Manonmaniam Sundaranar University,
Directorate of Distance & Continuing Education,
Tirunelveli- 627 012, Tamil Nadu, India

OPEN AND DISTANCE LEARNING (ODL) PROGRAMMES

(FOR THOSE WHO JOINED THE PROGRAMMES FROM THE ACADEMIC YEAR 2023–2024)

M. Sc. Physics

Course Material

Practical III

SPHP31

Prepared By

Dr. S. Shailajha
Dr. B. Bagyalakshmi

Assistant Professor
Department of Physics
Manonmaniam Sundaranar University
Tirunelveli – 12

PRACTICAL – III

SPHP31

Advanced Physics Experiments – I and Microprocessor 8085 & Microcontroller 8051 Programming

Section – A (any 6 experiment)

Section B : Microprocessor 8085 and Microcontroller 8051 Programming
(Any 6 Experiments)

1. Cauchy's Constants Of A Prism Using A Spectrometer

Aim:

To determine the Cauchy's constants (A and) for the given prism material by studying the variation of refractive index (n) with wavelength (λ) and fitting a straight line using experimental data.

Apparatus and Software Requirement:

Spectrometer, Prism, Mercury lamp, Spirit level, Origin or Excel

Formula:

1. The refractive index (μ) of the material of the prism is given by:

$$\mu = \left(\frac{\sin\left(\frac{A + \delta_m}{2} \right)}{\sin\left(\frac{A}{2} \right)} \right)$$

Where:

A = Angle of the prism

δ_m = Angle of minimum deviation

2. Variation of refractive index with wavelength is represented by Cauchy's relation:

$$\mu = A + \frac{B}{\lambda^2}$$

Where:

A and B are Cauchy's constants

λ = Wavelength of light in m

Cauchy's constants are calculated as:

$$B = \frac{\mu_b - \mu_g}{\frac{1}{\lambda_b^2} - \frac{1}{\lambda_g^2}} \left(m^2 \right)$$

$$A = \mu_b - \frac{B}{\lambda_b^2}$$

Diagram

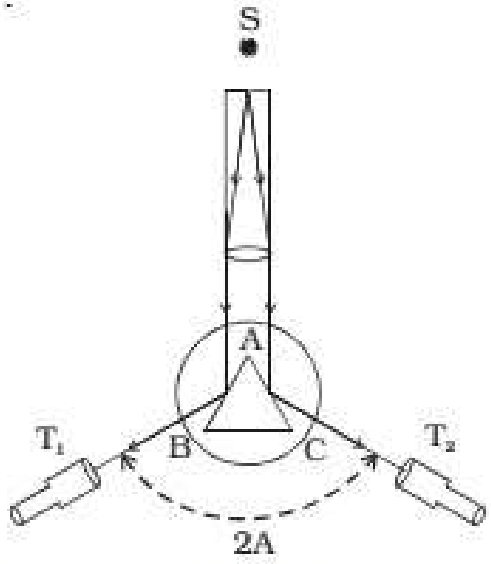


Fig. Angle of the prism

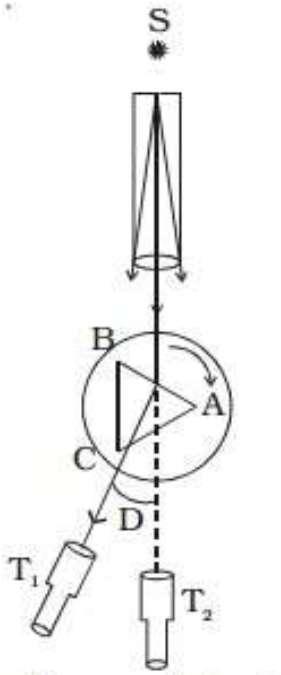
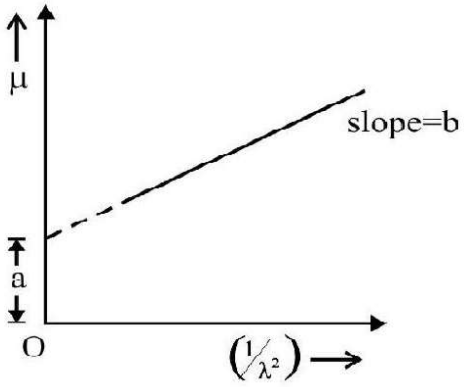


Fig. Angle of minimum deviation

Model Graph



Procedure:

- Step 1: Calibration of Spectrometer

  Adjust the spectrometer to focus the crosshairs of the telescope.

  Calibrate it using a known wavelength (e.g., sodium D-line at 589 nm).

- Step 2: Prism Setup

  Place the prism on the prism table with one face towards the collimated light beam.

  Adjust the prism and spectrometer to achieve the minimum deviation position for each wavelength.

- Step 3: Measure Angle of Minimum Deviation (δm)

  Rotate the telescope to observe the refracted ray.

  Adjust the position of the prism and the telescope to find the angle of minimum deviation for each wavelength.

- Step 4: Calculate Refractive Index (μ)

  The refractive index (μ) of the prism material is calculated using: $\mu = \sin((A + \delta_m)/2) / \sin(A/2)$. where A is the angle of the prism.

- Step 5: Tabulate Observations

  Record the refractive index (μ) for each wavelength (λ). μ_b & μ_g are refractive index of blue and green colour respectively. Use the formula to find A and B.

- Step 6: Fit Data Using Software

  Use any curve-fitting software (origin, MATLAB, Excel, Python, etc.) to fit the data to Cauchy's equation.

  Plot μ vs. $1/\lambda^2$. Determine the slope (B) and intercept (A) from the linear fit.

For Graph plotting

- In software like Origin, you can fit the experimental data (μ vs. λ) to this equation.
- Convert λ to λ^2 for simplicity, and perform a linear fit if rearranging to:
- $\mu = A + Bx$, where $x = 1/\lambda^2$
- Plot μ on the y-axis and $1/\lambda^2$ on the x-axis.
- Use a linear fitting tool in Origin to calculate A (intercept) and B (slope).

Observations:

1. Least Count Calculation:

Main Scale Division (x) = _____ Degree

Total Vernier Scale Divisions (n) = _____

Least Count = x/n = _____ Degree

2. Angle of Prism (A) Measurements:

Vernier	Telescope Readings for Reflection from						Difference 2A = b-a (Degree)	Angle of the Prism (A)	Mean (A)
	First face (a) Degree			Second face (b) Degree					
	MSR	VSC	TR	MSR	VSC	TR			
V1									
V2									

3. Table for the Angle of Minimum Deviation (δ_m):

Colour	Vernier	Telescope Readings for						Difference δ_m = b-a (Degree)	Mean δ_m (Degree)
		Refracted Ray			Direct Ray				
		MSR	VSC	TR	MSR	VSC	TR		
Blue	V1								
	V2								
Green	V1								
	V2								

Calculation

Refractive index for each color is calculated using:

$\mu = \sin((A + \delta_m)/2) / \sin(A/2)$

Cauchy's constants are calculated using:

$$B = \frac{\mu_b - \mu_g}{\frac{1}{\lambda_b^2} - \frac{1}{\lambda_g^2}}$$

$$A = \mu_b - \frac{B}{\lambda_b^2}$$

Result:

1. The refractive indices of the prism for various colours are:

2. The Cauchy's constants are:

From calculation A = _____ From graph A = _____

B = _____ m^2 B = _____ m^2

2. Design and Simulation of an Astable Multivibrator Using a 555 Timer for a Given Frequency

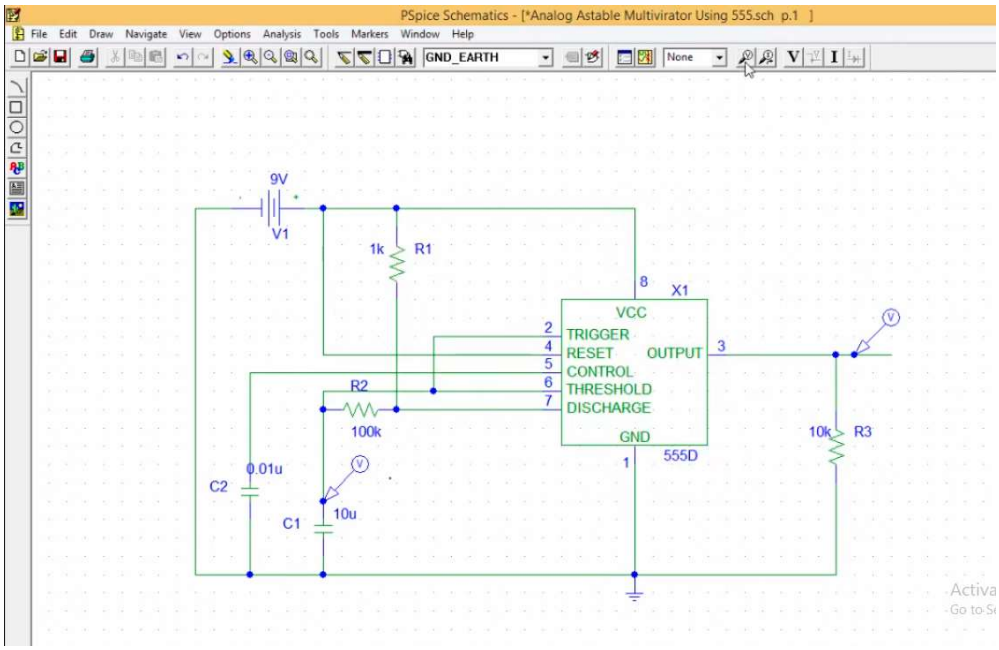
Aim

To design and simulate an Astable Multivibrator circuit using a 555 Timer in PSpice for generating a square wave with a specified frequency and duty cycle.

Apparatus and Software required:

1. PSpice Software
2. 555 Timer IC Model
3. Resistors and Capacitors (virtual components in PSpice)
4. Oscilloscope (virtual)

Circuit Diagram:



Theory:

The 555 timer is a widely used IC for generating clock pulses, delays, and oscillations. In the **Astable Mode**, the circuit oscillates continuously between HIGH and LOW states, producing a square wave. The output frequency (f) and duty cycle (D) depend on the resistor and capacitor values:

1. Frequency (f):

$$f = \frac{1.44}{(R_1 + 2R_2)C}$$

2. Duty Cycle (D):

$$D = \frac{(R_1 + R_2)}{(R_1 + 2R_2)} \times 100$$

Design Steps:

1. **Specification:**
   - Desired Frequency (f): User-defined (e.g., 1 kHz)
   - Duty Cycle (D): Optional (e.g., 50%)
2. **Calculate Components:** Using the frequency formula:

$$C = \frac{1.44}{f(R_1 + 2R_2)}$$

Select standard values for R_1, R_2, and calculate C.

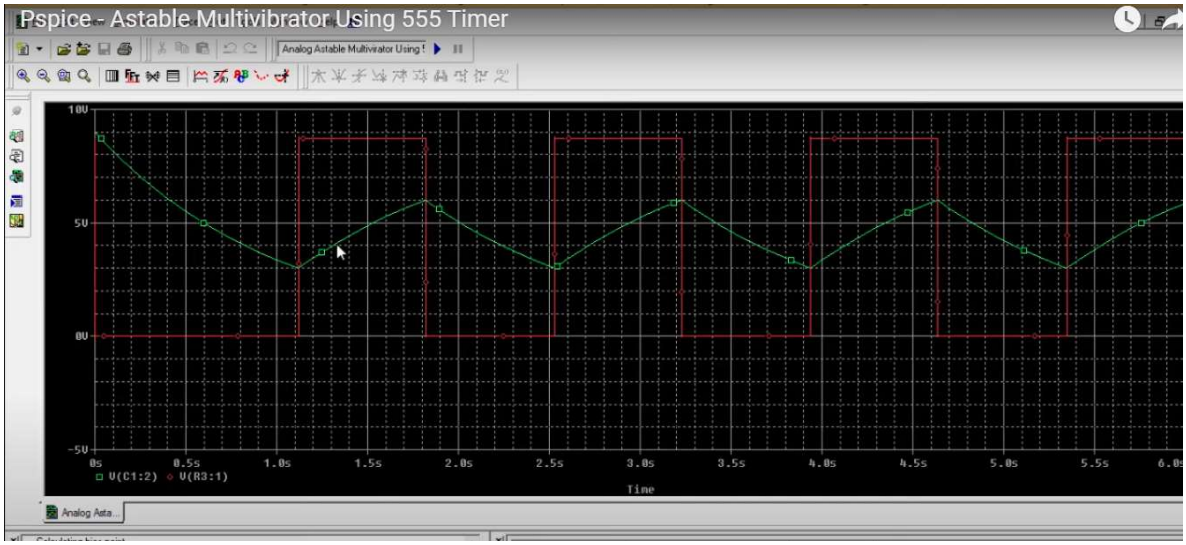
Simulation Steps in PSpice:

1. **Start a New Project:**
   - Open PSpice.
   - Create a new project and choose "Analog or Mixed A/D."
2. **Place Components:**
   - Add a 555 Timer IC.
   - Place R1, R2, and CC according to the circuit diagram.
   - Add a DC power source (V_{CC}) and ground.
3. **Connect the Circuit:**
   - Wire all components as per the design.
4. **Add Probes:**
   - Place voltage probes at the output pin (pin 3 of the 555 timer).
5. **Set Simulation Parameters:**
   - Choose a transient analysis.
   - Set the total simulation time (e.g., 10 ms for a 1 kHz frequency).
6. **Run Simulation:**

Start the simulation and observe the output waveform on the oscilloscope

Observations:

1. Record the output waveform.
2. Measure the frequency and compare it with the calculated value.
3. Verify the duty cycle.



Result:

The Astable Multivibrator was successfully simulated in PSpice. The output waveform, frequency, and duty cycle were observed and matched the theoretical calculations.

3. Simulation of Zener Diode Characteristics and Voltage Regulator in PSpice

Aim

1. To simulate and study the V-I characteristics of a Zener diode in PSpice.
2. To design and simulate a voltage regulator circuit using a Zener diode.

Apparatus and Software required:

1. PSpice Software
2. Zener Diode Model
3. Resistors (virtual)
4. DC Voltage Source (virtual)

1: Zener Diode Characteristics

Circuit Diagram:

The circuit consists of:

- Zener diode
- Variable DC voltage source
- Series resistor (for current limiting)

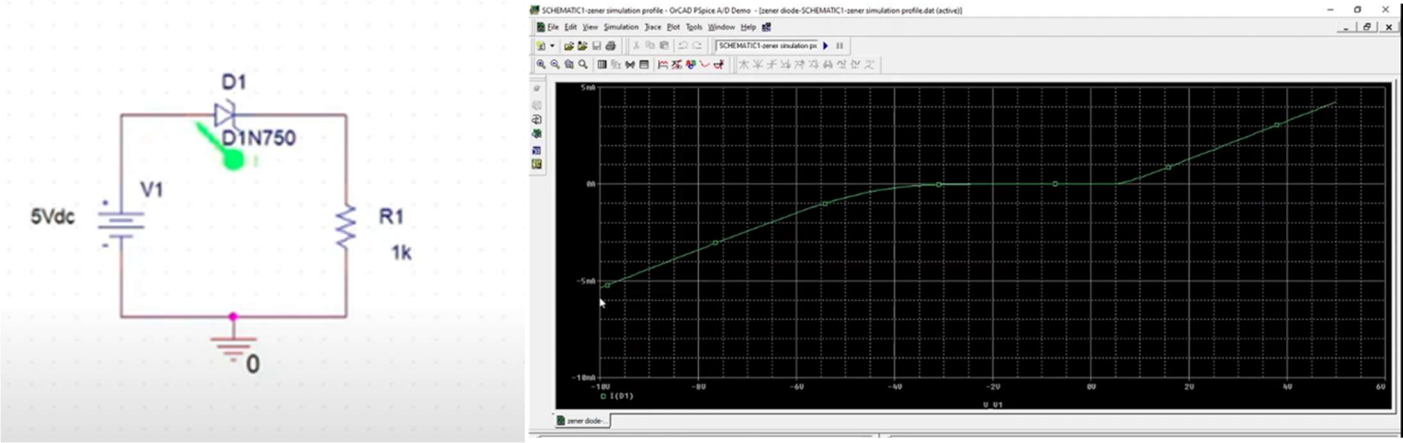


Fig. 1: Zener diode : I – V characteristics

Simulation Steps:

1. **Start a New Project:**
   - Open PSpice and create a new project.
2. **Place Components:**
   - Add a Zener diode (e.g., 1N4733 or as per your specification).
   - Add a DC voltage source.
   - Place a resistor in series with the Zener diode.
3. **Connect the Circuit:**

- Wire the components as per the circuit diagram for both forward and reverse bias.

