $ABC + \overline{A}BC + A\overline{B}C = A(B + C)$ .

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

6.  $\overline{A}C + ABC + A\overline{C}D + CD = A + CD$ .





## Exp. No. (2)

### Exclusive OR Gate and it's Applications

#### Object

To study the logic function of exclusive OR (XOR) gate, and become familiar with some of it's applications.

#### Theory

The output of XOR gate, is logic (1) when both inputs are different, and is logic (0) when inputs are the same, Fig. (1) gives the symbol and truth table for this gate.

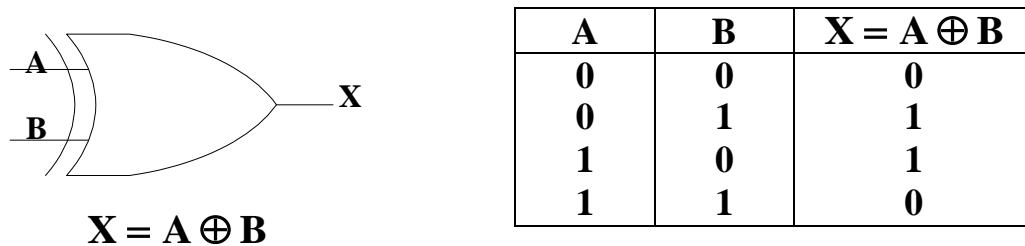
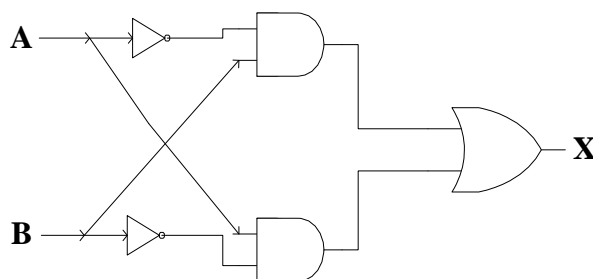


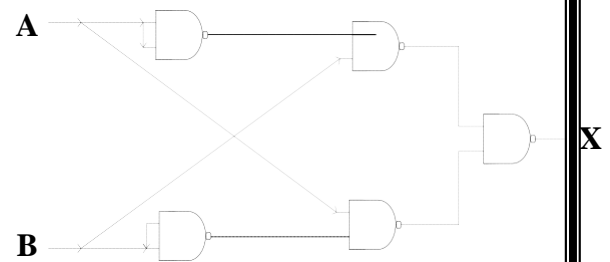
Fig. (1)

#### Exclusive OR gate with it's Truth Table

Algebraically XOR output can be written as  $X = \bar{A}B + A\bar{B}$ , and can be implement as in Fig. (2):



(A) Different Gates.



(B) NAND Gates only.

Fig. (2)

#### Implementation of XOR gate

## Applications of XOR gate

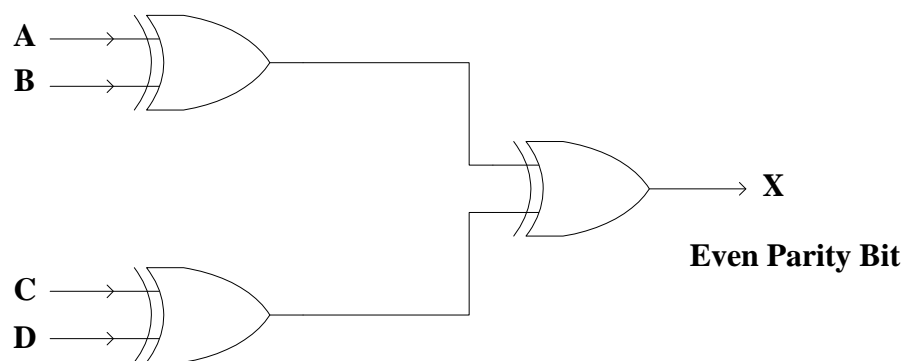
There are many applications for XOR gate such as:

### **1. Arithmetic Operations:**

The XOR gate also called (Modulo Two Adder) , since it is used to give the sum of two binary numbers, it has been used in many arithmetic circuits (it will be explained in latter experiments).

### **2. Parity Checker:**

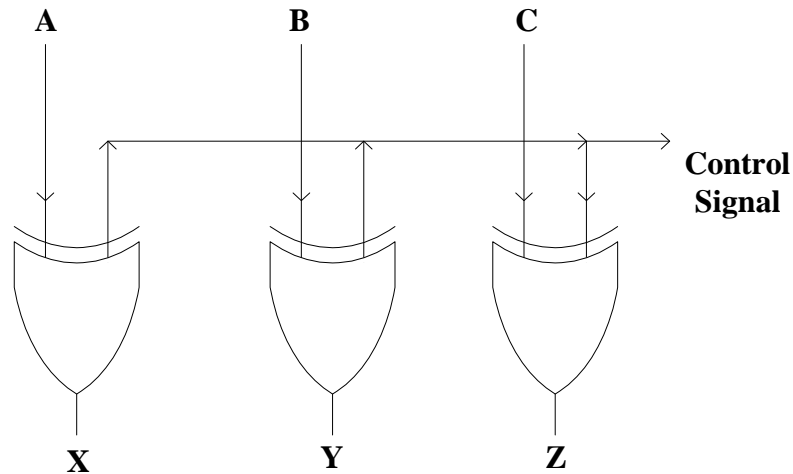
One of the advantages of using digital system, is it's capability of detecting and correction errors. This is used specially when digital information is transmitted or stored. One of the simplest form of error detecting is the parity checker, assume that, we have four bit word, to detect the occurrence of an odd number of errors in this word, a single bit will be added to the word that makes the number of "ones" in the word either even number "Even Parity" or odd number "Odd Parity", so, if an odd number of error occurred in the word then the total number of ones will not remain the same, it will change from odd to even or from even to odd. The XOR gate is the most suitable circuit to provide parity checker. Fig. (2) given the circuit of four bit even parity checker, to have an odd parity checker, we need to complement the output.



**Fig. (3)**  
**Four Bit Even Parity Checker.**

### 3. *Controlled Inverter:*

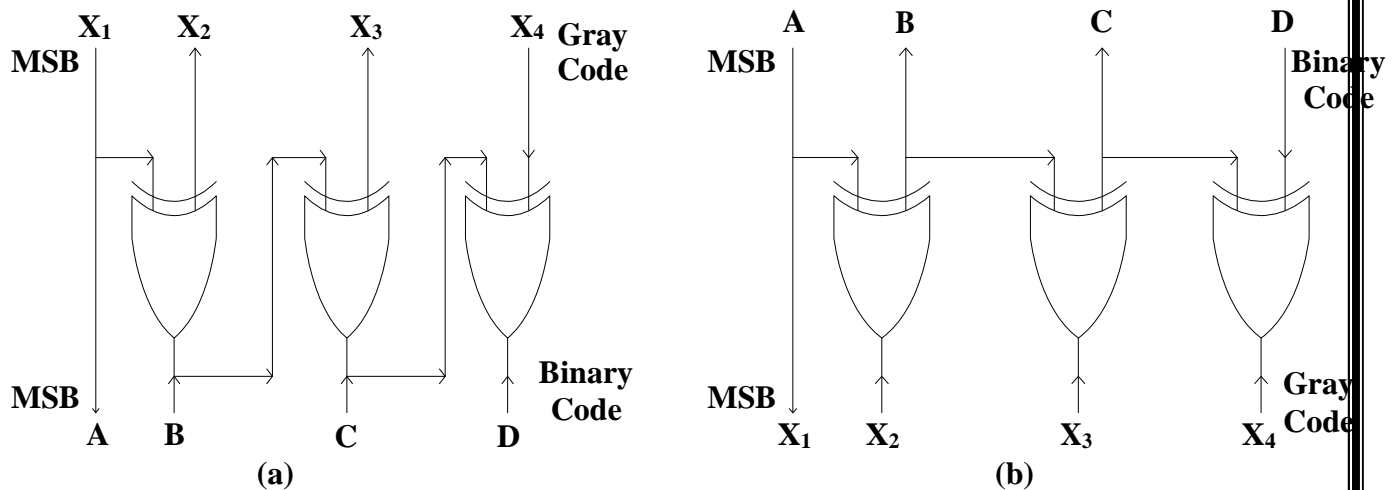
The XOR gate can be used as a "NOT" gate by connecting one of the inputs to the logic (1), for this reason it can be used to complement a word by using one of the inputs as control line, as shown in Fig. (4), when control signal is logic (0) then,  $X = A$ ;  $Y = B$ ;  $Z = C$ . When control signal is logic (1) then,  $X = \bar{A}$ ;  $Y = \bar{B}$ ;  $Z = \bar{C}$



**Fig. (4)**  
**Controlled Inverter.**

### 4. *Binary to Gray / Gray to Binary Conversion:*

The gray code is widely used in many digital systems, specially in shaft register encoders and analog to digital conversion, but it is difficult to use the gray-code in arithmetic operations, since there are only one bit change between two consecutive gray code number, and it is unweighted code, and the XOR gate is the most suitable gate for this purpose as shown in Fig. (5).



**Fig. (5) (a). Gray to Binary (b). Binary to Gray**

### 5. Combinational Logic Circuit Minimization:

Another useful application for XOR gate is, its use in minimizing combinational circuit which will be dealt with in detail in other experiment.

### 6. Digital Comparator:

Many practical applications require the comparator of two numbers A & B searching for either (a) equality or (b) non equality.

If equality is what we looking for, this means  $A = B$ , then the output of the logic network ( $Z_1$ ) is expressed by the function concluded from the truth table below (Table 1).

$$Z_1 = AB + \bar{A}\bar{B}$$

When non-equality is what we require, this means  $A \neq B$ , then we can concluded the Boolean expression for ( $Z_2$ ) from the truth table (Table 1).

$$Z_2 = A\bar{B} + \bar{A}B$$

If the comparison is such that the states of one number with respect to the other is to be specified one of the three conditions  $A > B$ ,  $A < B$ , or  $A \equiv B$  should be known the simple Boolean expressions are:

$$\bar{A}\bar{B} \text{ for } A > B.$$

$$\bar{A}B \text{ for } A < B.$$

$$\& AB + \bar{A}\bar{B} \text{ for } A \equiv B.$$

A	B	$A \oplus B$	$AB + \bar{A}\bar{B}$ ( $Z_1$ )	$\bar{A}B + A\bar{B}$ ( $Z_2$ )	$A > B$	$A < B$
0	0	0	1	0	0	0
0	1	1	0	1	0	1
1	0	1	0	1	1	0
1	1	0	1	0	0	0

*Table (1)*

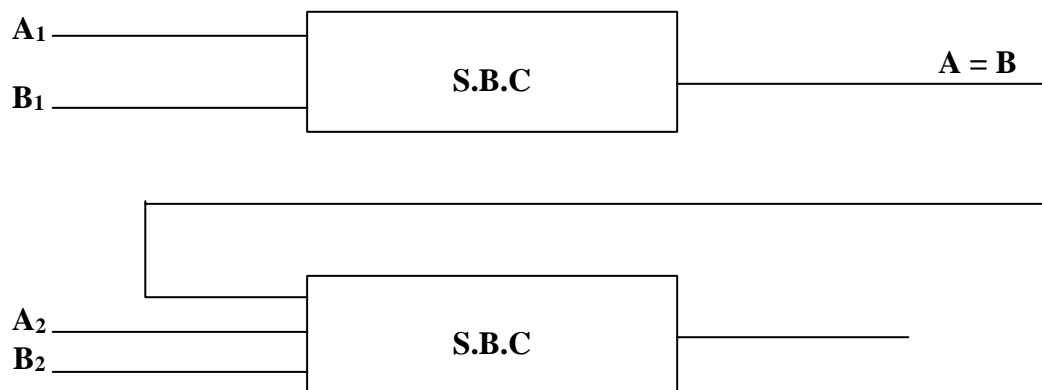
The comparison for two number of more than one binary bit may be summarized by the following steps:

Suppose the two numbers  $X = A_1 A_2$

$Y = B_1 B_2$

1.  $X = Y$  when  $A_1 = B_1$  and  $A_2 = B_2$ .
2.  $X > Y$  If  $A_1 > B_1$  or  $A_1 = B_1$  and  $A_2 > B_2$ .
3.  $X < Y$  If  $A_1 < B_1$  or  $A_1 = B_1$  and  $A_2 < B_2$ .

It is clear that to search for any of the three conditions it is only needed to found the equality condition for each higher bit to the next lower bit and so on. The way that equality condition is added to the single bit combater is shown:



**Fig. (6)**  
**Block diagram of Two-Two Bit Comparator.**



## **Procedure**

1. Connect the circuit shown in Fig. (1) and Fig. (2-A) and find the truth table for the circuit.
2. Connect the circuit in Fig. (3), find the even parity bit for the numbers -----  
---,-----,----- and -----, then find the odd parity bit for the numbers -----,-----,----- and -----.
3. Connect a circuit that convert four bit word using XOR gate, find one's complement for the numbers -----,-----,----- and -----.
4. Connect 5 bit gray to binary converter circuit and find the equivalent binary for the following gray code -----,-----,----- and -----, then connect 3 bit binary to gray code converter circuit and find the equivalent gray code for these binary numbers -----,------,----- and -----.
5. Referring to the truth table shown in Table (1) shown in the theory implement a circuit for single bit comparator.

## **Discussion**

1. Draw the circuit diagram for four input XOR gate. Explain the circuit.
2. Draw a circuit diagram that detect errors is an odd parity 5 bit word.
3. Can you add more applications for XOR gates.
4. Write down the equivalent gray code for the numbers 0-15.
5. Design a logic circuit that can compare between two-two bit binary number. Draw the circuit and find it's truth table.

Suggest some application where comparator is useful



## Exp. No. (3)

### Binary Additional (Half and Full Adders)

#### Object

To be able to design a circuit to add two or three binary digits.

#### Theory

##### 1. The Half Adder:

Computer addition is performed using certain rules which are similar to those used in adding decimal numbers. Started for the binary system, these rules are as following:-

Rule (1)	$0+0$	$=$	0
Rule (2)	$0+1$	$=$	1
Rule (3)	$1+1$	$=$	0 (With Carry)

A truth table for the rules of binary addition is shown in Fig. (1). A and B represent two-one bit binary numbers to be added. As the table shows, a sum will be produced when A is one or B one, but not when both bits are one. This can be stated in Boolean terms as  $\overline{A}B + A\overline{B}$  which, you may recall, is the output expression for the Exclusive OR (XOR) network. The expression for the carry output, derived from the table, is (AB). Which may capable of performing the addition of two-one bit number.

## 2. The Full Adder:

In computer operation, the half-adder is not sufficient for adding more than two –one numbers. Adding binary numbers that have more than two bit position required the useful of Full Adder, which is capable of adding the two bits of each position. For example, adding 0101 and 0101:

$$\begin{array}{rcccc}
 & 1 \leftarrow & & 0 \leftarrow & & 1 \leftarrow & & & \\
 & & & & & & & & \\
 0 & | & 1 & | & 0 & | & 1 & & \\
 0 & & 1 & & 0 & & 1 & & \\
 \hline
 1 & & 0 & & 1 & & 0 & = \text{Sum} \\
 0 & \text{---} & 1 & \text{---} & 0 & \text{---} & 1 & = \text{Carry-Out}
 \end{array}$$

The least significant bits of the two binary numbers are added. In this case, "1+1 = 0" with carry. A zero comes out to the adder as the least significant bit of the sum, and the "1" carry-out produced by this addition is stored. When the next addition is performed, the carry-out resulting from the first addition is fed into the adder along with two (0) bits. The addition of these two bits and the carry-in (carry-out from the first addition) is interpreted as "0+0+1 = 1". This equals (1) with no-carry. Therefore, the second addition produces a sum bit of (1) and no carry-out for the next addition. The third addition, in this case "1+1+0 = 0", equals (0) with carry. The carry is again stored, becoming a carry-in for the final addition. When the final addition is performed, then, "0+0+1 = 1" produced (1) with no carry. Table (2) shows all possible combinations for producing the sum and carry outputs, in truth table form.

Column A & B represented the two bits about to be added. The carry-in column shown the carry-out which will be produced by the pervious addition. The last two column show the sum and carry-out which will be produced by adding the content of the first three columns. The condition required to produce

a (1) in the SUM column of the truth table are defined by the Boolean expression:

$$\text{SUM} = \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC$$

By factoring "C" out of the first and last terms, the expression becomes:

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

Next, by factoring " $\bar{C}$ " out of two middle terms of the original expression, we get:

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

$$\text{SUM} = C(\overline{A \oplus B}) + \bar{C}(A \oplus B)$$

$$\text{SUM} = A \oplus B \oplus C$$

Similarly, the condition necessary to produced a "1" in the carry-out column of the truth table can be defined by the expression:

$$\text{CARRY-OUT} = ABC\bar{C} + A\bar{B}C + ABC + \bar{A}BC$$

The carry-out expression can be simplified as shown below:

$$\text{CARRY-OUT} = ABC\bar{C} + A\bar{B}C + ABC + \bar{A}BC$$

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

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

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

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

$$= ABC\bar{C} + AC + BC$$

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

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

$$= AB + AC + BC$$

It is more common practice to construct a full adder using two-half adder, and OR gate. The basic technique is shown in Fig. (1).

## **Procedure**

1. Referring to the truth table (1) shown in the theory, implement a half adder circuit.
2. Referring to the truth table (2) shown in the theory, implement a full adder circuit.
3. What modification to the adder circuit are necessary to be able to perform binary half-subtract?

Connect this circuit and find its truth table.

4. Design a full subtract circuit and find its truth table.

## **Discussion**

1. Explain how can you implement the half and full adder without using XOR gates, show the logical circuit and compare with one you have used.
2. With the aid of logic circuit, design an adder to add two-three bit binary numbers.
3. Discuss the methods used to speed up addition.
4. Design a half adder/subtract circuit. Using control line (Z), when  $Z = 1$  we get half adder, and when  $Z = 0$  we get half subtract.

A	B	SUM	CARRY OUT
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

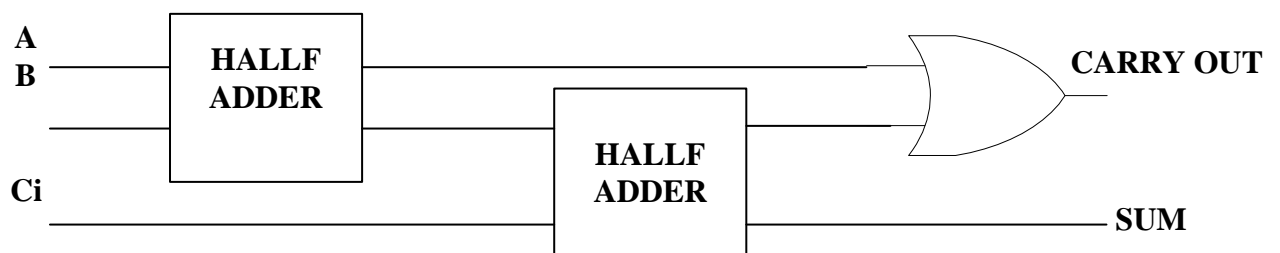
**Table (1)**  
**Truth Table of Half Adder.**

A	B	CARRY IN (Ci)	SUM	CARRY OUT (Co)
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$\text{SUM} = A \oplus B \oplus C$$

$$\text{CARRY OUT} = AB + AC + BC$$

**Table (2)**  
**Truth Table of Full Adder**



**Fig. (1)**  
**Full Adder Using Two**  
**Half Adder.**



## Exp. No. (4)

### Decoders and Encoders

#### Object

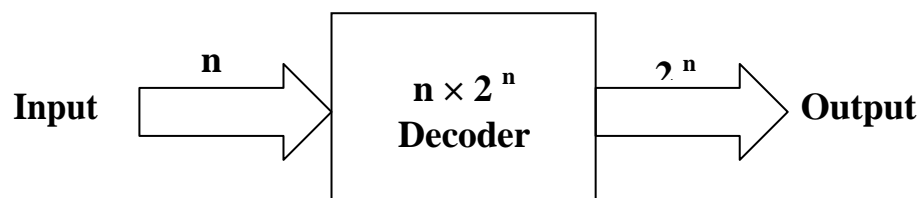
To be familiar with basics of conversion from binary to decimal by using decoder networks.

#### Theory

##### 1. Decoder

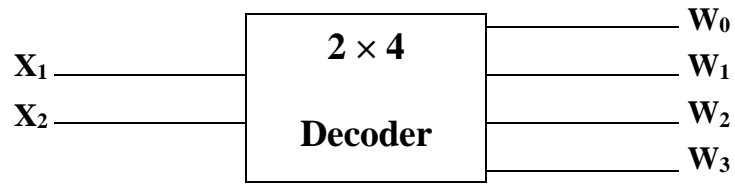
The process of taking some type of code and determining what it represents in terms of a recognizable number or character is called decoding. A decoder is a combinational logic circuit that performs the decoding function, and produce an output that indicates the (meaning) of the input code.

The decoder is an important part of the system which selects the cells to be read from and write into. This particular circuit is called a decoder matrix, or simply a decoder, and has a characteristic that for each of the possible  $2^n$  binary input number which can be taken by the  $n$  input cells, the matrix will have a unique one of its  $2^n$  output lines selected.



