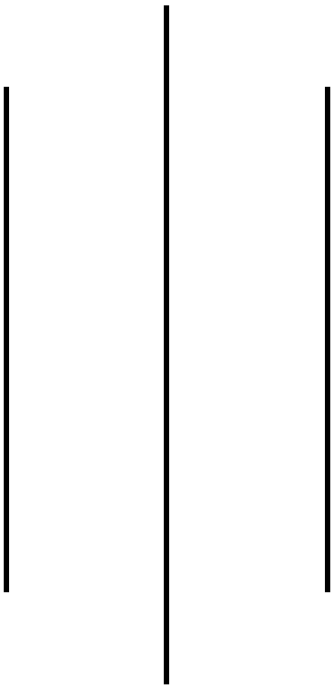
Far Western University

सुदूरपश्चिमाञ्चल विश्वविद्यालय

School Of Engineering



"A *REPORT* ON *Programming in 8085 Microprocessor*"

Instructed By :

Er. Kamal Lekhak

Lecturer

School Of Engineering

Submitted By :

Nitesh Joshi (26)

Lalit Bhatt (19)

Birendra Bohara (09)

Basant Bhat (06)

Table of Content

Introduction

Introduction to Programming in 8085 :-

The 8085 microprocessor is an 8-bit microprocessor used for learning low-level programming, hardware interaction, and understanding how CPUs work internally.

When we write a program for 8085, we use assembly language—a human-readable form of machine instructions.

The 8085 microprocessor contains a set of registers, an Arithmetic and **Logic Unit (ALU)**, a **control unit**, and a **16-bit address bus** capable of addressing up to **64 KB of memory**. It also includes a data bus, flag register, stack pointer, and program counter, all of which play important roles in program execution. By writing assembly language programs, users can perform operations such as arithmetic calculations, data transfer, logical operations, and decision-making.

A typical 8085 program involves loading instructions into memory, executing them sequentially, and observing changes in registers or memory locations. The instructions are categorized into data transfer, arithmetic, logical, branching, and machine control instructions. These instructions allow the processor to manipulate data, perform computations, and control program flow.

Learning 8085 programming helps us to develop a strong foundation in microprocessor architecture, addressing modes, and instruction execution. It strengthens the understanding of how software directly interacts with hardware, which is essential in fields such as embedded systems, digital electronics, and computer engineering.

Data Transfer Instructions

Data transfer instructions are used to move data from one location to another within the 8085 microprocessor system. These instructions do not modify the actual data; they simply copy it from the source to the destination. The source remains unchanged after the operation. Data can be transferred between registers, between memory and registers, and between the accumulator and memory.

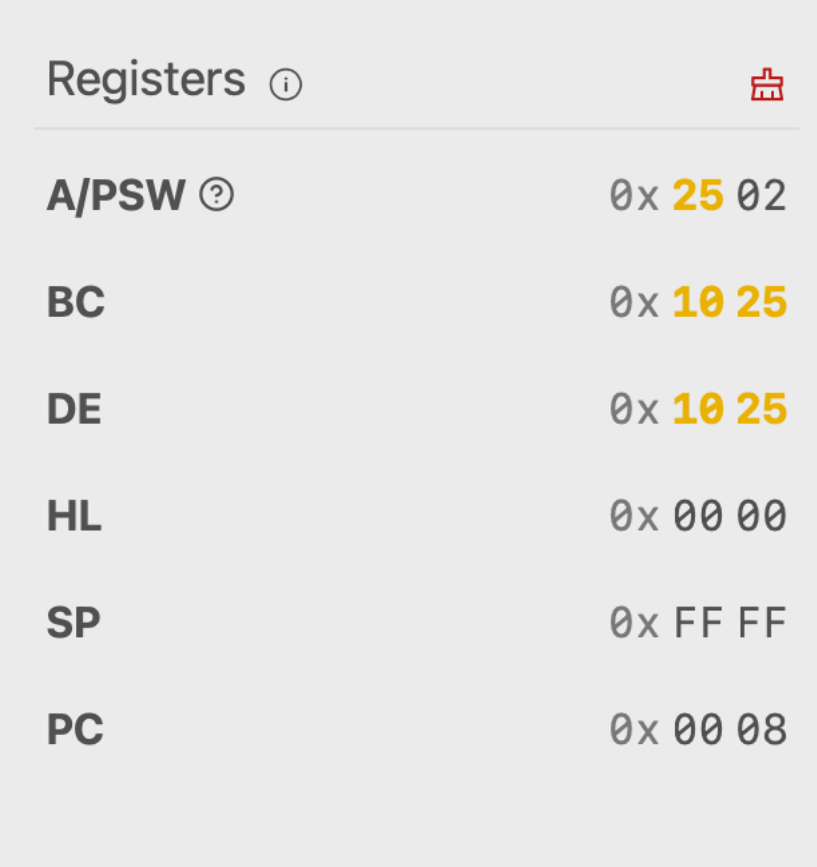
Code Implementations:

Problem1 : To demonstrate simple register-to-register and immediate data transfer instructions.

Code :

```
MVI A, 25H      ; Load 25H into A
MVI B, 10H      ; Load 10H into B
MOV C, A        ; Copy A → C
MOV D, B        ; Copy B → D
MOV E, C        ; Copy C → E
HLT             ; Stop execution
```

Output:



Description:

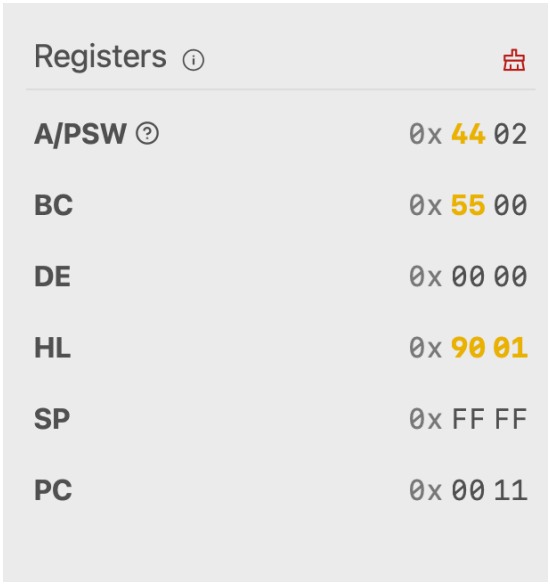
This program demonstrates simple data transfer operations in the 8085 microprocessor. It loads immediate 8-bit values into registers using the MVI instruction and then copies data between registers using the MOV instruction. The program shows how data moves from one register to another without modification. It is the simplest form of data handling on the 8085 and helps understand basic register operations.

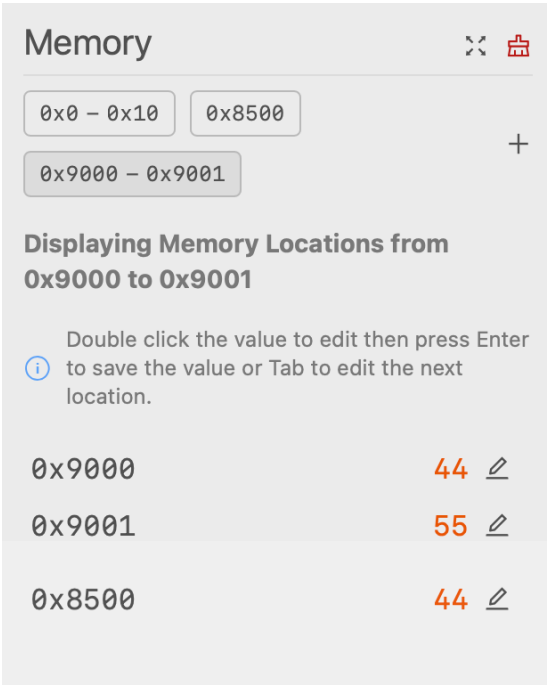
Problem 2 : To demonstrate memory load/store and register pair instructions using HL.

Code :

```
; Advanced Data Transfer Instructions

    LXI H, 9000H     ; HL = 9000H
    MVI A, 44H       ; Load 44H into A
    MOV M, A         ; Store A at memory 9000H

    INX H            ; HL = 9001H
    MVI B, 55H       ; Load 55H into B
    MOV M, B         ; Store B at memory 9001H

    LDA 9000H        ; Load A from memory 9000H
    STA 8500H        ; Store A to memory 8500H

    HLT              ; Terminate the program
```

Output :





Description :

This program demonstrates advanced data transfer operations involving both registers and memory. It uses the LXI instruction to load a 16-bit address into the HL register pair and stores data into consecutive memory locations through indirect addressing. The program also shows how to load data from memory into the accumulator using LDA, and how to store the accumulator contents back to a different memory location using STA. This example introduces memory handling and register-pair-based addressing in 8085 programming.