4. **Set Simulation Parameters:**
   - Use DC Sweep Analysis:
     - Sweep voltage source from 0 to a value above V_Z (e.g., 0–10 V).
     - Set the increment step (e.g., 0.1 V).
5. **Run Simulation:**
   - Observe the current (I_D) through the diode and voltage (V_D) across it.

Observations:

- Plot V_D vs. I_D to obtain the Zener diode characteristics.
- Note the Zener breakdown voltage (V_Z) in reverse bias.

Result:

The Zener diode V-I characteristics were successfully simulated, and the Zener breakdown voltage was verified.

2: Zener Diode Voltage Regulator

Circuit Diagram:

The circuit consists of:

- Zener diode
- Resistor (Rs) for current limiting
- Load resistor (R_L)
- DC voltage source (V_{in})

Design Steps:

1. Specification:

- Input Voltage (V_{in}): e.g., 10 V
- Desired Output Voltage (V_{out}): Zener Voltage (V_Z), e.g., 5.1 V
- Load Current (I_L): e.g., 20 mA

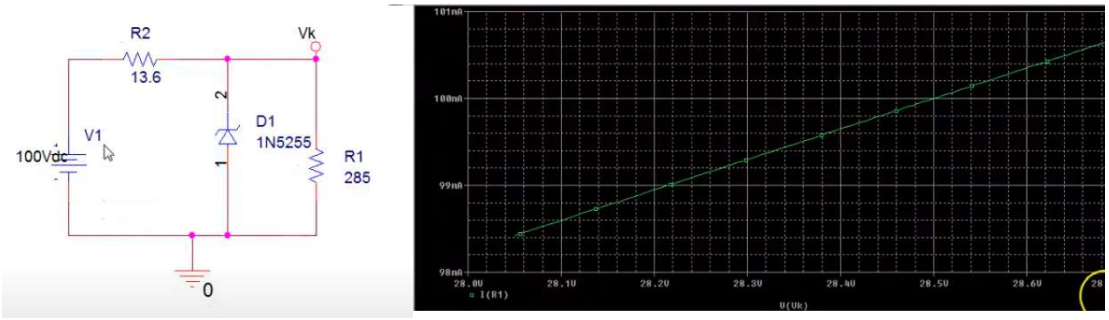
2. Calculate Resistor Values:

- Series Resistor (R_s)

$$R_s = \frac{V_{in} - V_Z}{I_L + I_Z}$$

Where I_z is the Zener current, chosen to keep the diode in breakdown region

Circuit Diagram



Simulation Steps:

1. **Start a New Project:**
   - Open PSpice and create a new project.
2. **Place Components:**
   - Add a Zener diode, resistor (R_2), load resistor (R_1), and DC voltage source (V_{in}).
3. **Connect the Circuit:**
   - Wire the components as per the voltage regulator circuit diagram.
4. **Set Simulation Parameters:**
   - Use DC Sweep Analysis:
     - Sweep V_{in} from a value below V_Z to a higher value (e.g., 0–15 V).
5. **Run Simulation:**
   - Observe the output voltage (V_{out}) across the load resistor (R_L).

Expected Results

Constant Output Voltage: The output voltage across the Zener diode remains steady near the breakdown voltage (e.g., 5.1V).

Effect of Load: Verify if the Zener can maintain regulation when the load resistance decrease

Result:

The Zener diode voltage regulator was successfully simulated, and a constant output voltage was observed across the load.

4. Characteristics Of Solar Cell

Aim

To study I-V Characteristics, Load Response, and Spectral Response of Photovoltaic

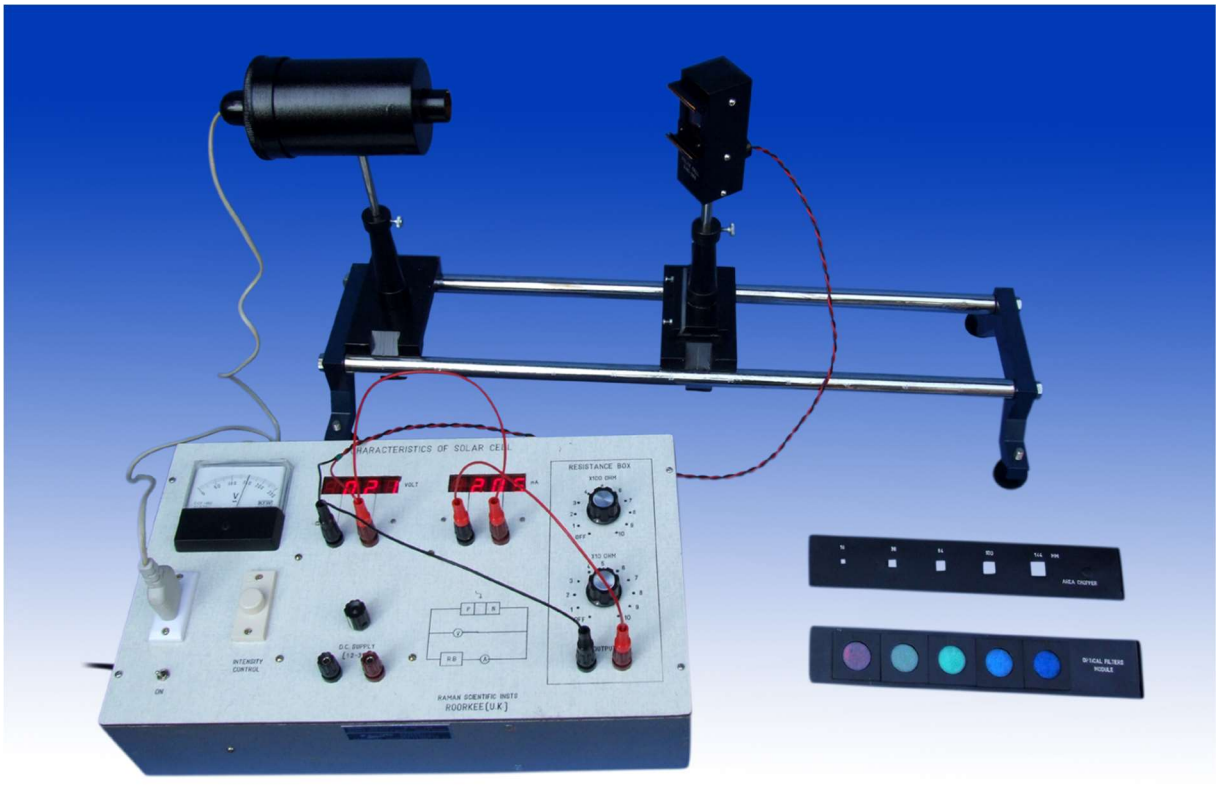
Solar Cell

Apparatus required:

Solar cell measurement kit/

1. Photovoltaic solar cell (small PV panel)
2. Adjustable light source (e.g., halogen lamp or solar simulator)
3. Variable resistive load or rheostat
4. Multimeter (for voltage and current measurements)
5. Spectrometer or filters for spectral response analysis
6. Power supply (for light source, if needed)
7. Connecting wires and breadboard (if required)

Experimental Setup



Formula used:

Parameter	Formula	Explanation	Observation
Short-Circuit Current (I_S)	Measured directly (when V=0)	The maximum current the solar cell can produce when its terminals are short-circuited.	The current value when the load resistance is zero.
Open-Circuit Voltage (V_{OC})	Measured directly (when I=0)	The maximum voltage the solar cell can produce when there is no current flow (open circuit).	The voltage across the solar cell with no connected load.
Maximum Power (P_{MPP})	$P_{MPP}=V_{MPP}\times I_{MPP}$	The product of voltage and current at the point where the solar cell delivers maximum power.	The power output is maximum at a specific combination of V_{MPP} and I_{MPP}.
Fill Factor (FF)	$FF = \frac{P_{MPP}}{V_{OC} \times I_{SC}}$	A measure of the quality of the solar cell, indicating how close the I-V curve is to a rectangle.	Higher FF indicates better performance.
Efficiency (η)	$\eta = \frac{P_{MPP}}{InputPower} \times 100$	The ratio of the solar cell's maximum power output to the power input from the light source.	Efficiency depends on the cell type and conditions.
Spectral Response (I_λ)	Measured directly for each wavelength	Indicates how sensitive the solar cell is to different wavelengths of light.	Higher response for specific wavelengths corresponds to better absorption by the solar cell.

Procedure: Part A: I-V Characteristics

1. **Setup:**
   - Connect the solar cell to a variable resistor (load).
   - Place the light source at a fixed distance from the solar cell.
   - Use a multimeter to measure current (I) and voltage (V).
2. **Measurements:**
   - Vary the load resistance from a very low value to a high value.
   - For each resistance, record the current and voltage readings.
3. **Plot:**
   - Plot the I-V curve with current (I) on the y-axis and voltage (V) on the x-axis.

Part B: Load Response

1. **Setup:**
   - Use the same setup as Part A.
2. **Measurements:**
   - Connect different resistive loads and measure the voltage across and current through each load.
   - Calculate the power $P=V\times I$ for each load.
3. **Plot:**
   - Plot the power vs. load resistance to observe the load response.

Part C: Spectral Response

1. **Setup:**
   - Use a spectrometer or place optical filters between the light source and solar cell to isolate specific wavelengths.
2. **Measurements:**
   - For each wavelength or filter, measure the current generated by the solar cell under constant illumination intensity.
3. **Plot:**
   - Plot the spectral response curve with current (I) or efficiency on the y-axis and wavelength (λ) on the x-axis.

Observation

Part A: I-V characteristcs

Intensity =

Load Resistance (R) ohm	Voltage (V)	Current (I) mA	Power (P=V×I) mW

Part B: Load Response

Load Resistance (R)	Voltage (V)	Current (I)	Power (P=V×I) mW

Part C: Spectral Response

Wavelength (λ)	Current (I)	Voltage (V)	Observations

Result:

1. The I-V characteristics show the solar cell's electrical behavior under varying loads.
2. The load response illustrates how the cell's output varies with different resistances.
3. The spectral response indicates the sensitivity of the solar cell to various wavelengths

5. DETERMINATION OF MAGNETORESISTANCE OF SEMICONDUCTORS

Aim

To study the magnetic field dependence of the transverse magnetoresistance of a given semiconductor sample

Apparatus Required

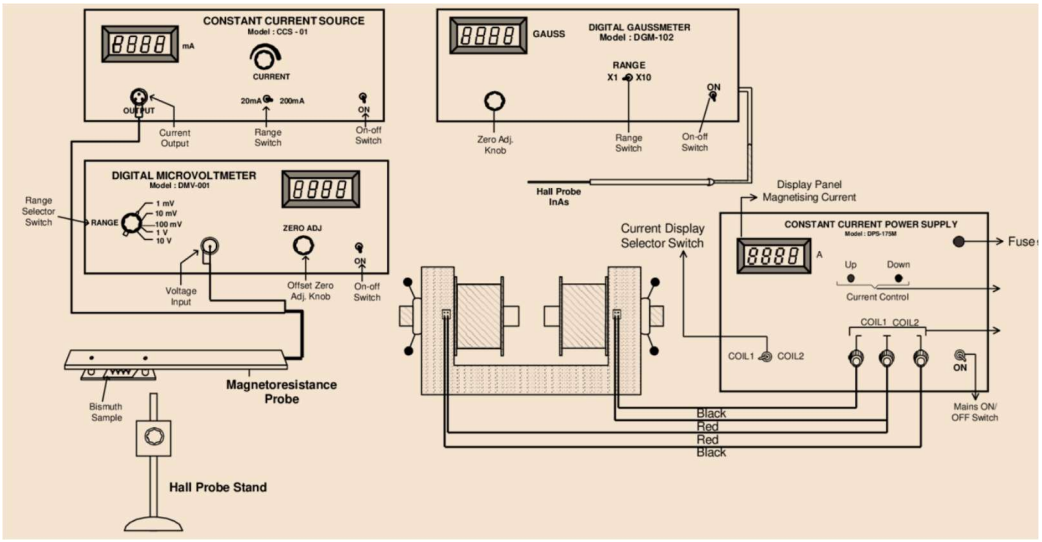
Four Probe Set-up, Sample (n-type germanium), Magnetoresistance Set-up, Electromagnet, Constant Power Supply, Digital Gauss meter.

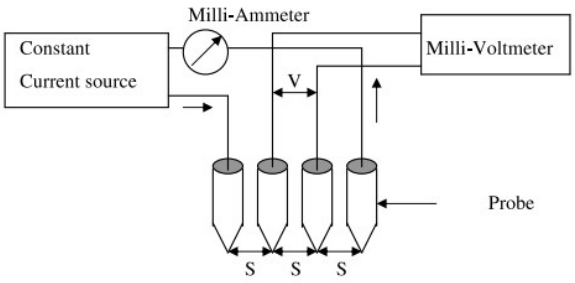
Formula

$$\frac{\Delta R}{R} = \frac{R_m - R}{R}$$

Where, R - Sample resistance without magnetic field (42.98 **Ω**) and Rm - Resistance of the sample with the magnetic field (**Ω**)

Experimental Setup





Four probe arrangement

Procedure

i. Set the pole piece distance of the electromagnet to nearly 19 mm.

ii. ii. Now place the hall probe of gauss meter the magnetic field as shown in figure 1 and switch on the electromagnet power supply and set it to maximum (4A). Rotate the Hall probe till it become perpendicular to magnetic field. Magnetic field will be maximum in this adjustment.

iii. Now lower the current in constant power supply to minimum and slowly increase the current and tabulate the magnetic field as in Table 1.

iv. Next unscrew the screws given at the top of probe to lower the case plate. Put the sample on the base plate of the four probe arrangement. Slowly screw both screws evenly to apply a very gentle pressure on the four spring probes. Check the continuity between the probes for proper electrical contacts.

v. Connect the outer pair of probes (red/black) leads to the current terminals and the inner pair (yellow and green lead) to the probe voltage terminals.

vi. Switch on the mains supply of magnetoresistance setup. Constant power supply and put the digital panel meter in the current measuring mode through the selector switch. In this position LED facing mA would glow. Adjust the current to a desired value

vii. Now put the digital panel meter in voltage measuring mode. In this position LED facing mV would glow and the meter would read the voltage between the probes.

viii. Now place the probe in the magnetic field as shown in figure 1. And switch on the electromagnet power supply and set it to maximum. Further rotate the magnetoresistance probe till it become perpendicular to magnetic field. Voltage will be maximum in this adjustment.

