MADHA ENGINEERING COLLEGE

(A Christian Minority Institution)

KUNDRATHUR, CHENNAI – 600 069

Electrical Machines Lab – II Manual

Name	:	
Subject	:	
Roll No.	:	
Semester	:	**Year:**

CONTENTS

Sl. No.	Name of the Experiment	Page No.
1	Load Test on Single Phase Induction Motor	
2	Load Test on Three Phase Squirrel Cage Induction Motor	
3	Regulation of Alternator by Synchronous Impedance (or) EMF Method	
4	Regulation of Alternator by Ampere-Turn (or) MMF Method	
5	No Load & Blocked Rotor Test on Single Phase Induction Motor	
6	Load Test on Three Phase Slip Ring Induction Motor	
7	V and inverted V curves of Synchronous motor	
8	Separation of No Load losses of 3Φ Induction motor	
9	No Load & Blocked Rotor Test on three phase squirrel cage Induction motor – Circle Diagram	
10	Study of Induction motor Starters	

1. Load Test on Single Phase Induction Motor

Aim:

To conduct load test on single-phase induction motor and to draw its performance characteristics

Apparatus required:

S. No.	Apparatus Name	Type / Rating	Quantity
1	Ammeter	(0 - 10A) MI	1
2	Voltmeter	(0 - 300V) MI	1
3	Wattmeter	300 V / 10A, UPF	1
4	1ϕ Auto Transformer	(0-270) V	1
5	Tachometer	Analog	1

Precautions:

(1) All the switches should be kept open initially

(2) The motor should be started and stopped without any load on the brake drum.

(3) Brake drum should be cooled with water during the entire test.

Theory:

Single-phase induction motor is not a self-starting one. To over come this draw back and make the motor self-starting, it is temporarily converted into two-phase motor during starting period. For this purpose an extra winding known as starting winding is added. One capacitor C and one centrifugal switch S are connected in series with the starting winding. The purpose of the capacitor is to provide the phase difference between the two currents (starting winding current and running winding current). The purpose of the centrifugal switch is to disconnect the starting winding from the supply, once the motor reaches 70 to 80 % of its rated speed. The currents (I_S and I_R) produce a revolving flux and hence make the motor self-starting.

Procedure:

The connections are given as shown in the circuit diagram. The DPST switch is closed. The motor is started using DOL (Direct On Line) starter. The input voltage is adjusted to rated value with the help of single phase auto transformer. Now the motor runs at a speed closure to the synchronous speed. The no-load readings of ammeter, voltmeter, wattmeter and speed of the motor are noted. The load on the brake drum is increased in suitable steps and the corresponding readings are noted.

Graphs:

(1) Output power Vs Torque
Vs Speed
Vs Efficiency

(2) Slip Vs Torque

Tabulation and Readings:

Voltage V volts	Current I amps	Spring balance		Speed N rpm	Torque N-m	Input P_i watts	Output P_m watts	η %	% slip s	Power factor
		S_1 kg	S_2 kg							

Model Calculations: (3rd set of readings)

Circumference of brake drum 2*π*R = ------ m

Radius of brake drum R = -----/ 2*π m

(1) Torque $T = (s_1 \sim s_2) * 9.81 * R$ N-m

(2) Input power P_i = (Wattmeter reading) x M.F (Multiplication Factor) watts

(3) Output power $P_m = 2 * \pi * N * T / 60$ watts

(4) Efficiency $\eta = (P_m / P_i) \times 100$ %

(5) % Slip $s = (N_S - N) / N_S \times 100$ $N_S = 1500$ rpm

(6) Power factor = Input power (P_i) / V I

Result:

Thus, the load test on single-phase induction motor is conducted and its performance characteristics are drawn.

2. Load Test on Three Phase Squirrel Cage Induction Motor

Aim:

To conduct the load test on three phase squirrel cage induction motor and to draw its performance characteristics.

Apparatus Required:

S. No.	Apparatus	Type / Rating	Quantity
1	Ammeter	(0 - 10A) MI	1
2	Voltmeter	(0 - 600V) MI	1
3	Wattmeter	600V / 10A, UPF	2
4	Tachometer	Analog	1
5	Connecting Wires	---	As Required

Precautions:

(1) All the switches are kept open initially.

(2) The motor should be started and stopped without any load on the Brake drum.

(3) The brake drum should be cooled with water during the entire test.

Theory:

The induction motors are basically AC motors. i.e. they need an alternating voltage for their operation. They can operate on either single phase or three phase as supply, however the single phase induction motors find very limited area of application. Almost 85% of industrial motors are three phase induction motors. Depending on the type of rotor, the induction motors are classified into two types, (i) slip ring induction motor (ii) squirrel cage induction motors.

The three phase stator winding of induction motor is connected to the three phase AC supply. Due to AC voltage applied, current stars flowing in the stator conductors. Due to the three phase stator current, a rotating magnetic field of constant amplitude and rotating at a constant speed is set up in the air gap between stator and rotor. The rotating magnetic field rotates at a speed called as synchronous speed (Ns)

The synchronous speed is given by

$$N_s = \frac{120f}{p}$$

Where

f - Stator Supply Frequency,

P - Number of Poles

This rotating magnetic field (RMF) interacts with the rotor and produces rotation.

Procedure:

The connections are given as shown in the circuit diagram. The TPST switch is closed. The motor is started using DOL (Direct On Line) starter. Now, the motor runs at a speed closure to the synchronous speed. The no-load readings of ammeter, voltmeter, wattmeter and speed of the motor are noted. The load on the brake drum is increased in suitable steps and the corresponding readings are noted.

Graphs:

(1) Output power Vs Torque

Vs Speed

Vs Efficiency

Vs Line Current

(3) Slip Vs Torque

Tabulation and Readings:

Line Voltage V_L volts	Line Ct. I_L Amps	Spring balance readings		Speed N rpm	Wattmeter readings		Torque in N-m	Input Power P_i W	Output P_m watts	η %	% slip % s	Power Factor Cos ϕ
		S_1 kg	S_2 kg		W_1	W_2						

Model Calculations: (3rd set of readings)

Circumference of brake drum $2*\pi*R$ = ------ m

Radius of brake drum R = ----- / $2*\pi$ m

(1) Torque $T = (s_1 \sim s_2)*9.81*R$ N-m

(2) Input power $P_i = W_1 + W_2$ watts

(3) Output power $P_m = 2*\pi*N*T / 60$ watts

(4) Efficiency $\eta = P_m / P_i \times 100$ %

(5) % Slip $s = (N_S - N)/N_S \times 100$ $N_S = 1500$ rpm

(6) Power factor = Input power / $\sqrt{3}\, V_L I_L$

Result:

Thus, the load test on three-phase squirrel cage induction motor is conducted and its performance characteristics are drawn.

3. Regulation of Alternator by Synchronous Impedance (or) EMF Method

Aim:

To determine the regulation of a three phase alternator by synchronous impedance method.

Apparatus required:

S. No.	Apparatus	Type / Rating	Quantity
1	Ammeter	(0 - 10 A) MI	1
2	Ammeter	(0 - 2 A) MI	1
3	Voltmeter	(0 - 600V) MI	1
4	Rheostat	250 Ω, 1.5 A	1
5	Rheostat	400 Ω, 1 A	1
6	Tachometer	Analog	1
7	Connecting Wires	---	As Required

Precautions:

(1) All the switches are kept open initially.

(2) The motor field rheostat should be kept at minimum position at the time of starting and stopping.

(3) Alternator field rheostat should be kept at maximum position at the time of starting and stopping.

Theory:

The voltage regulation of an alternator is defined as the change in terminal voltage from no-load to the load concerned as a percentage of the rated terminal voltage when the field excitation and speed remains constant.

$$\% \text{ regulation} = (E_0 - V) / V \times 100$$

where

E_0 - Terminal voltage on no-load

V - Terminal voltage on load

This method requires the following characteristics

(1) Open circuit characteristics

(2) Short circuit characteristics

(3) Armature resistance

Armature resistance can be found by using either multi meter or by voltmeter-ammeter method. In the EMF method, the armature reaction is treated along with leakage reactance. But in the MMF method, leakage reactance is treated as an additional armature reaction.

Procedure:

(1) Open circuit test:

The connections are given as shown in the circuit diagram. The DPST switch is closed and the motor is started using 3 point starter. The speed of the motor is adjusted to rated speed by varying the motor field rheostat. The generator field DPST switch is closed. For various values of excitation current (Field current), the induced EMF is noted.

(2) Short circuit test:

The connections are given as shown in the circuit diagram. The DPST switch is closed and the motor is started using 3 point starter. The speed of the motor is adjusted to rated speed by varying the motor field rheostat. Now, the generator field DPST switch and TPST switch are closed. The field current is increased till the ammeter reads rated current. The field current and Short circuit current are noted and the motor alternator set is disconnected from the supply.

O.C. Test:

Sl. No.	Field current I_f amps	Line voltage V_L volts	Phase voltage $V_{ph} = V_L/\sqrt{3}$
1			
2			
3			
4			
5			
6			
7			
8			
9			

S.C. Test:

Field current I_f amps	Short circuit current I_{sc} amps

Tabulation:

Sl. No.	Power factor angle in degrees	Power factor	Induced EMF E_O volts	% regulation
	Leading p.f.			
1	30	0.866		
2	45	0.707		
3	60	0.5		
	Lagging p.f.			
1	30	0.866		
2	45	0.707		
3	60	0.5		

Model Calculations:

Synchronous impedance Z_S = Open circuit voltage / Short circuit current

$= E_1 / I_1\ \Omega$

R_a = ---- Ω (using multimeter)

$X_S = \sqrt{Z_S^2 - R_a^2}\ \Omega$

For lagging power factor,

$$E_O = \sqrt{(V \cos \Phi + I R_a)^2 + (V \sin \Phi + I X_s)^2}$$

$$\% \text{ reg} = (E_O - V) / V \times 100$$

For leading power factor,

$$E_O = \sqrt{(V \cos \Phi + I R_a)^2 + (V \sin \Phi - I X_s)^2}$$

$$\% \text{ reg} = (E_O - V) / V \times 100$$

For unity power factor,

$$E_O = \sqrt{(V \cos \Phi + I R_a)^2 + (I X_s)^2}$$

$$\% \text{ reg} = (E_O - V) / V \times 100$$

Result:

Thus the regulation of alternator is predetermined by synchronous impedance (EMF) method.

4. Regulation of Alternator by Ampere -Turn (or) MMF Method

Aim:

To determine the regulation of a three phase alternator by Ampere-turn method.

Apparatus required:

S. No.	Apparatus	Type / Rating	Quantity
1	Ammeter	(0 - 10 A) MI	1
2	Ammeter	(0 - 2 A) MI	1
3	Voltmeter	(0 - 600V) MI	1
4	Rheostat	250 Ω, 1.5 A	1
5	Rheostat	400 Ω, 1 A	1
6	Tachometer	Analog	1
7	Connecting Wires	---	As Required

Precautions:

(1) All the switches are kept open initially.

(2) The motor field rheostat should be kept at minimum position at the time of starting and stopping.

(3) Alternator field rheostat should be kept at maximum position at the time of starting and stopping.

Theory:

To determine the voltage regulation of an alternator by Ampere-turn method, it is necessary to perform open circuit test and short circuit test. The open circuit test is conducted by allowing the alternator to run on no-load at rated speed. The terminal voltage of the alternator on no-load is measured at various values of excitation current. The graph drawn between no-load voltage along Y-axis and field current along X-axis gives the open circuit characteristics of the alternator.

The short circuit test is conducted on the alternator at its rated speed. The output terminals are short-circuited using one ammeter and the excitation current is increased till the ammeter reads rated current. The graph is drawn between the short circuit current along Y-axis and the field current along X-axis.

Procedure:

(1) Open circuit test:

The connections are given as shown in the circuit diagram. The DPST switch is closed and the motor is started using 3 point starter. The speed of the motor is adjusted to rated speed by varying the motor field rheostat. The generator field DPST is closed. For various values of excitation current (Field current), the induced EMF is noted.

(2) Short circuit test:

The connections are given as shown in the circuit diagram. The DPST switch is closed and the motor is started using 3 point starter. The speed of the motor is adjusted to rated speed by varying the motor field rheostat. Now, the generator field DPST switch and TPST switch are closed. The field current is increased till the ammeter reads rated current. The field current and short circuit current are noted and the motor alternator set is disconnected from the supply.

O.C. Test:

Sl. No.	Field current I_f amps	Line voltage V_L volts	Phase voltage $V_{ph} = V_L/\sqrt{3}$
1			
2			
3			
4			
5			
6			
7			

S.C. Test:

Field current I_f amps	Short circuit current I_{sc} amps

Tabulation:

Sl. No.	Power factor angle in degrees	Power factor	Induced EMF E_O volts	% regulation
	Leading p.f.			
1	30	0.866		
2	45	0.707		
3	60	0.5		
	Lagging p.f.			
1	30	0.866		
2	45	0.707		
3	60	0.5		

Model Calculations:

% regulation = $(E_0 - V) / V \times 100$

E_0 - Terminal voltage on no-load
V - Terminal voltage on load

Result:

Thus the regulation of alternator is predetermined by Ampere-turn (MMF) method.

5. No Load and Blocked Rotor Test on Single Phase Induction Motor

Aim:

To draw the Equivalent circuit of single phase Induction motor by conducting no load and blocked rotor test.

Apparatus required:

S. No.	Apparatus	Type / Rating	Quantity
1	Ammeter	(0 - 10 A) MI	1
2	Voltmeter	(0 - 150 V) MI	1
3	Voltmeter	(0 - 300 V) MI	1
4	Wattmeter	150 V / 10 A, UPF	1
5	Wattmeter	300 V / 10 A, UPF	1
6	1ϕ Auto Transformer	(0 - 270) V	1
7	Connecting Wires	---	As Required

Precautions:

(1) The Autotransformer should be kept at minimum position initially.

(2) During Blocked rotor test the rotor should not be allowed to rotate.

Procedure:

(1) No load test:

The connections are given as shown in the circuit diagram. Rated voltage is applied to the motor, by varying the autotransformer. The ammeter, voltmeter and wattmeter reading are noted.

(2) Blocked rotor test:

The connections are given as shown in the circuit diagram. The current should be set to the rated value by varying the autotransformer. The readings of voltmeter and wattmeter are noted.

Tabulation and Readings:

(1) No Load Test:

No Load Voltage V_o Volts	No Load Current I_o Amps	No Load Power W_o Watts	
		Observed	Actual

No Load Power W_o (Actual) = Observed Reading x MF (Multiplication Factor)

(2) Blocked Rotor Test:

Blocked Rotor Voltage V_b Volts	Blocked Rotor Current I_b Amps	Blocked Rotor Power W_b Watts	
		Observed	Actual

Blocked Rotor Power W_b (Actual) = Observed x MF (Multiplication Factor)

Equivalent Circuit Parameters from No Load Test:

No load Wattmeter reading $W_0 = V_0 I_0 \cos\phi_0$

No load Power Factor, $\cos\phi_0 = W_0 / V_0 I_0$

$=$

Loss Component of no load current, $I_w = I_0 \cos\phi_0$

$=$

Magnetizing Component of no load current, $I_m = I_0 \sin\phi_0$

$=$

Resistance to account for the iron loss, $R_0 = V_0 / I_w$

$=$

Reactance to account for the magnetization, $X_0 = V_0 / I_m$

=

Equivalent Circuit Parameters from Blocked Rotor Test:

Equivalent Impedance per phase referred to stator , Z_{01} = V_b / I_b

=

Equivalent Resistance per phase referred to stator , R_{01} = W_b / I_b^2

=

Equivalent leakage Reactance per phase referred to stator , X_{01} = $\sqrt{Z_{01}^2 - R_{01}^2}$

=

Stator winding Resistance per phase, R_1 = (using multi meter)

Rotor resistance per phase referred to stator , R_2' = $R_{01} - R_1$

=

X_1 and X_2' are assumed equal, then $X_1 = X_2' = X_{01} / 2$ =

Equivalent circuit of single phase Induction motor:

Result:

Thus, the equivalent circuit of single phase Induction motor is drawn by conducting no load and blocked rotor test.

6. Load Test on Three Phase Slip Ring Induction Motor

Aim:

To conduct the load test on three phase slip ring induction motor and to draw its performance characteristics.

Apparatus required:

S. No.	Apparatus	Type / Rating	Quantity
1	Ammeter	(0 - 10A) MI	1
2	Voltmeter	(0 - 600V) MI	1
3	Wattmeter	600V / 10A, UPF	2
4	Tachometer	Analog	1
5	Connecting Wires	---	As Required

Precautions:

(1) The motor should be started and stopped without any load on the brake drum.

(2) The brake drum should be cooled with water during the entire test.

Theory:

When 3ϕ supply is given to the stator of a 3-phase induction motor a rotating magnetic field (RMF) is produced which rotates at synchronous speed. This revolving flux sweeps over the rotor conductors, an EMF is produced in the rotor by Faraday's laws of electromagnetic induction. In order to reduce the relative speed between the rotor and the rotating magnetic flux, the rotor starts rotating in the same direction as that of stator flux with a speed, which is less than the synchronous speed. This difference in speed is called slip speed.

Procedure:

The connections are given as shown in the circuit diagram. The TPST switch is closed. The motor is started using Auto transformer starter. The rotor resistance switch is moved from maximum to minimum position. Now, the motor runs at a speed closure to the synchronous speed. The no-load readings of ammeter, voltmeter, wattmeter and speed of the motor are noted. The load on the brake drum is increased and the corresponding readings are noted.

Graphs:

(1) Output power Vs Torque
Vs Speed
Vs Efficiency
Vs Line Current

(2) Slip Vs Torque

Line Voltage V_L volts	Line Ct. I_L Amps	Spring balance readings		Speed N rpm	Wattmeter readings		Torque in N-m	Input Power P_i W	Output P_m watts	η %	% slip s	Power Factor Cos ϕ
		S_1 kg	S_2 kg		W_1	W_2						

Model Calculations:

Circumference of brake drum 2*π*R = ------ m

Radius of brake drum R = -----/ 2*π m

(1) Torque $T = (s_1 \sim s_2) * 9.81 * R$ N-m

(2) Input power $P_i = W_1 + W_2$ watts

(3) Output power $P_m = 2 * \pi * N * T / 60$ watts

(4) Efficiency $\eta = P_m / P_i \times 100$ %

(5) % Slip $s = (N_S - N)/N_S \times 100$ $N_S = 1500$ rpm

(6) Power factor = Input power / $\sqrt{3}\, V_L I_L$

Result:

Thus the load test on three-phase slip ring induction motor is conducted and its performance characteristics are drawn.

7. V and Inverted V curves of Synchronous motor

Aim:

To draw the V and inverted V curves of synchronous motor under no-load condition.

Apparatus required:

S. No.	Apparatus	Type / Rating	Quantity
1	Ammeter	(0 – 10 A) MI	1
2	Ammeter	(0 – 2 A) MC	1
3	Voltmeter	(0 – 600 V) MI	1
4	Wattmeter	600 V / 10 A, UPF	2
5	Rheostat	400 Ω / 1 A	1
6	Connecting Wires	---	As Required

Precautions:

(1) All the switches are kept opened initially.

(2) The potential divider should be kept at minimum position at the time of starting and stopping.

Theory:

When the excitation is normal, the power factor is unity and the armature current is minimum. For excitation greater than the normal excitation, the value of armature current increases and the power factor is leading. For excitation less than the normal excitation, the value of armature current also increases but the power factor is lagging. The curve between armature current and field current of a synchronous motor is called V curve. The curve between power factor and field current is known as inverted V curve.

Procedure:

The connections are given as per the circuit diagram. The TPST switch is closed. The motor is started by using DOL starter. The DPST switch on the field side is closed. The field current is varied by varying the field rheostat and the corresponding values of line voltage, line current and wattmeter readings are noted.

Tabulation:

Sl. No.	Line voltage V_L Volts	Line current I_L Amps	Field current I_L Amps	W_1 Watts	W_2 Watts	Power Factor Cos Φ

Model Calculation:

$\Phi = \tan^{-1}\sqrt{3}\,(W_2 - W_1) / (W_1 + W_2)$

Power factor = cos Φ

Graphs:

(i) Field Current Vs Armature Current

(ii) Field Current Vs Power Factor (cos Φ)

Result:

Thus the V and inverted V curves of synchronous motor were drawn.

8. Separation of No Load Losses of 3Φ Induction Motor

Aim:

To separate the no load losses of three phase Induction motor.

Apparatus required:

S. No.	Apparatus	Type / Rating	Quantity
1	Ammeter	(0 – 10 A) MI	1
2	Voltmeter	(0 – 600 V) MI	1
3	Wattmeter	600 V / 10A, UPF	2
4	Connecting Wires	---	As Required

Precautions:

(1) The Autotransformer should be kept at minimum position initially.

(2) The entire experiment should be conducted at No load.

Theory:

The no load losses are the constant losses which include core loss &friction and windage loss. The separation between the two can be carried out by no load test conducted from variable voltage, rated frequency supply.

When the voltage is decreased below the rated value, the core loss reduces as nearly square of voltage. The slip does not increase significantly and the friction and windage losses remain constant.

The voltage is reduced till the machine slip suddenly begins to increase and the motor tends to stall. At no load , this takes place at a sufficiently reduced voltage. The graph for power at no load Vs voltage is extrapolated to V=0 which gives friction and windage loss as iron or core loss as zero at zero voltage.

Procedure:

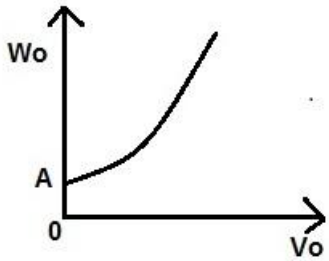
The connections are given as per the circuit diagram. The TPST switch is closed. The motor is started by using Autotransformer starter and set to the rated voltage. The no load voltmeter, ammeter and wattmeter readings are to be noted. Reduce the voltage gradually and note down the corresponding meter readings. From the readings taken draw the graph for no load power Vs voltage.

Tabulation:

Sl. No.	Line Voltage V_o Volts	Line Current I_o Volts	W_1 Watts	W_2 Watts	$W= W_1+ W_2$ Watts

Graphs:

No Load Power W_0 Vs Voltage V_0



From the graph, OA = Friction and Windage losses

The stator copper loss is given by

$$P_{SCL} = 3\ I_0^2 R_1$$

Where R_1 = Stator resistance per phase (use multi meter)

Then, the core loss of the induction motor is given by

$$\text{Core loss} = W_0 - P_{SCL} - \text{Friction and Windage losses}$$

Result:

Thus the no load losses of three phase Induction motor are separated.

9. No Load and Blocked Rotor Test on three phase squirrel cage Induction motor - Circle Diagram

Aim:

To draw the performance characteristics of three phase squirrel cage induction motor by no-load and blocked rotor test.

Apparatus required:

(1) Ammeter - (0 - 5) A, MI
(2) Voltmeter - (0 - 600) V, MI
(3) Voltmeter - (0 - 600) V, MI
(4) Watt meter - (600V/5A), LPF
(5) Watt meter - (600V/5A), LPF
(6) Watt meter - (150V/5A), UPF
(7) Watt meter - (150V/5A), UPF
(8) Tachometer
(9) 3Φ Autotransformer

Precautions:

(1) The autotransformer should be kept at minimum position while starting and stopping.
(2) During blocked rotor test, the rotor should not be allowed to rotate.

Theory:

To draw the circle diagram of a three phase induction motor, the following tests are to be performed in the motor

(1) No-load test
(2) Blocked rotor test

Using the data's obtained in the above tests, the circle diagram is drawn. From the circle diagram for various values of line current, the slip, input power, output power, torque and power factor are calculated. A graph is drawn by taking the output power in the X - axis and the remaining in the Y - axis.

Procedure:

(1) No load test:

The connections are given as shown in the circuit diagram. Rated voltage is applied to the motor, by varying the autotransformer. The ammeter, voltmeter and wattmeter reading are noted.

(2) Blocked rotor test:

The connections are given as shown in the circuit diagram. The ammeter reading is adjusted to rated value by varying the autotransformer. The readings of voltmeter and wattmeter are noted.

Test	Voltmeter reading in volts	Ammeter reading in amps	Wattmeter readings	
			W_1 watts	W_2 watts
O.C. test				
S.C. test				

Sl. No.	Line current I_L amps	Motor input Watts	Rotor output watts	Efficiency in %	% slip s	Power factor
1	3					
2	4					
3	5					
4	6					
5	7					
6	8					

Model Calculation:

From O.C. test,

$$W_0 = W_1 + W_2$$

$$\cos \phi_0 = W_0 / \sqrt{3}\, V_0 I_0$$

$$\phi_0 = \cos^{-1}(W_0 / \sqrt{3}\, V_0 I_0)$$

From S.C. test,

$W_s = W_1 + W_2$

$\cos \phi_s = W_s / \sqrt{3}\, V_s I_s$

$\phi_s = \cos^{-1}(W_s / \sqrt{3}\, V_s I_s)$

Short circuit current, when rated voltage is applied to the rotor

$I_{SN} = I_S \times V_{rated} / V_S$

Construction of circle diagram:

(1) Draw X and Y-axis. Take current scale 1 cm = 1 amps.

(2) Draw I_0 at an angle ϕ_0 from the origin. (Y-axis reference)

(3) Draw O'D parallel to X-axis as shown in figure.

(4) Locate point A at an angle ϕ_s and the length OA = I_{SN}.

(5) Join O and A. Draw perpendicular for the line O'A as shown in figure. Locate point C.

(6) Draw a semicircle with C as center and O'C as radius.

(7) Join AG as shown in figure.

(8) For various values of line current, find the efficiency, slip and power factor using the formula's given in the model calculation (cut the circle from the origin for 3,4,5,6,7,8 cm and join with the X-axis like LK in the diagram)

To locate point E:

AE / EF = Rotor copper loss / Stator copper loss

$= (W_s - 3I_s^2R_a) / 3I_s^2R_a$ *(AE = 1.67 EF)

Where R_a - armature resistance / phase in ohms

From the graph,

AF = AE + EF

EF = --- cm (sub AE interms of EF) (measure AF from graph)

Total power input = AG in cm

Blocked rotor input $y = W_{sc} \times (V_{rated} / V_s)^2$

Power scale = y / AG

Motor input	= LK x power scale
Rotor output	= ML x power scale
Efficiency	= ML / LK x 100
% Slip	= MN / NL x 100
Power factor	= LK / OL

Graphs:

(1) Output power Vs Efficiency

Vs % Slip

Vs Power factor

Result:

Thus the performance characteristics of three-phase induction motor are drawn from the circle diagram.

10. Study of Induction Motor Starters

Aim:

To study the different types of starters used for induction motors.

Apparatus Required:

S. No.	Apparatus	Type / Rating	Quantity
1	Star Delta Starter	---	1
2	Auto Transformer Starter	---	1
3	Direct on Line Starter	---	1

1. **Star Delta Starter:**

   Copy the diagram and theory from Electrical Machines II Class Notes

2. **Auto Transformer Starter:**

   Copy the diagram and theory from Electrical Machines II Class Notes

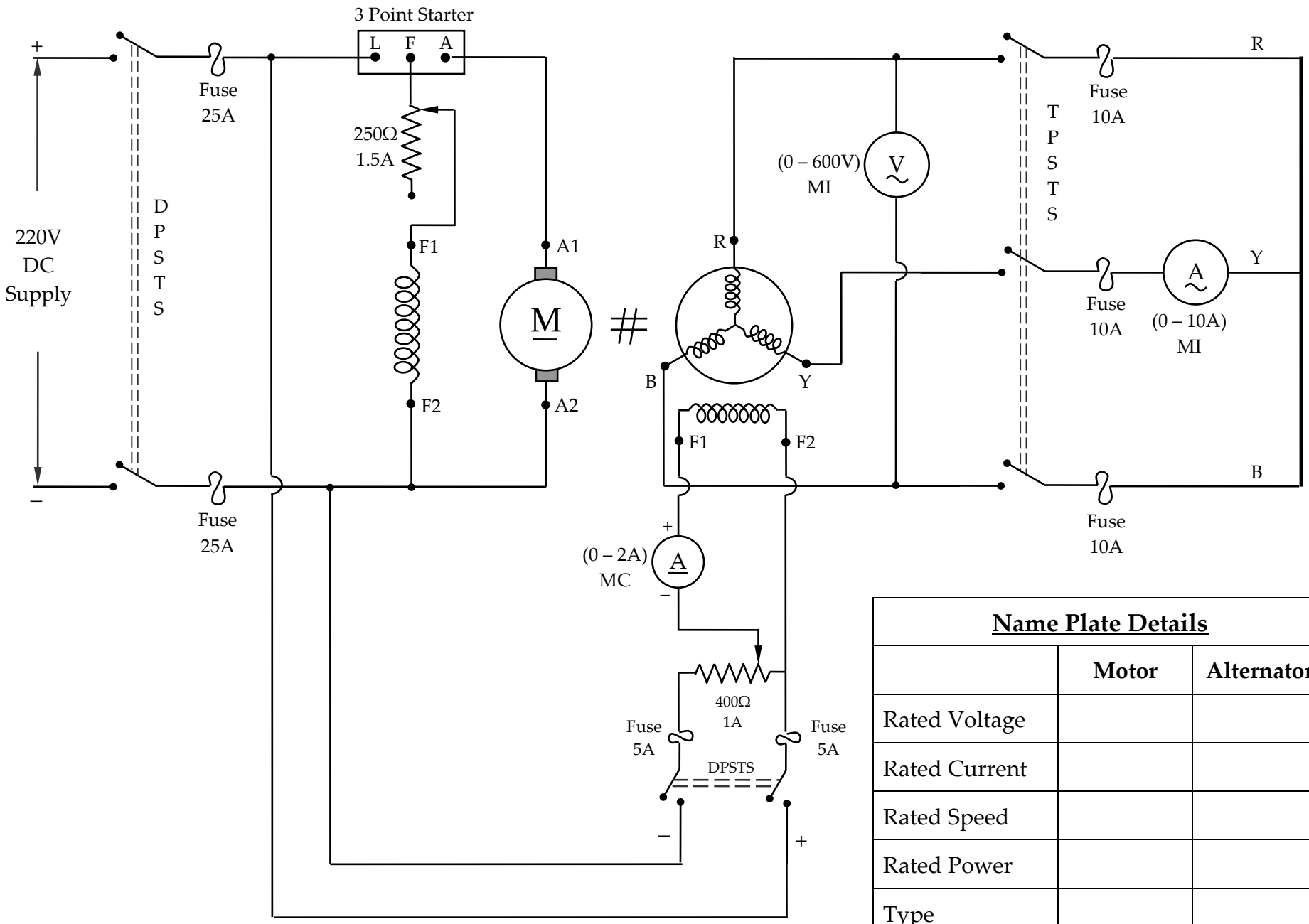
3. **Direct on Line Starter:**

   Copy the diagram and theory from Electrical Machines II Class Notes

Result:

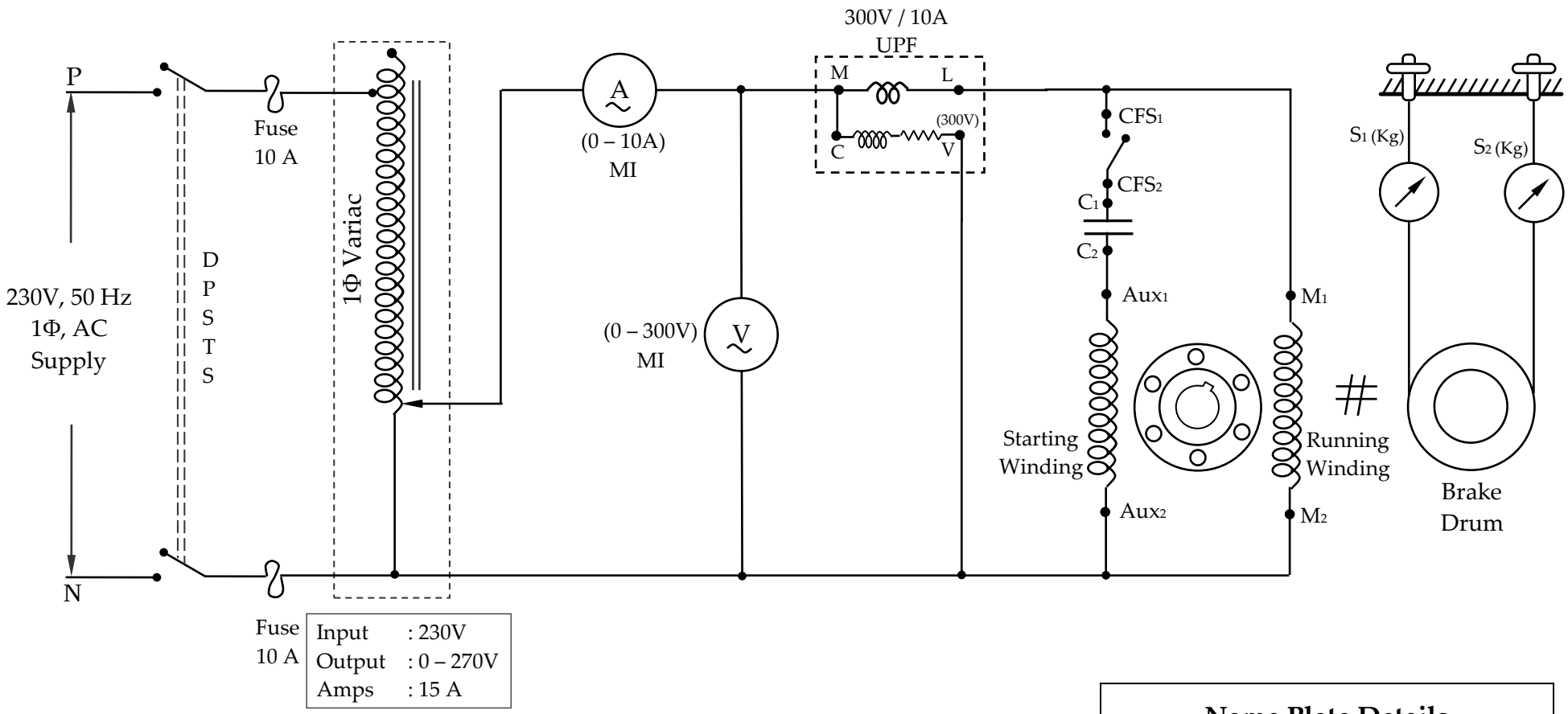
Thus the different types of starters used for induction motors were studied.

Circuit Diagram for Regulation of 3ϕ Alternator by MMF Method



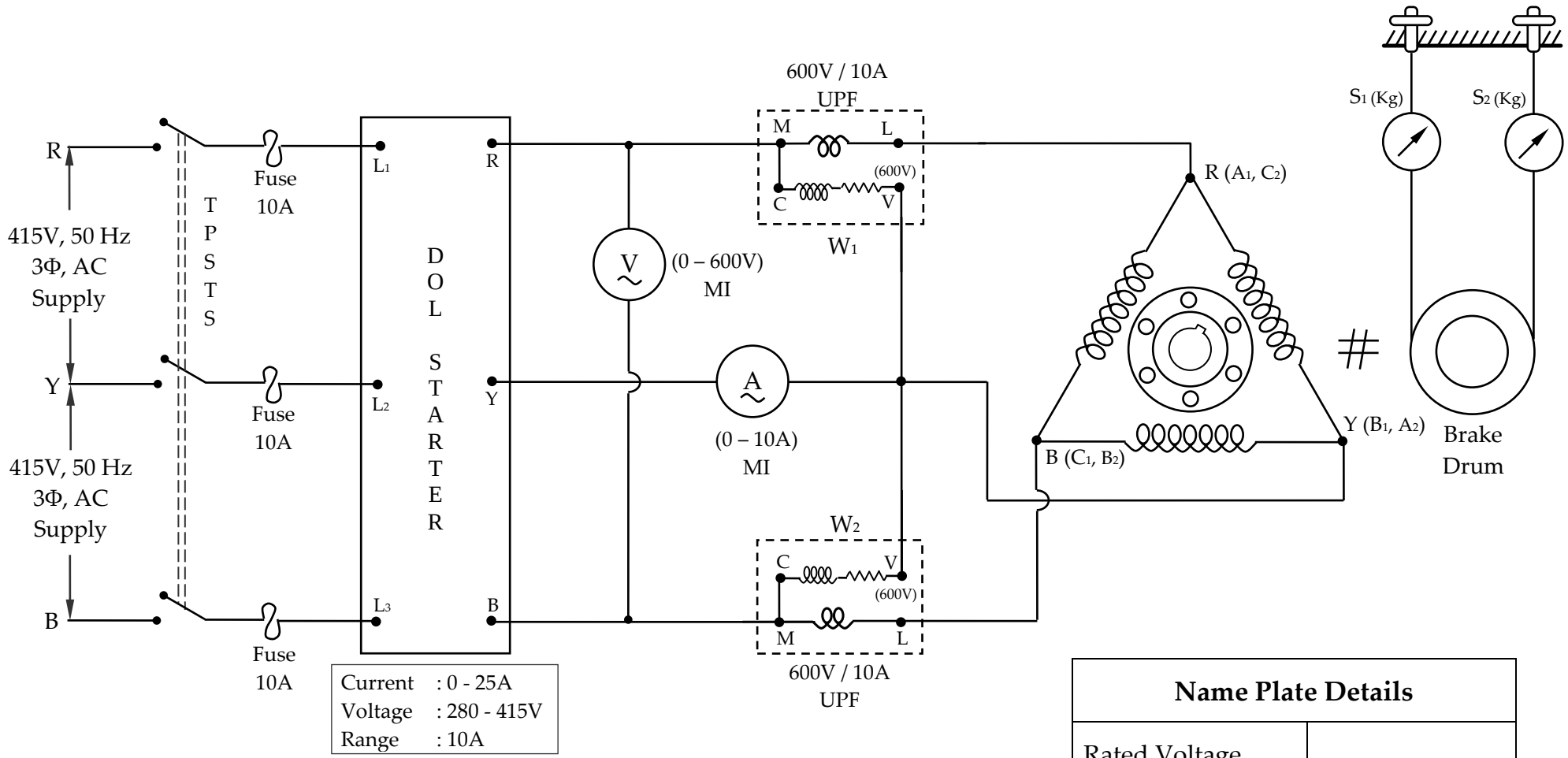
Name Plate Details		
	Motor	**Alternator**
Rated Voltage		
Rated Current		
Rated Speed		
Rated Power		
Type		

Circuit Diagram for Load Test on 1ϕ Induction Motor



Name Plate Details	
Rated Voltage	
Rated Current	
Rated Speed	
Rated Power	

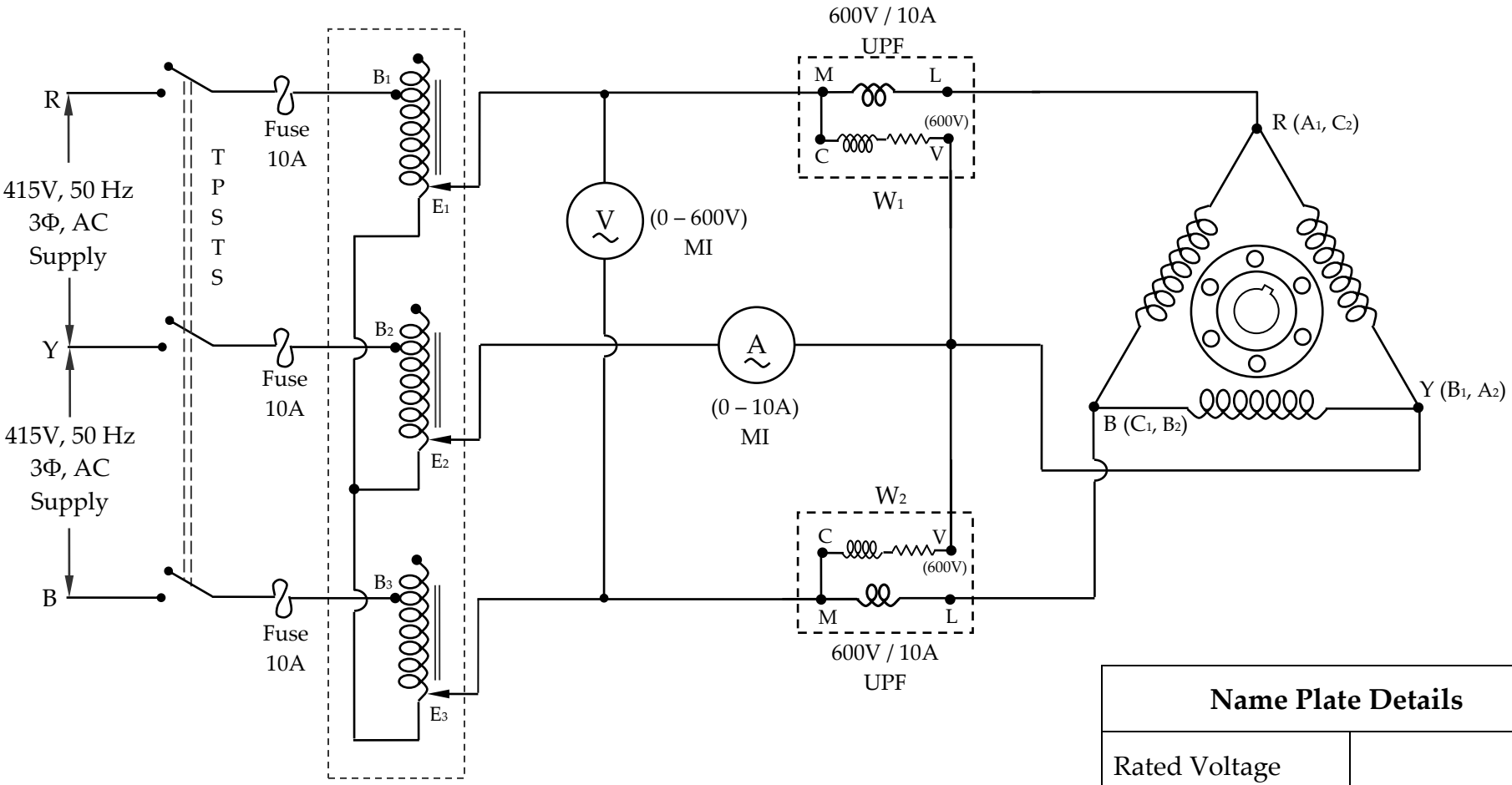
Circuit Diagram for Load Test on 3ϕ Squirrel Cage Induction Motor



Current	: 0 - 25A
Voltage	: 280 - 415V
Range	: 10A

Name Plate Details	
Rated Voltage	
Rated Current	
Rated Speed	
Rated Power	

Circuit Diagram for No Load Test on 3ϕ Squirrel Cage Induction Motor

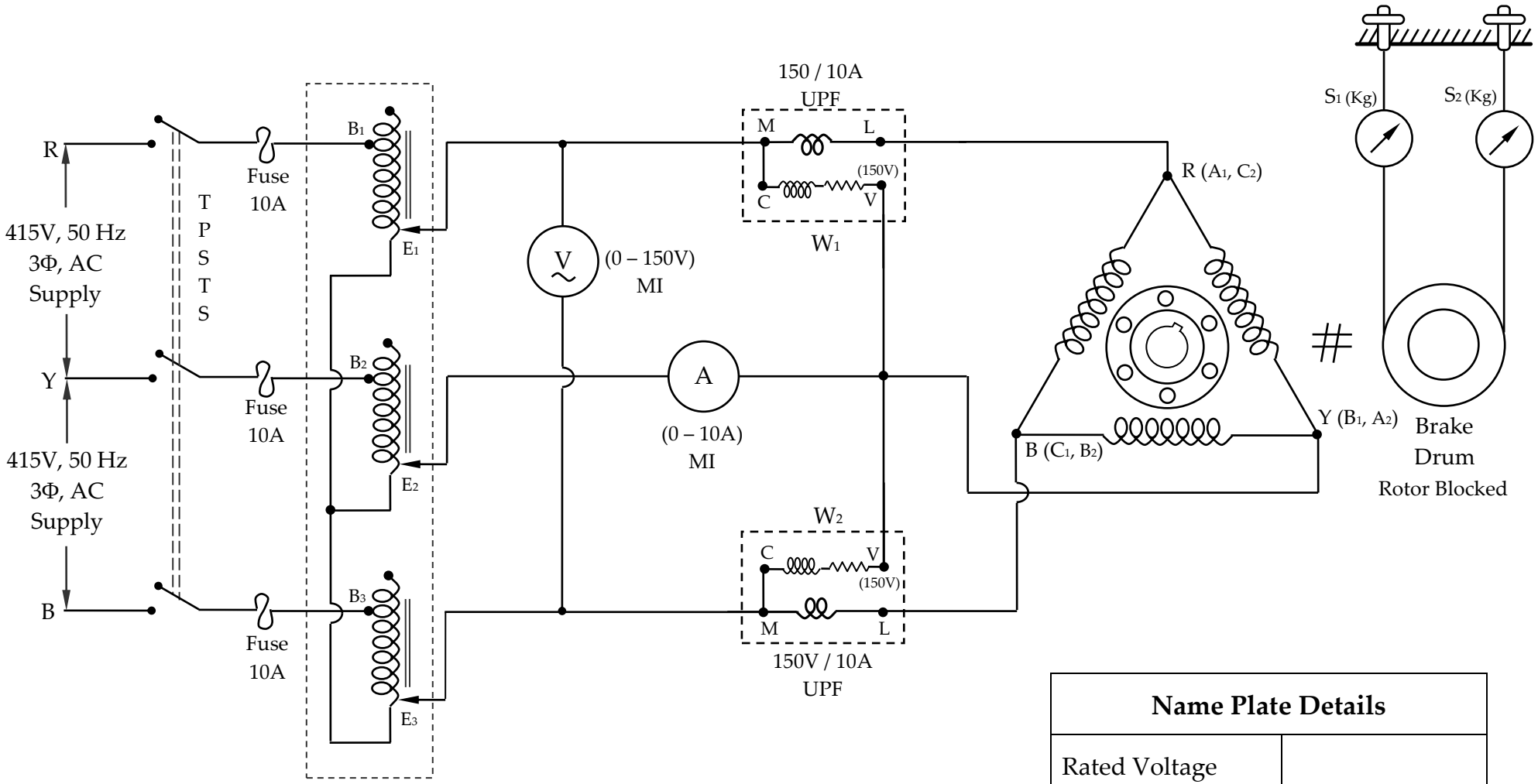


3Φ Variac

Input Voltage	: 415V
Output Voltage	: 0 - 470V
Output Current	: 15A / Line
Max Power	: 12.211 KVA

Name Plate Details	
Rated Voltage	
Rated Current	
Rated Speed	
Rated Power	

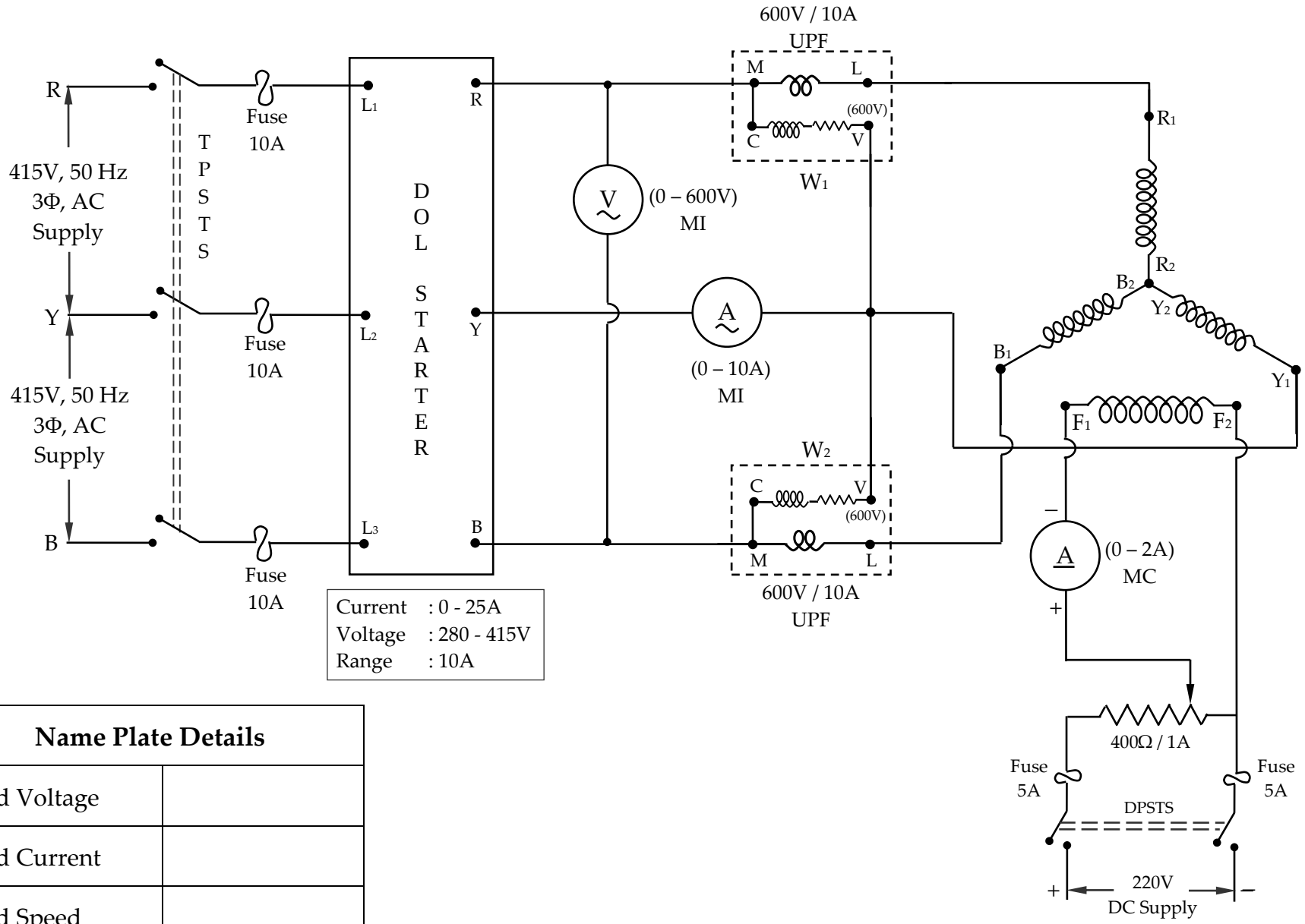
Circuit Diagram for Blocked Rotor Test on 3ϕ Squirrel Cage Induction Motor



Input Voltage	: 415V
Output Voltage	: 0 - 470V
Output Current	: 15A / Line
Max Power	: 12.211 KVA

Name Plate Details	
Rated Voltage	
Rated Current	
Rated Speed	
Rated Power	

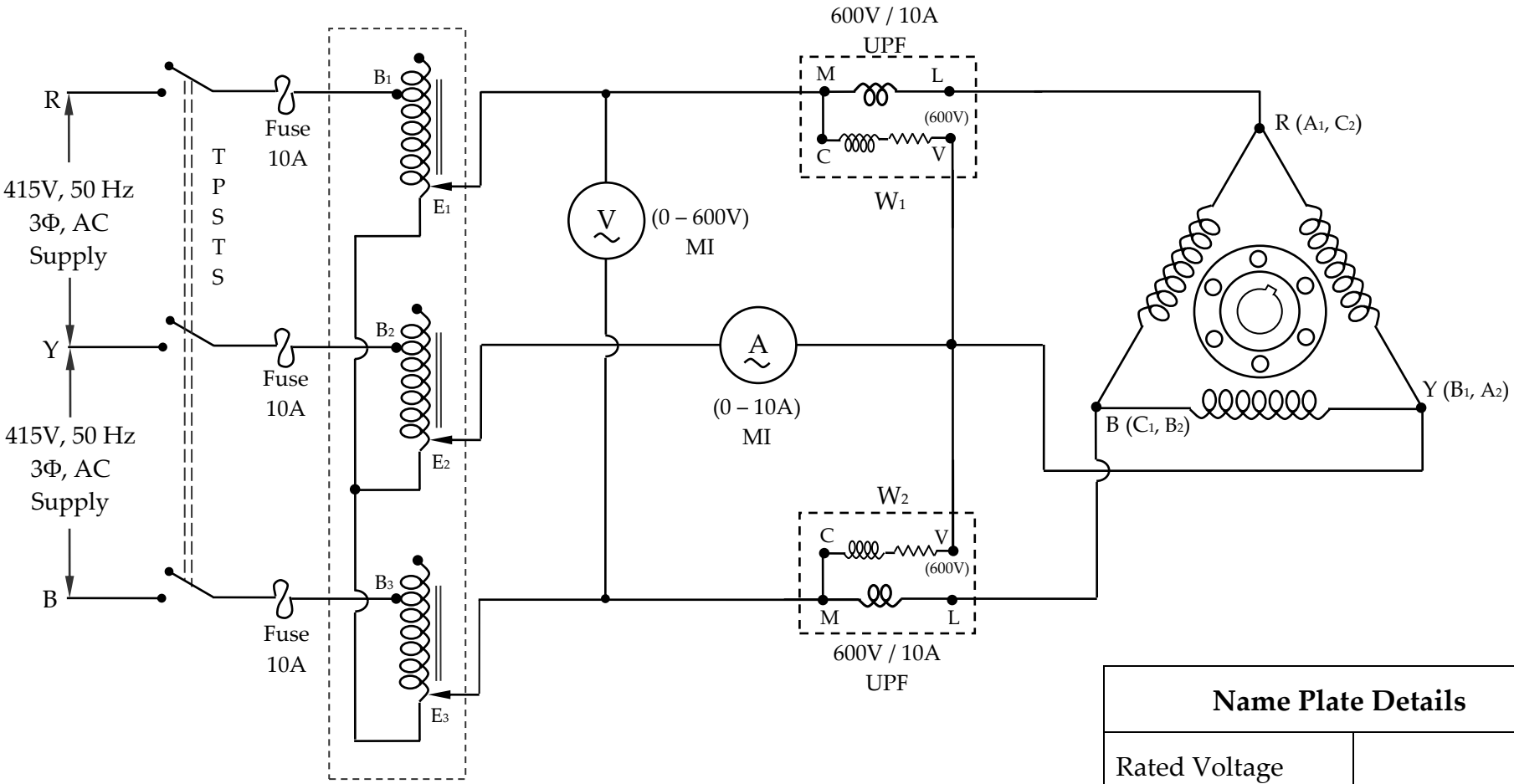
Circuit Diagram for "V" and Inverted "V" Curves of Synchronous Motor



Current	: 0 - 25A
Voltage	: 280 - 415V
Range	: 10A

Name Plate Details	
Rated Voltage	
Rated Current	
Rated Speed	
Rated Power	

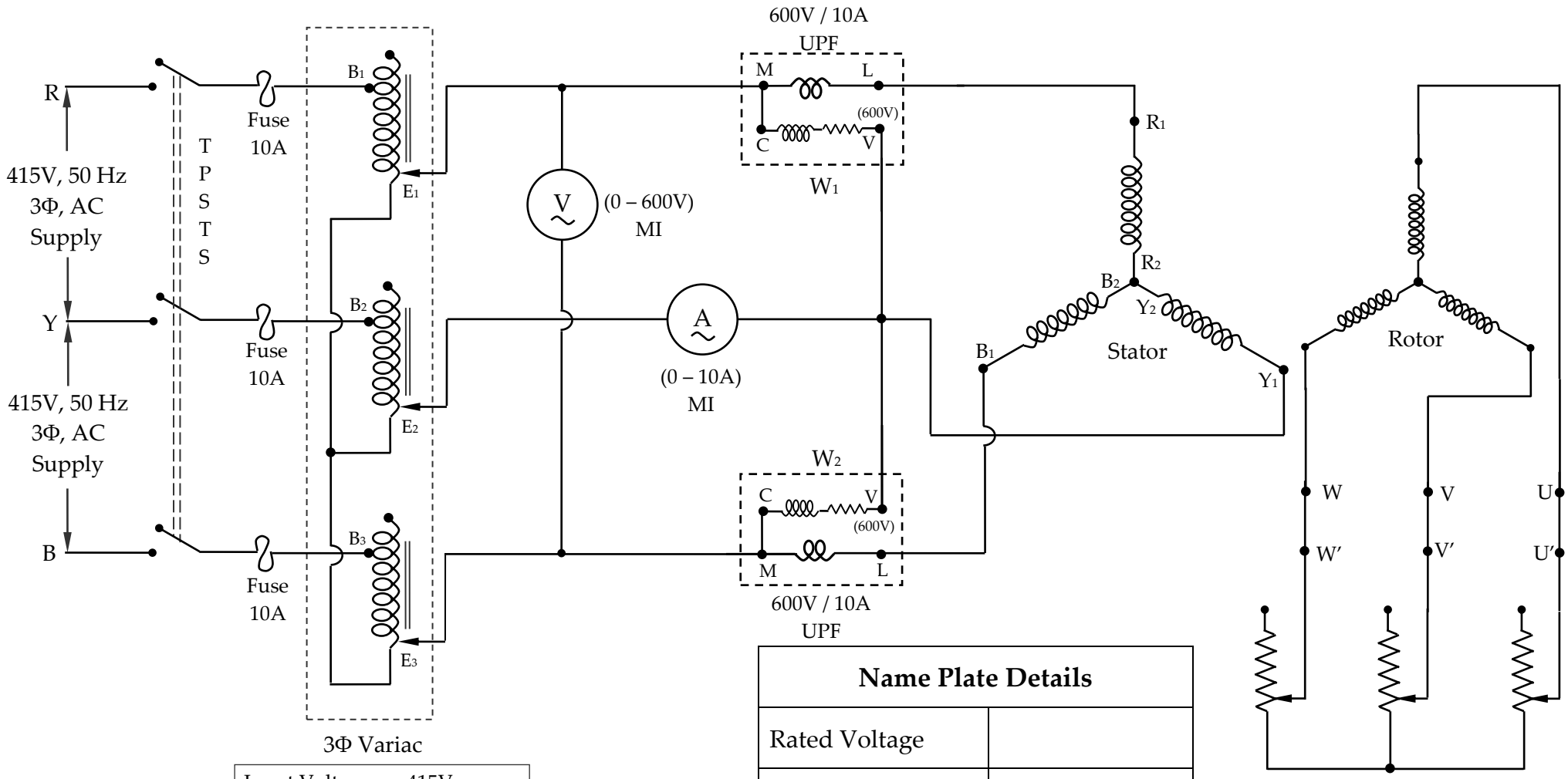
Separation of No-Load Losses of 3Φ Squirrel Cage Induction Motor