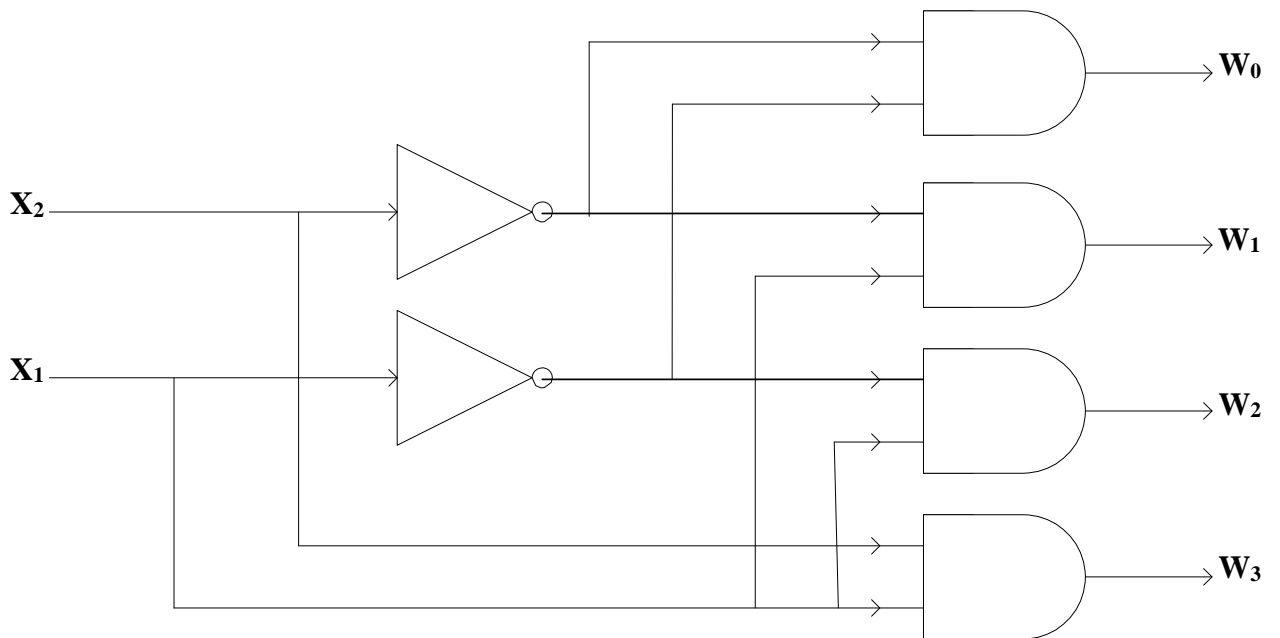


The decoder is called n to m where  $m < 2^n$  for example two to four line decoder, Fig. (1) shows a two to four line decoder and its truth table.



INPUTS		OUTPUTS			
$X_2$	$X_1$	$W_0$	$W_1$	$W_2$	$W_3$
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

**Table (1)**  
**Two to Four Decoder Truth Table**



**Fig. (1)**  
**Two to Four Line Decoder**

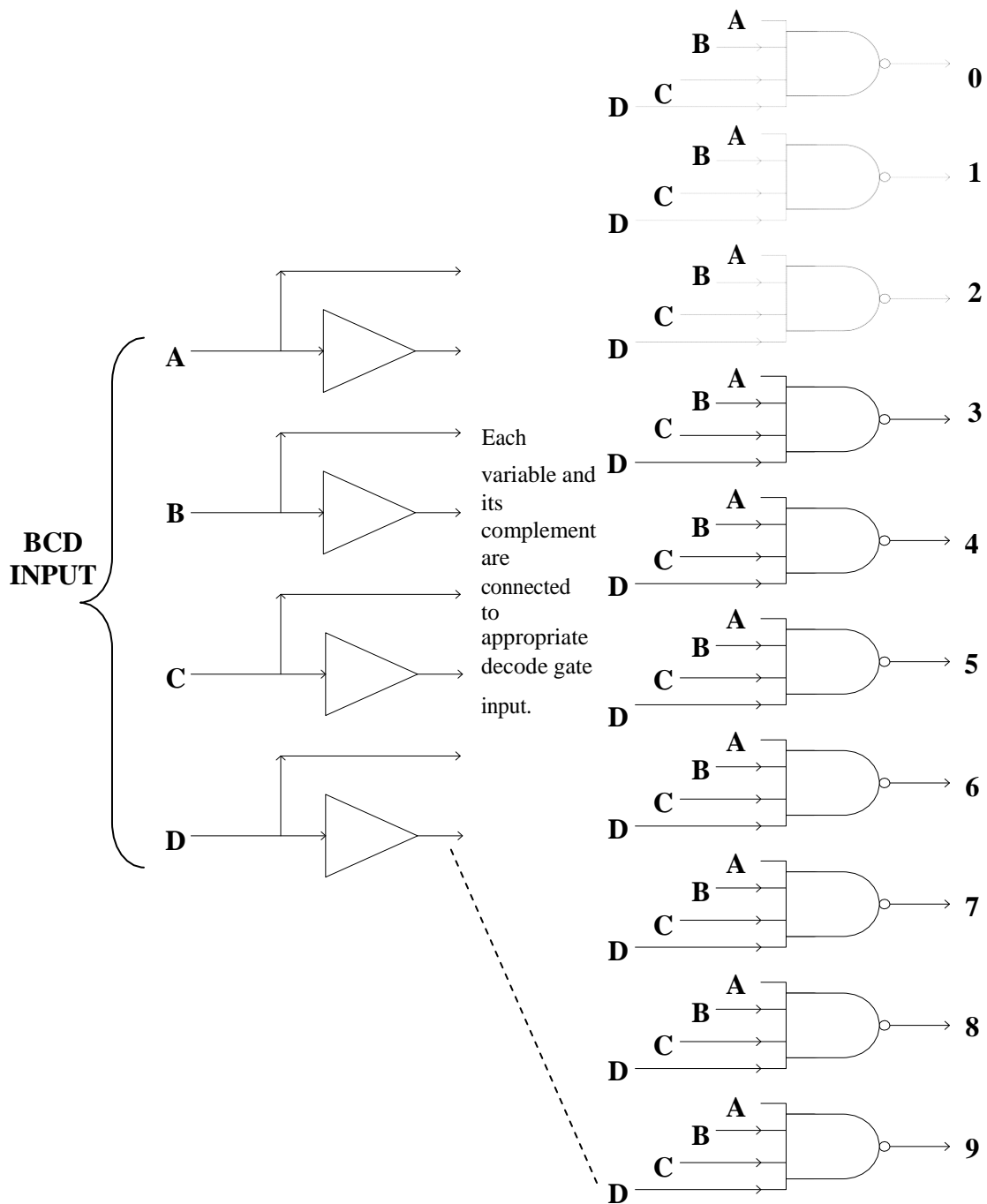
### The BCD Decoder

The BCD decoder converts each BCD code (8421) into one of ten possible decimal digit indications. It is typically referred to as a 1 of 10 or 4 to 10 lines decoder, although other types of decoder also fall into this category (such as an ExeCs – 3 decoder). A list of the ten BCD code words and their corresponding decoding functions is shown in Table (2). Each of these decoding functions is implemented with NAND gates to provide active LOW outputs, as shown in Fig. (2).

DECIMAL DIGIT	OUTPUTS										LOGIC FUNCTION (X)
	0	1	2	3	4	5	6	7	8	9	
0	0	1	1	1	1	1	1	1	1	1	$\overline{D}\overline{C}\overline{B}A$
1	1	0	1	1	1	1	1	1	1	1	$\overline{D}\overline{C}BA$
2	1	1	0	1	1	1	1	1	1	1	$\overline{D}C\overline{B}A$
3	1	1	1	0	1	1	1	1	1	1	$\overline{D}CBA$
4	1	1	1	1	0	1	1	1	1	1	$\overline{D}C\overline{B}A$
5	1	1	1	1	1	0	1	1	1	1	$\overline{D}CBA$
6	1	1	1	1	1	1	0	1	1	1	$\overline{D}CBA$
7	1	1	1	1	1	1	1	0	1	1	$\overline{D}CBA$
8	1	1	1	1	1	1	1	1	0	1	$\overline{D}C\overline{B}A$
9	1	1	1	1	1	1	1	1	1	0	$\overline{D}CBA$

**Table (2)**  
**Truth Table of BDC to Decimal Decoder**

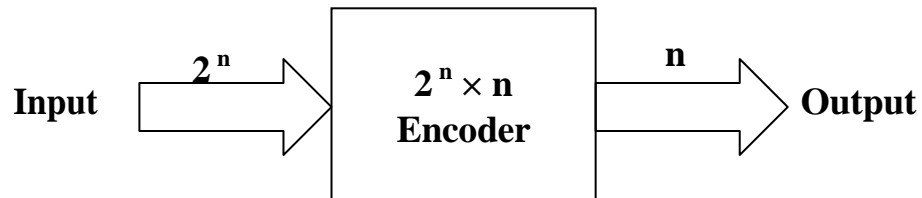
The 7442 is an integrated circuit BCD to Decimal decoder. Note that on this device the inputs are A, B, C, and D where A is the least significant bit.



**Fig (2)**  
**Logic for BCD Decoder.**

## 2. Encoder

An encoder is a combinational logic circuit that generate  $n$  output lines from  $2^n$  (or less) inputs. It has the reverse function of the decoder.

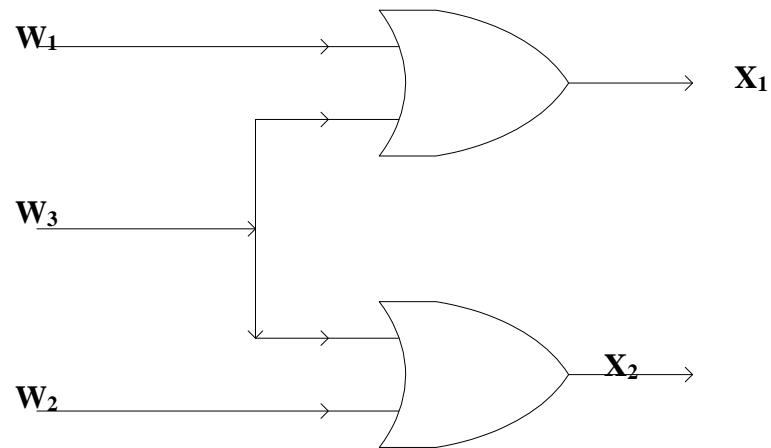


An encoder accepts digit on its inputs, such as a decimal or octal digit, and converts it to a coded output, such as a binary or BCD. Encoder can also be devised to encode various symbol and alphabetic characters. This process of converting from familiar symbols or numbers to a coded format is called encoding.

Figure (2) shown a four to two line encoder and its truth table.

INPUTS				OUTPUTS	
$W_3$	$W_2$	$W_1$	$W_0$	$X_2$	$X_1$
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

**Table (3)**  
**Truth Table of Four to Two Line Encoder.**



**Fig. (3)**  
**Four to Two line Encoder**

### **Procedure**

1. Construct a circuit as shown in Fig. (1), set data switches as shown in the two to four lines decoder output table. Record the output indications of  $L_1$  to  $L_4$ .
2. Install one 7442 BCD to Decimal Decoder in the logic lab. breadboard. Set data switches as shown in the BCD to Decimal Decoder output table in Fig. (2). Record the output indications output pins.
3. Construct the circuit as shown in Fig. (3), set data switches as shown in the four to two line encoder truth table. Record the output indications of  $L_1$  &  $L_2$ .

### **Discussion**

1. Design a full adder circuit using decoder.
2. Design  $3 \times 8$  decoder from  $2 \times 4$  decoder.
3. Design  $4 \times 16$  decoder from  $3 \times 8$  decoder.
4. Design octal to binary encoder.



## Exp. No. (5)

### Multiplexer and Demultiplexer

#### Object

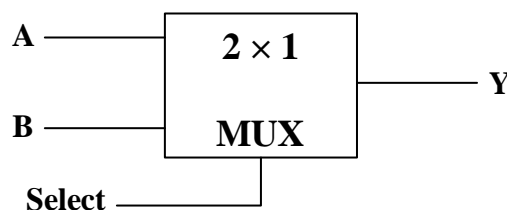
To demonstrate a basic Multiplexer / Demultiplexer system, and become familiar with different types of multiplexer and demultiplexer.

#### Theory

##### *1. Multiplexer*

It is not necessary to use only discrete gates (AND, OR, NAND, NOR, EXOR, EXNOR) in the design of the combinational logic circuit, with the availability of the medium scale integrated (MSI) and large scale integrated (LSI), it is possible to design a very complicated circuits with a simple procedure, for example it is waste of time in most cases to try to minimize combinational logic circuit which has eight input using tabular method, while it will simpler if we used multiplexers.

A multiplexer is a network that has many inputs and one output, and the value of the output will be the value of one of inputs which will be decided by some select lines. The simplest type of multiplexer is the two line to one line data multiplexer. Let A be one of the inputs and B is the other input and Y is the output as in Fig. (1), and S is the select line, then

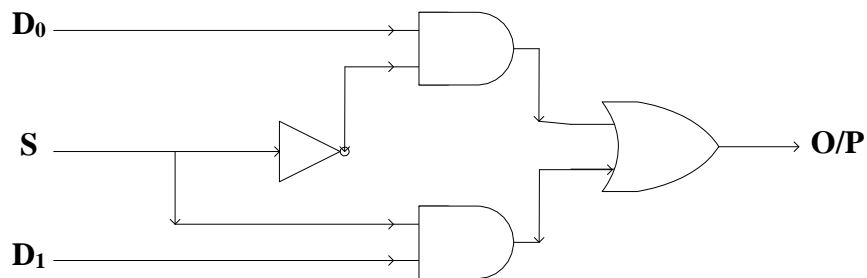


**Fig. (1)**  
**Two to One Line Multiplexer**

$Y = A$  if Select = 0.

$Y = B$  if Select = 1.

The logic circuit diagram of the Two to One line Multiplexer is shown in Fig. (2).



**Fig. (2)**  
**Logic Circuit of Two to One Line Multiplexer.**

There are many 2 to 1 data selectors as a MSI, for example (7498, 74157, 74158) which contains four (quadruple) two-to-one data selectors in one chip.

There are other types of multiplexers 4-to-1 line, 8-to-1 line, and 16-to-1 line multiplexer, and the number of select lines of these multiplexer are 2, 3, and 4 lines respectively. Fig.(3) shows the four to one line multiplexer and its function block diagram.

To use the multiplexer in the design of combinational logic circuit, usually the truth table of K-map of function is used in which the table or the map is divided into 2, 4, 8, or 16 equal parts according to the type of multiplexer used. Some of the inputs of the combinational circuit is connected directly to the select lines while data lines of the multiplexer will be a function to the other inputs according to the sun map or sub tables.



**Example:**

Design the following expression using multiplexer.

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

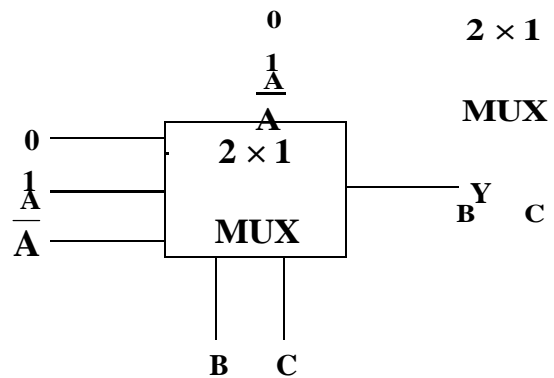
**Solution:**

Number of variables = 3, it is better to use 4-to-1 line multiplexer, i.e.:

Number of selection lines = Number of variable - 1.

The truth table of the function is shown below:

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

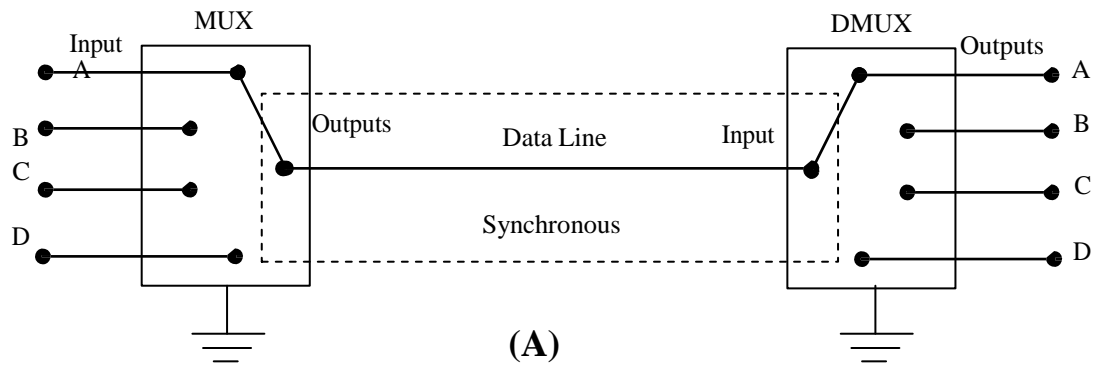


**Fig. (4)**

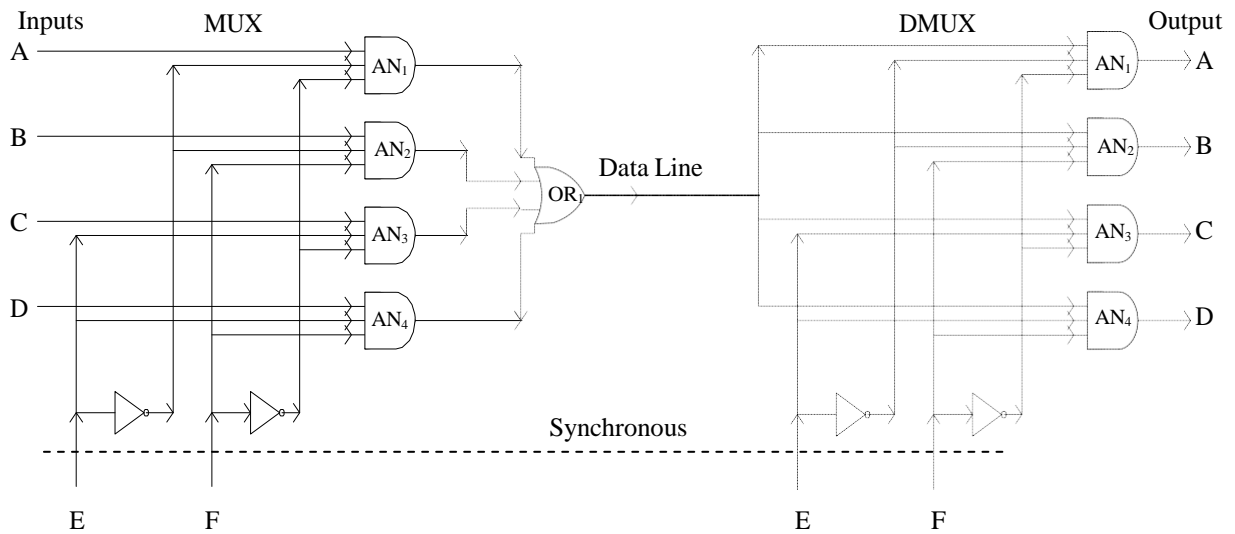
**2. Demultiplexer**

A demultiplexer basically reverses the multiplexing function. It is take data from one line and distribute them to given number of output lines. Fig. (3) shown a one to four line demultiplexer circuit. The input data line goes to all of the AND gates. The two select lines enable only one gate at a time and the data appearing on the input line will pas through the selected gate to the associated output line.

The simplest type of demultiplexer is the one to two lines DMUX. as shown in Fig. (5).



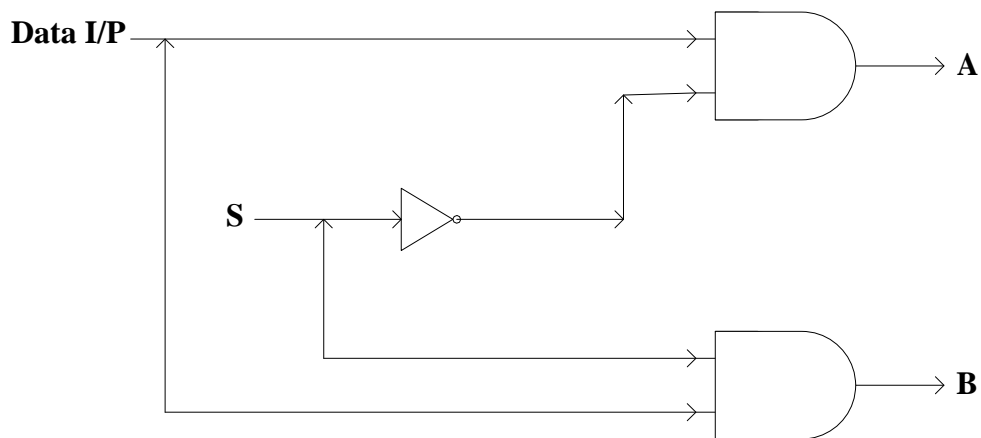
(A)



Synchronous

(B)

**Fig. (3)**  
**MUX/DMUX System: (A). Switch Analog. (B). Logic Gate Circuit .**



**Fig. (5)**  
**One to Two Lines Demultiplexer**

## **Procedure**

1. Connect the circuit as shown in Fig. (2).
2. Apply a signal to (A) input from clock (High Speed) of the logic INTIKIT unit. Draw the wave form.
3. Apply signal to (B) input from the pulse generator of amplitude = 5 Volt (p-p) frequency = 50 KHz. Draw the wave form.
4. Set the selector control input ( $S = 0$ ), draw the output waveform from the multiplexer.
5. Set  $S = 1$ , draw the output waveform of the multiplexer.
6. Connect the output of MUX to the DMUX circuit of Fig. (5) and find the output of demultiplexer when  $S = 0$  and when  $S = 1$ .