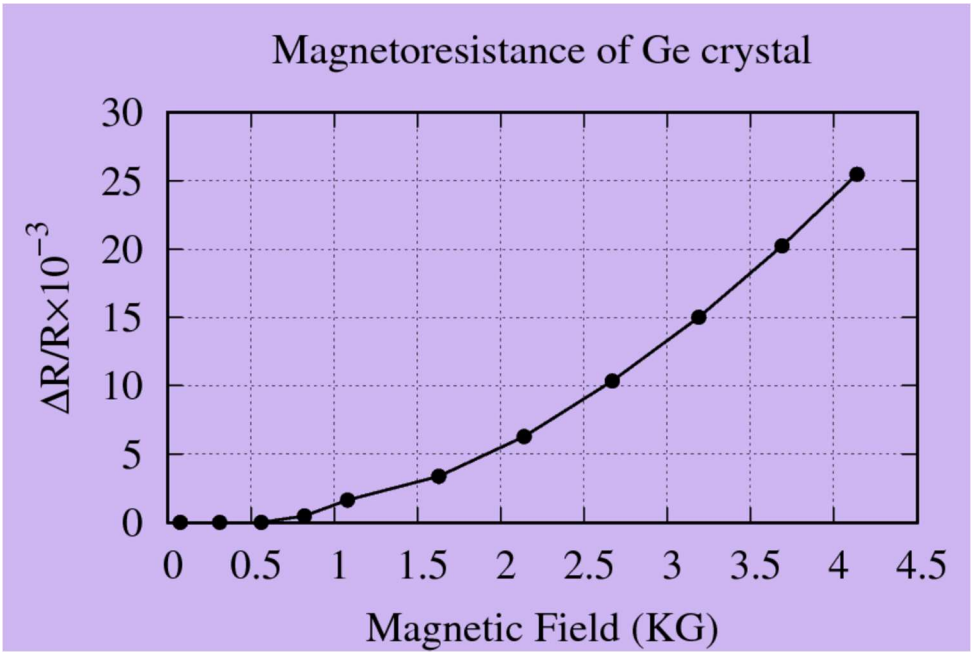
ix. Vary the magnetic field by varying the current step by step and note the change in voltage reading as in Table 1.

Observation

Table 1: Data for Magnetoresistance of n-Ge Probe Current I= -------mA

S. No.	Current (mA)	Magnetic Field (H) in KG	Voltage (Vm) in mV	$R_m = \frac{V_m}{I}$ in (Ω)	$\frac{\Delta R}{R}$	Log H	Log $\frac{\Delta R}{R}$

Model Graph



Result:

The dependence of resistivity with external applied magnetic field was studied

6. Determination of Dielectric Constant of a Liquid

Aim

To determine the dielectric constant (k) of a liquid

Apparatus Required

Op-Amp IC-741, cylindrical capacitor, capacitors, resistors, Jar containing liquid, dual power supply, signal generator, connecting wires, etc

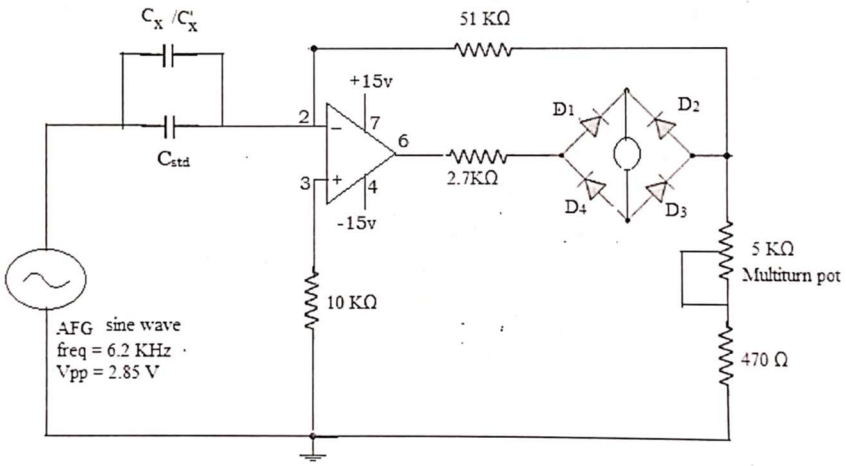
Formula

$$\text{Dielectric Constant } (k) = \frac{C_x^1}{C_x}$$

where C_x-Capacitance of unknown cylindrical capacitor with air as medium

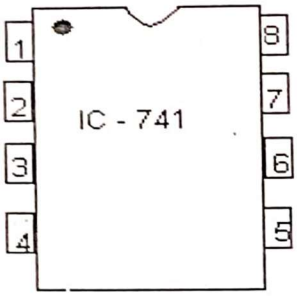
C^1_x-Capacitance of unknown cylindrical capacitor with liquid as medium

Circuit Diagram



IC - 741- Operational Amplifier, C_{std} - Known capacitance, C_x - unknown Cylindrical capacitor, R- Known Resistances.

PIN DIAGRAM of IC – 741 Op-amp:



1 & 5: offset
2: Inverting
3: Non-inverting
4: $-V_{cc}$
6: Output
7: $+V_{cc}$

Procedure:

1. Connect all circuit components as shown in circuit diagram using C_{std} capacitor with A and C points. (C_{AC} =500 pF is used).

2. Turn on dual power supply with +15 V and -15V.

3. Turn on AFG. In this unit choose sine function. Set frequency 6.2 KHz, set input voltage, V_{rms} =1 V.(V_p =1.425 V or V_{FF} =2.85 V).

4. Measurement of Unknown Capacitance (C_x) in Air Medium

i. **Calibration Using Standard Capacitors (Cstd):**
   a. Connect Cstd (C_{AC}=500 pF) between points P and Q.
   b. Record deflection (θ_1) on the microammeter.
   c. Repeat the process using C_{AD}=333 pF, C_{AE}=250 pF, C_{AF}=200 pF, and C_{AG}=166 pF, recording $\theta_1$in each case.
   d. This ensures linearity of calibration. Turn off the AFG (Arbitrary Function Generator) after calibration.

ii. **Measuring Unknown Capacitance (C_x):**
   a. Connect C_x (unknown capacitor) between points A and D. This places C_x in parallel with C_{AD}.
   b. Connect points A and D to P and Q. Turn on the AFG with the same settings and record θ_2.
   c. Calculate the total capacitance and determine C_x from it.

iii. **Repeat the Process for Different Standard Capacitors:**
   a. Bring C_x in parallel with C_{AE}, C_{AF}, and C_{AG} respectively.
   b. In each case, connect the combination to points P and Q, record θ_2, and determine C_x.
   c. Turn off the AFG after recording data.

iv. **Convert Deflections (θ_1 and θ_2):**
   a. Use the recorded θ_1 and θ_2 values to calculate capacitances in pF.
   b. Average the values of C_x obtained from different configurations for better accuracy.

5. Measurement of unknown capacitance in dielectric medium C'_x – Here the deflection is noted as θ_3

Observation

S.No	When P and Q are connected to	Calibration of standard capacitors without connecting any capacitor in parallel		Measurement of C_x in air medium with C_x parallel to C_{std}		Measurement of C'_x in dielectric medium with C'_x parallel to C_{std}	
		Deflection θ_1	Value of $C_{std}= \theta_1 x10$ (pF)	Deflection θ_2	Value of $C_x= (\theta_2-\theta_1)$ x10 (pF)	Deflection θ_3	Value of $C'_x= (\theta_2-\theta_1)$ x10 (pF)

Average C_x = ---------pF C'_x = ---------pF

Calculation:

$$(k) = \frac{C_x^1}{C_x}$$

Result

Dielectric constant of a given liquid k = ----------

7. Characteristics of Phototransistor

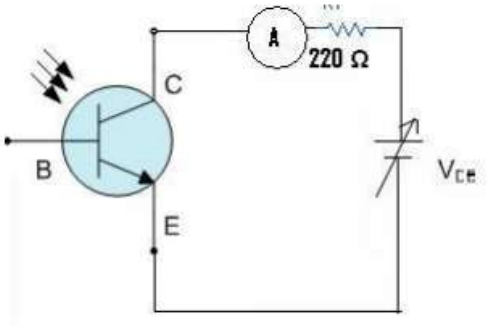
AIM:

To obtain the V-I characteristics of the given photo transistor.

APPARATUS:

Photo transistor IR 3MM 935NM, R.P.S (O-30V) 2Nos, Resistors 220 ohm, Bread board and connecting wires

CIRCUIT DIAGRAM:



PROCEDURE:

1. Connect the circuit as per the circuit diagram.

2. Keep the input light excitation fixed. Then vary the Vce in steps of 1V till the maximum voltage rating of the transistor is reached and then note down the corresponding values of Ic.

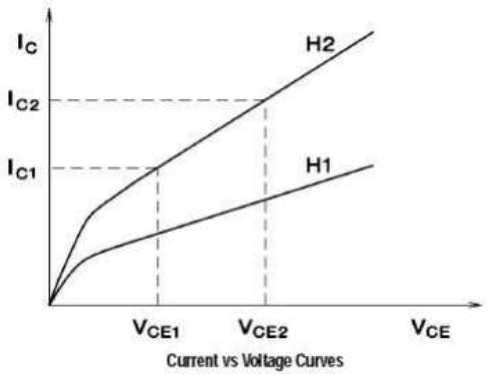
3. Tabulate the readings. For various values of input excitation record the values of Vce and Ic and plot the characteristics of the photo transistor.

OBSERVATIONS:

V-I Characteristics:

V_{ce} (V)	I_c (mA)

MODEL GRAPH:



Current vs Voltage Curves

Result

V-I characteristics of photo transistor is studied for various excitation

1. A. ADDITION OF TWO 8-BIT AND TWO 16-BIT NUMBERS USING 8085 MICROPROCESSOR

Aim

To perform the addition of two 8-bit numbers and two 16-bit numbers using 8085 microprocessor assembly language programming.

Apparatus Required

8085 Microprocessor Trainer Kit / Emulator Software, Power Supply (5V DC), Hexadecimal Keypad and Display Interface.

Algorithm

8-bit Addition:

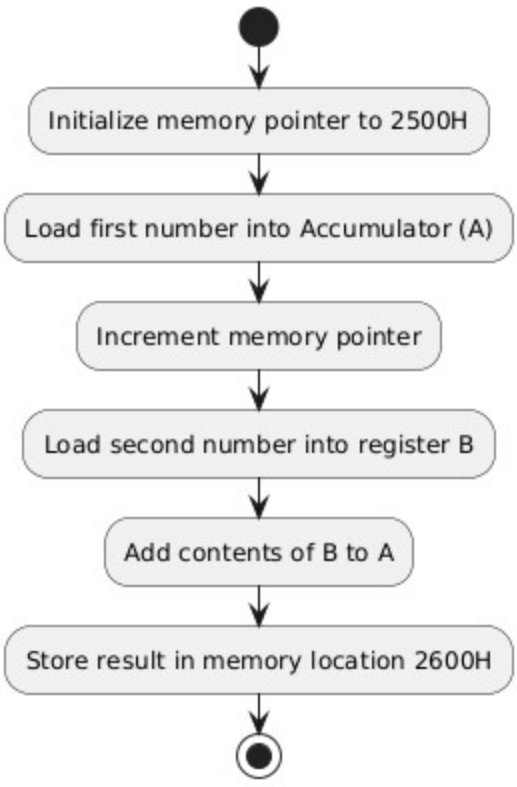
1. Load the first number into the accumulator.
2. Add the second number to the accumulator using the ADD instruction.
3. Store the result.
4. Check the carry and store it separately.

16-bit Addition:

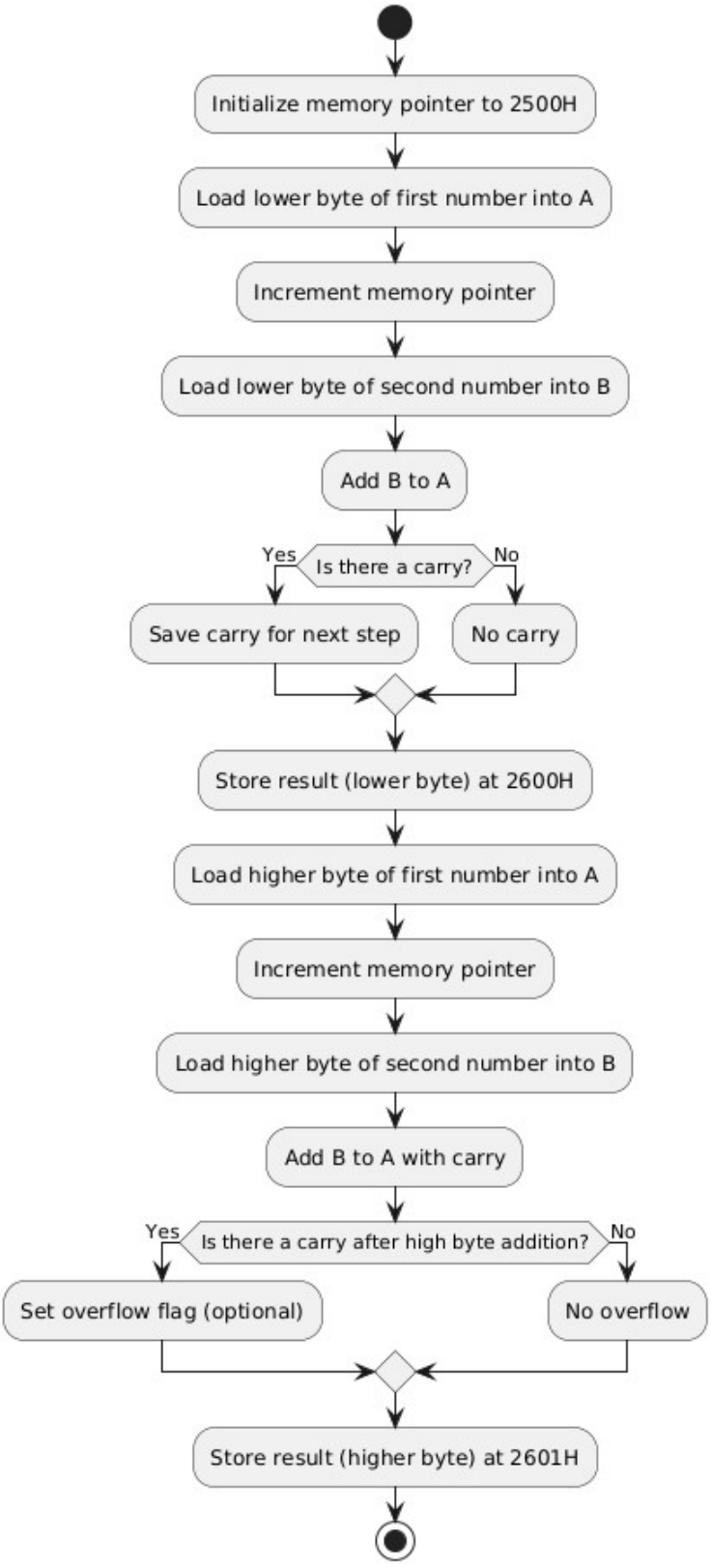
1. Load the lower bytes of both numbers.
2. Perform 8-bit addition for the lower bytes and save the result.
3. Add the higher bytes with carry using ADC.
4. Store the result and the carry.

Flow Chart

8-bit Addition



16 – bit Addition



Program

8-bit Addition

Address	Mnemonic	Operand	Comment
2000H	LXI	H, 2500H	Load memory pointer to 2500H
2003H	MOV	A, M	Load first number into Accumulator (A)
2004H	INX	H	Increment memory pointer
2005H	MOV	B, M	Load second number into register B
2006H	ADD	B	Add B to A
2007H	STA	2600H	Store the result at memory location 2600H
200AH	HLT		Halt program execution

Example Input Output for 8-bit Addition

Memory Address	Data	Comment
2500H	45H	First number (69 in decimal)
2501H	36H	Second number (54 in decimal)
Output	**Result**	**Comment**
2600H	7BH	Result of addition (123 in decimal)

16-bit Addition

Address	Mnemonic	Operand	Comment
3000H	LXI	H, 2500H	Load memory pointer to 2500H
3003H	MOV	A, M	Load lower byte of the first number
3004H	INX	H	Increment memory pointer
3005H	MOV	B, M	Load lower byte of the second number
3006H	ADD	B	Add B to A
3007H	STA	2600H	Store the lower byte of the result
300AH	MOV	C, A	Save carry in C
300BH	INX	H	Increment memory pointer
300CH	MOV	A, M	Load higher byte of the first number
300DH	INX	H	Increment memory pointer
300EH	MOV	B, M	Load higher byte of the second number
300FH	ADC	B	Add B to A with carry
3010H	STA	2601H	Store the higher byte of the result
3013H	HLT		Halt program execution

Example Input Output for 16-bit Addition

Memory Address	Data	Comment
2500H	34H	Lower byte of first number (52)
2501H	12H	Higher byte of first number (18)
2502H	56H	Lower byte of second number (86)
2503H	03H	Higher byte of second number (3)
Output	**Result**	**Comment**
2600H	8AH	Lower byte of result (138)
2601H	15H	Higher byte of result (21)

1. B. SUBTRACTION OF 8-BIT AND 16-BIT NUMBERS USING 8085 MICROPROCESSOR

Aim

To understand the subtraction operation in the 8085 microprocessor.