Arithmetic Instructions

Arithmetic instructions in the 8085 microprocessor are used to perform basic mathematical operations such as addition, subtraction, increment, and decrement. These operations are carried out by the Arithmetic and Logic Unit (ALU) using the accumulator as the primary register. Most arithmetic instructions affect the flags (Zero, Carry, Sign, Auxiliary Carry, and Parity), which helps in decision-making and branching.

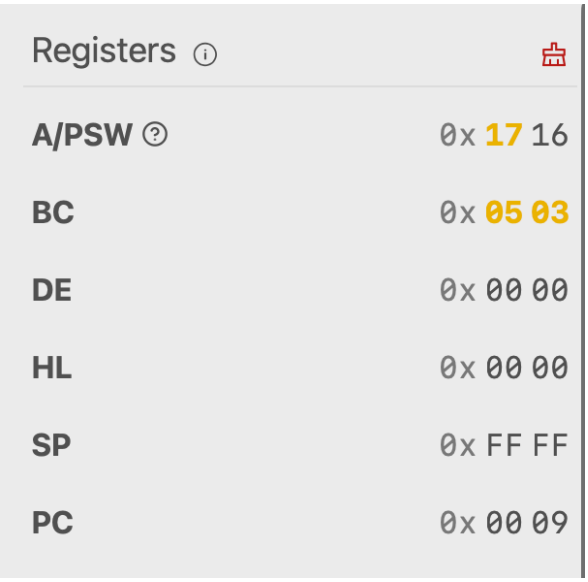
Code Implementations :-

Problem 1 : To perform simple addition and subtraction using accumulator-based arithmetic instructions.

Code :

```
; Basic Arithmetic Operations

    MVI A, 15H      ; Load 15H into A
    MVI B, 05H      ; Load 05H into B
    ADD B           ; A = A + B = 15H + 05H = 1AH

    MVI C, 03H      ; Load 03H into C
    SUB C           ; A = A - C = 1AH - 03H = 17H

    HLT             ; Stop execution
```

Output:



Description :

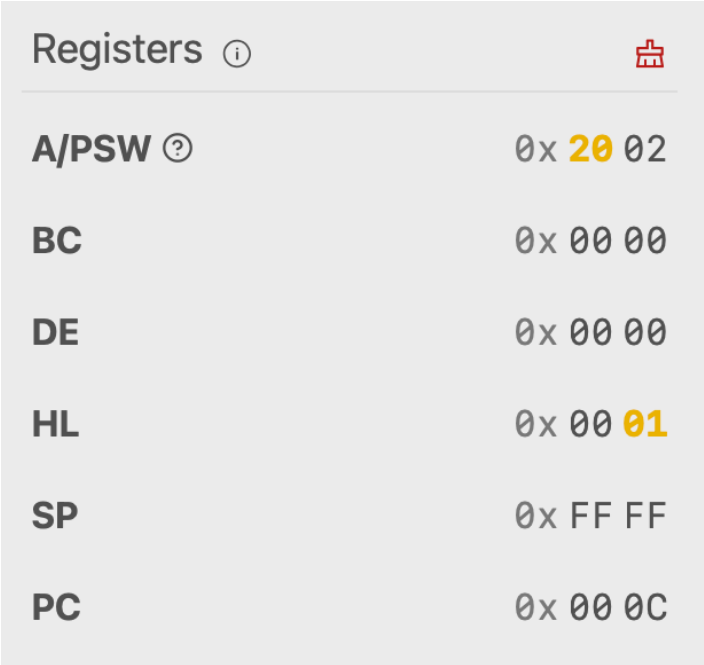
This program performs simple arithmetic using the accumulator. It adds the content of register B to A, then subtracts the content of register C from A, demonstrating basic use of ADD and SUB instructions.

Problem 2 : To demonstrate increment, decrement, and immediate addition operations using 8085 arithmetic instructions.