## **Discussion**

1. Construct  $8 \times 1$  MUX by using  $2 \times 1$  MUX.
2. Construct  $16 \times 1$  MUX by using  $4 \times 1$  MUX and  $2 \times 4$  Decoder.
3. Give some applications for the Multiplexer.
4. How many chips of  $2 \times 1$  MUX you will need if you want to design  $16 \times 1$  MUX.
5. Design the following expression using multiplexer:

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

Put the selection circuit  $S_1, S_0 = A C, A B, B C$ .



## Exp. No. (6)

### Flip – Flops

#### Object

1. To familiar the student with Flip-Flops.
2. To differentiate between various types of the Flip-Flops.
3. How to determine the next state of each type of Flip-Flops.

#### Theory

A Flip-Flop (F-F) is a multivibrator which has two stable state (Bistable) High and Low. Flip-Flop (F-F) are useful devices for applications such as counting, storing binary data and data conversion from serial to parallel

There are many different types of Flip-Flops:

1. a) Simple Set-Reset (S-R) Flip-Flop.  
b) Clocked (S-R) Flip-Flop.
2. J-K Flip-Flop.
3. D-Type Flip-Flop.
4. T-Type Flip-Flop.

#### **1. a) The Set-Reset (S-R) Flip-Flop:**

Fig. (1) shows the basic construction of S-R F-F. It has two output called Q and  $\bar{Q}$  and like the toggle switch, when it's (1) output is high, it's (0) output is low and vice-versa.

When the power is applied to the machine the F-F will go either to its (Set) or (Reset) state. The initial state is arbitrary depending on the relative characteristics of the components which comprise the two logic gates.

Fig. (1-b) shows the meaning of Set and Reset states in terms of binary voltage levels. To see how F-F operates, refer to Fig. (1-a), Assume that the initial state is the (Set) state ( $L_1$  ON and  $L_2$  OFF), and that both the Set and Reset lines are low (0). The Set and Reset lines are both stable when they are low. In the Set state gate  $N_1$  provides a high level to upper inputs of the  $N_2$ . Therefore, both input to  $N_2$  are high causing it to produce a low output which turns  $L_2$  OFF. The low output of  $N_2$  is also applied to the lower input terminal of  $N_1$ . This produces a high output from gate  $N_1$ , turning  $L_1$  ON. In the set state then,  $N_2$  is continually enabling  $N_1$ , and  $N_1$  continually disabling  $N_2$ .

The two gates are latched in a condition that will remain until the input conditions on the S-R lines are changes.

If the F-F is initially in the Reset state ( $L_2$  ON and  $L_1$  OFF), in this case  $N_1$  is continually enabling  $N_2$  and  $N_2$  is continually disabling  $N_1$ . The circuit is again latched output but in the opposite direction.

Apply a high to the S terminal with the F-F in the Reset state ( $L_2$  ON and  $L_1$  OFF) at this case the output  $N_1$  go to High, reversing the latch condition of the previous case. If a high is then applied to the R terminal, the latch condition is reverses again (i.e. the  $L_1$  is OFF and  $L_2$  is ON).

The truth table for the S-R F-F is shown in Fig.(1-b). As the first line of the truth table shows no change (N.C.) occur in the F-F state if both S and R lines are Low (0).

The second line shows the input conditions needed to cause the F-F to go to the Reset state.

The third line of the truth table shows the input condition needed to cause the F-F to go to the Set State.

The fourth line shows a set of "illegal" input conditions which are usually avoided by the machine designer.

The F-F cannot "remember" this state. Since the primary value of the F-F is in its memory capability. The "illegal" input-output situation can be ignored.

### **1. b) The Clocked Set-Rest ( Clock S-R) Flip-Flop:**

It is also called Steered S-R Flip-Flop. When a F-F is used, it is frequently desirable to establish the desired Set or Rest state first, then have it go to that state at some later point in time.

The process of pre-establishing the desired state is called (Steering).

Fig.(2) shows a simple form of clocked S-R F-F. It is the same S-R F-F of Fig.(1-a) except that a steering network comprised of additional logic gates ( $N_3$  and  $N_4$ ) has been added to the input circuit. The input levels at steering terminals A and B cannot by themselves change the state of the F-F. A "trigger" or "clock" pulse or EXT. signal shot must be applied. When this occurs, the F-F will go to the state directed by the HIGH-LOW conditions at the steering terminal. If A is High when the clock pulse is applied, gate  $N_4$  will be enabled. This provide a low level to  $N_2$  switching the F-F to the reset state.

### **2. The J-K Flip-Flop:**

The J-K F-F combines the capability of the S-R and clocked S-R Flip-Flops into a single element. Fig.(4-a) shows the combination of the J-K F-F, and Fig. (4-b) shows the truth table of the J-K F-F.

### **3. D-Type Flip-Flop:**

A D-type Flip-Flop can constructed from the S-R F-F as shown in Fig. (3) and from J-K F-F as shown in Fig.(6).

### **4. T-Type Flip-Flop:**

A T-type F-F can be constructed from the J-K F-F as shown in Fig. (7).

## **Procedure:**

1. Hook up the simple S-R F-F circuit shown in Fig. (1-a) by using integrated circuit (I.C) type ----- and state it's truth table.

### **Note:**

Logic (1) = + 5 Volt = + VCC = HIGH.

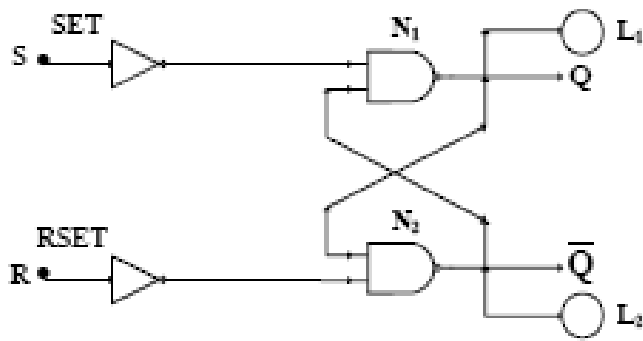
Logic (0) = 0 Volt = GND = LOW.

2. Hook up the steering circuit for the clock S-R F-F of Fig.(2) and determine the next state truth table when:
  - a. Clock equal to zero (GND) (Logic 0).
  - b. Clock equal to + VCC (Logic 1).
3. Hook up D-type F-F in Fig. (3) by using (I.C.), type ----- with clock terminal set to EXT signal shot, when D in it's low position (GND). State the output Q, and when D in it's high position (+VCC) state the output Q.
4. Hook up J-K F-F shown in Fig.(5-a) and determine the output of the truth table of Fig. (5-b) by using (I.C.) type ----- .
5. Hook up the circuit shown in Fig. (6) D-type F-F by using (I.C.) type ----- and state it's truth table.
6. Repeat step 5 for Fig. (7) T-type F-F.

## **Discussion:**

1. For the simple S-R F-F try to implement the same F-F using two NOR gates, give it's truth table.
2. What is the advantage of using J-K F-F over other types of Flip-Flops.
3. Give an electronic circuit using discrete components i.e. transistors. Diodes and resistors for J-K F-F.
4. From the clocked S-R F-F, show how can you form D-type and T-type Flip-Flops.





I/P		O/P	
S	R	Q	$\bar{Q}$
0	0	N.C	N.C
0	1	0	1
1	0	1	0
1	1	1	1

Fig. (1-a) S-R Flip-Flop Construction. Fig. (1-b) Truth Table of S-R Flip-Flop .

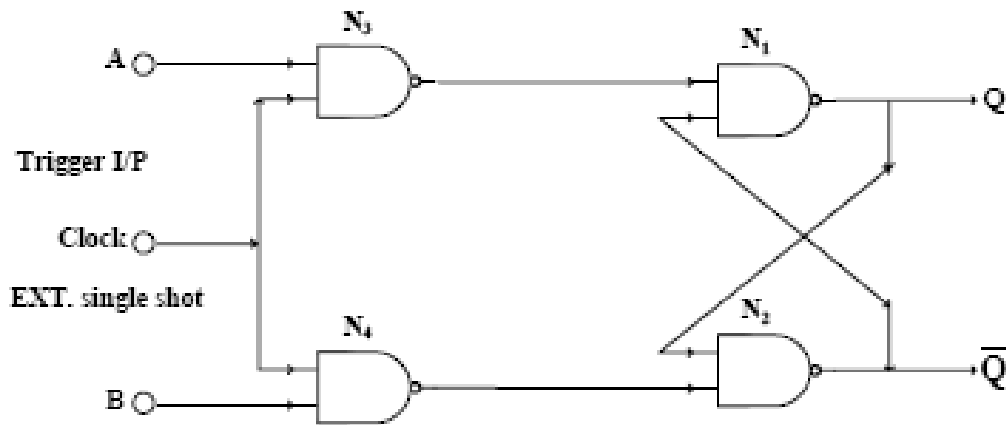


Fig. (2)  
Steering Circuit or Clocked S-R Flip-Flop.

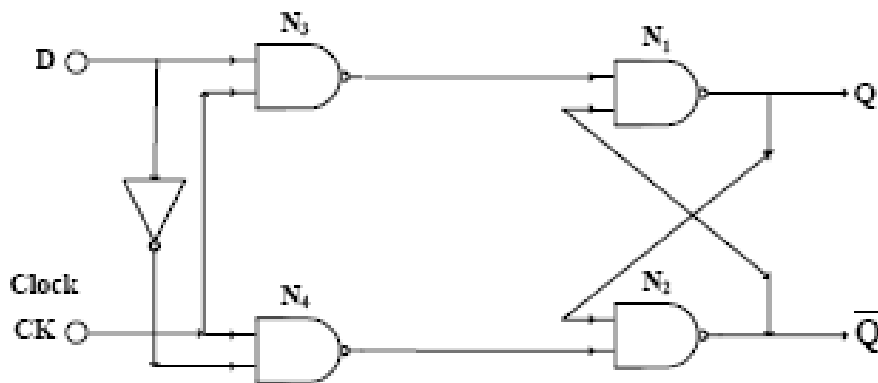


Fig. (3)  
D-Type Flip-Flop Construction.

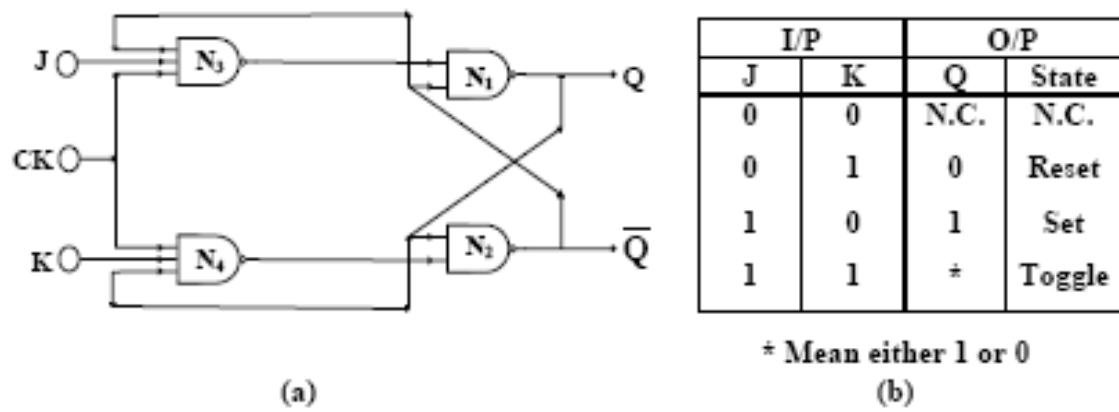


Fig. (4)  
 (a): J-K F-F Construction. / (b): J-K F-F Truth Table.

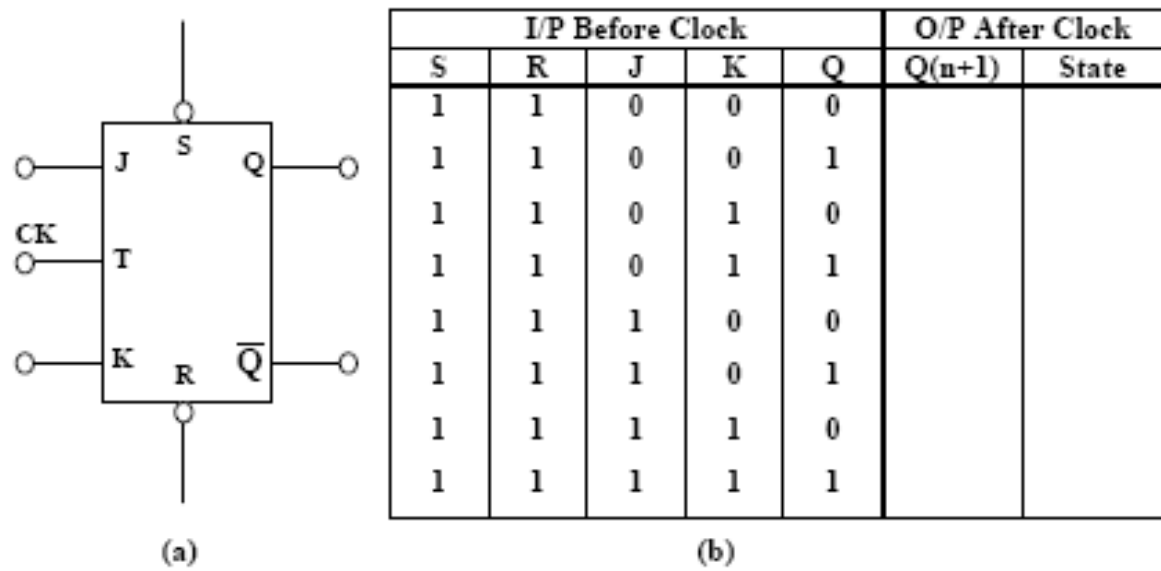
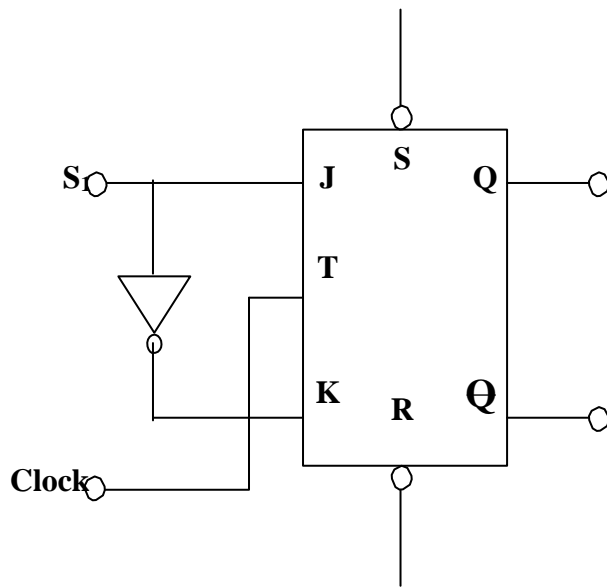
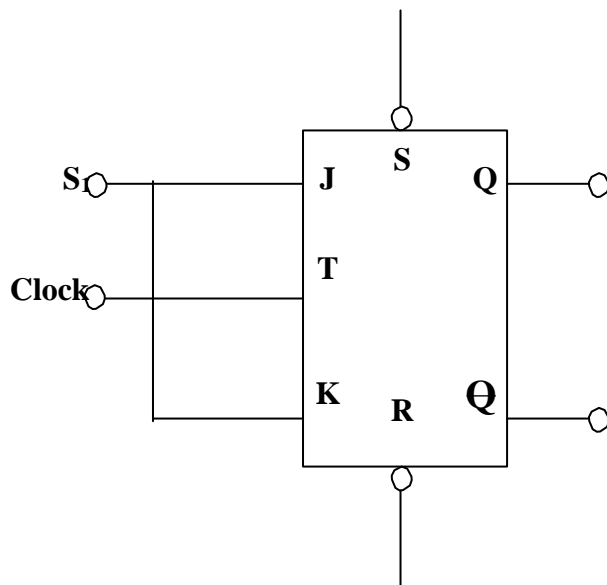


Fig. (5)  
 (a): J-K F-F Chip. / (b): J-K F-F Truth Table.



**Fig. (6)**  
**D-Type Flip-Flop**



**Fig. (7)**  
**T-Type Flip-Flop**



---

**Exp. No. (7)**  
**Digital Counter**

**Object**

After completing this experiment, you will be able to:

1. Build and analyze various a synchronous up and down counter.
2. Change the model of the counter.
3. Use an IC counter and determine how to truncate its count sequence.

**Theory**

Digital counters are classified as synchronous or asynchronous, depending on how they are clocked.

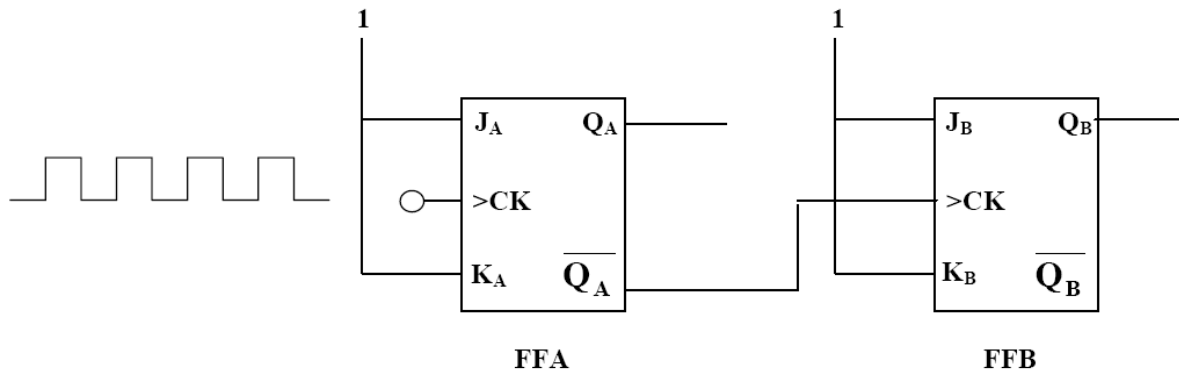
Synchronous counters are a series of Flip-Flops, each clocked at the same time, causing the outputs of the stages (Flip-Flops) to change together. By contrast, asynchronous counters are a series of Flip-Flops, each clocked by the previous stage, one after the other.

Since all stages of the counter are not clocked together, a "ripple" effect propagates as various Flip-Flops are clocked. For this reason, asynchronous counters are called ripple counters. You can easily make a ripple counter from D or J-K Flip-Flops by connecting them in a toggle mode. The modulus of a counter is the number of different output states the counter may take. You can change the modulus of a ripple counter by decoding any output state and using the decoded state to synchronously preset or clear the current count. Ripple counters can be made to count either up or down (it can be made to count both up and down, but usually it is easier to use a synchronous counter for an up/down

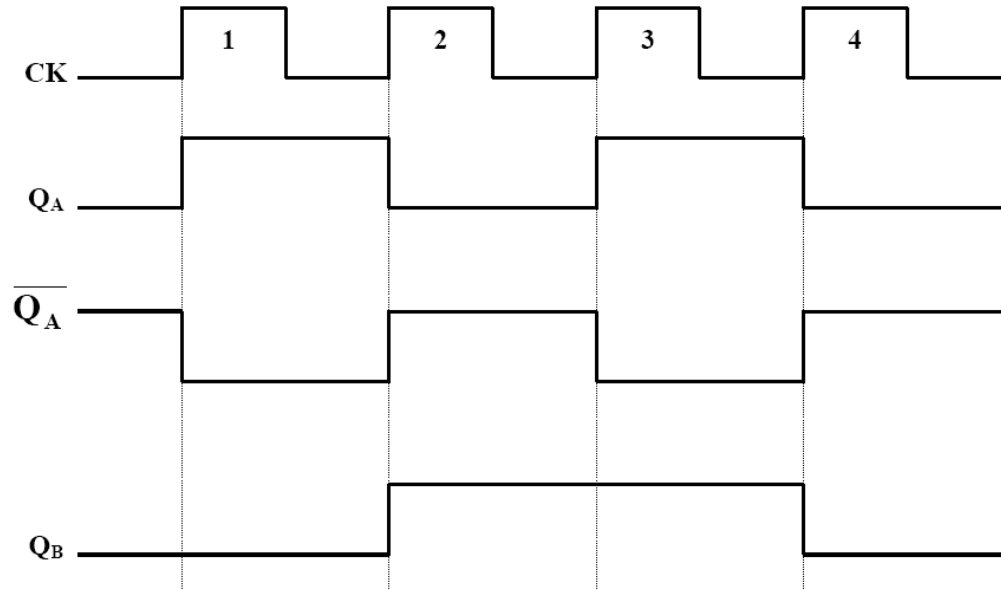
counter).

Fig. (1) shows a two-stage counter connected for asynchronous operation, notice that the clock line is connected to the clock (CK) input of only the first stage, FFA. The second stage, FFB, is triggered by the  $Q_A$  output of FFA. FFA changes state at the positive-going edge of each clock pulse, but FFB change only when triggered by a positive-going transition of the  $Q_A$  output of FFA.