To perform the subtraction of two 8-bit numbers.

To perform the subtraction of two 16-bit numbers.

Apparatus Required

8085 Microprocessor Trainer Kit / Emulator Software, Power Supply (5V DC), Hexadecimal Keypad and Display Interface.

Algorithm for 8-bit Subtraction

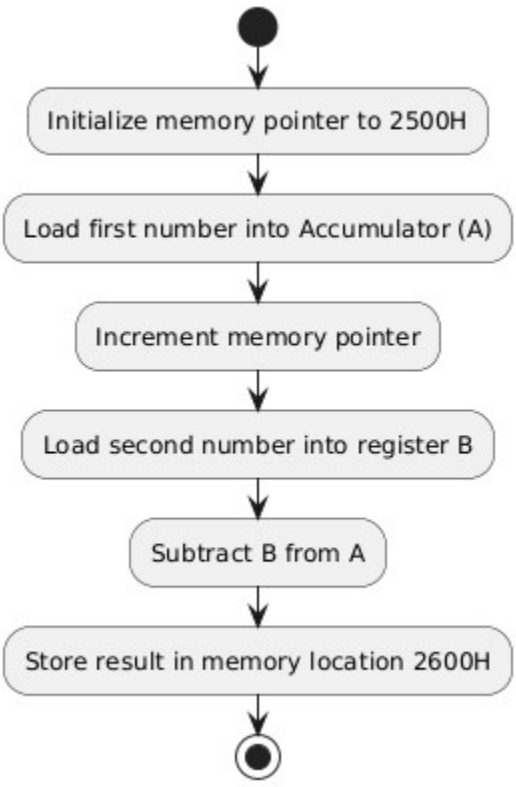
1. Start the program.
2. Initialize the memory pointer to the starting address of the input numbers (e.g., 2500H).
3. Load the first 8-bit number into the accumulator (A) from the memory.
4. Increment the memory pointer to the next location.
5. Load the second 8-bit number into a register (e.g., B).
6. Perform subtraction using the SUB instruction, which subtracts the contents of register B from A.
7. Store the result in a specified memory location (e.g., 2600H).
8. Stop the program.

Algorithm for 16-bit Subtraction

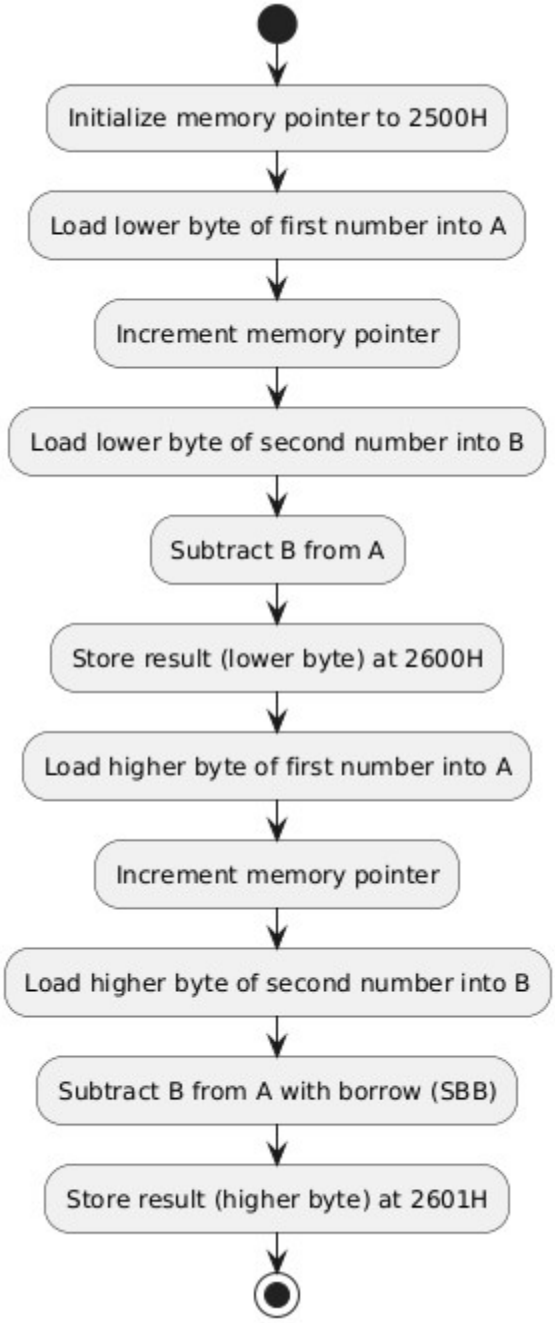
1. Start the program.
2. Initialize the memory pointer to the starting address of the input numbers (e.g., 2500H).
3. Load the lower byte of the first number into the accumulator (A) from memory.
4. Increment the memory pointer to the next location.
5. Load the lower byte of the second number into a register (e.g., B).
6. Perform subtraction using the SUB instruction.
7. Store the lower byte of the result at the specified memory location (e.g., 2600H).
8. Save the borrow flag in a register (e.g., C) if a borrow occurs.
9. Increment the memory pointer to load the higher bytes of the first and second numbers.
10. Perform subtraction of the higher bytes using the SBB instruction (subtract with borrow).
11. Store the higher byte of the result at the specified memory location (e.g., 2601H).
12. Stop the program.

Flow Chart

8-bit Subtraction



16-bit Subtraction



Program

8-bit Subtraction

Address	Mnemonic	Operand	Comment
8000H	LXI	H, 8000H	Load address of the first number into HL pair.
8003H	MOV	A, M	Move the first number into the accumulator.
8004H	INX	H	Increment HL to point to the second number.
8005H	SUB	M	Subtract the second number from accumulator.
8006H	MOV	M, A	Store the result at memory location 8002H.
8007H	HLT		Halt the program.

Example Input Output for 8-bit Subtraction

Memory Address	Content	Description
8000H	35H	First number (53 in decimal).
8001H	12H	Second number (18 in decimal).
Output (8002H)	23H	Result (53 - 18 = 35 in decimal).

16-bit Subtraction

Address	Mnemonic	Operand	Comment
8000H	LXI	H, 8000H	Load address of the first 16-bit number into HL.
8003H	MOV	A, M	Load the lower byte of the first number into A.
8004H	INX	H	Increment HL to point to the higher byte.
8005H	MOV	B, M	Load the higher byte of the first number into B.
8006H	INX	H	Increment HL to point to the second number.
8007H	MOV	C, M	Load the lower byte of the second number into C.
8008H	INX	H	Increment HL to point to the higher byte.
8009H	MOV	D, M	Load the higher byte of the second number into D.
800AH	MOV	A, C	Move the lower byte of the second number into A.
800BH	SUB	M	Subtract the lower bytes.
800CH	MOV	M, A	Store the lower byte of the result at 8004H.
800DH	INX	H	Increment HL to point to the higher byte.
800EH	SBB	B	Subtract the higher bytes with borrow.
800FH	MOV	M, A	Store the higher byte of the result at 8005H.
8010H	HLT		Halt the program.

Example Input Output for 16-bit Subtraction

Memory Address	Content	Description
8000H	34H	Lower byte of the first 16-bit number.
8001H	12H	Higher byte of the first 16-bit number.
8002H	12H	Lower byte of the second 16-bit number.
8003H	0FH	Higher byte of the second 16-bit number.
Output (8004H)	22H	Lower byte of the result (802 in decimal).
Output (8005H)	03H	Higher byte of the result (802 in decimal).

1. C. MULTIPLICATION OF 8-BIT AND 16-BIT NUMBERS USING 8085 MICROPROCESSOR

Aim

To understand the multiplication operation in the 8085 microprocessor.

To perform the multiplication of two 8-bit numbers.

To perform the multiplication of two 16-bit numbers.

Apparatus Required

8085 Microprocessor Trainer Kit / Emulator Software, Power Supply (5V DC), Hexadecimal Keypad and Display Interface.

Algorithm

8-bit Multiplication

1. Load the first 8-bit number into the accumulator (A).
2. Load the second 8-bit number into register C (serves as the counter).
3. Clear register B to accumulate the result (set B = 00H).
4. Repeat the following steps until the counter C becomes zero:
    - Add the accumulator value (A) to register B.
    - Decrement the counter (C).
5. Store the result from B into the memory.
6. Halt the program.

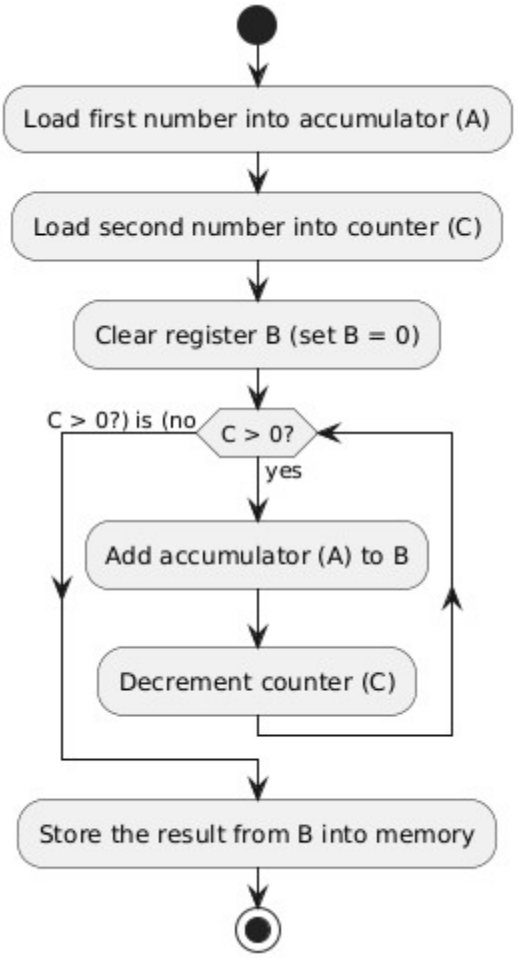
16-bit Multiplication

1. Load the lower byte of the first 16-bit number into register E.

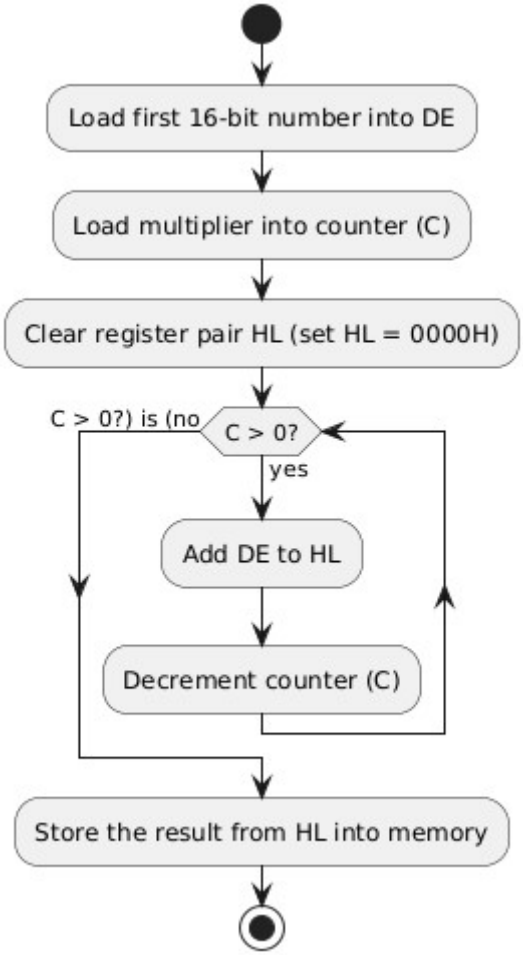
2. Load the higher byte of the first 16-bit number into register `D`.
3. Load the multiplier (8-bit number) into the counter register `C`.
4. Clear `HL` to accumulate the result (set `HL = 0000H`).
5. Repeat the following steps until the counter `C` becomes zero:
    - Add the 16-bit number (`DE`) to the accumulator pair (`HL`).
    - Decrement the counter (`C`).
6. Store the result from `HL` into memory.
7. Halt the program.

Flow Chart

8-bit Multiplication



16-bit Multiplication



Program

8-bit Multiplication

Address	Mnemonic	Operand	Comment
8000H	LXI	H, 8000H	Load memory address of the first number.
8003H	MOV	A, M	Load the first number into accumulator.
8004H	INX	H	Increment address to point to the second number.
8005H	MOV	C, M	Load the second number into C.
8006H	MVI	B, 00H	Clear register B to accumulate result.
8008H	LOOP: ADD	A	Add the first number (in A) to B.
8009H	MOV	B, A	Accumulate the result in B.
800AH	DCR	C	Decrement counter C.
800BH	JNZ	LOOP	Repeat until C becomes zero.
800CH	STA	8002H	Store the result at memory location 8002H.
800DH	HLT		Halt the program.

Input-Output Table (8-Bit Multiplication)

Memory Address	Content	Description
8000H	04H	First number (4 in decimal).
8001H	03H	Second number (3 in decimal).
Output (8002H)	0CH	Result (4 × 3 = 12 in decimal).

16-bit Multiplication

Address	Mnemonic	Operand	Comment
8000H	LXI	H, 8000H	Load address of the first 16-bit number.
8003H	MOV	E, M	Load the lower byte of the first number into E.
8004H	INX	H	Increment address to point to the higher byte.
8005H	MOV	D, M	Load the higher byte of the first number into D.
8006H	INX	H	Increment address to point to the multiplier.
8007H	MOV	C, M	Load the multiplier into C.
8008H	LXI	H, 0000H	Clear HL (used to accumulate result).
800BH	LOOP: DAD	D	Add the 16-bit number in DE to HL.
800CH	DCR	C	Decrement counter C.
800DH	JNZ	LOOP	Repeat until C becomes zero.
800EH	SHLD	8004H	Store the result at memory location 8004H-8005H.
800FH	HLT		Halt the program.

Input-Output Table (16-Bit Multiplication)

Memory Address	Content	Description
8000H	12H	Lower byte of first 16-bit number (18).
8001H	01H	Higher byte of first 16-bit number (1).
8002H	03H	Multiplier (3 in decimal).
Output (8004H)	36H	Lower byte of result (54 in decimal).
Output (8005H)	00H	Higher byte of result (0 in decimal).

Result

Arithmetic operation such as Addition, Subtraction and Multiplication of two 8-bit and 16-bit numbers are executed successfully using 8085 microprocessors and the outputs are verified.