Code :-

```
; Advanced Arithmetic Instructions

    MVI A, 20H      ; Load A = 20H
    ADI 10H         ; A = A + 10H = 30H

    INR A           ; A = A + 1 = 31H
    DCR A           ; A = A - 1 = 30H

    MVI B, 00H      ; Load B = 00H
    SUI 10H         ; A = A - 10H = 20H

    INX H           ; Increment HL register pair

    HLT
```

Output:-



Description:

This program shows advanced arithmetic operations such as immediate addition (ADI), immediate subtraction (SUI), register increment/decrement (INR, DCR), and register-pair increment/decrement (INX, DCX). It demonstrates how the ALU handles arithmetic with both single registers and register pairs.

Logical Instructions

Logical instructions in the 8085 microprocessor are used to perform bitwise operations such as AND, OR, XOR, compare, and complement. These operations are carried out in the ALU and mainly affect the flags (Zero, Sign, Parity, Carry, and Auxiliary Carry), which is useful for decision-making in programs.

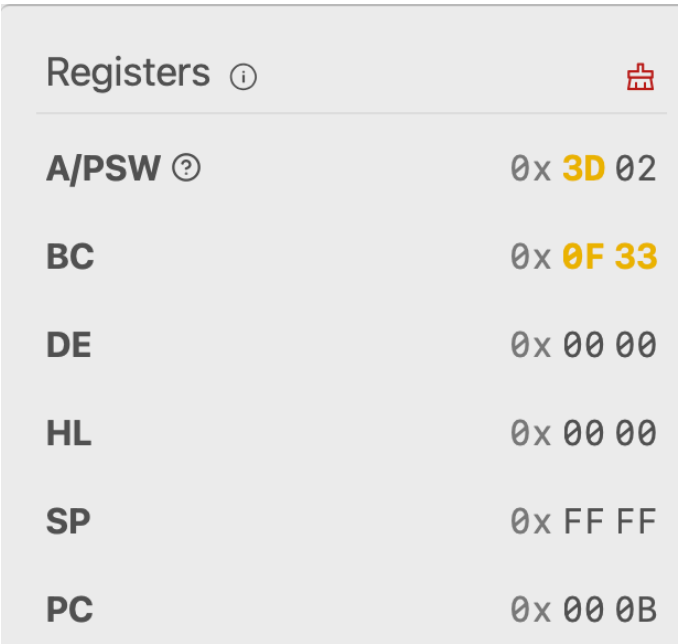
Code Implementations:-

Problem 1 : To perform AND, OR, and XOR operations using logical instructions.

Code :

```
; Basic Logical Instructions

    MVI A, 55H      ; A = 55H (01010101)
    MVI B, 0FH      ; B = 0FH (00001111)

    ANA B           ; A = A AND B = 05H

    MVI C, 33H      ; C = 33H
    ORA C           ; A = A OR C

    XRI 0AH         ; A = A XOR 0AH

    HLT
```

Output :



Description:

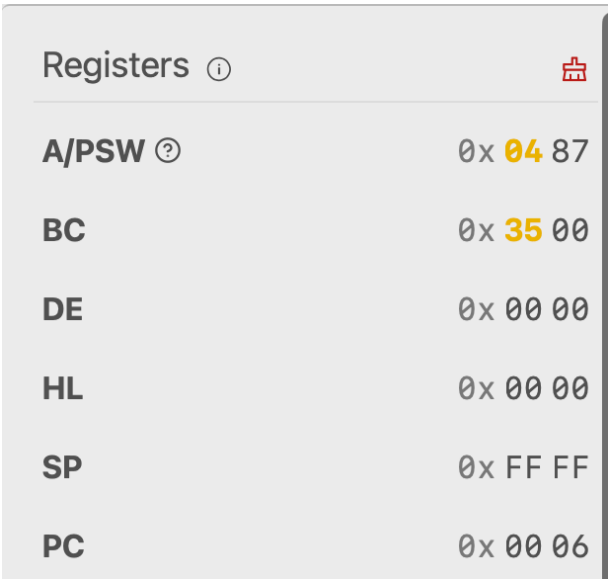
This program demonstrates simple logical operations: AND, OR, and XOR using registers and immediate values.