Input Voltage	: 415V
Output Voltage	: 0 - 470V
Output Current	: 15A / Line
Max Power	: 12.211 KVA

Name Plate Details	
Rated Voltage	
Rated Current	
Rated Speed	
Rated Power	

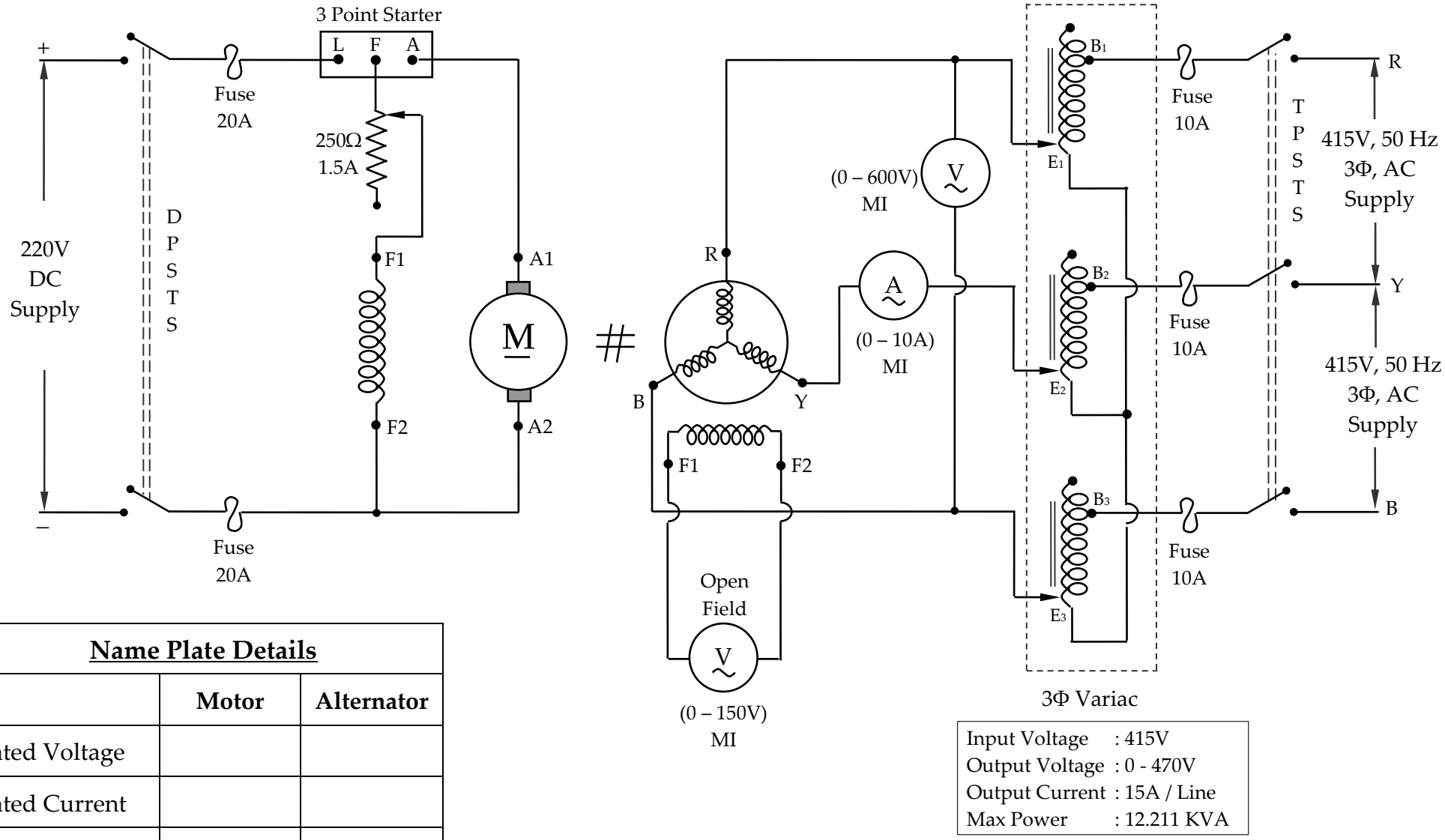
Circuit Diagram for Load Test on 3ϕ Slip Ring Induction Motor



Input Voltage	: 415V
Output Voltage	: 0 - 470V
Output Current	: 15A / Line
Max Power	: 12.211 KVA

Name Plate Details	
Rated Voltage	
Rated Current	
Rated Speed	
Rated Power	

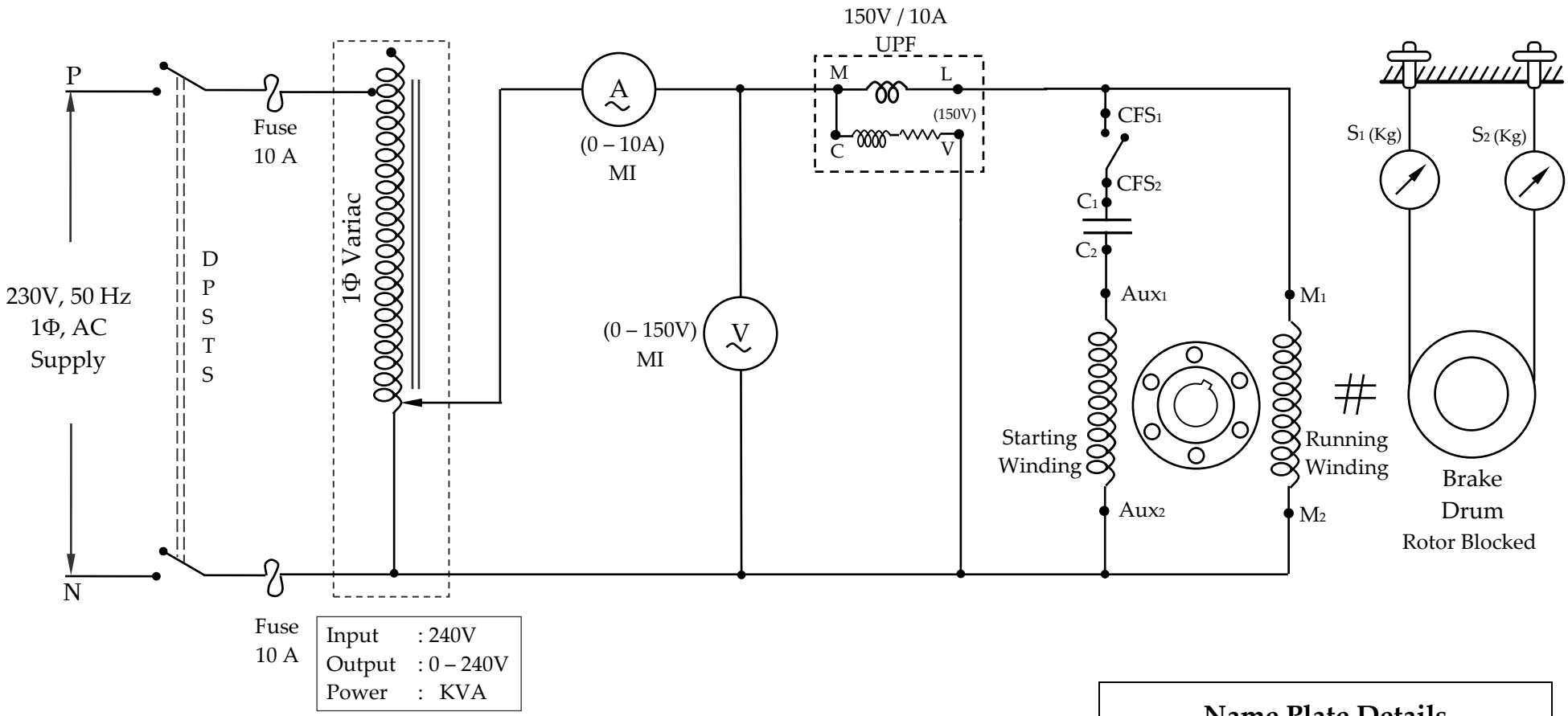
Circuit Diagram for Slip Test on 3Φ Alternator



Input Voltage	: 415V
Output Voltage	: 0 - 470V
Output Current	: 15A / Line
Max Power	: 12.211 KVA

Name Plate Details		
	Motor	**Alternator**
Rated Voltage		
Rated Current		
Rated Speed		
Rated Power		
Type		

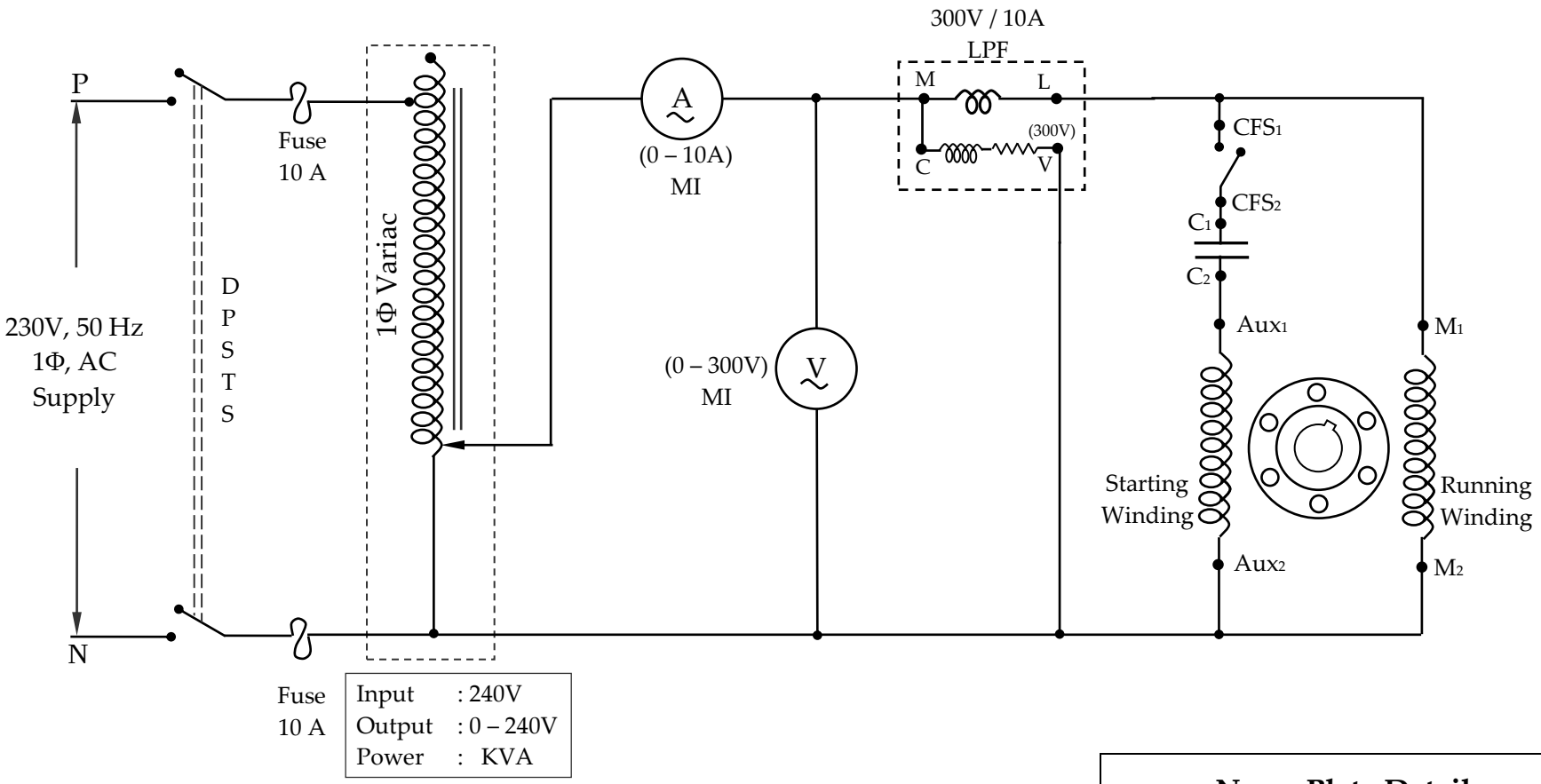
Circuit Diagram for Blocked Rotor Test on 1ϕ Induction Motor



Input	: 240V
Output	: 0 – 240V
Power	: KVA

Name Plate Details	
Rated Voltage	
Rated Current	
Rated Speed	
Rated Power	

Circuit Diagram for No Load Test on 1ϕ Induction Motor



Name Plate Details	
Rated Voltage	
Rated Current	
Rated Speed	
Rated Power	

MADHA ENGINEERING COLLEGE

(A Christian Minority Institution)

KUNDRATHUR, CHENNAI – 600 069

Linear and Digital Circuits Lab Manual

Name	:	
Subject	:	
Roll No.	:	
Semester	:	**Year:**

EX. NO.: 1
DATE: **STUDY OF LOGIC GATES**

AIM:

To study about logic gates and verify their truth tables.

APPARATUS REQUIRED:

SL.NO.	COMPONENT	SPECIFICATION	QTY
1.	AND GATE	IC 7408	1
2.	OR GATE	IC 7432	1
3.	NOT GATE	IC 7404	1
4.	NAND GATE 2 I/P	IC 7400	1
5.	NOR GATE	IC 7402	1
6.	X-OR GATE	IC 7486	1
7.	NAND GATE 3 I/P	IC 7410	1
8.	IC TRAINER KIT	-	1
9.	PATCH CARD	-	14

THEORY:

Circuit that takes the logical decision and the process are called logic gates. Each gate has one or more input and only one output.

OR, AND and NOT are basic gates. NAND, NOR and X-OR are known as universal gates. Basic gates form these gates.

AND GATE:

The AND gate performs a logical multiplication commonly known as AND function. The output is high when both the inputs are high. The output is low level when any one of the inputs is low.

OR GATE:

The OR gate performs a logical addition commonly known as OR function. The output is high when any one of the inputs is high. The output is low level when both the inputs are low.

NOT GATE:

The NOT gate is called an inverter. The output is high when the input is low. The output is low when the input is high.

AND GATE:

The NAND gate is a contraction of AND-NOT. The output is high when both inputs are low and any one of the input is low .The output is low level when both inputs are high.

NOR GATE:

The NOR gate is a contraction of OR-NOT. The output is high when both inputs are low. The output is low when one or both inputs are high.

X-OR GATE:

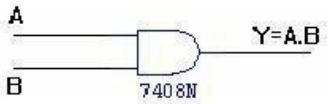
The output is high when any one of the inputs is high. The output is low when both the inputs are low and both the inputs are high.

PROCEDURE:

(i) Connections are given as per circuit diagram.
(ii) Logical inputs are given as per circuit diagram.
(iii) Observe the output and verify the truth table.

AND GATE

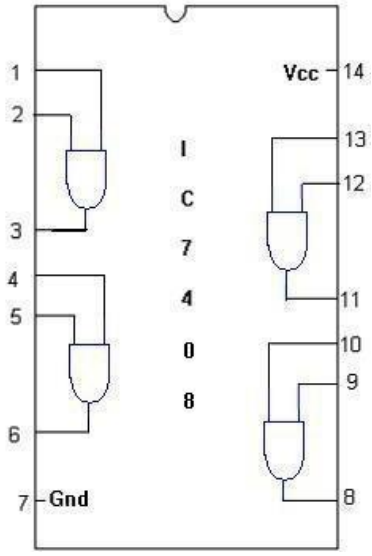
SYMBOL



TRUTH TABLE

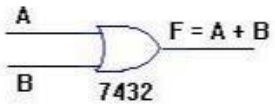
A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

PIN DIAGRAM



OR GATE

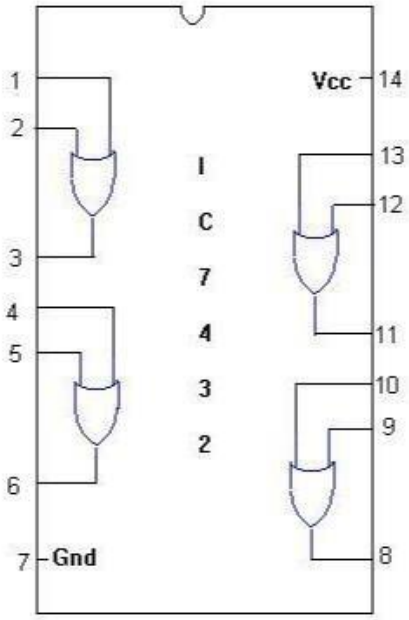
SYMBOL :



TRUTH TABLE

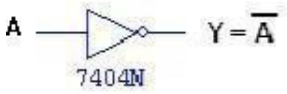
A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

PIN DIAGRAM :

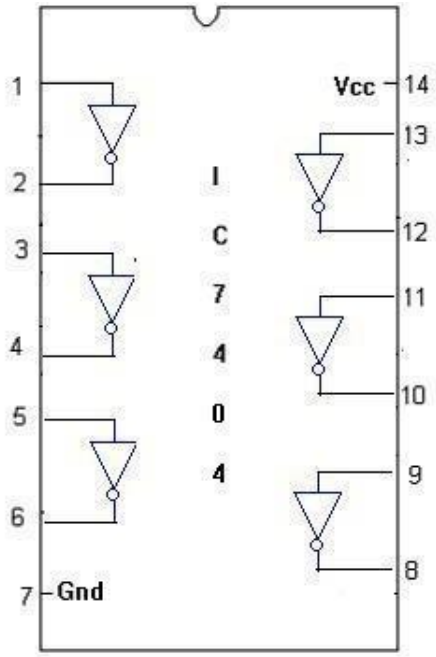


NOT GATE

SYMBOL



PIN DIAGRAM

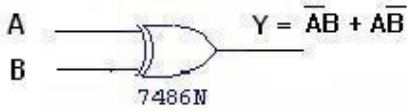


TRUTH TABLE :

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

EX-OR GATE

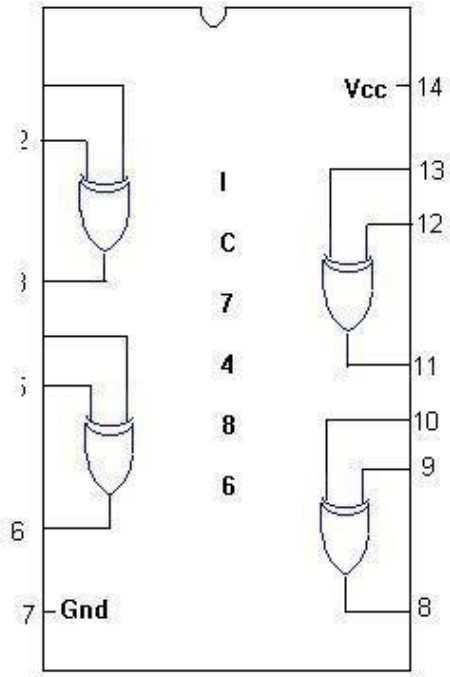
SYMBOL



TRUTH TABLE :

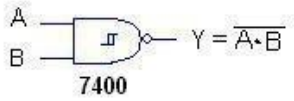
A	B	$\overline{A}B + A\overline{B}$
0	0	0
0	1	1
1	0	1
1	1	0

PIN DIAGRAM

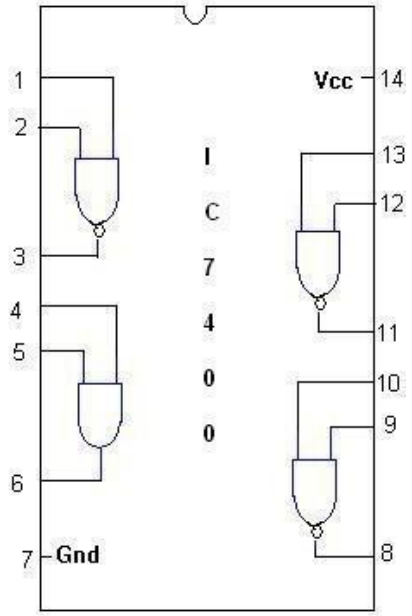


2-INPUT NAND GATE

SYMBOL



PIN DIAGRAM

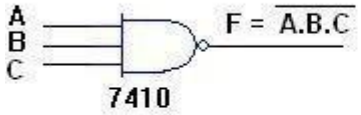


TRUTH TABLE

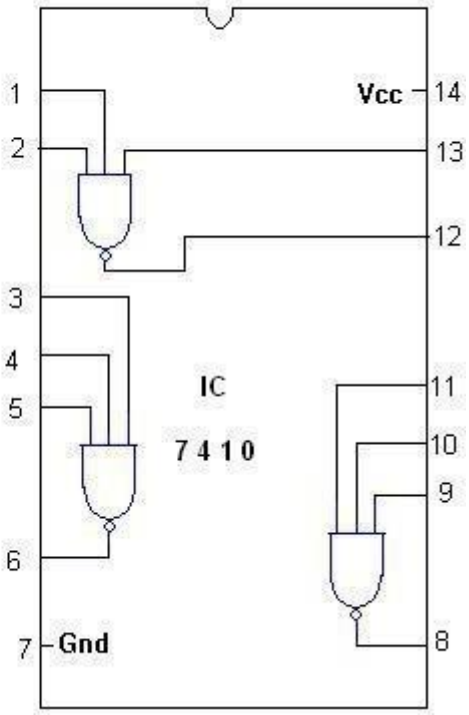
A	B	$\overline{A \cdot B}$
0	0	1
0	1	1
1	0	1
1	1	0

3-INPUT NAND GATE

SYMBOL :



PIN DIAGRAM :

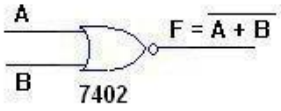


TRUTH TABLE

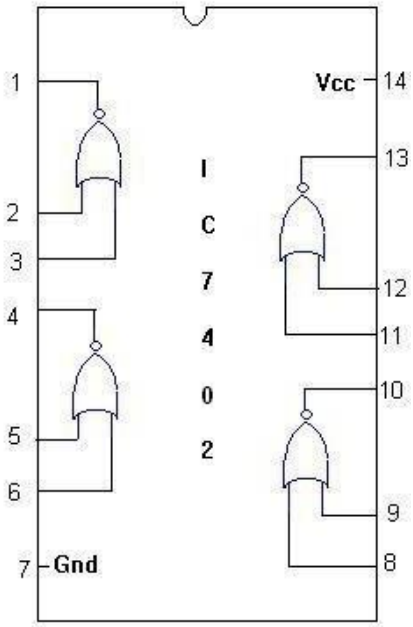
A	B	C	$\overline{A.B.C}$
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

NOR GATE

SYMBOL :



PIN DIAGRAM :



TRUTH TABLE

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

RESULT:

The logic gates are studied and its truth tables are verified.

EX. NO.: 2 VERIFICATION OF BOOLEAN THEOREMS USING DIGITAL LOGIC GATES

AIM:

To verify the Boolean Theorems using logic gates.

APPARATUS REQUIRED:

SL. NO.	COMPONENTS	SPECIFICATION	QTY.
1.	AND GATE	IC 7408	1
2.	OR GATE	IC 7432	1
3.	NOT GATE	IC 7404	1
4.	IC TRAINER KIT	-	1
5.	CONNECTING WIRES	-	As per required

THEORY:

BASIC BOOLEAN LAWS

1. Commutative Law

The binary operator OR, AND is said to be commutative if,

1. A+B = B+A
2. A.B=B.A

2. Associative Law

1. A+(B+C) = (A+B)+C
2. A.(B.C) = (A.B).C

3. Distributive Law

The binary operator OR, AND is said to be distributive if,

1. A+(B.C) = (A+B).(A+C)
2. A.(B+C) = (A.B)+(A.C)

4. Absorption Law

1. A+AB = A
2. A+AB = A+B

5. Involution (or) Double complement Law

1. A = A

6. Idempotent Law

1. A+A=A
2. A.A=A

7. Complementary Law

1. A+A' = 1
2. A.A' = 0

8. De Morgan's Theorem

1. The complement of the sum is equal to the sum of the product of the individual complements.

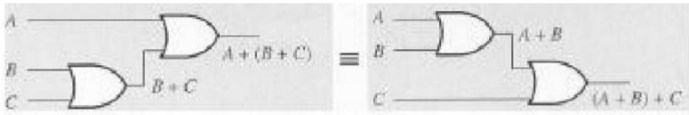
   A+B = A.B

2. The complement of the product is equal to the sum of the individual complements.

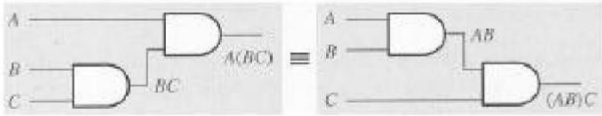
   A.B = A+B

Associative Laws of Boolean Algebra

A + (B + C) = (A + B) + C



A • (B • C) = (A • B) • C



Proof of the Associative Property for the OR operation: (A+B)+C = A+(B+C)

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

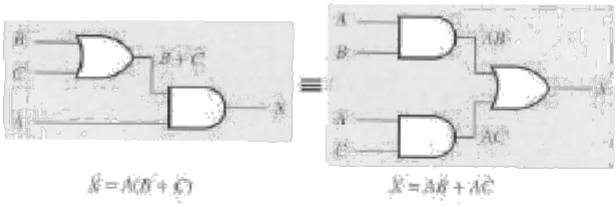
Proof of the Associative Property for the AND operation: (A·B)·C = A·(B·C)

A	B	C	(A·B)	(B·C)	A· (B·C)	(A·B)·C
0	0	0	0	0	0	0
0	0	1	0	0	0	0
0	1	0	0	0	0	0
0	1	1	0	1	0	0
1	0	0	0	0	0	0
1	0	1	0	0	0	0
1	1	0	1	0	0	0
1	1	1	1	1	1	1

Distributive Laws of Boolean Algebra

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

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



Proof of Distributive Rule

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

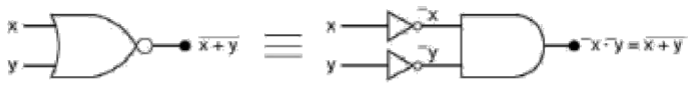
Proof of Distributive Rule

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

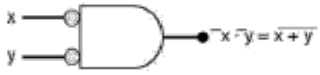
Demorgan's Theorem

a) Proof of equation (1):

Construct the two circuits corresponding to the functions A'. B'and (A+B)' respectively. Show that for all combinations of A and B, the two circuits give identical results. Connect these circuits and verify their operations.



(a)



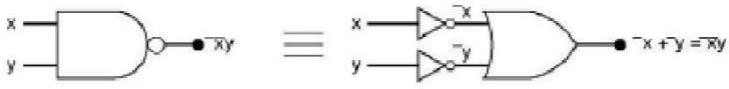
(b)

Proof (via Truth Table) of DeMorgan's Theorem $\overline{A \bullet B} = \overline{A} + \overline{B}$

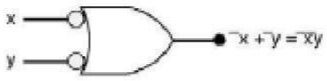
A	B	A·B	$\overline{A \bullet B}$	$\overline{A}$	$\overline{B}$	$\overline{A}+\overline{B}$
0	0	0	1	1	1	1
0	1	0	1	1	0	1
1	0	0	1	0	1	1
1	1	1	0	0	0	0

b) Proof of equation (2)

Construct two circuits corresponding to the functions A'+B'and (A.B)' A.B, respectively. Show that, for all combinations of A and B, the two circuits give identical results. Connect these circuits and verify their operations.



(a)



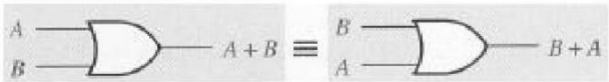
(b)

Proof (via Truth Table) of DeMorgan's Theorem $\overline{A + B} = \overline{A} \bullet \overline{B}$

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

Commutative Laws of Boolean Algebra

A + B = B + A



A • B = B • A

We will also use the following set of postulates:

P1: Boolean algebra is closed under the AND, OR, and NOT operations.

P2: The identity element with respect to • is one and + is zero. There is no identity element with respect to logical NOT.

P3: The • and + operators are commutative.

P4: • and + are distributive with respect to one another. That is,

A • (B + C) = (A • B) + (A • C) and A + (B • C) = (A + B) • (A + C).

P5: For every value A there exists a value A' such that A•A' = 0 and A+A' = 1. This value is the logical complement (or NOT) of A.

P6: • and + are both associative. That is, (A•B)•C = A•(B•C) and (A+B)+C = A+(B+C).

You can prove all other theorems in boolean algebra using these postulates.

PROCEDURE:

1. Obtain the required IC along with the Digital trainer kit.
2. Connect zero volts to GND pin and +5 volts to Vcc .
3. Apply the inputs to the respective input pins.
4. Verify the output with the truth table.

RESULT:

Thus the above stated Boolean laws are verified.

EX. NO.: 3 CODE CONVERTOR
DATE:

AIM:

To design and implement 4-bit

(i) Binary to gray code converter
(ii) Gray to binary code converter
(iii) BCD to excess-3 code converter
(iv) Excess-3 to BCD code converter

APPARATUS REQUIRED:

S. NO.	COMPONENT	SPECIFICATION	QTY.
1.	X-OR GATE	IC 7486	1
2.	AND GATE	IC 7408	1
3.	OR GATE	IC 7432	1
4.	NOT GATE	IC 7404	1
5.	IC TRAINER KIT	-	1
6.	PATCH CORDS	-	35

THEORY:

The availability of large variety of codes for the same discrete elements of information results in the use of different codes by different systems. A conversion circuit must be inserted between the two systems if each uses different codes for same information. Thus, code converter is a circuit that makes the two systems compatible even though each uses different binary code.

The bit combination assigned to binary code to gray code. Since each code uses four bits to represent a decimal digit. There are four inputs and four outputs. Gray code is a non-weighted code.

The input variable are designated as B3, B2, B1, B0 and the output variables are designated as C3, C2, C1, Co. from the truth table, combinational circuit is designed. The Boolean functions are obtained from K-Map for each output variable.

A code converter is a circuit that makes the two systems compatible even though each uses a different binary code. To convert from binary code to Excess-3 code, the input lines must supply the bit combination of elements as specified by code and the output lines generate the corresponding bit combination of code. Each one of the four maps represents one of the four outputs of the circuit as a function of the four input variables.

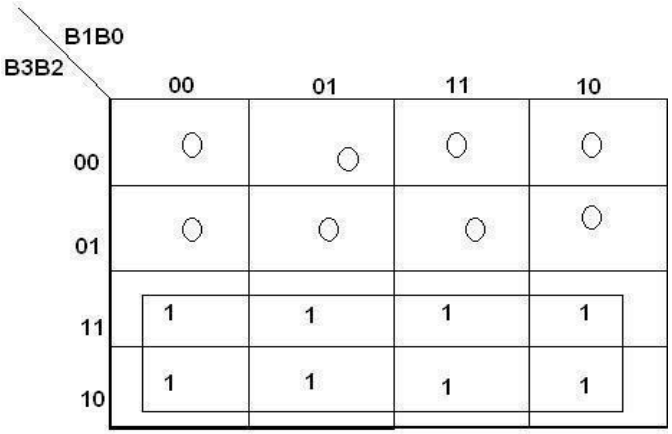
A two-level logic diagram may be obtained directly from the Boolean expressions derived by the maps. These are various other possibilities for a logic diagram that implements this circuit. Now the OR gate whose output is C+D has been used to implement partially each of three outputs.

BINARY TO GRAY CODE CONVERTOR

TRUTH TABLE:

Binary Input				Gray Code Output			
B3	**B2**	**B1**	**B0**	**G3**	**G2**	**G1**	**G0**
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

K-Map for G3



G3=B3

K-Map for G2

B3B2 \ B1B0	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	0	0
10	1	1	1	1

$$G2 = B3 \oplus B2$$

K-Map for G1

B3B2 \ B1B0	00	01	11	10
00	0	0	1	1
01	1	1	0	0
11	1	1	0	0
10	0	0	1	1

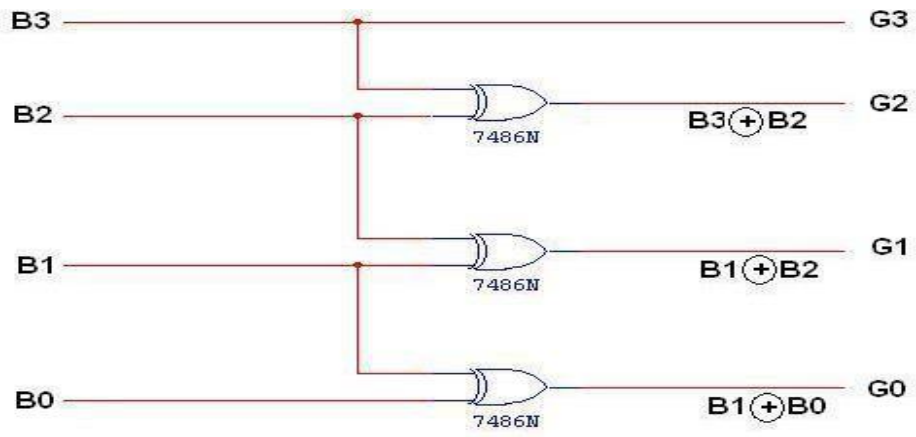
$$G1 = B1 \oplus B2$$

K-Map for G0

B3B2 \ B1B0	00	01	11	10
00	0	1	0	1
01	0	1	0	1
11	0	1	0	1
10	0	1	0	1

$$G0 = B1 \oplus B0$$

LOGIC DIAGRAM:



GRAY CODE TO BINARY CONVERTOR

TRUTH TABLE:

GRAY CODE				BINARY CODE			
G3	**G2**	**G1**	**G0**	**B3**	**B2**	**B1**	**B0**
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	1	0	0	1	0
0	0	1	0	0	0	1	1
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
0	1	0	1	0	1	1	0
0	1	0	0	0	1	1	1
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	1
1	1	1	1	1	0	1	0
1	1	1	0	1	0	1	1
1	0	1	0	1	1	0	0
1	0	1	1	1	1	0	1
1	0	0	1	1	1	1	0
1	0	0	0	1	1	1	1

K-Map for B3:

G3G2 \ G1G0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	1	1	1	1

B3=G3

K-Map for B2:

G3G2 \ G1G0	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	0	0
10	1	1	1	1

B2 = G3⊕G2

K-Map for B1:

G3G2 \ G1G0	00	01	11	10
00	0	0	1	1
01	1	1	0	0
11	0	0	1	1
10	1	1	0	0

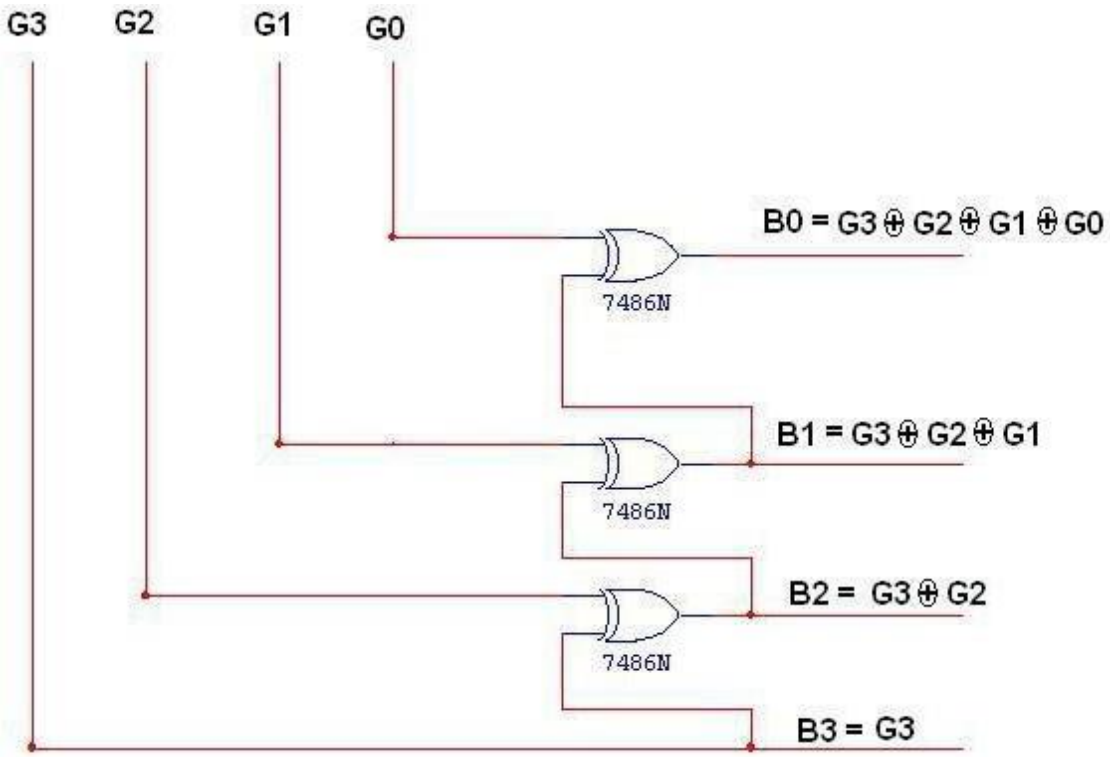
B1 = G3⊕G2⊕G1

K-Map for B0:

G3G2 \ G1G0	00	01	11	10
00	0	1	0	1
01	1	0	1	0
11	0	1	0	1
10	1	0	1	0

$$B0 = G3 \oplus G2 \oplus G1 \oplus G0$$

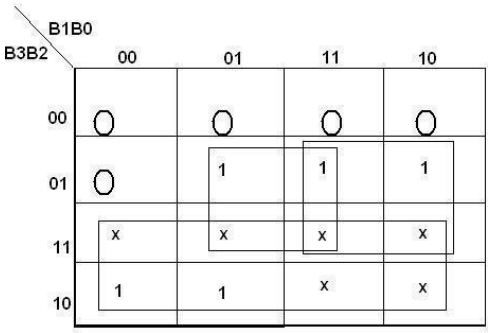
LOGIC DIAGRAM:



TRUTH TABLE: **BCD TO EXCESS-3 CONVERTOR**

BCD input				**Excess – 3 output**			
B3	**B2**	**B1**	**B0**	**G3**	**G2**	**G1**	**G0**
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	x	x	x	x
1	0	1	1	x	x	x	x
1	1	0	0	x	x	x	x
1	1	0	1	x	x	x	x
1	1	1	0	x	x	x	x
1	1	1	1	x	x	x	X

K-Map for E3:



E3=B3+B2 (B0+B1)

K-Map for E2:

B3B2 \ B1B0	00	01	11	10
00	0	1	1	1
01	1	0	0	0
11	x	x	x	x
10	0	1	x	x

$$E2 = B2 \oplus (B1 + B0)$$

K-Map for E1:

B3B2 \ B1B0	00	01	11	10
00	1	0	1	0
01	1	0	1	0
11	x	x	x	x
10	1	0	x	x

$$E1 = B1 \overline{\oplus} B0$$

K-Map for E0:

B3B2 \ B1B0	00	01	11	10
00	1	0	0	1
01	1	0	0	1
11	x	x	x	x
10	1	0	x	x

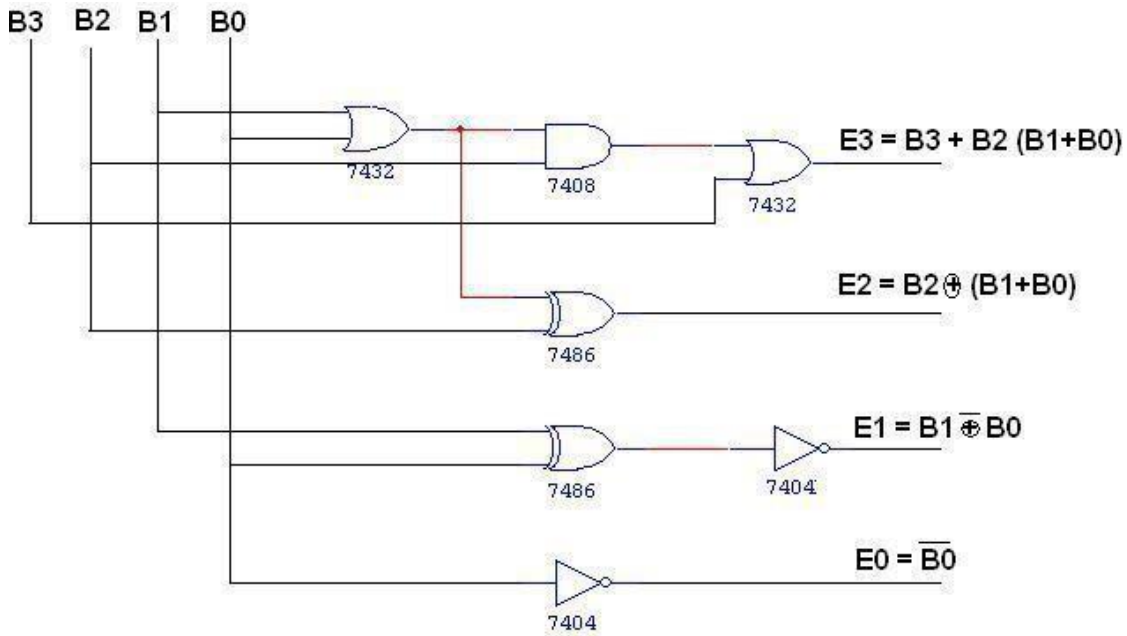
$$E0 = \overline{B0}$$

EXCESS-3 TO BCD CONVERTOR

TRUTH TABLE:

Excess – 3 Input				BCD Output			
B3	**B2**	**B1**	**B0**	**G3**	**G2**	**G1**	**G0**
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	1
0	1	0	1	0	0	1	0
0	1	1	0	0	0	1	1
0	1	1	1	0	1	0	0
1	0	0	0	0	1	0	1
1	0	0	1	0	1	1	0
1	0	1	0	0	1	1	1
1	0	1	1	1	0	0	0
1	1	0	0	1	0	0	1

LOGIC DIAGRAM:



EXCESS-3 TO BCD CONVERTOR

K-Map for A:

X1 X2 \ X3 X4	00	01	11	10
00	X	X	0	X
01	0	0	0	0
11	1	X	X	X
10	0	0	1	0

A=X1X2+X3X4X1

K-Map for B:

X1 X2 \ X3 X4	00	01	11	10
00	X	X	0	X
01	0	0	1	0
11	0	X	X	X
10	1	1	0	1

$$B = X2 \oplus (\overline{X3} + \overline{X4})$$

K-Map for C:

X1 X2 \ X3 X4	00	01	11	10
00	X	X	0	X
01	0	1	X	1
11	0	X	X	X
10	X	1	0	1

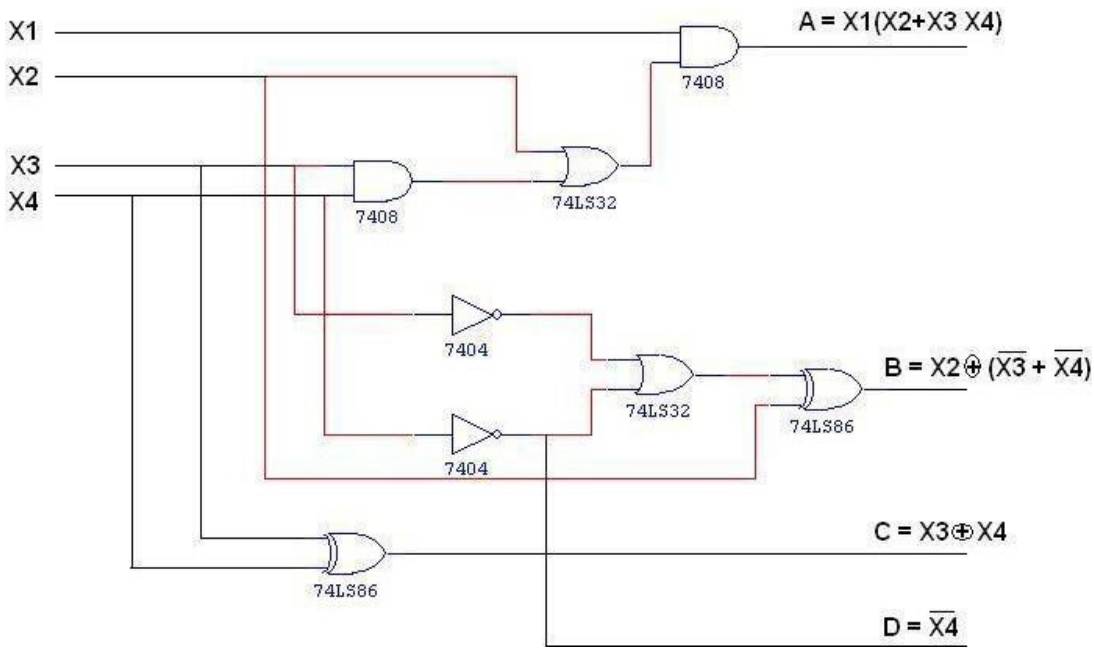
$$C = X3 \oplus X4$$

K-Map for D:

X1 X2 \ X3 X4	00	01	11	10
00	X	X	0	X
01	1	0	0	1
11	1	X	X	X
10	1	0	0	1

$$D = \overline{X4}$$

EXCESS-3 TO BCD CONVERTOR



PROCEDURE:

(i) Connections were given as per circuit diagram.
(ii) Logical inputs were given as per truth table
(iii) Observe the logical output and verify with the truth tables.

RESULT:

Thus the following 4-bit converters are designed and constructed.

(i) Binary to gray code converter
(ii) Gray to binary code converter
(iii) BCD to excess-3 code converter
(iv) Excess-3 to BCD code converter

EX. NO.: 4 **ADDER AND SUBTRACTOR**
DATE:

AIM:

To design and construct half adder, full adder, half subtractor and full subtractor circuits and verify the truth table using logic gates.

APPARATUS REQUIRED:

SL.NO.	COMPONENTS	SPECIFICATION	QTY.
1.	AND GATE	IC 7408	1
2.	X-OR GATE	IC 7486	1
3.	NOT GATE	IC 7404	1
4.	OR GATE	IC 7432	1
5.	IC TRAINER KIT	-	1
6.	PATCH CORDS	-	23

THEORY:

HALF ADDER:

A half adder has two inputs for the two bits to be added and two outputs one from the sum ‘ S’ and other from the carry ‘ c’ into the higher adder position. Above circuit is called as a carry signal from the addition of the less significant bits sum from the X-OR Gate the carry out from the AND gate.

FULL ADDER:

A full adder is a combinational circuit that forms the arithmetic sum of input; it consists of three inputs and two outputs. A full adder is useful to add three bits at a time but a half adder cannot do so. In full adder sum output will be taken from X-OR Gate, carry output will be taken from OR Gate.

HALF SUBTRACTOR:

The half subtractor is constructed using X-OR and AND Gate. The half subtractor has two input and two outputs. The outputs are difference and borrow. The difference can be applied using X-OR Gate, borrow output can be implemented using an AND Gate and an inverter.

FULL SUBTRACTOR:

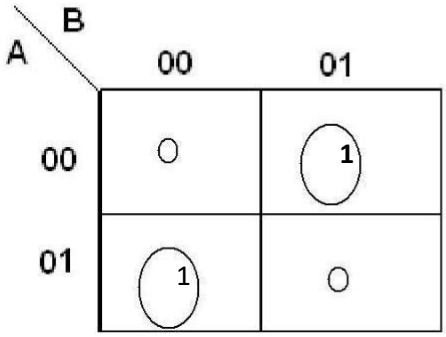
The full subtractor is a combination of X-OR, AND, OR, NOT Gates. In a full subtractor the logic circuit should have three inputs and two outputs. The two half subtractor put together gives a full subtract or .The first half subtractor will be C and A B. The output will be difference output of full subtractor. The expression AB assembles the borrow output of the half subtract or and the second term is the inverted difference output of first X-OR.

HALF ADDER

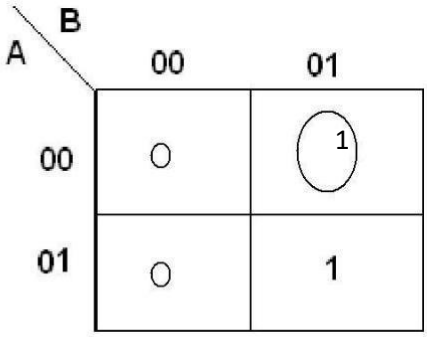
TRUTH TABLE:

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

K-Map for SUM:



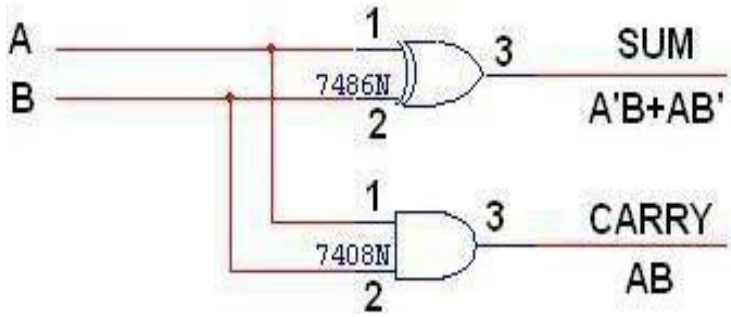
K-Map for CARRY:



SUM = A'B + AB'

CARRY = AB

LOGIC DIAGRAM:

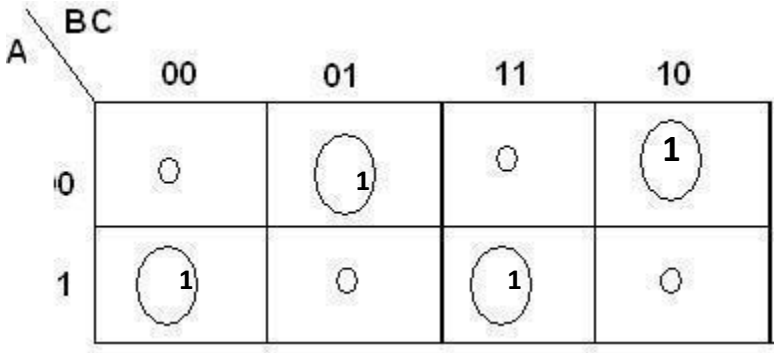


FULL ADDER

TRUTH TABLE:

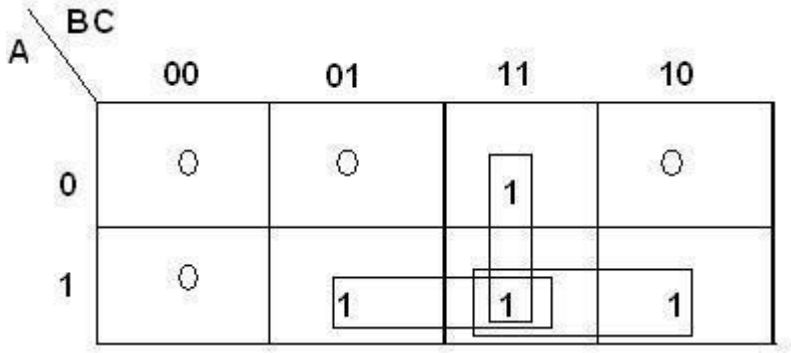
A	B	C	CARRY	SUM
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

K-Map for SUM



SUM = A'B'C + A'BC' + ABC' + ABC

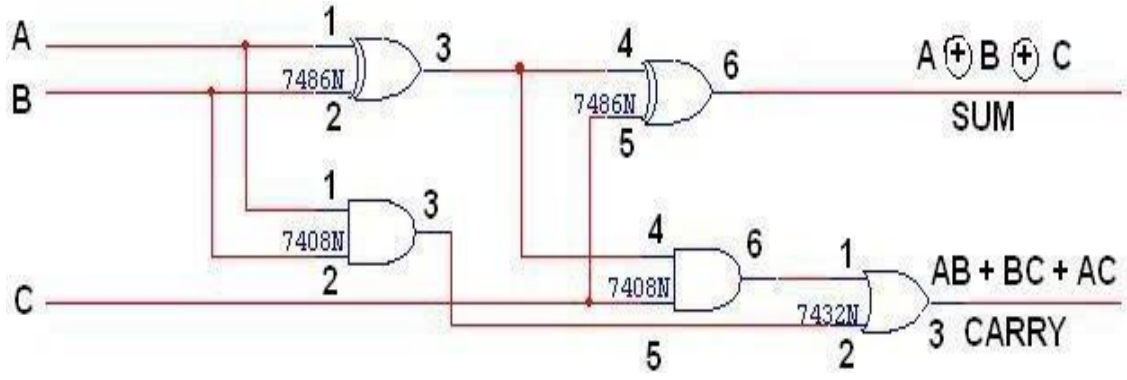
K-Map for CARRY



CARRY = AB + BC + AC

LOGIC DIAGRAM:

FULL ADDER USING TWO HALF ADDER

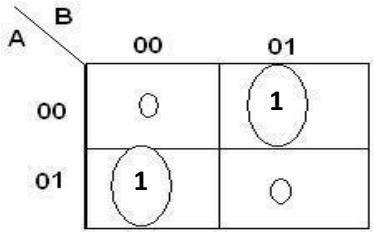


HALF SUBTRACTOR

TRUTH TABLE:

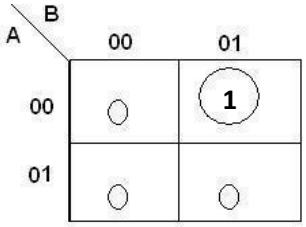
A	B	BORROW	DIFFERENCE
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

K-Map for DIFFERENCE



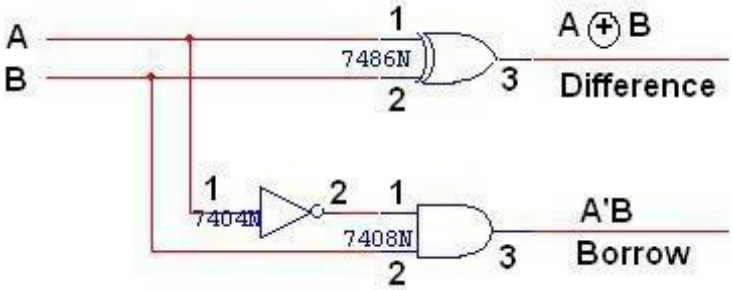
DIFFERENCE = A'B + AB'

K-Map for BORROW



BORROW = A'B

LOGIC DIAGRAM

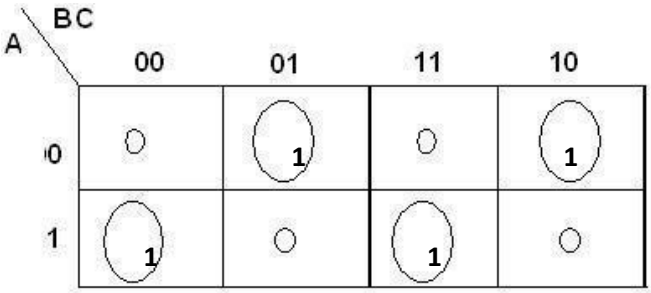


FULL SUBTRACTOR

TRUTH TABLE:

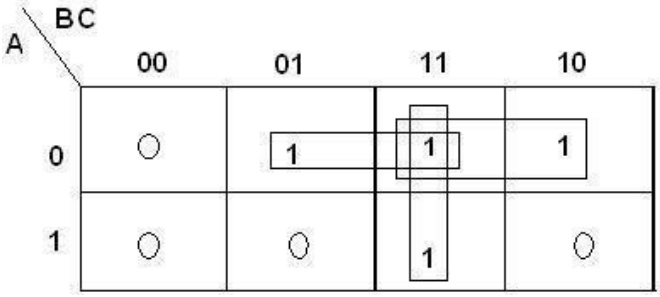
A	B	C	BORROW	DIFFERENCE
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

K-Map for Difference



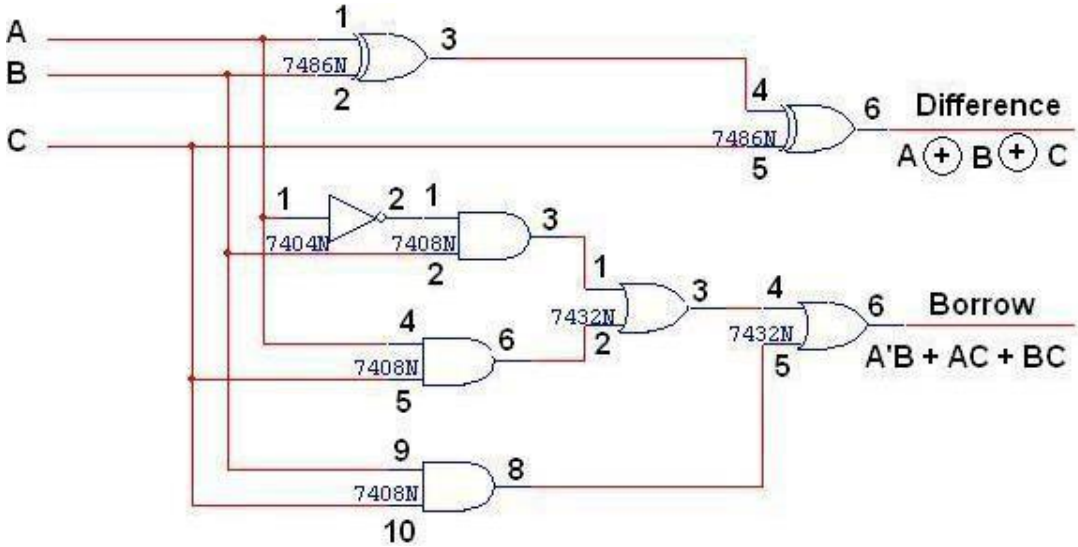
Difference = A'B'C + A'BC' + AB'C' +

ABC K-Map for Borrow

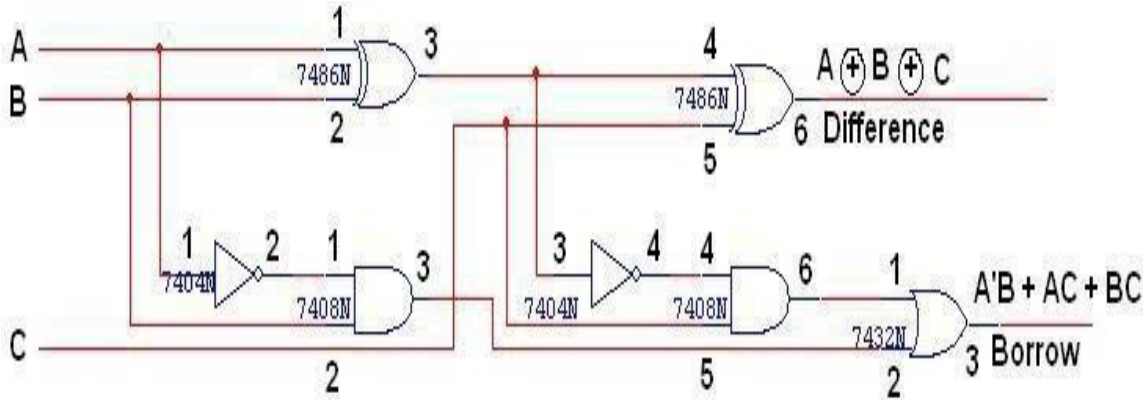


Borrow = A'B + BC + A'C

LOGIC DIAGRAM:



FULL SUBTRACTOR USING TWO HALF SUBTRACTOR



PROCEEDURE:

(i) Connections are given as per circuit diagram.