Fig. (2) shows the timing diagram of QA and QB outputs.



**Fig. (1)**  
**Two Stage Asynchronous Binary Counter.**

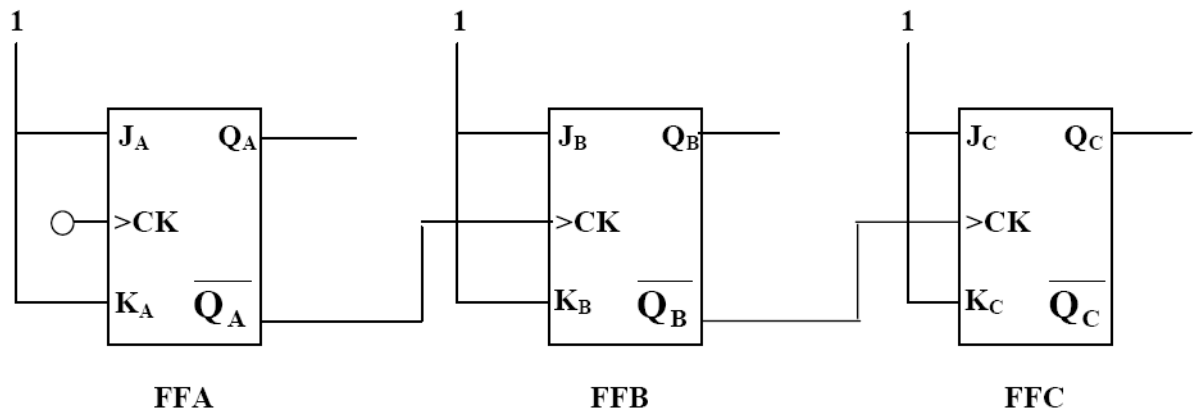


**Fig. (2)**  
**Timing Diagram For The Counter Of Fig. (1).**

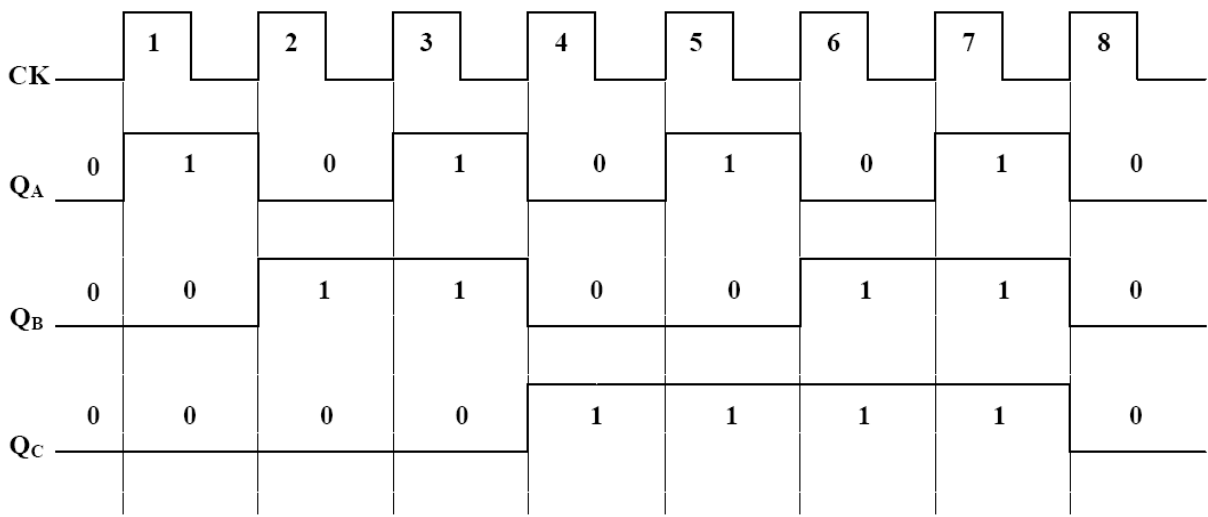
A three-stage asynchronous binary counter is shown in Fig. (3-a). The basic operation is the same as that of the two-stage counter just discussed, except that it has eight states due to its three states. A timing diagram appears in Fig. (3-b) for eight clock pulse.

<b>Clock Pulse</b>	<b>QC</b>	<b>QB</b>	<b>QA</b>
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

**Table (1) State Table for a Three-Stage Binary Counter.**



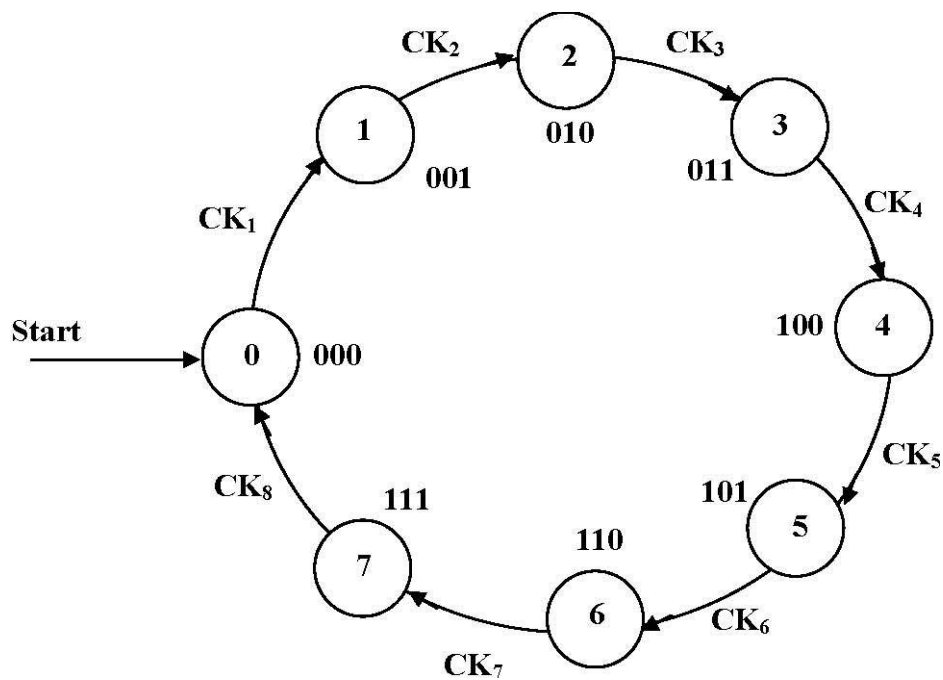
**Fig. (3-a)**  
**Three-Stages Asynchronous Counter.**



**Fig. (3-b)**  
**Timing Diagram of Three-Stage Asynchronous Counter.**

Notice that the counter progress through a binary count of 0 to 7 and then recycles to the 0 state.

This counter sequence is presented in the state diagram shown in Fig. (4).



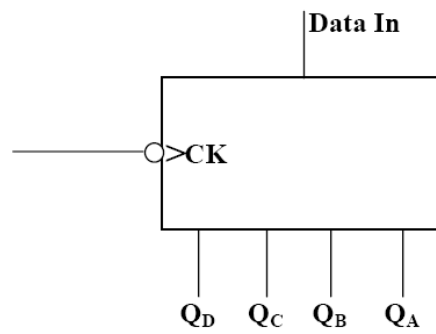
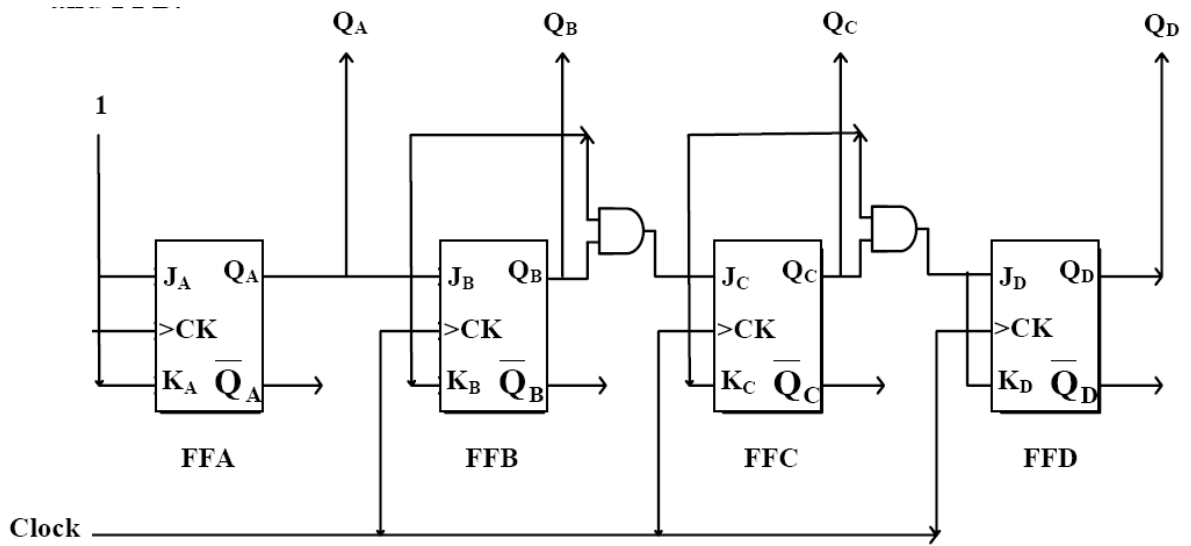
**Fig. (4) State Diagram For A Three-Stage Binary Counter.**

The synchronous counter is also called a parallel counter because the clock line is connected in parallel to each Flip-Flop. Notice that an arrangement different from that for the asynchronous counter. Fig. (5) shows a four-stage binary counter and its equivalent logic symbol.

This circuit can be connected by using IC 74163 which has several features in addition to basic functions previously discussed for the general synchronous binary counter. First, the counter can be preset to any four-bit binary number by applying the proper levels to the data inputs.

The circuit below is a 4-bit synchronous binary counter. The J and K inputs of FFA are connected to HIGH. FFB has its J-K inputs connected to the output of FFA, and the J-K inputs of FFC are connected to the output of an AND gate that is fed by the outputs of FFA and FFB, also the J-K inputs of FFD are connected to the output of AND gate that is fed by the output of FFC and FFB.





**Fig. (5)**  
**A Four-Stage Synchronous Binary Counter.**

## *Decade Counter*

Decade counters are very important category of digital counter because of their wide application, a decade counter has ten states in its sequence that is, it has modulus of ten. It consist of four stages and can have any given sequence of states as long as there are ten. A very common type of decade counter is the BCD (8421) counter, which exhibits a binary-coded-decimal sequence as shown in Table (2).

As you can see, the BCD decade counter goes through a straight binary sequence through the binary 9 state, rather than going to the binary 10 state, it recycles to the 0 state. A synchronous BCD decade counter is shown in Fig. (6).

<b>CK</b>	<b>QD</b>	<b>QC</b>	<b>QB</b>	<b>QA</b>
<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>
<b>1</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>1</b>
<b>2</b>	<b>0</b>	<b>0</b>	<b>1</b>	<b>0</b>
<b>3</b>	<b>0</b>	<b>0</b>	<b>1</b>	<b>1</b>
<b>4</b>	<b>0</b>	<b>1</b>	<b>0</b>	<b>0</b>
<b>5</b>	<b>0</b>	<b>1</b>	<b>0</b>	<b>1</b>
<b>6</b>	<b>0</b>	<b>1</b>	<b>1</b>	<b>0</b>
<b>7</b>	<b>0</b>	<b>1</b>	<b>1</b>	<b>1</b>
<b>8</b>	<b>1</b>	<b>0</b>	<b>0</b>	<b>0</b>
<b>9</b>	<b>1</b>	<b>0</b>	<b>0</b>	<b>1</b>

*Table (2) States Of BCD Decade Counter*

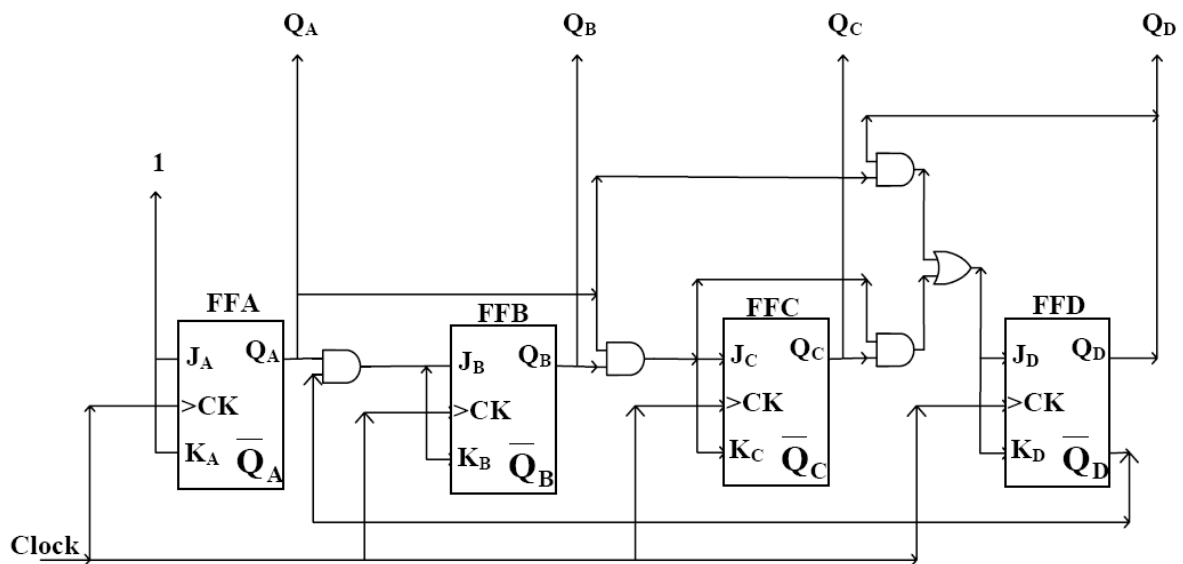


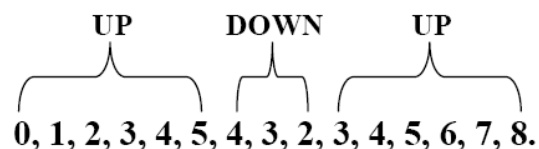
Fig. (6)  
BCD Synchronous Decade Counter

### Up-Down Counters

An up-down counter is one that is capable of progressing in either direction through a certain sequence. An up-down counter some time called a bi-directional counter and can have any specified sequence of states.

In general, most up-counters can be reversed at any point in their sequence.

For example the following sequence:



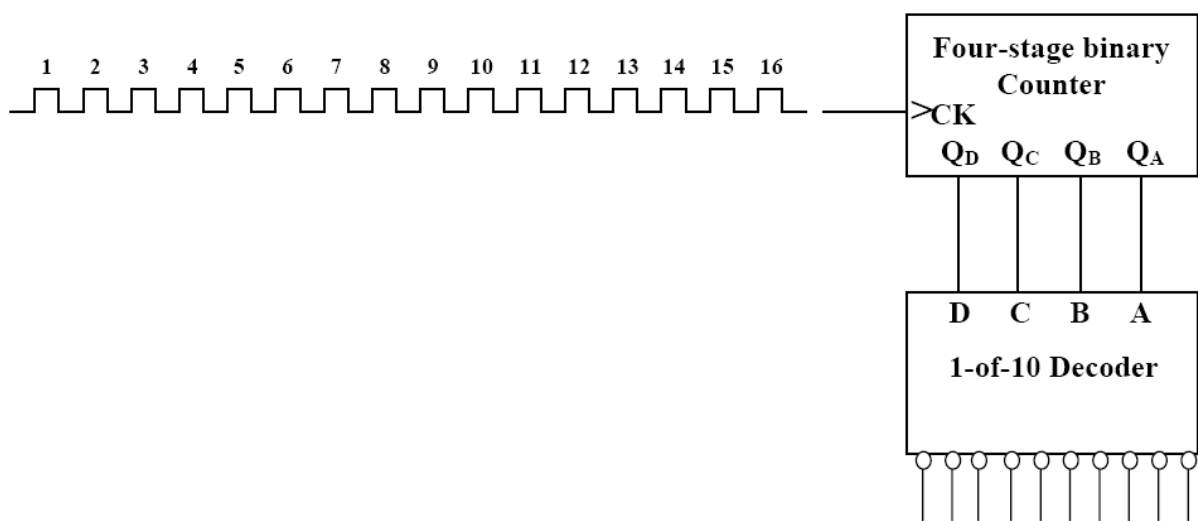
### Procedure

1. Connect the circuit shown in Fig. (1) and find the truth table and its timing diagram.
2. Connect the circuit shown in Fig. (3-a) and find its truth table and the timing diagram.
3. Use 74163 to design a four-stage synchronous counter, and find its truth table and timing diagram.

4. Use 74190 to design the BCD up/down counter, then find its truth table and timing diagram for BCD up counter.
5. Design parallel counter (Up/Down) using J-K F-F for the following sequence (3, 4, 5, 6, 7, 4, 3, 2).

### Discussion

1. For the four-stage binary counter connected to the 1-of-10 decoder in Figure below, determine each of the decoder output wave form in relation to the clock pulses. QA is the LSB of counter, and A is LSB of decoder.

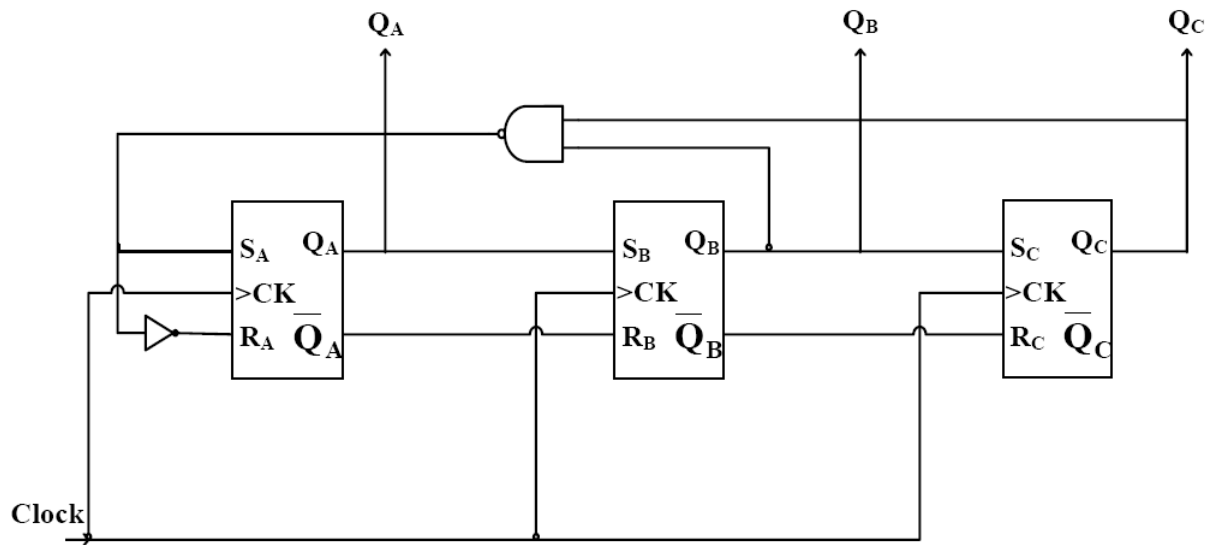


2. Given a BCD decade counter, show the decoding logic required to decode each the following states and how it should be connected to the counter. A HIGH output indication is required for each decade state.

MSB is to the left.

- (a). 0101                      (b). 0111                      (c). 1000

3. Determine the sequence of the following counter:



4. What are the applications of counter?

5. Design a counter that counts the following sequence: (1, 3, 5, 7, 9, 11) and repeat using J-K F-F.



## Exp. N0. (8)

### The Shift Registers and There Applications

#### Object

1. To familiar the students with the shift register.
2. To make use of shift register in data transfer.

#### Theory

Any binary machine is said to have a particular "Word Length". These terms defines the number of bits required to represent data,

In other words, a machine which said to have a four-bit word length has its flip flops arranged in groups of four. The group of flip flops are consider as a single unit called a "Register".

The binary number is "Shifted" one bit at time from one flip flop to the next. The device used in this type of transfer operation it called a "Shift Register"

A shift register is a series of interconnected flip flops used for temporary storage of data as shown in Fig. (1). The output of one flip flop becomes the input of another, all the flip flops in the shift register have a common clock signal connection and all can be set or reset at the same time. Because the data were loaded to the circuit one bit after another and the shift register shifted them from one flip flop to another, this sequence is referred to as serial data loading and the circuit is called a "4-BIT SERIAL IN-SERIAL OUT SHIFT REGISTER" as shown in Fig. (1).

This type of shift register accepts digital data serially that is one bit at the time on one line. It produces the stored information on its output also in serial form.

The alternative to serial loading of the shift register is parallel loading, for a register with parallel data input, the bits are entered simultaneously into their respective stages on parallel-lines, rather than on a bit-by-bit basis on one line as with serial data inputs. Fig. (2) shows a "4 BIT PARALLEL IN-PARALLEL OUT REGISTER". In the parallel output register the output of each stage is available, once the data are stored, each bit appears on its respective output line and all bits are available simultaneously, rather than on a bit-by-bit basis as with the serial output.