2. DATA MANIPULATION - A) ARRANGE THE GIVEN DATA ITEMS IN ASCENDING ORDER

Aim

To arrange the given data into ascending order using 8085 microprocessor

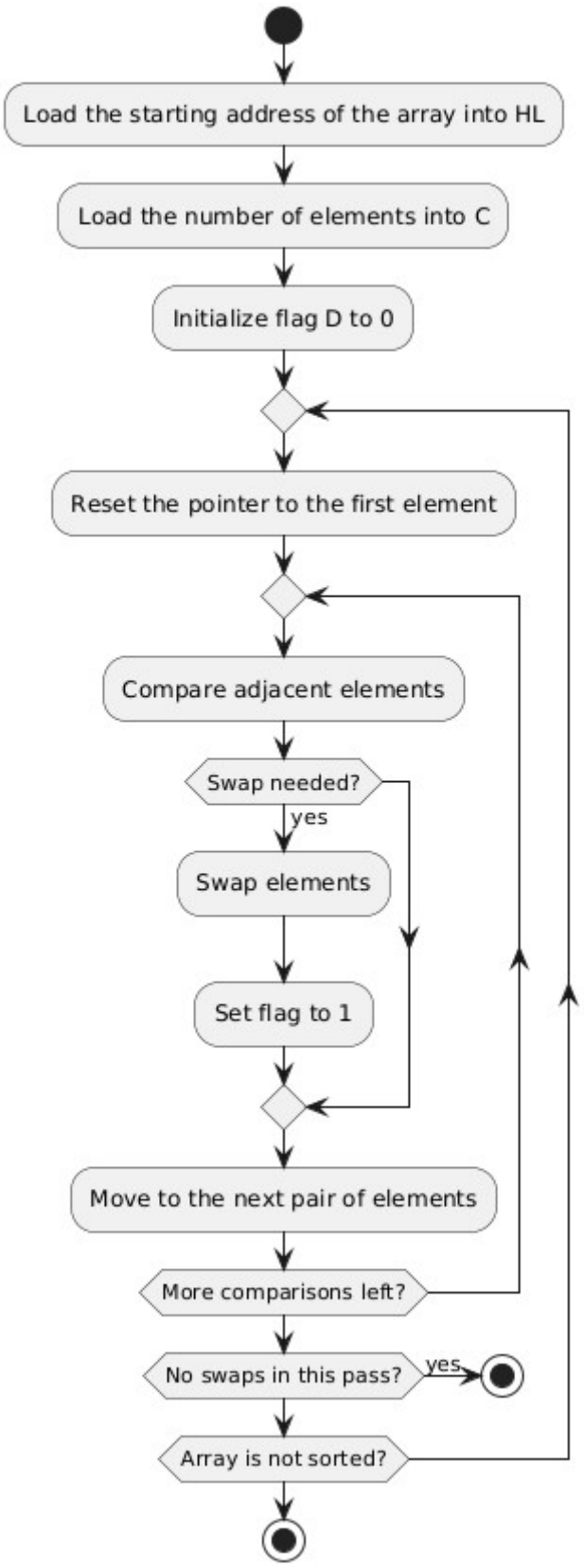
Apparatus Required

8085 Microprocessor Trainer Kit / Emulator Software, Power Supply (5V DC), Hexadecimal Keypad and Display Interface

Algorithm

1. Initialize Registers:
2. Load the starting address of the array into the HL pair.
3. Load the total number of elements in the array into register C.
4. Outer Loop:
   Set a flag to indicate if any swapping occurred during the pass.
5. Inner Loop:
   Compare adjacent elements.
   Swap them if they are not in the desired order (ascending or descending).
   If a swap occurs, set the flag.
6. Repeat Outer Loop:
   If a swap occurred during the previous pass, repeat the process.
   If no swap occurred, the array is sorted.
7. Store Result:
8. Store the sorted array back in memory.
9. Halt the Program:
   End the program using HLT

Flow Chart


Load the starting address of the array into HL
Load the number of elements into C
Initialize flag D to 0
Reset the pointer to the first element
Compare adjacent elements
Swap needed?
yes
Swap elements
Set flag to 1
Move to the next pair of elements
More comparisons left?
No swaps in this pass?
yes
Array is not sorted?

Program

Ascending Order

Address	Mnemonic	Operand	Comment
8000H	LXI	H, 8002H	Load the starting address of the array.
8003H	MOV	C, M	Load the number of elements into register C.
8004H	DCR	C	Decrement C as array size - 1 is needed.
8005H	MVI	D, 00H	Initialize flag D to 0 (no swaps yet).
8006H	OUTER: LXI	B, 0000H	Reset inner loop pointer to array start.
8009H	MOV	A, M	Load the first element into A.
800AH	INX	H	Point to the next element.
800BH	CMP	M	Compare with the next element.
800CH	JC	NOSWAP	Jump if in the correct order for ascending.
800FH	MOV	E, M	Swap: Load the second element into E.
8010H	MOV	M, A	Move the first element to the second location.
8011H	DCX	H	Point to the first element again.
8012H	MOV	M, E	Store the second element in the first location.
8013H	MVI	D, 01H	Set flag D to indicate a swap occurred.
8014H	NOSWAP: INX	H	Move to the next pair of elements.
8015H	INX	B	Increment the loop counter.
8016H	DCR	C	Decrement the counter.
8017H	JNZ	OUTER	If more comparisons needed, repeat outer loop.
8018H	MOV	A, D	Check if any swaps occurred.
8019H	CPI	00H	If no swaps, array is sorted.
801AH	JZ	DONE	Jump to halt if sorted.
801BH	JMP	8006H	Otherwise, repeat the outer loop.
801EH	DONE: HLT		Halt the program.

Example Input and Output Table

Input (Memory)

Memory Address	Content	Description
8002H	05H	Number of elements (5).
8003H	3CH	First data element (60).
8004H	29H	Second data element (41).
8005H	6AH	Third data element (106).
8006H	11H	Fourth data element (17).

Memory Address	Content	Description
8007H	4FH	Fifth data element (79).

Output (Sorted in Ascending Order)

Memory Address	Content	Description
8003H	11H	First data element (17).
8004H	29H	Second data element (41).
8005H	3CH	Third data element (60).
8006H	4FH	Fourth data element (79).
8007H	6AH	Fifth data element (106).

2. DATA MANIPULATION - b) FINDING THE MINIMUM VALUE IN THE GIVEN DATA SET.

Aim

To find the minimum value in the given data set using 8085 microprocessor

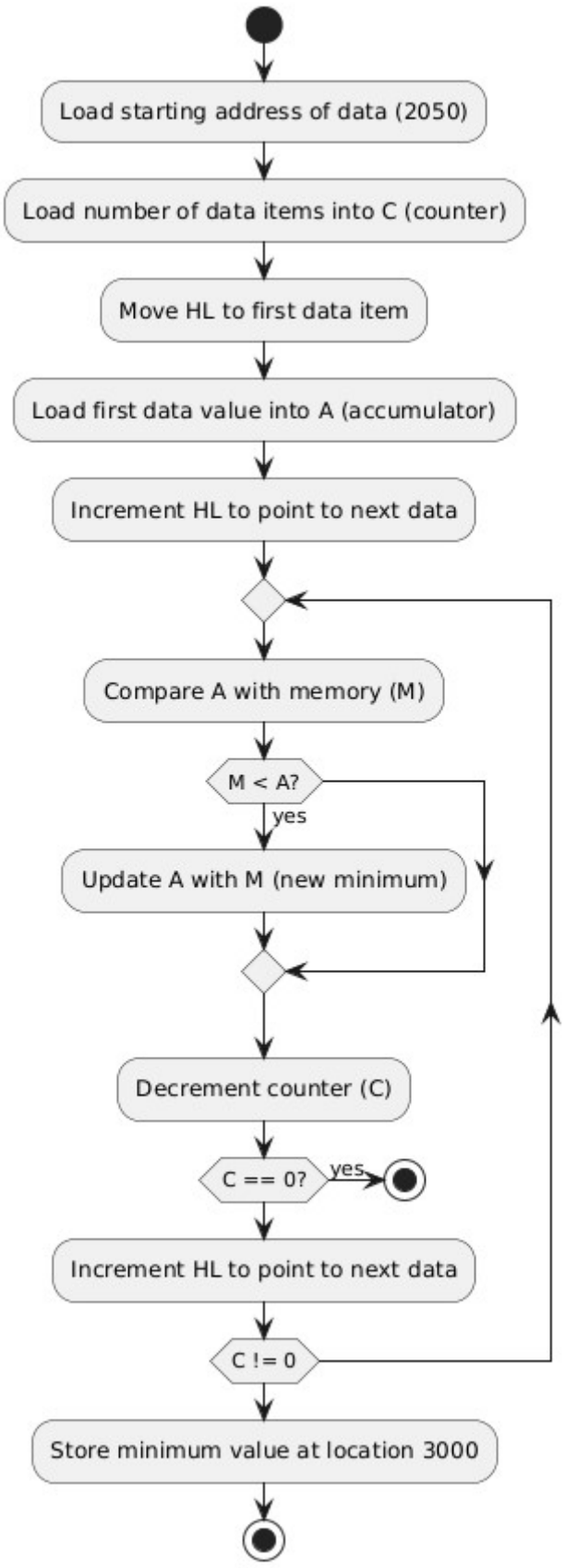
Apparatus Required

8085 Microprocessor Trainer Kit / Emulator Software, Power Supply (5V DC), Hexadecimal Keypad and Display Interface

Algorithm

1. Start the program.
2. Load the starting address of the dataset into the HL register pair.
3. Load the number of elements in the dataset into a counter register (C).
4. Move HL to point to the first data item.
5. Load the first data value into the accumulator (A). This becomes the initial minimum.
6. Increment HL to point to the next data item.
7. Repeat the following steps until the counter (C) becomes zero:
8. Compare the current value in memory (pointed by HL) with the value in the accumulator (A).
9. If the memory value is smaller, update the accumulator (A) with this new value.
10. Decrement the counter (C).
11. Increment HL to point to the next data item.
12. Store the minimum value from the accumulator (A) into a predefined memory location.
13. End the program.

Flow Chart


Load starting address of data (2050)
Load number of data items into C (counter)
Move HL to first data item
Load first data value into A (accumulator)
Increment HL to point to next data
Compare A with memory (M)
M < A?
yes
Update A with M (new minimum)
Decrement counter (C)
C == 0?
yes
Increment HL to point to next data
C != 0
Store minimum value at location 3000

Program

Address	Mnemonic	Operand	Comments
2000	LXI H	2050	Load starting address of data into HL pair
2003	MOV C	M	Load the number of data elements into C
2004	INX H		Increment HL to point to first data item
2005	MOV A	M	Load first data value into accumulator
2006	INX H		Increment HL to point to next data item
2007	DCR C		Decrement count (C)
2008	JZ	2011	If count is zero, jump to end of loop
200B	CMP M		Compare A with memory (M)
200C	JC	200F	If memory (M) is smaller, jump to store
200E	INX H		Increment HL to point to next data item
200F	MOV A	M	Load memory (M) into A (new minimum)
2010	JMP	2007	Jump back to loop start
2011	STA	3000	Store minimum value at address 3000
2014	HLT		Halt the program

Input Table

Memory Location	Content	Description
2050	04	Number of elements in the dataset
2051	10	First data item
2052	25	Second data item
2053	05	Third data item
2054	15	Fourth data item

Output Table

Memory Location	Content	Description
3000	05	Minimum value from the dataset

The maximum value in the data set can be found by changing the conditional jump line i.e one can use `JNC` instead of `JC` for comparison line

2. C. DATA MANIPULATION - SEARCH OF A GIVEN NUMBER IN THE GIVEN DATA SET

Aim

To Search of a given number in the given data set using 8085 microprocessor

Apparatus Required

8085 Microprocessor Trainer Kit / Emulator Software, Power Supply (5V DC), Hexadecimal Keypad and Display Interface

Algorithm

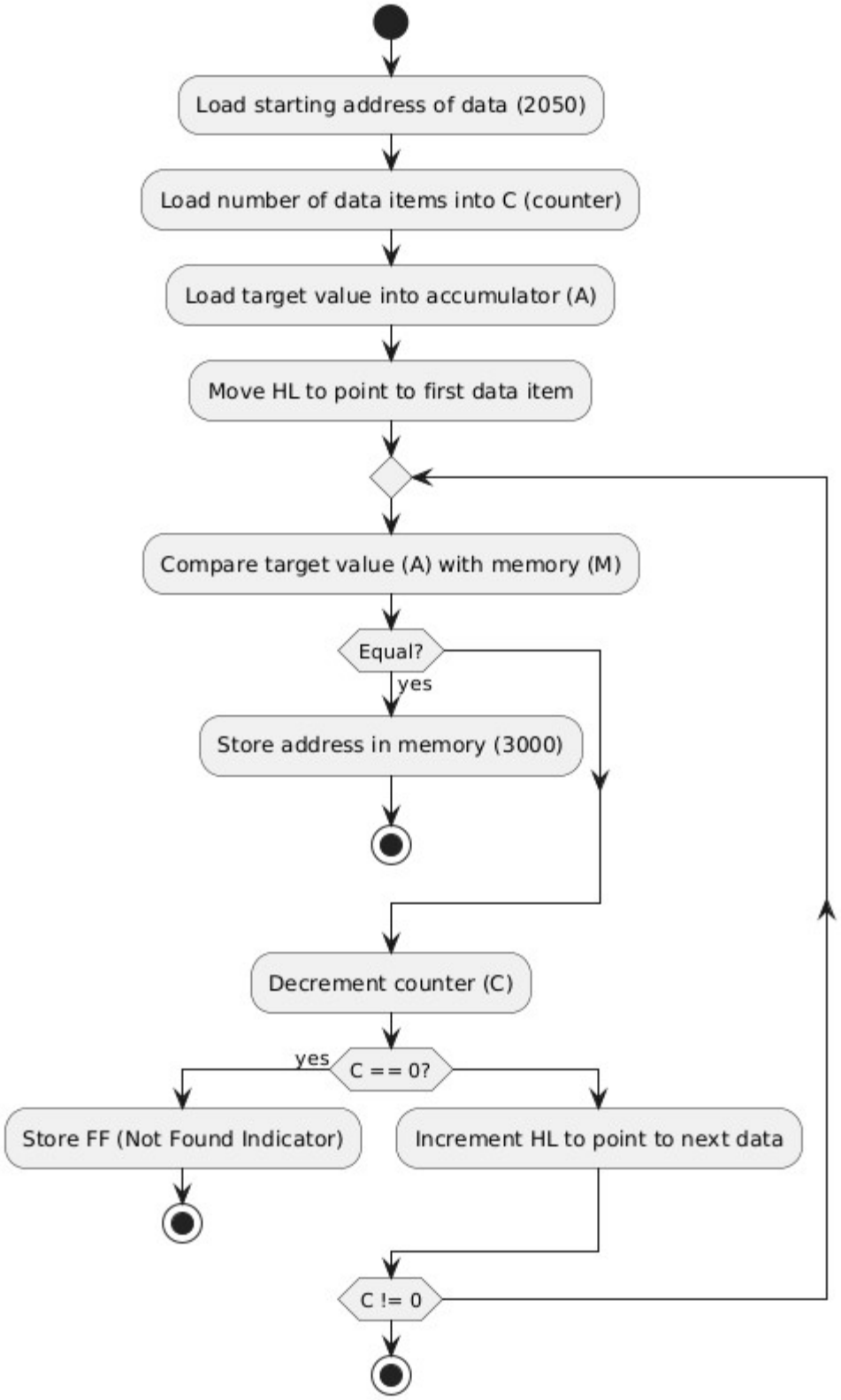
1. **Start** the program.
2. Load the starting address of the dataset into the HL register pair.
3. Load the number of elements in the dataset into the counter register (C).
4. Load the target value (to be searched) into the accumulator (A).
5. Increment HL to point to the first data item.
6. **Repeat the following steps until the counter (C) becomes zero:**
    - Compare the value in the accumulator (A) with the memory value (M) pointed to by HL.
    - If they are equal, store the address of the match and jump to the end.
    - Decrement the counter (C).
    - Increment HL to point to the next data item.
7. If the counter reaches zero, indicate that the value is not found.
8. **End** the program

Program

Address	Mnemonic	Operand	Comments
2000	LXI H	2050	Load starting address of the dataset
2003	MOV C	M	Load the number of data elements into C
2004	INX H		Increment HL to point to the target value
2005	MOV A	M	Load the target value into accumulator
2006	INX H		Increment HL to point to first data item
2007	DCR C		Decrement counter
2008	JZ	2013	If count is zero, jump to "Not Found"
200B	CMP M		Compare A with memory (M)
200C	JZ	2015	If equal, jump to "Found"
200F	INX H		Increment HL to point to next data item
2010	JMP	2007	Jump back to loop start
2013	MVI A	FF	Load FF (Not Found Indicator) into A
2014	JMP	2017	Jump to end of program
2015	MOV A	H	Store HL high byte (address) in A
2016	STA	3000	Store found address at memory location

Address	Mnemonic	Operand	Comments
2017	HLT		Halt the program

Flow Chart



Input and Output Table

Example 1: Target Value Found

Memory Location	Content	Description
2050	05	Number of elements in the dataset
2051	15	Target value to search
2052	10	First data item
2053	25	Second data item
2054	15	Third data item
2055	30	Fourth data item
2056	40	Fifth data item

Output

Memory Location	Content	Description
3000	2054	Address of the target value

Example 2: Target Value Not Found

Memory Location	Content	Description
2050	04	Number of elements in the dataset
2051	50	Target value to search
2052	10	First data item
2053	20	Second data item
2054	30	Third data item
2055	40	Fourth data item

Output

Memory Location	Content	Description
3000	FF	Indicator that the value was not found

Result

Assembly language programs for data Manipulation such as Arranging the given data items in Ascending order, Finding the Minimum value in the given data set and Search of a given number in the given data set are executed successfully using 8085 microprocessor and ouput are verified.