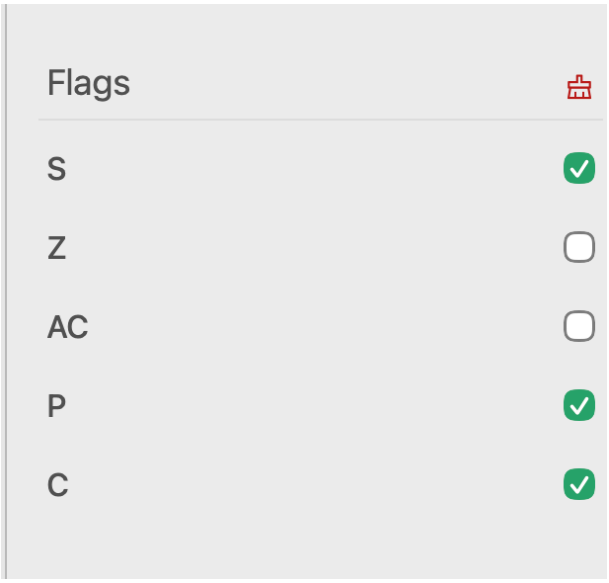
Problem 2 : To compare accumulator content with another register using CMP instruction.

Code :-

```
; Comparing Two Values

    MVI A, 04H      ; A = 04H
    MVI B, 35H      ; B = 35H

    CMP B           ; Compare A with B (sets flags)
                    ; A < B → Carry flag = 1

    HLT
```

Output :





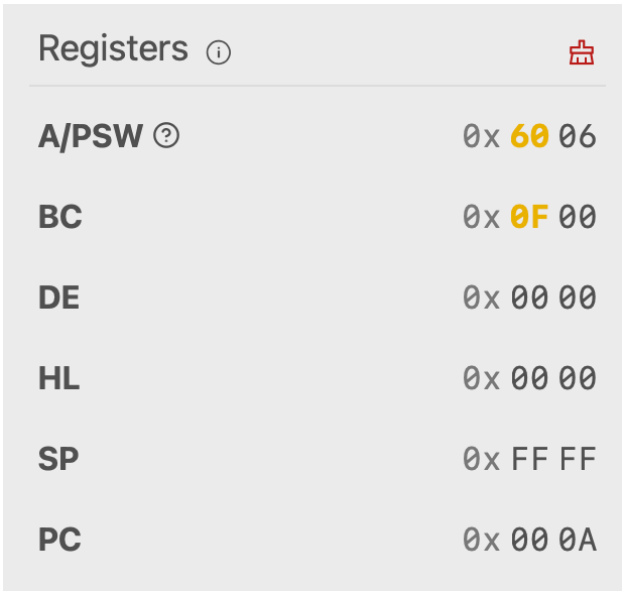
Description: This program compares two 8-bit values using CMP. It sets flags based on the result (useful for branching and decision-making).

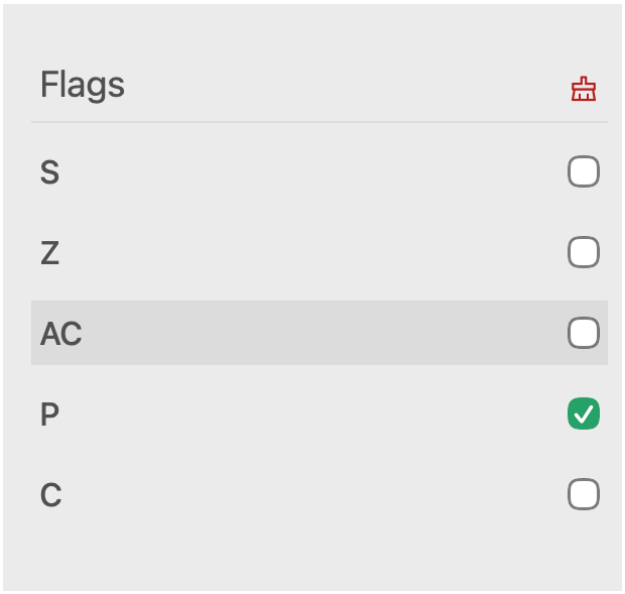
Problem 3 : To demonstrate complementing the accumulator and manipulating carry flag.

Code :

```
; Advanced Logical Instructions

    MVI A, 96H      ; Load A = 96H
    CMA             ; Complement A (bitwise NOT)

    STC             ; Set Carry flag
    CMC             ; Complement Carry flag (toggle)

    MVI B, 0FH      ; B = 0FH
    ANI 0F0H        ; A = A AND 0F0H (mask lower bits)

    HLT
```

Output:





Description: This program demonstrates advanced logical operations like accumulator complement (CMA), carry flag manipulation (STC/CMC), and masking bits using AND immediate.