If the data are loaded serially and read out in parallel, the shift register is functioning as a "SERIAL-TO-PARALLEL CONVERTER". If the data is loaded in parallel and shifted out serially, the shift register is functioning as a "PARALLEL-TO-SERIAL CONVERTER". Some shift registers are configured to allow shifting the data in both the right and left direction. These shift registers are usually called "Universal Shift Register", because they can shift data in either right or left direction, can load data either serially or in parallel and can output data either serially or in parallel.

## **Procedure**

1. Connect the circuit as shown in Fig. (1) by using 74174 I.C.
2. To clear the register set the pin CLR to GND then to VCC.
3. Show how you can store binary number 1010 in this chip. Find the state of each flip flop after each clock pulse as shown in Table bellow and draw the timing diagram for this case.

CLK	D <sub>1</sub>	Q <sub>1</sub>	Q <sub>2</sub>	Q <sub>3</sub>	Q <sub>4</sub>
0	0	0	0	0	0
1	1				

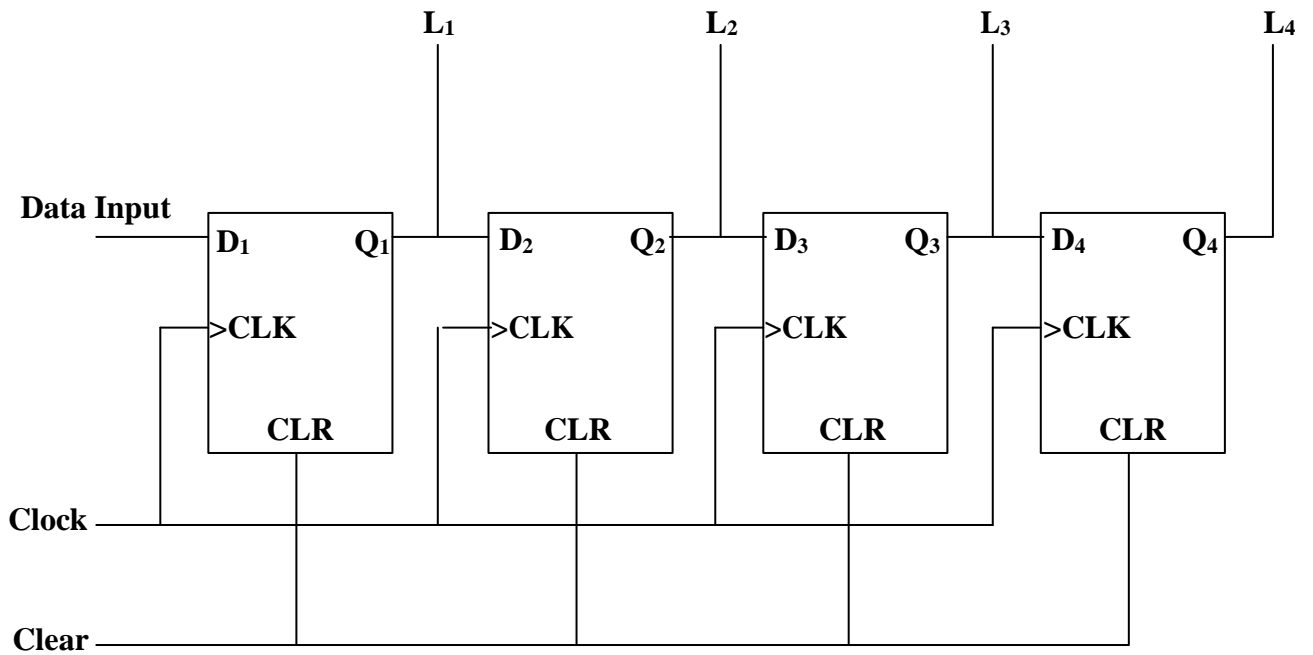
2	0	
3	1	
4	0	

4. Connect the circuit as shown in Fig. (2), show how you can store the binary number 011011 in this chip.

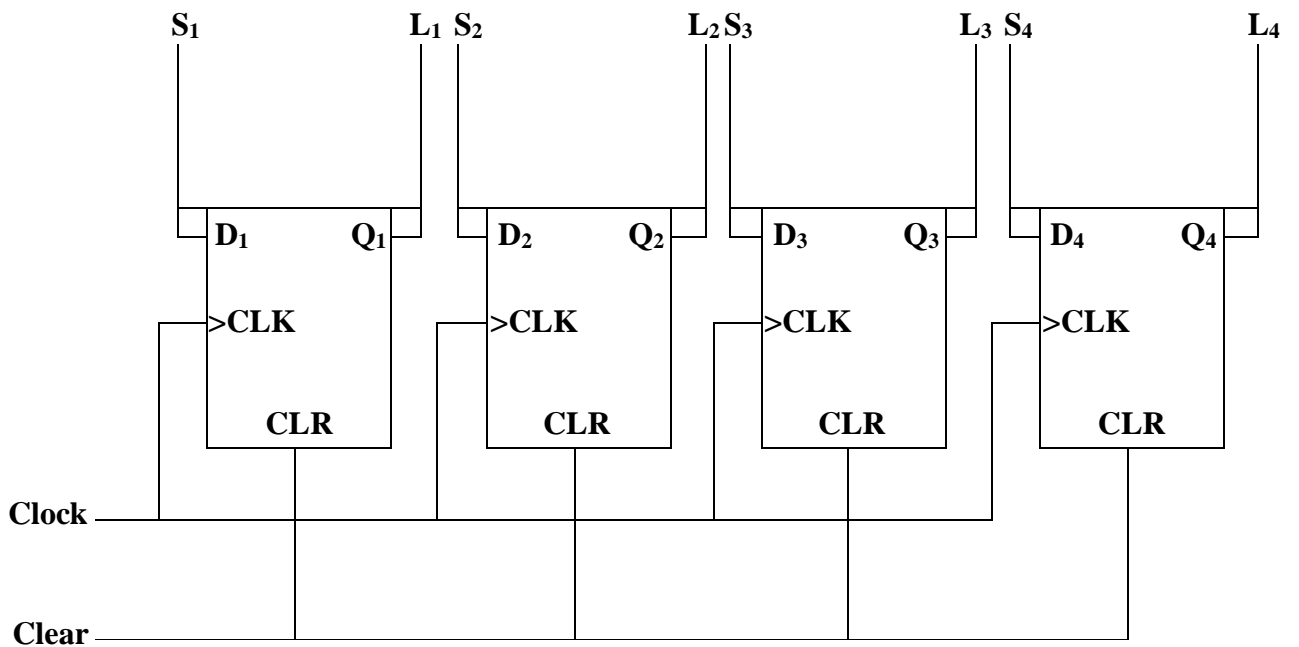
### **Discussion**

1. What is the output binary number after 3-clock pulses if the number stored in the register is 1101, and  $D_1$  is connected to GND (0).
2. In Fig. (1) if the output  $Q_4$  is returned to the  $D_1$  input of flip flop 1, and the number stored in the register is binary 1101, what would be the output after 7-clock pulse?
3. Design 4-bit Parallel In-Serial Out shift register (PISO), using D-type Flip-Flop.
4. Design 3-bit Serial In-Parallel Out shift register (SIPO), using J-K Flip-Flop.





**Fig. (1)**  
**Serial In – Serial Out Shift Register.**



**Fig. (2)**  
**Parallel In – Parallel Out Shift Register.**

## **Exp no. (9)**

### **Microprocessor Training Kit**

#### **Object:-**

General description to Microprocessor training kit M85-0X .

#### **Introduction:-**

The M85-0x LCD Kit is a single board Microprocessor Training Kit based on 8085 Microprocessor , which is widely used to train engineers to develop software/hardware for any industrial process & control.

M85-0x LCD Kit provides powerful monitor EPROM & user's RAM with Battery Backup as shown in figure (1). The kit has 101 IBM compatible PC keyboard & 20x2 LCD display for any Data entry / display.

This kit has line Assemble feature so that one can enter the program in Assemble language . The kit also has the capability of interacting with PC (computer) through RS-232C Serial link .

The input/output structure of M85-0x LCD kit provided 48 programmable I/O lines Using 2Nos. of 8255. It has got 16 bit 3channel programmable Timer/Counter using 8253.

The on board residents system monitor software is very powerful and provides various software utilities like INSERT,DELETE,BLOCK MOVE, RELOCATE, STRING, FILL&MEMORY COMPARE etc. which are very helpful in debugging/ developing The software.

## **THE 8085 MICROPROCESSOR:**

The 8085 represents the first generation of microprocessor chips. It is an 8 bit micro with 8 bit data bus, 16 bit address bus and a 6 bit control bus.

The data bus is used for the transfer of information between the micro chip and the rest of the system.

The address bus is used for addressing the required place of memory. Since the address bus is a 16 bit, then the maximum size of memory is  $(2)^{16}=64K$  byte of memory.

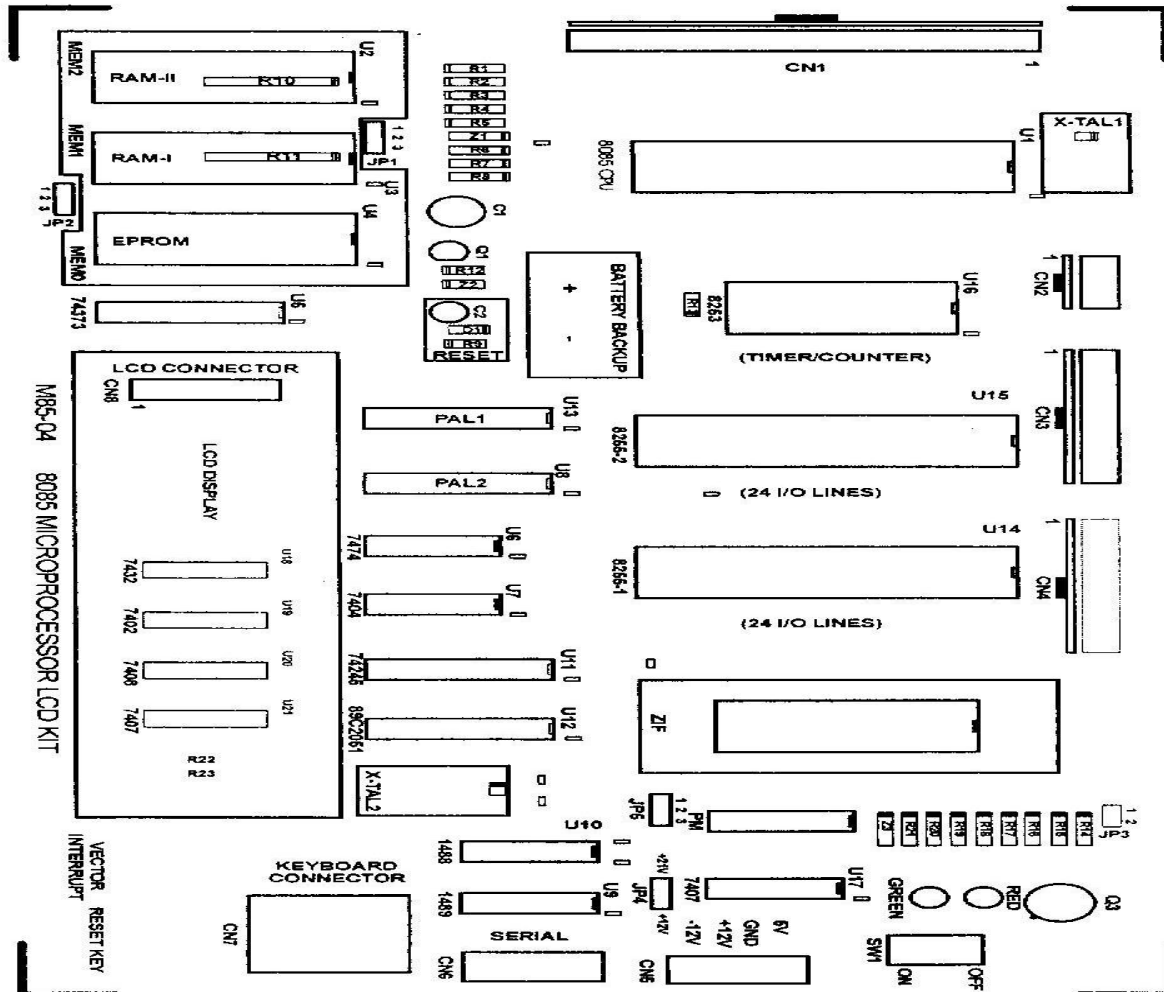
The control bus is composed of six signals to control the (Read/Write) of memory and (I/O) selection.

Internally the 8085 chip contain 8 bit accumulator, used as part of all arithmetic and logic operations performed by the microprocessor. The flag is an 8 bit register, but utilizing only five of the bits namely: zero, sign, carry, parity, and auxiliary carry. Also the 8085 contain six another 8 bit general purpose register, namely: B, C, D, E, H and L.

General purpose registers can be combined as register pairs – BC, DE, and HL – to perform some 16-bit operations. For register pair BC, register C can be determined as the low register and register B as the high. So that registers (D and E) and registers (H and L).

The Program Counter (PC) is 16 bit register used to point the location of the next program step to be executed. The Stack Pointer (SP) is also a 16 bit register used to point the location of the stack area in memory. The Processor Status Word (PSW), is another 16 bit register which comprises the accumulator and the flag register

(PSW: high byte=A, Low byte=Flag register).



Fig(1-a)

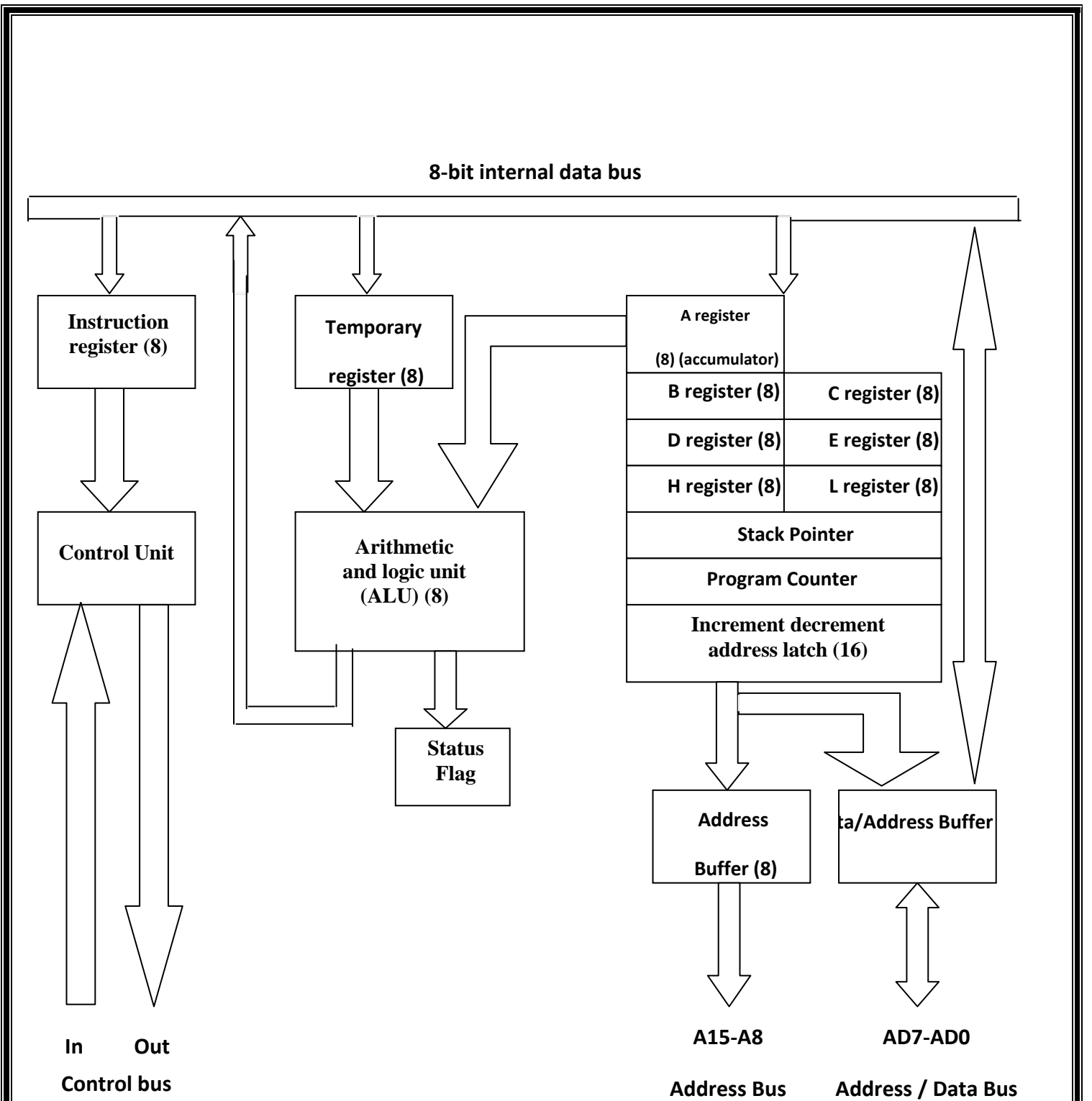


Figure (1-B)

The Intel 8085 CPU

### **\*Keyboard Description:-**

The M85-0x LCD kit has 101 ASCII keys and 20x2 liquid Crystal Display to communicate with the outside world. As M85-0x LCD kit is switched on, a message “8085 LCD TRAINER KIT M85-0X\_” is displayed on the LCD display and all keys are in command mode.

### **LIST OF ASCKII KEYBOARD COMMANDS**

1. L :- list a memory block .
2. M :- Examine/Modify Memory .
3. E :- Enter a memory block .
4. R :- Examine /modify Register .
5. S :- Single Step .
6. G :- Go .
7. B :- Block Move .
8. I :- Insert .
9. D :- Delete .
10. N :- Insert Data .
11. O :- Delete Data .
12. F :- Fill .
13. H :- Relocate .
14. J :- Memory compare
15. K :- String

The flag register is affected by the result of arithmetic and logical operations only. The **structure of flag register** is as follow:

**S (Sign flag):**

This bit is set (logic 1) if the most significant bit of the result of an operation is “1” otherwise it is reset (logic 0).

**Z (Zero flag):**

This bit is set if the content of the accumulator after an operation is zero, otherwise it is reset.

**CY (Carry flag):**

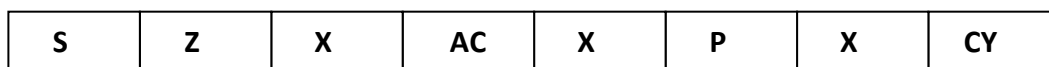
This bit is set if an operation causes carry or borrow out of the most significant bit of the accumulator.

**AC (Auxiliary Carry):**

This bit is set if there is a carry between the fourth and fifth bits of the accumulator.

**P (Parity flag):**

This bit is set if the number of ones in the accumulator is even, otherwise it is reset.



Bit 7

Bit 0

X : Don't care

Flag register diagram

## THE 8085 INSTRUCTION SET

The 8085 instruction set includes five different types of instructions.

1. Data Transfer Group:

Move data between registers or between memory and register.

2. Arithmetic Group:

Add, subtract increment or decrement data in registers or memory.

3. Logical Group:

AND, OR, EXCLUSIVE-OR, compare, rotate or complement data in registers or in memory.

4. Branch Group:

Conditional and unconditional jump instructions, subroutine call instructions and return instructions.

5. Stack, I/O, and Machine Control Group:

Includes I/O instructions, as well as instructions for maintaining the stack and internal control flags.

**Note:** only Arithmetic and logical instructions affected the flag register.

The 8085 can operate either on the internal CPU registers (A, B, C, D, E, H, & L) or in the system memory (RAM or ROM). The different addressing capabilities on the 8085 are:

a. Implied:

Meaning that the operation involves an operation on one of the registers. (E.g. ADD B, INR C).

b. Immediate:

Which involves an operation with a type supplied immediately after the instruction. (E.g. ADI 03, ORI 02).

c. Direct:

Which involves an operation with data found on the address supplied as two bytes after the instruction. (e.g. LDA 2050, STA 2051).



## ASSEMBLY LANGUAGE PROGRAM

Examples of instructions:

No	Instruction	Type	No. of Bytes	Function	Effect on flags
1	MVI rd,byte	Data transfer	2	rd=byte	None
2	MOV rd,rs	Data transfer	1	rd=rs	None
3	INR r	arithmetic	1	r=r+1	All but CY
4	DCR r	arithmetic	1	r=r-1	All but CY
5	HLT	Machine Control	1	Stop processing	None
6	RST5	Restart	1	Pc=0008	None

r: register (8-bit)

rd: destination register (8-bit)

rs: source register (8-bit)

Assembly language programs are written in a standard format as follows:

<u>Address</u>	<u>Hexcode</u>	<u>Label</u>	<u>Opcode</u>	<u>Operand</u>	<u>Comment</u>
2000	3E	START:	MVI	A, 03	; A=3
2001	03				
2002	3C		INR	A	; A=A+1=4 ; S=0, Z=0, Ac=0, P=0
2003	EF		RST5		; End

The address field specifies the address of the respective instruction.

Language program. This field contains the data to be entered to the machine.

The label field specifies the label for the program.

Opcode field specifies the 8085 instructions to be executed.

Operand field specifies the data to be operated by the corresponding instruction.

Comment field is an optional field used to comment lines.