3. System Call and Rolling Character

Aim

a. To calculate the time delay for a given interval using 8085 microprocessor instructions.
b. To roll a given character from Left to Right or Right to Left on 7-segment displays with a specified time interval.

Apparatus Required:

8085 Microprocessor Kit, Power Supply (5V), 7-Segment Display. Connecting Wires
Stopwatch (optional, for time delay verification)

Theory:

1. **System Call (Delay Calculation):**
   The 8085 microprocessor does not have an internal timer. Time delays are generated using loop instructions. The delay duration depends on the clock frequency of the microprocessor and the number of cycles consumed by instructions in the loop.

   Time delay formula:

$$T_{delay} = (n \times C \times T_{clock})$$

   Where:

   n: Number of iterations in the loop
   C: Number of clock cycles per iteration

$$T_{clock} = \frac{1}{ClockFrequency}$$

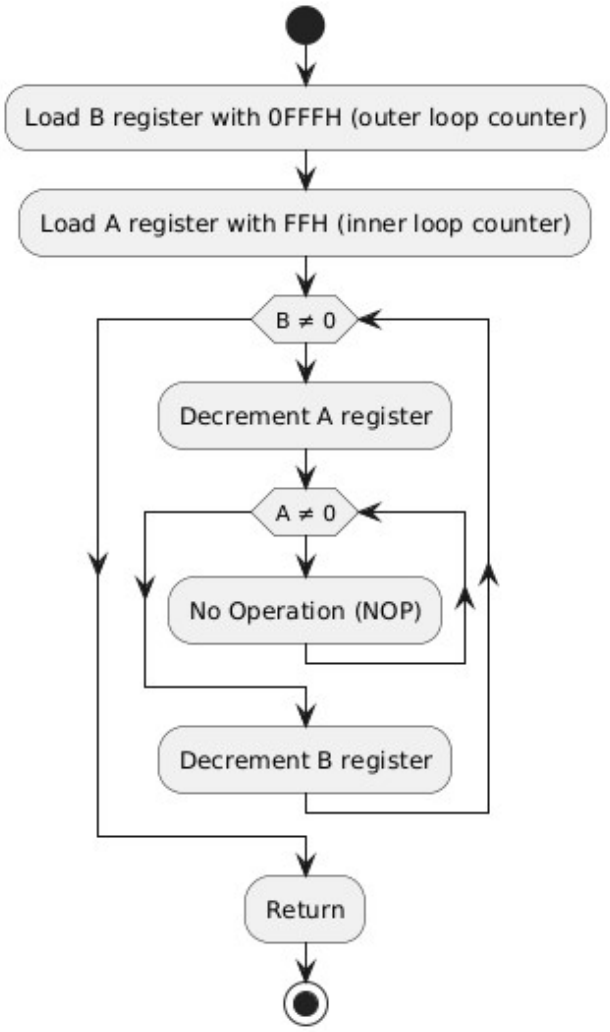
Algorithm

a. Time Delay Calculation

1. Load the register with a counter value.
2. Use nested loops for larger delays.
3. Repeat decrementing the counter until it reaches zero.
4. Return to the main program after completing the delay.

Flow chart

a. Time Delay Calculation



Program Time Delay Calculation

Address	Mnemonic	Operand	Comment
4000H	LXI	B, 0FFFH	Load outer loop counter (0FFFH)
4003H	DELAY		Label for delay loop
4004H	MOV	C, A	Move Accumulator to C-register
4006H	MVI	A, FFH	Load A-register with FFH (inner loop counter)
4008H	INNER		Label for inner loop
4009H	NOP		No operation, 4 T-cycles delay
400AH	DCR	A	Decrement A-register

Address	Mnemonic	Operand	Comment
400BH	JNZ	INNER	Jump to INNER if A ≠ 0
400CH	DCR	B	Decrement B-register (outer loop counter)
400DH	JNZ	DELAY	Jump to DELAY if B ≠ 0
400EH	RET		Return from the program

Input and Output: Time Delay Calculation

Input: This program does not take any external input from the user. It works by generating a time delay based on the internal loop count.

Output: The program will generate a time delay, typically in the millisecond range, which can be used for other operations that require timing in embedded systems.

b. Rolling Character on 7-segment display

Theory:

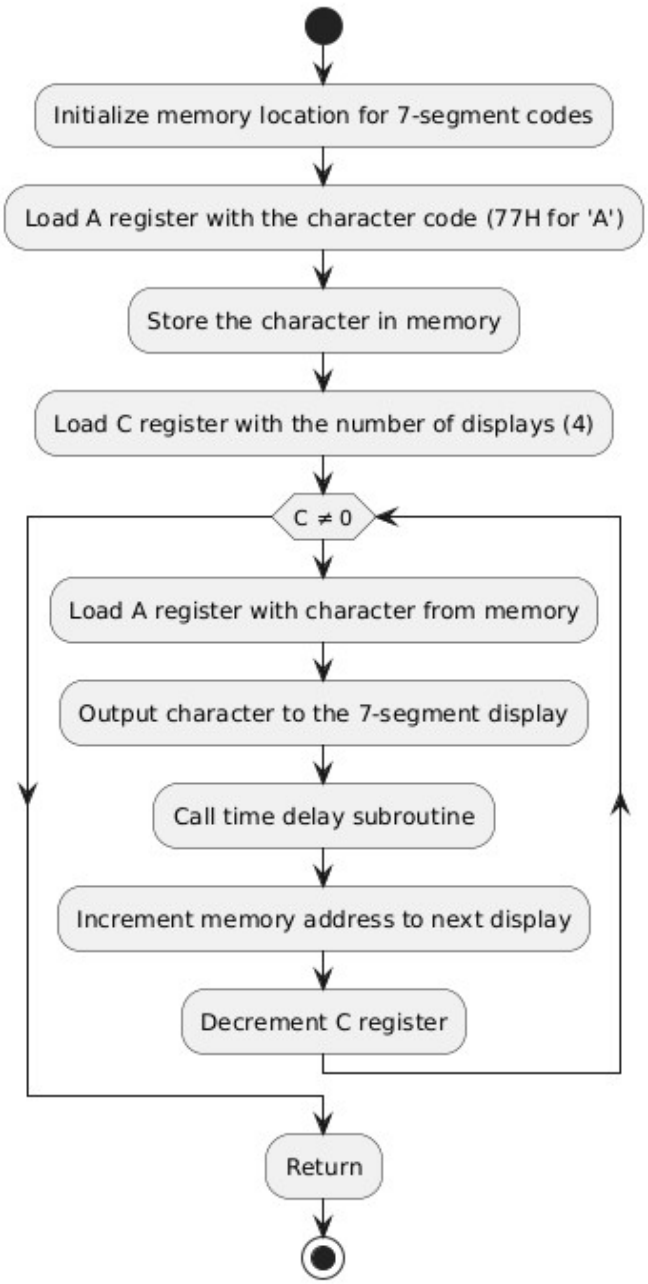
Rolling Character:

A character is displayed on 7-segment displays by lighting the appropriate segments. Rolling means shifting the character from one display to the next (Left-to-Right or Right-to-Left) after a time interval. This is achieved by refreshing each display in sequence with a delay in between.

Algorithm:

1. Initialize the data for the character to be displayed.
2. Set up ports for output to 7-segment displays.
3. Load the character data and output it to the first display.
4. Introduce a delay, then shift the character to the next display.
5. Repeat the process for Left-to-Right or Right-to-Left rolling.

Flow chart: b. Rolling Character



Program Rolling Character on 7-Segment Display (8085 Microprocessor)

Address	Mnemonic	Operand	Comment
4000H	LXI	H, 4000H	Initialize memory address for 7-segment codes

Address	Mnemonic	Operand	Comment
4003H	MVI	A, 77H	Load A-register with the code for character 'A' (7-segment encoding)
4005H	MOV	M, A	Store the character code in memory
4007H	MVI	C, 04H	Load C-register with number of displays (4)
4009H	ROLL		Label for rolling loop
400AH	MOV	A, M	Load character code into A-register
400BH	OUT	00H	Output data to the first 7-segment display
400CH	CALL	DELAY	Call the time delay subroutine
400EH	INX	H	Increment H to point to next 7-segment display
400FH	DCR	C	Decrement display counter
4010H	JNZ	ROLL	Repeat if C ≠ 0
4012H	RET		Return from the program

Input and Output:

Input: The program takes an ASCII character (e.g., 'A') which corresponds to a specific 7-segment display code (in this case, 77H for 'A').

Output: The character is displayed on multiple 7-segment displays, rolling either from left to right or right to left, based on the logic of the program. The character will appear sequentially on each display with a specified time interval in between.

Result

The experiment demonstrates the use of 8085 microprocessor instructions to calculate time delays and control 7-segment displays for character rolling.

4. DATA TRANSFER AND EXCHANGE

Aim

1. To transfer N bytes of data from memory location A: XXH to memory location B: YYH.
2. To exchange N bytes of data between memory locations A: XXH and B: YYH.

Apparatus Required

MCU8051 IDE (Freeware Simulator).

Algorithm

Program 1: Transfer N Bytes of Data

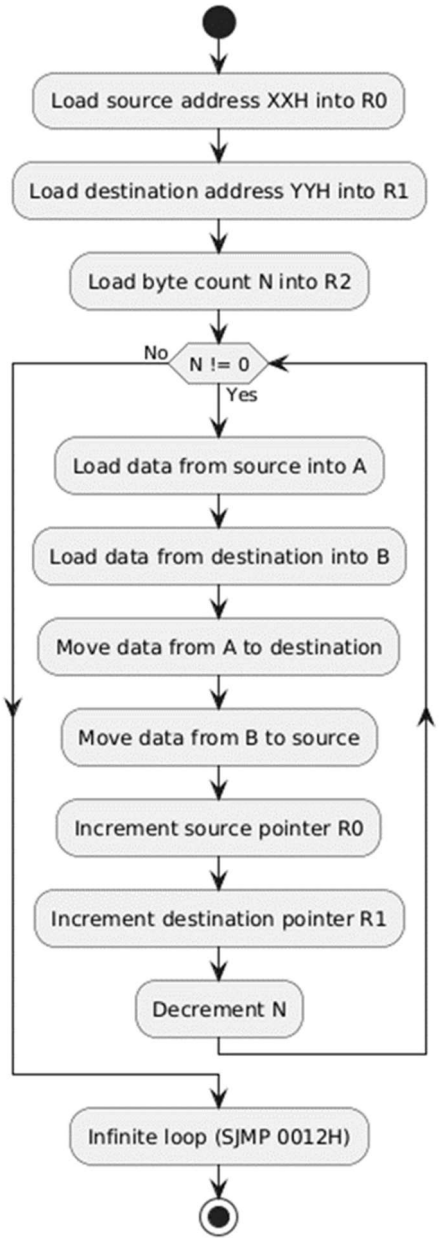
1. Load the source address (XXH) into R0.
2. Load the destination address (YYH) into R1.
3. Load the number of bytes N into a register (e.g., R2).
4. Start a loop:
    - Read a byte from the source location (@R0).
    - Write the byte to the destination location (@R1).
    - Increment both R0 and R1.
    - Decrement the counter R2.
5. Repeat the loop until N becomes zero.
6. Halt the program.

Program 1: Transfer N Bytes of Data

Address	Instruction	Operands	Description
0000H	MOV	R0, #XXH	Load source address XXH into register R0.
0002H	MOV	R1, #YYH	Load destination address YYH into register R1.
0004H	MOV	R2, #N	Load byte count N into register R2.
0006H	MOV	A, @R0	Load data from the source into the accumulator.
0008H	MOV	@R1, A	Move data from the accumulator to the destination.
000AH	INC	R0	Increment source pointer R0.
000BH	INC	R1	Increment destination pointer R1.
000CH	DJNZ	R2, 0006H	Decrement N and repeat if not zero.

Address	Instruction	Operands	Description
000EH	SJMP	000EH	Stop the program (infinite loop).

Flow Chart



Input:

- Source Address XXH: 05H, 06H, 07H.
- Destination Address YYH: Empty.
- Number of Bytes N: 03H.

Output:

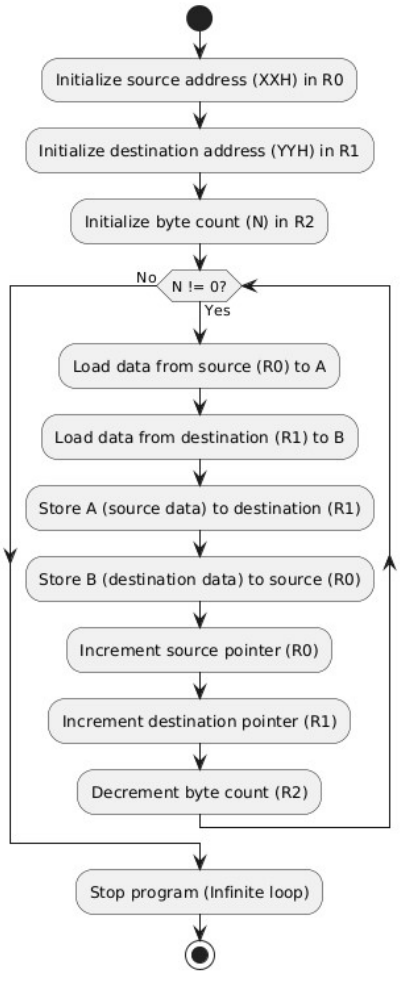
- Destination Address YYH: 05H, 06H, 07H.

Exchange N Bytes of Data

Algorithm

1. Load the source address (XXH) into R0.
2. Load the destination address (YYH) into R1.
3. Load the number of bytes N into a register (e.g., R2).
4. Start a loop:
   - Read a byte from the source location (@R0) into A.
   - Swap it with the byte at the destination location (@R1).
   - Store the original byte from A at the destination location (@R1).
   - Increment both R0 and R1.
   - Decrement the counter R2.
5. Repeat the loop until N becomes zero.
6. Halt the program.

Flow Chart



Exchange N Bytes of Data

Input and Output

Program 2: Exchange Example

- **Input**:
  - Source Address XXH: 0AH, 0BH, 0CH.
  - Destination Address YYH: 01H, 02H, 03H.
  - Number of Bytes N: 03H.
- **Output**:
  - Source Address XXH: 01H, 02H, 03H.
  - Destination Address YYH: 0AH, 0BH, 0CH.

Result

The programs demonstrate how to perform data transfer and exchange operations in the internal RAM of the 8051 microcontroller using assembly language. This practical example shows how loop-based operations and register manipulation can be applied effectively.

5. Data Manipulation

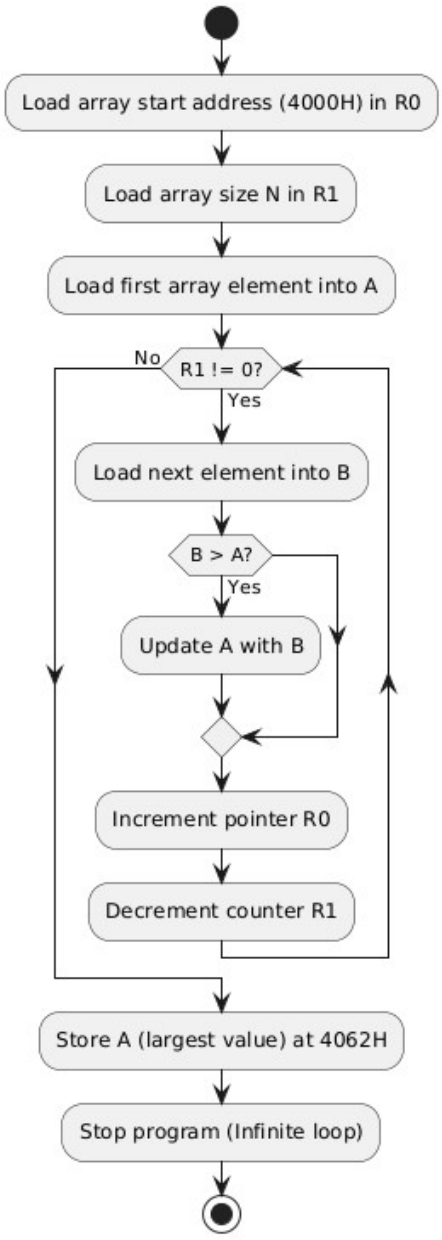
Aim

1. a) Write an assembly language program to find the largest element in a given array of N =___ H bytes at location 4000H. Store the largest element at location 4062H.
2. b) Write an assembly language program to count number of ones and zeros in an eight bit number.