Branching Instuctions

Branching instructions in the 8085 microprocessor are used to change the normal sequential flow of program execution. Instead of executing instructions line by line, branching allows the program to jump to a new memory location based on conditions or unconditionally. These instructions are essential for loops, decision-making, and implementing logic structures like IF–ELSE.

Branching instructions are of two types:

- Unconditional Branching:
  The jump happens every time.
  Examples: JMP, CALL, RET
- Conditional Branching:
  The jump happens only if specific flags (Zero, Carry, Sign, Parity) meet a condition.
  Examples: JZ, JNZ, JC, JNC, JP, JM

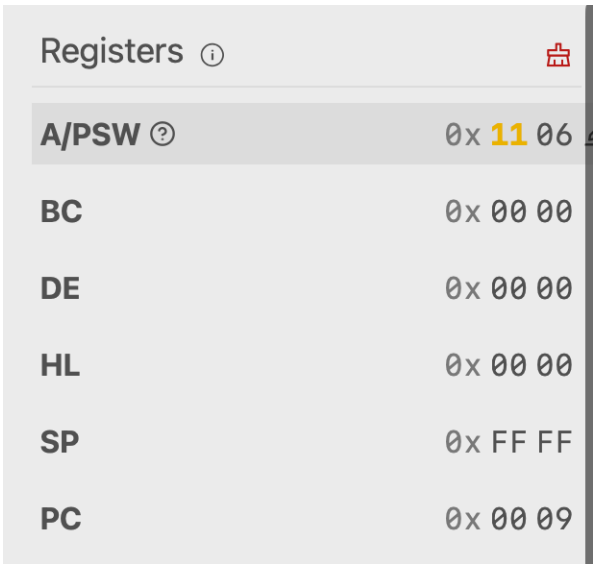
Code Implementations:-

Problem 1 : To demonstrate the use of an unconditional jump using JMP.

Code :

```
; Unconditional Jump Example

        MVI A, 10H      ; A = 10H
        JMP SKIP        ; Jump to label SKIP

        MVI A, 99H      ; This line will be skipped

SKIP:   INR A           ; A = A + 1 = 11H
        HLT
```

Output :



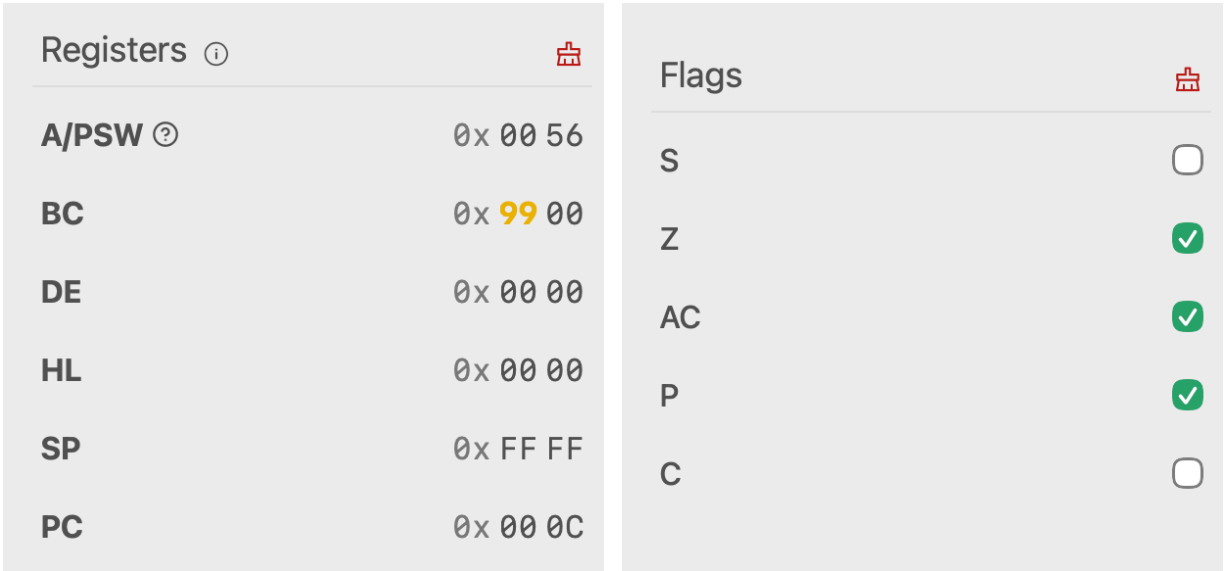
Description : This program shows how JMP directly changes the execution flow by skipping instructions.