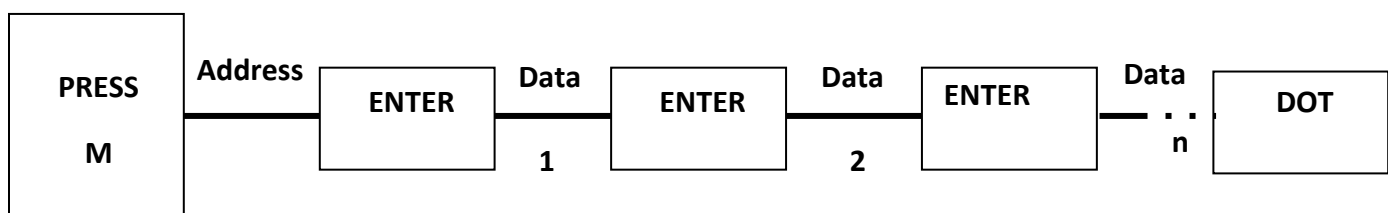
### Memory Modification and Program Entry:

**Reset** key is used to reset the system.

**M** key from keyboard is used to locate memory location to examine or modify its contents.

**ENTER** key from keyboard is used to store the display data byte in memory and go to the next memory Location and display its contents.

**DOT (.)** key from keyboard is used to start execution or to terminate program



e.g. Suppose we want to enter the following values in the computer:

2000	3E,	MVI	A, 3
	03		
2002	3C	INR	A
2003	EF	RST5	

Then the following steps are to be performed:

1. Press M
2. Press 2000 ENTER
3. Press 3E ENTER
4. Press 03 ENTER
5. Press 3C ENTER
6. Press EF ENTER
7. Press DOT

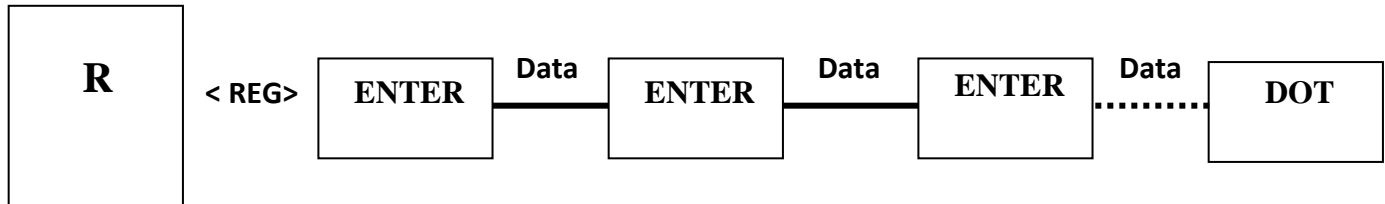
In order to check the entries of the program, we do the following steps:

1. Press M
2. Press 2000 ENTER
3. Press ENTER
4. Press ENTER

5. Press ENTER
6. Press DOT

**Register Display and Modification:**

**R** key from keyboards used to display the contents of registers.



e.g. To display the contents of the register, do the following steps:

1. Press R key from keyboard .
2. Press A and ENTER, the contents of the acc. Is displayed.
3. Press ENTER, the contents of the register B is displayed.
4. Press ENTER, the contents of the register C is displayed.

Continuing in the same way, the contents of the registers D, E, F (flags), I (Interrupt mask), H, L, SPH, SPL, PCH and PCL will be displayed.

**Note:-** If you need to examine the register contents, terminates the program with the instruction RST5 (machine code EF) instead of HLT (machine code 76). The RST

instruction stops the user's program and returns control of the computer to the monitor program.

**Run and Single Step Modes:**

**S** key is used to execute the program, one instruction at a time.

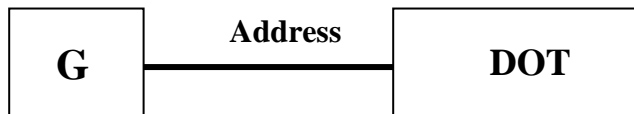
To execute a program in a single step mode, do the following:



e.g. To execute a program at address 2000. Do the following:

1. Press **S**
2. Press 2000 ENTER
3. Press ENTER
4. Press ENTER
5. Press DOT

To run the program at address 2000, we use:



**G** key Press from keyboard is used to locate memory location to start execution.

To execute program at address 2000, do the following:

1. Press **G**
2. Press 2000 DOT

**Lab work**

**Q) Shown below is a coding of a sample assembly program:**

<u>Address</u>	<u>Hexcode</u>	<u>Label</u>	<u>Opcode</u>	<u>Operand</u>	<u>Comments</u>
2000	3E	START:	MVI	A, 05	; A=5
2001	05				
2002	3D		DCR	A	; A=A-1=4 ; S=0, Z=0, Ac=0, P=0
2003	3D		DCR	A	; A=A-1=3 ; S=0, Z=0, Ac=0, P=1
2004	3D		DCR	A	; A=A-1=2 ; S=0, Z=0, Ac=0, P=0
2005	EF		RST5		; End

1. Enter the above program in the memory.
2. Check the values of the entered program.
3. Run the program using single step command and check the values of the registers.
4. Run the program using direct run mode.

## **Home Work**

Write programs with effects

1.  $B=C+1$  when  $C=50h$ .
2. Enter  $A=10$  then decrement 4 from register A.

## **Exp.no. (10)**

### **Command Description**

#### **Object:**

To explain the command description & understand what meaning each character in the keyboard.

#### **Theory:**

The M85-0X LCD kit has 101 ASCII keys and 20x2 liquid crystal display to communicate with the outside world. As M85-0X LCD kit is switched on, a message “8085 LCD TRAINER KIT M85-0X” is displayed on the LCD display and all keys are in command mode.

#### **List of ASCII keyboard commands**

##### ➤ **1. memory block (L)**

L command dump specified memory block in the PC screen .

#### **FORMAT**

**L low address, High address (.)**

Type L followed by the starting address of the memory block to be listed , followed by a comma (,) and then the end address of the memory block followed by (.) dot



## **Example**

Suppose you want to list the data from 2000 to 2010 .

**L 2000,2010 (.)**

```
2000  21  00  21  46  AF  23  86  05  C2  05
200A  20  32  00  21  FF  FF  FF  (.)
```

## ➤ **2.EXAMINE/MODIFY MEMORY (M)**

The M command allows you to examine and modify memory locations individually.

The command functions as follows:

### **Format**

**M Address,(Data),.....(.)**

- ❖ Type M, followed by the hexadecimal address of the first memory location you wish to examine, followed by a space or comma.
- ❖ The contents of the location are displayed followed by a dash ‘-’.
- ❖ To modify the contents of the location displayed, type the new data , followed by a comma. The higher memory location will automatically be displayed as the step (2). A (.) at any stage terminates the command.

### Example

**M 2000 , C3 – 3E , 23 – 01 , FC – 06 , 02 , 21 – 80 , 3E – EF (.)**

The contents of 2000 to 2002 , 2004 & 2005 are changed from C3 to 3E , 23 to 01 , FC to 06 , 21 to 80 & 3E to EF respectively where as the data at 2003 remains as it is .

Location	Old contents	New contents
2000	C3	3E
2001	23	01
2002	FC	06
2003	02	02
2004	21	80
2005	3E	EF

### ➤ **3.ENTER A MEMORY BLOCK (E)**

**E** command allows user to enter a program or a block of data in the RAM.

### Format

**E address : data , data .....(.)**

- ❖ Type E followed by the starting address of the memory block to be entered, followed by a colon (:).
- ❖ Each byte followed by a comma as it is entered from the SIOD is deposited in the consecutive location of the memory.

- ❖ In case the terminator is colon (:) the proceeding parameter is taken as a fresh address and the subsequent data bytes are stored in memory location starting from the fresh address.
- ❖ A (.) terminates the command.

**Example**

E 2000 (enter)

2000: 3E , 01 , 06 , 02 , 80 , EF(.)

➤ **4.EXAMINE/MODIFY REGISTER (R)**

Display & modification of CPU register is accomplished via ‘R’ command.

**Format**

R (Register identifier.....< . > )

Register Identifier	Register
A	Register A
B	Register B
C	Register C
D	Register D
E	Register E
F	Flag byte
I	Interrupt Mask
H	Register H
L	Register L
S	Stack Point MSB
P	Program Counter MSB

- ❖ Type 'R' Followed by a single alphabet register identifier . The contents may
- ❖ now be changed if so desired. In case you do not want to modify the contents, Just enter a comma. The contents of the next register will be printed. The register identifiers for various CPU registers are given above.

**Example**

RA-11,B-22,C-33

➤ **5.SINGLE INSTRUCTIONS (S)**

This command allows executing the program one instruction at a time.

**Format**

S (Starting Address),

- ❖ Pressing of 'S' Key will list the PC and first byte of the program. In case one want to modify it, user has to entered the new address and then press comma. The new address will be listed. In this way one can execute the program in single instruction mode.
- ❖ This command can be terminated at any time by enter (.)

**Example**

The following program is to be executed in single instruction mode:

Address	Opcode	Instruction
2000	3E 01	MVI A,01
2002	06 02	MVI B,02
2004	80	ADD B
2005	EF	RST 5

On executing 'S' command S2000:3E/,2002:06/,2004:80/.if one wants to execute Further , press enter otherwise one presses (.)

## ➤ 6.GO COMMAND (GO)

This command is used to execute the program in full clock speed. On pressing this key, the program counter contents are displayed. Enter the starting address of the program and press (.) key . The CPU will starts executing the program.

### **FORMAT**

The format for this command will be as follows:

G (Starting address) (.)

Pressing of 'G' Key will display the PC content and the first byte of the instruction.

To modify it, enter the desired address & then press comma, the PC will be modified

with new contents & the corresponding data will be listed . When (.) key is pressed ,

CPU starts executing the program.

Suppose the program starts from 2000 then the format will be G 2000 (.)

### **EXAMPLE**

Execute the below sample program from 2000 location.

Address	opcode	instruction
2000	3E 01	MVI A,01
2002	06 02	MVI B,02
2004	80	ADD B
2005	EF	RST 5

G 2000 (.)

## ➤ **7. Block move command (B)**

This command allows to move the block of data from one memory location to another location. On pressing this key, 'B\_' is displayed, type starting address, ending address of the block to be moved and enter the destination address and press (.) key. The format for the command is as follows:

### **FORMAT**

B(Starting address of the source),(End address of the source),(Starting address of the destination)(.)

## ➤ **8. Insert Command (I)**

This command allows to insert one or more instructions in the user's program with automatic modification of the memory referenced instructions. On pressing this command, 'I-' is displayed, enter the starting address of the program press (enter), enter the ending the byte or bytes are to be entered press (enter), enter the no. of bytes press (enter) enter the required data and press (.). When all the bytes are entered '8085 LCD TRAINER KIT M85-0X\_' is displayed. The format for this command is as follows:

### **FORMAT**

I (Starting address of the program),(End address of the program),(Address from where the byte or bytes are to be entered),(No. of bytes),(Data),,(.)

## EXAMPLE

Address	opcode	instruction
2000	3E 01	MVI A,01
2002	06 02	MVI B,02
2004	80	ADD B
2005	EF	RST 5

Now we want to insert three bytes at location 2005 (32 00 21) i.e. STA 2100 instruction.

I 2000,2005,2005,3,32,00,21,

Verify that the bytes have been inserted using examine memory command.

### ➤ 9.DELETE COMMAND (D)

This command allows to delete one or more instruction from the user program. In this command all the memory referenced instructions also get modified accordingly to keep the logic of the program. On pressing this command, D\_ is displayed, enter the starting address of the program press (enter) enter Ending address of the program press (enter) starting addresses from where the bytes are to be deleted press (enter) ending address till where the bytes are to be deleted and press(.) after that '8085 LCD TRAINER KIT M85-0X\_' is displayed. The format for this command is as follows:

## **FORMAT**

D(Starting address of the program),(End address of the program),(Starting address from where the bytes are to be deleted),(End address till where the bytes are to be deleted)(.).

### ➤ **10.FILL(F)**

This command allows the user to fill a memory area(RAM) with a constant data .

## **FORMAT**

F(Starting address of the program),(End address of the program),(Data to be filled)(.)

## **EXAMPLE**

Suppose the RAM area, from 2000 to 2010 is to be filled with FF.The format will be:

F 2000,2010,XX-FF(.)

Verify that the memory area from 2000 to 2010 is filled with constant data FF by using Examine memory command.



## ➤ **11.RELOCATE (H)**

This command allows the user to relocate the program from one memory area to another memory area. The relocate command is somewhat different from Block Move command. This can be set to be the intelligent mode of Block Move command i.e. if user wants to execute a program from a different RAM area, this command will set the program as required. This can be understood clearly after working with the example given below.

### **FORMAT**

H(Starting address),(Ending Address),(Destination address) (.)

## ➤ **12.MEMORY COMPARE (J)**

This command allows the user to compare two blocks of memory for equality. If they are not equal the address of the first block at which there is a difference will be displayed.

### **FORMAT**

J(Starting address of first block),(Ending address of first block),(Straying address of second block)(.)

## **EXAMPLE**

Enter the following data using Examine Memory command:

Address	opcode	instruction
2000	3E 01	MVI A,01
2002	06 02	MVI B,02
2004	80	ADD B
2005	EF	RST 5

Now block move this block to 2100 using Block move command. Now user Memory

Compare command as follows:

J 2000,2005,2100(.)

If the two blocks are identical, '8085 LCD TRAINER KIT M85-0X\_' is displayed.

Now change the contents of 2004 location to 10 using EXAMINE MEMORY Command. Again use the memory compare command as mentioned above. You will see that an address 2004 with its data will be displayed. On pressing (enter) '8085 LCD TRAINER KIT M85-0X\_' is displayed. It mean other contents are compared same.

### ➤ **13.STRING(K)**

This command allows to find the address or addresses at which a particular string of data is lying with in a specified program. The word string here means a few bytes of data lying consecutively one after another.

## **FORMAT**

K(Starting address),(Ending address),(Address where the first byte lies),(Address where the last byte lies)(.)

## **EXAMPLE**

Suppose in the example given below , if you want to find out the addresses at which the CALL DELAY instruction is lying.

Address	opcode	instruction
2000	3E 80	MVI A,80
2002	D3 03	OUT 03
2004	3E 55	MVI A,55
2006	D3 00	OUT 00
2008	CD A6 03	CALL DELAY
200B	3E AA	MVI A,AA
200D	D3 00	OUT 00
200F	CD A6 03	CALL DELAY
2012	C3 04 20	JMP 2004

K 2000,2014,2006,2008(.)

In the above example, note the two addresses at which CALL DELAY is lying.

These addresses will be 2008 and 200F. Verify by Examine Memory command that CD A6 03 is lying at 2008 onwards.As the string found at location 2008, the message"STRING FOUND AT:2008"will come. Now go ahead by pressing (enter).

## **Lab work:**

1. Inter to register A (04) and to register B (05) and then change the content of register B to (02).
2. Delete three bytes of the program from 2005 to 2007.

Address	opcode	instruction
2000	3E 01	MVI A,01
2002	06 02	MVI B,02
2004	80	ADD B
2005	32 00 21	STA 2100H
2008	EF	RST5

3. Move the program from location 2000-2005 to location 2100.

Address	opcode	instruction
2000	3E 01	MVI A,01
2002	06 02	MVI B,02
2004	80	ADD B
2005	EF	RST 5

### Home work

1. Write a program to add two number (1,2)hex and move the program to address 2100 and compare with the program that starting from address 2000.
2. Relocate the program lying from 2000-200A to 2200,the program is given below.

Address	OpCode	Instruction
2000	3E 01	MVI A,01
2002	0602	MVI B,02
2004	80	ADD B
2005	32 00 21	STA 2100H
2008	C3 02 20	JMP 2002

3. How to fill the memory area to constant data(FF) .

## EXP. NO. (11)

### DATA TRANSFER AND ARITHMETIC INSTRUCTIONS

#### OBJECT:

To understand and to program the 8085 microprocessor with programs utilizing the data transfer and arithmetic instruction groups.

#### THEORY

An instruction in the data transfer group, transfers data from a source location to a destination location. Either of these may be a memory location or a register.

#### The data transfer operations are:

1. MVI rd , byte :- This instruction moves the immediate data given after the instruction into register rd.
2. MOV rd , rs :- This instruction moves data from register rs to register rd.
3. LXI rp, 16-bit:- This instruction moves 16-bit data or address to register pair.
4. XCHG: 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.

No	Instruction	Type	No. of Bytes	Function	Effect on flags
1.	MVI rd,byte	Data transfer	2	rd=byte	None
2.	MOV rd,rs	Data transfer	1	rd=rs	None
3.	LXI rp,16-bit	Data transfer	3	rp=16-bit (data/ address) Lreg=Lbyte, Hreg=Hbyte	None
4.	XCHG	Data Transfer	1	HL $\longleftrightarrow$ DE	None

rd: destination register (8-bit), rs: source register (8-bit).

rp: register pair (16-bit): (BC, DE, and HL)

#### The arithmetic operations are:

The arithmetic instructions perform operations on the data stored in memory or registers and affect the flag register. These instructions are:

1. INR r :- This instruction increments the contents of the register r.  
$$r=r+1$$
2. ADD r :- This instruction adds the contents of the register r to the accumulator, and store the results back into the accumulator.  
$$A=A+r$$

3. ADI byte:- This instruction adds 8-bit data (byte) to the accumulator, and stores the results back into the accumulator.

$$A=A+\text{byte}$$

4. ADC r :- This instruction adds the contents of the register r to the accumulator, but also adds the carry from pervious step, and store the results back into the accumulator.

$$A=A+r+\text{Cy}$$

5. ACI byte :- This instruction adds 8-bit data to the accumulator, but also adds the carry from pervious step, and store the results back into the accumulator.

$$A=A+\text{byte}+\text{Cy}$$

6. DCR r :- This instruction decrements the contents of the register r.

$$r=r-1$$

7. SUB r :- This instruction subtracts the contents of the register r from the accumulator, and stores the results back into the accumulator.

$$A=A-r$$

8. SUI byte:- This instruction subtracts 8-bit data (byte) from the accumulator, and stores the results back into the accumulator.

$$A=A-\text{byte}$$

9. SBB r :- This instruction subtracts the contents of the register r from the accumulator, but also subtracts the carry from pervious step, and store the results back into the accumulator.

$$A=A-r-\text{Cy}$$

10. SBI byte :- This instruction subtracts 8-bit data from the accumulator, but also subtracts the carry from pervious step, and store the results back into the accumulator.

$$A=A-\text{byte}-\text{Cy}$$

11. DAA:- This instruction converts the contents of accumulator 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.

The conversion procedure is as follow:-

If the value of the low-order 4-bits in the accumulator is greater than 9 or if the 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.

$$A=\text{BCD number (A)}$$

12. INX rp :- This instruction increments the contents of the register pair rp.

$$rp=rp+1$$

13. DCX rp :- This instruction decrements the contents of the register pair rp.

$$rp=rp - 1$$

14. DAD rp:- This instruction adds the contents of the register pair rp to the contents of register pair HL and store the result back into the HL.

$$HL=HL +rp$$

No	Instruction	Type	No. of Bytes	Function	Effect
1.	INR r	arithmetic	1	$r=r+1$	All but CY
2.	ADD r	arithmetic	1	$A=A+r$	All
3.	ADI byte	arithmetic	2	$A=A+byte$	All
4.	ADC r	arithmetic	1	$A=A+r+CY$	All
5.	ACI byte	arithmetic	2	$A=A+byte+CY$	All
6.	DCR r	arithmetic	1	$r=r-1$	All but CY
7.	SUB r	arithmetic	1	$A=A-r$	All
8.	SUI byte	arithmetic	2	$A=A-byte$	All
9.	SBB r	arithmetic	1	$A=A-r-CY$	All
10.	SBI byte	arithmetic	2	$A=A-Byte-CY$	All
11.	DAA	arithmetic	1	$A=BCD \text{ number (A)}$	All
12.	INX rp	arithmetic	1	$rp=rp+1$	NONE
13.	DCX rp	arithmetic	1	$rp=rp-1$	NONE
14.	DAD rp	arithmetic	1	$HL=HL+rp$	CY

### Notes:-

- 1- Most of the arithmetic instructions affect the contents of an important CPU register; namely the flag register.
- 2- Most of arithmetic instructions using 8-bit registers are done using the accumulator.
- 3- The increment or the decrement operations can be performed in any register.
- 4- For the addition of 8-bit registers, the accumulator is always the 1<sup>st</sup> operand, but the addition of 16-bit registers, the HL register pair is always the 1<sup>st</sup> operand.

### Example

Find the summation of register B and register C and put the result in register D, when B=5 and C=3

<u>Address</u>	<u>Hexcode</u>	<u>Label</u>	<u>Opcode</u>	<u>Operand</u>	<u>Comments</u>
2000	06	START:	MVI	B, 5	; B=5
2001	05				
2002	0E		MVI	C,3	; C=3
2003	03				
2004	78		MOV	A,B	; A=B=5
2005	81		ADD	C	; A=A+C=8 ; S=0, Z=0, Ac=0, P=0, Cy=0
2006	57		MOV	D,A	; D=A=8
2007	CF		RST1		; End

## Lab Work

1.  $A=(B-C)-(D+E)$  when  $BC=0a502h$ ,  $DE=403h$
2.  $C=(B+20)+(D-30)-1$  when  $B=20h$ ,  $D=40h$
3.  $HL=BC+DE$  when  $BC=2ffh$  and  $DE=102h$  using Instructions use Register Pair

## Home Work

Write programs with effects

1. Exchange the content of DE with HL when  $DE=15ah$ ,  $HL=8b1ch$  Instructions use Register Pair.
2. Put the same value (30h) in register H and register L, then subtract 10h from register H and add 10h to register L, after that increment register H by 1 and decrement register L by 1.
3.  $HL=BC+DE$  when  $BC=2ffh$  and  $DE=102h$  using Instructions use 8-bit Registers
4.  $H=(A-10h)-(C+3ah)+1$  when  $A=5fh$ ,  $C=10h$
5. What is the result of each instruction of the following program and its effect?

```
MVI A,2Fh
SUI 20
INR A
MVI B,8
ADD B
DCR A
MVI C,7
SUB C
SUI 10
RST1
```



## EXP. NO. (12)

# LOGICAL INSTRUCTIONS OF THE 8085 MICROPROCESSOR

### OBJECT:

To study the logical capabilities of the 8085 microprocessor.