Apparatus Required

MCU8051 IDE (Freeware Simulator).

Flow Chart



Algorithm

a. Finding the Largest Element in an Array

Initialize pointers:

1. Load the starting address of the array (`4000H`) into a pointer register (`R0`).
2. Load the count of elements `N` into a register (`R1`).
3. Initialize the largest value in accumulator `A` with the first element.

Loop through the array:

4. Compare the current value in the accumulator with the next value in the array.
5. If the next value is larger, update the accumulator with this value.
6. Increment the pointer to move to the next element.
7. Decrement the counter `R1` and repeat the loop if the counter is not zero.

Store the largest value:

8. Store the value in the accumulator (`A`) at `4062H`.

Stop the program:

9. Enter an infinite loop (`SJMP`).

Program : Finding the Largest Element in an Array

Address	Instruction	Operand	Description
0000H	MOV	R0, #4000H	Load starting address of the array into register R0.
0003H	MOV	R1, #N	Load the size of the array into register R1.
0006H	MOV	A, @R0	Load the first element of the array into accumulator A.
0008H	INC	R0	Increment the source pointer R0.
0009H	DEC	R1	Decrement the counter R1.
000AH	JZ	DONE	If R1 = 0, jump to DONE (end of loop).
000DH	MOV	B, @R0	Load the next element of the array into register B.
000FH	CJNE	A, B, NEXT	Compare accumulator A and B; if A ≠ B, jump to NEXT.
0012H	MOV	A, B	Move the value in register B (if larger) to accumulator A.
0014H	NEXT: INC	R0	Increment the source pointer R0.
0015H	SJMP	0009H	Jump back to check and process the next element in the loop.
0018H	DONE: MOV	4062H, A	Store the largest value in memory location 4062H.

Address	Instruction	Operand	Description
001BH	SJMP	001BH	Infinite loop to halt program execution.

Input and Output

Input:

Memory address 4000H: Array elements (e.g., 12H, 34H, 56H, 45H).

N: Number of elements (e.g., 04H for four elements).

Output:

Memory address 4062H: The largest element in the array (e.g., 56H).

Algorithm: Counting Ones and Zeros in an 8-Bit Number

Initialize:

1. Load the number into the accumulator A.
2. Set counters for ones (R0) and zeros (R1) to 0.

Loop 8 times:

3. Rotate the accumulator left (or right) to bring each bit into the carry flag.
4. If the carry flag is set, increment the counter for ones (R0).
5. Otherwise, increment the counter for zeros (R1).

Stop the program:

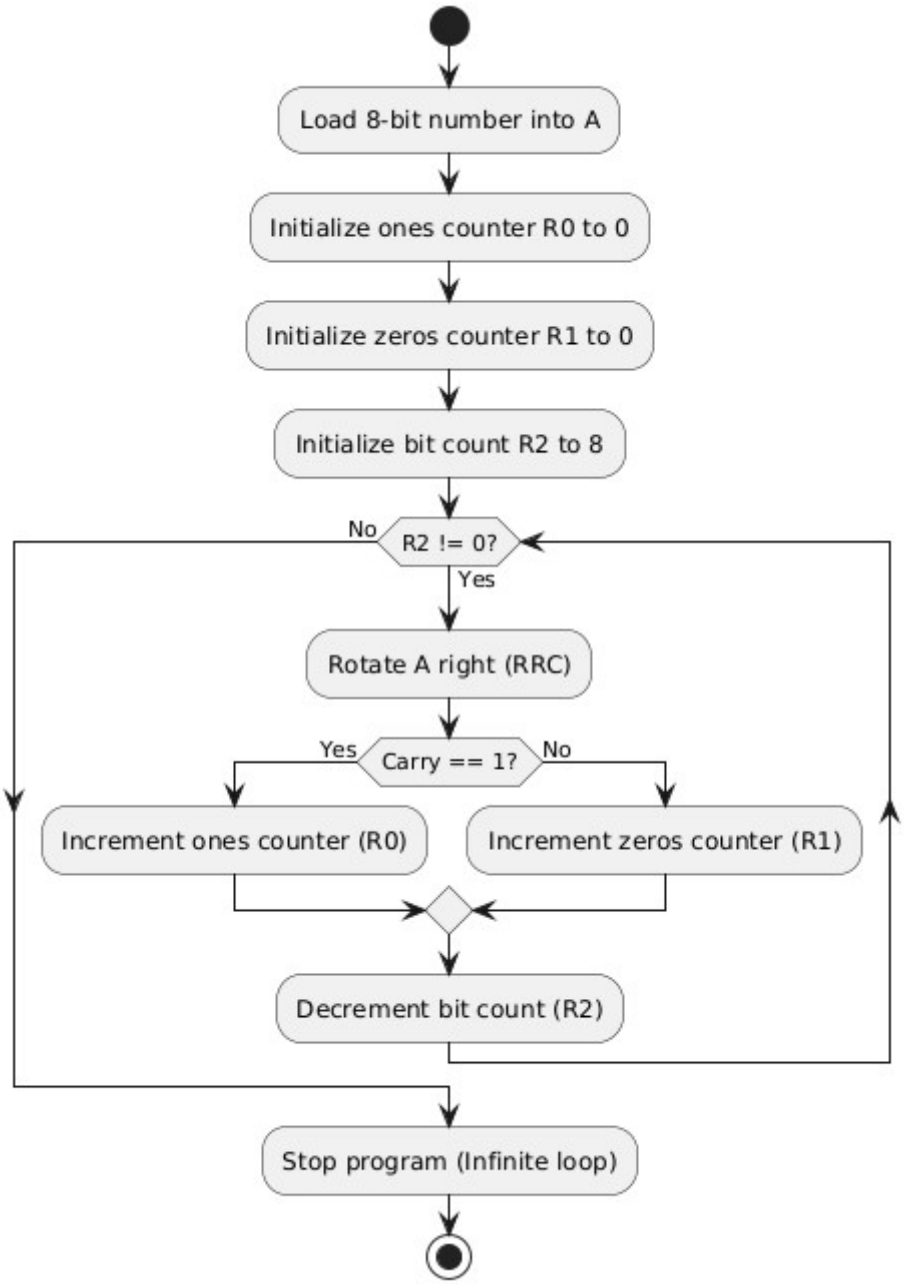
6. Enter an infinite loop (SJMP).

Program: Counting Ones and Zeros in an 8-Bit Number

Address	Instruction	Operand	Description
0000H	MOV	A, #data	Load the 8-bit number into accumulator A.
0002H	MOV	R0, #00H	Initialize the ones counter in register R0 to 0.
0004H	MOV	R1, #00H	Initialize the zeros counter in register R1 to 0.
0006H	MOV	R2, #08H	Load the number of bits (8) into register R2.
0008H	LOOP: RRC	A	Rotate the accumulator right; bit enters the carry flag.
0009H	JC	INCR_ONES	If the carry flag is set, jump to increment the ones counter.
000CH	INC	R1	Increment the zeros counter in register R1.
000DH	SJMP	NEXT	Skip to the next iteration of the loop.
0010H	INCR_ONES: INC	R0	Increment the ones counter in register R0.
0012H	NEXT: DJNZ	R2, LOOP	Decrement R2 and repeat the loop if not zero.

Address	Instruction	Operand	Description
0015H	SJMP	0015H	Infinite loop to halt program execution.

Flow Chart



Input and Output

1. **Input**:

- Accumulator `A`: 8-bit binary number (e.g., `A = 5AH` or `01011010` in binary).

2. **Output**:

   Register `R0`: Number of ones in the binary number (e.g., `4` for `01011010`).

   Register `R1`: Number of zeros in the binary number (e.g., `4` for `01011010`).

Result

Data manipulation program such as finding largest number in the given set of numbers and counting number of ones and zeros in the given number are written and executed successfully using 8051 microcontroller.

6. Arithmetic Programming

Aim

1. To perform addition of two 16-bit numbers.
2. To perform subtraction of two 16-bit numbers.
3. To perform multiplication of two 8-bit numbers.
4. To calculate the square of a given number.

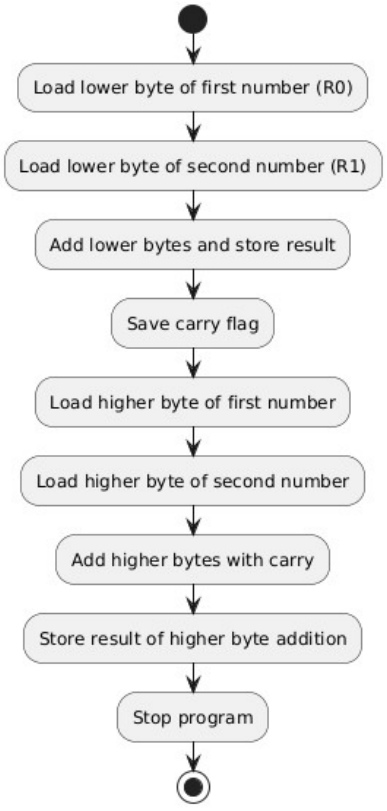
Apparatus Required:

8051 Microcontroller Trainer Kit or **MCU8051 IDE Simulator**

Algorithm: a) Addition of Two 16-bit Numbers

1. Load the lower bytes of the first and second numbers into registers.
2. Add the lower bytes and store the result. Save the carry.
3. Load the higher bytes of the first and second numbers.
4. Add the higher bytes along with the carry from the previous addition.
5. Store the result in the specified memory location.

Flow Chart



Program: a) Addition of Two 16-bit Numbers

Address	Instruction	Operand	Description
0000H	MOV	R0, #XXH	Load the lower byte of the first number.
0002H	MOV	R1, #YYH	Load the lower byte of the second number.
0004H	ADD	A, R0	Add the lower bytes of the numbers.
0005H	MOV	30H, A	Store the result of the lower byte addition.
0007H	MOV	R0, #ZZH	Load the higher byte of the first number.
0009H	MOV	R1, #AAH	Load the higher byte of the second number.
000BH	ADDC	A, R0	Add the higher bytes with the carry.
000CH	MOV	31H, A	Store the result of the higher byte addition.
000EH	SJMP	$	Infinite loop to stop program execution.

Input and Output: a) Addition of Two 16-bit Numbers

- Input:
  - First number: 1234H (Lower byte: 34H, Higher byte: 12H).
  - Second number: 5678H (Lower byte: 78H, Higher byte: 56H).
- Output:
  - Result: 68ACH stored at 30H (lower byte: 8CH) and 31H (higher byte: 68H).

Algorithm: b) Subtraction of Two 16-bit Numbers

1. Load the lower bytes of the first and second numbers into registers.
2. Subtract the lower bytes and save the borrow.
3. Load the higher bytes of the first and second numbers.
4. Subtract the higher bytes along with the borrow from the previous subtraction.
5. Store the result.

Program: b) Subtraction of Two 16-bit Numbers

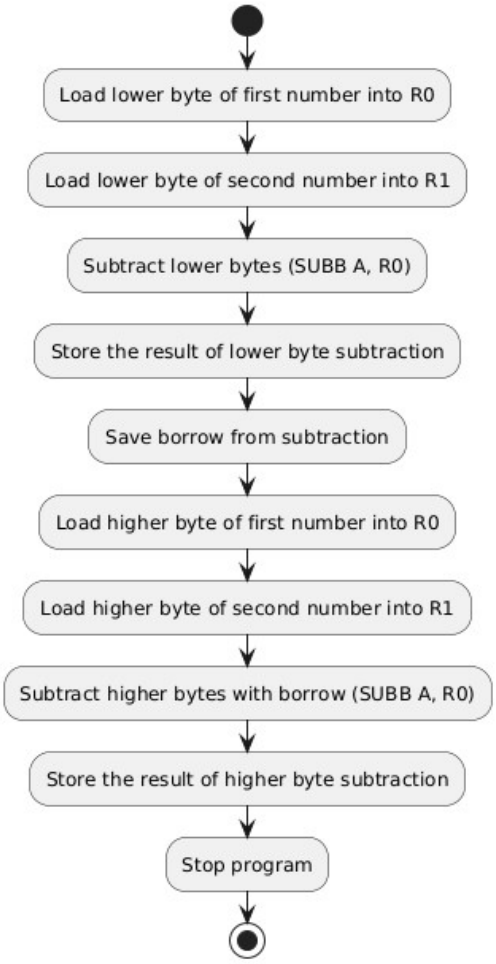
Address	Instruction	Operand	Description
0000H	MOV	R0, #XXH	Load the lower byte of the first number.
0002H	MOV	R1, #YYH	Load the lower byte of the second number.
0004H	SUBB	A, R0	Subtract the lower bytes of the numbers.
0005H	MOV	30H, A	Store the result of the lower byte subtraction.
0007H	MOV	R0, #ZZH	Load the higher byte of the first number.
0009H	MOV	R1, #AAH	Load the higher byte of the second number.
000BH	SUBB	A, R0	Subtract the higher bytes with the borrow.
000CH	MOV	31H, A	Store the result of the higher byte subtraction.
000EH	SJMP	$	Infinite loop to stop program execution.

Sample Input and Output

- Input:
  - First number: 5678H.
  - Second number: 1234H.
- Output:

Result: 4444H stored at 30H (lower byte: 44H) and 31H (higher byte: 44H)

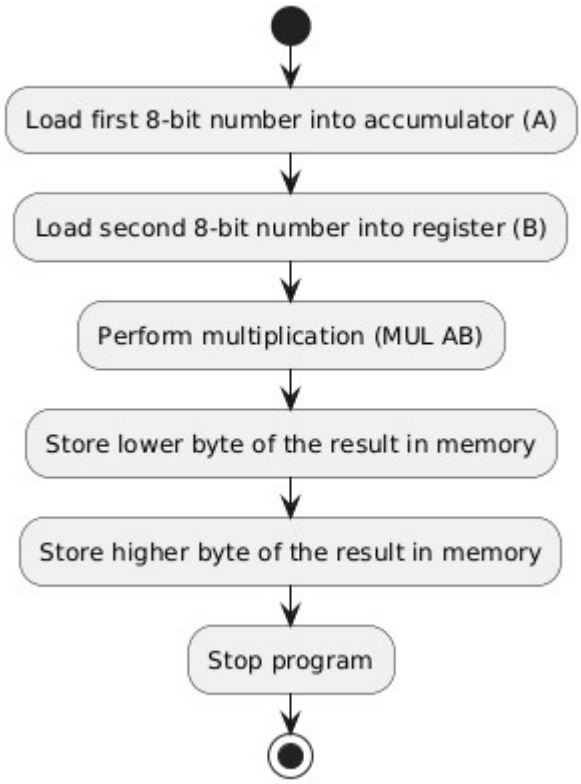
Flow Chart : b) Subtraction of Two 16-bit Numbers



Algorithm: c) Multiplication of Two 16-bit Numbers

1. Load the two 8-bit numbers into registers.
2. Perform the multiplication using the `MUL AB` instruction.
3. Store the lower byte of the result in `A` and the higher byte in `B`.