(ii) Logical inputs are given as per circuit diagram.

(iii) Observe the output and verify the truth table.

RESULT:

Thus, the half adder, full adder, half subtract or and full subtractor circuits are designed, constructed and verified the truth table using logic gates.

EX. NO.: 5 **4-BIT ADDER AND SUBTRACTOR**
DATE:

AIM:

To design and implement 4-bit adder and subtractor using basic gates and MSI device IC 7483.

APPARATUS REQUIRED:

SL.NO.	COMPONENT	SPECIFICATION	QTY.
1.	IC	IC 7483	1
2.	EX-OR GATE	IC 7486	1
3.	NOT GATE	IC 7404	1
3.	IC TRAINER KIT	-	1
4.	PATCH CORDS	-	40

THEORY:

4 BIT BINARY ADDER:

A binary adder is a digital circuit that produces the arithmetic sum of two binary numbers. It can be constructed with full adders connected in cascade, with the output carry from each full adder connected to the input carry of next full adder in chain. The augends bits of 'A' and the addend bits of 'B' are designated by subscript numbers from right to left, with subscript 0 denoting the least significant bits. The carries are connected in chain through the full adder. The input carry to the adder is C_0 and it ripples through the full adder to the output carry C_4.

4 BIT BINARY SUBTRACTOR:

The circuit for subtracting A-B consists of an adder with inverters, placed between each data input 'B' and the corresponding input of full adder. The input carry C_0 must be equal to 1 when performing subtraction.

4 BIT BINARY ADDER/SUBTRACTOR:

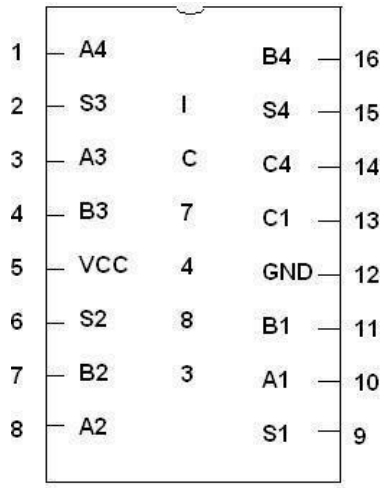
The addition and subtraction operation can be combined into one circuit with one common binary adder. The mode input M controls the operation. When M=0, the circuit is adder circuit. When M=1, it becomes subtractor.

4 BIT BCD ADDER:

Consider the arithmetic addition of two decimal digits in BCD, together with an input carry from a previous stage. Since each input digit does not exceed 9, the output sum cannot be greater than 19, the 1 in the sum being an input carry. The output of two decimal digits must be represented in BCD and should appear in the form listed in the columns.

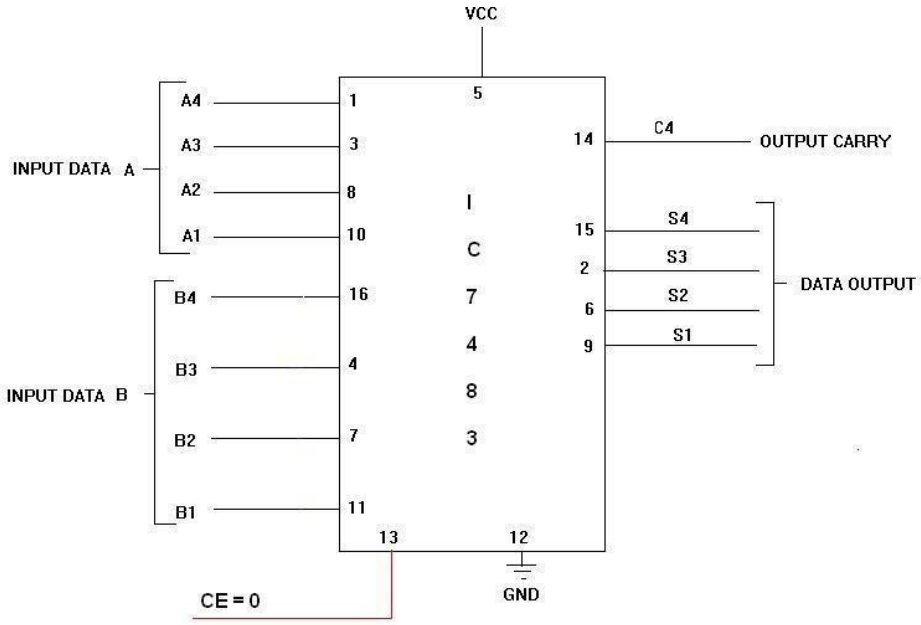
ABCD adder that adds 2 BCD digits and produce a sum digit in BCD. The 2 decimal digits, together with the input carry, are first added in the top 4 bit adder to produce the binary sum.

PIN DIAGRAM FOR IC 7483:



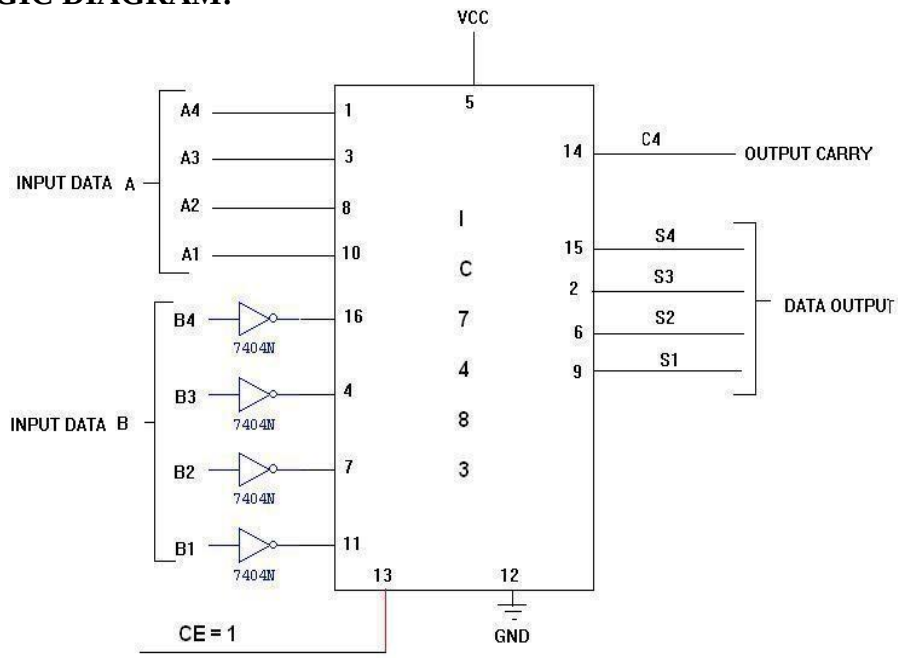
<u>4-BIT BINARY ADDER</u>

LOGIC DIAGRAM:



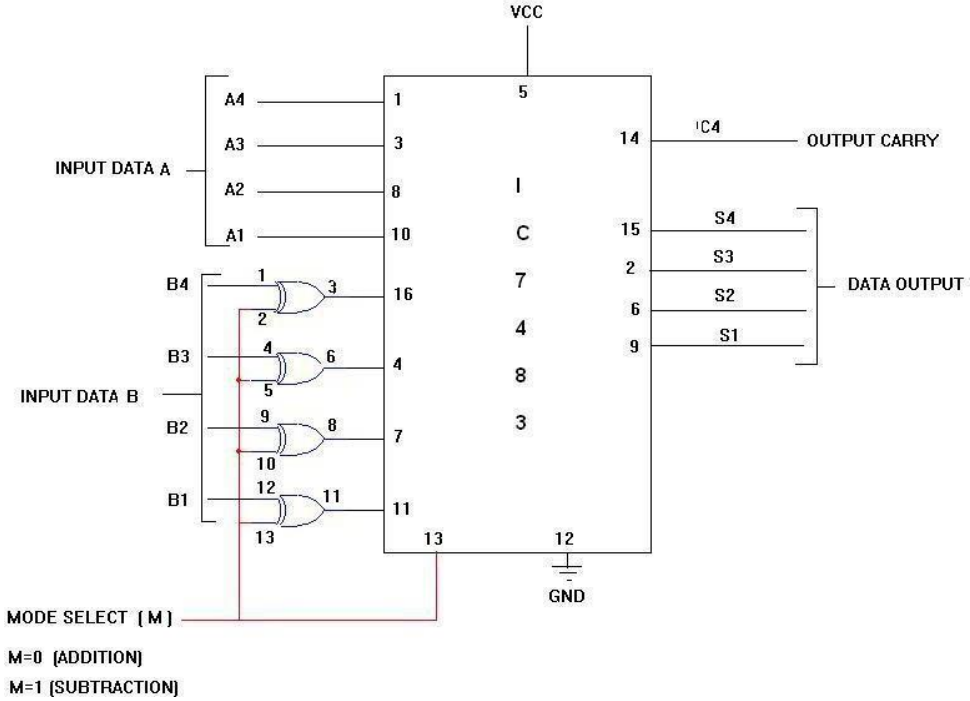
4-BIT BINARY SUBTRACTOR

LOGIC DIAGRAM:



4-BIT BINARY ADDER/SUBTRACTOR

LOGIC DIAGRAM:



TRUTH TABLE:

Input Data A				Input Data B					Addition					Subtraction			
A4	A3	A2	A1	B4	B3	B2	B1	C	S4	S3	S2	S1	B	D4	D3	D2	D1
1	0	0	0	0	0	1	0	0	1	0	1	0	1	0	1	1	0
1	0	0	0	1	0	0	0	1	0	0	0	0	1	0	0	0	0
0	0	1	0	1	0	0	0	0	1	0	1	0	0	1	0	1	0
0	0	0	1	0	1	1	1	0	1	0	0	0	0	1	0	1	0
1	0	1	0	1	0	1	1	1	0	0	1	0	0	1	1	1	1
1	1	1	0	1	1	1	1	1	1	0	1	0	0	1	1	1	1
1	0	1	0	1	1	0	1	1	0	1	1	1	0	1	1	0	1

PROCEDURE:

(i) Connections were given as per circuit diagram.

(ii) Logical inputs were given as per truth table

(iii) Observe the logical output and verify with the truth tables.

RESULT:

Thus the 4-bit adder and subtractor using basic gates and MSI device IC 7483 is designed and implemented.

EX. NO.: 6 **PARITY GENERATOR AND CHECKER**
DATE:

AIM:

To design and verify the truth table of a three bit Odd Parity generator and checker.

APPARATUS REQUIRED:

SL. NO.	NAME OF THE APPARATUS	RANGE	QUANTITY
1.	Digital IC trainer kit		1
2.	EX-OR gate	IC 7486	
3.	NOT gate	IC 7404	
4.	Connecting wires		As required

THEORY:

A parity bit is used for the purpose of detecting errors during transmission of binary information. A parity bit is an extra bit included with a binary message to make the number of 1's either odd or even. The message including the parity bit is transmitted and then checked at the receiving end for errors. An error is detected if the checked parity does not correspond with the one transmitted. The circuit that generates the parity bit in the transmitter is called a parity generator and the circuit that checks the parity in the receiver is called a parity checker.

In even parity the added parity bit will make the total number of 1's an even amount and in odd parity the added parity bit will make the total number of 1's an odd amount.

In a three bit odd parity generator the three bits in the message together with the parity bit are transmitted to their destination, where they are applied to the parity checker circuit. The parity checker circuit checks for possible errors in the transmission.

Since the information was transmitted with odd parity the four bits received must have an odd number of 1's. An error occurs during the transmission if the four bits received have an even number of 1's, indicating that one bit has changed during transmission. The output of the parity checker is denoted by PEC (parity error check) and it will be equal to 1 if an error occurs, i.e.,

if the four bits received has an even number of 1's.

ODD PARITY GENERATOR

TRUTH TABLE:

SL.NO.	INPUT			OUTPUT
	(Three bit message)			(Odd Parity bit)
	A	B	C	P
1.	0	0	0	1
2.	0	0	1	0
3.	0	1	0	0
4.	0	1	1	1
5.	1	0	0	0
6.	1	0	1	1
7.	1	1	0	1
8.	1	1	1	0

From the truth table the expression for the output parity bit is,

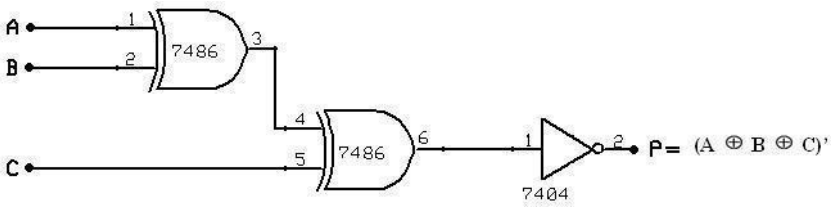
$$P(A,B,C)=\Sigma(0,3,5,6)$$

Also written as,

$$P = A'B'C' + A'BC + AB'C + ABC' = (A \oplus B \oplus C)'$$

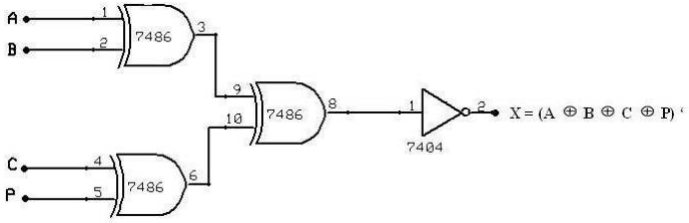
ODD PARITY GENERATOR

CIRCUIT DIAGRAM:



ODD PARITY CHECKER

CIRCUIT DIAGRAM:



ODD PARITY CHECKER

TRUTH TABLE:

SL.NO.	INPUT				OUTPUT
	(4 - Bit Message Received)				(Parity Error Check)
	A	B	C	P	X
1.	0	0	0	0	1
2.	0	0	0	1	0
3.	0	0	1	0	0
4.	0	0	1	1	1
5.	0	1	0	0	0
6.	0	1	0	1	1
7.	0	1	1	0	1
8.	0	1	1	1	0
9.	1	0	0	0	0
10.	1	0	0	1	1
11.	1	0	1	0	1
12.	1	0	1	1	0
13.	1	1	0	0	1
14.	1	1	0	1	0
15.	1	1	1	0	0
16.	1	1	1	1	1

From the truth table the expression for the output parity checker bit is,

$$X (A, B, C, P) = \Sigma (0, 3, 5, 6, 9, 10, 12, 15)$$

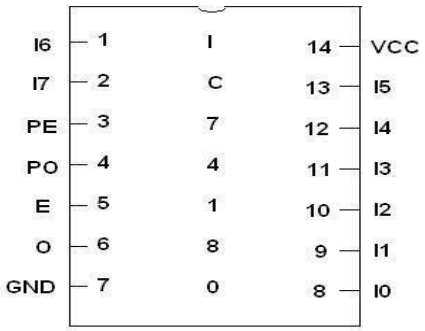
The above expression is reduced as,

$$X=(A \oplus B \oplus C \oplus P)$$

PROCEDURE:

1. Connections are given as per the circuit diagrams.
2. For all the ICs 7th pin is grounded and 14th pin is given +5 V supply.
3. Apply the inputs and verify the truth table for the Parity generator and checker.

PIN DIAGRAM FOR IC 74180:

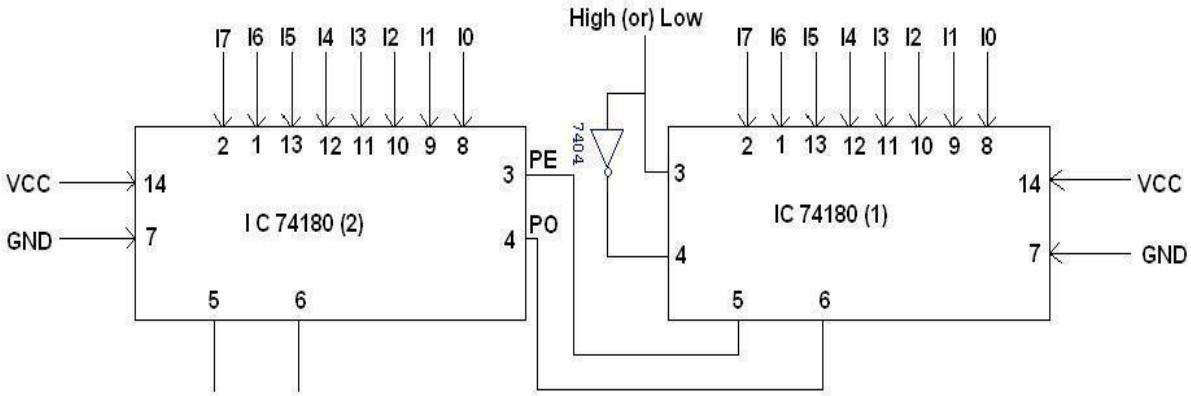


FUNCTION TABLE:

INPUTS			OUTPUTS	
Number of High Data Inputs (I0 – I7)	PE	PO	∑E	∑O
EVEN	1	0	1	0
ODD	1	0	0	1
EVEN	0	1	0	1
ODD	0	1	1	0
X	1	1	0	0
X	0	0	1	1

16 BIT ODD/EVEN PARITY GENERATOR

LOGIC DIAGRAM:

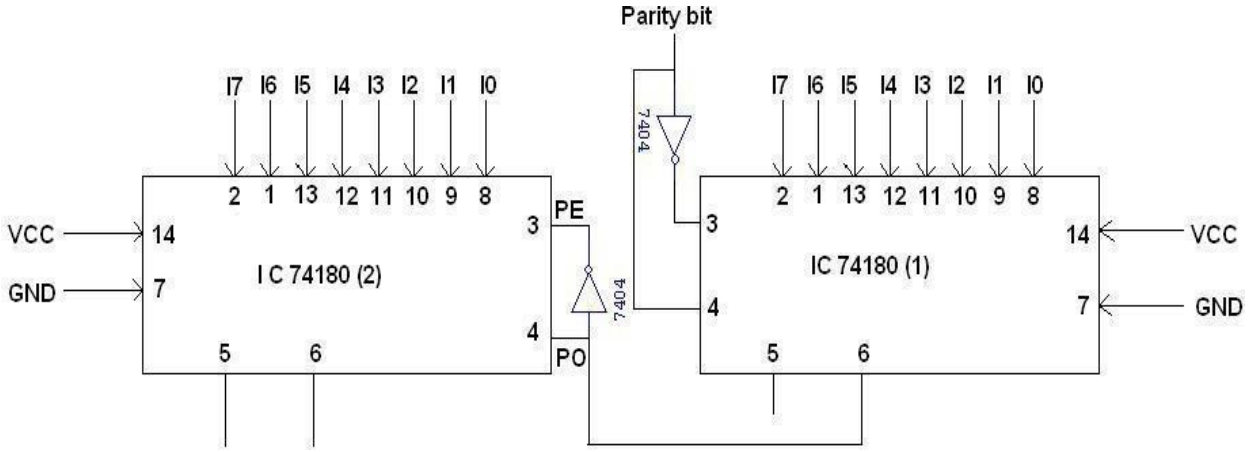


TRUTH TABLE:

I7 I6 I5 I4 I3 I2 I1 I0	I7 I6 I5 I4 I3 I2 I1 I0	Active	∑E	∑O
1 1 0 0 0 0 0 0	1 1 0 0 0 0 0 0	1	1	0
1 1 0 0 0 0 0 0	1 1 0 0 0 0 0 0	0	0	1
1 1 0 0 0 0 0 0	0 1 0 0 0 0 0 0	0	1	0

16 BIT ODD/EVEN PARITY CHECKER

LOGIC DIAGRAM



TRUTH TABLE:

I7 I6 I5 I4 I3 I2 I1 I0	I7'I6'I5'I4'I3'I2'11' I0'	Active	∑E	∑O
0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 0	1	1	0
0 0 0 0 0 1 1 0	0 0 0 0 0 1 1 0	0	1	0
0 0 0 0 0 1 1 0	0 0 0 0 0 1 1 0	1	0	1

RESULT:

Thus the three bit and 16 bit odd Parity generator and checker circuits were designed, implemented and their truth tables were verified.

EX. NO.: 7
DATE:

DESIGN AND IMPLEMENTATION OF ENCODER AND DECODER

AIM:

To design and implement encoder and decoder using logic gates and study of IC 7445 and IC 74147.

APPARATUS REQUIRED:

SL. NO.	COMPONENTS	SPECIFICATION	QTY.
1.	3 I/P NAND GATE	IC 7410	2
2.	OR GATE	IC 7432	3
3.	NOT GATE	IC 7404	1
2.	IC TRAINER KIT	-	1
3.	PATCH CORDS	-	27

THEORY:

ENCODER:

An encoder is a digital circuit that perform inverse operation of a decoder. An encoder has 2^n input lines and n output lines. In encoder the output lines generates the binary code corresponding to the input value. In octal to binary encoder it has eight inputs, one for each octal digit and three output that generate the corresponding binary code. In encoder it is assumed that only one input has a value of one at any given time otherwise the circuit is meaningless. It has an ambiguila that when all inputs are zero the outputs are zero. The zero outputs can also be generated when D0 = 1.

DECODER:

A decoder is a multiple input multiple output logic circuit which converts coded input into coded output where input and output codes are different. The

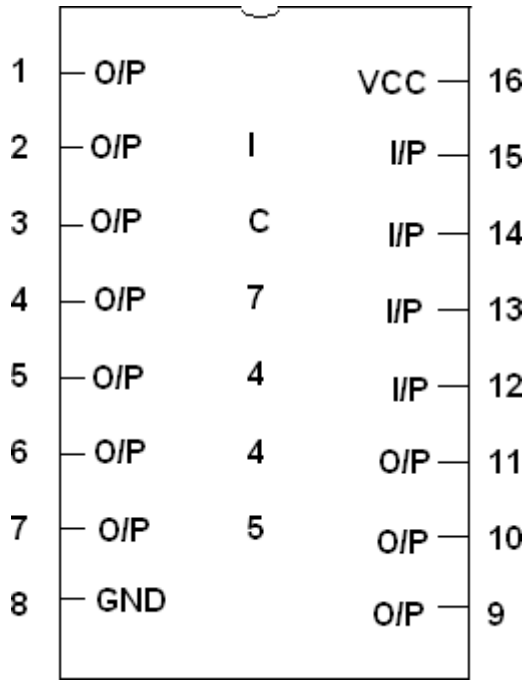
input code generally has fewer bits than the output code. Each input code word produces a different output code word i.e there is one to one mapping can be expressed in truth table. In the block diagram of decoder circuit the encoded information is present as n input producing 2^n possible outputs. 2^n output values are from 0 through out $2^n – 1$.

PROCEDURE:

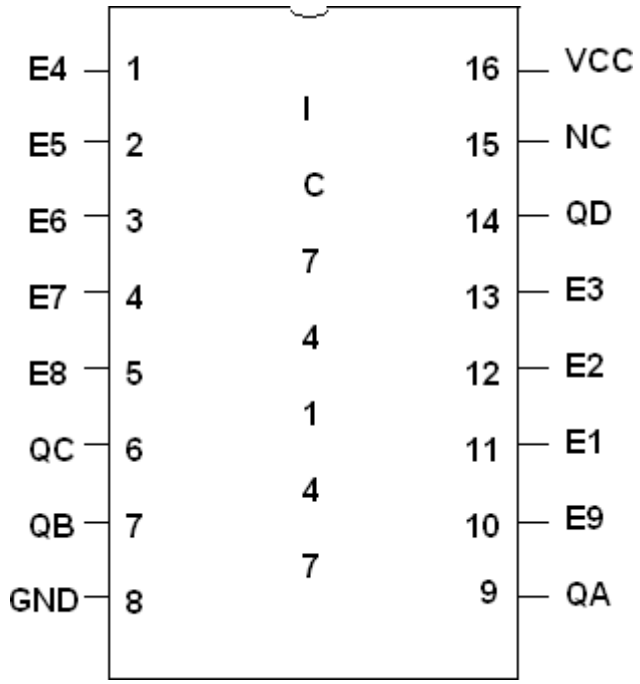
1. Connections are given as per circuit diagram.
2. Logical inputs are given as per circuit diagram.
3. Observe the output and verify the truth table.

BCD TO DECIMAL DECODER:

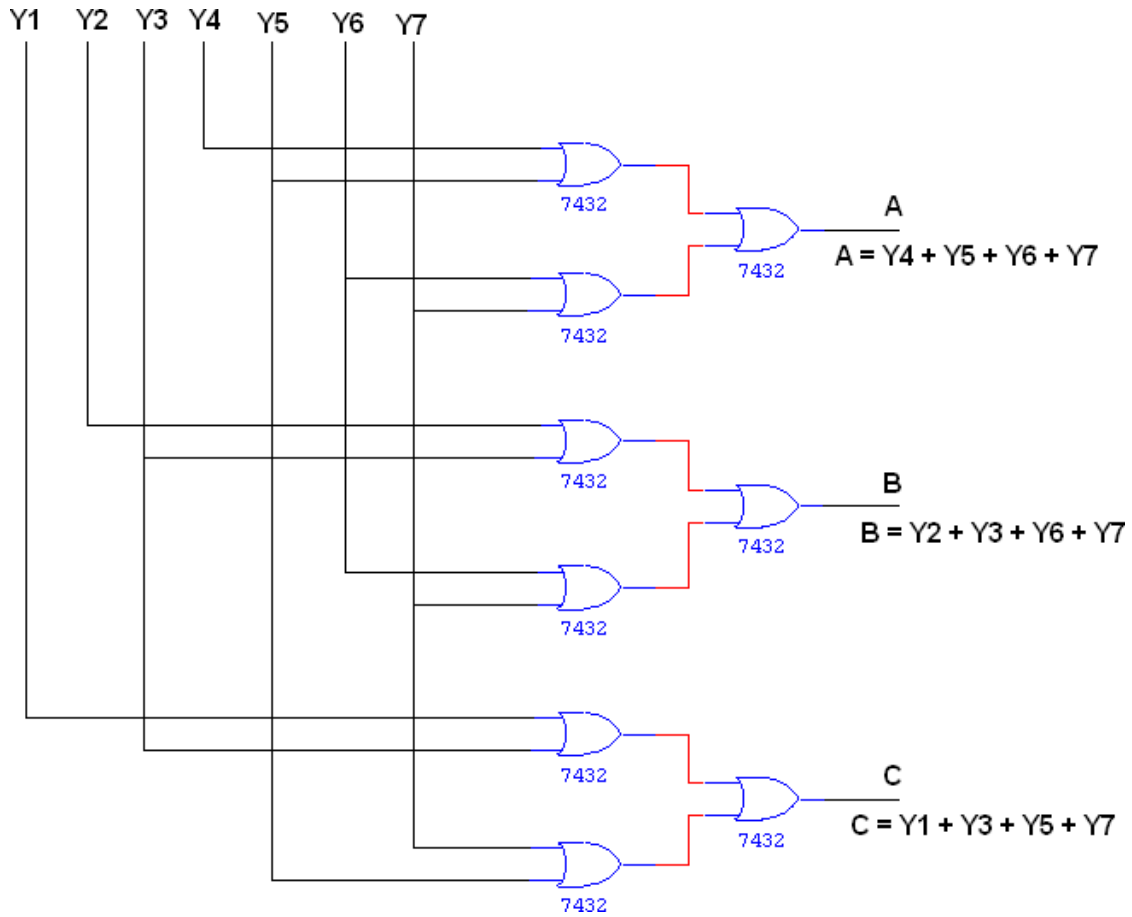
PIN DIAGRAM FOR IC 74155: 2x4 Decoder



PIN DIAGRAM FOR IC 74147(Encoder)



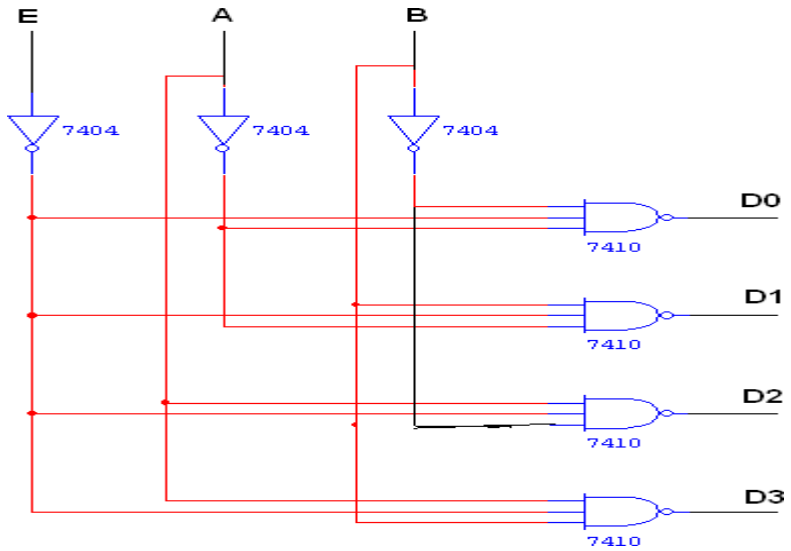
LOGIC DIAGRAM FOR ENCODER:



TRUTH TABLE:

INPUT							OUTPUT		
Y1	Y2	Y3	Y4	Y5	Y6	Y7	A	B	C
1	0	0	0	0	0	0	0	0	1
0	1	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	1	1
0	0	0	1	0	0	0	1	0	0
0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	1	1	1	1

LOGIC DIAGRAM FOR DECODER:



TRUTH TABLE:

INPUT			OUTPUT			
E	A	B	D0	D1	D2	D3
1	0	0	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

RESULT:

EX. NO.: 8

DATE:

CONSTRUCTION AND VERIFICATION OF 4 BIT RIPPLE COUNTER AND MOD 10/MOD 12 RIPPLE COUNTER

AIM:

To design and verify 4 bit ripple counter mod 10/ mod 12 ripple counter.

APPARATUS REQUIRED:

SL.NO.	COMPONENTS	SPECIFICATION	QTY.
1.	JK FLIP FLOP	IC 7476	2
2.	NAND GATE	IC 7400	1
3.	IC TRAINER KIT	-	1
4.	PATCH CORDS	-	30

THEORY:

A counter is a register capable of counting number of clock pulse arriving at its clock input. Counter represents the number of clock pulses arrived. A specified sequence of states appears as counter output. This is the main difference between a register and a counter. There are two types of counter, synchronous and asynchronous. In synchronous common clock is given to all flip flop and in asynchronous first flip flop is clocked by external pulse and then each successive flip flop is clocked by Q or Q output of previous stage. A soon the clock of second stage is triggered by output of first stage. Because of inherent propagation delay time all flip flops are not activated at same time which results in asynchronous operation.

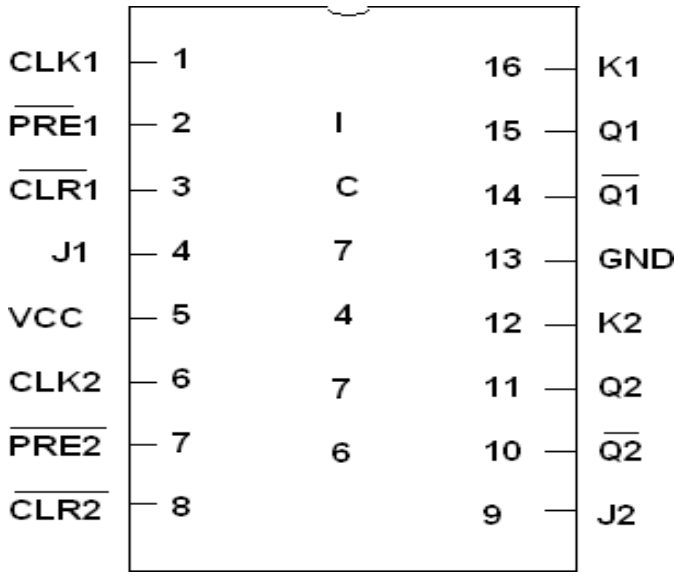
PROCEDURE:

(iv) Connections are given as per circuit diagram.

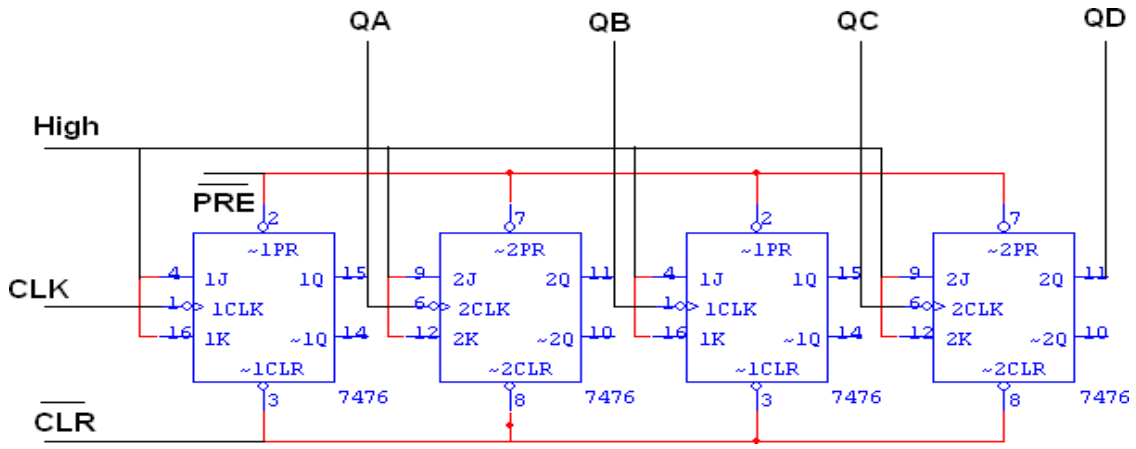
(v) Logical inputs are given as per circuit diagram.

(vi) Observe the output and verify the truth table.

PIN DIAGRAM FOR IC 7476:



LOGIC DIAGRAM FOR 4 BIT RIPPLE COUNTER:



TRUTH TABLE:

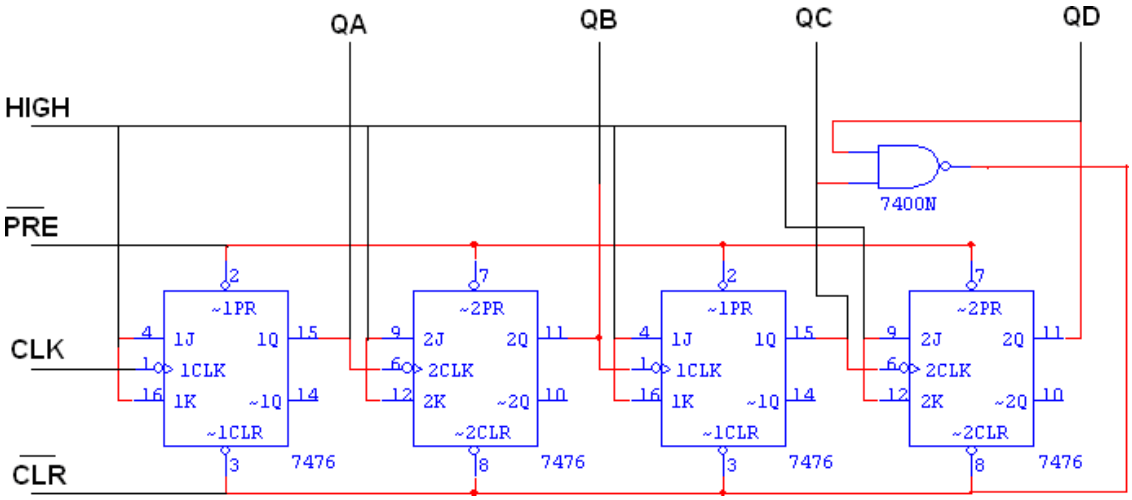
CLK	QD	QC	QB	QA
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1
16	0	0	0	0

MOD - 10 RIPPLE COUNTER

TRUTH TABLE:

CLK	QD	QC	QB	QA
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	0	0	0	0

LOGIC DIAGRAM:

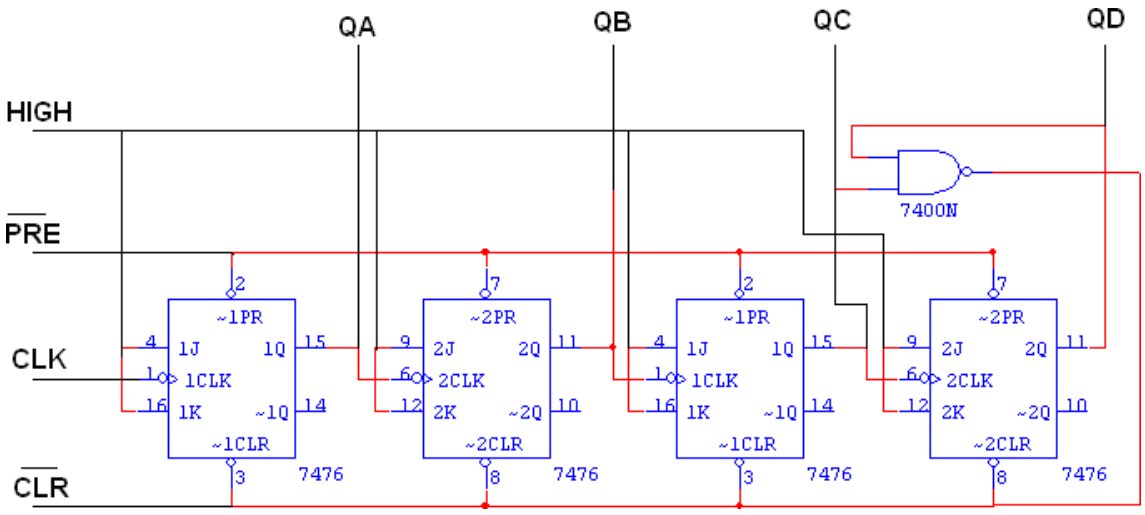


MOD - 12 RIPPLE COUNTER

TRUTH TABLE:

CLK	QD	QC	QB	QA
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	0	0	0	0

LOGIC DIAGRAM:



RESULT:

EX. NO.: 9 **MULTIPLEXER AND DEMULTIPLEXER**
DATE:

AIM:

To design and implement the multiplexer and demultiplexer using logic gates and study of IC 74150 and IC 74154.

APPARATUS REQUIRED:

SL. NO.	COMPONENTS	SPECIFICATION	QTY.
1.	3 I/P AND GATE	IC 7411	2
2.	OR GATE	IC 7432	1
3.	NOT GATE	IC 7404	1
2.	IC TRAINER KIT	-	1
3.	PATCH CORDS	-	32

THEORY:

MULTIPLEXER:

Multiplexer means transmitting a large number of information units over a smaller number of channels or lines. A digital multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line. The selection of a particular input line is controlled by a set of selection lines. Normally there are 2^n input line and n selection lines whose bit combination determine which input is selected.

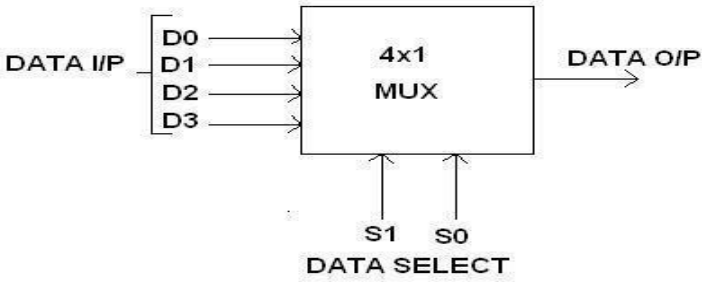
DEMULTIPLEXER:

The function of Demultiplexer is in contrast to multiplexer function. It takes information from one line and distributes it to a given number of output lines. For this reason, the demultiplexer is also known as a data distributor. Decoder can also be used as demultiplexer.

In the 1: 4 demultiplexer circuit, the data input line goes to all of the AND gates. The data select lines enable only one gate at a time and the data on the data input line will pass through the selected gate to the associated data output line.

4:1 MULTIPLEXER

BLOCK DIAGRAM FOR 4:1 MULTIPLEXER:



FUNCTION TABLE:

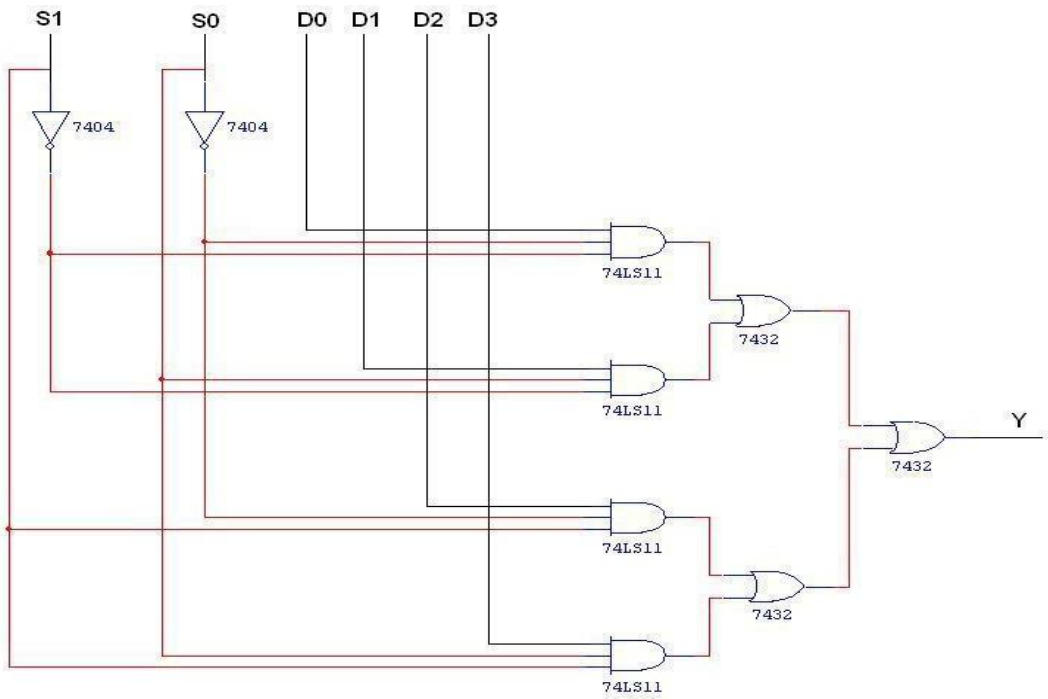
S1	S0	INPUTS Y
0	0	D0 → D0 S1' S0'
0	1	D1 → D1 S1' S0
1	0	D2 → D2 S1 S0'
1	1	D3→D3S1S0

Y=D0S1'S0'+D1S1'S0+D2S1S0'+D3S1S0

TRUTH TABLE:

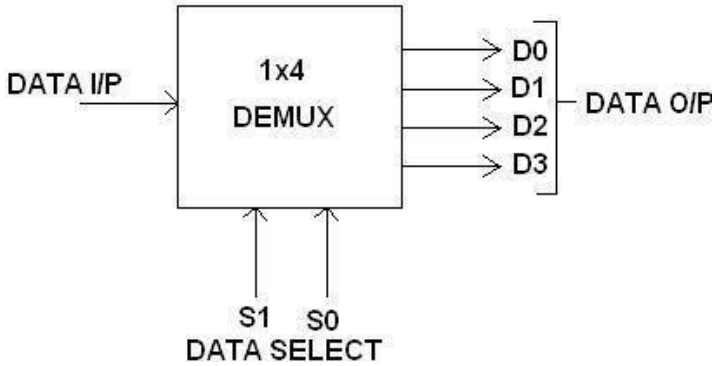
S1	S0	Y = OUTPUT
0	0	D0
0	1	D1
1	0	D2
1	1	D3

CIRCUIT DIAGRAM FOR MULTIPLEXER:



1:4 DEMULTIPLEXER

BLOCK DIAGRAM FOR 1:4 DEMULTIPLEXER:



FUNCTION TABLE:

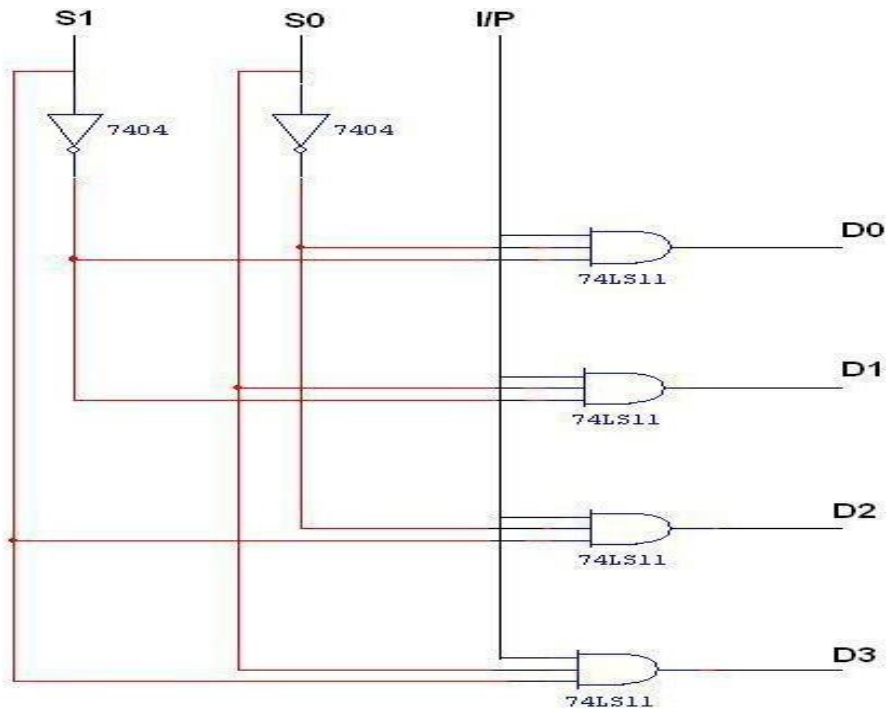
S1	S0	INPUT
0	0	X→D0=XS1'S0'
0	1	X→D1=XS1'S0
1	0	X→D2=XS1S0'
1	1	X→D3=XS1S0

Y=XS1'S0'+XS1'S0+XS1S0'+XS1S0

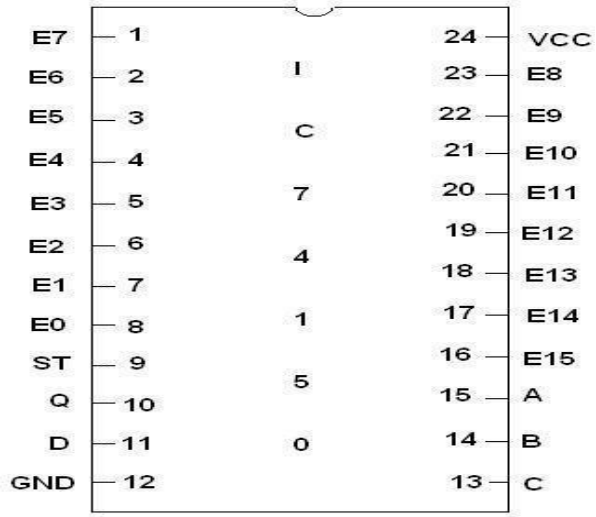
TRUTH TABLE:

INPUT			OUTPUT			
S1	**S0**	**I/P**	**D0**	**D1**	**D2**	**D3**
0	0	0	0	0	0	0
0	0	1	1	0	0	0
0	1	0	0	0	0	0
0	1	1	0	1	0	0
1	0	0	0	0	0	0
1	0	1	0	0	1	0
1	1	0	0	0	0	0
1	1	1	0	0	0	1

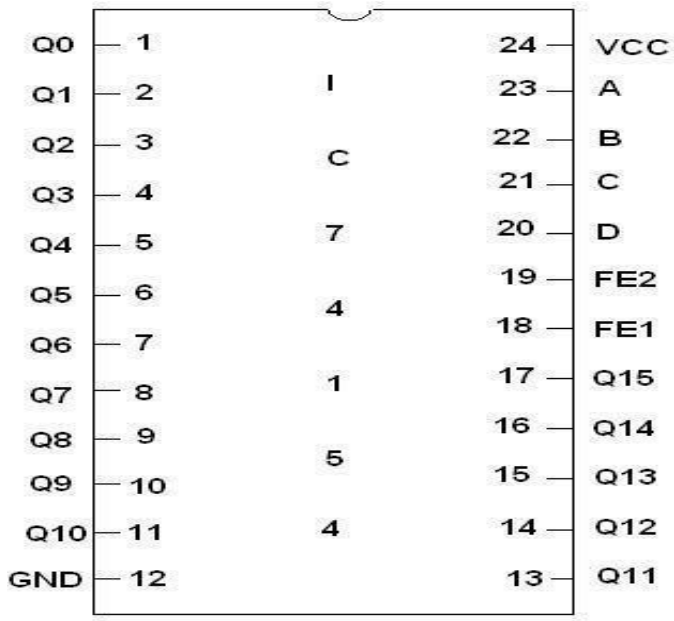
LOGIC DIAGRAM FOR DEMULTIPLEXER:



PIN DIAGRAM FOR IC 74150:



PIN DIAGRAM FOR IC 74154:



PROCEDURE:

(i) Connections are given as per circuit diagram.
(ii) Logical inputs are given as per circuit diagram.
(iii) Observe the output and verify the truth table.

RESULT:

Thus the multiplexer and demultiplexer using logic gates are designed and implemented.

Ex. No.: 10 **SHIFT REGISTER**
DATE:

AIM:

To design and implement the following shift registers
(i) Serial in serial out
(ii) Serial in parallel out
(iii) Parallel in serial out
(iv) Parallel in parallel out

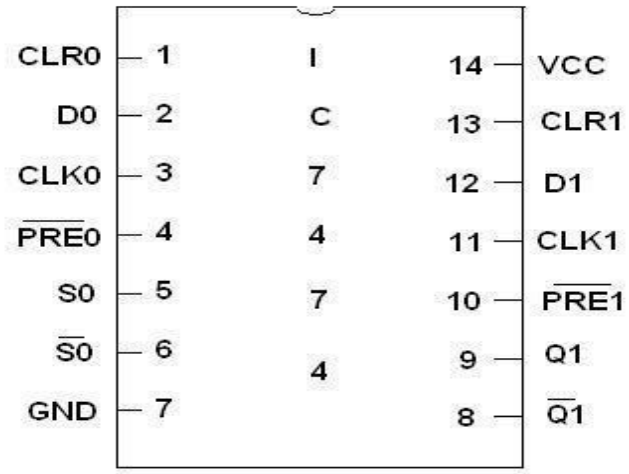
APPARATUS REQUIRED:

SL.NO.	COMPONENT	SPECIFICATION	QTY.
1.	D FLIP FLOP	IC 7474	2
2.	OR GATE	IC 7432	1
3.	IC TRAINER KIT	-	1
4.	PATCH CORDS	-	35

THEORY:

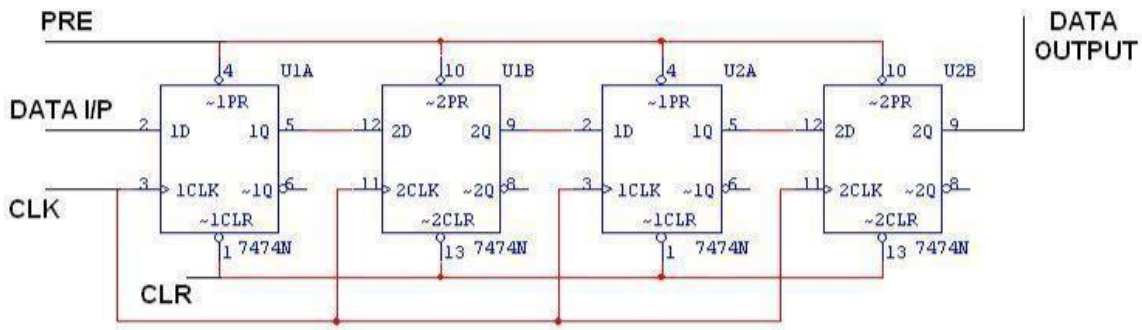
A register is capable of shifting its binary information in one or both directions is known as shift register. The logical configuration of shift register consist of a D-Flip flop cascaded with output of one flip flop connected to input of next flip flop. All flip flops receive common clock pulses which causes the shift in the output of the flip flop. The simplest possible shift register is one that uses only flip flop. The output of a given flip flop is connected to the input of next flip flop of the register. Each clock pulse shifts the content of register one bit position to right.