Problem 2 : To jump based on the Zero flag using JZ (Jump if Zero).

Code :

```
; Conditional Jump Example (JZ)

        MVI A, 05H      ; Load A = 5
        SUI 05H         ; A = A - 5 → A becomes 00H (Zero flag = 1)

        JZ ZERO_LABEL   ; Jump if A is zero

        MVI B, 55H      ; This line is skipped if zero

ZERO_LABEL:
        MVI B, 99H      ; Executes only if Zero flag is set
        HLT
```

Output:



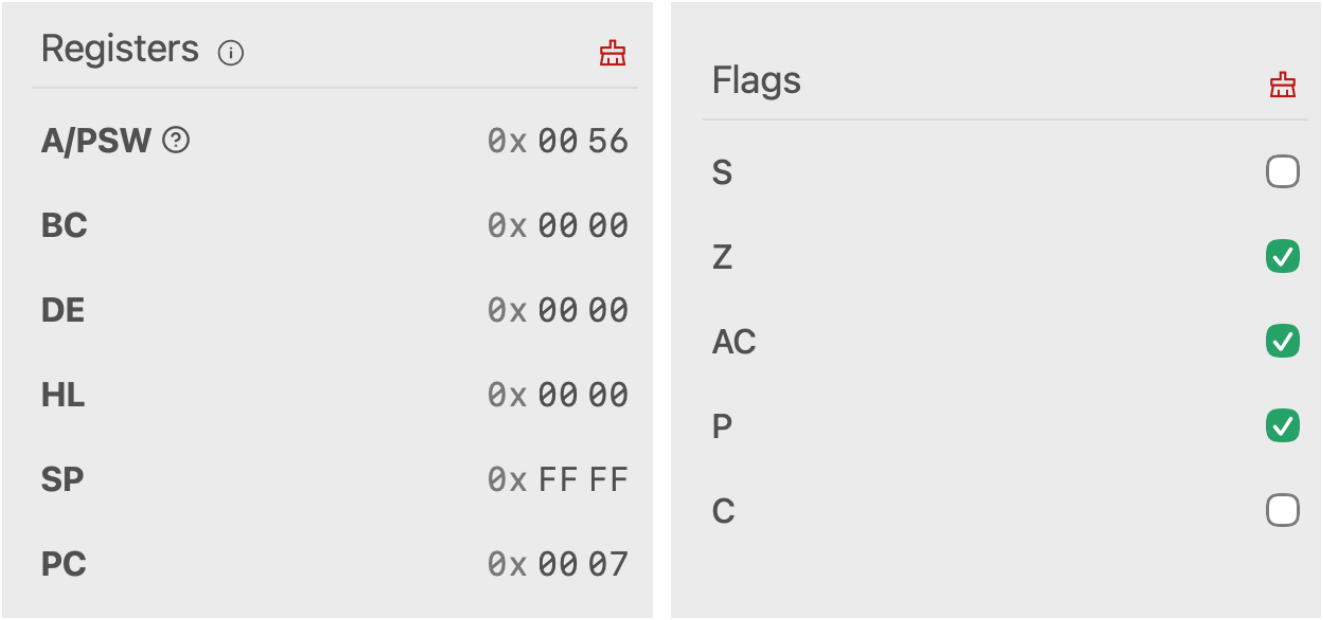
Description:
This program uses JZ to perform a jump only when the result becomes zero.

Problem 3: To demonstrate loop formation using DCR and JNZ.

Code :

```
; Loop Example Using JNZ

        MVI C, 05H        ; C = 5 (loop count)

LOOP:   DCR C             ; C = C - 1
        JNZ LOOP          ; Repeat loop until C = 0

        HLT
```

Output:



Description :

This program creates a loop that repeats 5 times using JNZ (Jump if Not Zero).

Machine Control Instructions

Machine control instructions are used to control the overall operation of the microprocessor rather than performing arithmetic or logic tasks. These instructions manage the system's hardware, control processor operations, and help in interrupt handling.

They do not affect flags (except a few related to interrupts).

Common machine control instructions include:

- NOP – No operation; processor does nothing for 1 cycle
- HLT – Halts the processor until reset
- EI – Enables all interrupts
- DI – Disables all interrupts
- SIM – Sets interrupt masks and controls serial output
- RIM – Reads interrupt masks and serial input
- RST n – Software restart at fixed locations

Machine control instructions help in processor synchronization, interrupt management, power-saving, and system-level control.