Flow Chart : c) Multiplication of Two 16-bit Numbers



Program: b) Multiplication of Two 16-bit Numbers

Address	Instruction	Operand	Description
0000H	MOV	A, #XXH	Load the first 8-bit number into accumulator A.
0002H	MOV	B, #YYH	Load the second 8-bit number into register B.
0004H	MUL	AB	Multiply the contents of A and B.
0005H	MOV	30H, A	Store the lower byte of the result in memory.
0007H	MOV	31H, B	Store the higher byte of the result in memory.
0009H	SJMP	$	Infinite loop to stop program execution.

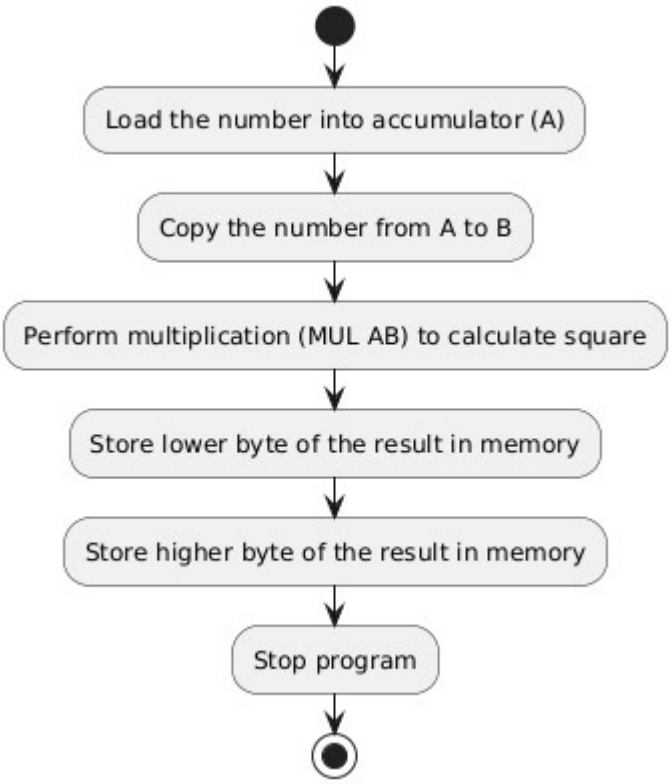
Input and Output: 16-bit Multiplication

- Input:
  - First number: 12H.
  - Second number: 34H.
- Output:
  - Result: 03C8H stored at 30H (C8H) and 31H (03H).

Algorithm: d) Square of a Given Number

1. Load the number into the accumulator.
2. Multiply the number by itself using `MUL AB`.
3. Store the result.

Flow Chart: d) Square of a Given Number



Program: d) Square of a Given Number

Address	Instruction	Operand	Description
0000H	MOV	A, #XXH	Load the number into accumulator A.
0002H	MOV	B, A	Copy the number to register B.
0004H	MUL	AB	Multiply the contents of A and B.
0005H	MOV	30H, A	Store the lower byte of the result in memory.
0007H	MOV	31H, B	Store the higher byte of the result in memory.
0009H	SJMP	$	Infinite loop to stop program execution.

Input and Output: d) Square of a Given Number

- Input:
  - Number: 06H.
- Output:

Square: 0036H stored at 30H (36H) and 31H (00H)

Result

The assembly language programs for addition, subtraction, multiplication, and squaring of numbers were executed successfully on the 8051 Microcontroller

7. Code Conversion Using 8051 Microcontroller

Aim

a. To convert a Binary Coded Decimal (BCD) number into ASCII.
b. To convert an ASCII number into Decimal.
c. To convert a Decimal number into ASCII.
d. To convert a binary (hex) number into Decimal.
e. To convert a BCD number to 7-Segment code.

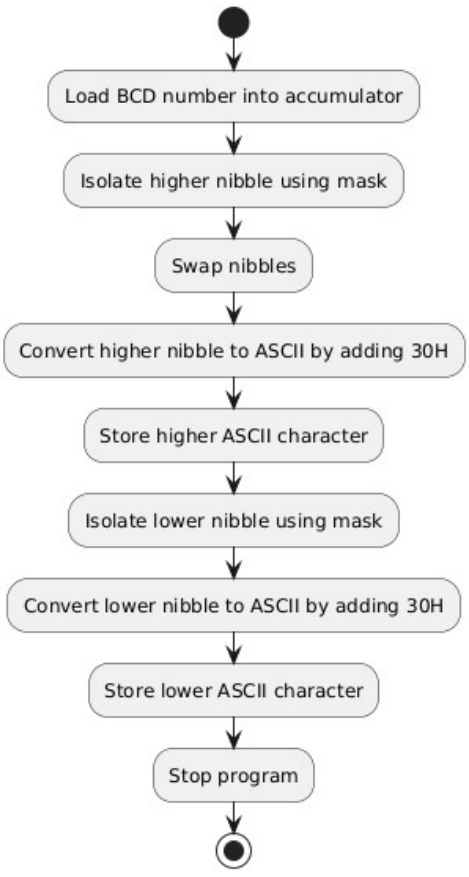
Apparatus Required:

1. 8051 Microcontroller Trainer Kit or MCU8051 IDE Simulator.
2. PC with assembler and IDE for simulation.
3. Power supply and connecting cables.

Algorithm : (a) Program to Convert a BCD Number to ASCII

1. Load the BCD number into accumulator A.
2. Separate the higher nibble (most significant digit) by masking the lower nibble using AND operation.
3. Convert the higher nibble to ASCII by adding 30H.
4. Store the higher ASCII character in memory.
5. Separate the lower nibble using masking and shift it into the higher nibble position.
6. Convert the lower nibble to ASCII by adding 30H.
7. Store the lower ASCII character in memory.

Flow Chart



Program : (a) Program to Convert a BCD Number to ASCII

Address	Instruction	Operand	Description
0000H	MOV	A, #BCD_NUM	Load the BCD number into accumulator A.
0002H	ANL	A, #0F0H	Mask the lower nibble to isolate the higher nibble.
0004H	SWAP	A	Swap nibbles to place the higher nibble in the lower nibble position.
0005H	ADD	A, #30H	Convert the higher nibble to ASCII.
0007H	MOV	30H, A	Store the higher ASCII character.
0009H	MOV	A, #BCD_NUM	Reload the BCD number.
000BH	ANL	A, #0FH	Mask the higher nibble to isolate the lower nibble.
000DH	ADD	A, #30H	Convert the lower nibble to ASCII.
000FH	MOV	31H, A	Store the lower ASCII character.
0011H	SJMP	$	Infinite loop to stop program execution.

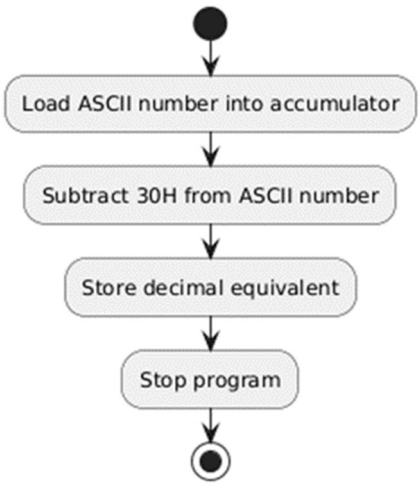
Input and Output: a) Program to Convert a BCD Number to ASCII

- **Input:**
  - BCD Number: 45H
- **Output:**
  - ASCII Representation: 34H (for 4) and 35H (for 5), stored at 30H and 31H.

Algorithm: b) Program to Convert an ASCII Number to Decimal

1. Load the ASCII number into accumulator A.
2. Subtract 30H from the ASCII number to convert it to decimal.
3. Store the result in memory.

Flow Chart: b) Program to Convert an ASCII Number to Decimal



b) Program to Convert an ASCII Number to Decimal

Address	Instruction	Operand	Description
0000H	MOV	A, #ASCII_NUM	Load the ASCII number into accumulator A.
0002H	SUBB	A, #30H	Subtract 30H from the ASCII number.
0004H	MOV	30H, A	Store the decimal equivalent in memory.
0006H	SJMP	$	Infinite loop to stop program execution.

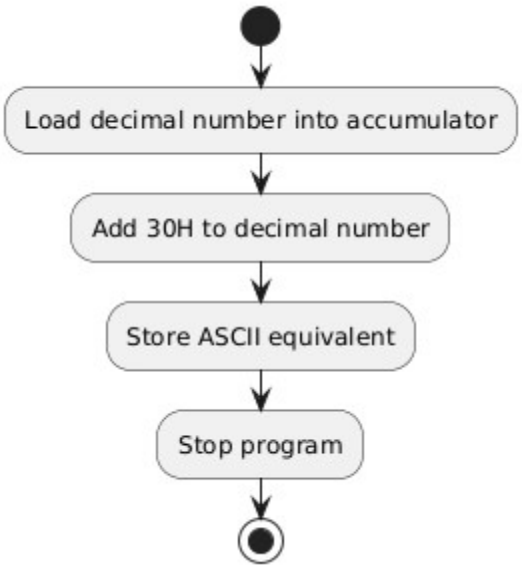
Input and Output: b) Conversion of ASCII Number to Decimal

- **Input**:
  - ASCII Number: 35H
- **Output**:
  - Decimal Equivalent: 05H, stored at 30H.

Algorithm: c) Conversion of a Decimal number into ASCII

1. Load the decimal number into accumulator A.
2. Add 30H to the decimal number to convert it to ASCII.
3. Store the result in memory.

Flow Chart: c) Conversion of a Decimal number into ASCII



c) Program to Convert Decimal Number to ASCII

Address	Instruction	Operand	Description
0000H	MOV	A, #DEC_NUM	Load the decimal number into accumulator A.
0002H	ADD	A, #30H	Add 30H to the decimal number.
0004H	MOV	30H, A	Store the ASCII equivalent in memory.
0006H	SJMP	$	Infinite loop to stop program execution.

Input and Output: Convert Decimal Number to ASCII

Input:

Decimal Number: 05H

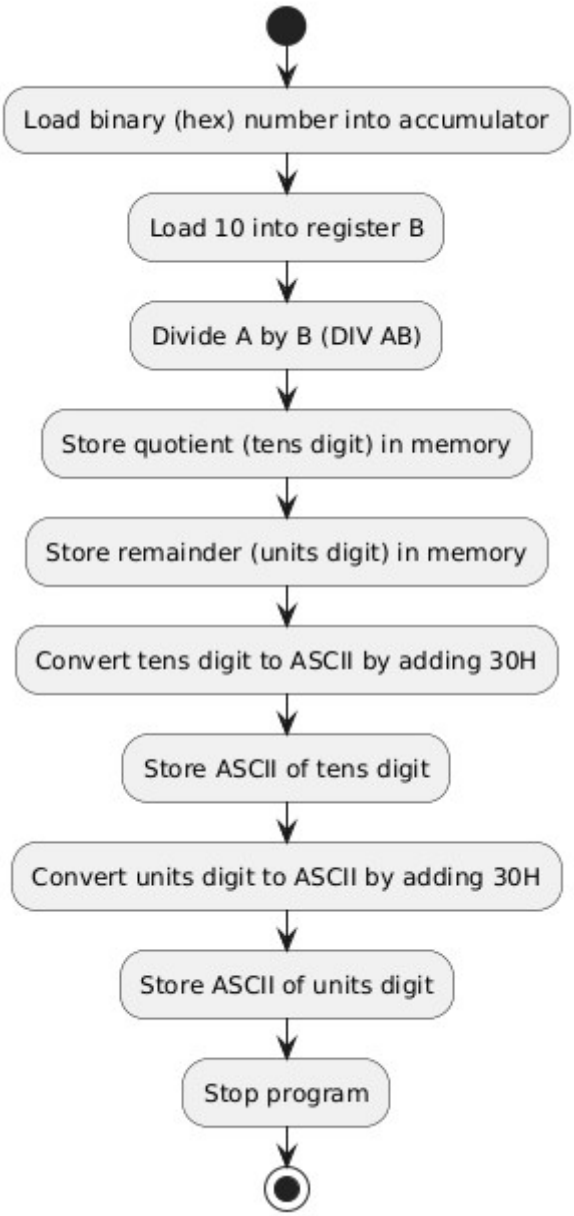
Output:

ASCII Representation: 35H, stored at 30H.

Algorithm: Conversion of Binary (Hex) number to Decimal

1. Load the binary (hexadecimal) number into accumulator A.
2. Divide the number by 10 to isolate the tens digit using the DIV AB instruction.
3. Store the quotient as the tens digit in memory.
4. Multiply the quotient by 10 and subtract from the original number to get the units digit.
5. Store the units digit in memory.
6. Convert both digits (tens and units) to ASCII by adding 30H.
7. Store the ASCII values in memory.

Flow Chart: Conversion of Binary (Hex) number to Decimal



Program: Conversion of Binary (Hex) number to Decimal

Address	Instruction	Operand	Description
0000H	MOV	A, #HEX_NUM	Load the binary (hex) number into accumulator A.
0002H	MOV	B, #0AH	Load the value 10 into register B for division.
0004H	DIV	AB	Divide A by B. The quotient is in A, remainder in B.
0005H	MOV	30H, A	Store the quotient (tens digit) in memory.
0007H	MOV	31H, B	Store the remainder (units digit) in memory.
0009H	MOV	A, 30H	Load the tens digit.

Address	Instruction	Operand	Description
000BH	ADD	A, #30H	Convert the tens digit to ASCII.
000DH	MOV	32H, A	Store the ASCII value of the tens digit.
000FH	MOV	A, 31H	Load the units digit.
0011H	ADD	A, #30H	Convert the units digit to ASCII.
0013H	MOV	33H, A	Store the ASCII value of the units digit.
0015H	SJMP	$	Infinite loop to stop program execution.

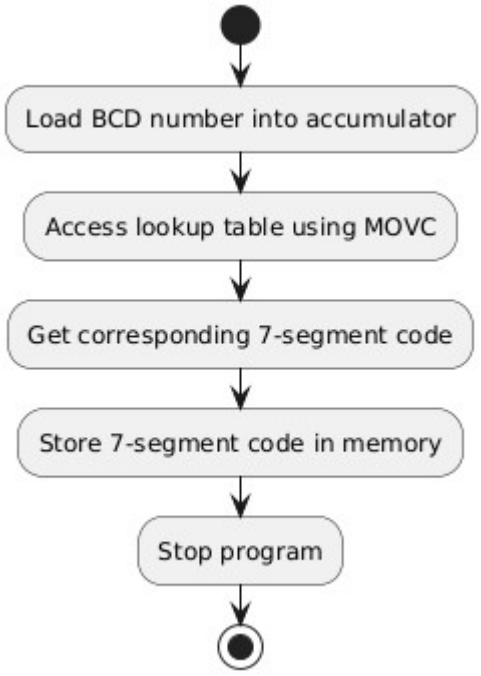
Input and Output: Conversion of Binary (Hex) number to Decimal

- **Input**:
  - Hex Number: `1CH` (28 in decimal).
- **Output**:
  - Decimal Representation: `2` and `8`.
  - ASCII Representation: `32H` (for 2) and `38H` (for 8), stored at `32H` and `33H`.

Algorithm: e) Convert BCD to 7-Segment Code

1. Load the BCD number into accumulator A.
2. Use a lookup table in memory to find the corresponding 7-segment code.
3. Store the 7-segment code in memory.

Flow Chart: e) Convert BCD to 7-Segment Code



Program: e) Convert BCD to 7-Segment Code

Address	Instruction	Operand	Description
0000H	MOV	A, #BCD_NUM	Load the BCD number into accumulator A.
0002H	MOVC	A, @A+DPTR	Use the lookup table to get the corresponding 7-segment code.
0004H	MOV	30H, A	Store the 7-segment code in memory.
0006H	SJMP	$	Infinite loop to stop program execution.

Input and Output

- **Input**:
  - BCD Number: 02H.
- **Output**:
  - 7-Segment Code: 5BH (corresponding to digit 2), stored at 30H.
- **7-Segment Code Lookup Table**

Digit	7-Segment Code (Hex)
0	3FH
1	06H
2	5BH
3	4FH
4	66H
5	6DH
6	7DH
7	07H
8	7FH
9	6FH

Result:

The assembly language programs for Binary (Hex) to Decimal conversion and BCD to 7-Segment code conversion were executed successfully on the 8051 Microcontroller