PIN DIAGRAM OF IC 7474:



SERIAL IN SERIAL OUT

LOGIC DIAGRAM:

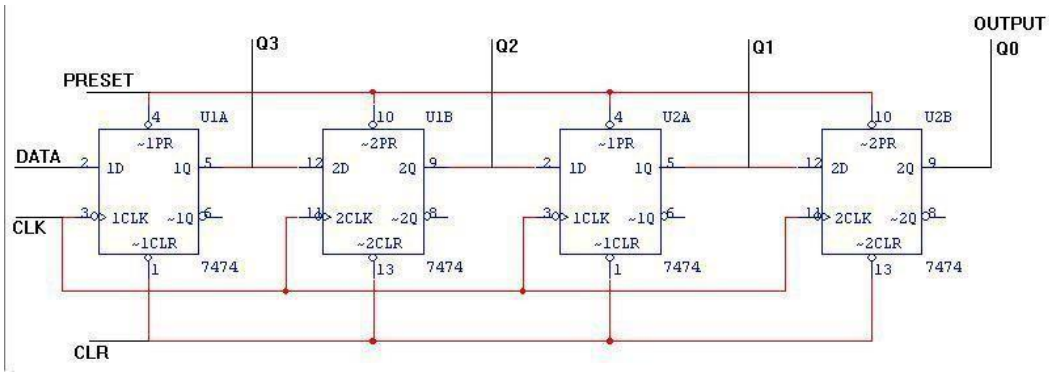


TRUTH TABLE:

CLK	Serial In	Serial Out
1	1	0
2	0	0
3	0	0
4	1	1
5	X	0
6	X	0
7	X	1

SERIAL IN PARALLEL

OUT LOGIC DIAGRAM:

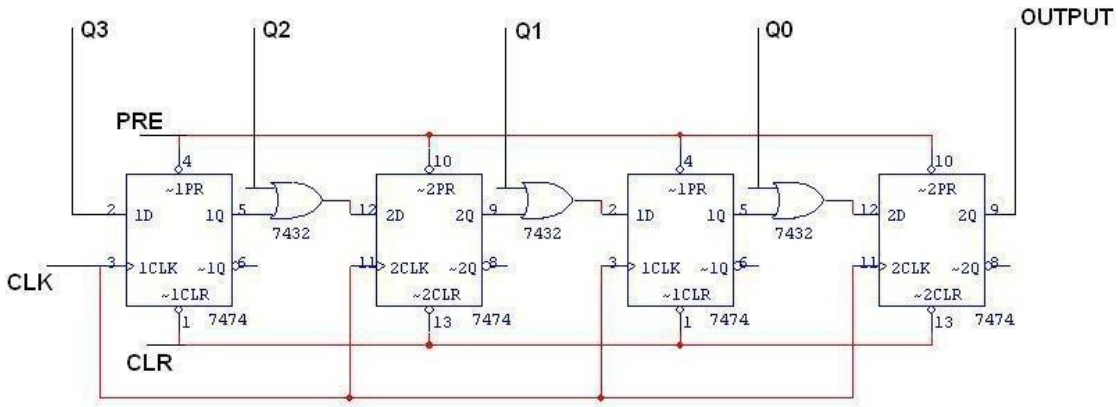


TRUTH TABLE:

CLK	DATA	OUTPUT			
		Q_A	Q_B	Q_C	Q_D
1	1	1	0	0	0
2	0	0	1	0	0
3	0	0	0	1	1
4	1	1	0	0	1

PARALLEL IN SERIAL OUT

LOGIC DIAGRAM:

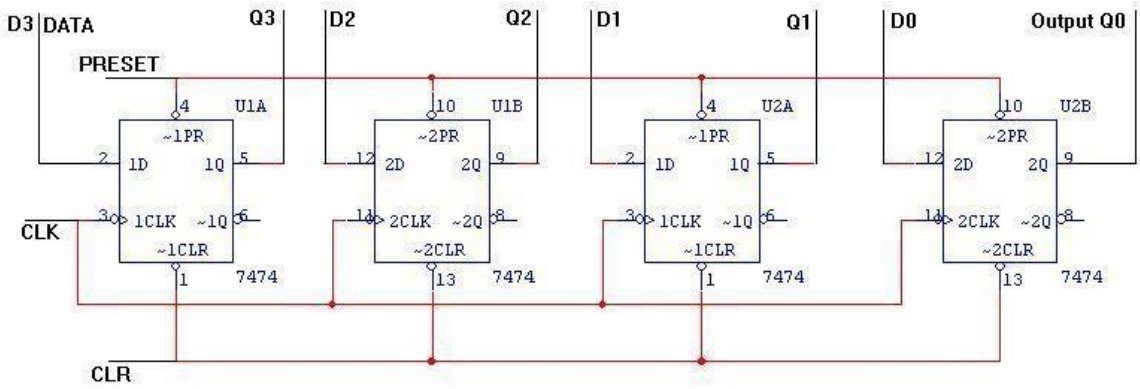


TRUTH TABLE:

CLK	Q3	Q2	Q1	Q0	O/P
0	1	0	0	1	1
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	1

PARALLEL IN PARALLEL OUT

LOGIC DIAGRAM:



TRUTH TABLE:

CLK	DATA INPUT				OUTPUT			
	D_A	D_B	D_C	D_D	Q_A	Q_B	Q_C	Q_D
1	1	0	0	1	1	0	0	1
2	1	0	1	0	1	0	1	0

PROCEDURE:

(i) Connections are given as per circuit diagram.
(ii) Logical inputs are given as per circuit diagram.
(iii) Observe the output and verify the truth table.

RESULT:

The Serial in serial out, Serial in parallel out, Parallel in serial out and Parallel in parallel out shift registers are designed and implemented.

Ex. No.: 11 **SYNCHRONOUS AND ASYNCHRONOUS COUNTER**
DATE:

AIM:

To design and implement synchronous and asynchronous counter.

APPARATUS REQUIRED:

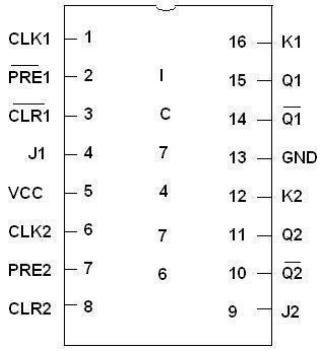
S.NO.	NAME OF THE APPARATUS	RANGE	QUANTITY
1.	Digital IC trainer kit		1
2.	JK Flip Flop	IC 7473	2
3.	D Flip Flop	IC 7473	1
4.	NAND gate	IC 7400	1
5.	Connecting wires		As required

THEORY:

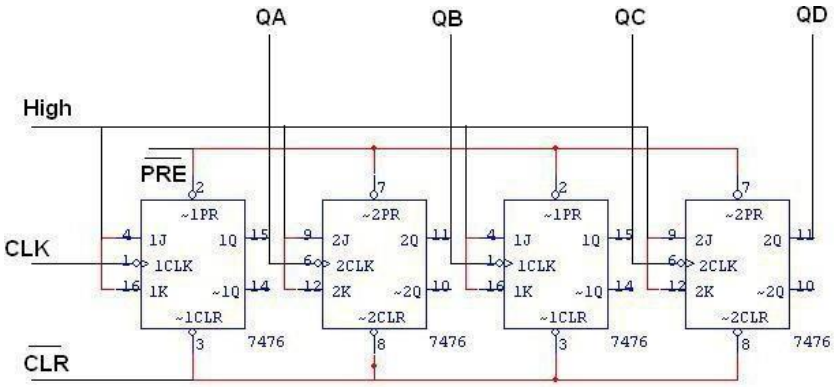
Asynchronous decade counter is also called as ripple counter. In a ripple counter the flip flop output transition serves as a source for triggering other flip flops. In other words the clock pulse inputs of all the flip flops are triggered not by the incoming pulses but rather by the transition that occurs in other flip flops. The term asynchronous refers to the events that do not occur at the same time. With respect to the counter operation, asynchronous means that the flip flop within the counter are not made to change states at exactly the same time, they do not because the clock pulses are not connected directly to the clock input of each flip flop in the counter.

A counter is a register capable of counting number of clock pulse arriving at its clock input. Counter represents the number of clock pulses arrived. A specified sequence of states appears as counter output. This is the main difference between a register and a counter. There are two types of counter, synchronous and asynchronous. In synchronous common clock is given to all flip flop and in asynchronous first flip flop is clocked by external pulse and then each successive flip flop is clocked by Q or Q output of previous stage. A soon the clock of second stage is triggered by output of first stage. Because of inherent propagation delay time all flip flops are not activated at same time which results in asynchronous operation.

PIN DIAGRAM FOR IC 7476:



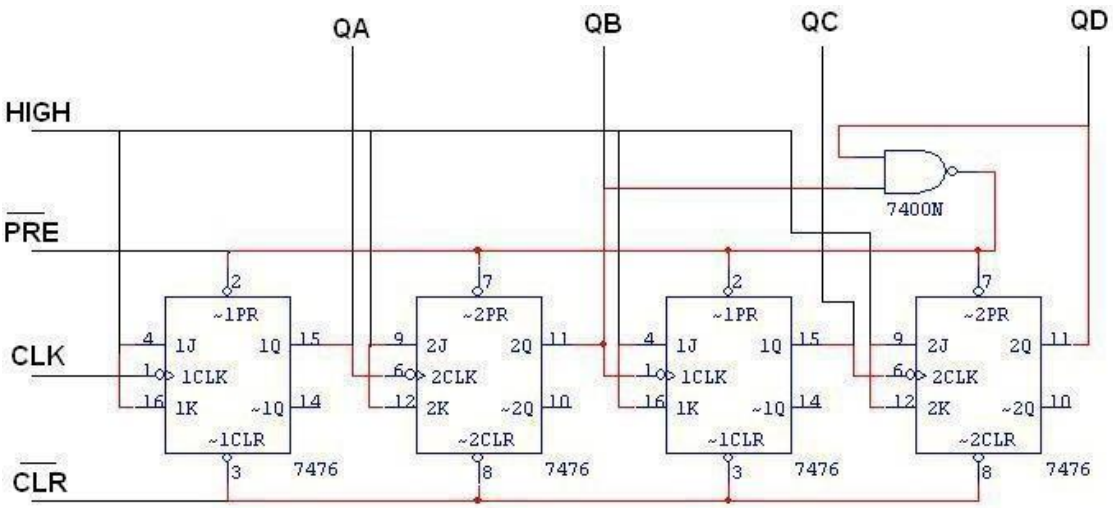
CIRCUIT DIAGRAM:



TRUTH TABLE:

CLK	QA	QB	QC	QD
0	0	0	0	0
1	1	0	0	0
2	0	1	0	0
3	1	1	0	0
4	0	0	1	0
5	1	0	1	0
6	0	1	1	0
7	1	1	1	0
8	0	0	0	1
9	1	0	0	1
10	0	1	0	1
11	1	1	0	1
12	0	0	1	1
13	1	0	1	1
14	0	1	1	1
15	1	1	1	1

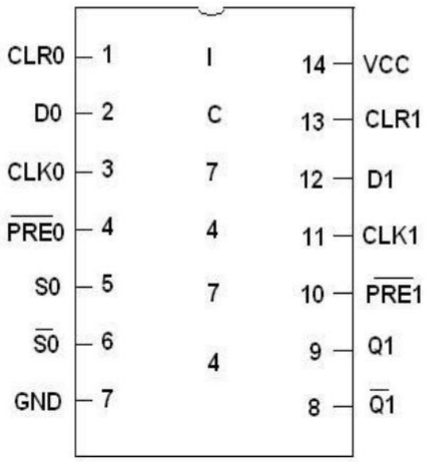
LOGIC DIAGRAM FOR MOD - 10 RIPPLE COUNTER:



TRUTH TABLE:

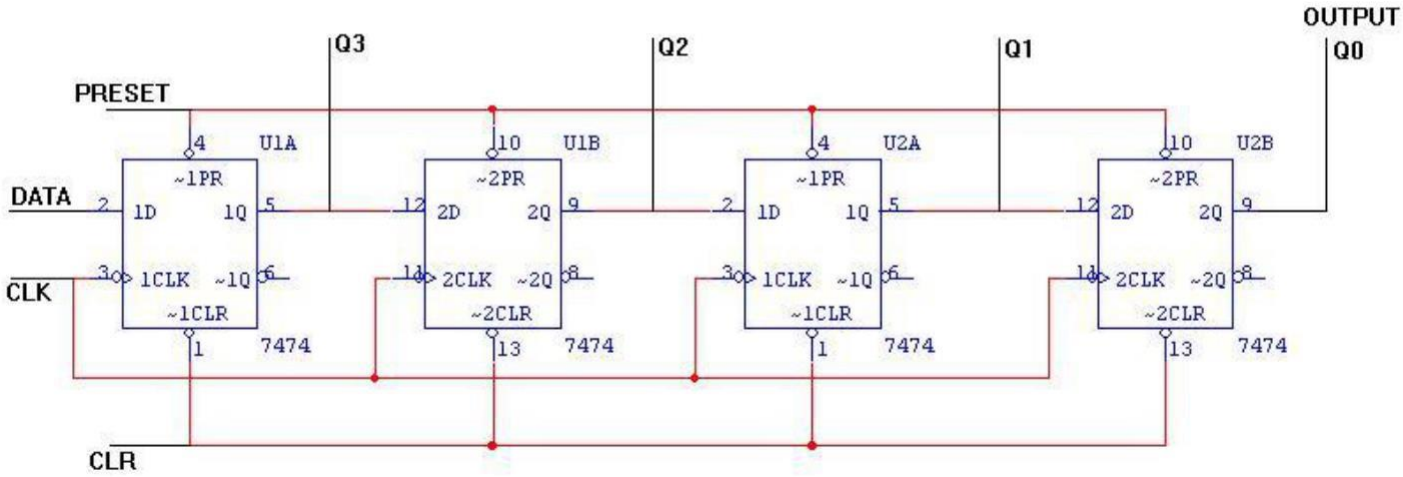
CLK	QA	QB	QC	QD
0	0	0	0	0
1	1	0	0	0
2	0	1	0	0
3	1	1	0	0
4	0	0	1	0
5	1	0	1	0
6	0	1	1	0
7	1	1	1	0
8	0	0	0	1
9	1	0	0	1
10	0	0	0	0

PIN DIAGRAM:



SYNCHRONOUS COUNTER

LOGIC DIAGRAM:



TRUTH TABLE:

CLK	DATA	OUTPUT			
		QA	QB	QC	QD
1	1	1	0	0	0
2	0	0	1	0	0
3	0	0	0	1	1
4	1	1	0	0	1

PROCEDURE:

(i) Connections are given as per circuit diagram.
(ii) Logical inputs are given as per circuit diagram.
(iii) Observe the output and verify the truth table.

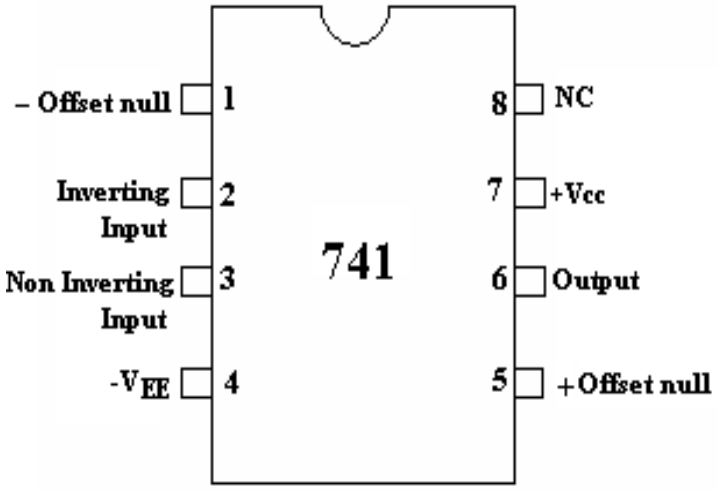
RESULT:

Thus the synchronous and asynchronous counter are designed and implemented.

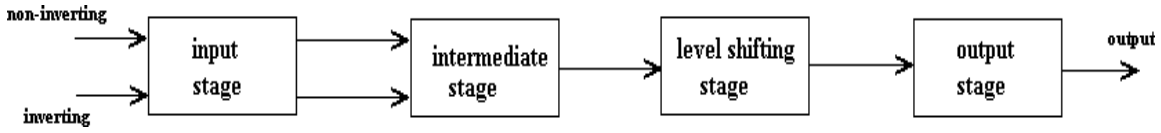
IC741-GeneralDescription:

The IC 741 is a high performance monolithic operational amplifier constructed using the planar epitaxial process. High common mode voltage range and absence of latch-up tendencies make the IC 741 ideal for use as voltage follower. The high gain and wide range of operating voltage provides superior performance in integrator, summing amplifier and general feedback applications.

Pin Configuration:



Block Diagram of Op-Amp:



Features:

1. No frequency compensation required.
2. Short circuit protection
3. Offset voltage null capability
4. Large common mode and differential voltage ranges
5. Low power consumption
6. No latch-up

SPECIFICATIONS:

1. Voltage gain A=∞typically2,00,000
2. Input resistance R_L=∞Ω,practically2MΩ
3. Output resistance R=0,practically75Ω
4. Bandwidth=∞Hz. It can be operate data any frequency
5. Common mode rejection ratio =∞
   (Ability of op amp to reject noise voltage)
6. Slew rate+∞V/μ sec
   (Rate of change of O/P voltage with respect to applied I/P)
7. When V_1=V_2,V_D=0
8. Input off set voltage (Rs≤10KΩ) max6mv
9. Input off set current=max200nA
10. Input bias current:500nA
11. Inputcapacitance:typicalvalue1.4pF
12. Offset voltage adjustment range:±15mV
13. Input voltage range: ±13V
14. Supply voltage rejection ratio:150μV/V
15. Output voltage swing:+13Vand–13VforR$_L$>2KΩ
16. Outputshort-circuitcurrent:25mA
17. supplycurrent:28mA
18. Powerconsumption:85mW
19. Transient response : rise time= 0.3μs Overshoot=5%

APPLICATIONS:-

1. AC and DC amplifiers.
2. Active filters.

EX. NO.: 12 **INTEGRATORANDDIFFERENTIATORUSINGOP-AMP**

DATE:

AIM:

To designaClipperandClamperusingop-ampIC741andtotesttheircharacteristics&Performance.

APPARATUSREQUIRED:

S.NO	COMPONENTS/EQUIPMENT	RANGE	QUANTITY
1.	IC 741	---	01
2.	RESISTORS	100Ω,1.5KΩ	Each02
		10KΩ,15KΩ	Each01
3.	CAPACITOR	0.1µf,0.01µf	Each01
		0.001µf,	05
4.	DIGITALTRAINERKIT	---	01
5.	SIGNALGENERATOR	(0-3)MHz	01
6.	CATHODERAYOSCILLOSCOPE	(0-30)MHz	01
7.	CONNECTINGWIRES	---	FEW

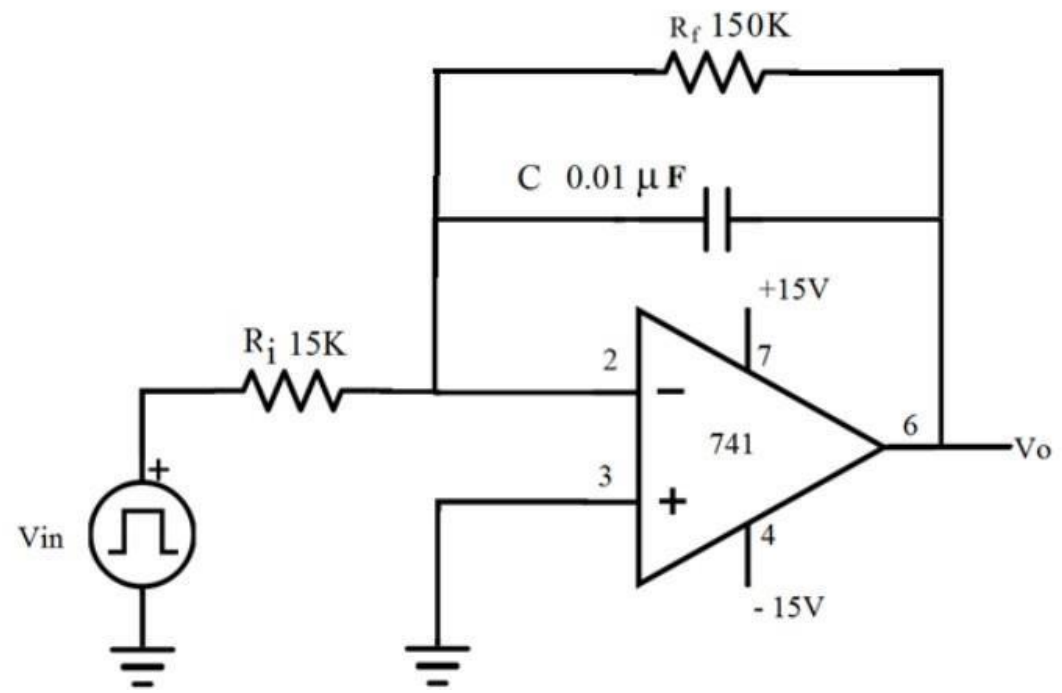
PROCEDURE:

1. From the given frequency f_a &f_b, the values of R_f,C_f,R_1&R_{comp} are calculated as given in the designprocedure.
2. Connect the circuit as shown in the circuit diagram.
3. Apply the sinusoidal input as the constant amplitude to the inverting terminal of op-amp.
4. Gradually increase the frequency & observe the output amplitude.
5. Calculate the gain with respect to frequency & plot its graph.

PROCEDURE:DIFFERENTIATOR

1. Select f_a equal to the highest frequency of the input signal to be differentiated. Calculate thecomponent values of C_1 & R_f.
2. Choose=20f_a & calculate the values of R_1&C_f , so that R_1C_1=R_fC_f.
3. Connect the components as shown in the circuit diagram.
4. Apply a sinusoidal& square wave input to the inverting terminal of op-amp through R_1C_1.
5. Observe the shape of the output signal for the given input in CRO.
6. Note down the reading and plot the graph of input versus output wave for both cases.

INTEGRATOR CIRCUITDIAGRAM:-



TABULATION:

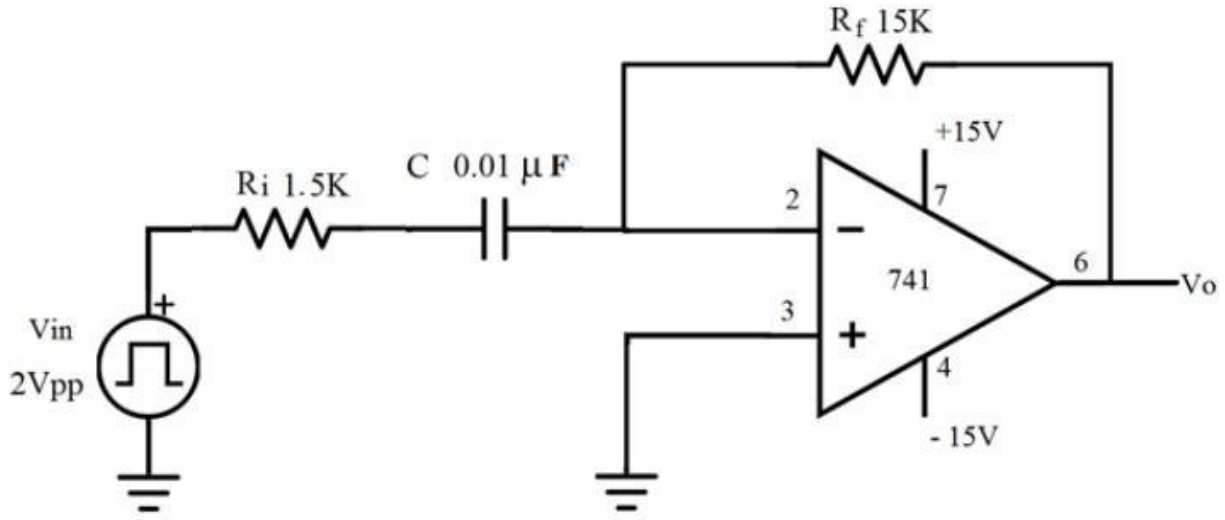
	Input	Output
Amplitude		
Time Period		

MODEL GRAPH: SINE WAVE FORM

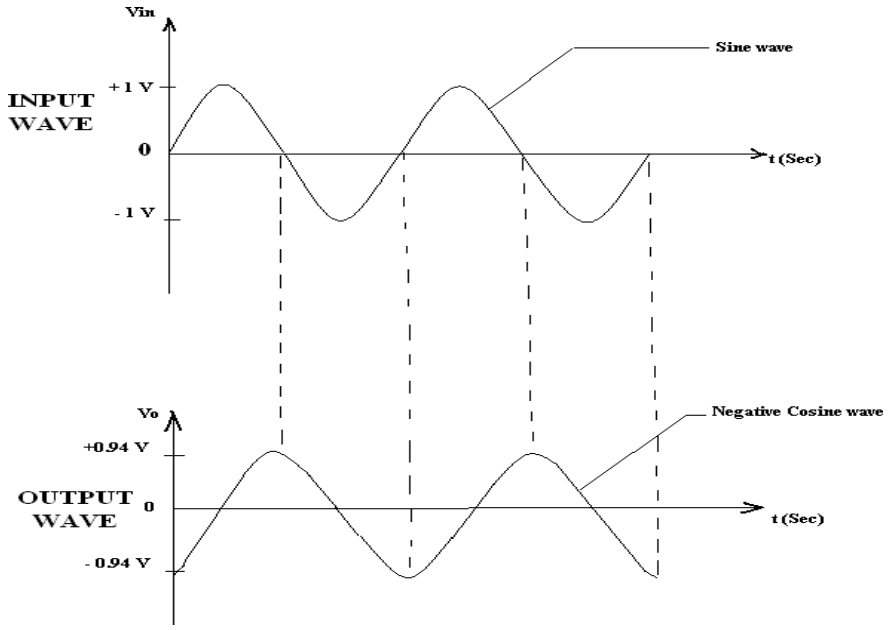
MODEL GRAPH: SQUARE WAVE FORM

	Input	Output
Amplitude		
Time period		

DIFFERENTIATOR:-CIRCUITDIAGRAM:



MODELGRAPH:



(ii)FORSINEWAVEINPUT

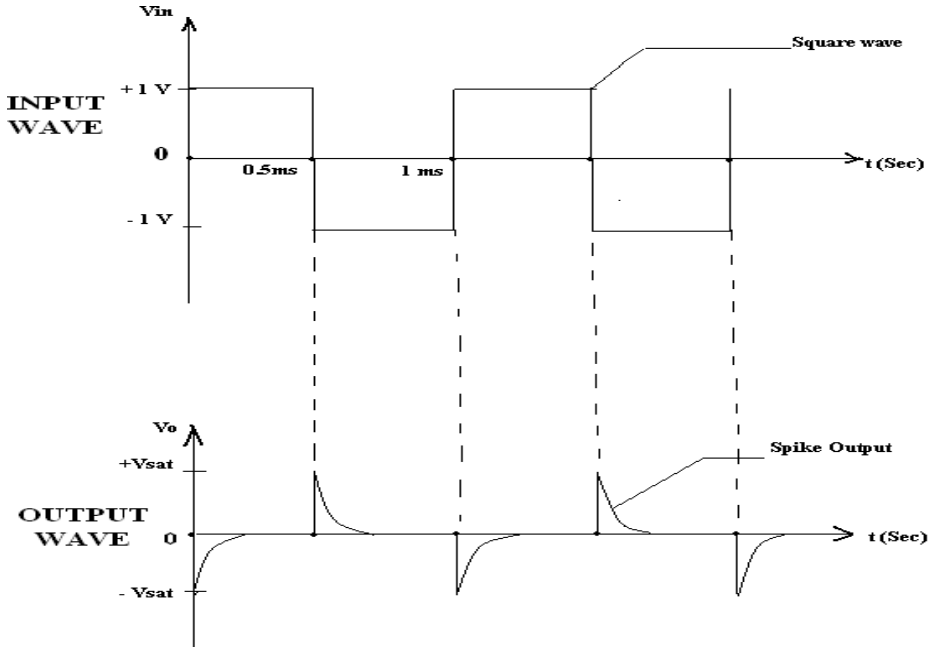
TABULATION:

	Input	Output
Amplitude		
Time period		

TABULATION:

	Input	Output
Amplitude		
Time period		

MODELGRAPH:SQUAREWAVEFORM



DESIGN PROCEDURE-(INTEGRATOR):-

Design of integrator to integrate at cut-off frequency 1 KHz.

Take fa = $\frac{1}{2\pi R_f C_f}$

= 1 KHz.

Always take $C_f < 1$ µf and

Let $C_f = 0.01$µf

$R_f = \frac{1}{2\pi C_f f_a}$

$R_f = 15.9K\Omega$

$R_f = 15\ K\Omega$

Take $f_b = \frac{1}{2\pi R_1 C_f}$ = 10 KHz.

$R_1 = \frac{1}{2\pi f_b C_f}$ = 1.59 KΩ.

$R_1 \approx 1.5K\Omega$

$R_{comp} = R_1 // R_f = \frac{R_1 R_f}{R_1 + R_f} \approx R_1$, Assume $R_L = 10K\Omega$

$R_{comp} = 1.5K\Omega$

DESIGN PROCEDURE-(DIFFERENTIATOR):-

Design a differentiator to differentiate an input signal that varies in frequency from 10Hz to 1KHz. Apply a sine wave & square wave of 2Vp-p & 1 KHz frequency & observe the output.

To find R_f & C_1

Given: $f_a = 1$ KHz.

$$f_a = \frac{1}{2\pi R_f C1}$$

$f_a = 1$KHz.

Assume $C_1 = 0.1\mu f$

$$R_f = 1.59K\Omega \approx 1.5K\Omega$$

To find R_1 & C_f

Select $f_b = 20f_a$ with $R_1C_1 = R_f C_f$

$$f_b = 20KHz = \frac{1}{2\pi R_1 C_1}$$

$$R_1 = 79.5\Omega \approx 100\Omega$$

$$C_f = \frac{R_1 C_1}{R_f} = \frac{82X0.1X10^{-6}}{1.5K\Omega}$$

$$C_f = 0.005\mu f.$$

$$R_{OM} \approx R_1 // R_f = 100\Omega$$

RESULT:

Thus an Integrator and Differentiator using op-amp are designed and their performance was successfully tested using op-amp IC741.

EX. NO: 13
DATE:

INVERTING, NON-INVERTING AND DIFFERENTIAL AMPLIFIER

AIM:

To design, construct and test inverting, non-inverting amplifier using IC 741.

APPARATUS REQUIRED:

S. No.	Name of the Apparatus	Range/Value	Qty
1.	Bread Board	-	1
2.	RPS	(0-30) V	2
3.	Dual Power Supply	±15 V	1
4.	Resistor	1k Ω,10k Ω	2,2
5.	IC 741 Op-Amp	-	1
6.	Connecting Wires	-	Few
7.	Function Generator	(0-3) MHz	1
8.	CRO	(0-30) MHz	1
9.	Voltmeter or Multi-meter	(0-30) V	1

DESIGN:

INVERTING AMPLIFIER:

To design an amplifier for the gain of -10.

Gain = R_f/R_1.

As the Gain is given negative,

the circuit is inverting amplifier.

Gain $A_v = R_f/R_1 = 10 \Rightarrow R_f = 10\ R_1$.

Let R_1= 1k, $R_f = 10 * R_1 = 10 * 1k = 10k$.

NON - INVERTING AMPLIFIER:

To design an amplifier for the gain of 11.

Gain = $1+ R_f/R_1$

As the Gain is given positive, the circuit is non-inverting amplifier.

Gain $A_v = 1+R_f/R_1 = 11 \Rightarrow R_f = 10\ R_1$.

Let R_1= 1k, $R_f = 10 * R_1 = 10 * 1k = 10k$.

THEORY:

INVERTING AMPLIFIER:

A typical inverting amplifier with input resistor $R_1$and a feedback resistor R_f is shown in the figure. Since the op-amp is assumed to be an ideal one the input bias current is zero and hence the non -inverting input terminal is at ground potential. The voltage at node „A" is Zero, as the non inverting input terminal is grounded

The nodal equation by KCL at node „A" is given by $V_i/R_1 + V_o/R_f = 0$ or $V_0 = -R_f (V_i/R_1)$.

NON- INVERTING AMPLIFIER:

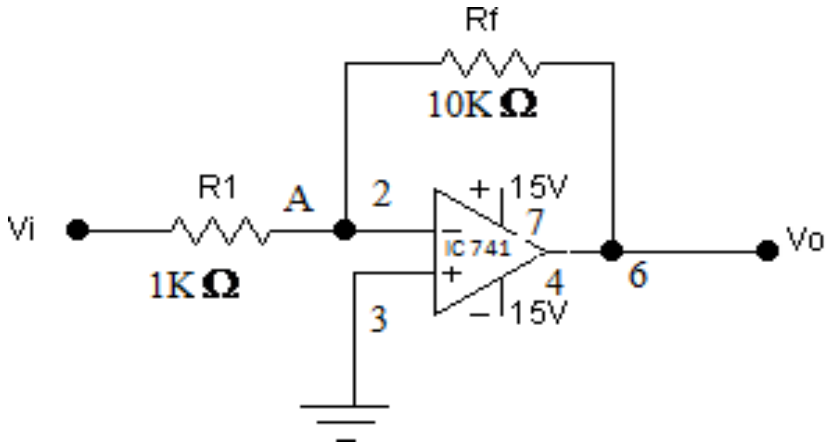
A typical non-inverting amplifier with input resistor R_1 and a feedback resistor R_f is shown in the figure. The input voltage is given to the positive terminal. The output voltage is given by $V_0= (1+R_f /R_1)\ V_i$

DIFFERENTIAL AMPLIFIER:

Basic differential amplifier is shown in figure, it amplifies the difference between the two input signal applied. The differential amplifier is characterized by the common mode rejection ratio (CMRR), which is the ratio of differential gain to common mode gain. The output voltage is given by $V_0 = (R_2 / R_1)(V_1 - V_2)$, where V_1 and V_2 are the input voltages.

CIRCUIT DIAGRAM:

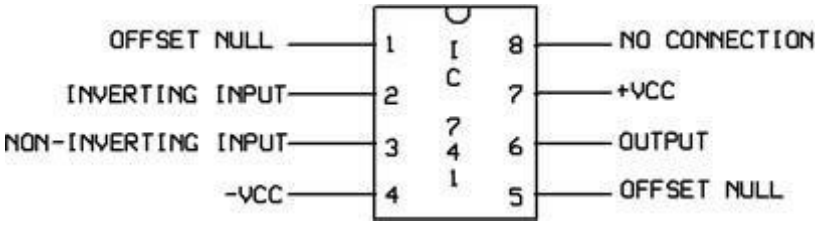
INVERTING AMPLIFIER



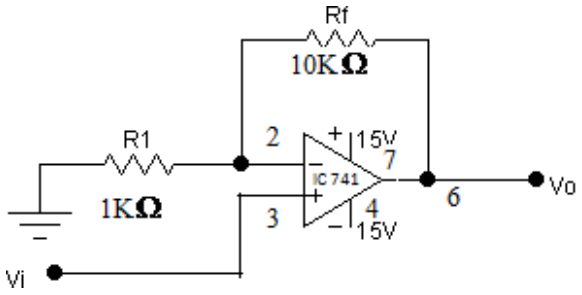
TABULATION:

Wave-form	Time Period in ms	Voltage in Volts	PracticalGain
Input (V_{in})			
Output (V_o)			

PIN DIAGRAM



NON INVERTING AMPLIFIER



TABULATION:

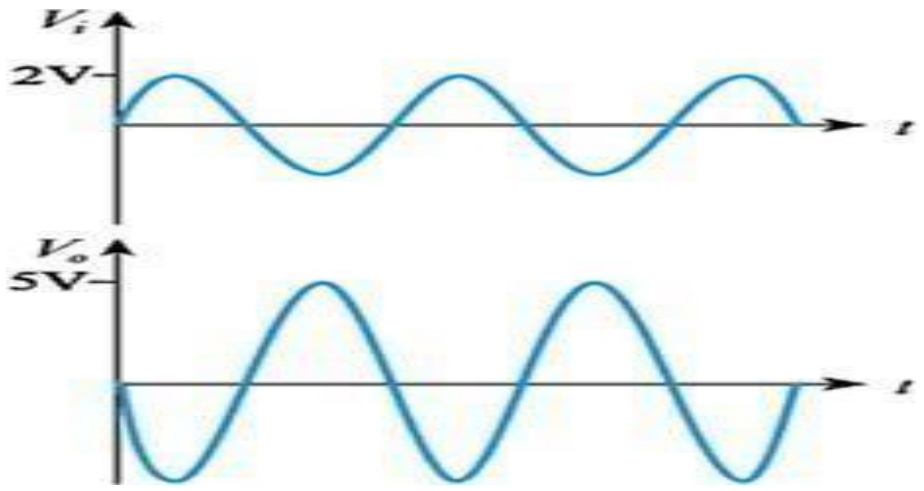
Wave-form	Time Period in ms	Voltage in Volts	PracticalGain
Input (V_{in})			
Output (V_o)			

PROCEDURE:

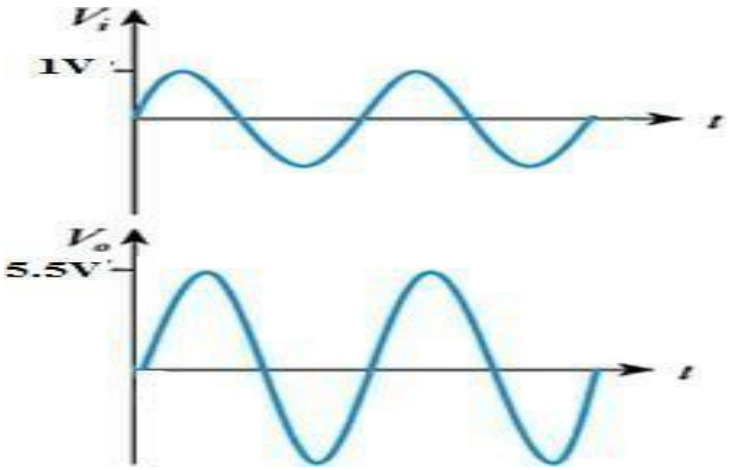
(i) Connect the inverting amplifier circuit as per the circuit diagram.
(ii) For various input voltage measure and record the output voltage.
(iii) Repeat the same for non- inverting and differential amplifier.

MODEL GRAPH:

INVERTING AMPLIFIER



NON-INVERTING AMPLIFIER



SPECIFICATION FOR IC 741

$+V_{cc} = +15V$, $-V_{cc} = -15V$

Ambient Temperature: 25^0 C

Input offset voltage : 6 mV(Max)

Input offset current : 200nA(Max)

Input bias current : 500nA(Max)

Input resistance : 2MΩ

Output resistance : 75Ω

Total Power dissipation : 85mW

RESULT:

The design and testing of the inverting, non-inverting amplifier is done and the input and output wave forms were drawn.

EX. NO: 14 **APPLICATIONS OF IC 741 AS ADDER, SUBTRACTOR, COMPARATOR**

DATE:

AIM:

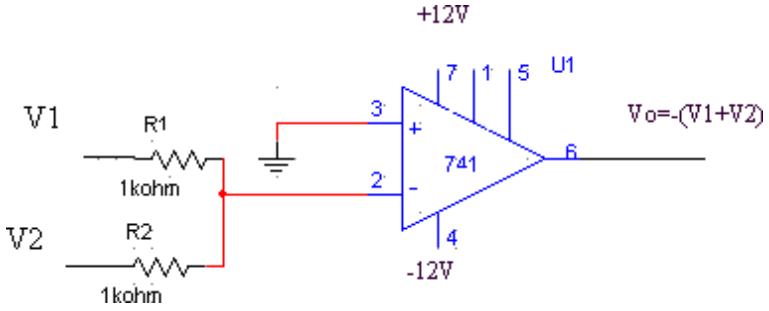
To study the applications of IC 741 as adder, subtractor, comparator

APPARATUS:

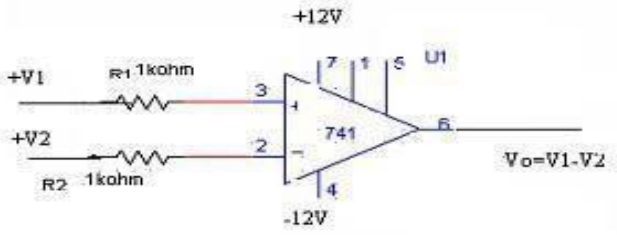
1. IC 741
2. Resistors (1KΩ)—4
3. Function generator
4. Regulated power supply
5. IC bread board trainer
6. CRO
7. Patch cards and CRO probes

CIRCUIT DIAGRAM:

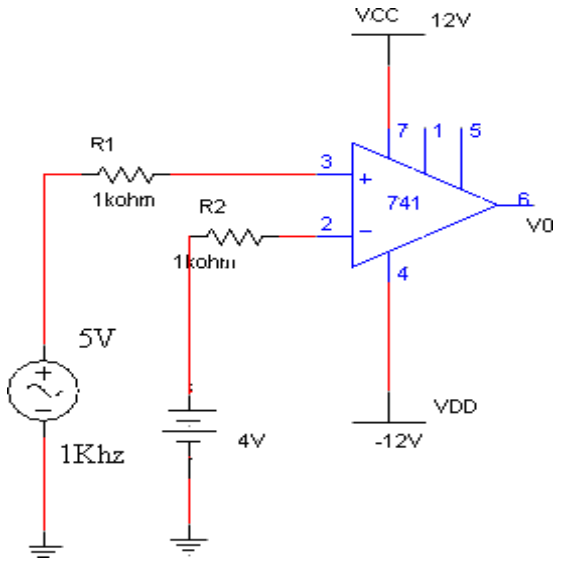
Adder:



Subtractor:



Comparator:



THEORY:

ADDER:

Op-Amp may be used to design a circuit whose output is the sum of several input signalssuchascircuitiscalledasummingamplifierorsummer.Wecanobtaineither inverting or non-inverting summer.

Thecircuitdiagramsshowsatwoinputinvertingsummingamplifier.Ithastwo input voltages V_1 and V_2, two input resistors R_1, R2and a feedback resistor R_f.

Assuming that op- amp is in ideal conditions and input bias current is assumed to be zero, there is no voltage drop across the resistor R_{comp} and hence the non-inverting input terminal is at ground potential.

By taking nodal equations

$V_1/R_1+V_2/R_2+V_0/R_f=0$

$V_0=-[(R_f/R_1)\ V_1+(R_f/R_2)\ V_2]$

And here $R_1=R_2=R_f=1K\Omega V_0=-(V_1+V_2)$

Thus output is inverted and sum of input.

SUBTRACTOR:

A basic differential amplifier can be used as a subtractor. It has two input signals V1and V2and two input resistances R1andR2 and a feedback resistor Rf.The input signals scaled to the desired values by selecting appropriate values for the external resistors.

From the figure, the output voltage of the differential amplifier with a gain of'1' is
$V_0=-R/R_f(V_2-V_1)$
$V_0=V_1-V_2$.
Also $R_1=R_2=R_f=1K\Omega$.

Thus, the output voltage eV_0 is equal to the voltage V_1applied to the non-inverting terminal minus voltage V_2applied to inverting terminal.
Hence the circuit is sub tractor.

COMPARATOR:

A comparator is a circuit which compares signal voltage applied at one input of an op-amp with a known reference Voltage at the other input. It is basically an open loop op-amp with output ±Vsat as in the ideal transfer characteristics.

It is clear that the change in the outputs that takes place with an increment in input Vi of only2mv.Thisistheuncertaintyregionwhereoutputcannotbedirectly defined There are basically 2 types of comparators.

1. Non inverting comparator and.
2. Inverting comparator.

The applications of comparator are zero crossing detector, window detector, time marker generator and phase meter.

OBSERVATIONS:

ADDER:

V_1(volts)	V_2(volts)	Theoretical $V_0=-(V_1+V_2)$	Practical $V_0=-(V_1+V_2)$

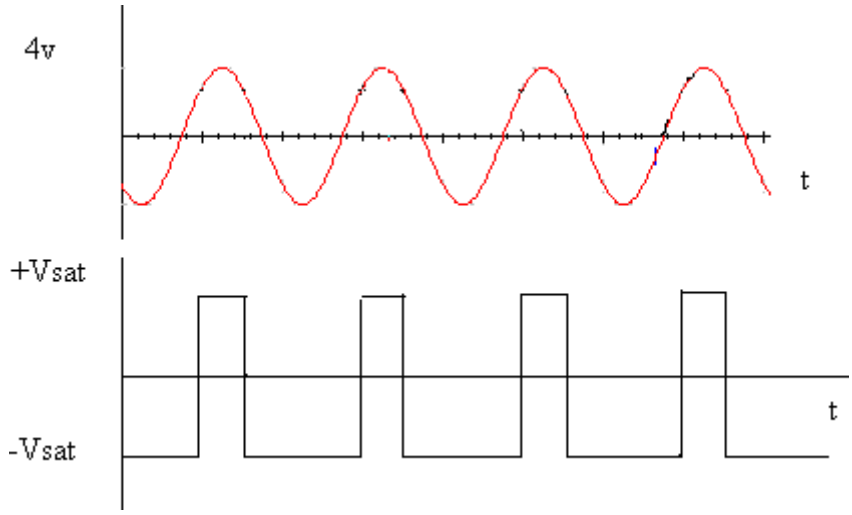
SUBTRACTOR:

V1(volts)	V2(volts)	Theoretical V0=(V1-V2)	Practical V0 =(V1-V2)

COMPARATOR:

Voltage input	V_{ref}	Observed square wave amplitude

MODEL GRAPH:



PROCEDURE:

ADDER:

1. Connections are made as per the circuit diagram.
2. Apply input voltage 1) V1=5V, V2=2V
   2) V1=5V, V2=5V
   3) V1=5V, V2=7V.
3. Using Millimeter measure the dc output voltage at the output terminal
4. For different values of V1and V2 measure the output voltage.

SUBTRACTOR:

1. Connections are made as per the circuit diagram.
2. Apply input voltage1) V_1=5v,V_2=2v

   2) V_1=5v,V_2=5v

   3) V_1=5v,V_2=7v.
3. Using Millimeter measure the dc output voltage at the output terminal.
4. For different values of $V_1$and V_2measure the output voltage.

COMPARATOR:

1. Connections are made as per the circuit diagram.
2. Select the sine wave of10V peak to peak, 1K Hz frequency.
3. Apply the reference voltage 2V and trace the input and output wave forms.
4. Superimpose input and output waveforms and measure sine wave amplitude with reference to V_{ref}.
5. Repeatsteps3and 4with referencevoltagesas2V, 4V,-2V,-4Vandobserve the wave forms.
6. Replace sine wave input with 5V dc voltage and V_{ref}=0V.
7. Observe dc voltage at output using CRO.
8. Slowly increase V_{ref}voltage and observe the change in saturation voltage.

PRECAUTIONS:

1. Make null adjustment before applying the input signal.
2. Maintain proper Vcc levels.

RESULT:

EX. NO: 15
DATE:

DESIGN OF ASTABLE MULTIVIBRATOR USING IC 555 TIMER

AIM:

To design and test an astable multivibrator for generating symmetrical andunsymmetrical square wave form for the given frequency and duty cycle.

APPARATUS REQUIRED:

S. No.	Name of the Apparatus	Range/Value	Qty
1.	Bread Board	-	1
2.	Resistor	3.6 k Ω, 7.2 k Ω	1, 2
3.	IC 555	-	1
4.	CRO	20 MHz.	1
5.	Capacitor	0.1µF, 0.01µF	1, 1
6.	RPS	(0-30) V/ 5V	1
7.	Diode		1
8.	Connecting Wires	-	Few

THEORY:

The 555 timer is connected as an astable multivibrator as shown in figure. In this mode of operation the timing capacitor charges up towards V_{cc} (assuming V_o is high initially) through (R_a + R_b) until the voltage across the capacitor reaches the threshold level (2/3) V_{cc}. At this point the internal upper comparator switches state causing the internal flip-flop output to go high. This turns on the discharge transistor and the timing capacitor C then discharges through R_b and the discharging transistor. The discharging continues until the capacitor voltage drops to (1/3) V_{cc} at which point the internal lower comparator switches states causing the internal flip-flop output to go low, turning off the discharge transistor. At this point the capacitor starts to charge again, thus completing the cycle.

DESIGN:

i. For Unsymmetrical waveform:

$f = 1/T = 1.44/(R_a+2R_b)C$;

Duty Cycle = D = t_{low}/ (t_{low} + t_{high}) => D= R_b/ (R_a +2R_b) ;

Where $t_{high} = 0.693(R_a + R_b)C$; $t_{low} = 0.69$;

Specifications: frequency = 1kHz; Duty cycle = 25%Design:

$t_{low} = 0.25ms = 0.693R_b C$;

Let C= 0.1µF => $R_b = 0.25/(0.693X0.1X10^{-6})$ =

$t_{high} = 0.693(R_a + R_b)C = 0.75$ ms => R_a =

- **For Symmetrical Wave form :**

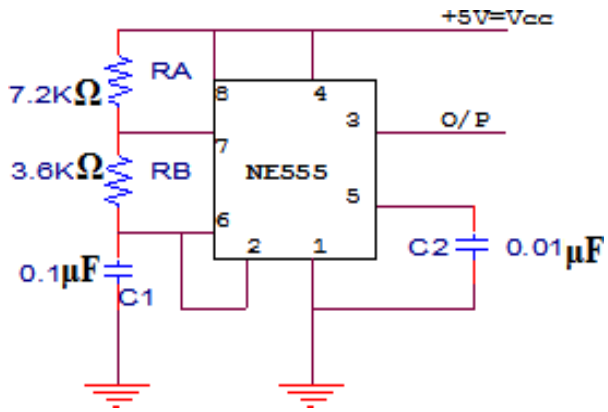
$t_{high} = 0.693 R_a C$; $t_{low} = 0.693 R_b C$

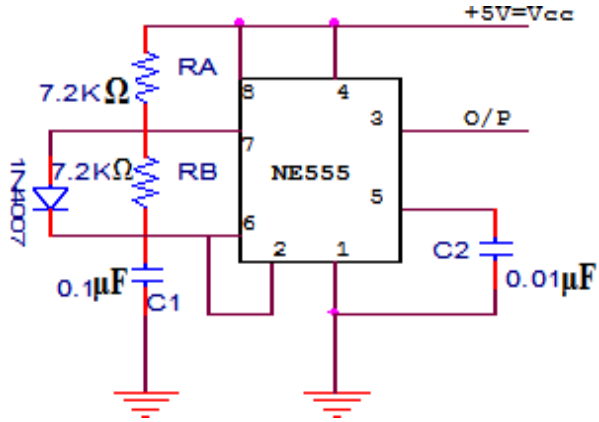
$f = 1/T = 1.44 /(R_a + R_b)C$ => $D= R_b /(R_a + R_b)$;

Specifications: frequency = 1 kHz; Duty cycle = 50% .

Design: $t_{low} = 0.5$ ms = $0.693 R_b C$;

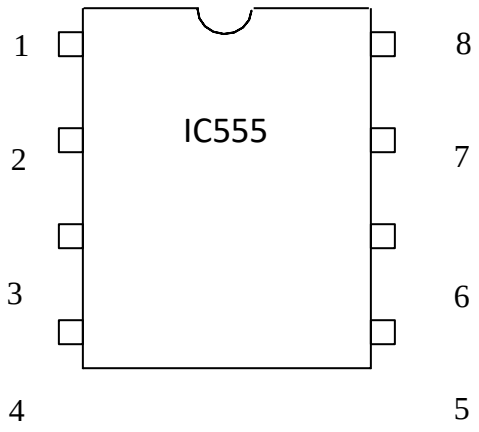
Let C= 0.1 µF; $R_b = t_{high} = 0.693 R_a C = 0.5$ ms ; R_a =





CIRCUIT DIAGRAM: Unsymmetrical: **Symmetrical:**

PIN DIAGRAM FOR IC555



1 = Ground, 2 = Trigger, 3 = output, 4 = Reset, 5 = Control voltage,

6 = Threshold, 7 = Discharge, 8 = +Vcc

PROCEDURE:

1. Connect the circuit as given using component values as obtained in designed part (i)
2. Observe and sketch the capacitor voltage waveform and output waveform.
3. Measure the frequency and duty cycle of the output waveform.
4. Connect the circuit using component values as obtained from designed part (ii).
5. Repeat step 2 and 3

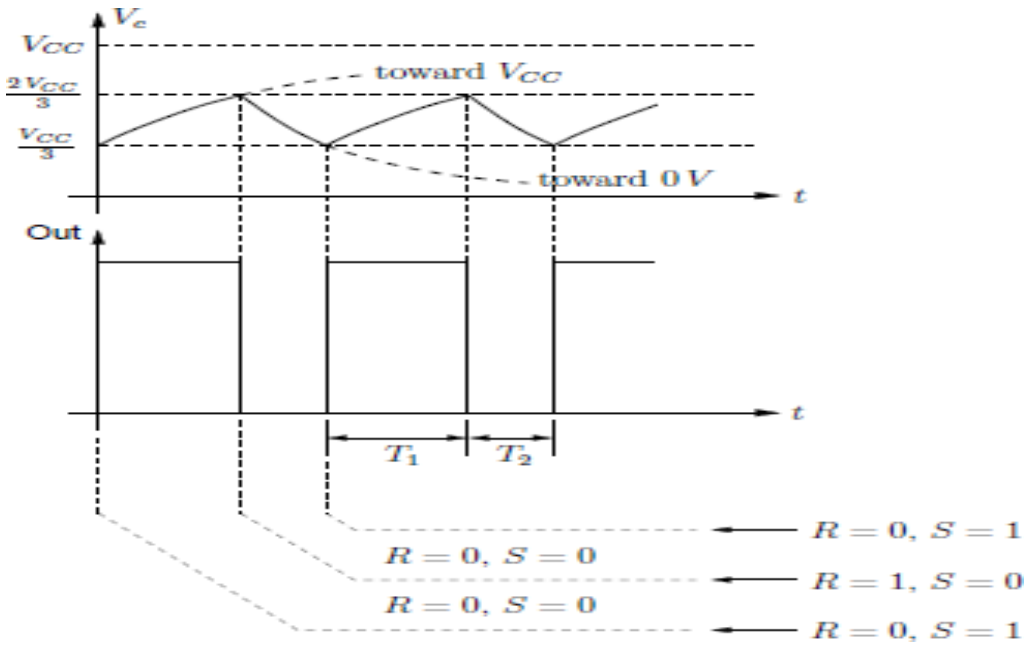
TABULATION:

Symmetrical: **Duty Cycle = 50 %**

tlow **(ms)**		t high **(ms)**		**Frequency (Hz)**		**Output Voltage (V)**	**Capacitor Voltage (V)**
Theoretical	Practical	Theoretical	Practical	Theoretical	Practical		

Unsymmetrical: **Duty Cycle = 25%**

tlow **(ms)**		t high **(ms)**		**Frequency (Hz)**		**Output Voltage (V)**	**Capacitor Voltage (V)**
Theoretical	Practical	Theoretical	Practical	Theoretical	Practical		



MODEL GRAPH:

RESULT:

Thus IC555 timer was operated in astable mode to generate square wave. Theoretical Duty cycle : 25% 50%

Practical Duty cycle : _____

EX. NO: 16 **MONOSTABLE MULTIVIBRATOR USINGIC 555**

DATE: **TIMER**

AIM:

To design, construct and test a monostable multivibrator using IC - 555 timer.

APPARATUS REQUIRED:

S. No.	Name of the Apparatus	Range/Value	Qty
1.	Bread Board	-	1
2.	Resistor	1.8 k Ω	1
3.	IC 555	-	1
4.	CRO	20 MHz.	1
5.	Function Generator	0-3 MHz.	1
6.	Capacitor	0.1µF, 0.01µF	1, 1
7.	RPS	(0-30) V/ 5V	1
8.	Connecting Wires	-	Few

THEORY:

Mono-stable multivibrator has only one stable state and one quasi-stable state. Transition is obtained from the stable to quasi-stable by triggering. The transition time due to external triggering is very short, whereas the time for the circuit to remain quasi-stable state is very large. The circuit returns to stable state from its quasi-stable state by itself, without requiring any external triggering signal. Because, after triggering, the circuit returns from quasi-stable state by itself after a certain time delay, therefore the circuit is also called a one shot multivibrator or univibrator.

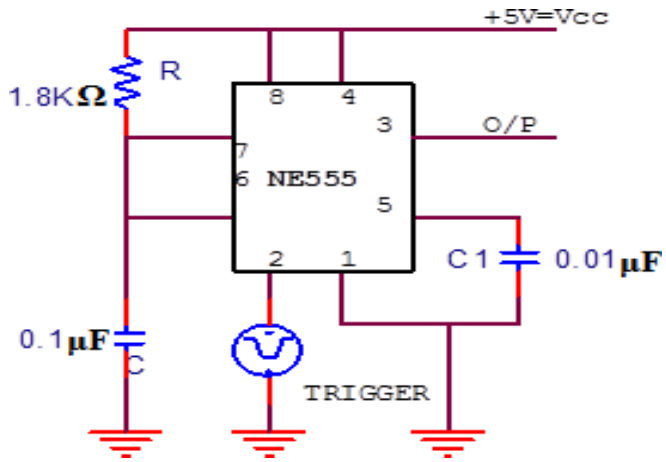
The mono-stable multivibrator is a regenerative device, which is used to generate rectangular output, pulse of predetermined width. The device can make a fast transition in time T after the application of input trigger and as such can be used as a delay circuit. The circuit is also referred to as gating circuit, because it generates rectangular wave form, which can be used to gate other circuits. The Pulse width is T

= 1.1 RC, where R is the resistor and C is the capacitor.