### THEORY

The logical instructions, as the name implies perform logical operations on the data stored in the memory or the register. These operations are supported by the three different modes of addressing which are; the implied, immediate and direct modes.

All the addressing modes use the accumulator as the second operand to perform one of the following instructions, and generated a result again in the accumulator. The corresponding flags are set according to the result after each instruction. The logical instructions supported by these operations are:

1. AND Operation (ANA, ANI) :- These instructions ANDs the accumulator with the required 8-bit data, and put the result again in the accumulator. These instructions are symbol as follows:

$$A=A.X$$

2. OR Operation (ORA, ORI) :- These instructions ORs the accumulator with the required 8-bit data, and put the results again in the accumulator. These instructions are symbol as follows:

$$A=A+X$$

3. EX-OR Operation (XRA, XRI) :- These instructions Exclusive-ORs the accumulator with the required 8-bit data, and put the results again in the accumulator. These instructions are symbol as follows:

$$A=A\oplus X$$

4. Rotate Operations (RLC, RRC, RAL and RAR) :- These instructions address the accumulator only. They perform a shift left or shift right of the accumulator contents.

5. Complement Operation (CMA) :- This instruction also addresses the accumulator only. It performs the ONEs complement of the accumulator contents.

$$A=\overline{A}$$

6. Also falling under this category are certain instructions that do not affect the contents of the accumulator or any other register, yet it affects the appropriate flags. These instructions are:

Compare Operations (CMP, CPI) :- These instructions compare the accumulator with the required 8-bit data, setting the appropriate flag and leaving the accumulator untouched.

No	Instruction	Type	No. of Bytes	Function	Effect
1.	ANA r	Logical	1	A=A and r	All
2.	ANI byte	Logical	2	A=A and byte	All
3.	ORA r	Logical	1	A=A or r	All
4.	ORI byte	Logical	2	A=A or byte	All
5.	XRA r	Logical	1	A=A xor r	All
6.	XRI byte	Logical	2	A=A xor byte	All
7.	CMA	Logical	1	A=A	None
8.	CMC	Logical	1	CY=CY	CY
9.	STC	Logical	1	CY=1	CY

**Notes:-**

- 1- Most of logical instructions affect the contents of an important CPU register; namely the flag register.
- 2- Except CMC and STC instructions. All Logical instructions use accumulator as the 1<sup>st</sup> operand.
- 3- You can use STC then CMC to reset the carry flag CY.

## Review of logical operations:

### 1. Complement

#### E.g.:

MVI A,5Ah

CMA

HLT

A= 5Ah= 01011010

$\overline{A}$ =0A5h= 10100101

### 2. And

#### E.g.:

MVI A,5Ah

MVI B,1Fh

ANA B ; or directly using ANI 1FH

HLT

A= 5Ah= 01011010

B= 1Fh= 00011111

A AND B=1Ah= 00011010

; 1=1 AND 1 ONLY, otherwise=0

### 3. Or

#### E.g.:

MVI A,5Ah

MVI B,1Fh

ORA B ; or directly using ORI 1FH

HLT

A= 5Ah= 01011010

B= 1Fh= 00011111

A OR B=5Fh= 01011111

; 0=0 OR 0 ONLY, otherwise=1

### 4. Xor

#### E.g.:

MVI A,5Ah

MVI B,1Fh

XRA B ; or directly using XRI 1FH

HLT

A= 5Ah= 01011010

B= 1Fh= 00011111

A XOR B=45h= 01000101

; 0=0 XOR 0, 0=1 XOR 1, otherwise=1

### Notes:-

- The instruction **XRA A** is used to **reset the accumulator** which is preferred than **MVI A, 0** because the 1<sup>st</sup> instruction is translated to one byte rather than the 2<sup>nd</sup> one which translated to two bytes.

#### E.g.:

Reset the Accumulator

MVI A,5bh

XRA A

HLT

```
A= 5bh=  01011011
A= 5bh=  01011011
          -----
A xor A=0 = 00000000
```

- To **complement specific bits** of the accumulator, use **XRI** instruction with byte when the corresponding bits are set and the other bits are reset.

#### E.g.:

Complement bit0 and bit7 of A, when A=5b

MVI A,5bh

XRI 81

HLT

```
A= 5bh=  01011011
81=     10000001
          -----
A xor 81=dah = 11011010
```

- To **set specific bits of the accumulator**, use **ORI** instruction with byte when the corresponding bits are set and the other bits are reset.

#### E.g.:

Set bit0 of A, when A=5ah

MVI A,5ah

ORI 1

HLT

```
A= 5ah=  01011010
1=     00000001
          -----
A OR 1=5bh= 01011011
```

- To **reset specific bits** of the accumulator, use **ANI** instruction with byte when the corresponding bits are reset and the other bits are set.

**E.g.:**

Reset bit0 of A, when A=5bh  
MVI A,5bh  
ANI 0feh  
HLT

```

A= 5bh=  01011011
        0Feh= 11111110
        -----
A AND 0feh=5ah = 01011010

```

- In other word reset bits means **mask these specific bits**. Bit masking involves isolating one or more bits in a binary quantity while hiding or masking the unwanted bits is usually done with the logical instructions. Then you can used unmask bits in decision making when using branch instructions. In more precise words this operation is called **checking status bit** which involve **ANI** instruction with byte when the corresponding bits are set and the other bits are reset. Then we can check the result if it is zero that is mean that the status bit still unready, otherwise the status bit is one and wanted state is ready.

**E.g.:**

Mask all bits of A=5bh except bit 4 (checking status bit 4)  
MVI A,5bh  
ANI 10  
HLT

```

A= 5bh=  01011011
        10=  00010000
        -----
A AND 10=10 = 00010000

```

## Rotate Instructions

**RAL:** Each binary bit of the accumulator is rotated left by one position through the carry flag. Bit  $D_7$  is placed in the bit in the carry flag and the carry flag is placed in the least significant position  $D_0$ .

**RAR:** Each binary bit of the accumulator is rotated right by one position through the carry flag. Bit  $D_0$  is placed in the carry flag and the bit in the carry flag is placed in the most significant position  $D_7$ .

**RLC:** Each binary bit of the accumulator is rotated left by one position Bit  $D_7$  is placed in the position of  $D_0$  as well as in the carry flag.

**RRC:** Each binary bit of the accumulator is rotated right by one position Bit  $D_0$  is placed in the position of  $D_7$  as well as in the carry flag.

### Examples

#### RAL

	<b>CY=0</b>		<b>CY=1</b>
MVI A,8fh	; A=8fh	STC	; CY=1
RAL	; A=1eh, CY=1	MVI A,8fh	; A=8fh
		RAL	; A=1fh, CY=1

#### RAR

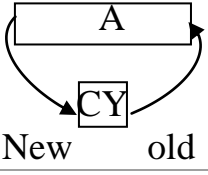
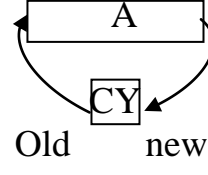
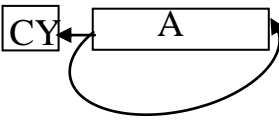
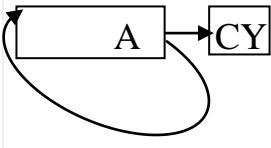
	<b>CY=0</b>		<b>CY=1</b>
MVI A,1fh	; A=1fh	STC	; CY=1
RAR	; A=0fh, CY=1	MVI A,1fh	; A=1fh
		RAR	; A=8fh, CY=1

#### RLC

	<b>CY=0</b>		<b>CY=1</b>
MVI A,8fh	; A=8fh	STC	; CY=1
RLC	; A=1fh, CY=1	MVI A,8fh	; A=8fh
		RLC	; A=1fh, CY=1

#### RRC

	<b>CY=0</b>		<b>CY=1</b>
MVI A,1fh	; A=1fh	STC	; CY=1
RRC	; A=8fh, CY=1	MVI A,1fh	; A=1fh
		RRC	; A=8fh, CY=1

No	Instruction	Type	No. of Bytes	Function	Benefits	Effect
1.	RAL (Rotate All Left)	Logical	1	Shift left A one pos., bit0=old cy New cy=bit7 	Delay, Mull. By 2 *reset cy before using this instruction	CY only
2.	RAR (Rotate All Right)	Logical	1	Shift Right A one pos., bit7=old cy, New cy=bit0 	Delay, Div. By 2 *reset cy before using this instruction	CY only
3.	RLC (Rotate Left with Carry)	Logical	1	Shift left A one pos., bit0=bit7, New cy=bit7 	Delay, Check bit7 which saved in the cy. Use RRC then to retrieve the old byte	CY only
4.	RRC (Rotate Right with Carry)	Logical	1	Shift Right A one pos., bit7=bit0, New cy=bit0 	Delay, Check bit0 which saved in the cy. Use RLC then to retrieve the old byte	CY only

### **Notice:-**

- 1- Each of these instructions manipulates the accumulator and the (CY) flag during the rotation process. The rotate instructions are often used to check the status of individual bit. This is done by rotating the bit into the carry flag, and then using the JC, JNC instructions to jump, based on the (CY) flag value.
- 2- Rotate instructions are also used to perform binary multiplication and division. A binary value is multiplied by (2) by shifting the bits left by one bit position. Similarly, a number is divided by two by shifting it right.

## LAB Work

1.  $C = (A + D) \text{ XOR } (C \text{ AND } 15h)$  when  $A = 0F1h$ ,  $D = 0E3h$ ,  $C = 36h$
2.  $D = (H \text{ OR } B) \text{ NAND } (C - D)$  when  $H = 15h$ ,  $B = 2Ah$ ,  $C = 0Ah$ ,  $D = 5$

## Home Work

Write programs with effects

- 1-  $HL = (BC + HL) \text{ OR } DE$  (use register pair when necessary), when  $BC = 105h$ ,  $HL = 340h$ ,  $DE = 180h$
- 2- Reset bits 0,2 of A and set bits 4,6,7 when  $A = 0A7H$
- 3-  $D = (C \text{ OR } 5) - (B \text{ XOR } D)$  when  $D = 3fh$ ,  $BC = 1da5h$
- 4- What is the effect of the following instructions:  
ANA A, ORA A, CMA, XRA A when  $A = 75h$
- 5- What is the result of each instruction of the following program and its effect?

```
MVI A,56h
CMA
STC
MVI B,0FH
ANA B
ANI 1
ORA B
ORI 80h
XRA B
XRA A
XRI 32h
STC
RST1
```



## EXP. NO. (13)

# COMPARSION AND JUMP INSTRUCTIONS OF THE 8085 MICROPROCESSOR

### OBJECT:

To study the comparison capabilities of the 8085 microprocessor and to further investigate the conditional and unconditional branch instructions.

### THEORY:

Compare Operations (CMP, CPI) :- These instructions compare the accumulator with the required 8-bit data, setting the appropriate flag and leaving the accumulator untouched.

No	Instruction	Type	No. of Bytes	Function	Effect
1.	CMP r	Logical	1	Compare r with A (A-r)	All
2.	CPI byte	Logical	2	Compare byte with A (A-byte)	All

The result of the comparison is shown by setting the flags as follows:-

If  $A = (r/\text{byte})$  then  $CY=0$  and  $Z=1$

If  $A < (r/\text{byte})$  then  $CY=1$  and  $Z=0$

If  $A > (r/\text{byte})$  then  $CY=0$  and  $Z=0$

### Notes of flags affection

We can see from the previous experiments that only arithmetic and logical instructions affect the flag register, and we notice some exceptions.

- 1- The four rotate instructions, STC, CMC, and DAD instructions affect carry flag only and we can write the effect is: CY.
- 2- DCR and INR instructions affect all the flag bits except the carry bit and we can write the effect is: All but CY.
- 3- INX, DCX, and CMA instructions don't affect any flag bits and we can write the effect is: None.
- 4- Other arithmetic and logical instructions affect all the flag bits and we can write the effect is: All.

It is known that the flow of some program may be deviated by specific jump instructions. These jumps test the status of the appropriate flags and jump accordingly to the specified address, given by the two bytes following the jump instruction in the order (Low Byte, High Byte). The types of JUMPs supported are:

1. JMP (address) :- This instruction jumps unconditionally to the specified address.
2. JZ (address) :- This instruction tests the zero flag bit, and jumps to the specified address if this bit is set.
3. JNZ (address) :- This instruction tests the zero flag bit, and jumps to the specified address if this bit is reset.
4. JC (address) :- This instruction tests the carry flag bit, and jumps to the specified address if this bit is set.
5. JNC (address) :- This instruction tests the carry flag bit, and jumps to the specified address if this bit is reset.
6. JM (address) :- This instruction tests the sign flag bit, and jumps to the specified address if this bit is set.
7. JP (address) :- This instruction tests the sign flag bit, and jumps to the specified address if this bit is reset.
8. JPE (address) :- This instruction tests the parity flag bit, and jumps to the specified address if this bit is set (even parity).
9. JPO (address) :- This instruction tests the parity flag, bit, and jumps to the specified address if this bit is reset (odd parity).

No	Instruction	Type	No. of Bytes	Function	Effect
1.	JMP address	Branch	3	Pc=address	None
2.	JZ address	Branch	3	Pc=address if Z=1	None
3.	JNZ address	Branch	3	Pc=address if Z=0	None
4.	JC address	Branch	3	Pc=address if CY=1	None
5.	JNC address	Branch	3	Pc=address if CY=0	None
6.	JM address	Branch	3	Pc=address if S=1	None
7.	JP address	Branch	3	Pc=address if S=0	None
8.	JPE address	Branch	3	Pc=address if P=1	None
9.	JPO address	Branch	3	Pc=address if P=0	None

Pc: program counter. **Notice:** Jumps inst. Check the flags but not affect the flags

**Example:-**

Check if A+B = 0 then increment the result

```
MVI A, ...
MVI B, ...
ADD B
JNZ end
INR A
End: RST1
```

**Example:-**

Check if A+B = 50 then increment the result

```
MVI A, ...
MVI B, ...
ADD B
CPI 50
JNZ end
INR A
End: RST1
```

**Example:-**

Check if A = 0 then increment A

```
MVI A, ...
ANA A ; (/ ORA A/
        ; CPI 0/ ADI 0)
JNZ end
INR A
End: RST1
```

## Counters

Designing a counter is a frequent programming application. Counters are used primarily to keep track of events.

A counter is designed by loading an appropriate count in a register. A loop is set up decrement the count for a down-counter (counts in the descending order) by using the DCR (decrement by one) instruction or to increment the count for an up-counter (counts in the ascending order) by using the INR (increment by one) instruction. A loop is established to update the counter, and each count is checked to determine whether it has reached the final number; if not the loop is repeated.

### Examples:-

1-  $1 \geq$  No. of loops  $< 256$

#### **Using down-counter:-**

```
      MVI C,8
Loop: DCR C
      JNZ Loop
      RST1
```

#### **Using UP-counter:-**

```
      MVI C,0
Loop: INR C
      MOV A,C
      CPI 8
      JNZ Loop
      RST1
```

2- No. of loops = 256

#### **Using down-counter:-**

```
      MVI C,0
Loop: DCR C
      JNZ Loop
      RST1
```

No. of loops  $\geq 256$

#### **Using UP-counter:-**

```
      LXI B,0
Loop: INX B
      MOV A,C
      CPI lowbyteno
      JNZ Loop
      MOV A,B
      CPI highbyteno
      JNZ loop
      RST1
```

3- No. of loops  $> 256$

#### **Using down-counter:-**

```
      LXI B, 3b5h
Loop: DCX B ; not effect Z flag
; To affect the Z flag and check if B or
C is not finished then continue the loop
      MOV A,C
      ORA B
      JNZ Loop
      RST1
```

**Note:-** You can see that down-counter is preferred to use than up-counter.

### Useful instructions:

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

Instruction	Type	No. of Bytes	Function	Effect
PCHL	Branch	1	PC=HL	None

### LAB Work

- 1- exclusive or register A and B, then add 3 to register C if the parity is even otherwise add 30h to C, when A=35h, B=20h, C=10h
- 2- Check if the content of register B is even then C=1, otherwise C=2. (by using two methods)
- 3- Calculate the sum of numbers between 10 and 1.

### Home Work

- 1- Compare the value of register A and B, then obtain the last value of A:  
A=A+10h when A=B  
A=A+5 when A>B  
A=A+20h when A<B
- 2- Calculate the result of (8\*8) by using two methods.
- 3- Calculate  $11/4=2$  and the remainder is 3
- 4- Calculate the result of  $C=B^2+5$  when B=5

## Exp.no. (14)

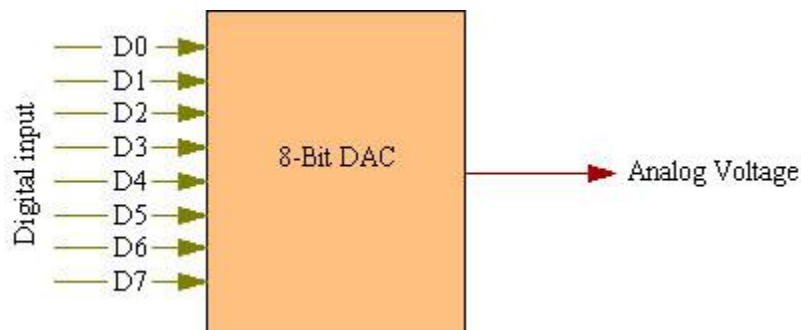
### Digital to Analog converters

#### OBJECT:-

To Interface Digital -to-Analog converter to 8085 using 8255 and write Assembly Language Program to generate Ramp Wave form.

#### THEORY:-

Digital-to-Analog Conversion or simply DAC, is a device that is used to convert a digital (usually binary) code into an analog signal (current, voltage, or electric charge). Digital-to-analog conversion is the primary means by which digital equipment such as computer-based systems are able to translate digital data into real-world signals that are more understandable to or useable by humans, such as music, speech, pictures, video. It also allows digital control of machines, equipment, household appliances. When data is in binary form, the 0's and 1's may be of several forms such as the TTL form where the logic zero may be a value up to 0.8 volts and the 1 may be a voltage from 2 to 5 volts. The data can be converted to clean digital form using gates which are designed to be on or off depending on the value of the incoming signal. Data in clean binary digital form can be converted to an analog form by using a summing amplifier. Here is a simplified functional diagram of an 8-bit DAC.



There are mainly two techniques used for digital to analog conversion

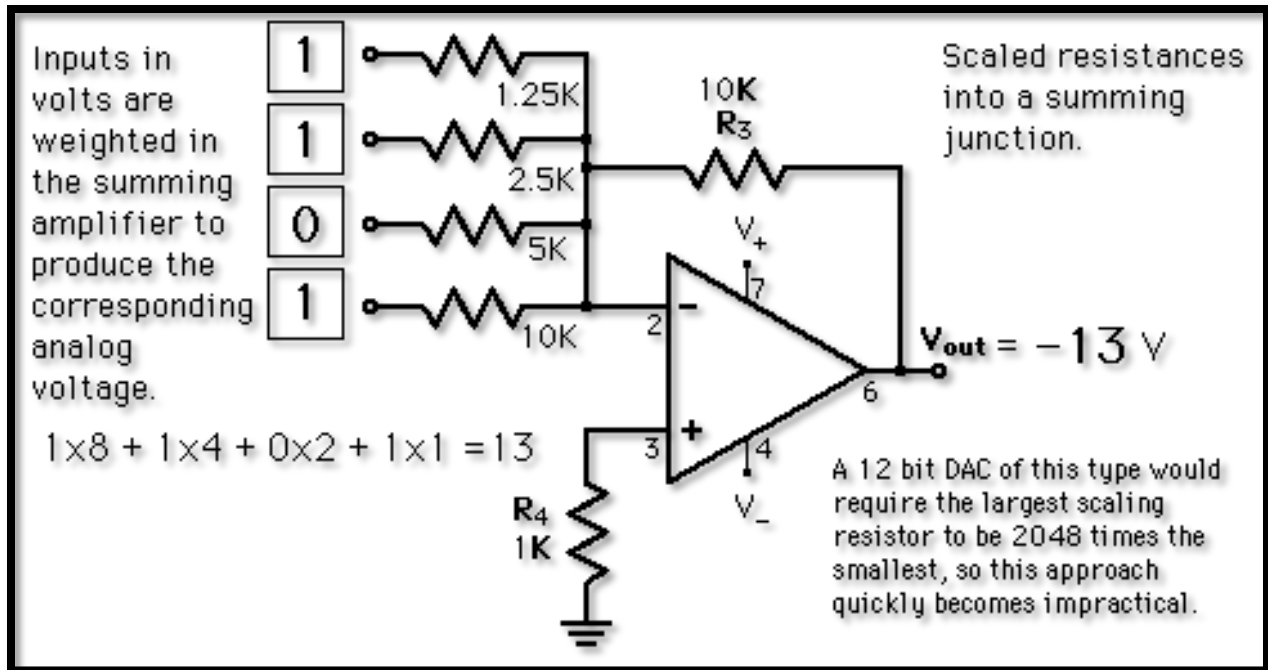
1. **Weighted Summing Amplifier**
2. **R-2R Network Approach**

### Weighted Sum DAC

One way to achieve D/A conversion is to use a summing amplifier.