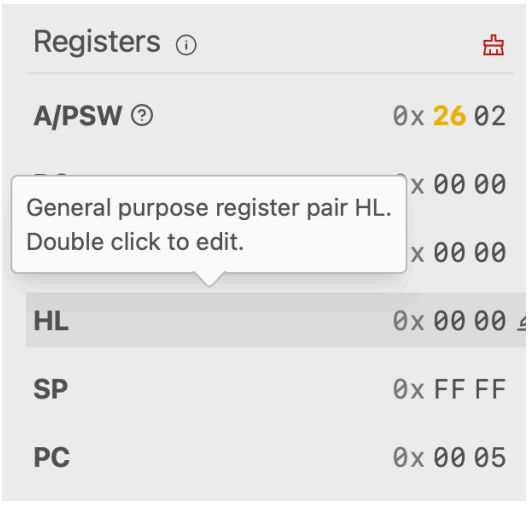
Code Implementations:-

Problem 1 : To demonstrate the use of NOP and HLT machine control instructions.

Code :

```
; Program demonstrating NOP and HLT

    MVI A, 25H      ; Load A = 25H
    NOP             ; Do nothing (wastes 1 machine cycle)
    INR A           ; A = A + 1 = 26H

    HLT             ; Stop processor
```

Output :



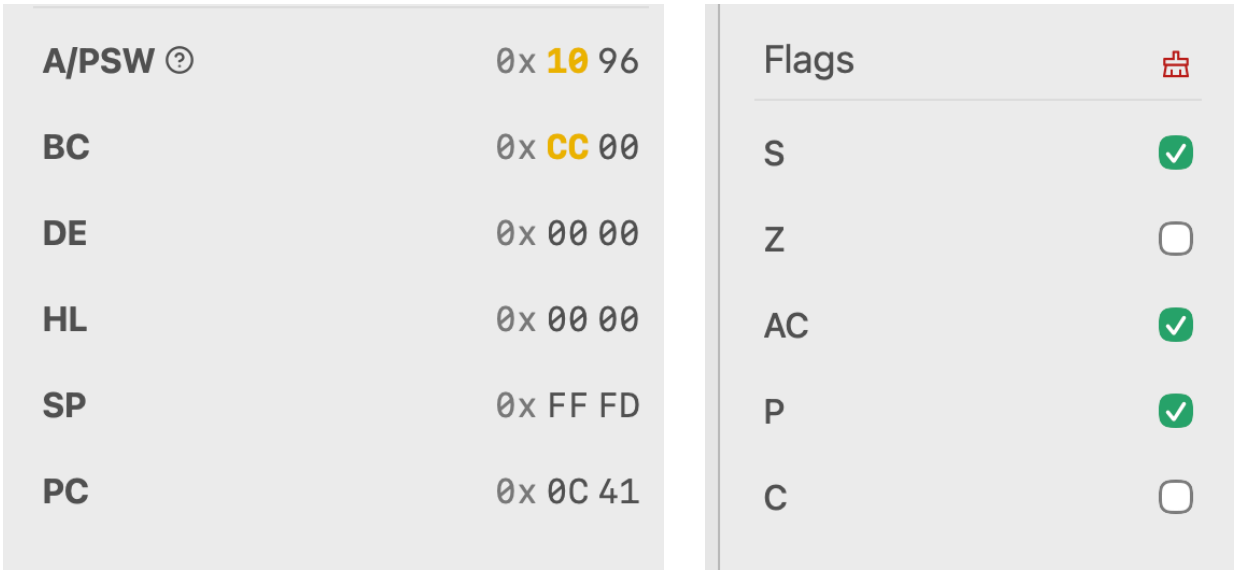
Description : This program shows how NOP performs no operation and how HLT stops the processor.

Problem 2 : To demonstrate interrupt control using EI, DI, and a software interrupt RST 5.

Code :

```
; Program using EI, DI, and RST

    DI              ; Disable all interrupts
    MVI A, 10H      ; Load A = 10H

    EI              ; Enable interrupts
    RST 5           ; Software restart at address 0028H

    HLT
```

Output:



Description : This program disables interrupts, performs a task, then re-enables interrupts.
It also uses RST 5, which is a software interrupt that jumps to a fixed memory location (0028H).