DESIGN:

T=1.1 RC;

Let T = 200 µsec; C= 0.1µF => R =



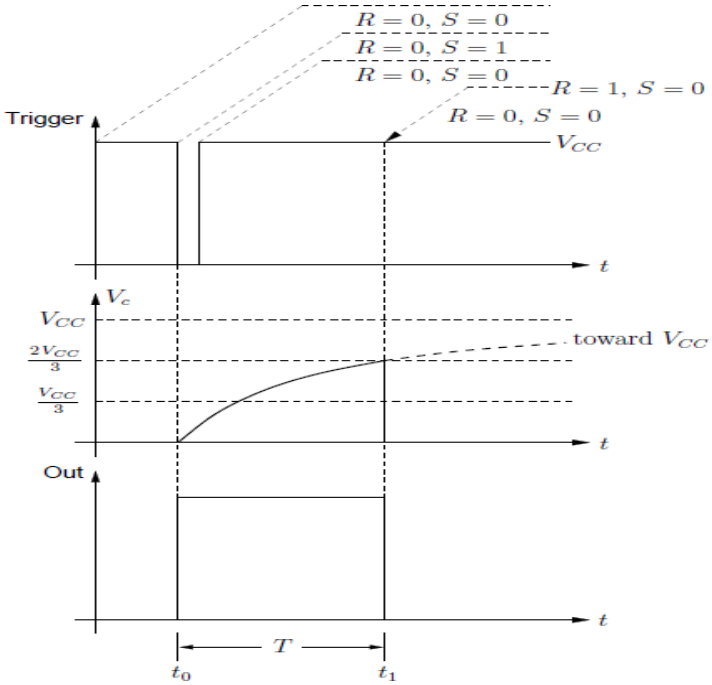
CIRCUIT DIAGRAM:

PROCEDURE:

1. Connect the circuit as shown in circuit diagram.
2. Apply negative trigger to pin 2.
3. Observe and sketch the output wave form at pin 3.
4. Observe the output pulse width for different values of C and tabulate.

TABULATION:

R (k Ω)	C (□F)	Pulse width T (Practical) (ms)	Pulse width T (Theoretical) (ms)



MODEL GRAPH:

RESULT:

Thus IC555 timer was operated in Mono stable mode to generate square waveform.

Theoretical pulse duration =

Practical pulse duration =

EX. NO: 17 PLL (IC 565) CHARACTERISTICS AND ITS USEAS
DATE: FREQUENCY MULTIPLIER

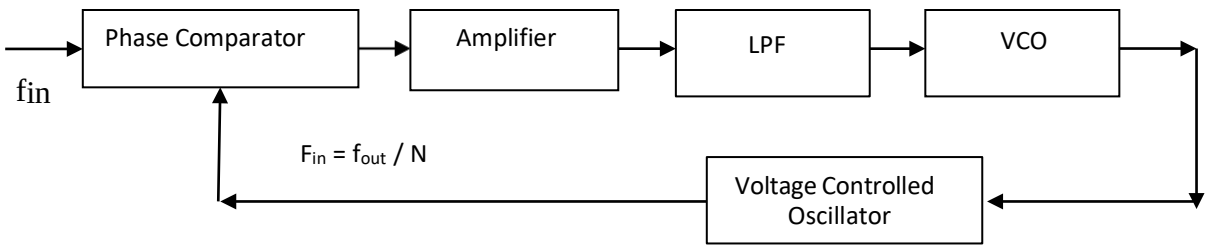
AIM:

a. To study the characteristics of a Phase Locked Loop (PLL)-IC 565.
b. To study the frequency multiplier circuit using PLL-IC 565.

THEORY:

a) PLL- It is basically a feedback control system that controls the phase of a voltage controlled oscillator (VCO). The input signal is applied to one input of a phase detector. The other input is connected to the output of VCO. Normally the frequencies of both signals will be nearly the same. The output of the phase detector is a voltage proportional to the phase difference between the two inputs. This signal is applied to the loop filter. It is the loop filter that determines the dynamic characteristics of the PLL. The filtered signal controls the VCO. The output of the VCO is applied to the phase detector. Normally the loop filter is designed to match the characteristics required by the application of the PLL. If the PLL is to acquire and track a signal the bandwidth of the loop filter will be greater than if it expects a fixed input frequency. The frequency range which the PLL will accept and lock on is called the capture range. Once the PLL is locked and tracking a signal the range of frequencies that the PLL will follow is called the tracking range. Generally the tracking range is larger than the capture range. Figure shows the block diagram of PLL

b) Frequency multiplier using the 565 PLL- The frequency divider is inserted between the VCO and the phase comparator. Since the output of the divider is locked to the input frequency f_{in}, the VCO is actually running at a multiple of the input frequency. The desired amount of multiplication can be obtained by selecting a proper divide by N network, where N is an integer. For example, to obtain the output frequency $f_{OUT} = 5 f_{in}$, a divide by N = 5 network is needed. The 4 bit binary counter (7490) is configuredas a divide by 5 circuits. The transistor Q is used as a driver stage to increase the driving capability of the NE 565. C_3 is used to eliminate possible oscillation. C_2 should be large enough to stabilize the VCO frequency.



DESIGN: a. <u>PLL Circuit</u>

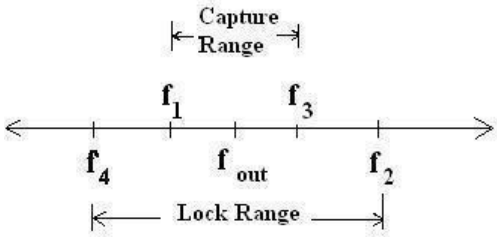
The Circuit components are R_1 = 12 KΩ, C_1 =0.01 µF, C_2 =10 µF & C_3 =0.001 µF.

The design formulae are: V= (+V)-(-V) = 20 Volt.

Free running frequency, f_{out}= 1.2 / [4 R_1 C_1] = 2.5 KHz.

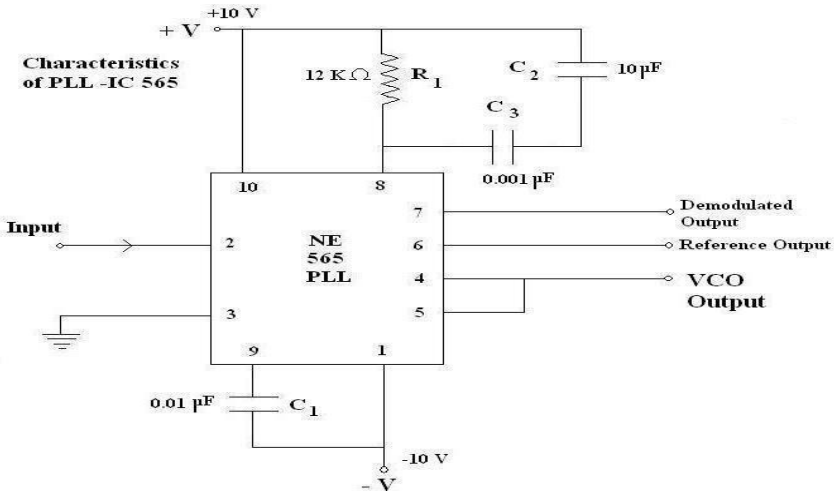
Lock Range, f_L= ± 8 X f_{out} / V = ± 1 KHz.

Capture Range, f_c= ± f_L / [2 π X 3.6 X 10^3 X C_2] = ± 66.49 Hz

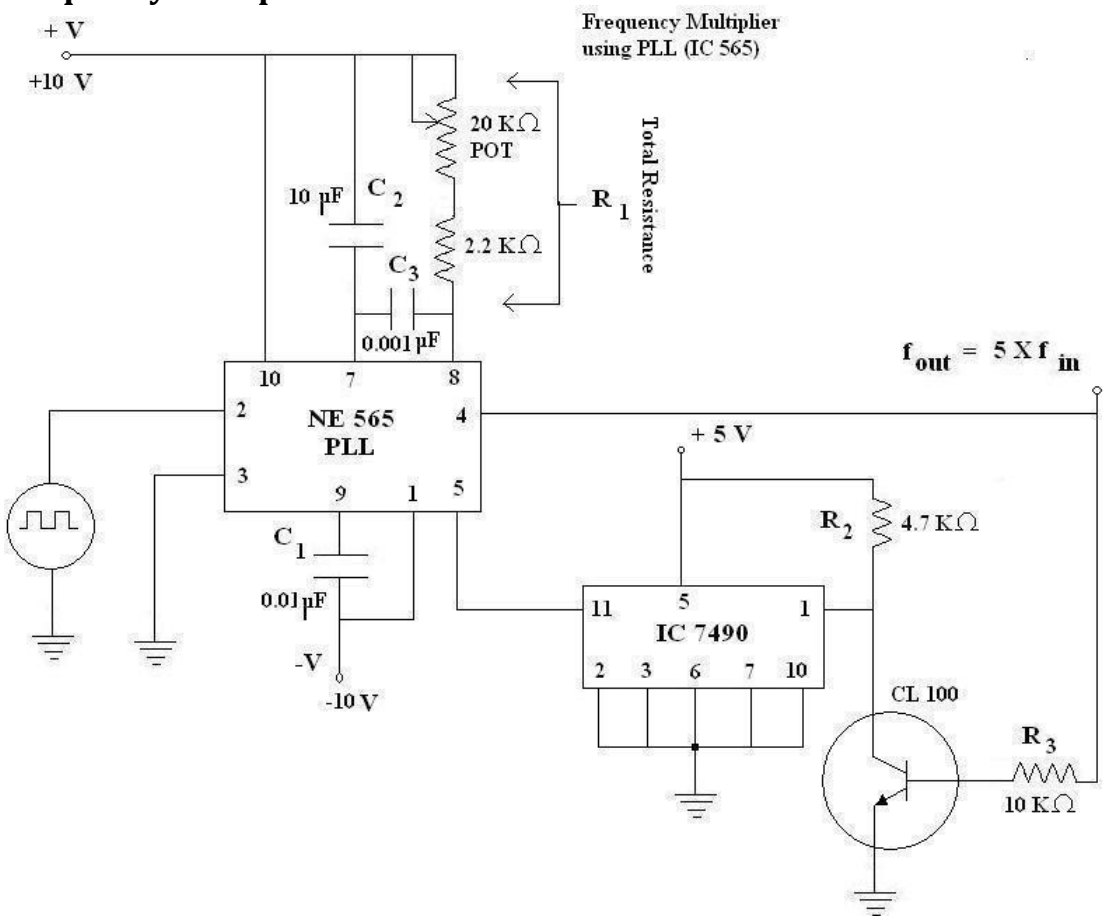


CIRCUIT DIAGRAM:

a. PLL Circuit



b. Frequency Multiplier



RESULT:

PLL is studied and used as frequency multiplier.

EX. NO: 18
DATE:

IC REGULATED DC POWER SUPPLY USING LM 723 and LM 317

AIM:

To design the regulated DC power supply using LM 723 and LM 317.

APPARATUS REQUIRED:

S. No.	Name of the Apparatus	Range/Value	Qty
1.	Bread Board	-	1
2.	Resistor	5 KΩ, 240Ω	1
3.	NE 565 , IC 7490	-	1 each
4.	Voltmeter	(0 – 30)V	1
6.	Capacitor	10µF, 0.1µF, 0.100pF	1 each
7.	RPS	(0-30) V/ 5V	1
8.	Connecting Wires	-	Few

THEORY:

A **voltage regulator** is designed to automatically maintain a constant voltage level. A voltage regulator may be a simple "feed-forward" design or may include negative feedback control loops. It may use an electromechanical mechanism, or electronic components. Depending on the design, it may be used to regulate one or more AC or DC voltages. Electronic voltage regulators are found in devices such as computer power supplies where they stabilize the DC voltages used by the processor and other elements. In automobile alternators and central power station generator plants, voltage regulators control the output of the plant. In an electric power distribution system, voltage regulators may be installed at a substation or along distribution lines so that all customers receive steady voltage independent of how much power is drawn from the line. The circuit diagram shows an IC 723 connected to operate as a positive voltage regulator**.** The output voltage can be set to any value between approximately 7 V (reference voltage) and 37 V by appropriate selection of resistors R1 and R2. A potentiometer may be included between R1 and R2, of course, to make the voltage adjustable. An external transistor may be Darlington connected to Q1 (as shown in earlier post) to handle large load current.

DESIGN:

a) **REGULATOR USING LM317** V_{out} = 6V (given). V_{out} = 1.25[1+R2/R1]Let R_2 = 240Ω, R_1 = R2/(0.8XV_{out}-1) =

b) **REGULATOR USING LM723**

V_{out} = 3V (given).

V_{out}= R_2V_{ref} /R_1 +R_2

V_{ref} = 7V. Choose R1+R2=10K Ω, C1 = 100pF.

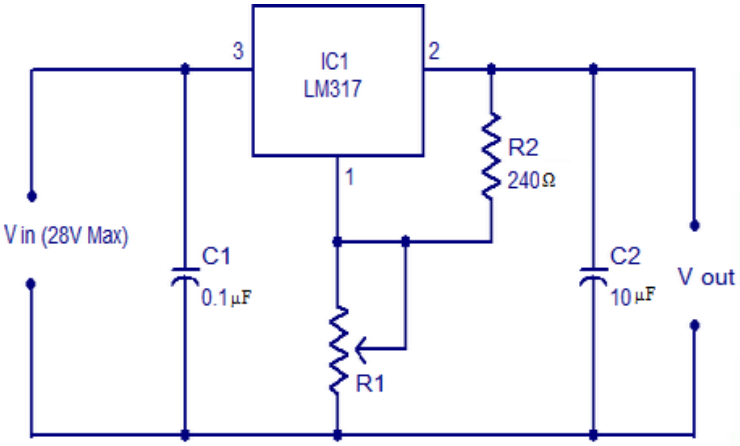
$R_2 = \frac{V_{out}(R_1+R_2)}{V_{ref}} = 3V \times 10 \times 10^3 / 7V = R \times \frac{R_2 R_1}{R_1 \times R_2} =$

PROCEDURE:

1. Connect the circuit as shown in circuit diagram.
2. Apply the unregulated power supply at pin 3.
3. Vary the voltage and observe the regulated output and tabulate the reading.
4. Plot the graph.

CIRCUIT DIAGRAM:

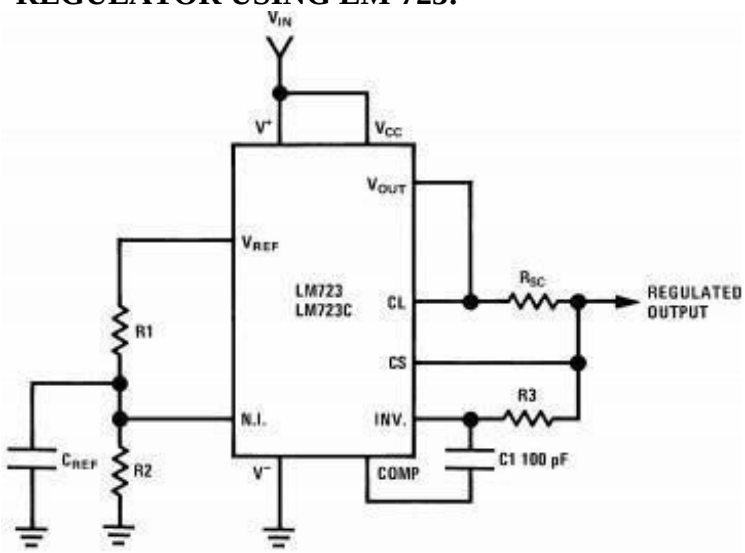
REGULATOR USING LM317



TABULATION:

S.No.	V_{in} in Volts	V_{out} in Volts

REGULATOR USING LM 723:



S.No.	V_{in} in Volts	V_{out} in Volts

RESULT:

Thus the DC Power supply using LM317 and LM 723 is designed and graph is plotted.

MADHA ENGINEERING COLLEGE

(A Christian Minority Institution)

KUNDRATHUR, CHENNAI – 600 069

C Programming and Data Structures Lab

Name	:	
Subject	:	
Roll No.	:	
Semester	:	**Year:**

Ex no:1a **ARRAY IMPLEMENTATION USING STACK ADT**

Aim:

To write a program for stack using array implementation.

Algorithm :

Step1:Define a array which stores stack elements..

Step 2: The operations on the stack are
a)PUSH data into the stack
b)POP data out of stack

Step 3: PUSH DATA INTO STACK
3a.Enter the data to be inserted into stack.
3b.If TOP is NULL
the input data is the first node in stack.
the link of the node is NULL.
TOP points to that node.
3c.If TOP is NOT NULL
the link of TOP points to the new node.
TOP points to that node.

Step 4: POP DATA FROM STACK
4a.If TOP is NULL
the stack is empty
4b.If TOP is NOT NULL
the link of TOP is the current TOP.
the pervious TOP is popped from stack.

Step 5. The stack represented by linked list is traversed to display its content.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
#define SIZE 5
int stack[SIZE],top=-1;
void push();
void pop();
void display();
void main()
{
int choice;
int isempty();
int length();
clrscr();
while(1)
{
        printf("\n 1.Push");
        printf("\n 2. POP");
        printf("\n 3.Display");
        printf("\n 4. Length ");
        printf("\n 5.Quit");
        printf("\n Enter the choice:");
       scanf("\n %d",&choice);
       switch(choice)
        {
                case 1: push();
                         break;
                        case 2: pop();
                                 break;
                        case 3: display();
                                break;
                        case 4: printf("\n No. of elements in the stack is %d",length());
                                break;
                        case 5: exit(0);
                                break;
                        default: printf("\n Invalid choice");
                }
        }
}

void push()
{
        int n;
if(top==SIZE-1)
```

```
printf("\n Stack is full");
else
{
        printf("\nEnter the no.");
        scanf("%d",&n);
        top++;
        stack[top]=n;
}

void pop()
{
         int n;
         if(isempty())
{
        printf("\nStack is empty");
        top=-1;
}
else
{
         n=stack[top];
        printf("\n %d is popped from the stack \n",n);
        --top;
}
}

void display()
{
         int i,temp=top;
if(isempty())
{
        printf("\n Stack Empty");
        return;
         }
printf("\n Elements in the stack:");
for(i=temp;i>=0;i--)
printf("%d \n",stack[i]);
}

int isempty()
{
         return (top==-1);
}
int length()
```

```
{
        return (top+1);
}
```

OUTPUT

1. Push
2. POP
3. Display
4. Length
5. Quit

Enter the choice: 1

Enter the no. 10

1. Push
2. POP
3. Display
4. Length
5. Quit

Enter the choice: 1

Enter the no. 20

1. Push
2. POP
3. Display
4. Length
5. Quit

Enter the choice: 1

Enter the no. 30

1. Push
2. POP
3. Display
4. Length
5. Quit

Enter the choice: 1

Enter the no. 40

1. Push
2. POP
3. Display
4. Length
5. Quit

Enter the choice: 3

Elements in the stack:
40
30
20
10

1. Push
2. POP
3. Display
4. Length
5. Quit

Enter the choice: 2

40 is popped from the stack

1. Push
2. POP
3. Display
4. Length
5. Quit

Enter the choice: 4
Number of elements in the stack is 3

1. Push
2. POP
3. Display
4. Length
5. Quit

Enter the choice: 5

Ex no:1b **ARRAY IMPLEMENTATION USING QUEUE ADT**

Aim:

To write a program for Queue using array implementation.

Algorithm :

Step1:Define a array which stores queue elements..

Step 2: The operations on the queue are
a)INSERT data into the queue
b)DELETE data out of queue

Step 3: INSERT DATA INTO queue
3a.Enter the data to be inserted into queue.
3b.If TOP is NULL
the input data is the first node in queue.
the link of the node is NULL.
TOP points to that node.
3c.If TOP is NOT NULL
the link of TOP points to the new node.
TOP points to that node.

Step 4: DELETE DATA FROM queue
4a.If TOP is NULL
the queue is empty
4b.If TOP is NOT NULL
the link of TOP is the current TOP.
the pervious TOP is popped from queue.

Step 5. The queue represented by linked list is traversed to display its content.

PROGRAM

```
# include<stdio.h>
# define MAX 5

int queue_arr[MAX];
int rear = -1;
int front = -1;

main()
{
int choice;
while(1)
{
printf("1.Insert\n");
printf("2.Delete\n");
printf("3.Display\n");
printf("4.Quit\n");
printf("Enter your choice : ");
scanf("%d",&choice);

switch(choice)
{
case 1 :
insert();
break;
case 2 :
del();
break;
case 3:
display();
break;
case 4:
exit(1);
default:
printf("Wrong choice\n");
}/*End of switch*/
}/*End of while*/
}/*End of main()*/

insert()
{
int added_item;
if (rear==MAX-1)
printf("Queue Overflow\n");
```

```
else
{
if (front==-1) /*If queue is initially empty */
front=0;
printf("Input the element for adding in queue : ");
scanf("%d", &added_item);
rear=rear+1;
queue_arr[rear] = added_item ;
}
}/*End of insert()*/

del()
{
if (front == -1 || front > rear)
{
printf("Queue Underflow\n");
return ;
}
else
{
printf("Element deleted from queue is : %d\n", queue_arr[front]);
front=front+1;
}
}/*End of del() */

display()
{
int i;
if (front == -1)
printf("Queue is empty\n");
else
{
printf("Queue is :\n");
for(i=front;i<= rear;i++)
printf("%d ",queue_arr[i]);
printf("\n");
}
}/*End of display() */
```

OUTPUT

```
1.Insert
2.Delete
3.Display
4.Quit
Enter your choice:1

Input the element for adding in queue :10

1.Insert
2.Delete
3.Display
4.Quit
Enter your choice:1

Input the element for adding in queue :20

1.Insert
2.Delete
3.Display
4.Quit
Enter your choice:1

Input the element for adding in queue :30

1.Insert
2.Delete
3.Display
4.Quit
Enter your choice:1

Input the element for adding in queue :40

1.Insert
2.Delete
3.Display
4.Quit
Enter your choice:3

Queue is :
40
30
20
```

10

1.Insert
2.Delete
3.Display
4.Quit
Enter your choice:2

Element deleted from queue is :10

1.Insert
2.Delete
3.Display
4.Quit
Enter your choice:3

Queue is :
40
30
20

1.Insert
2.Delete
3.Display
4.Quit
Enter your choice:4

EX : 2 **ARRAY IMPLEMENTATION OF LIST ADT**

Aim:

To write a program for stack using linked list implementation.

Algorithm :

Step1:Define a C-struct for each node in the stack. Each node in the stack contains data and link to the next node. TOP pointer points to last node inserted in the stack.

Step 2: The operations on the stack are
a)PUSH data into the stack
b)POP data out of stack

Step 3: PUSH DATA INTO STACK
3a.Enter the data to be inserted into stack.
3b.If TOP is NULL
the input data is the first node in stack.
the link of the node is NULL.
TOP points to that node.
3c.If TOP is NOT NULL
the link of TOP points to the new node.
TOP points to that node.

Step 4: POP DATA FROM STACK
4a.If TOP is NULL
the stack is empty
4b.If TOP is NOT NULL
the link of TOP is the current TOP.
the pervious TOP is popped from stack.

Step 5. The stack represented by linked list is traversed to display its content.

PROGRAM

```
# include<stdio.h>
# include<conio.h>

struct node
{
int info;
struct node *link;
} *top=NULL;

main()
{
int choice;
while(1)
{ printf("1.Push\n");
printf("2.Pop\n");
printf("3.Display\n");
printf("4.Quit\n");
printf("Enter your choice : ") ;
scanf("%d", &choice);

switch(choice)
{
case 1:
push();
break;
case 2:
pop();
break;
case 3:
display();
break;
case 4:
exit(1);
default :
printf("Wrong choice\n");
}/*End of switch */
}/*End of while */
}/*End of main() */

push()
{
struct node *tmp;
int pushed_item;
```

```
tmp = (struct node *)malloc(sizeof(struct node));
printf("Input the new value to be pushed on the stack : ");
scanf("%d",&pushed_item);
tmp->info=pushed_item;
tmp->link=top;
top=tmp;
}/*End of push()*/

pop()
{
struct node *tmp;
if(top == NULL)
printf("Stack is empty\n");
else
{ tmp=top;
printf("Popped item is %d\n",tmp->info);
top=top->link;
free(tmp);
}

}/*End of pop()*/

display()
{ struct node *ptr;
ptr=top;
if(top==NULL)
printf("Stack is empty\n");
else
{
printf("Stack elements :\n");
while(ptr!= NULL)
{
printf("%d\n",ptr->info);
ptr = ptr->link;
}/*End of while */
}/*End of else*/
}/*End of display()*/
```

OUTPUT

1.Push
2. POP
3.Display
5.Quit

Enter the choice: 1

Input the new value to be pushed on the stack :. 10

1.Push
2. POP
3.Display
5.Quit

Enter the choice: 1

Input the new value to be pushed on the stack : 20

1.Push
2. POP
3.Display
5.Quit

Enter the choice: 1

Input the new value to be pushed on the stack : 30

1.Push
2. POP
3.Display
5.Quit

Enter the choice: 1

Input the new value to be pushed on the stack : 40

1.Push
2. POP
3.Display
5.Quit

Enter the choice: 3

Elements in the stack:
40
30
20
10

1.Push
2. POP
3.Display
5.Quit

Enter the choice: 2

40 is popped from the stack

1.Push
2. POP
3.Display
5.Quit

Enter the choice: 5

Ex:3a

LINKED LIST IMPLEMENTATION USING LIST

AIM:

To implement a linked list and do all operations on it.

ALGORITHM:

Step 1 : Start the process.

Step 2: Initialize and declare variables.

Step 3: Enter the choice. INSERT / DELETE.

Step 4: If choice is INSERT then

a. Enter the element to be inserted.

b. Get a new node and set DATA[NEWNODE] = ITEM.

c. Find the node after which the new node is to be inserted.

d. Adjust the link fields.

e. Print the linked list after insertion.

Step 5: If choice is DELETE then

a. Enter the element to be deleted.

b. Find the node containing the element (LOC) and its preceding node (PAR).

c. Set ITEM = DATA[LOC] and delete the node LOC.

d. Adjust the link fields so that PAR points to the next element. ie
LINK[PAR] = LINK [LOC].

e. Print the linked list after deletion.

Step 6: Stop the process.

PROGRAM

```
#include<stdio.h>
#include<stlib.h>
#include<conio.h>

struct node;
typedef struct node *ptr;
typedef ptr list;
typedef ptr position;
typedef int data;

struct node
{
        data element;
        struct node *next;
}

//function prototypes
void makeempty(void);           //to make empty list
int isempty(void);              //to check list is empty or not
void create(void);              //to create initial set of elements
position findprevious(data);    //to find position of previous element
void delet(data);               //to delete given element
void display(void);             //to display all the elements
void insert(data, int);         //to insert a new element
position getprevposition(int);  //to find position of previous element
data getelement(int);           //to find the element at given position
int getposition(data);          //to find position of given element

//global variable declarations
position first;
position last;
position L;
int length;

//to make empty list
void makeempty(void)
{
        position tmp;
        tmp = malloc(sizeof(list));
        tmp->next = NULL;
        L = tmp;
        first = last = NULL;
}
```

```
//to check list is empty or not
int isempty(void)
{
	if (L->next = NULL)
		return 1;
	else
		return 0;
}

//to create initial set of elements
void create(void)
{
	data e;
	int n, i;
	position tmp;
	makeempty();
	printf("Enter number of element : \		");
	scanf("%d", &n);
	for (i=0; i<n; i++)
	{
		printf("Enter an element : ");
		scanf("%d", &e);
		tmp = malloc(sizeof(list));
		tmp->element = e;
		tmp->next = NULL;
		if (L->next == NULL)
		{
			L->next = tmp;
			first = last = tmp;
		}
		else
		{
			last->next = tmp;
			last = tmp;
		}
	}
}

//to display all the elements
void display()
{
	position t;
	for(t=first; t!=NULL; t=t->next)
		printf("%d --> ", t->element);
	getch();
}
```

```
//to find position of previous element
position getprevposition(int index)
{
	position tmp;
	int count = 1;
	if (index>length)
	{
		printf("Invalid Position");
		return NULL;
	}
	else
	{
		for (tmp=first; count<index-1; tmp=tmp->next)
			count++;
		return tmp;
	}
}

//to insert a new element
void insert(data x, int p)
{
	position pos, tmp;
	tmp = malloc(sizeof(list));
	tmp->element=x;
	if (p==1)			//first position
	{
		tmp->next = first;
		L->next = tmp;
		first = tmp;
		length++;
	}
	else if (p == length)		//last position
	{
		last->next = tmp;
		last = tmp;
		tmp->next = NULL;
	}
	else				//arbitrary position
	{
		pos = getpreviousposition(p);
		if (pos == NULL)
		{
			printf("Invalid position");
			getch();
		}
		else
```

```
		{
			tmp->next = pos->next;
			pos->next = tmp;
			length++;
		}
	}
}

//to find position of previous element
position findprevious(data x)
{
	position p;
	p = L;
	while (p->next->element!=x && p->next!=NULL)
		p = p->next;
	return p;
}

//to delete given element
void delet(data x)
{
	position p, tmp;
	if (isempty())
	{
		printf("List is empty");
		getch();
	}
	else
	{
		p = findprevious(x);
		if (p->next = NULL)
		{
			printf("Element not found");
			getch();
		}
		else
		{
			if (p->next == last)
			{
				free (p->next);
				p->next = NULL;
				last = p;
				length--;
				return;
			}
			if (p == L)
```

```
                        {
                                first = first->next;
                        }
                        tmp = p->next;
                        p->next = tmp->next;
                        free(tmp);
                        length--;
                }
        }
}

int menu()
{
        int ch;
        printf("1. Create\n2. Display\n3.Insert\n4.Get Element\n5.Get Position\n6. Delete\n7.
Exit\n\n Enter your choice : ");
        scanf("%d", &choice);
        return choice;
}

//to find the element at given position
data getelement(int pos)
{
        position p;
        int i;
        p = L;
        if (pos > length)
                return NULL;
        else
        {
                for(i=0; i<pos; i++)
                        p = p->next;
                return p->element;
        }
}

//to find position of given element
int getposition(data e)
{
        position p;
        int i=0;
        for (p=first; p!=NULL; p=p->next)
        {
                if (p->element == e)
                        return i+1;
                else
```

```
                        i++;
        }
        return NULL;
}

void main()
{
        int ch;
        data n, p;
        while(1)
        {
                clrscr();
                ch = menu();
                switch (ch)
                {
                        case 1:
                                create();
                                break;
                        case 2:
                                display();
                                break;
                        case 3:
                                printf("Enter an element : ");
                                scanf("%d", &n);
                                printf("Enter Position : ");
                                scanf("%d", &p);
                                insert (n, p);
                                break;
                        case 4:
                                printf("Enter an element : ");
                                scanf("%d", &n);
                                delet (n);
                                break;
                        case 5:
                                printf("Enter position : ");
                                scanf("%d", &p);
                                if (p<1 || p>length)
                                        printf("Invalid position");
                                else
                        printf("Element at position %d is %d", p, getelement(p));
                                getch();
                                break;
                        case 6:
                                printf("Enter an element : ");
                                scanf("%d", &n);
                                if (getposition(n) == NULL)
```

```
                        printf("Element doesn't Exist");
                else
                        printf("%d exists at position %d", n, getposition(n));
                getch();
                break;
        default:
                printf("Invalid Choice");
                getch();
        }
    }
}
```

OUTPUT

1. Create
1. Display
2. Insert
3. Delete
4. Get element
5. Get position
6. Exit

Enter your Choice: 1

Enter number of element: 3
Enter an element: 10
Enter an element: 20
Enter an element: 30

Enter your Choice: 3

Enter element: 25
Enter Position: 3

Enter your Choice: 2

10 --> 20 --> 25 --> 30

Enter your Choice: 6

Enter an element:20
20 exists at position 2

Enter your Choice: 4

Enter an element 30

Enter your Choice: 2

10 --> 20 --> 25

Enter your Choice: 6

Ex:3b

LINKED LIST IMPLEMENTATION USING STACK

Aim:

To write a program for stack using linked list implementation.

Algorithm :

Step1:Define a C-struct for each node in the stack. Each node in the stack contains data and link to the next node. TOP pointer points to last node inserted in the stack.

Step 2: The operations on the stack are
a)PUSH data into the stack
b)POP data out of stack

Step 3: PUSH DATA INTO STACK
3a.Enter the data to be inserted into stack.
3b.If TOP is NULL
the input data is the first node in stack.
the link of the node is NULL.
TOP points to that node.
3c.If TOP is NOT NULL
the link of TOP points to the new node.
TOP points to that node.

Step 4: POP DATA FROM STACK
4a.If TOP is NULL
the stack is empty
4b.If TOP is NOT NULL
the link of TOP is the current TOP.
the pervious TOP is popped from stack.

Step 5. The stack represented by linked list is traversed to display its content.

PROGRAM

```
# include<stdio.h>
# include<conio.h>

struct node
{
int info;
struct node *link;
} *top=NULL;

main()
{
int choice;
while(1)
{ printf("1.Push\n");
printf("2.Pop\n");
printf("3.Display\n");
printf("4.Quit\n");
printf("Enter your choice : ") ;
scanf("%d", &choice);

switch(choice)
{
case 1:
push();
break;
case 2:
pop();
break;
case 3:
display();
break;
case 4:
exit(1);
default
:
printf("Wrong choice\n");
}/*End of switch */
}/*End of while */
}/*End of main() */

push()
{
struct node *tmp;
int pushed_item;
tmp = (struct node *)malloc(sizeof(struct node));
printf("Input the new value to be pushed on the stack : ");
```

```
scanf("%d",&pushed_item);
tmp->info=pushed_item;
tmp->link=top;
top=tmp;
}/*End of push()*/

pop()
{
struct node *tmp;
if(top == NULL)
printf("Stack is empty\n");
else
{ tmp=top;
printf("Popped item is %d\n",tmp->info);
top=top->link;
free(tmp);
}

}/*End of pop()*/

display()
{ struct node *ptr;
ptr=top;
if(top==NULL)
printf("Stack is empty\n");
else
{
printf("Stack elements :\n");
while(ptr!= NULL)
{
printf("%d\n",ptr->info);
ptr = ptr->link;
}/*End of while */
}/*End of else*/
}/*End of display()*/
```

OUTPUT

1.Push
2. POP
3.Display
5.Quit

Enter the choice: 1

Input the new value to be pushed on the stack :. 10

1.Push
2. POP
3.Display
5.Quit

Enter the choice: 1

Input the new value to be pushed on the stack : 20

1.Push
2. POP
3.Display
5.Quit

Enter the choice: 1

Input the new value to be pushed on the stack : 30

1.Push
2. POP
3.Display
5.Quit

Enter the choice: 1

Input the new value to be pushed on the stack : 40

1.Push
2. POP
3.Display
5.Quit

Enter the choice: 3

Elements in the stack:
40
30
20
10

1.Push
2. POP
3.Display
5.Quit

Enter the choice: 2

40 is popped from the stack

1.Push
2. POP
3.Display
5.Quit

Enter the choice: 5

Ex:3c

LINKED LIST IMPLEMENTATION USING QUEUE

Aim:

To write a program for Queue using Linked implementation.

Algorithm :

Step1: Define a C-struct for each node in the queue. Each node in the queue contains data and link to the next node. Front and rear pointer points to first and last node inserted in the queue.

Step 2: The operations on the queue are
a)INSERT data into the queue
b)DELETE data out of queue

Step 3: INSERT DATA INTO queue
3a.Enter the data to be inserted into queue.
3b.If TOP is NULL
the input data is the first node in queue.
the link of the node is NULL.
TOP points to that node.
3c.If TOP is NOT NULL
the link of TOP points to the new node.
TOP points to that node.

Step 4: DELETE DATA FROM queue
4a.If TOP is NULL
the queue is empty
4b.If TOP is NOT NULL
the link of TOP is the current TOP.
the pervious TOP is popped from queue.

Step 5. The queue represented by linked list is traversed to display its content.

PROGRAM

```
#include<stdio.h>
#include<malloc.h>
#define MAXSIZE 10

void insertion();
void  deletion();
void display();

struct node
{
int info;
struct node *link;
}*new,*temp,*p,*front=NULL,*rear=NULL;
typedef struct node N;


main()
{
int ch;
do
{
printf("\n\t\t\tLinked queue");
printf("\n 1.Insertion");
printf("\n 2.Deletion");
printf("\n 3.Display");
printf("\n 4.Exit");
printf("\n Enter your choice : ");
scanf("%d",&ch);
switch(ch)
{
case 1:
insertion();
break;
case 2:
deletion();
break;
case 3:
display();
break;
default:
break;
}
}
while(ch<=3); }
void insertion()
```

```
{
int item;
new=(N*)malloc(sizeof(N));
printf("\nEnter the item : ");
scanf("%d",&item);
new->info=item;
new->link=NULL;
if(front==NULL)
front=new;
else
rear->link=new;
rear=new;
}

void deletion()
{
if(front==NULL)
printf("\nQueue is empty");
else
{
p=front;
printf("\nDeleted element is : %d",p->info);
front=front->link;
free(p);
}
}

void display()
{
if(front==NULL)
printf("\nQueue is empty");
else
{
printf("\nThe elements are : ");
temp=front;
while(temp!=NULL)
{
printf("%d",temp->info);
temp=temp->link;
}
}
}
```

OUTPUT

```
1.Insertion
2.Deletion
3.Display
4.Exit
Enter your choice:1
Enter the item :10

1.Insertion
2.Deletion
3.Display
4.Exit
Enter your choice:1
Enter the item :20

1.Insertion
2.Deletion
3.Display
4.Exit
Enter your choice:1
Enter the item :30

1.Insertion
2.Deletion
3.Display
4.Exit
Enter your choice:1
Enter the item :40

1.Insertion
2.Deletion
3.Display
4.Exit
Enter your choice:3

The elements are :
40
30
20
10

1.Insertion
2.Deletion
3.Display
```

```
4.Exit
Enter your choice:2

Deleted element is : 10

1.Insertion
2.Deletion
3.Display
4.Exit
Enter your choice:3

The elements are :
40
30
20

1.Insertion
2.Deletion
3.Display
4.Exit
Enter your choice:4
```

Ex:4a

POLYNOMIAL MANIPULATION

Aim

To implement polynomial manipulation using doubly linked lists.

Algorithm

POLYADD(POLY1: POLY2:POLY)

HEAD:POLY

Step 1: Assign HEAD+=NULL

Step2: While (POLY !=null)

Step3: HEAD=INSERTNODE(HEAD,COPYNODE,(POLY1,1))

Step4: POLY1=POLY1_NEXT

Step5: [End of Step2 while structure]

Step6: While(POLY2 1=NULL)

Step7: HEAD =INSERTNODE(HEAD,COPYNODE(POLY2,1))

Step8: POLY2=POLY2_NEXT

Step9: [End of Step 6 while Structure]

Step10: Return HEAD

END POLYADD()

Algorithm for polynomial subtraction

POLYSUB(POLY1:POLY, POLY2:POLY)

HEAD:POLY

Step1: Assign HEAD=NULL

Step2: While(POLY1!=NULL)

Step3: HEAD=INSERTNODE(HEAD,COPYNODE(POLY1,1))

Step4: POLY1=POLY1_ NEXT

Step5: [End of Step2 while Structure]

Step6:While(POLY2!=NULL)

Step7: HEAD=INSERTNODE(HEAD,COPYNODE(POLY2,1))

Step8: POLY2=POLY2_NEXT

Step9: [End of Step 6 While Structure]

Step10: Return HEAD

END POLYSUB()

PROGRAM

```
#include<malloc.h>
#include<conio.h>
struct link
{
int coeff;
int pow;
struct link *next;
};
struct link *poly1=NULL,*poly2=NULL,*poly=NULL;
void create(struct link *node)
{
char ch;
do
{
printf("\nEnter the coefficient :");
scanf("%d",&node>
coeff);
printf("\nEnter the power :");
scanf("%d",&node>
pow);
node>
next=(struct link *)malloc(sizeof(struct link));
node=node>
next;
node>
next=NULL;
printf("\nContinue??? (Y/N) :");
ch=getch();
}while(ch=='y' || ch=='Y');
```

```
}
void display(struct link *node)
{
while(node>
next!=NULL)
{
printf("%dx^%d",node>
coeff,node>
pow);
node=node>
next;
if(node>
next!=NULL)
printf(" + ");
}
}
void polyadd(struct link *poly1,struct link *poly2,struct link *poly)
{
while(poly1>
next && poly2>
next)
{
if(poly>
pow > poly2>
pow)
{
poly>
pow=poly1>
pow;
poly>
coeff=poly1>
```

```
coeff;
poly1=poly1>
next;
}
else if(poly1>
pow < poly2>
pow)
{
poly>
pow=poly2>
pow;
poly>
coeff=poly2>
coeff;
poly2=poly2>
next;
}
else
{
poly>
pow=poly1>
pow;
poly>
coeff=poly1>
coeff+poly2>
coeff;
poly1=poly1>
next;
poly2=poly2>
next;
}
```

```
poly>
next=(struct link *)malloc(sizeof(struct link));
poly=poly>
next;
poly>
next=NULL;
}
while(poly1>
next||poly2>
next)
{
if(poly1>
next)
{
poly>
pow=poly1>
pow;
poly>
coeff=poly1>
coeff;
poly1=poly1>
next;
}
if(poly2>
next)
{
poly>
pow=poly2>
pow;
poly>
coeff=poly2>
```

```
  coeff;
  poly2=poly2>
  next;
  }
  poly>
  next=(struct link *)malloc(sizeof(struct link));
  poly=poly>
  next;
  poly>
  next=NULL;
  }
  }
  void main()
  {
  poly1=(struct link *)malloc(sizeof(struct link));
  poly2=(struct link *)malloc(sizeof(struct link));
  poly=(struct link *)malloc(sizeof(struct link));
  clrscr();
  printf("\nEnter the first polynomial::");
  create(poly1);
  printf("\nFirst polynomial is :: \n");
  display(poly1);
  printf("\nEnter the second polynomial::");
  create(poly2);
  printf("\nSecond polynomial is :: \n");
  display(poly2);
  polyadd(poly1,poly2,poly);
  printf("\nAddition of the two polynomials::");
  display(poly);
  getch();
}
```

OUTPUT

```
Enter the first polynomial:
Enter the coefficient :5
Enter the power :3
Continue??? (Y/N) :Y
Enter the coefficient :3
Enter the power :2
Continue??? (Y/N) :
First polynomial is ::
5x^3 + 3x^2
Enter the second polynomial::
Enter the coefficient :7
Enter the power :3
Continue??? (Y/N) :
Second polynomial is ::
7x^3
Addition of the two polynomials::12x^3 + 3x^2
```

Ex: 4b **INFIX TO POSTFIX CONVERSION**

Aim

To implement infix to postfix conversion using stack.

Algorithm

Step 1. Push left parenthesis onto STACK and add right parenthesis at the end of Q.

Step 2. Scan Q from left to right and repeat step 3 to 6 for each element of Q until the STACK is empty.

Step 3. If an operand is encountered add it to P.

Step 4. If a left parenthesis is encountered push it onto the STACK.

Step 5. If an operator is encountered, the Repeatedly pop from STACK and add to P each operator which has same precedence as or higher precedence than the operator encountered.Push the encountered operator onto the STACK.

Step 6. If a right parenthesis is encountered, then Repeatedly pop from the STACK and add to P each operator until a left parenthesis is encountered.Remove the left parenthesis; do not add it to P.

Step 7. Exit

PROGRAM

```
include<stdio.h>
char stack[20];
int top = -1;
void push(char x)
{
    stack[++top] = x;
}

char pop()
{
    if(top == -1)
        return -1;

    else
        return stack[top--];
}

int priority(char x)
{
    if(x == '(')
        return 0;
    if(x == '+' || x == '-')
        return 1;
    if(x == '*' || x == '/')
        return 2;
}

main()
{
    char exp[20];
    char *e, x;
    printf("Enter the expression :: ");
    scanf("%s",exp);
```

```
    e = exp;
    while(*e != '\0')
    {
        if(isalnum(*e))
            printf("%c",*e);
        else if(*e == '(')
            push(*e);
        else if(*e == ')')
        {
            while((x = pop()) != '(')
                printf("%c", x);


        }
        else
        {
            while(priority(stack[top]) >= priority(*e))
                printf("%c",pop());
            push(*e);
        }
        e++;
    }
    while(top != -1)
    {
        printf("%c",pop());
    }
}
```

OUTPUT

Enter the expression :: a+b*c

abc*+

EX.N0. 5 **BINARY TREE**

Aim:

To write a c program for Implementation of binary tree.

Algorithm:

1. Declare pointer right and left
2. Create a structure for a tree contains left pointer and right pointer.
3. Insert an element is by checking the top node and the leaf node and the operation will be performed.
4. Deleting an element contains searching the tree and deleting the item.
5. Display the Tree elements.

PROGRAM

```
#include<stdlib.h>
#include<stdio.h>

struct bin_tree {
int data;
struct bin_tree * right, * left;
};
typedef struct bin_tree node;

void insert(node ** tree, int val)
{
    node *temp = NULL;
    if(!(*tree))
    {
        temp = (node *)malloc(sizeof(node));
        temp->left = temp->right = NULL;
        temp->data = val;
        *tree = temp;
        return;
```

```
    }

    if(val < (*tree)->data)
    {
        insert(&(*tree)->left, val);
    }
    else if(val > (*tree)->data)
    {
        insert(&(*tree)->right, val);
    }

}

void print_preorder(node * tree)
{
    if (tree)
    {
        printf("%d\n",tree->data);
        print_preorder(tree->left);
        print_preorder(tree->right);
    }

}

void print_inorder(node * tree)
{
    if (tree)
    {
        print_inorder(tree->left);
        printf("%d\n",tree->data);
        print_inorder(tree->right);
```

```
    }
}

void print_postorder(node * tree)
{
    if (tree)
    {
        print_postorder(tree->left);
        print_postorder(tree->right);
        printf("%d\n",tree->data);
    }
}

void deltree(node * tree)
{
    if (tree)
    {
        deltree(tree->left);
        deltree(tree->right);
        free(tree);
    }
}

node* search(node ** tree, int val)
{
    if(!(*tree))
    {
        return NULL;
    }

    if(val < (*tree)->data)
```

```
    {
        search(&((*tree)->left), val);
    }
    else if(val > (*tree)->data)
    {
        search(&((*tree)->right), val);
    }
    else if(val == (*tree)->data)
    {
        return *tree;
    }
}

void main()
{
    node *root;
    node *tmp;
    //int i;

    root = NULL;
    /* Inserting nodes into tree */
    insert(&root, 9);
    insert(&root, 4);
    insert(&root, 15);
    insert(&root, 6);
    insert(&root, 12);
    insert(&root, 17);
    insert(&root, 2);

    /* Printing nodes of tree */
    printf("Pre Order Display\n");
```

```
    print_preorder(root);

    printf("In Order Display\n");
    print_inorder(root);

    printf("Post Order Display\n");
    print_postorder(root);

    /* Search node into tree */
    tmp = search(&root, 4);
    if (tmp)
    {
        printf("Searched node=%d\n", tmp->data);
    }
    else
    {
        printf("Data Not found in tree.\n");
    }

    /* Deleting all nodes of tree */
    deltree(root);
}
```

OUTPUT

Pre Order Display

9

4

2

6

15

12

17

In Order Display

2

4

6

9

12

15

17

Post Order Display

2

6

4

12

17

15

9

Searched node=4

EX: 6 **BINARY SEARCH TREE**

Aim:

To write a c program for binary search tree.

Algorithm:

1. Declare function add(),search(),findmin().find(),findmax(),Display().
2. Create a structure for a tree contains left pointer and right pointer.
3. Insert an element is by checking the top node and the leaf node and the operation will be performed.
4. Deleting an element contains searching the tree and deleting the item.
5. display the Tree elements.

PROGRAM

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>

struct searchtree
{
	int element;
	struct searchtree *left,*right;
}*root;
typedef struct searchtree *node;
typedef int ElementType;

node insert(ElementType, node);
node delete(ElementType, node);
void makeempty();
node findmin(node);
node findmax(node);
node find(ElementType, node);
void display(node, int);

void main()
{
	int ch;
	ElementType a;
	node temp;
	makeempty();
	while(1)
	{
		printf("\n1. Insert\n2. Delete\n3. Find min\n4. Find max\n5. Find\n6.
Display\n7. Exit\nEnter Your Choice : ");
		scanf("%d",&ch);
		switch(ch)
		{
```

```
case 1:
        printf("Enter an element : ");
        scanf("%d", &a);
        root = insert(a, root);
        break;
case 2:
        printf("\nEnter the element to delete : ");
        scanf("%d",&a);
        root = delet(a, root);
        break;
case 3:
        printf("\nEnter the element to search : ");
```

```
        scanf("%d",&a);
        temp = find(a, root);
        if (temp != NULL)
                printf("Element found");
        else
                printf("Element not found");
        break;
case 4:
        temp = findmin(root);
        if(temp==NULL)
                printf("\nEmpty tree");
        else
                printf("\nMinimum element : %d", temp->element);
        break;
case 5:
        temp = findmax(root);
        if(temp==NULL)
                printf("\nEmpty tree");
        else
                printf("\nMaximum element : %d", temp->element);
        break;
case 6:
        if(root==NULL)
                printf("\nEmpty tree");
        else
                display(root, 1);
        break;
 case 7:
        exit(0);
default:
        printf("Invalid Choice");
```

```
                }
        }
}

node insert(ElementType x,node t)
{
        if(t==NULL)
        {
                t = (node)malloc(sizeof(node));
                t->element = x;
                t->left = t->right = NULL;
        }
        else
        {
                if(x < t->element)
                        t->left = insert(x, t->left);
                else if(x > t->element)
                        t->right = insert(x, t->right);
        }
        return t;
}

node delet(ElementType x,node t)
{
        node temp;
        if(t == NULL)
                printf("\nElement not found");
        else
        {
                if(x < t->element)
                        t->left = delet(x, t->left);
```

```
		else if(x > t->element)
			t->right = delet(x, t->right);
		else
		{
			if(t->left && t->right)
			{
				temp = findmin(t->right);
				t->element = temp->element;
				t->right = delet(t->element,t->right);
			}
			else if(t->left == NULL)
				t=t->right;
			else
				t=t->left;
		}
	}
	return t;
}
void makeempty()
{
	root = NULL;
}



node findmin(node temp)
{
	if(temp == NULL || temp->left == NULL)
		return temp;
	return findmin(temp->left);
}
```

```
node findmax(node temp)
{
        if(temp==NULL || temp->right==NULL)
                return temp;
        return findmin(temp->right);
}

node find(ElementType x, node t)
{
        if(t==NULL) return NULL;
        if(x<t->element) return find(x,t->left);
        if(x>t->element) return find(x,t->right);
        return t;
}
 void display(node t,int level)
{
        int i;
        if(t)
        {
                display(t->right, level+1);
                printf("\n");
                for(i=0;i<level;i++)
                        printf(" ");
                printf("%d", t->element);
                display(t->left, level+1);
        }
}
```

OUTPUT

```
1. Insert
2. Delete
3. Find
4. Find Min
5. Find Max
6. Display
7. Exit
Enter your Choice : 1
Enter an element : 10

1. Insert
2. Delete
3. Find
4. Find Min
5. Find Max
6. Display
7. Exit
Enter your Choice : 1
Enter an element : 20


1. Insert
2. Delete
3. Find
4. Find Min
5. Find Max
6. Display
7. Exit
Enter your Choice : 1
```

```
Enter an element : 5


1. Insert
2. Delete
3. Find
4. Find Min
5. Find Max
6. Display
7. Exit
Enter your Choice : 4
The smallest Number is 5

1. Insert
2. Delete
3. Find
4. Find Min
5. Find Max
6. Display
7. Exit
Enter your Choice : 3
Enter an element : 100
Element not Found

1. Insert
2. Delete
3. Find
4. Find Min
5. Find Max
6. Display
7. Exit
```

```
Enter your Choice : 2
Enter an element : 20

1. Insert
2. Delete
3. Find
4. Find Min
5. Find Max
6. Display
7. Exit
Enter your Choice : 6
  5
10

1. Insert
2. Delete
3. Find
4. Find Min
5. Find Max
6. Display
7. Exit
Enter your Choice : 7
```

Ex.no.7

IMPLEMENTATION OF AVL TREES

Date:

Aim:

To write a C program to perform implementation of AVL tree.

ALGORITHM

The following two cases are possible-

Case-01:

- After the insertion, the balance factor of each node is either 0 or 1 or -1.
- In this case, the tree is considered to be balanced.
- Conclude the operation.
- Insert the next element if any.
-

Case-02:

- After the insertion, the balance factor of at least one node is not 0 or 1 or -1.
- In this case, the tree is considered to be imbalanced.
- Perform the suitable rotation to balance the tree.
- After the tree is balanced, insert the next element if any.

PROGRAM

```
#include<conio.h>
#include<stdio.h>
typedef enum {FALSE,TRUE}bool;
struct node
{
int info;
int balance;
struct node *lchild;
struct node *rchild;
}*root;
struct node *search(struct node *ptr,int info)
{
if(ptr!=NULL)
if(info<ptr>
info)
ptr=search(ptr>
lchild,info);
else if(info>ptr>
info)
ptr=search(ptr>
rchild,info);
return (ptr);
}
struct node *insert(int info,struct node *pptr,int *ht_inc)
```

```
{
struct node *aptr;
struct node *bptr;
if(pptr==NULL)
{
pptr=(struct node *)malloc(sizeof(struct node));
pptr>
info=info;
pptr>
lchild=NULL;
pptr>
rchild=NULL;
pptr>
balance=0;
*ht_inc=TRUE;
return(pptr);
}
if(info<pptr>
info)
{
pptr>
lchild=insert(info,pptr>
lchild,ht_inc);
if(*ht_inc==TRUE)
{
switch(pptr>
balance)
{
case 1:
pptr>
```

```
balance=0;
*ht_inc=FALSE;
break;
case 0:
pptr>
balance=1;
break;
case 1:
aptr=pptr>
lchild;
if(aptr>
balance==1)
{
printf("Left to Left Rotation\n");
pptr>
lchild=aptr>
rchild;
aptr>
rchild=pptr;
pptr>
balance=0;
aptr>
balance=0;
pptr=aptr;
}
else
{
printf("Left to Right Rotation\n");
bptr=aptr>
rchild;
```

```
aptr>
rchild=bptr>
lchild;
bptr>
lchild=aptr;
pptr>
lchild=bptr>
rchild;
bptr>
rchild=pptr;
if(bptr>
balance==1)
pptr>
balance=1;
else
pptr>
balance=0;
if(bptr>
balance==1)
aptr>
balance=1;
else
aptr>
balance=0;
bptr>
balance=0;
pptr=bptr;
}
*ht_inc=FALSE;
}
```

```
}
}
if(info>pptr>
info)
{
pptr>
rchild=insert(info,pptr>
rchild,ht_inc);
if(*ht_inc==TRUE)
{
switch(pptr>
balance)
{
case 1:
pptr>
balance=0;
*ht_inc=FALSE;
break;
case 0:
pptr>
balance=1;
break;
case 1:
aptr=pptr>
rchild;
if(aptr>
balance==1)
{
printf("Right to Right Rotation\n");
pptr>
rchild=aptr>
```

```
lchild;
aptr>
lchild=pptr;
pptr>
balance=0;
aptr>
balance=0;
pptr=aptr;
}
else
{
printf("Right to Left Rotation\n");
bptr=aptr>
lchild;
aptr>
lchild=bptr>
rchild;
bptr>
rchild=aptr;
pptr>
rchild=bptr>
lchild;
bptr>
lchild=pptr;
if(bptr>
balance==1)
pptr>
balance=1;
else
pptr>
```

```
balance=0;
if(bptr>
balance==1)
aptr>
balance=1;
else
aptr>
balance=0;
bptr>
balance=0;
pptr=bptr;
}
*ht_inc=FALSE;
}
}
}
return (pptr);
}
main()
{
bool ht_inc;
int info;
int choice;
clrscr();
root=(struct node *)malloc(sizeof(struct node));
root=NULL;
printf("1.Insert\n2.Display\n3.Exit\n");
while(1)
{
printf("Enter your choice :");
```

```
scanf("%d",&choice);
switch(choice)
{
case 1:
printf("Enter the value to be inserted ::");
scanf("%d",&info);
if(search(root,info)==NULL)
root=insert(info,root,&ht_inc);
else
printf("Duplicate value ignored\n");
break;
case 2:
if(root==NULL)
{
printf("Tree is empty");
continue;
}
printf("Tree is \n");
display(root,1);
printf("\n\n");
printf("Inorder Traversal :: ");
inorder(root);
printf("\n");
break;
default:
printf("Invalid Choice !!!");
exit(0);
}
}
}
```

```
display(struct node *ptr,int level)
{
int i;
if(ptr!=NULL)
{
display(ptr>
rchild,level+1);
printf("\n");
for(i=0;i<level;i++)
printf("");
printf("%d",ptr>
info);
display(ptr>
lchild,level+1);
}
}
inorder(struct node *ptr)
{
if(ptr!=NULL)
{
inorder(ptr>
lchild);
printf("%d ",ptr>
info);
inorder(ptr>
rchild);
}
}
```

OUTPUT

```
1.Insert
2.Display
3.Exit
Enter your choice :1
Enter the value to be inserted ::15
Enter your choice :1
Enter the value to be inserted ::12
Enter your choice :1
Enter the value to be inserted ::24
Enter your choice :1
Enter the value to be inserted ::6
Enter your choice :2
Tree is
24
15
12
6
Inorder Traversal :: 6 12 15 24
Enter your choice :3
```

Ex.no.:8 **PRIORITY QUEUE USING HEAP**

Aim:

To implement priority queue using Heap in C program.