This approach is not satisfactory for a large number of bits because it requires too much precision in the summing resistors.

This problem is overcome in the R-2R network DAC.

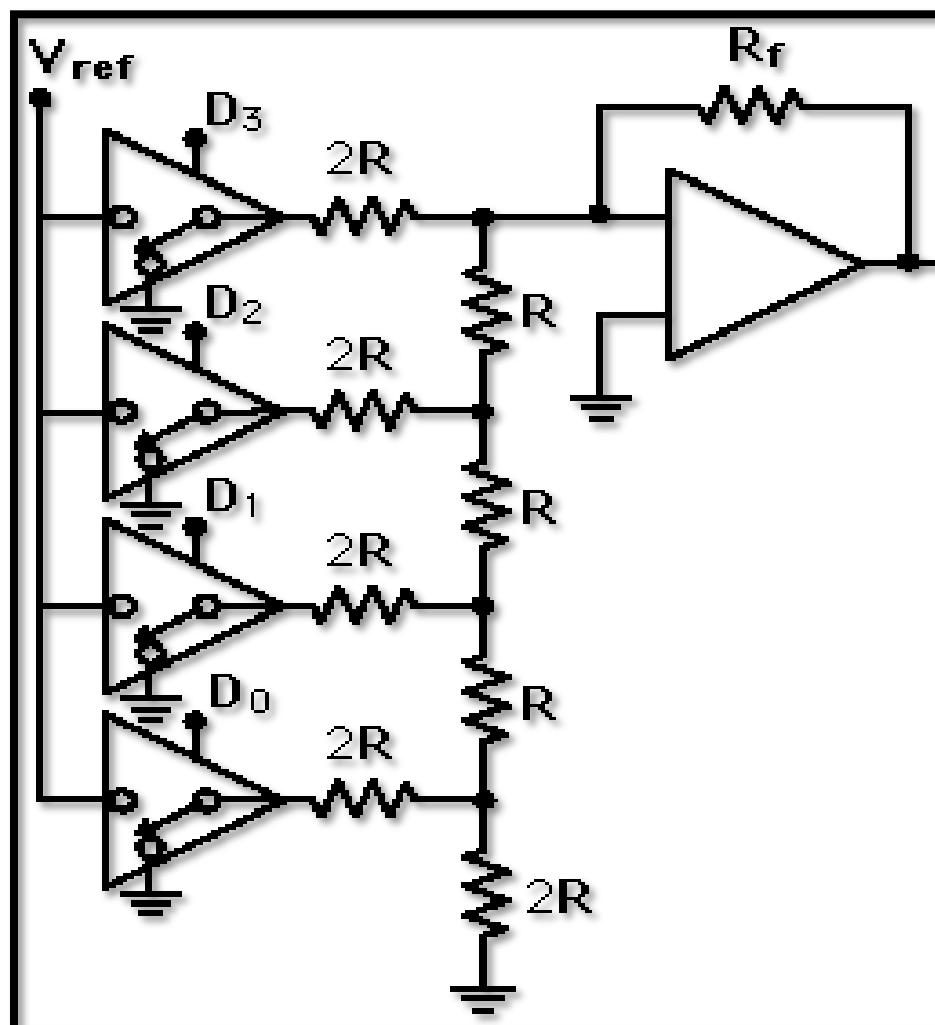


## R-2R Ladder DAC

The summing amplifier with the R-2R ladder of resistances shown produces the output where the D's take the value 0 or 1.

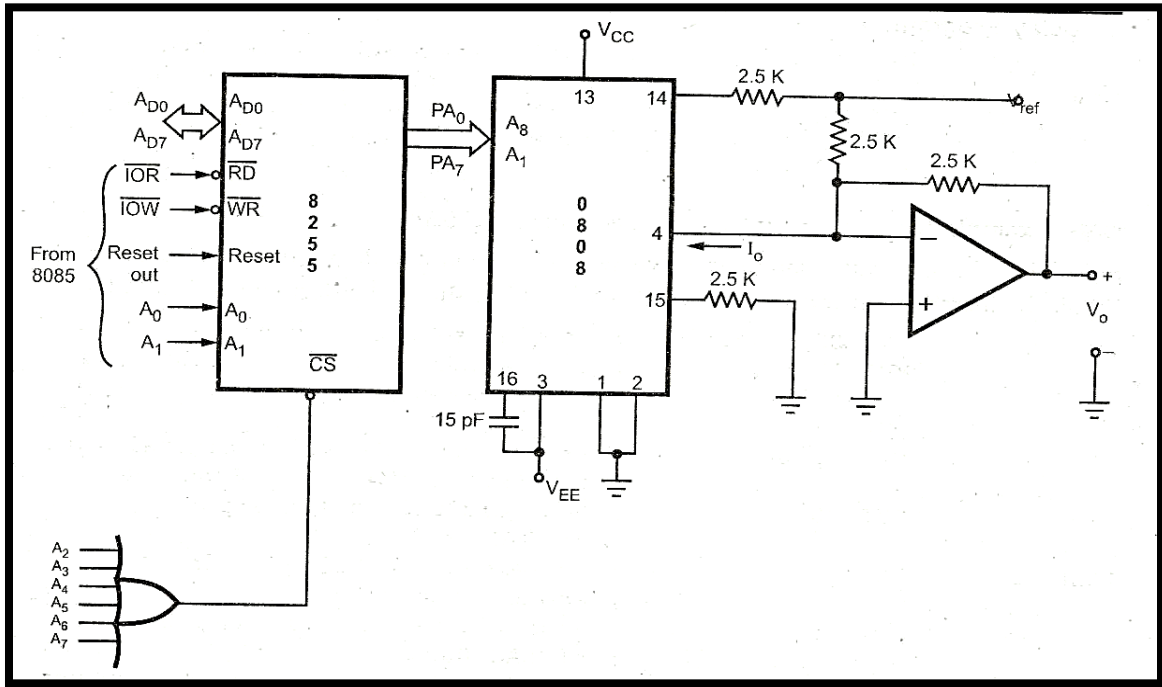
The digital inputs could be TTL voltages which close the switches on a logical 1 and leave it grounded for a logical 0.

This is illustrated for 4 bits, but can be extended to any number with just the resistance values R and 2R.



$$V_{out} = \frac{R_f}{R} V_{ref} \left[ \frac{D_0}{16} + \frac{D_1}{8} + \frac{D_2}{4} + \frac{D_3}{2} \right]$$

The interfacing of DAC 0808 with microprocessor 8085 is shown below. Here, programmable peripheral interface, 8255 is used as parallel port to send the digital data to DAC.





## Interfacing Digital-To-Analog converter to 8085 using 8255

Figure below shows the interfacing of DAC 0808 with microprocessor 8085. Here, programmable peripheral interface, 8255 is used as parallel port to send the digital data to DAC.

*I/O Map for 8255*

Port/Register	Address
Port A	00
Port B	01
Port C	02
Control Register	03

Program:

```
MVI A, 80H ; Initialization -control word for 8255 to Configure all ports as output ports
OUT 03

MVI A, DATA ; Load 8-bit data to be sent at the input of 0808 DAC
OUT 00 ; Send data on port A.
```

### A Circuit Description of DAC module

When chip select of DAC is enabled then DAC will convert digital input value given through portlines PB0-PB7 to analog value. The analog output from DAC is a current quantity. This current is converted to voltage using OPAMP based current-to-voltage converter. The voltage outputs (+/- 8V for bipolar 0 to 8V for unipolar mode) of OPAMP are connected to CRO to see the wave form. Port A & Port B are connected to channel 1 and channel 2 respectively. A reference voltage of 8V is generated using 723 and is given to Verify points of the DAC 0800. The standard output voltage will be 7.98V when FF is outputted and will be 0V when 00 is outputted. The Output of DAC-0800 is fed to the operational amplifier to get the final output as **X OUT** and **YOUT**.

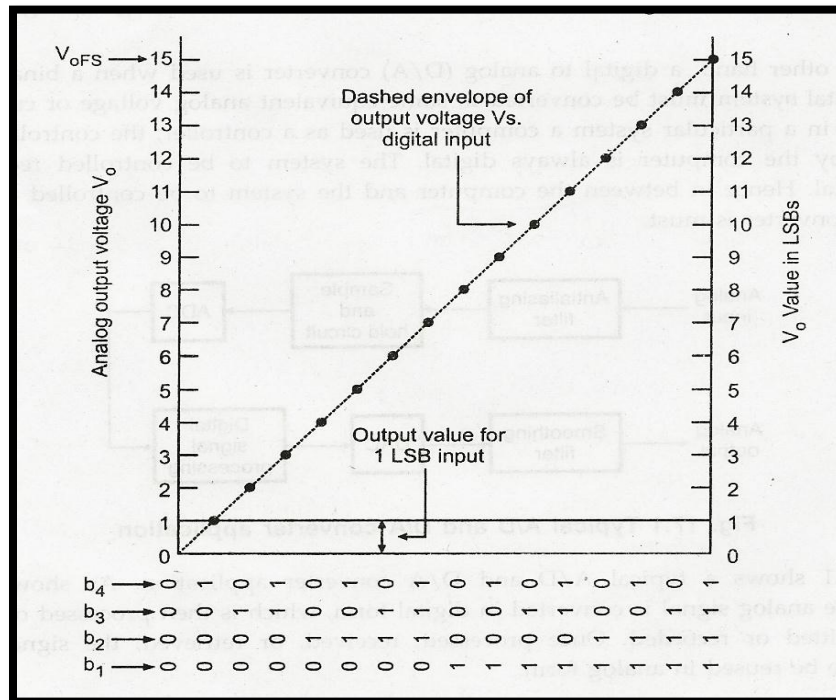


Figure shows analog output voltage  $v_o$  is plotted against all 16 possible digital input words.

### Performance Parameters of DAC:

The performance parameters of DAC are:

#### 1. Resolution

Resolution is defined in two ways.

- o Resolution is the number of different analog output values that can be provided by a DAC. For an n-bit DAC

$$\text{Resolution} = 2^n \dots\dots\dots (1)$$

- o Resolution is also defined as the ratio of a change in output voltage resulting from a change of 1 LSB at the digital inputs. For an n-bit DAC it can be given as:

$$\text{Resolution} = \text{VoFs} / 2^n - 1 \dots\dots\dots(2)$$

Where, VoFs = Full scale output voltage

From equation(1), we can say that, the resolution can be determined by the number of bits in the input binary word. For an 8-bit resolution can be given as

$$\text{resolution} = 2^n = 2^8 = 256$$

If the full scale output voltage is 10.2 V then by second definition the resolution for an 8-bit can be given as

$$\begin{aligned} \text{Resolution} = \text{VoFs} / 2^n - 1 &= 10.2 / 2^8 - 1 = 10.2 / 255 \\ &= 40 \text{ mV/LSB} \end{aligned}$$

Therefore, we can say that an input change of 1 LSB causes the output to change by 40 mv

## 2. Accuracy

It is a comparison of actual output voltage with expected output. It is expressed in percentage. Ideally, the accuracy of DAC should be, at worst,  $\pm 1/2$ , of its LSB. If the full scale output voltage is 10.2 V then for an 8-bit DAC accuracy can be given as

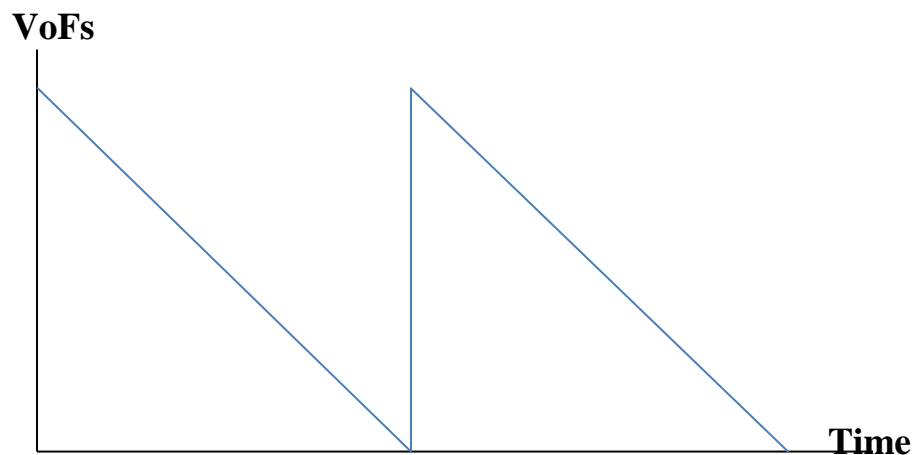
$$\begin{aligned} \text{Accuracy} &= \text{VoFs} / (2^n - 1)_2 \\ &= 10.2 / 255 \times 2 = 20 \text{ mV} \end{aligned}$$

## **PROCEDURE:-**

1. Connect power supply 8V & GND to both microprocessor trainer kit & DAC interfacing kit.
2. Connect data bus between microprocessor trainer kit & DAC interfacing kit.
3. Enter the program to generate Ramp Wave.
4. Execute your program from respective locations and observe the waveform on oscilloscope.

## **HOME WORK**

- **An 8 bit DAC has an output voltage range of 0 – 2.55 V. Define its resolution in two ways.**
- **A 12-bit DAC has a step size of 8 mv. Determine the full scale output voltage and percentage resolution.**
- **Write a program to output this signal**



## **Programmable Peripheral Interface-8255:-**

The 8255 is a general purpose programmable I/O device used for parallel data transfer. It has 24 I/O pins which can be grouped in three 8-bit parallel ports: Port A, Port B and Port C. The eight bits of port C can be used as individual bits or be grouped in two 4-bit ports: Copper ( $C_u$ ) and C lower, (CL).

The 8255, primarily, can be programmed in two basic modes Bit Set/Reset (BSR) mode and I/O mode. The BSR mode is used to set or reset the bits in port C.

### **The I/O mode is further divided into three modes:**

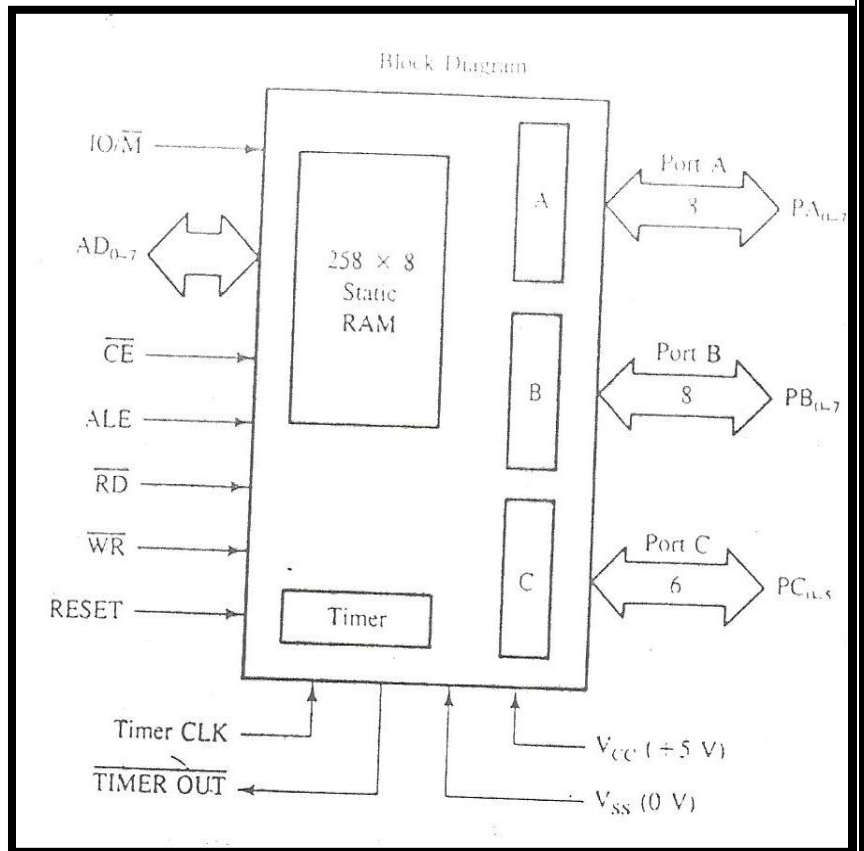
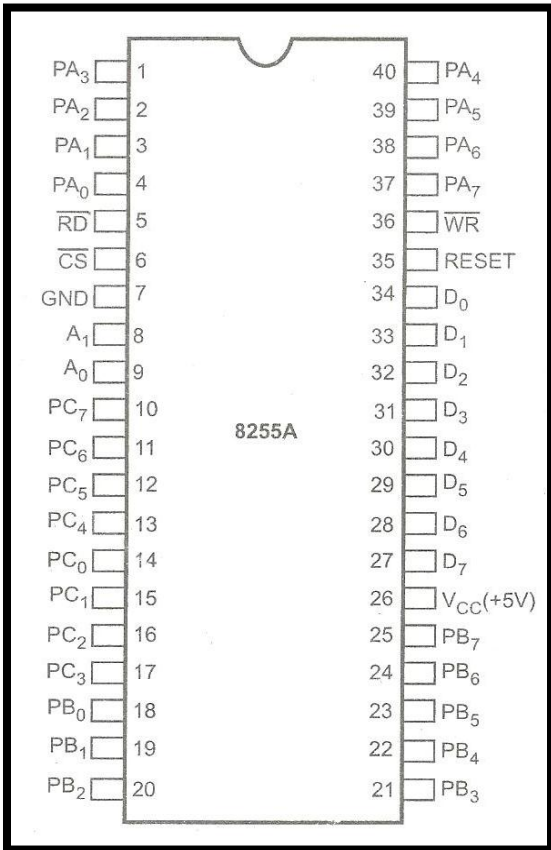
Mode0: Simple Input/output.

Mode1: Input/output with handshake.

Mode2: Bi-directional I/O data transfer.

The function of I/O pins (input or output) and modes of operation of I/O ports can be programmed by writing proper control word in the control word register. Each bit in the control word has a specific meaning and the status of these bits decides the function and operating mode of the I/O ports.

# PIN Diagram:



Pin Symbols	Function
D <sub>0</sub> -D <sub>7</sub> (Data Bus)	These bi-directional, tri-state data bus lines are connected to the system data bus. They are used to transfer data and control word from microprocessor (8085) to 8255 or to receive data or status word from 8255 to the 8085.
PA <sub>0</sub> -PA <sub>7</sub> (Port A)	These 8-bit bi-directional I/O pins are used to send data to output device and to receive data from input device. It functions as an 8-bit data output latch/buffer, when used in output mode and an 8-bit data input buffer, when used in input mode.
PB <sub>0</sub> -PB <sub>7</sub> (Port B)	These 8-bit bi-directional I/O pins are used to send data to output device and to receive data from input device. It functions as an 8-bit data, output latch/buffer when used in output mode and an 8-bit data input buffer, when used in input mode.
PC <sub>0</sub> -PC <sub>7</sub>	These 8-bit bi-directional I/O pins are divided into two groups PC <sub>L</sub> (PC <sub>3</sub> -PC <sub>0</sub> ) and PC <sub>U</sub> (PC <sub>7</sub> -PC <sub>4</sub> ). These groups individually can transfer data in or out when programmed for simple I/O, and used as handshake signals when programmed for handshake or bi-directional modes.
$\overline{\text{RD}}$ (Read)	When this pin is low, the CPU can read the data in the ports or the status word, through the data buffer.
$\overline{\text{WR}}$ (Write)	When this input pin is low, the CPU can write data on the ports or in the control register through the data bus buffer.
$\overline{\text{CS}}$ (Chip Select)	This is an active low input which can be enabled for data transfer operation between the CPU and the 8255.
RESET	This is an active high input used to reset 8255. When RESET input is high, the control register is cleared and all the ports are set to the input mode. Usually RESET.OUT signal from 8085 is used to reset 8255.
A <sub>0</sub> and A <sub>1</sub>	These input signals along with $\overline{\text{RD}}$ and $\overline{\text{WR}}$ inputs control the selection of the control/status word registers or <u>one of the three</u> ports. Table. 12.15 summarizes the status of A <sub>0</sub> , A <sub>1</sub> , $\overline{\text{CS}}$ , $\overline{\text{RD}}$ and $\overline{\text{WR}}$ to access the control word/ports. A <sub>0</sub> and A <sub>1</sub> are generally connected to the A <sub>0</sub> , A <sub>1</sub> pins of the address bus; the 8255 therefore occupies four consecutive locations in the I/O space.

## For I/O Mode

The mode definition format for I/O mode is shown in Figure below. The control words for both, mode definition and Bit Set-Reset are loaded into the same control register, with bit D7 used for specifying whether the word loaded into the control register is a mode definition word or Bit Set-Reset word. If D7 is high, the word is taken as a mode definition word, and if it is low, it is taken as a Bit Set-Reset word. The appropriate bits are set or reset depending on the type of operation desired, and loaded into the control register.