Algorithm:

Step 1: [Include necessary header files]

Step 2: [Define maxsize as 15]

Step 3: [Declare necessary variables]

Step 4: READ option, opt

IF opt is 1 THEN CALL INSERT()

IF opt is 2 THEN CALL DELMAX()

IF opt is 3 THEN CALL DIS()

Step 5: [END OF MAIN FUNCTION]

Algorithm For INSERT()

Step 1: I ne1+1

Step 2: IF (I MAXSIZE)

WRITE (“ Heap size exceeded”)

RETURN FALSE

IF ((I> 1) && (arraysize [i/2]< item))

array[I] array[i/2]

I I/2

Array[I] item

RETURN TRUE

Algorithm For DELMAX()

Step 1: IF (!nel)

WRITE (“HEAP IS EMPTY”)

ELSE

*item array [I]

Array[i] array [nel]

CALL adjust (array,I,nel)

PROGRAM

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#include<malloc.h>
typedef struct heapstruct *pqueue;
struct heapstruct
{
int capacity;
int size;
int *elements;
};
void insert(int,pqueue);
pqueue initialize(int);
int deletemin(pqueue);
int isfull(pqueue);
int isempty(pqueue);
void display(pqueue);
void main()
{
pqueue heap;
int i,max,ele,ch,t;
clrscr();
printf("\nEnter the maximum no.of elements in the priority queue:");
scanf("%d",&max);
heap=initialize(max);
do
{
printf("\nMENU\n");
printf("\n1. Insertion\n");
printf("\n2.DeleteMin\n");
```

```
printf("\n3. Display\n");
printf("\n4. Exit\n");
printf("\nEnter your choice:");
scanf("%d",&ch);
switch(ch)
{
case 1: printf("\nEnter the element to be inserted:");
scanf("%d",&ele);
insert(ele,heap);
printf("\nThe element is inserted");
break;
case 2: t=deletemin(heap);
printf("\nThe minimum element %d is deleted\n",t);
break;
case 3: printf("\nThe elements in the HEAP are:");
display(heap);
break;
case 4: exit(0);
break;
}
}while(ch<4);
getch();
}
pqueue initialize(int max)
{
pqueue h;
if(max<3)
{
printf("\nPriority queue size is too small\n");
exit(0);
```

```
}
h=(heapstruct*)malloc(sizeof(struct heapstruct));
if(h==NULL)
exit(0);
h>
capacity=max;
h>
size=0;
return h;
}
void insert(int x,pqueue h)
{
int i;
if(isfull(h))
{
printf("\nPriority queue is full");
return;
}
if(h>
size==0)
{
h>
elements[1]=x;
h>
size++;
}
else
{
for(i=++h>
size;h>
```

```
elements[i/2]>x;i/=2)
h>
elements[i]=h>
elements[i/2];
h>
elements[i]=x;
}
}
int deletemin(pqueue h)
{
int i,child,minelement,lastelement;
if(isempty(h))
printf("\nPriority queue is empty");
exit(0);
}
minelement=h>
elements[1];
lastelement=h>
elements[h>
size]
;
for(i=1;i*2<=h>
size;i=child)
{
child=i*2;
if(child!=h>
size&&h>
elements[child+1]<h>
elements[child])
child++;
```

```
if(lastelement>h>
elements[child])
h>
elements[i]=h>
elements[child];
else
break;
}
h>
elements[i]=lastelement;
return minelement;
}
void display(pqueue h)
{
int i;
for(i=1;i<=h>
size;i++)
printf("\n%d",h>
elements[i]);
}
int isfull(pqueue h)
{
if(h>
size==h>
capacity)
return 1;
else
return 0;
}
int isempty(pqueue h)
{
```

```
if(h>
size==0)
return 1;
else
return 0;
}
```

OUTPUT

Enter the maximum no.of elements in the priority queue:5

MENU

1. Insertion

2.DeleteMin

3. Display

4. Exit

Enter your choice:1

Enter the element to be inserted:67

The element is inserted

MENU

1. Insertion

2.DeleteMin

3. Display

4. Exit

Enter your choice:1

Enter the element to be inserted:24

The element is inserted

MENU

1. Insertion

2.DeleteMin

3. Display

4. Exit

Enter your choice:1

```
Enter the element to be inserted:35
The element is inserted
MENU
1. Insertion
2.DeleteMin
3. Display
4. Exit
Enter your choice:3
The elements in the HEAP are:
24
67
35
MENU
1. Insertion
2.DeleteMin
3. Display
4. Exit
Enter your choice:2
The minimum element 24 is deleted
MENU
1. Insertion
2.DeleteMin
3. Display
4. Exit
Enter your choice:3
The elements in the HEAP are:
35
67
Enter your choice:4
```

Ex no 9 **GRAPH TRAVERSAL USING DEPTH -FIRST SEARCH**

Algorithm :

Step 1: Choose any node in the graph . Designate it as the search node and mark it as vivited.

Step 2: Using the adjacency matrix of the graph, find a node adjacent to the search node that has not been visited yet. Designate this as the new search node and mark it as visited.

Step 3: Repeat step 2 using t he new search node. If no nodes satisfying(2) can be found, return to the previous search node and continue from there.

Step 4: When a return to the previous search in(3) is impossible,the serach from the originally choosen search node is complete.

Step 5: If the graph still contains unvisited nodes,choose any node that has not been visited and repeat step(1) through(4).

PROGRAM

```
#include<stdio.h>
#include<conio.h>
int a [10][10],visited[10].n;
void main()
{
        int i,j;
        void search from(int);
        clrscr();
        printf("enter the no. of nodes\n");
        scanf("%d",&n);
        printf("enter the adjacency matrix\n");
for(i=1;<=n;i++)
for(j=1;<=n;j++)
scanf("%d",&a[i][j]);
for(i=1;i<=n;i++)
visited[i]=0;
printf("Depth First Path:");
for(i=1;i<=n;i++)
if(visited[i]==0)
searchfrom(i);
}
void search from(int k)
{
        int i;
        printf("%d\t",k);
        visited[k]=1;
        for(i=1;i<=n;i++)
        if(visited[i]==0)
        searchfrom(i);
        return;
}
```

OUTPUT

```
 Enter the no. of nodes
4
Enter the adjacency matrix
0 1 0 1
0 0 1 1
0 0 0 1
0 0 0 0

Depth First Path
1     2     3     4
```

Ex.no.10

DIJKSTRA'S ALGORITHM

Aim

To implement Dijkstra's algorithm to find the shortest path.

Algorithm

Step1: [Include all the header files]

Step2: Call allSelected( )

Step3: Call Shortpath( )

Step4: Access the functions from main

Step5: End

Algorithm For ALLSELECTED( )

Step1: Initialise i=0

Step2: Check whether i<max

Step3: Check whether Selected[i]=0

Return 0

Step4: Else Return 1

Step5: Return

Algorithm For SHORTPATH( )

Step1: Initialise i=0 , Check i<max

Distance[i]=INFINITE

Step2: Assign selected[current].distance[0]=0,

Current=0

Step3: While(!allSelected(Selected))

Perform(Selected[i]= =0)

Current=k

Selected[current]=1

Print k

PROGRAM

```
#include<stdio.h>
#include<conio.h>
```


www.vidyarthiplus.com

```
#define max 4
#define INFINITE 998
int allselected( int *selected)
{
int i;
for(i=0;i<max;i++)
if(selected[i]==0)
return 0;
return 1;
}
void shortpath(int cost[][max],int *preceed,int *distance)
{
int selected[max]={0};
int current=0,i,k,dc,smalldist,newdist;
for(i=0;i<max;i++)
distance[i]=INFINITE;
selected[current]=1;
distance[0]=0;
current=0;
while(!allselected(selected))
{
smalldist=INFINITE;
dc=distance[current];
for(i=0;i<max;i++)
{
if(selected[i]==0)
{
newdist=dc+cost[current][i];
```

```
 if(newdist<distance[i])
 {
 distance[i]=newdist;
 preceed[i]=current;
 }
 if(distance[i]<smalldist)
 {
 smalldist=distance[i];
 k=i;
 }
 }}
 current=k;
 selected[current]=1;
 }
 }
 int main()
 {
 int
 cost[max][max]={{INFINITE,2,4,INFINITE},{2,INFINITE,1,5},{4,1,INFINITE,2},{INFINITE
 ,5,2,INFINITE}};
 int preceed[max]={0},i,distance[max];
 clrscr();
 shortpath(cost,preceed,distance);
 for(i=0;i<max;i++)
 {
 printf("The shortest path from 0 to %d is ",i);
 printf("%d\n",distance[i]);
 }
 return 0;
 getch();
}
```

OUTPUT

The shortest path from 0 to 0 is 0

The shortest path from 0 to 1 is 2

The shortest path from 0 to 2 is 3

The shortest path from 0 to 3 is 5

EX NO 11(A) IMPLEMENTATION OF SEARCHING ALGORITHM

Algorithm :

Step 1: Read the elements of the list.

Step 2: Sort the input list.

Step 3: Find the mid value.

Step 4: Look at the element in the middle. If the key is equal to that, the search is finished.

Step 5: If the key is less than the middle element, do a binary search on the first half.

Step 6: If it's greater, do a binary search of the second half.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int a[25],i,j,temp,s,n,low,mid,high;
        clrscr();
        printf("\nEnter the Limilt : ");
        scanf("%d",&n);
        printf("\n\nEnter the elements\n");
        for(i=0;i<n;i++)
        {

            scanf("%d",&a[i]);
        }
        for(i=0;i<n-1;i++)
        {
          for(j=0;j<n-1;j++)
          {
            if(a[j]>a[j+1])
            {
              temp=a[j];
              a[j]=a[j+1];
              a[j+1]=temp;
            }
          }
        }
        printf("\n\nSorted list");
        for(i=0;i<n;i++)
        {
         printf("\n%d",a[i]);
        }
        printf("\n\nEnter the elements to be searched : ");
        scanf("%d",&s);
        high=n-1;
        low=0;
        while(low<=high)
        {
            mid=(low+high)/2;
            if(s>a[mid])
            low=mid+1;
            else if(s<a[mid])
            high=mid-1;
            else if(s==a[mid])
            {
                printf("\n\nThe element %d is found",s);
                getch();
                exit(0);
            }
         }
```

```
        printf("\n\nThe element %d is not found",s);
        getch();
}
```

OUTPUT

```
 Enter the elements
5
4
3
2
1

Sorted list
1
2
3
4
5

Enter the element to be searched : 5

The element 5 is found.
```

Ex no 11(b) **IMPLEMENTATION SELECTION SORT**

ALGORITHM:

1. Find the minimum value in the list
2. Swap it with the value in the first position
3. Repeat the steps above for the remainder of the list (starting at the second position and advancing each time)

PROGRAM

```
#include<stdio.h>
#include<conio.h>
int n,i=0,j=0,t=0,k=0,a[30];
void main()
{
	clrscr();
	printf("\nEnter how many numbers you want to sort\n");
	scanf("%d",&n);
	printf("\nEntyer the numbers \n");
	for (i=0;i<n;i++)
	{
		scanf("%d",&a[i]);
	}
	for (i=1;i<n;i++)
	{
		printf("\n\nPASS %d-->",i);
		t=a[i];
		for(j=i-1;((j>=0)&&(t<a[j]));j--)
		a[j+1]=a[j];
		a[j+1]=t; //j decreases
		for(k=0;k<n;k++)
		printf("%d ",a[k]);
	}
	printf("\n\nThe sorted list is : ");
	for (j=0;j<n;j++)
	printf("%d ",a[j]);
getch();
}
```

OUTPUT

```
Enter how many elements you want to sort

5

Enter the numbers
5
4
3
2
1

PASS->1  1 5 4 3 2
PASS->2  1 2 5 4 3
PASS->3  1 2 3 5 4
PASS->4  1 2 3 4 5

Final sorted list is 1 2 3 4 5
```

Ex no 12 **HASHING - COLLISION TECHNIQUE**

ALGORITHM

1.Create an array of linked list(i.e) hash table.
2. take a key and a value to be stored in hash table as input.
3.Using the generated index extract the linked list sorted in that array index.
4.Incase of absence of a linked list, create one and insert a data item into it.
5. incase list exit search for the key in the linked list and add the data item at the end of the list.
6. To display all the element of hash table, linked list at each index is extracted and element are read until we reach at its end.
7.TO remove a key from hash table we will first calculate its index and extract its linked list.

PROGRAM

```
#include <stdio.h>
#include <conio.h>
int tsize;

int hasht(int key)
{
 int i ;
 i = key%tsize ;
 return i;
}

//-------LINEAR PROBING-------
int rehashl(int key)
{
 int i ;
 i = (key+1)%tsize ;
 return i ;
}

//-------QUADRATIC PROBING-------
int rehashq(int key, int j)
{
 int i ;
 i = (key+(j*j))%tsize ;
 return i ;
}

void main()
{
    int key,arr[20],hash[20],i,n,s,op,j,k ;
    clrscr() ;
    printf ("Enter the size of the hash table:  ");
    scanf ("%d",&tsize);

    printf ("\nEnter the number of elements: ");
    scanf ("%d",&n);

    for (i=0;i<tsize;i++)
 hash[i]=-1 ;

    printf ("Enter Elements: ");
    for (i=0;i<n;i++)
    {
 scanf("%d",&arr[i]);
    }

    do
    {
 printf("\n\n1.Linear Probing\n2.Quadratic Probing \n3.Exit \nEnter your option: ");
 scanf("%d",&op);
```

```
switch(op)
{
case 1:
    for (i=0;i<tsize;i++)
    hash[i]=-1 ;

    for(k=0;k<n;k++)
    {
 key=arr[k] ;
 i = hasht(key);
 while (hash[i]!=-1)
 {
     i = rehashl(i);
 }
 hash[i]=key ;
    }
    printf("\nThe elements in the array are: ");
    for (i=0;i<tsize;i++)
    {
 printf("\n  Element at position %d: %d",i,hash[i]);
    }
    break ;

case 2:
    for (i=0;i<tsize;i++)
 hash[i]=-1 ;

    for(k=0;k<n;k++)
    {
 j=1;
 key=arr[k] ;
 i = hasht(key);
 while (hash[i]!=-1)
 {
     i = rehashq(i,j);
     j++ ;
 }
 hash[i]=key ;
    }
    printf("\nThe elements in the array are: ");
    for (i=0;i<tsize;i++)
    {
 printf("\n Element at position %d: %d",i,hash[i]);
    }
    break ;

 }
   }while(op!=3);

   getch() ;
}
```

OUTPUT

```
Enter the size of the hash table:  10

Enter the number of elements: 8
Enter Elements: 72 27 36 24 63 81 92 101


1.Linear Probing
2.Quadratic Probing
3.Exit
Enter your option: 1

The elements in the array are:
  Element at position 0: -1
  Element at position 1: 81
  Element at position 2: 72
  Element at position 3: 63
  Element at position 4: 24
  Element at position 5: 92
  Element at position 6: 36
  Element at position 7: 27
  Element at position 8: 101
  Element at position 9: -1

1.Linear Probing
2.Quadratic Probing
3.Exit
Enter your option: 2

The elements in the array are:
 Element at position 0: -1
 Element at position 1: 81
 Element at position 2: 72
 Element at position 3: 63
 Element at position 4: 24
 Element at position 5: 101
 Element at position 6: 36
 Element at position 7: 27
 Element at position 8: 92
 Element at position 9: -1
```

MADHA ENGINEERING COLLEGE

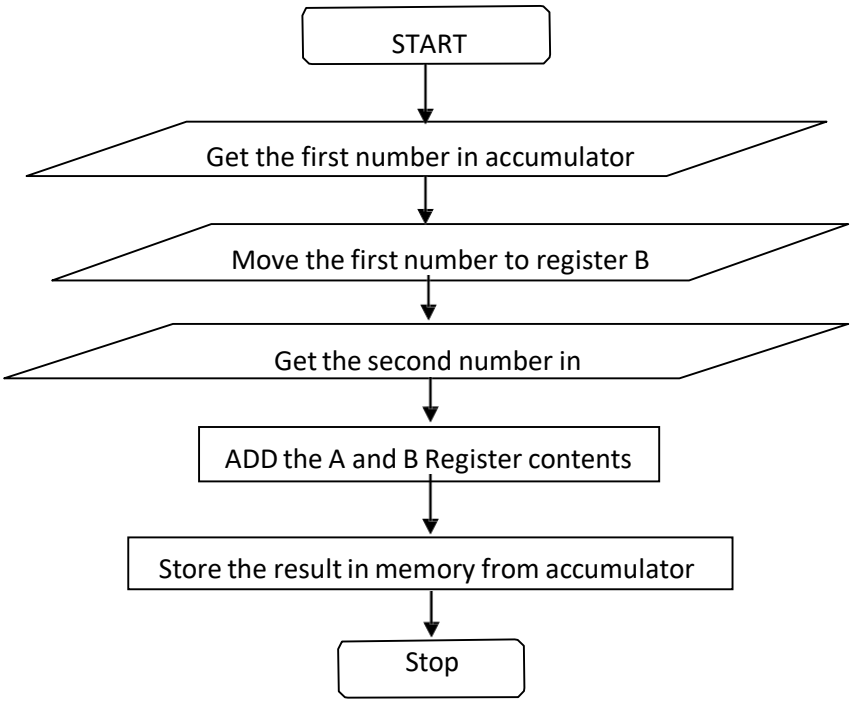
(A Christian Minority Institution)

KUNDRATHUR, CHENNAI – 600 069

Microprocessor and Microcontroller Lab Manual

Name	**:**	
Subject	**:**	
Roll No.	**:**	
Semester	**:**	**Year:**

FLOW CHART:



INPUT & OUTPUT TABULATION:

Memory Address	**Input data**	**Memory Address**	**Output data**
8200 8201		8202	
8200 8201		8202	

Ex. No.: 1 A

Date : **ADDITION OF TWO 8-BIT DATA WITHOUT CARRY**

AIM:

To add two 8 bit numbers stored at consecutive memory location using 8085 microprocessor without carry.

APPARATUS REQUIRED:

- 8085 microprocessor kit
- OPcode sheet

ALGORITHM:

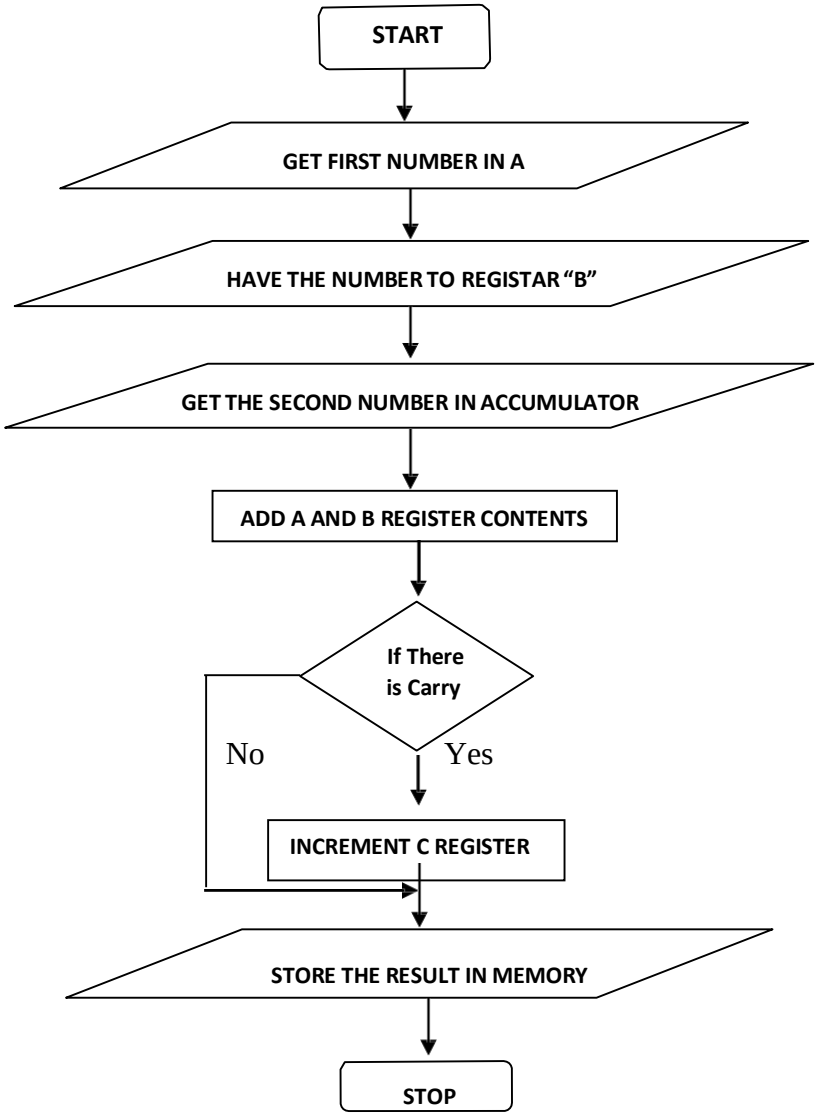
1. Start the program by initializing memory pointer to data location.
2. Get the first number and store in accumulator.
3. Move the first number to register B.
4. Get second number and store in accumulator A.
5. Add two numbers and result is in accumulator A.
6. Store the result from accumulator to memory.
7. Stop the program.

PROGRAM:

ADDRESS	LABEL	PNEUMONIC	OPCODE	COMMENTS
8100	START	LDA 8200	3A	Load the first number in accumulator from Memory
8101			00	
8102			82	
8103		MOV B, A	47	Move the data from accumulator to B
8104		LDA 8201	3A	Load the Second number in accumulator from Memory
8105			01	
8106			82	
8107		ADD B	80	Addition of B with A register values.
8108		STA 8202	32	Store the result from accumulator to Memory
8109			02	
810A			82	
810B		HLT	76	Stop the program

RESULT:

FLOWCHART:


START
GET FIRST NUMBER IN A
HAVE THE NUMBER TO REGISTAR "B"
GET THE SECOND NUMBER IN ACCUMULATOR
ADD A AND B REGISTER CONTENTS
If There is Carry
No
Yes
INCREMENT C REGISTER
STORE THE RESULT IN MEMORY
STOP

Ex. No.: 1 B

Date : ADDITION OF TWO 8-BIT DATA WITH CARRY

AIM:

To add two 8-bit numbers stored at consecutive memory location using 8085 microprocessor with carry.

APPARATUS REQUIRED:

- 8085 microprocessor kit
- OPcode sheet

ALGORITHM:

1. Start the program by initializing the memory pointer to data location.
2. Get the first number or data in accumulator.
3. Move the first number to register B.
4. Get the second number in accumulator A.
5. Add two numbers and result is in accumulator A.
6. If carry is present, increment register C by one, otherwise go to next step.
7. Store the result in memory from accumulator and register C.
8. Stop the program.

PROGRAM:

ADDRESS	LABEL	PNEMONICS	OPCODE	COMMENTS
8100	START	LDA 8200	3A	Load First Data in Accumulator A
8101			00	
8102			82	
8103		MOV B, A	47	Move Data from Accumulator To B
8104		LDA 8201	3A	Load Second Data in Accumulator A
8105			01	
8106			82	
8107		MVI C,00	0E	Clear C Register
8108			00	
8109		ADD B	80	Addition of B With A
810A		JNC LOOP	D2	Jump to Loop , If Result does not have Carry
810B			0E	
810C			81	
810D		INR C	0C	Increment C Register
810E	LOOP	STA 8202	32	Store the Result in Memory from Accumulator
810F			02	
8110			82	

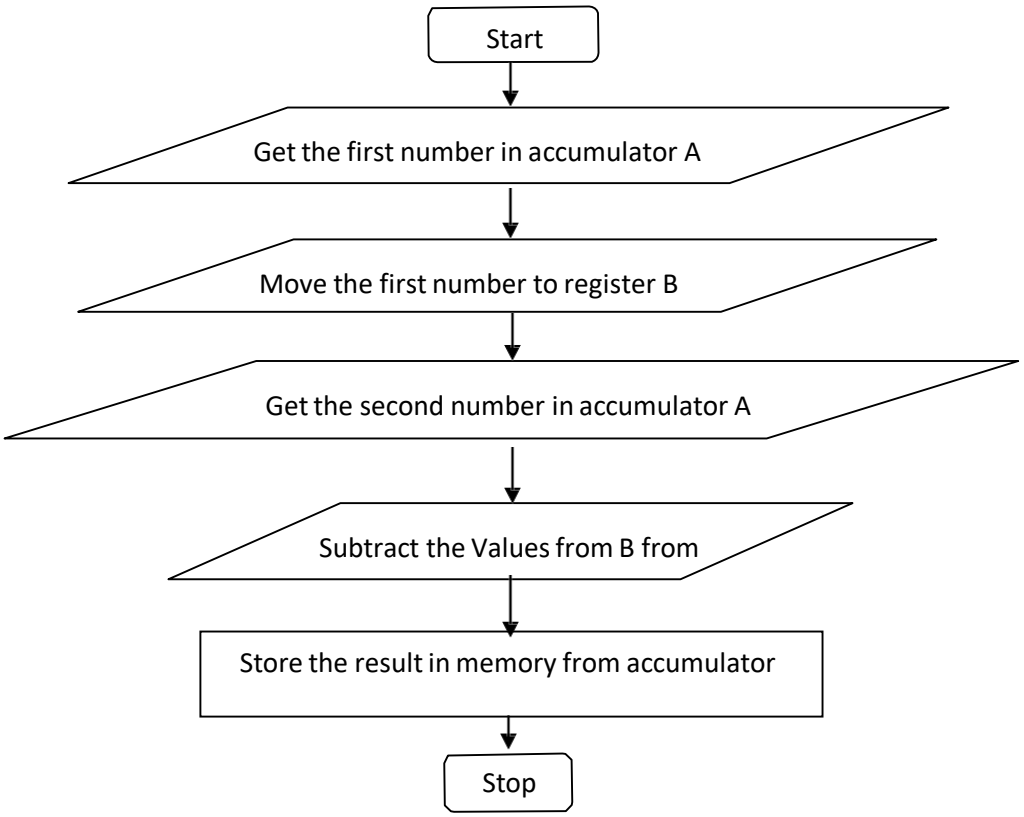
8111		MOV A,C	79	Move the Carry from C to Accumulator & Store Carry in Memory from Accumulator
8112		STA 8203	32	
8113			03	
8114			82	
8115		HLT	76	Stop the Program

INPUT & OUTPUT TABULATION:

MEMORY ADDRESS	**INPUT DATA**	**MEMORY ADDRESS**	**OUTPUT DATA**
8200 8201		8202 8203	
8200 8201		8202 8203	

RESULT:

FLOW CHART:



INPUT & OUTPUT TABULATION:

Memory address	Input data	Memory address	Output data
8200 8201		8202	
8200 8201		8202	

Ex. No.: 2 A

Date : SUBTRACTION OF TWO 8 BIT DATA WITHOUT CARRY

AIM:

To subtract two 8 bit data's stored at memory location without carry using 8085 microprocessor

APPARATUS REQUIRED:

- 8085 microprocessor kit
- OPcode

sheetALGORITHM:

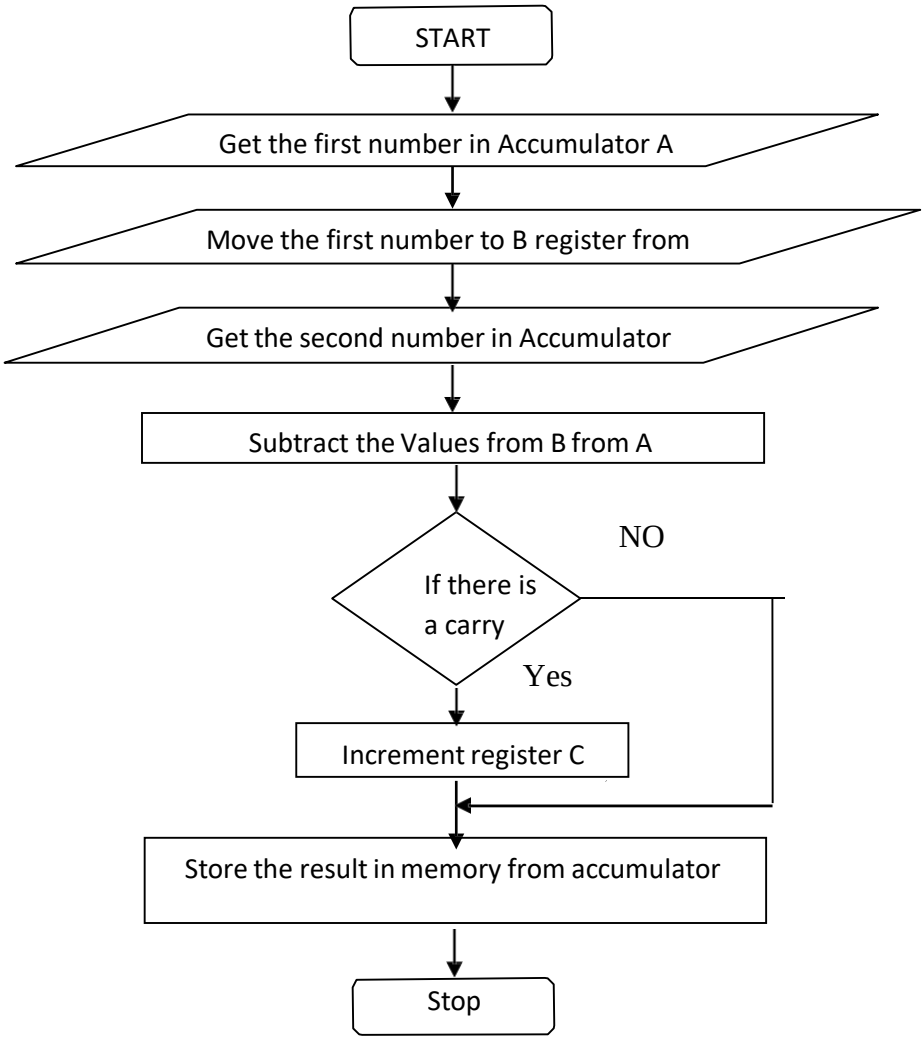
1. Start the program by initializing the memory pointer to data location
2. Get the first number from memory to accumulator
3. Move the first number to register B
4. Get the second number in accumulator from memory
5. Store the result in memory from accumulator
6. Stop the program

PROGRAM:

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
8100	START	LDA 8200	3A	Load the first data in accumulator A from memory
8101			00	
8102			82	
8103		MOV B,A	47	Move the first data to register B form accumulator A
8104		LDA 8201	3A	Load the Second data in accumulator A from memory
8105			01	
8106			82	
8107		SUB B	90	Subtract the value from B from A
8108		STA 8202	32	Store the result in memory from accumulator
8109			02	
810A			82	
810B		HLT	76	Stop the program

RESULT:

FLOWCHART:


START
Get the first number in Accumulator A
Move the first number to B register from
Get the second number in Accumulator
Subtract the Values from B from A
If there is a carry
NO
Yes
Increment register C
Store the result in memory from accumulator
Stop

Ex. No.: 2 B

Date : SUBTRACTION OF TWO 8-BIT DATA WITH CARRY

AIM:

To subtract two 8-bit numbers stored at consecutive memory location using 8085.

APPARATUS REQUIRED:

- 8085 microprocessor kit
- OPcode sheet

ALGORITHM:

1. Start the program by initializing the memory location to data pointer.
2. Get the first number from memory in accumulator.
3. Move the first number to register B.
4. Get the second number from memory in accumulator.
5. Subtract two numbers (B from A) and store it in accumulator.
6. Store the result in memory from accumulator.
7. Stop the program.

PROGRAM:

ADDRESS	LABEL	PNEUMONIC	OPCODE	COMMENTS
8100	START	LDA 8200	3A	Load the first data in accumulator A from memory
8101			00	
8102			82	
8103		MOV B,A	47	Move data from A to B
8104		LDA 8201	3A	Load the second data in accumulator A from memory
8105			01	
8106			82	
8107		MVI C,00	0E	Clear C register
8108			00	
8109		SUB B	90	Subtract B from A

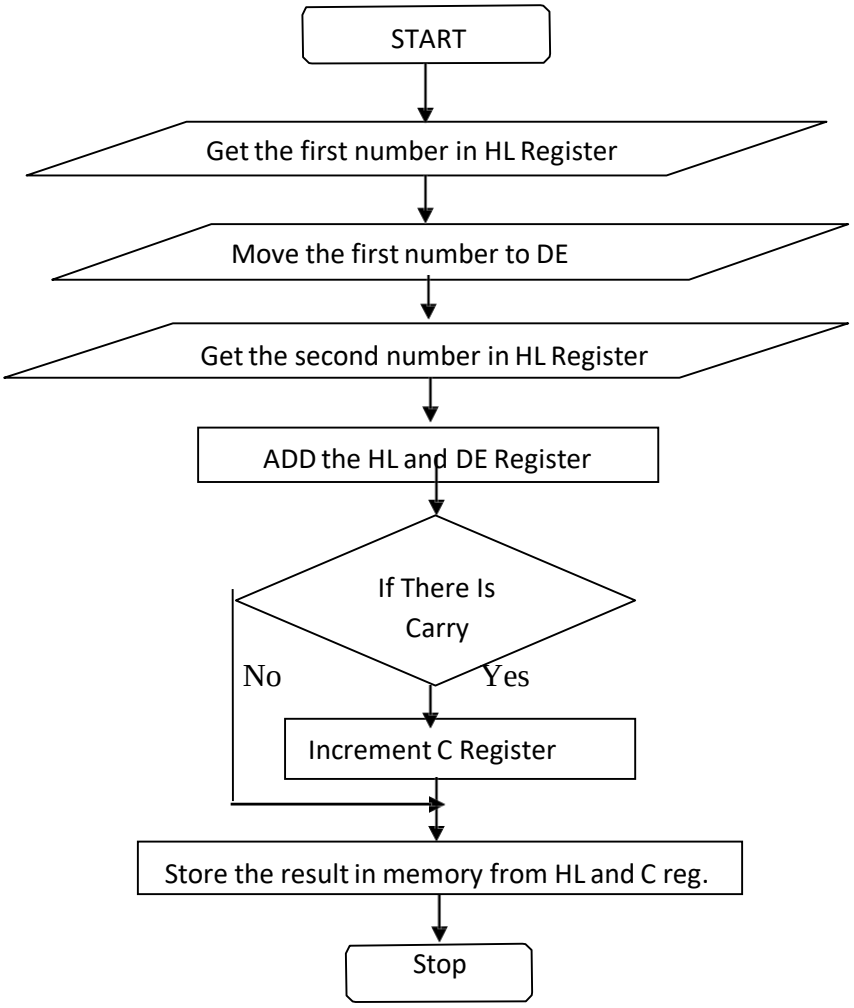
INPUT & OUTPUT TABULATION:

Memory Address	Input data	Memory Address	Input data
8200 8201		8202 8203	
8200 8201		8202 8203	

810A		JNC LOOP	D2	Jump to location of the result doesn't have carry
810B			0E	
810C			81	
810D		INRC	0C	Increment C register
810E	LOOP	STA 8202	32	Store the result from accumulator
810F			02	
8110			82	
8111		MOV A,C	79	Move Borrow from C to A
8112		STA 8203	32	Store carry value from accumulator
8113			03	
8114			82	
8115		HLT	76	Stop the program

RESULT:

FLOW CHART:


START
Get the first number in HL Register
Move the first number to DE
Get the second number in HL Register
ADD the HL and DE Register
If There Is Carry
No
Yes
Increment C Register
Store the result in memory from HL and C reg.
Stop

Ex. No.: 3 A

Date : ADDITION OF TWO 16-BIT DATA

AIM:

To add two 16 bit numbers stored at consecutive memory location using 8085 microprocessor with carry.

APPARATUS REQUIRED:

- 8085 microprocessor kit
- OPcode sheet

ALGORITHM:

1. Start the program by initializing memory pointer to data location.
2. Get the first number and store in HL register.
3. Move the first number to register DE register.
4. Get second number and store in HL register.
5. Add two numbers and result is in HL register and C register.
6. Store the result from HL & C register to memory.
7. Stop the program.

PROGRAM:

ADDRESS	LABEL	PNEMONICS	OPCODE	COMMENTS
8100	START	MVI C,00	0E	Clear C Register
8101			00	
8102		LHLD 8200	2A	Load First Data in HL register
8103			00	
8104			82	
8105		XCHG	EB	Move Data To DE register
8106		LHLD 8202	2A	Load Second Data in HL register
8107			02	
8108			82	
8109		DAD D	19	Add HL & DE registers
810A		JNC LOOP	D2	Jump to Loop , If Result does not have Carry
810B			0E	
810C			81	
810D		INR C	0C	Increment C Register
810E	LOOP	SHLD 8300	22	Store the Result in Memory from HL register
810F			00	
8110			83	

8111		MOV A,C	79	Move the Carry from C to Accumulator & Store Carry in Memory from Accumulator
8112		STA 8302	32	
8113			03	
8114			82	
8115		HLT	76	Stop the Program

INPUT & OUTPUT TABULATION:

Memory Address	**Input data**	**Memory Address**	**Output data**
8200 8201		8300 8301	
8202 8203		8302	

RESULT:

Ex. No.: 3 B

Date : SUBTRACTION OF TWO 16-BIT DATA

AIM:

To subtract two 16-bit numbers stored at consecutive memory location using 8085.

APPARATUS REQUIRED:

- 8085 microprocessor kit
- OPcode sheet

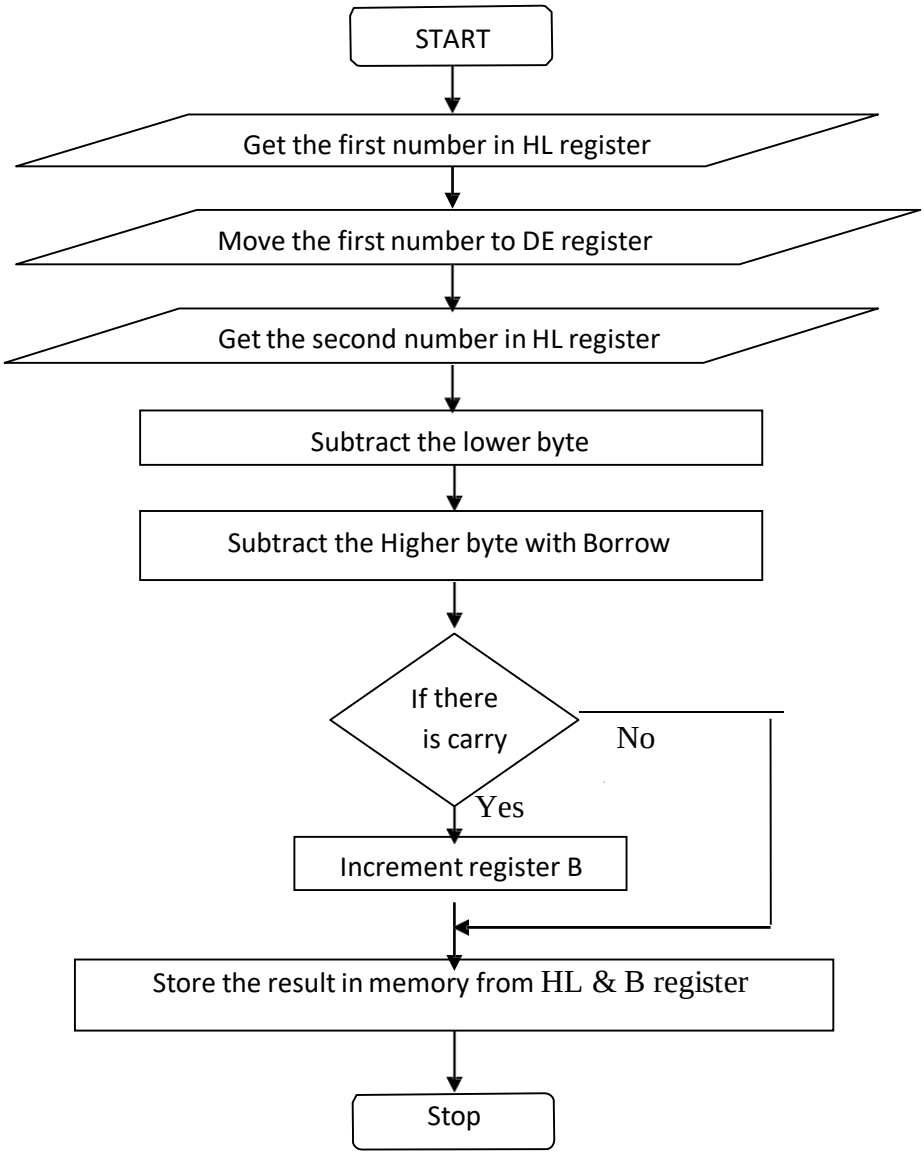
ALGORITHM:

1. Start the program by initializing the memory location to data pointer.
2. Get the first number from memory in HL register.
3. Move the first number to DE register.
4. Get the second number from memory in HL register.
5. First Subtract Lower byte and then Higher byte with borrow.
6. If Borrow is present increment the B register.
7. Store the result in memory from HL & B register.
8. Stop the program.

PROGRAM:

ADDRESS	LABEL	PNEUMONIC	OPCODE	COMMENTS
8100		LXI B,0000	01	Clear B register
8101			00	
8102			00	
8103		LHLD 8200	2A	Load the first data in HL register from memory
8104			00	
8105			82	
8106		XCHG	EB	Move data to DE register
8107		LHLD 8202	2A	Load the second data in HL register from memory
8108			02	
8109			82	

FLOWCHART:


START
Get the first number in HL register
Move the first number to DE register
Get the second number in HL register
Subtract the lower byte
Subtract the Higher byte with Borrow
If there is carry
No
Yes
Increment register B
Store the result in memory from HL & B register
Stop

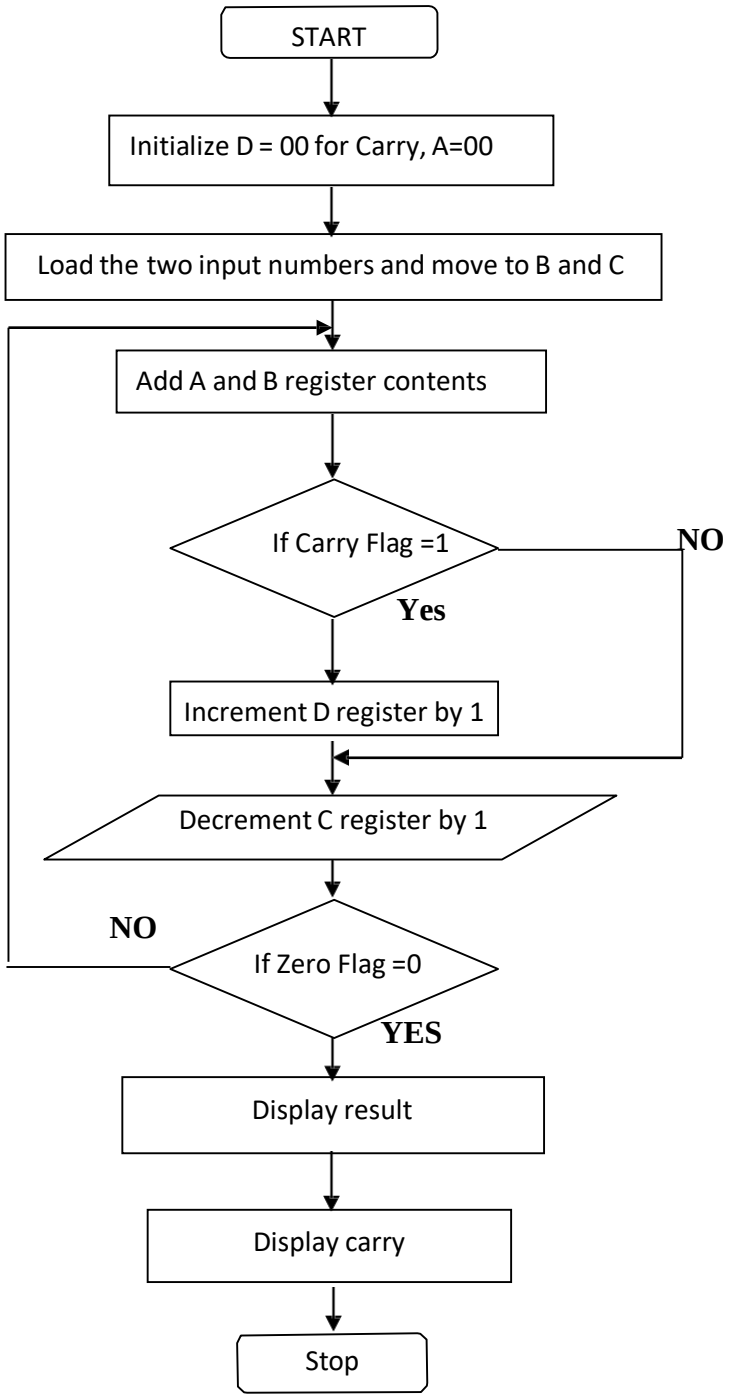
810A		MOV A,E	7B	Subtract lower bytes and move lower byte result to L register.
810B		SUB L	95	
810C		MOV L,A	6F	
810D		MOV A,D	7A	Subtract higher bytes
810E		SBB H	9C	
810F		JNC LOOP	D2	Jump to location of the result doesn't have carry
8110			13	
8111			81	
8112		INX B	03	Increment B register
8113	LOOP	MOV H,A	67	Move higher byte result to H register. Finally Store the result to memory from HL register.
8114		SHLD 8300	22	
8115			00	
8116			83	
8117		MOV A,B	78	Move borrow from B to A
8118		STA 8302	32	Store Borrow value from accumulator
8119			02	
811A			83	
811B		HLT	76	Stop the program

INPUT & OUTPUT TABULATION:

Memory Address	**Input data**	**Memory Address**	**Input data**
8200 8201		8300 8301	
8202 8203		8302	

RESULT:

FLOWCHART:


START
Initialize D = 00 for Carry, A=00
Load the two input numbers and move to B and C
Add A and B register contents
If Carry Flag =1
NO
Yes
Increment D register by 1
Decrement C register by 1
NO
If Zero Flag =0
YES
Display result
Display carry
Stop

Ex. No.: 4 A

Date : MULTIPLICATION OF TWO 8- BIT DATA

AIM:

To multiply two 8-bit numbers stored at consecutive memory location using 8085 microprocessor.

APPARATUS REQUIRED:

- 8085 microprocessor kit
- OPcode sheet

ALGORITHM:

1. Start the program and Initialize D = 00 for Carry, A=00
2. Load the memory address to HL register pair
3. Move the data to a B register
4. Get the second data and move into C register.
5. Add the two register B & C contents
6. If carry is present increment the D register by 1, Otherwise go to next step.
7. Decrement the C register by 1 and repeat the step 5 until C=0.
8. Store the value of product and carry in memory location
9. Terminate the program

PROGRAM:

ADDRESS	LABEL	PNEUMONIC	OPCODE	COMMENTS
8100	START	MVI D,00	16	Initialize register D to 00 for carry.
8101			00	
8102		MVI A,00	3E	Initialize Accumulator content to 00
8103			00	
8104		LXI H 8200	21	Get the first number in memory
8105			00	
8106			82	
8107		MOV B,M	46	Move the first number to B- register
8108		INX H	23	Increment memory by 1
8109		MOV C,M	4E	Get the second number in C – register
810A	LOOP	ADD B	80	Add content of A register with B

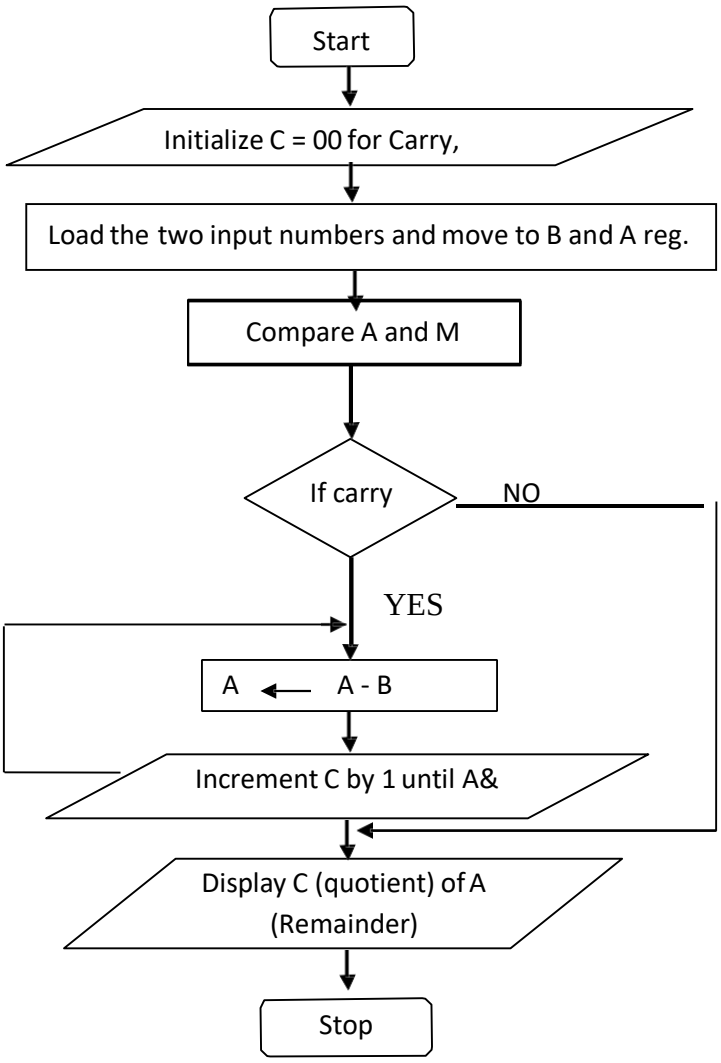
INPUT & OUTPUT TABULATION:

Memory Address	Input data	Memory Address	Output data
8200 8201		8202 8203	

810B		JNC NEXT	D2	Jump no carry to NEXT
810C			0F	
810D			81	
810E		INC D	14	Increment content of register D
810F	NEXT	DCR C	0D	Decrement content of register C
8110		JNZ LOOP	C2	Jump on no zero to LOOP
8111			0A	
8112			81	
8113		STA 8202	32	Store the result in memory
8114			02	
8115			82	
8116		MOV A,D	7A	Move D to A
8117		STA 8303	32	Store the MSB of result in memory
8118			03	
8119			82	
811A		HLT	76	Terminate the program

RESULT:

FLOW CHART:


Start
Initialize C = 00 for Carry,
Load the two input numbers and move to B and A reg.
Compare A and M
If carry
NO
YES
A ← A - B
Increment C by 1 until A&
Display C (quotient) of A
(Remainder)
Stop

Ex. No.: 4 B

Date : DIVISION OF TWO 8-BIT DATA

AIM:

To perform the division of two 8-bit numbers using 8085 microprocessor.

APPARATUS REQUIRED:

- 8085 microprocessor kit
- OPcode sheet

ALGORITHM:

1. Start the program by loading HL register pair with address of memory location.
2. Move the data to a register (B-register).
3. Get the second data and load into accumulator.
4. Compare the two numbers (A &B reg.) to check for carry, if carry present go to step 8.
5. Subtract the two numbers (A &B reg.).
6. Increment the value of C register for quotient.
7. If ZF=0, then repeat the step 4.
8. Store the value of remainder and Quotient in memory location.
9. Terminate the program.

PROGRAM:

ADDRESS	LABEL	PNEUMONIC	OPCODE	COMMENTS
8100	START	LXI H, 8200	21	Get the first number in memory
8101			00	
8102			82	
8103		MOV B, M	46	Get the dividend in B – register
8104		MVI C,00	0E	Clear C – register for quotient
8105			00	
8106		INX H	23	Increment memory by 1
8107		MOV A,M	7E	Get the divisor in A register
8108	NEXT	CMP B	B8	Compose A register with register B
8109		JC LOOP	DA	Jump on a carry to loop
810A			11	
810B			81	
810C		SUB B	90	Subtract A – register from B – register

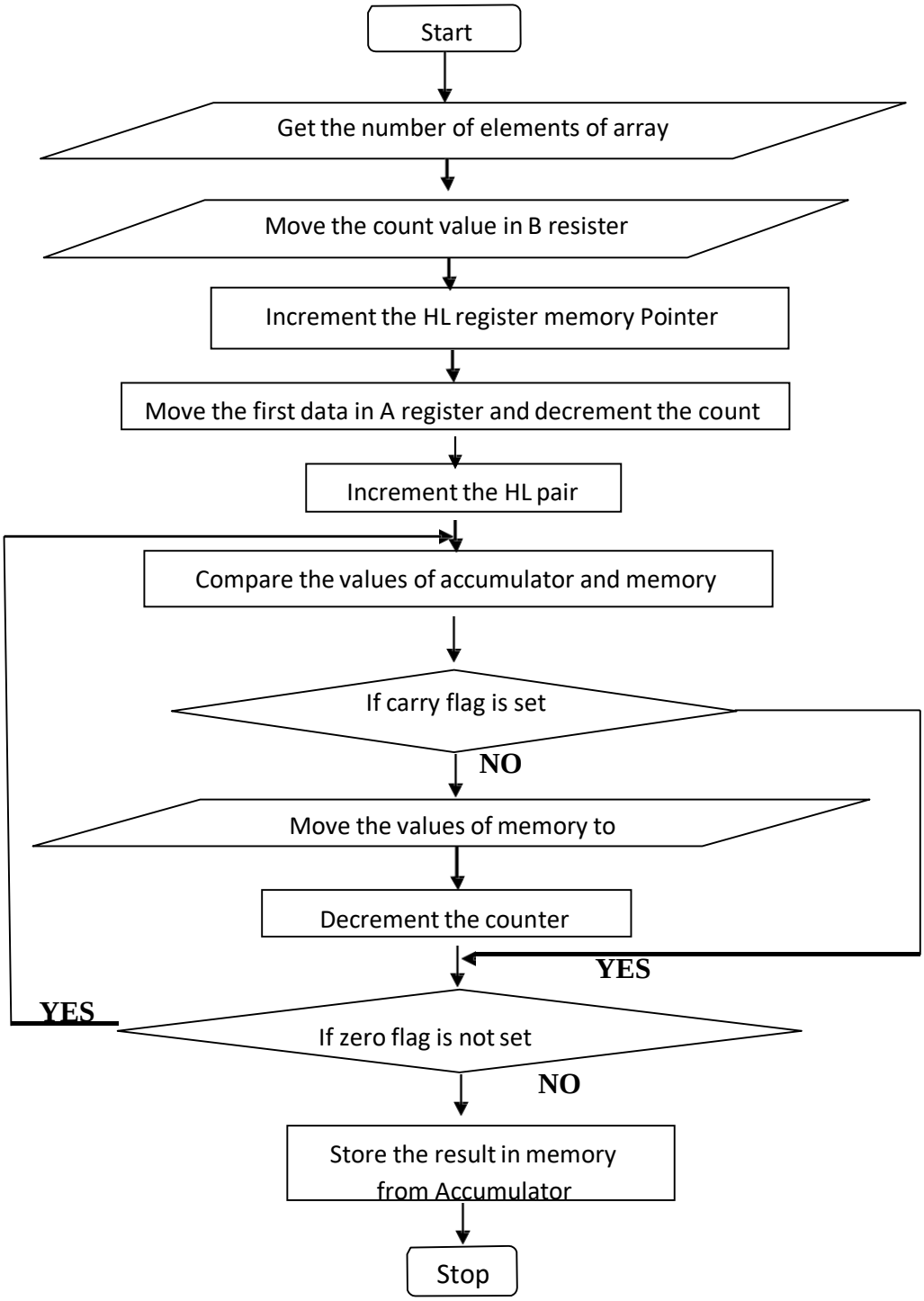
INPUT & OUTPUT TABULATION:

Memory Address	Input data	Memory Address	Output data
8200 8201		8202 8203	

810D		INR C	0C	Increment content Of register C
810E		JNZ NEXT	C2	Jump no zero to NEXT label.
810F			08	
8110			81	
8111	LOOP	STA 8202	32	Store the reminder in memory
8112			02	
8113			82	
8114		MOV A,C	79	Move C register value to Accumulator.
8115		STA 8203	32	Store the Quotient in memory
8116			03	
8117			82	
8118		HLT	76	Stop the program

RESULT:

FLOW CHART:


Start
Get the number of elements of array
Move the count value in B resister
Increment the HL register memory Pointer
Move the first data in A register and decrement the count
Increment the HL pair
Compare the values of accumulator and memory
If carry flag is set
NO
Move the values of memory to
Decrement the counter
YES
YES
If zero flag is not set
NO
Store the result in memory from Accumulator
Stop

Ex. No.: 5 A

Date : SMALLEST NUMBER IN AN ARRAY OF DATA

AIM:

To find the smallest number in an array of datas using 8085 microprocessor

APPARATUS REQUIRED:

- 8085 microprocessor kit
- OPcode sheet

ALGORITHM:

1. Load the address of the first element (count) of an array in HL pair.
2. Load the count and move it in to the B –register
3. Increment the HL pair as a pointer
4. Move the first data to A-register form memory which is pointed by HL pair.
5. Decrement the count (B reg.)
6. Increment the pointer (HL reg. pair)
7. Compare the content of memory addressed by HL pair with content of A-register
8. If carry =1 go for step 10 otherwise go to step-9
9. Move the content of memory addressed by HL pair to A-register
10. Decrement the count (B reg.)
11. Check for zero of the count if ZF=0 go to step 6 otherwise go to next step
12. Store the smallest data in memory from Accumulator.

PROGRAM:

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
8100	Start	LXI H, 8200	21	Set pointer for array.
8101			00	
8102			82	
8103		MOV B,M	46	Move the first data from memory to B- reg (Count)
8104		INX H	23	Increment the HL pair
8105		MOV A,M	7E	Move the second data from memory to accumulator.
8106		DCR B	05	Decrement the count

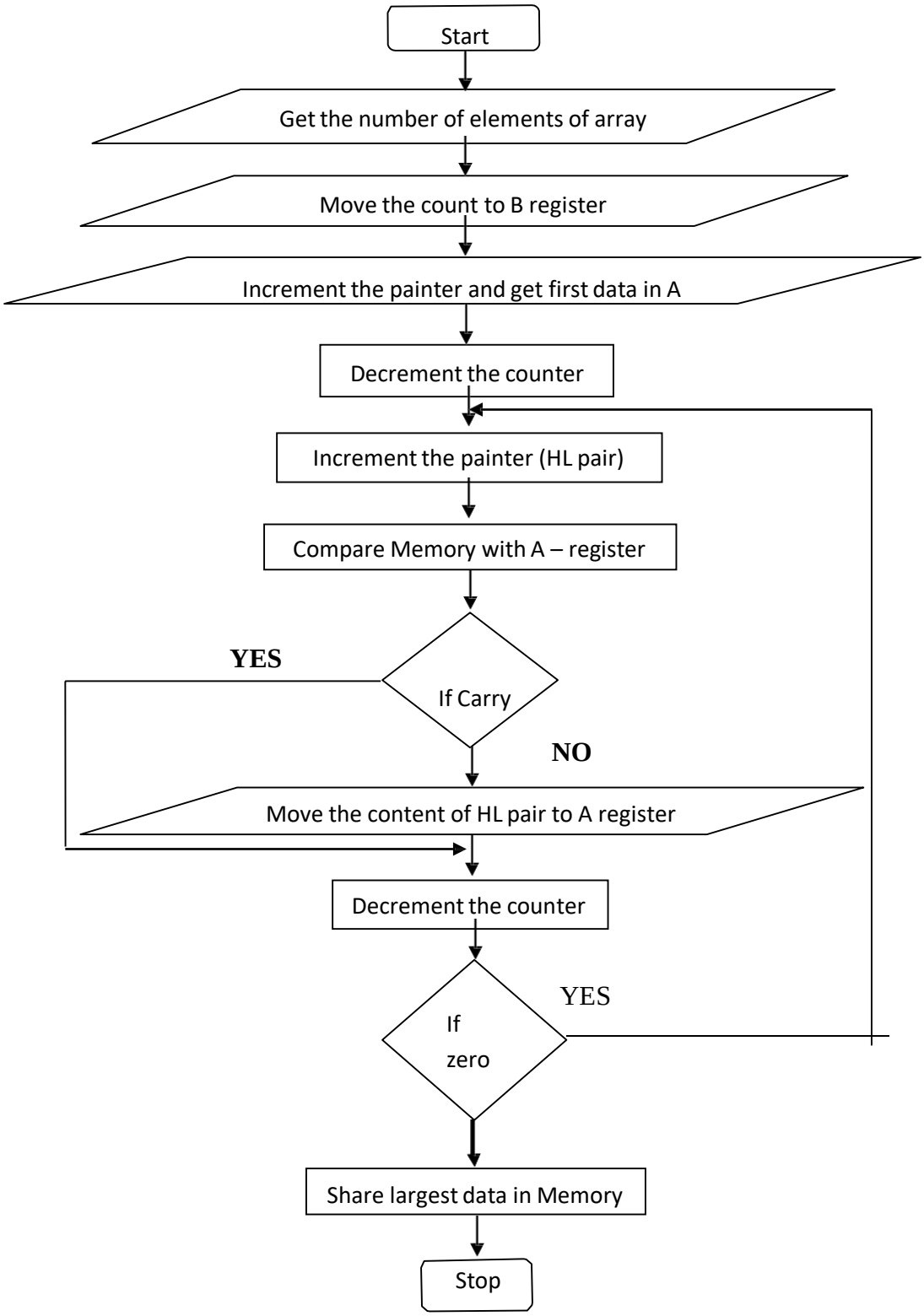
INPUT & OUTPUT TABULATION:

Memory Address	Input data	Memory Address	Output data
8200 (count)			
8201			
8202		8300	
8203			
8204			
8205			

8107		INX H	23	Increment HL Pair
8108		CMP M	BE	Compare the content of memory with accumulator
8109		JC AHEAD	DA	If CF=1, go to Label AHEAD, otherwise go to next step.
810A			OD	
810B			81	
810C		MOV A,M	7E	Set the new values at Large
810D	AHEAD	DCR B	05	Decrement the value of B
810E		JNZ LOOP	C2	Repeat the comparison till B= 0 (ie.ZF=1)
810F			07	
8110			81	
8111		STA 8300	32	Store the largest value in memory from accumulator.
8112			00	
8113			83	
8114		HLT	76	Stop the program.

RESULT:

FLOW CHART:


Start
Get the number of elements of array
Move the count to B register
Increment the painter and get first data in A
Decrement the counter
Increment the painter (HL pair)
Compare Memory with A – register
YES
If Carry
NO
Move the content of HL pair to A register
Decrement the counter
YES
If
zero
Share largest data in Memory
Stop

Ex. No.: 5 B

Date : LARGEST NUMBER IN AN ARRAY OF DATA

AIM:

To write and execute the program of largest in an array of data using 8085 microprocessor

APPARATUS REQUIRED:

- 8085 microprocessor kit
- OPcode sheet

ALGORITHM:

1. Load the address of the first element (count) of an array in HL pair.
2. Load the count and move it in to the B –register
3. Increment the HL pair as a pointer
4. Move the first data to A-register form memory which is pointed by HL pair.
5. Decrement the count (B reg.)
6. Increment the pointer (HL reg. pair)
7. Compare the content of memory addressed by HL pair with content of A-register
8. If carry =0 go for step 10 otherwise go to step-9
9. Move the content of memory addressed by HL pair to A-register
10. Decrement the count (B reg.)
11. Check for zero of the count if ZF=0 go to step 6 otherwise go to next step
12. Store the largest data in memory from Accumulator.

PROGRAM:

INPUT & OUTPUT TABULATION:

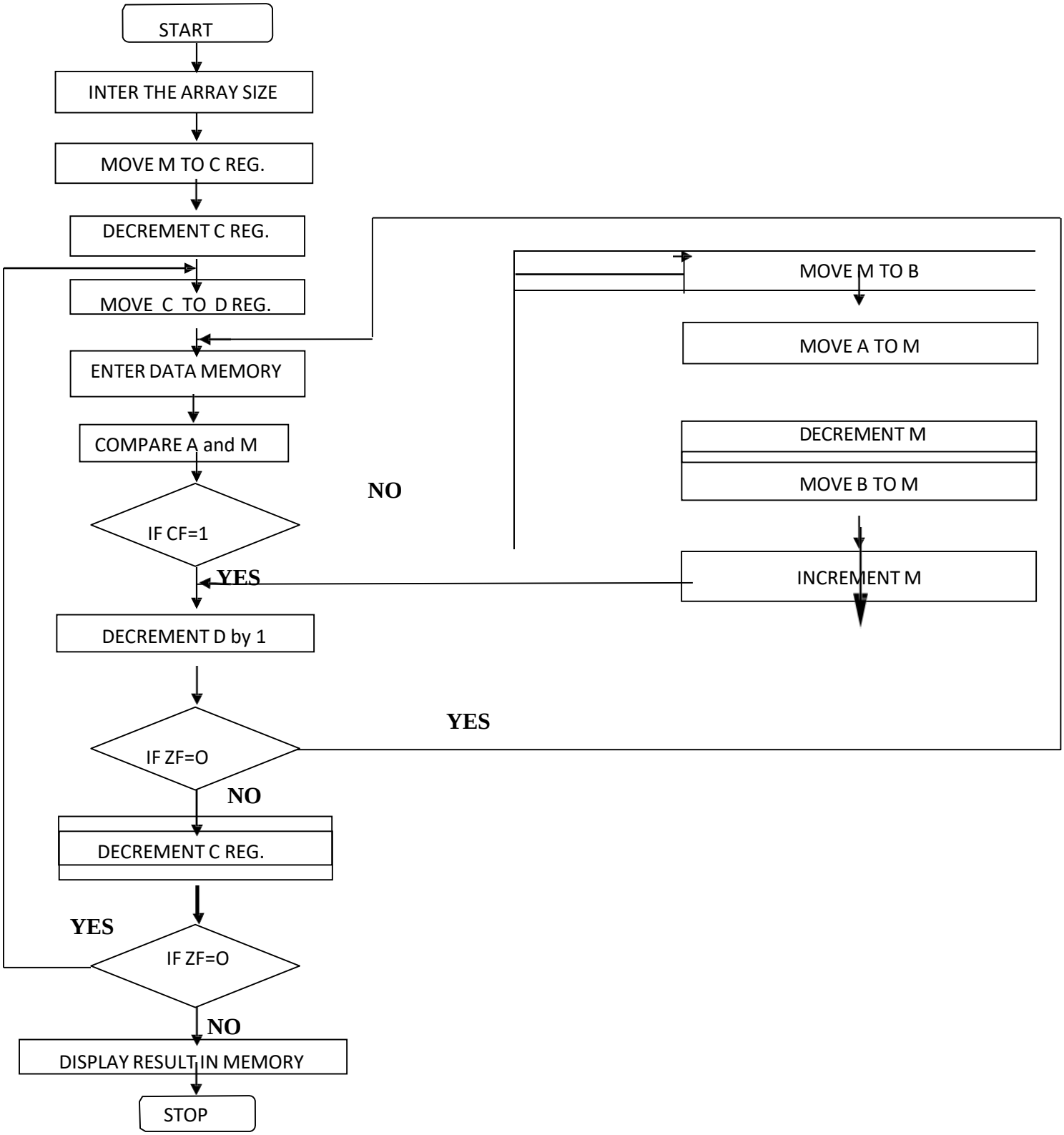
ADDRESS	LABEL	PNEUMONIC	OPCODE	COMMENTS
8100		LXI H, 8200	21	Set Pointers for array
8101			00	
8102			82	
8103		MOV B,M	46	Move the first data from (count) memory
8104		INX H	23	Increment the HL Pair
8105		MOV A,M	7E	Move the second data from memory to accumulator
8106		DCR B	05	Decrement the count

Memory Address	**Input data**	**Memory Address**	**Output data**
8200 (count) 8201 8202 8203 8204		8300	

8107	LOOP	INX H	23	Increment the HL pair
8108		CMP M	BE	Complements of memory with accumulator
8109		JNC AHEAD	D2	If A >M go to label AHEAD
810A			0D	
810B			81	
810C		MOV A,M	7E	Set the new values at large
810D	AHEAD	DCR B	05	Decrement the values of B
810E		JNZ LOOP	C2	Repeat the comparison till B = 0
810F			07	
8110			81	
8111		STA 8300	32	Store the largest value in memory from accumulator
8112			00	
8113			83	
8114		HLT	76	Stop the program

RESULT:

FLOW CHART:


START
INTER THE ARRAY SIZE
MOVE M TO C REG.
DECREMENT C REG.
MOVE C TO D REG.
ENTER DATA MEMORY
COMPARE A and M
IF CF=1
NO
YES
MOVE M TO B
MOVE A TO M
DECREMENT M
MOVE B TO M
INCREMENT M
DECREMENT D by 1
IF ZF=O
YES
NO
DECREMENT C REG.
YES
IF ZF=O
NO
DISPLAY RESULT IN MEMORY
STOP

Ex. No.: 6 A

Date : ARRANGE AN ARRAY OF DATA IN ASCENDING ORDER

AIM:

To write a program to arrange an array of data in ascending order by using 8085 microprocessor.

APPARATUS REQUIRED:

- 8085 microprocessor kit
- OPcode sheet

ALGORITHM:

1. Initialize the HL pair as memory pointer
2. Move the count to C-register
3. Decrement the count
4. Copy the count in D-register
5. Load the address of the element in HL pair
6. Move the first data in A- register from memory, which is pointed by HL pair.
7. Increment the HL pointer
8. Compare the content of the memory with Accumulator
9. If they are out of order exchange the contents of A register and memory
10. Decrement D-register content by 1
11. Repeat step 9 and10 till the value in D register becomes Zero
12. Decrement C-register content by 1
13. Repeat steps 4-12 till the value in C register becomes Zero

PROGRAM:

ADDRESS	LABEL	PNEMONICS	OPCODE	COMMENTS
8100	START	LXI H, 8200	21	Set the pointes for array
8101			00	
8102			82	
8103		MOV C, M	4E	Move the Count from Memory to C reg.
8104		DCR C	0D	Decrement the count (C reg.)
8105	REPEAT	MOV D, C	51	Move the data in C to D register
8106		LXI H, 8201	21	Load the Pointer to load next data
8107			01	
8108			82	

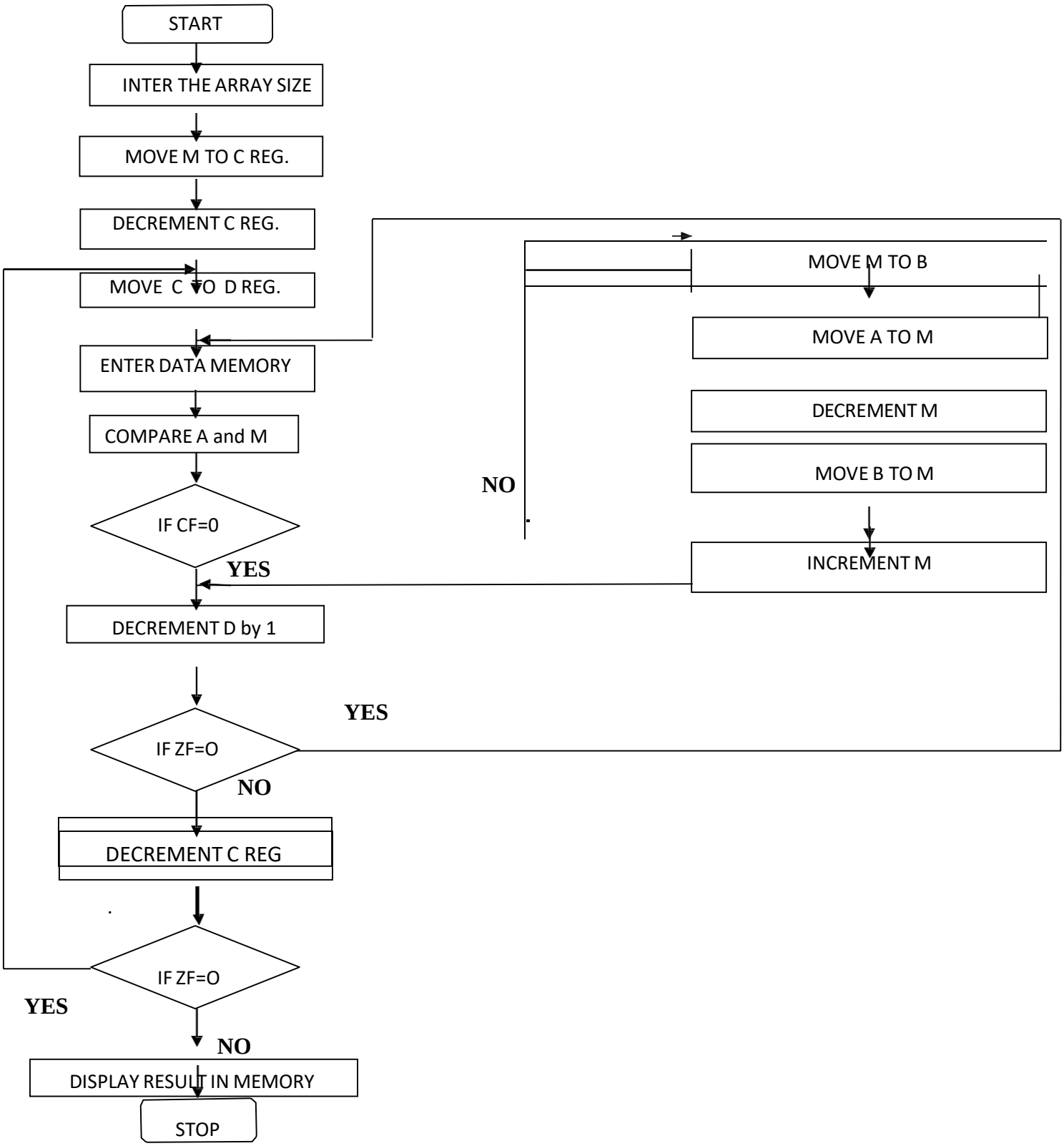
INPUT & OUTPUT TABULATION:

Memory Address	Input data	Memory Address	Output data
8200 (Count)			
8201		8201	
8202		8202	
8203		8203	
8204		8204	

8109	LOOP	MOV A ,M	7E	Set the new value of large
810A		INX H	23	Increment the HL pair.
810B		CMP M	BE	Compare the content of memory with Accumulator
810C		JC SKIP	DA	If CF=1, then go to SKIP label.
810D			14	
810E			81	
810F		MOV B,M	46	Exchange the contents of A register and memory
8110		MOV M,A	77	
8111		DCX H	2B	
8112		MOV M,B	70	
8113		INX H	23	
8114	SKIP	DCR D	15	Decrement the count(D reg.)
8115		JNZ LOOP	C2	Check for ZF, if ZF = 0 then go to LOOP label.
8116			09	
8117			81	
8118		DCR C	0D	Decrement the count
8119		JNZ REPEAT	C2	Check for ZF, if ZF = 0 then go to REPEAT label.
811A			05	
811B			81	
811C		HLT	76	Stop the program.

RESULT:

FLOW CHART:


START
INTER THE ARRAY SIZE
MOVE M TO C REG.
DECREMENT C REG.
MOVE C TO D REG.
ENTER DATA MEMORY
COMPARE A and M
IF CF=0
YES
NO
MOVE M TO B
MOVE A TO M
DECREMENT M
MOVE B TO M
INCREMENT M
DECREMENT D by 1
IF ZF=O
YES
NO
DECREMENT C REG
IF ZF=O
YES
NO
DISPLAY RESULT IN MEMORY
STOP

Ex. No.: 6 B

Date : ARRANGE AN ARRAY OF DATA IN DESCENDING ORDER

AIM:

To write a program to arrange an array of data in descending order by using 8085 microprocessor.

APPARATUS REQUIRED:

- 8085 microprocessor kit
- OPcode sheet

ALGORITHM:

1. Initialize the HL pair as memory pointer
2. Move the count to C-register
3. Decrement the count
4. Copy the count in D-register
5. Load the address of the element in HL pair
6. Move the first data in A- register from memory, which is pointed by HL pair.
7. Increment the HL pointer
8. Compare the content of the memory with Accumulator
9. If they are out of order exchange the contents of A register and memory
10. Decrement D-register content by 1
11. Repeat step 9 and10 till the value in D register becomes Zero
12. Decrement C-register content by 1
13. Repeat steps 4-12 till the value in C register becomes Zero

PROGRAM:

ADDRESS	LABEL	MNEMONICS	OPCODE	COMMENTS
8100	START	LXI H, 8200	21	Set the pointes for array
8101			00	
8102			82	
8103		MOV C, M	4E	Move the Count from Memory to C reg.
8104		DCR C	0D	Decrement the count (C reg.)
8105	REPEAT	MOV D, C	51	Move the data in C to D register
8106		LXI H, 8201	21	Load the Pointer to load next data
8107			01	
8108			82	

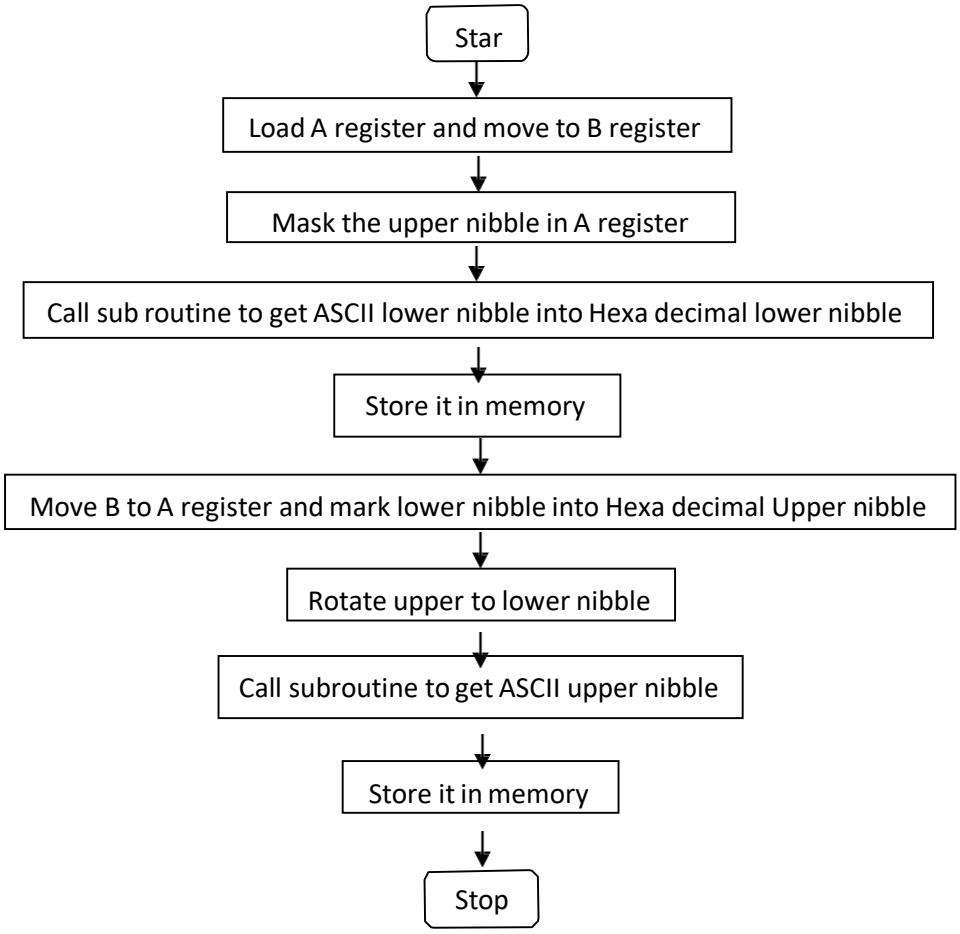
INPUT & OUTPUT TABULATION:

Memory Address	Input data	Memory Address	Output data
8200 (Count)			
8201		8201	
8202		8202	
8203		8203	
8204		8204	

8109	LOOP	MOV A ,M	7E	Set the new value of large
810A		INX H	23	Increment the HL pair.
810B		CMP M	BE	Compare the content of memory with Accumulator
810C		JNC SKIP	D2	If CF=0, then go to SKIP label.
810D			14	
810E			81	
810F		MOV B,M	46	Exchange the contents of A register and memory
8110		MOV M,A	77	
8111		DCX H	2B	
8112		MOV M,B	70	
8113		INX H	23	
8114	SKIP	DCR D	15	Decrement the count (D reg.)
8115		JNZ LOOP	C2	Check for ZF, if ZF = 0 then go to LOOP label.
8116			09	
8117			81	
8118		DCR C	0D	Decrement the count
8119		JNZ REPEAT	C2	Check for ZF, if ZF = 0 then go to REPEAT label.
811A			05	
811B			81	
811C		HLT	76	Stop the program.

RESULT:

FLOW CHART:


Star
Load A register and move to B register
Mask the upper nibble in A register
Call sub routine to get ASCII lower nibble into Hexa decimal lower nibble
Store it in memory
Move B to A register and mark lower nibble into Hexa decimal Upper nibble
Rotate upper to lower nibble
Call subroutine to get ASCII upper nibble
Store it in memory
Stop

Ex. No.: 7 A

Date : CODE CONVERSIONS - ASCII TO HEXAAIM:

To write and execute the program for convert ASCII to HEXA DECIMAL number using 8085 microprocessor.

APPARATUS REQUIRED:

- 8085 microprocessor kit
- OPcode sheet

ALGORITHM:

1. Load the given data in A register
2. Move the content of A to B register
3. Mask the upper nibble of the hexadecimal number in A register
4. Call suborning to get ASCII of lower nibble into hexadecimal lower nibble
5. Store it in memory
6. Move B register value to A- register and mask the lower nibble
7. Rotate the upper nibble to lower nibble position
8. Call subroutine to get ASCII of upper nibble in to hexadecimal
9. Store it in memory
10. Terminate the program

PROGRAM:

Address	Label	Mnemonics	Opcode	Comments
8100	START	LDA 8200	3A	Load accumulator
8101			00	
8102			82	
8103		MOV B,A	47	Move accumulator to B register
8104		ANI 0F	E6	Mask the Upper nibble
8105			0F	
8106		CALL SUB1	CD	Call suborning to get ASCII of lower nibble
8107			1A	
8108			81	
8109		STA 8201	32	Store ASCII of lower nibble in memory
810A			01	
810B			82	
810C		MOV A,B	78	Move B register to accumulator

Conversion Table for Hexadecimal, Decimal and ASCII

Hexa	Decimal	ASCII
30	48	0
31	49	1
32	50	2
33	51	3
34	52	4
35	53	5
36	54	6
37	55	7
38	56	8
39	57	9
41	65	A
42	66	B
43	67	C
44	68	A
45	69	B
46	70	C
47	71	A
48	72	B

Hexa	Decimal	ASCII
49	73	C
50	74	A
51	75	B
52	76	C
53	77	A
54	78	B
55	79	C
56	80	A
57	81	B
58	82	C
59	83	A
60	84	B
61	85	C
62	86	A
63	87	B
64	88	C
65	89	A
5A	90	B

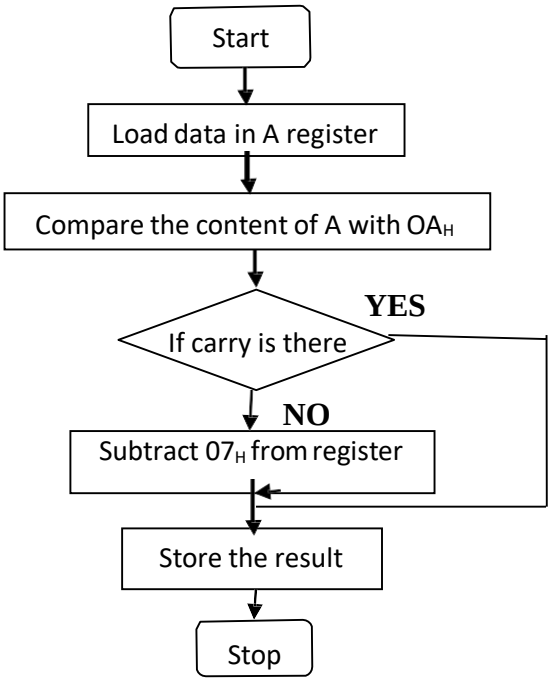
INPUT & OUTPUT TABULATION:

Memory address	Input data	Memory address	Output data
8200		8201 8202	

810D		ANI F0	E6	Mask the lower nibble
810E			F0	
810F		RLC	07	Rotate left through Carry
8110		RLC	07	
8111		RLC	07	
8112		RLC	07	
8113		CALL SUB1	CD	Call suborning to get ASCII of Upper nibble
8114			1A	
8115			81	
8116		STA 8202	32	Store ASCII of Upper nibble in memory
8117			02	
8118			82	
8119		HLT	76	Stop the program
811A		CPI 0A	FE	Compare A with immediate data
811B			0A	
811C		JC SKIP	DA	Jump on carry to SKIP label
811D			21	
811E			81	
811F		ADI 07	C6	Count the number , add accumulator with 07
8120			07	
8121	SKIP	ADI 30	C6	Add accumulator with immediate data
8122			30	
8123		RET	C9	Return to Main program

RESULT:

FLOW CHART


Start
Load data in A register
Compare the content of A with OA_H
If carry is there
YES
NO
Subtract 07_H from register
Store the result
Stop

Ex. No.: 7 B

Date : CODE CONVERSION - HEXA TO ASCII

AIM:

To convert given character (HEXA) in to its equivalent ASCII using 8085 microprocessor

APPARATUS REQUIRED:

- 8085 microprocessor kit
- OPcode sheet

ALGORITHM:

1. Load the given data in A-register
2. Subtract 30_H from A- register
3. Compose the content of A-register with OA_H
4. If $A<0A_H$ jump to step6, else proceed To next step
5. Subtract $O7_H$ from A-register
6. Store the result
7. Stop the program

PROGRAM:

Address	Label	Mnemonics	OPcode	Comments
8100	START	LDA 8200	3A	Load the input data into the accumulator
8101			00	
8102			82	
8103		SUI 30	D6	Subtract accumulator with immediate data
8104			30	
8105		CPI OA	FE	Compare A with immediate data.
8106			0A	
8107		JC SKIP	DA	Jump on carry to SKIP label
8108			0C	
8109			81	
810A		SUI 07	D6	Subtract accumulator with 07
810B			07	

Conversion Table for Hexadecimal, Decimal and ASCII

Hexa	Decimal	ASCII
30	48	0
31	49	1
32	50	2
33	51	3
34	52	4
35	53	5
36	54	6
37	55	7
38	56	8
39	57	9
41	65	A
42	66	B
43	67	C
44	68	A
45	69	B
46	70	C
47	71	A
48	72	B

Hexa	Decimal	ASCII
49	73	C
50	74	A
51	75	B
52	76	C
53	77	A
54	78	B
55	79	C
56	80	A
57	81	B
58	82	C
59	83	A
60	84	B
61	85	C
62	86	A
63	87	B
64	88	C
65	89	A
5A	90	B

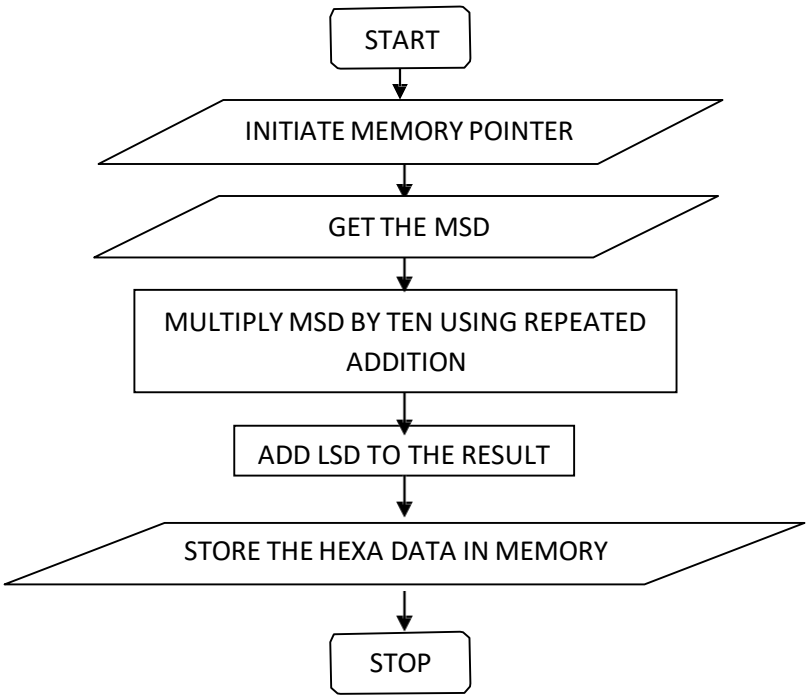
INPUT & OUTPUT TABULATION:

Memory Address	Input data	Memory Address	Output data
8200		8201	

810C	SKIP	STA 8201	32	Store the result in memory from accumulator
810D			01	
810E			82	
810F		HLT	76	Stop the Program

RESULT:

FLOW CHART:


START
INITIATE MEMORY POINTER
GET THE MSD
MULTIPLY MSD BY TEN USING REPEATED ADDITION
ADD LSD TO THE RESULT
STORE THE HEXA DATA IN MEMORY
STOP

Ex. No.: 8 A

Date : **CODE CONVERSION - BCD TO HEXA**

AIM:

To convert two BCD numbers in memory to its equivalent HEXA number using 8085 microprocessor.

APPARATUS REQUIRED:

- 8085 microprocessor kit
- OPcode sheet

ALGORITHM:

1. Initialize memory pointer to 8100H.
2. Load the most significant digit (MSD).
3. Multiply the MSD by ten using repeated addition.
4. Add the least significant digit (LSD) to the result obtained in previous step.
5. Store the HEXA data in memory.

PROGRAM:

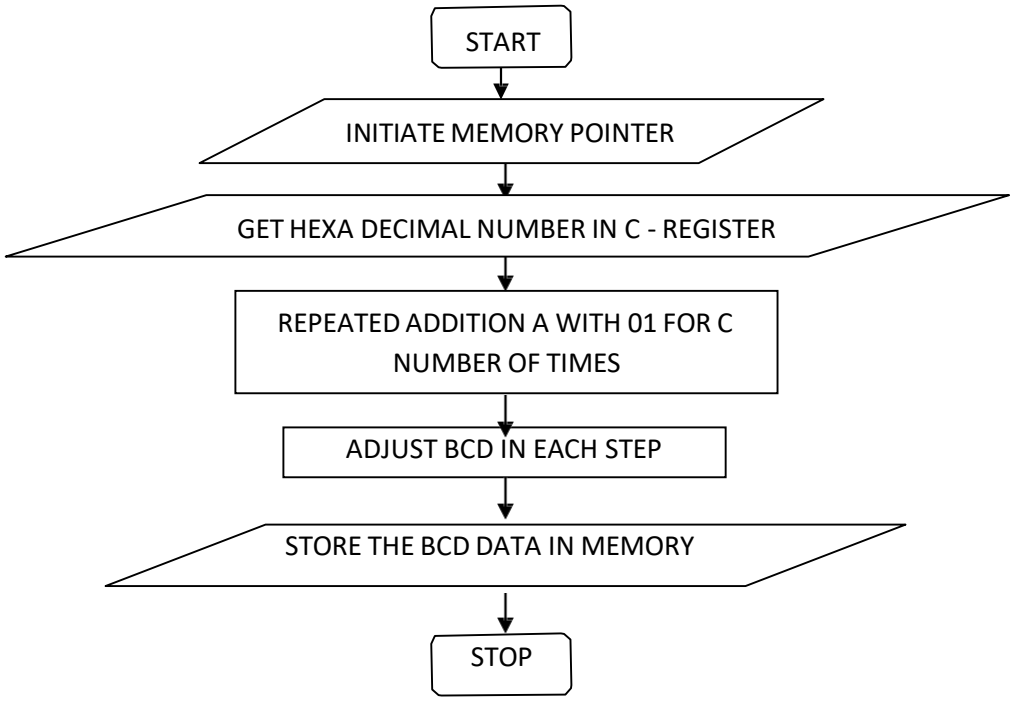
Address	Label	Pneumonic	Opcode	Comments
8100	START	LXI H,8150	21	Load the input data into the accumulator
8101			50	
8102			81	
8103		MOVA,M	7E	Move memory to accumulator
8104		ADD A	87	Add accumulator content with Accumulator. Ie) MSD*2
8105		MOV B,A	47	Move Accumulator content to B
8106		ADD A	87	MSD is multiplied by 4
8107		ADD A	87	MSD is multiplied by 8
8108		ADD B	80	Add accumulator content with B reg.
8109		INX H	23	Increment the memory
810A		ADD M	86	Add Accumulator and memory
810B		INX H	23	Increment the memory
810C		MOV M,A	77	Move the accumulator to memory for result
810D		HLT	76	Stop the program

INPUT & OUTPUT TABULATION:

Memory Address	Input data	Memory Address	Output data
8150 8151		8152	

RESULT:

FLOW CHART:


START
INITIATE MEMORY POINTER
GET HEXA DECIMAL NUMBER IN C - REGISTER
REPEATED ADDITION A WITH 01 FOR C NUMBER OF TIMES
ADJUST BCD IN EACH STEP
STORE THE BCD DATA IN MEMORY
STOP

Ex. No.: 8 B

Date : CODE CONVERSION - HEXA TO BCD

AIM:

To convert given HEXA decimal number into its equivalent BCD number using 8085 microprocessor.

APPARATUS REQUIRED:

- 8085 microprocessor kit
- OPcode sheet

ALGORITHM:

1. Initialize memory pointer to 8100H
2. Get the hexadecimal number in C register
3. Perform repeated addition for C number of times
4. Adjust for BCD in each step
5. Store the BCD data in memory

PROGRAM:

Address	Label	Pneumonic	Opcode	Comments
8100	START	LXI H, 8150	21	Initialize memory pointer for input
8101			50	
8102			81	
8103		MVI D,00	16	Clear D register for most significant byte
8104			00	
8105		XRA A	AF	Clear accumulator
8106		MOV C,M	4E	Get Hexadecimal input data from memory
8107	LOOP2	ADI 01	C6	Count the number One by one adjust BCD count
8108			01	
8109		DAA	27	Adjust accumulator for BCD
810A		JNC LOOP1	D2	Jump on no carry to Loop1
810B			0E	
810C			81	
810D		INR D	14	Increment D register
810E	LOOP1	DCR C	OD	Decrement C register

INPUT & OUTPUT TABULATION:

Memory Address	**Input data**	**Memory Address**	**Output data**
8150		8151 8152	

810F		JNZ LOOP2	C2	Jump on no zero to Loop2
8110			07	
8111			81	
8112		STA 8151	32	Store the least Significant byte in memory
8113			51	
8114			81	
8115		MOV A,D	7A	Move D to accumulator
8116		STA 8152	32	Store the most Significant byte in memory
8117			52	
8118			81	
8119		HLT	76	Termite ate the program

RESULT:

Ex. No.: 9 A

Date : **ADDITION OF TWO 8-BIT DATA**

AIM:

To perform the arithmetic operation addition by using 8051 microcontroller.

APPARATUS REQUIRED:

- 8051 microcontroller kit
- OPcode sheet

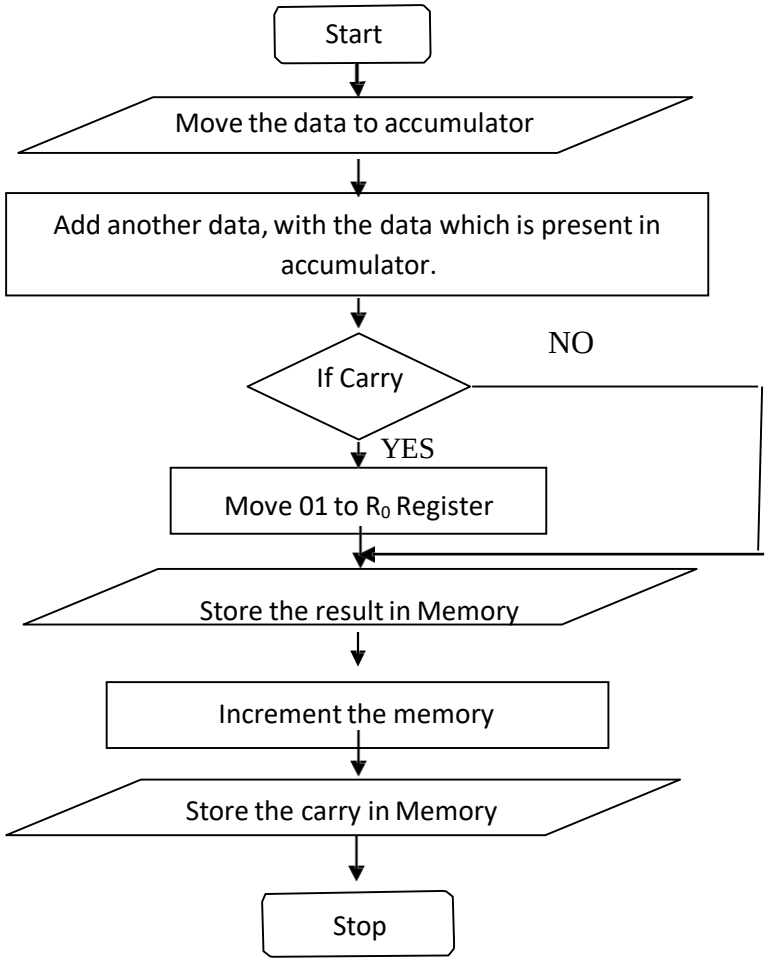
ALGORITHM:

1. Start the program
2. Get the Input data at the accumulator.
3. Add the adder data with the data which is already in accumulator.
4. Move the result to 8500 memory location.
5. If any carry is available then move 01 to R_0 Register
6. Store the result in Memory
7. Stop the program

PROGRAM:

Address	Label	Pneumonic	OPcode	Comments
8100	START	MOVA,09	74	Move 09 to accumulator
8101			09	
8102		ADD A,04	24	Add 04 with accumulator
8103			04	
8104		JNC L00P	50	On No carry Jump to 100p
8105			02	
8106		MOV R_0 , 01	78	Move R_0 register into 01 for carry
8107			01	

FLOW CHART:


Start
Move the data to accumulator
Add another data, with the data which is present in accumulator.
If Carry
NO
YES
Move 01 to R_0 Register
Store the result in Memory
Increment the memory
Store the carry in Memory
Stop

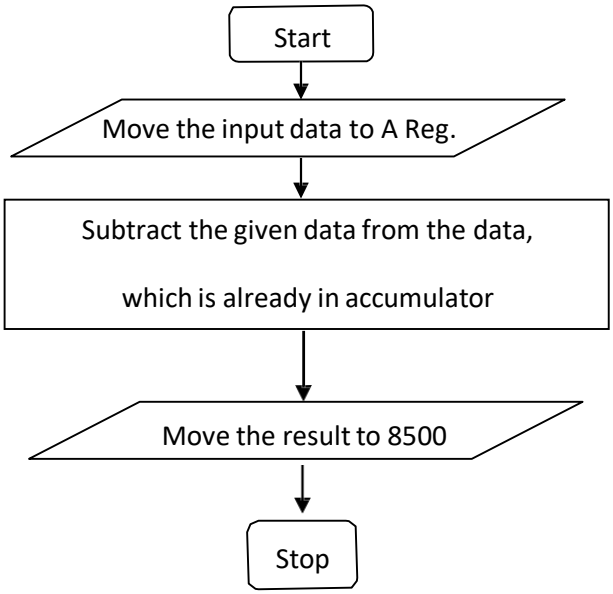
INPUT & OUTPUT TABULATION:

Memory Address	Input data	Memory address	Output data
8101		8500	
8103			
8101		8500	
8103			

8108	LOOP	MOV DPTP,#8500	90	Move the memory address to DPTR
8109			85	
810A			00	
810B		MOVX @DPTR , A	FO	Store the sum in memory
810C		INC DPTR	A3	Increment DPTR
810D		MOV A,R_O	E8	Move R_O to A
810E		MOVX @DPTR , A	FO	Store the Carry in memory
810F	LOOP1	SJMP LOOP1	80	Stop the program
8110			FE	

RESULT:

FLOW CHART:



INPUT & OUTPUT TABULATION:

Memory Address	Input data	Memory address	Output data
8101		8500	
8103			
8101		8500	
8103			

Ex. No.: 9 B

Date : SUBTRACTION OF TWO 8-BIT DATA

AIM:

To subtract the two given number by using 8051 micro controller.

APPARATUS REQUIRED:

- 8051 microcontroller kit
- OPcode sheet

ALGORITHM:

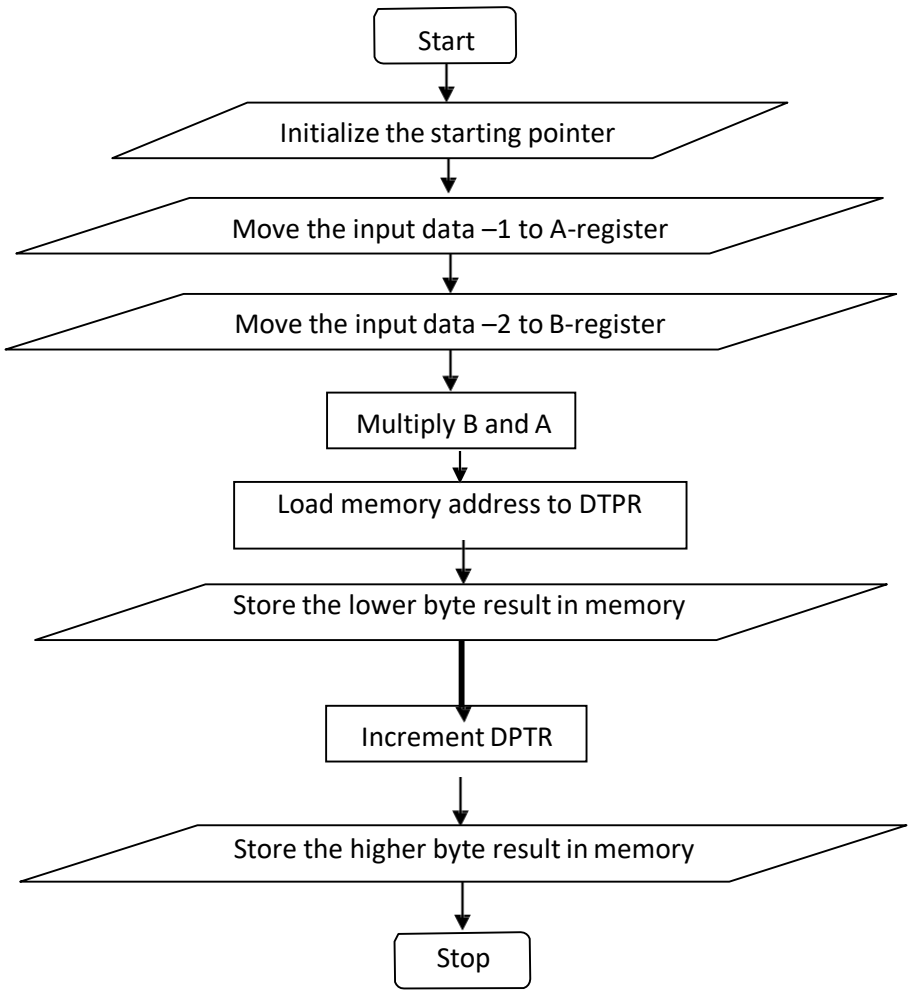
1. Start the program.
2. Get data to the accumulator.
3. Subtract another data from the data which is already stored in the accumulator.
4. Move the result to the memory 8500.
5. If any carry, than store it in accumulator.
6. Stop the program.

PROGRAM:

Address	Label	Pneumonic	OPcode	Comments
8100	START	MOV A ,09	74	Move the data 09 to the accumulator
8101			09	
8102		SUBB A ,04	94	Subtract the data 04 with data in accumulator
8103			04	
8104		MOV DPTR, #8500	90	store the result in 8500
8105			85	
8106			00	
8107		MOV @DPTR , A	FO	Carry is stored
8108	LOOP	STMP LOOP	80	Short jump
8109			FE	Stop the program

RESULT:

FLOW CHART:



INPUT& OUTPUT TABULATION:

MEMORY ADDRESS	INPUT DATA	MEMORY ADDRESS	OUTPUT DATA
8101 8104		8500 8501	
8101 8104		8500 8501	

Ex. No.: 10 A

Date : **MULTIPLICATION OF TWO 8-BIT DATA**

AIM:

To perform 8-bit multiplication by using 8051 microcontroller

APPARATUS REQUIRED:

- 8051 microcontroller kit
- OPcode sheet

ALGORITHM:

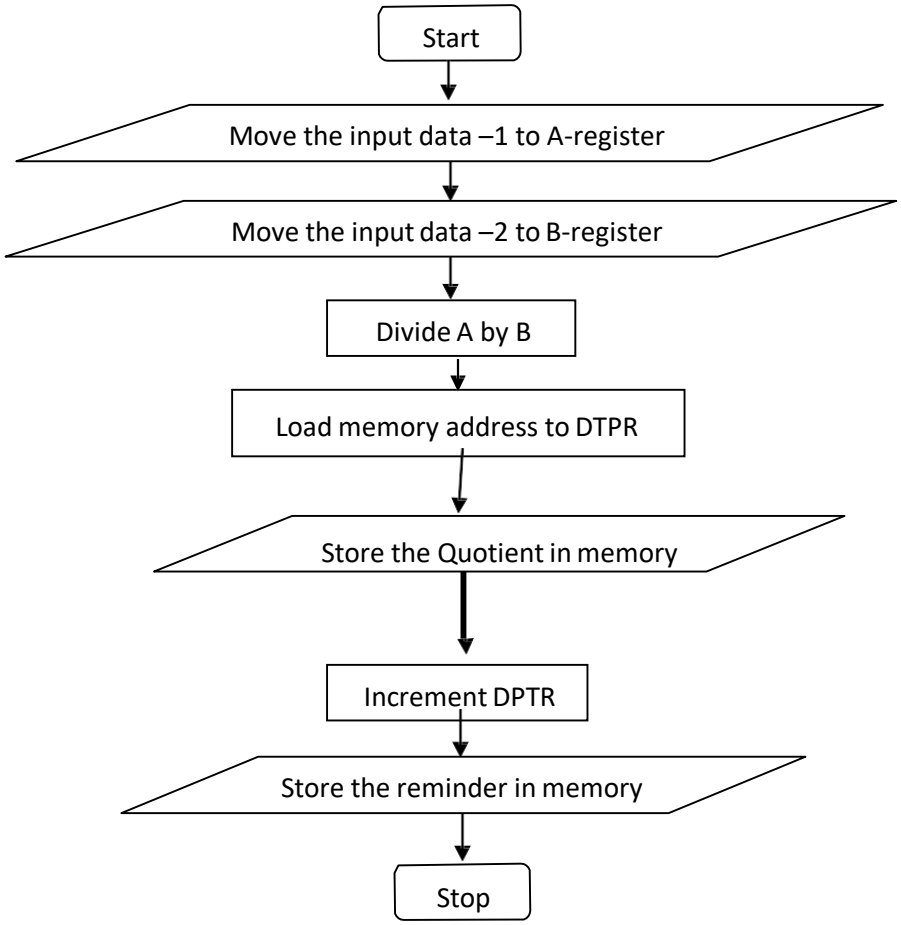
1. Start the program.
2. Initialize the starting pointer.
3. Move the input data 1 to accumulator
4. Move the input data 2 to the B-register
5. Multiply both B and A
6. Load the memory address to DPTR and store the lower byte result.
7. Increment DPTR and move B to A.
8. Store the higher byte result
9. Stop the program

PROGRAM:

ADDRESS	LABEL	PNEUMONIC	OPCODE	COMMENTS
8100	START	MOV A , #02	74	Move the input data 1 to accumulator
8101			02	
8102		MOV B , #03	75	Move the input data 2 to the B-register
8103			F0	
8104			03	
8105		MUL A,B	A4	Multiply the input data's ie) A and B
8106		MOV DPTR , #8500	90	Load the memory address to DPTR
8107			85	
8108			00	
8109		MOVX @DPTR , A	F0	store the lower byte result
810 A		INC DPTR	A3	Increment DPTR
810 B		MOV A,B	F5	Move B to A register
810 C			F0	
810 D		MOVX @DPTR , A	F0	Store the higher byte result
810 E	LOOP	SJMP LOOP	80	Stop the program
810F			FE	

RESULT:

FLOW CHART:



INPUT & OUTPUT TABULATION:

MEMORY ADDRESS	INPUT DATA	MEMORY ADDRESS	OUTPUT DATA
8101 8104		8500 8501	
8101 8104		8500 8501	

Ex. No.: 10 B

Date : DIVISION OF TWO 8-BIT DATA

AIM:

To performs the 8-bit division using 8051 microcontroller

APPARATUS REQUIRED:

- 8051 microcontroller kit
- OPcode sheet

ALGORITHM:

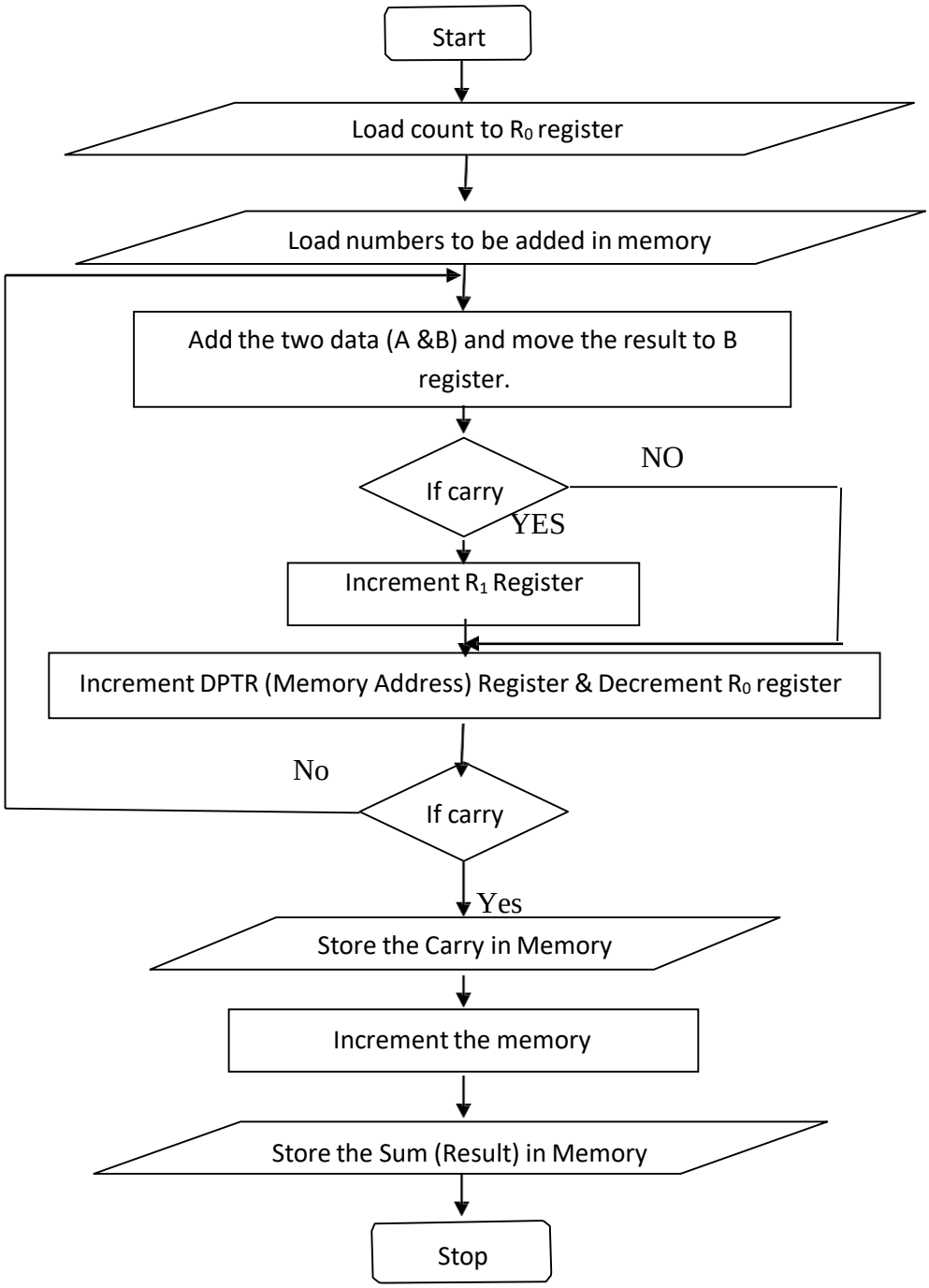
1. Start the program.
2. Initialize the starting pointer.
3. Move the input data 1 to accumulator
4. Move the input data 2 to the B-register
5. Divide the content of A by B
6. Load the memory address to DPTR and store the Quotient in memory.
7. Increment DPTR and move B to A.
8. Store the Reminder in memory.
9. Stop the program

DIVISION ON USING 8051 PROGRAM:

ADDRESS	LABLE	PREMONICS	OPCODE	COMMENTS
8100	START	MOV A,#05	74	Move the input data 1 toaccumulator
8101			05	
8102		MOV B, #03	75	Move the input data 2 to the B-register
8103			FO	
8104			03	
8105		DIV A,B	84	Divide the content of A,B
8106		MOV DPTR,8150	90	Load the memory address to DPTR
8107			81	
8108			50	
8109		MOVX @DPTR , A	FO	Store the Quotient in memory
810A		INC DPTR	A3	Increment DPTR
810B		MOV A , B	E5	Move B to A register
810C			F0	
810D		MOV X @DPTR , A	F0	Store the reminder in memory
810E	LOOP	SJMP LOOP	80	Stop the program
810F			FE	

RESULT:

FLOW CHART:


Start
Load count to R_0 register
Load numbers to be added in memory
Add the two data (A &B) and move the result to B register.
If carry
NO
YES
Increment R_1 Register
Increment DPTR (Memory Address) Register & Decrement R_0 register
No
If carry
Yes
Store the Carry in Memory
Increment the memory
Store the Sum (Result) in Memory
Stop

Ex. No.: 11 A

Date : **SUM OF THE ELEMENTS**

AIM:

To perform the sum of the numbers by using 8051 microcontroller.

APPARATUS REQUIRED:

- 8051 microcontroller kit
- OPcode sheet

ALGORITHM:

1. Start the program
2. Get the Count (Ro) & Input data in memory.
3. Add the two data and move the result to B register..
4. If Carry is present, increment R_1Register, otherwise go to next step.
5. Increment DPTR register for next data.
6. Decrement the R_0 register for count.
7. If Zero flag is not set go to step 3, otherwise go to next step.
8. Store the result in Memory
7. Stop the program

PROGRAM:

Address	Label	Pneumonic	OPcode	Comments
8100		MOV DPTR,#8200	90	Move Memory address to DPTR
8101			82	
8102			00	
8103		MOVX A, @DPTR	E0	Move the first data to acc and then R_0 for count
8104		MOV R_0,A	F8	

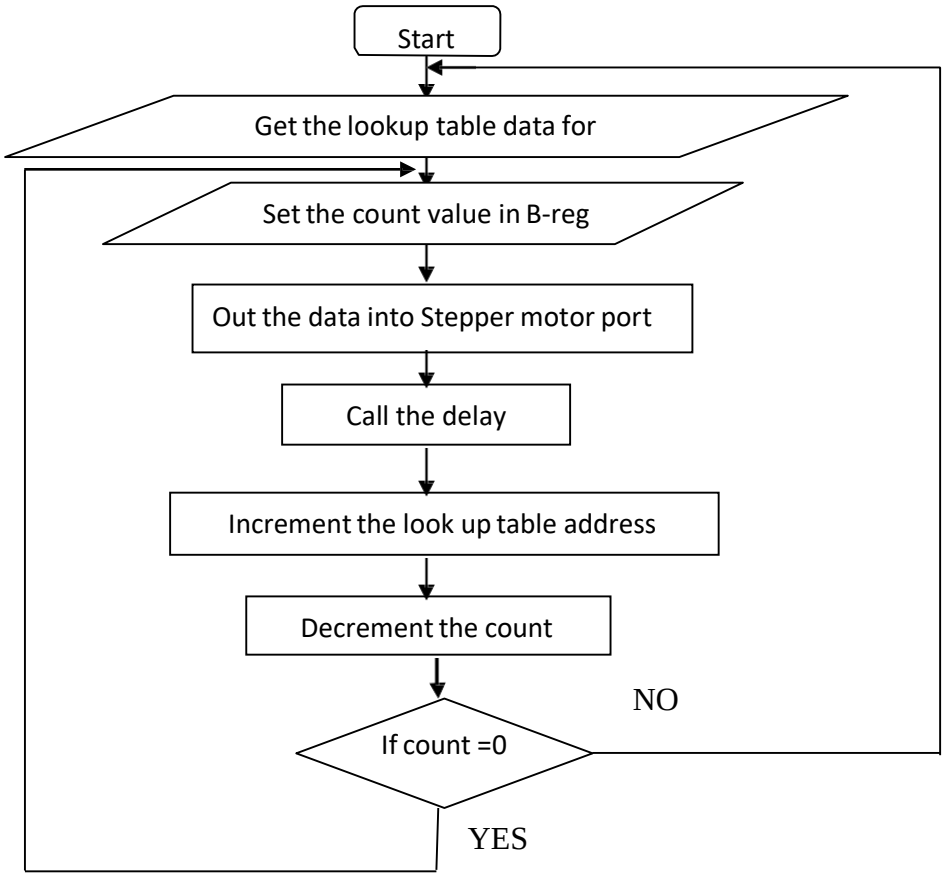
INPUT & OUTPUT TABULATION:

Memory Address	Input data	Memory address	Output data
8200		8500	
8201			
8202		8501	
8203			
8204			
8205			

8105		MOV B, #00	75	Clear B register
8106			F0	
8107			00	
8108		MOV R_1, B	A9	Move B register content to R_1
8109			F0	
810A		INC DPTR	A3	Increment Memory address
810B	LOOP1	MOVX A,@DPTR	E0	Move the data from memory to Acc.
810C		ADD A,B	25	Add A & B registers contents.
810D			F0	
810E		MOV B,A	F5	Move the result from A to b register.
810F			F0	
8110		JNC LOOP	50	Jump no carry then LOOP label
8111			01	
8112		INC R_1	09	Increment R_1 for carry
8113	LOOP	INC DPTR	A3	Increment Memory address
8114		DJNZ R_0, LOOP1	D8	Decrement R_0 (Count) value and if R0≠ 0 then jump to LOOP1 label.
8115			F5	
8116		MOV DPTP,#8500	90	Move the memory address to DPTR for Result
8117			85	
8118			00	
8119		MOV A,R_1	E9	Move R_1 to A
811A		MOVX @DPTR , A	F0	Store the Carry in memory
811B		INC DPTR	A3	Increment DPTR
811C		MOV A,B	E5	Move B to A
811D			F0	
811E		MOVX @DPTR , A	F0	Store the Sum in memory
811F	LOOP1	SJMP LOOP1	80	Stop the program
8120			FE	

RESULT:

FLOW CHART:


Start
Get the lookup table data for
Set the count value in B-reg
Out the data into Stepper motor port
Call the delay
Increment the look up table address
Decrement the count
NO
If count =0
YES

Ex. No.: 11 B

Date : **STEPPER MOTOR INTERFACE USING 8051 MICROCONTROLLER**

AIM:

To run stepper a motor at desired speed in two directions using 8051 microcontroller.

APPARATUS REQUIRED:

- 8051 Microcontroller kit
- OPcode sheet
- Stepper motor interface

THEORY:

A motor in which the rotor is able to assume only discrete stationery angular position is a stepper motor. The rotary motion occurs in a stepwise manner from an equilibrium position to the next. Stepper motor are widely used in (simple position control systems in the open closed loop mode)a verity of application such as complete peripherals (printers, disk drive etc)and in the areas of process control machine tools, medicine numerically controller machine robotics.

ALGORITHM:

1. Load the stepping sequence number
2. Then load the motor port addressing 8015 memory
3. Move stepping data into accumulator
4. Out the accumulator value in to the stepper motor
5. Call the delay
6. Increment the DPTR (memory address)
7. Repeat the processor for all stepping data
8. Jump to step 1and repeat all steps

PROGRAM:

Address	Label	Pneumonic	OPcode	Comments
8100	START	MOV B, #04	75	Move total no of stepping data to B register
8101			F0	
8102			04	
8103		MOV $R_{O,}$#82	78	Move starting address of stepping sentence to R_0,R_1
8104			82	
8105		MOV R_1, #00	79	
8106			00	

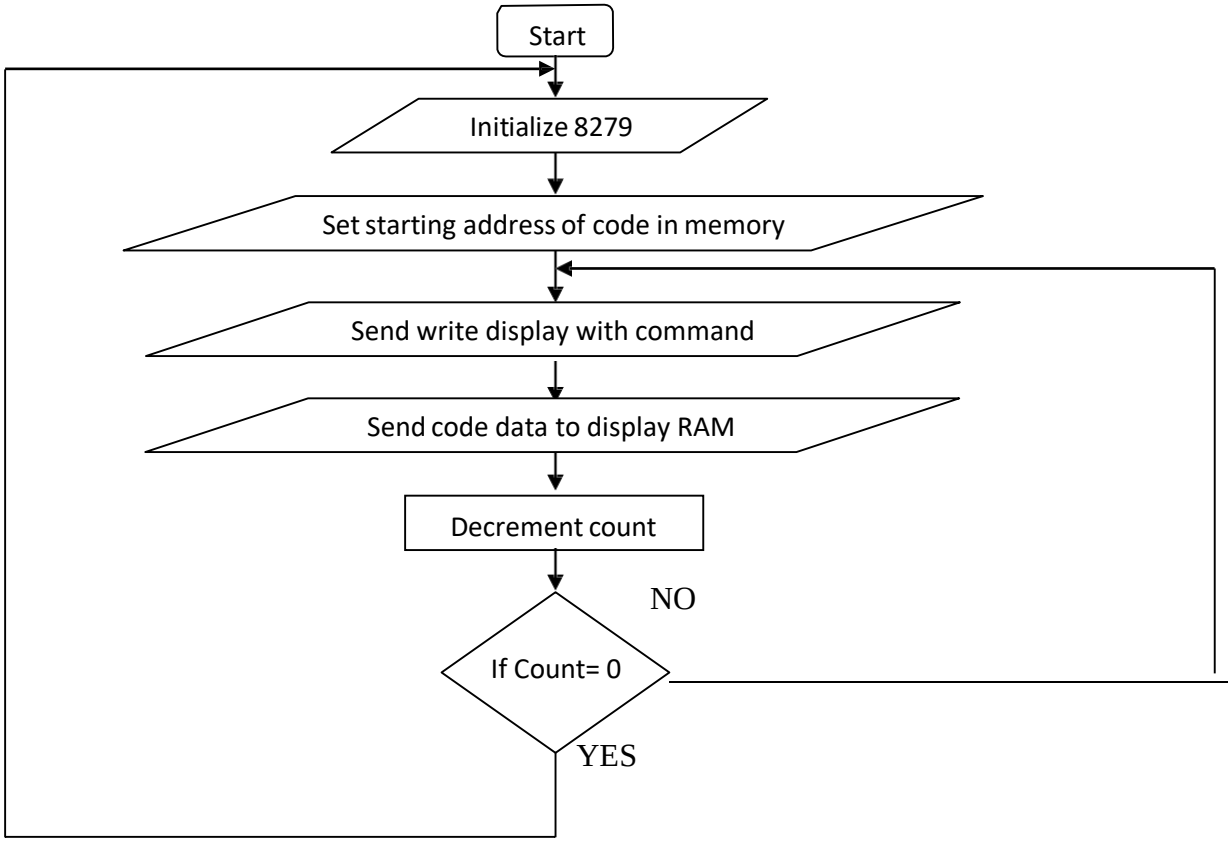
WAVE SCHEME (UNIPOLAR OPERATION)

ADDERSS	ANTI CLOCKWISE	CLOCKWISE
8200	08	02
8201	01	04
8202	04	01
8203	02	08

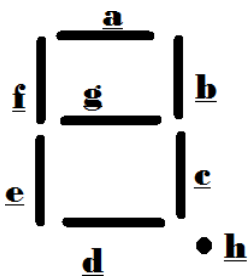
8107		MOV DPTR, #E0C0	90	Motor port address in DPTR
8108			E0	
8109			C0	
810A	LOOP	MOV DP_H , R_O	88	Save data R_0,R_1 in data
810B			83	
810C		MOV DP_L , R_1	89	
810D			82	
810E		MOV A , @DPTR	E0	Move stepping data to accumulator
810F		INC DPTR	A3	Increment DPTR
810O		MOV R_0 , DP_H	A8	Save data R_0,R_1 in DPTR
8111			83	
8112		MOV R_1 , DP_L	A9	
8113			82	
8114		MOV DPTR ,#E0C0	90	Motor port address in DPTR
8115			E0	
8116			C0	
8117		MOV X @DPTR , A	F0	Move data in accumulator to DPTR
8118		CALL DELAY	12	Call delay routine
8119			81	
811 A			21	
811B		DJNZ B , LOOP	D5	Decrement B and jump to loop (810A)if B#O
811C			F0	
811D			EC	
811E		JMP START	02	Jump to start (8100)
811F			81	
8120			00	
8121	DELAY	MOV R_2 , #12	7A	MOV data to r register
8122			12	
8123	DLY 1	MOV R_3, #FF	7	Move data to register
8124			FF	
8125	DLY 2	DJNZ R_3,DLY2	DB	Decrement R_3,and to DY 218125 if R_3,#0
8126			FE	
8127		DJNZ R_2,DLY1	DA	Decrement R AND jump to DLYLL 8123 Y R_2#0
8128			FA	
8129		RET	22	Return to main program

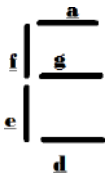
RESULT:

FLOW CHART:



SEVEN SEGMENT DISPLAY



For Example	**d**	**c**	**b**	**a**	**h**	**g**	**f**	**e**	
	**0**	**1**	**1**	**0**	**1**	**0**	**0**	**0**	**-- 68H**

Ex. No.: 12

Date : **INTERFACING 8279 WITH 8085 MICROPROCESSOR (ROLLING DISPLAY)**

AIM:

To interface 8279 programmable keyboard display controller with 8085 microprocessor and write and execute the assembly language program to roll the word to display.

APPARATUS REQUIRED:

- 8085 microprocessor kit
- 8279 keyboard display
- OPcode sheet

ALGORITHM:

1. Start the program by initializing memory pointer
2. Initialize 8279 keyboard display controller
3. Set mode and display in 8279 IC
4. Clear display in 8279 keyboard display controller
5. Write display & Read FIFO status
6. Write display RAM from location auto-increases of mode
7. Move data input from Memory to Accumulator
8. Send code data to display Ram and call delay subroutine
9. Decrement the counter and repeat the steps from 5 to 9 until the counter becomes zero.
10. Stop the program.

PROGRAM:

ADDRESS	LABEL	PNEUMONIC	OPCODE	COMMENT
8100	START	LXI H, 8150	21	Set pointer to memory
8101			50	
8102			81	
8103		MVI D, 1C	16	Initialize counter in D register
8104			1C	
8105		MVI A,10	3E	Set mode and Display in 8279 IC
8106			10	
8107		OUT C2	D3	Clear the display
8108			C2	

INPUT & OUTPUT TABULATION:

INPUT:

INPUT ADDRESS	INPUT DATA
8150	
8151	
8152	
8153	
8154	
8155	
8156	
8157	
8158	
8159	
815A	
815B	
815C	
815D	
815E	
815F	

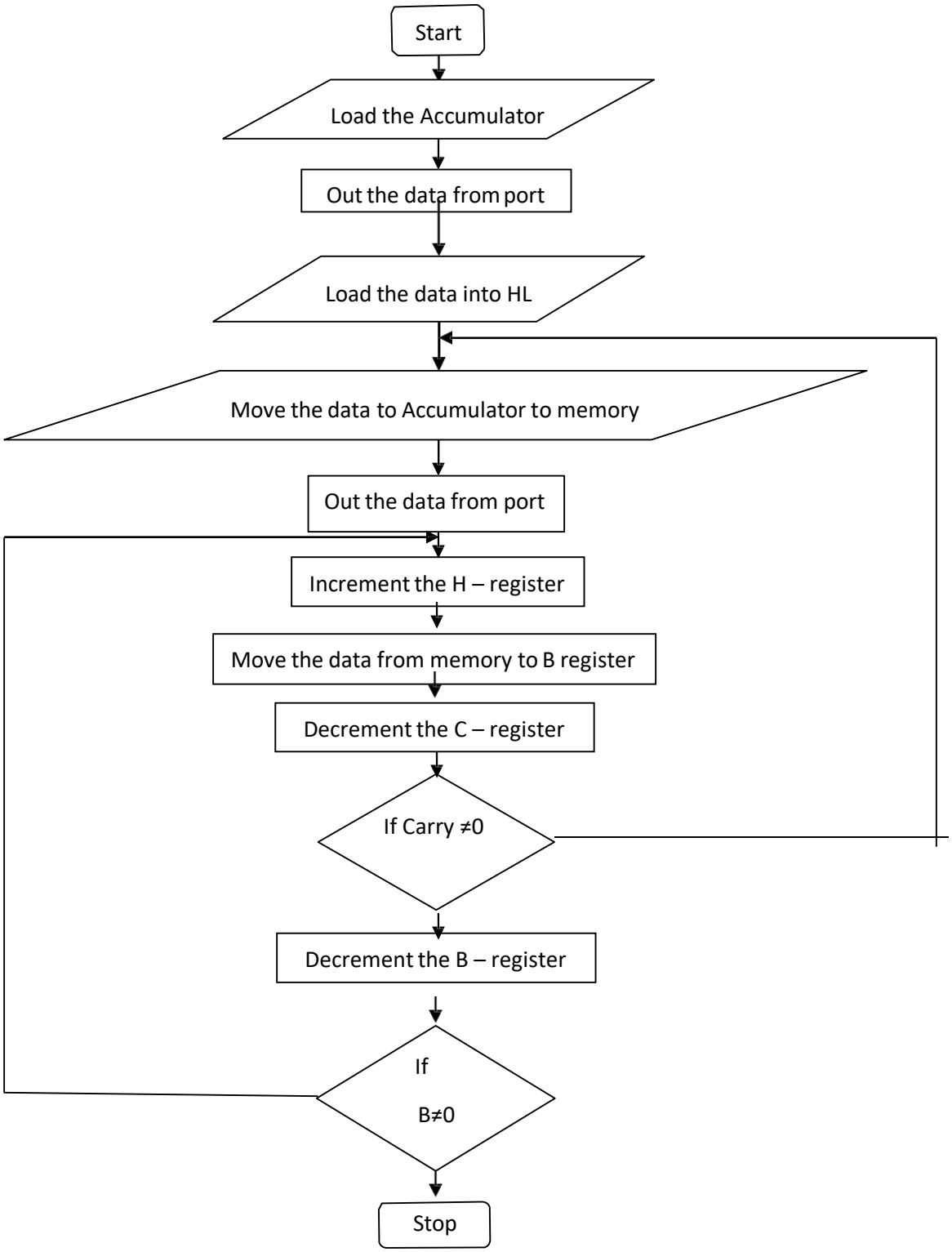
OUTPUT

8109		MVI A, CC	3E	
810A			CC	
810B		OUT C2	D3	
810C			C2	
810D		MVI A, 90	3E	Write display
810E			90	
810F		OUT C2	D3	
8110			C2	
8111	LOOP	MOV A,M	7E	
8112		OUT C0	D3	
8113			C0	
8114		CALL DELAY	CD	Call delay Subroutine
8115			1F	

8116			81	
8117		INX H	23	Increment the memory pointer
8118		DCR D	15	Decrement counter
8119		JNZ LOOP	C2	Jump if no zero to loop
811A			11	
811B			81	
811C		JMP START	C3	For Rolling the Display
811D			00	
811E			81	
811F	DELAY	MVI B, A0	06	Delay Subroutine
8120			A0	
8121	LOOP1	MVI C,FF	0E	
8122			FF	
8123	LOOP2	DCR C	0D	
8124		JNZ LOOP2	C2	
8125			23	
8126			81	
8127		DCR B	05	
8128		JNZ LOOP1	C2	
8129			21	
812A			81	
812B		RET	C9	

RESULT:

FLOW CHART:


Start
Load the Accumulator
Out the data from port
Load the data into HL
Move the data to Accumulator to memory
Out the data from port
Increment the H – register
Move the data from memory to B register
Decrement the C – register
If Carry ≠0
Decrement the B – register
If
B≠0
Stop

Ex. No.: 13

Date : **TRAFFIC LIGHT CONTROL SYSTEM USING 8085 MICROPROCESSOR**

AIM:

To perform the traffic light controlling using 8085 microprocessor.

APPARATUS REQUIRED:

- 8085 microprocessor kit
- OPcode sheet
- Traffic light control interface board.

ALGORITHM:

1. Start the program
2. Move the data to accumulator
3. Output data from port
4. Load HL register pair with immediate data
5. Move the data content to C register
6. Move data from memory to A
7. Output data from part A
8. Increment the HL register pair & Move data from memory to A
9. Output data from port B and port C one by one
10. Move data from memory to B register
11. Call delay for some time and increment HL register
12. Decrements C registers if ZF is not set go to step 6 else repeat the whole process
13. Perform OR operation with A and D register
14. If ZF=0, call delay else decrement B register until ZF=1.
15. Stop the program

TRAFFIC LIGHT CONTROL SYSTEM THE TRAFFIC LIGHT CONTROLLER WORKS IN FOLLOWING SEQUENCE:

- Provide green signal for road 1, green signal for pedestrian on road 4, red signal for other roads and other pedestrian 6secs
- Put Yellow signal for road 1, and maintain other signals in the previous state for 3 sacs.
- Provide green signal for road 2, green signal for pedestrian on road 1, red signal for other roads and other pedestrians for 6 sacs.
- Put yellow signal for road 2, and maintain other signals in the previous state for 3secs
- Provide green signal for road 3, green signal for pedestrian on road 2, red signal for other roads and other pedestrians for 6secs.

ROAD 1:

Colour	Indication	Port lines
Bi colour (Red)	Pedestrian stop	PA0
Bi colour (green)	Pedestrian Go	PA1
Green 2	Go Right	PA2
Red	Stop	PA3
Yellow	Before stop	PA4
Green 1	Go straight	PA5

ROAD 2:

Colour	Indication	Port lines
Bi colour (Red)	Pedestrian stop	PA6
Bi colour (green)	Pedestrian Go	PA7
Green 2	Go Right	PB0
Red	Stop	PB1
Yellow	Before stop	PB2
Green 1	Go straight	PB3

ROAD 3:

Colour	Indication	Port lines
Bi colour (Red)	Pedestrian stop	PB4
Bi colour (green)	Pedestrian Go	PB5
Green 2	Go Right	PB6
Red	Stop	PB7
Yellow	Before stop	PC0
Green 1	Go straight	PC1

ROAD 4:

Colour	Indication	Port lines
Bi colour (Red)	Pedestrian stop	PC2
Bi colour (green)	Pedestrian Go	PC3
Green 2	Go Right	PC4
Red	Stop	PC5
Yellow	Before stop	PC6
Green 1	Go straight	PC7

- Put yellow signal for road 3, and maintain other signals in the previous state for 3 sacs
- Provide green signal for road 4, green signal for pedestrian on road 3, red signal for other roads and other pedestrians for 6 sacs.
- Put yellow signal for road 4, and maintain other signals in the previous state for 3 sacs.
- Stop the process.

PROGRAM:

Address	Label	Mnemonics	Opcode	Comments
8100		MVI A, 80	3E	Move immediate data to accumulator
8101			80	
8102		OUT PCNT	D3	Out the data from port
8103			1B	
8104	START	LXI H,8150	21	Set pointer to the register pair
8105			50	
8106			81	
8107		MVI C,08	0E	Move immediate data to C - register
8108			08	
8109	LOOP1	MOV A,M	7E	Move data from Memory from accumulator
810A		OUT PA	D3	Out the data from port A
810B			18	
810C		INX H	23	Increment HL pair
810D		MOV A,M	7E	Move data from M to A
810E		OUT PB	D3	Out the data from port B
810F			19	
8110		INX H	23	Increment Hl pair
8111		MOV B,M	46	Move content of M to B
8112		OUT PC	D3	Out the data from port C
8113			19	
8114		INX H	23	Increment Hl Pair
8115		MOV B,M	46	Move the content M to B

For 6 seconds

Green signal on Road 1

Green signal for pedestrian stop on Road 4

Red signal for other Road

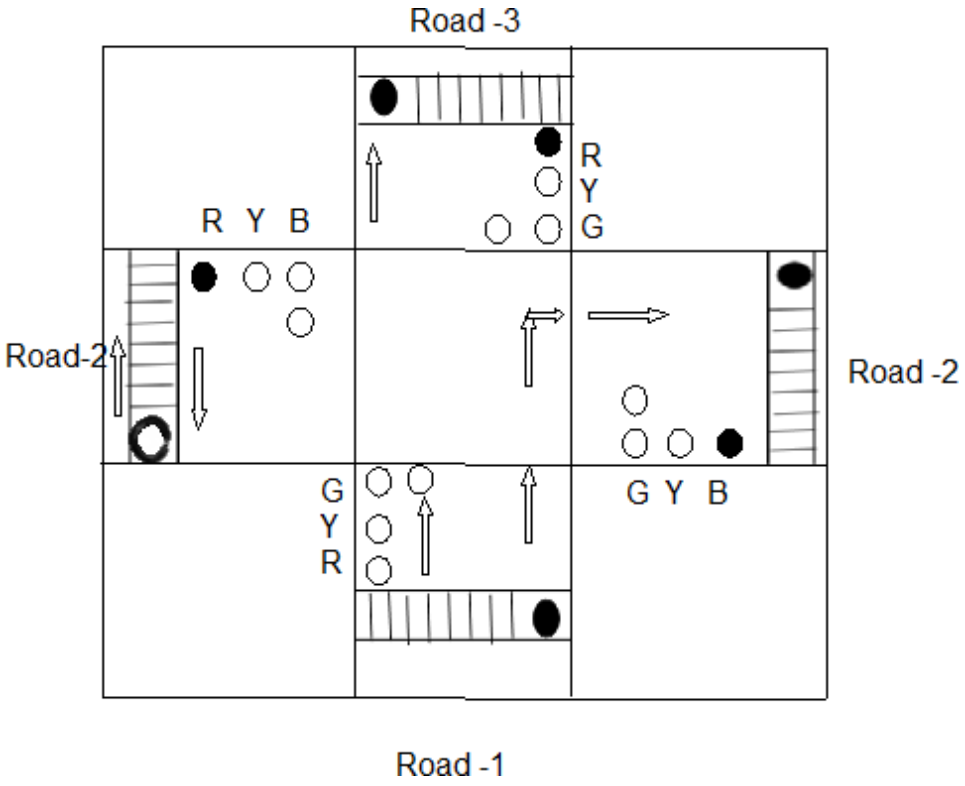
PA_7	PA_6	PA_5	PA_4	PA_3	PA_2	PA_1	PA_0	Hoax decimal value
0	1	1	0	0	1	0	1	65 (PA)
1	0	0	1	0	0	1	0	92(PB)
0	0	1	0	1	0	0	0	28(PC)
0	0	0	0	0	1	1	0	06 (time delay)

INPUT DATA

MEMORY ADDRESS	INPUT DATA
8150	65
8151	92
8152	28
8153	06
8154	51
8155	92
8159	28
815A	03
815B	4A
815C	99
815D	24
815E	06
815F	4A
8160	94
8161	24
8162	03

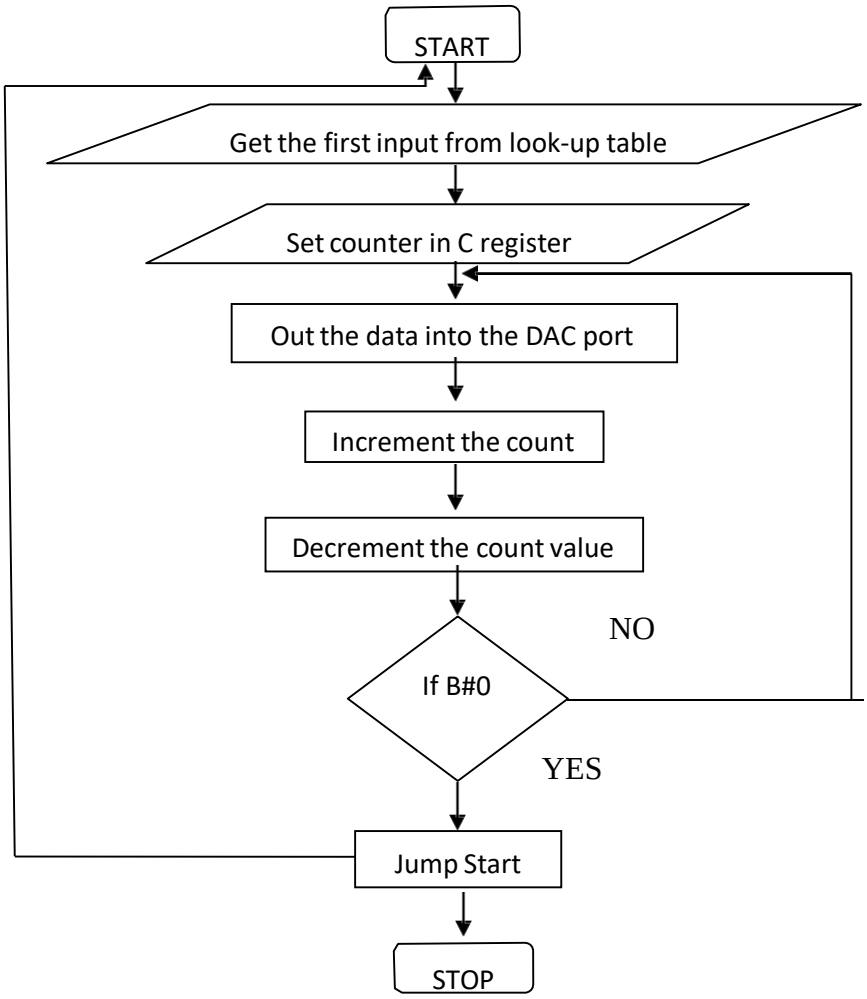
MEMORY ADDRESS	INPUT DATA
8163	89
8164	52
8165	26
8166	06
8167	89
8168	12
8169	25
816A	03
816B	49
816C	A2
816D	94
816E	06
8170	46
8171	A2
8172	44
8173	03

8116		CALL DELAY	CD	Call the delay Label
8117			21	
8118			81	
8119		INX H	23	Increment HL pair
811A		DCR C	OD	Decrement C register
811B		JNZ LOOP1	C2	Go to Loop 1 if no zero
811C			09	
811D			81	
811E		JMP START	C3	Jump to start
811F			04	
8120			81	
8121	DELAY	LXI D,FFFF	11	Delay Program Load Immediate memory content in to D register
8122			FF	
8123			FF	
8124	DLY	DCX D	1B	Decrement DE register
8125		MOV A,E	7B	Move content of E to A
8126		ORA B	B2	
8127		JNZ DLY	C2	Jump on no zero to (Delay) DLY
8128			24	
8129			81	
812A		DCR B	05	Decrement D register
812B		JNZ DELAY	C2	Jump on no zero to Delay
812C			21	
812D			81	
812E		RET	C9	Return to Main program



RESULT:

FLOW CHART:


START
Get the first input from look-up table
Set counter in C register
Out the data into the DAC port
Increment the count
Decrement the count value
NO
If B#0
YES
Jump Start
STOP

Ex. No.: 14 **INTERFACING OF D TO A CONVERTER USING 8085MICROPROCESSOR**

Date :

AIM:

To generator triangular wave at DAC output using 8085 microprocessor.

APPARATUS REQUIRED:

- 8085 microprocessor kit
- Opcode sheet
- DAC interface board.

ALGONTHM:

1. Start the program
2. Get the Fust input from lookup table
3. Set the count in c-register
4. Out the data in to the DAC port
5. Increment the look-up table address
6. Increment the count value
7. If carry is equal to zero go to jump start
8. If no carry is go to Out data
9. Stop program

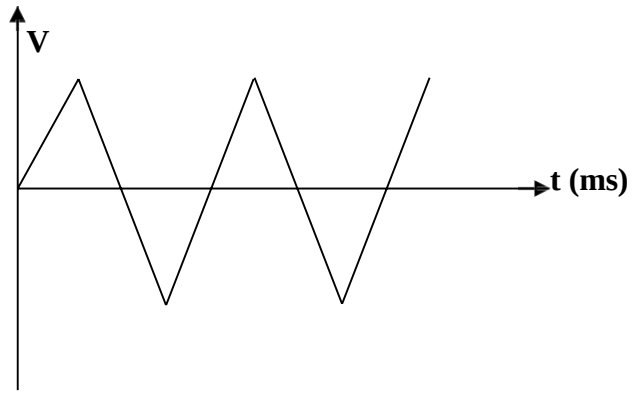
PROGRAM:

ADDRESS	LABLE	PNEMONICS	OPCODE	COMMENTS
8100	START	LXI H, 8110	21	Load the memory address for input data
8101			10	
8102			81	
8103		MVI C , 41	0E	Move 41 to C register
8104			41	
8105	LOOP	MOV A , M	7E	Move the data from M to A
8106		OUT C0	D3	Out the data from port.
8107			C0	
8108		INX H	23	Increment memory pointer

DAC 0800 is an 8-bit DAC and the output voltage varies in between -5v and +5v. The output voltage varies in steps of 10/256 =0.04 (approx) the digital data inputs and the corresponding output voltage are presented in the following table.

Input data in Hex	**Output voltage (V)**
00	0.00
01	0.04
02	0.08
.	.
.	.
.	.
7F	2.15
.	.
.	.
.	.
FD	4.92
FE	4.96
FE	5.00

Model Graph:



TABULATION:

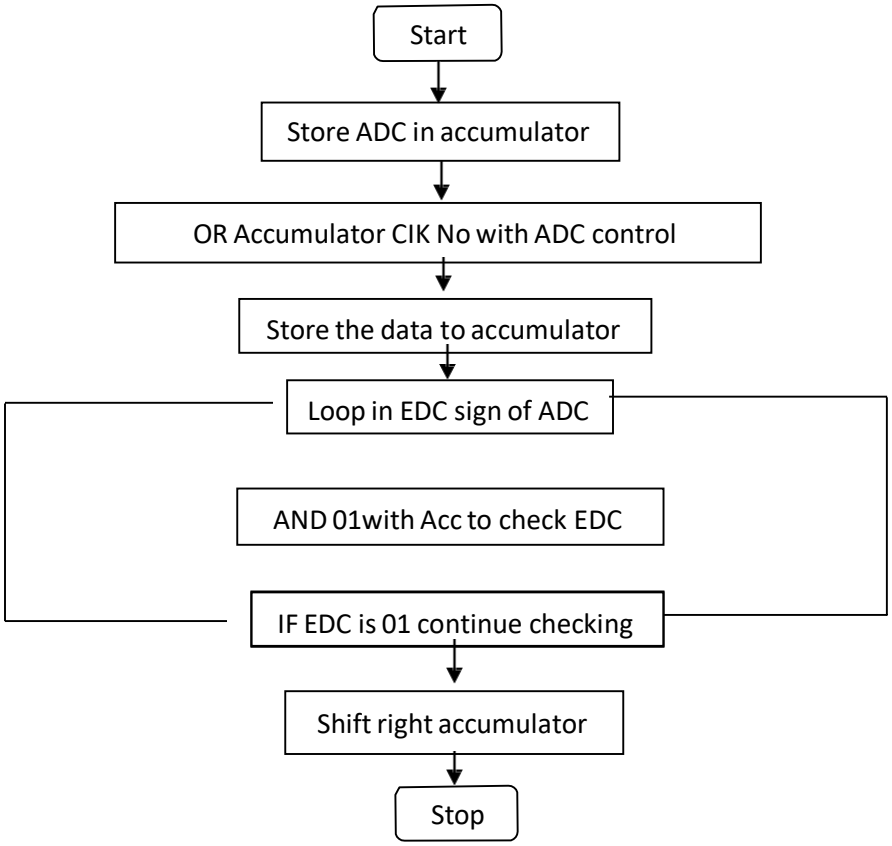
Amplitude	Time period

OUTPUT:

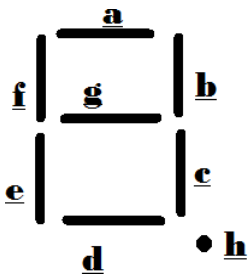
8109		DCR C	OP	Decrement C register.
810A		JNZ LOOP	C2	Jump On no Zero to LOOP
810B			05	
810C			81	
810D		JMP START	C3	Jump to start
810E			00	
810F			81	
8110		LOOK-UP TABLE	00,08,10,18	Look Up data for generation of Triangular Waveform.
8114			20,28,30,38	
8118			40,48,50,58	
811C			60,68,70,78	
8120			80,88,90,98	
8124			A0,A8,B0,B8	
8128			C0,C8,D0,D8	
812C			E0,E8,F0,F8	
8130			FF,F8,F0,E8	
8134			E0,D8,D0,C8	
8138			C0,B8,B0,A8	
813C			A0, 98, 90,88	
8140			80,78,70,68	
8144			60,58,50,48	
8148			40,38,30,28	
814C			20,18,10,08	
8150			00	

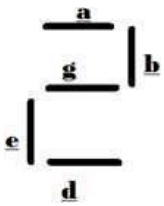
RESULT:

FLOWCHART:



SEVEN SEGMENT DISPLAY



For Example	**d**	**c**	**b**	**a**	**h**	**e**	**g**	**f**	
	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**-- 49_H**

Ex. No.: 15

Date : INTERFACING OF A TO D CONVERTER USING 8085 MICROPROCESSOR

AIM:

To write an assembly level language program to interface A to D converter using 8085 microprocessor.

APPARATUS REQUIRED:

- 8085 microprocessor kit
- OPcode sheet
- ADC interface.

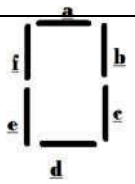
THEORY:

The A/D Conversion is a quantizing process whereby an analog signal is represented by equivalent binary states. This is opposite to b/a conversion process. Analog – to- digital converters can be classified in two general group based on the conversion technique. One technique involves comparing a given analog signal with the internally generated equivalent signal. This group includes successive approximation, counter and flash hypes converters. The second technique involves changing an analog into or frequency and comparing these new parameters against known values this group includes integrator converters and voltage to frequency converters the tradeoff between the two techniques is based on accuracy Vs speed. The successive approximation and the flash hope are faster but generally lees accurate than the in territory and the voltage to frequency hype converters.

ALGORITHM:

1. Start the program
2. Set the ADC control ward to accumulator
3. Load the input value in the accumulator
4. Out the data from ADC port
5. The output are taken in digital form
6. Store the result
7. Stop the program.

INPUT & OUTPUT TABULATION:

Memory Address	Data	7 segment display	d	c	b	a	h	e	g	f	Hex Value (i/p data)
8200	0		0	0	0	0	1	0	1	0	0A
8201	1		1	0	0	1	1	1	1	1	9F
8202	2		0	1	0	0	1	0	0	1	49
8203	3										0D
8204	4										9C
8205	5										2C
8206	6										28
8207	7										8F

PROGRAM

Address	Label	Mnemonics	OPcode	Comments
8100	START	MVI A,00	3E	Store the ADC channel no to Acc.
8101			00	
8102		OUT C8	D3	Output the control word to ADC control reg.
8103			C8	
8104		ORI 08	F6	Logically OR with A and ALE signal through 08.
8105			08	
8106		OUT 08	D3	Output the control word to ADC control reg.
8107			C8	
8108		NOP	00	Wait for few nano sec.
8109		NOP	00	
810A		NOP	00	
810B		ANI F7	E6	Logically AND with A and Reset ALE signal through F7.
810C			F7	
810D		OUT C8	D3	Output the control word to ADC
810E			C8	
810F		NOP	00	Wait for few nano sec.
810O		NOP	00	
8111		NOP	00	
8112		MVI A,10	3E	Move control word 10 to accumulator
8113			10	
8114		OUT C8	D3	Output the control word to ADC
8115			C8	
8116		NOP	00	Wait for few nano sec.
8117		NOP	00	
8118		NOP	00	
8119		MVI A,20	3E	Move control word 20 to accumulator
811 A			20	
811B		OUT C8	D3	Output the control word to ADC
811C			C8	
811D	LOOP	IN C0	DB	Input the EOC signal from ADC
811E			C0	
811F		ANI 01	E6	Logically AND with 01 and A, to check EOC signal.
8120			01	
8121		JNZ LOOP	C2	Jump on no zero to loop
8122			ID	
8123			81	
8124		IN C4	DB	Input the digital signal from ADC
8125			C4	
8126		MOV B,A	47	Move data from A to B

Memory Address	Data	7 segment display	d	c	b	a	h	e	g	f	Hex Value (i/p data)
8208	8										08
8209	9										8C
820A	A										88
820B	B										38
820C	C										6A
820D	D										19
820E	E										68
820F	F										E8

8127		LXI H, 8200	21	Load the starting address of the lookup table
8128			00	
8129			82	
812A		MVI A,94	3E	Move control word 94 to accumulator
812B			94	
812C		OUT 01	D3	Output the control word to ADC
812D			01	
812E		MOV A , B	78	Move data from B to A
812F		ANI OF	E6	Logically AND with 0F and A, to get MSD of the digital output.
8130			0F	
8131		RLC	07	Rotate left through Carry
8132		RLC	07	
8133		RLC	07	
8134		RLC	07	
8135		MOV L , A	0F	Move data from A to L
8136		MOV A,M	7E	Move data from M to A
8137		OUT 00	D3	Output the 1st data to ADC
8138			00	
8139		MOV A , B	78	Move data from B to A
813A		ANI OF	E6	Logically AND with 0F and A, to get LSD of the digital output.
813B			0F	
813C		MOV L , A	6F	Move data from A to L
813D			7E	Move data from M to A
813E		OUT 00	D3	Output the 2nd data to ADC
813F			00	
8140		JMP START	C3	Jump to start label.
8141			00	
8142			81	
8143		HLT	76	Stop the program

RESULT:

Ex. No.: 16 **SERIAL PORT INTERFACE USING 8085MICROPROCESSOR**

Date :

AIM:

To write a program to transmit the data 55 using serial port Interface 8251

APPARATUS REQUIRED:

- 8085 micro processor kit
- Opcode sheet
- Serial port Interface

ALGORITHM

1) Initialize Timer for 9600 baud rate
2) OUT the data 00 into the USART Port
3) Initialize 8251
4) Transmit the data in to USART Port
5) Receive the same data through USART port
6) Stop the program

PROGRAM:

Address	Label	Pneumonic	OPcode	Comments
		ORG 8100H	C3	TIMER CONT
		EQU C3H	00	
			C0	
		EQU C0	00	CHANNAL 0
			C5	
		EQU C5	00	USART CONT
			C4	
		EQU C4	00	USART DATA
			3E	
8100		MVI A , 36	36	Move control word 36 to A – Register
8101			D3	
8102		OUT TIMER CONT	C6	Output the control word to 8251 SPI
8103			3E	
8104		MVI A ,0A	0A	Move data 0A to ACC.
8105			D3	
8106		OUT CHANNAL 0	C0	Output the control word to 8251 SPI
8107				

	8108		MVI A , 00	3E	Clear the Accumulator
	8109			00	
	810A		OUT CHANNEL 0	D3	Output the control word to 8251 SPI
	810B			C0	
	810C		MVI A , 00	3E	Clear the Accumulator
	810D			00	
	810E		OUT USARTCONT	D3	Output the control word to 8251 SPI
	810F			CA	
	8110		OUT USARTCONT	D3	Output the control word to 8251 SPI
	8111			CA	
	8112		OUT USARTCONT	D3	Output the control word to 8251 SPI
	8113			CA	
	8114		MVI A , 40	3E	Move data 04 to accumulator
	8115			40	
	8116		OUT USARTCONT	D3	Output the control word to 8251 SPI
	8117			CA	
	8118		MVI A , 4E	3E	Move data 4 E to Accumulator
	8119			4E	
	811A		OUT USARTCONT	D3	Output the control word to 8251 SPI
	811B			CA	
	811C		MVI A , 37	3E	Move data 37 to Accumulator
	811D			37	
	811E		OUT USARTCONT	D3	Output the control word to 8251 SPI
	811F			CA	
	8120	TXDNRDY	IN USARTCONT	DB	Input the USARTCONT to SPI
	8121			CA	
	8122		ANI 04	E6	Logical AND with Acc and 04
	8123			04	
	8124		JZ TXDNRDY	CA	Jump on zero to TXDNRDY label
	8125			20	
	8126			81	
	8127		MVI A , 55	3E	Move data 55 to Accumulator
	8128			55	
	8129		OUT USARTDATA	D3	Output the control word to 8251 SPI
	812A			C8	
	812B	RXNRDY	IN USARTCONT	DB	Input the USARTCONT to SPI

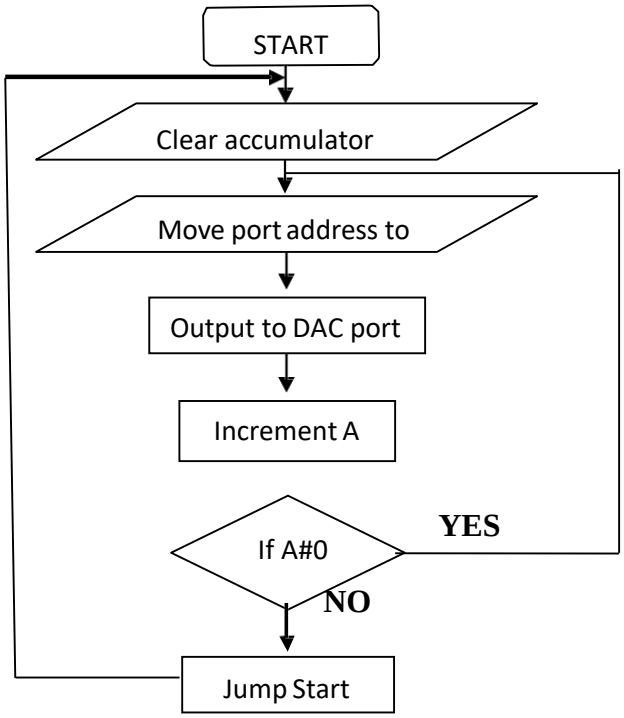
	812C			CA	
	812D		ANI 02	E6	Logical AND with Acc and 02
	812E			02	
	812F		JZ RXNRDY	CA	Jump on zero to RXNRDY label
	8130			2B	
	8131			81	
	8132		IN USARTCONT	DB	Input the USARTCONT to SPI
	8133			C8	
	8134		STA 8500	32	Store the data in 8500
	8135			00	
	8136			85	
	8137		HLT	76	Stop the Program.

INPUT & OUTPUT TABULATION:

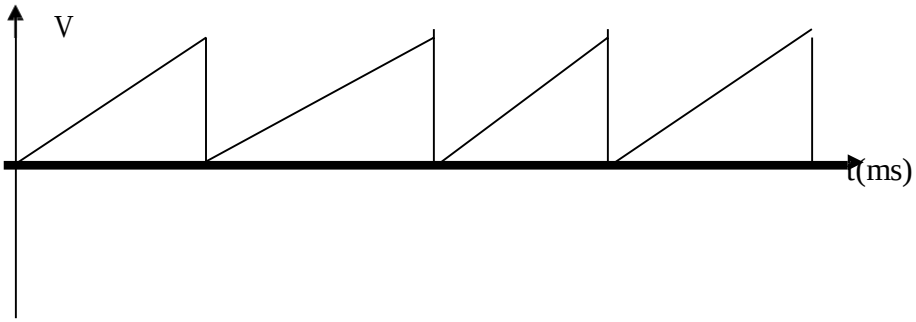
RESULT:

Memory Address	**Input data**	**Memory Address**	**Output data**
8128		8500	

FLOW CHART:



MODEL GRAPH:



TABULATION:

Amplitude	Time period

OUTPUT:

Ex. No.: 17 INTERFACING D TO A CONVERTER USING 8051MICROCONTROLLER

Date :

AIM:

To generate saw both wave at digital to analog converter output.

Apparatus Required:

- 8051 microcontroller
- OPcode sheet
- DAC Interface Board

ALGORITHM:

1. Start the program
2. Clear accumulator
3. Move the port address to DPTR
4. Output to DAC port
5. Increment the accumulator
6. If A is not Zero go to step 4
7. Long jump to Step 1.

PROGRAM:

Address	Label	Mnemonics	Opcode	Comments
8100	START	MOVA,#00	74	Move 00 to accumulator
8101			00	
8102		MOV DPTR, # E0C0	90	DAC1, address in port
8102			E0	
8104			C0	
8105	LOOP	MOVX @DPDR , A	F0	Output to data port
8106		INC A	04	Increment A
8107		JNZ LOOP	70	If A is not zero go to l
8108			FC	
8109		LJMP START	02	Go to start
810A			81	
810B			00	

RESULT:

Ex. No.: 1818 **INTERFACING A TO D CONVERTER USING 8051**

Date:

AIM:

To generate saw tooth wave at analog to digital converter output.

Apparatus Required:

- 8051 microcontroller
- OPcode sheet
- ADC Interface Board

PROGRAM:

Address	Label	Mnemonics	Opcode	Comments
8100	START	MOV A,#00	74,00	ADC select channel
8102		MOV DPTR, #E0C8	90 ,E0,C8	ADC Control port address
8105		MOVX @DPTR,A	F0	Out ADC Channel no to ADC control port
8106		NOP	00	
8107		NOP	00	
8108		NOP	00	
8109		MOV A,#08	74, 08	Send ALE to ADC
PORT				
810B		MOVX @DPTR,A	F0	
810C		NOP	00	
810D		NOP	00	
810E		NOP	00	
810F		MOV A,#10	74,10	Start of conversion
8111		MOVX @DPTR, A	F0	
8112		NOP	00	
8113		NOP	00	
8114		NOP	00	
8115		MOV A,#10	74,20	Output enable
8117		MOVX @DPTR,A	F0	
8118		NOP	00	
8119		NOP	00	
811A		NOP	00	

811B		MOV DPTR,#E0 C0	90, E0, C0	EOC port address
811E		MOVX A,@DPTR	E0	Get end of the conversion
811F		ANL A,#01	54,01	
8121		JZ 811E	60,FB	If low get EOC again
8123		MOV DPTR, #E0C4	90,E0,C4	Data port address
8126		MOVX A,@DPTR	E0	
8127		MOV DPTR,#8500	90,8500	Store data
812A		MOVX @DPTR,A	F0	
812B		LIMP 8100	02, 812B	

RESULT:

Ex. No.: 19 INTERFACING OF DC MOTOR USING 8051 MICROCONTROLLER

Date :

AIM:

To control the speed of a DC motor using 8253.

ALGORITHM:

- Initialize 8253 counter 0 in mode 3 (Square wave generator). It gives frequency input to FTOV converter for the desired speed.
- Load counter 0 with count proportional to the speedrequired.
- Give input frequency for the speed required at 8200H inhex.

PROGRAM:

ADDR	OPCODES	MNEMONICS	COMMENTS
;To give frequency input to FTOV convertor			
8100	74 36	MOV A, #36H	; 8253 counter 0 in mode 3 ; square wave generator
8102	90 E0 0B	MOV DPTR,#E00B	
8105	F0	MOVX@DPTR,A	
;To load the count in 8253 counter 0			
8106	90 82 00	MOV DPTR, #8200H	;Read LSB count ; from 8200H
8109	E0	MOVX A, @DPTR	
810A	90 E0 08	MOV DPTR, #E008H	;counter 0 addr.
810D	F0	MOVX @DPTR, A	;Output MSB count
810E	90 82 01	MOV DPTR, #8201H	;Read MSB count from 8201H
8111	E0	MOV A, @DPTR	
8112	90 E0 08	MOV DPTR, #E008H	;counter 0 addr.
8115	F0	MOVX @DPTR, A	;output MSB count
8116	80 FE	SJMP HERE	

Verification:

Refer the verification procedure in 8085 programming enclosed at the previous section.

LOOKUP TABLE

INPUT	SPEED (RPM)
FFFF	100
FC54	150
F8A9	200
F4FE	250
F153	300
EDA8	350
E9FD	400
E652	450
E2A7	500
DEFC	550
DB51	600
D7A6	650
D3FB	700
D050	750
CCA5	800
C8FA	850
C54F	900
C1A4	950
BDF9	1000

BA4F	1050
B6A3	1100
B2F8	1150
AF4D	1200
ABA2	1250
A7F7	1300
A44C	1350
A0A1	1400
9CF6	1450
994B	1500
95A0	1550
91F5	1600
8E4A	1650
8A9F	1700
86F4	1750
8349	1800
7F9E	1850
7BF3	1900
7848	1950
749D	2000
70F2	2050
6D47	2100
699C	2150
65F1	2200
6246	2250
5E9B	2300

5AF0	2350
5745	2400
539A	2450
4FEF	2500

RESULT:

Ex. No.: 20 INTERFACING OF AC MOTOR USING 8051 MICROCONTROLLER

Date :

AIM:

To Control the speed of AC motor by controlling the firing pulses.

Requirement:

- AC motor Speed Controller interface Board
- Ac motor
- MP/MC trainer kit
- 26 pin interface cable

Procedure:

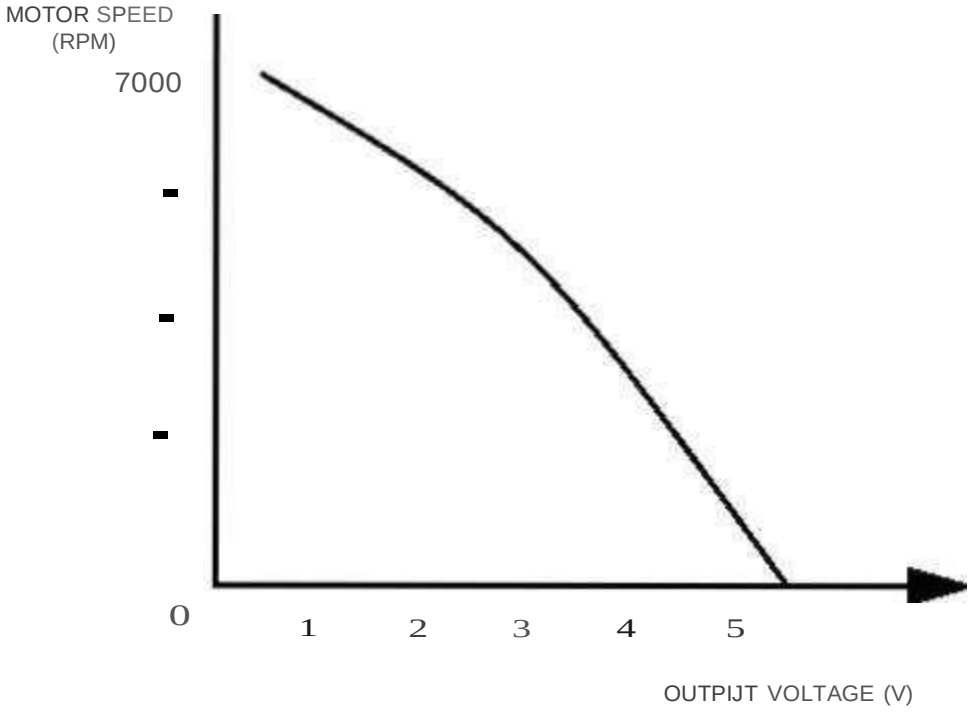
- Interface the MP/MC kit with AC Motor speed controller board using 26 pin FRC cable provided.
- Switch ON the Trainer
- Type the Program given below in the memory location with the starting address 8100H.

❖ Execute the following program and observe that the output voltage at DAC1.Change the value in A and observe the corresponding output voltage at DAC1. Give Digital input for the speed required at 8107H inhex.

PROGRAM:

			ORG8100H	
ADDR	**OPCODE**	**LABEL**	**MNEMONICS**	**COMMENTS**
8100	74 80		MOV A, #80H	
8102	90 E0 1B		MOV DPTR,#E01BH	
8105	F0 74 7F		MOV A, #7FH	;Move ‘7F’ to acc
8108	90 E0 18		MOV DPTR,#E018H	;DAC1 address in DPTR
810B	F0		MOVX @DPTR,A	;Output to DAC1
810C	80 FE	HERE:	SJMP HERE	;Jump here itself

MODEL GRAPH:



DATA TABLE:

INPUT DATA IN HEX	OUTPUT VOLTAGE (V)	MOTOR SPEED (RPM)
00	0.00	20000
01	0.04	-
02	0.08	-
.	.	.
.	.	.
7F	2.50	-
.	.	.
.	.	.
FE	4.96	-
FF	5.00	0

RESULT: