UNIT-III

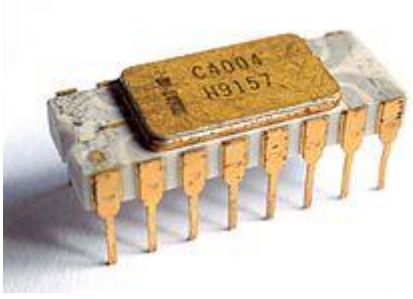
Introduction To 8085 Microprocessor

Microprocessors: The Brains of the Machine

- *Microprocessors, often called CPUs (Central Processing Units),* are the tiny electronic brains inside our computers and many other devices.
- They are like the conductors of an orchestra, receiving instructions, processing data, and controlling all the other components.

Evolution of Microprocessors:

- **Early Days (1970s):** The first microprocessors were very basic, capable of performing only simple calculations. The Intel 4004, introduced in 1971, is considered the first commercially available microprocessor.



- **Advancement (1980s & 1990s):** Microprocessors became more powerful with each generation, allowing for faster processing speeds and more complex tasks.
- The introduction of the IBM PC in the early 1980s, powered by the Intel 8086 microprocessor, is a major milestone in this era.

IBM PC with Intel 8086 microprocessor

- **Modern Era (2000s - Today):** Microprocessors today are incredibly sophisticated, containing billions of transistors and capable of performing trillions of calculations per second. Modern CPUs like the Intel Core i series or AMD Ryzen processors are examples.

Features of Microprocessors:

- **Microprocessor on a Chip:** They integrate the essential processing elements like the Arithmetic Logic Unit (ALU) for calculations, the Control Unit (CU) for managing instructions, and registers for temporary data storage, all on a single chip.

- **Instruction Fetch, Decode, and Execute Cycle:** Microprocessors follow a basic cycle of fetching instructions from memory, decoding them to understand what needs to be done, and then executing those instructions.
- **Clock Speed:** Measured in Gigahertz (GHz), it determines how many cycles a processor can complete in a second. Higher clock speeds generally indicate faster performance.
- **Number of Cores:** Modern CPUs often have multiple cores, allowing them to handle multiple tasks simultaneously and improving overall performance.

Applications of Microprocessors:

- **Computers:** From desktops and laptops to servers and supercomputers, microprocessors are the heart of all modern computers.

- **Mobile Devices:** Smartphones and tablets rely on microprocessors for everything from running apps to displaying graphics.
- **Embedded Systems:** Microprocessors are present in countless devices, including cars, appliances, medical equipment, and even toys, controlling their functions.

In Conclusion:

Microprocessors have revolutionized the way we live and work. Their continuous evolution has led to immense computing power that underpins modern technology.

The Intel 8085 Microprocessor

The 8085 microprocessor, introduced by Intel in 1977, was an 8-bit processor that played a significant role in the early days of personal computing. It was known for being:

- **Simple and Easy to Use:** Compared to its predecessor, the 8080, the 8085 required less external circuitry, making it a

more affordable and accessible option for building microcomputers.

- **8-bit Processing:** It could handle 8 bits of data at a time, which was common for microprocessors of that era.
- **Binary Compatible with the 8080:** This meant that programs written for the 8080 could also run on the 8085 with some adjustments.

Key Features of the 8085:

- **Internal Registers:** The 8085 had several registers for storing data temporarily during program execution. These included an accumulator for performing calculations and general-purpose registers for additional data storage.
- **Memory Addressing:** It could access up to 64 KB of memory, which was a significant capacity for its time.
- **Instruction Set:** The 8085 had a set of instructions that it could understand and execute to perform various

operations like arithmetic, logical operations, data transfer, and control flow.

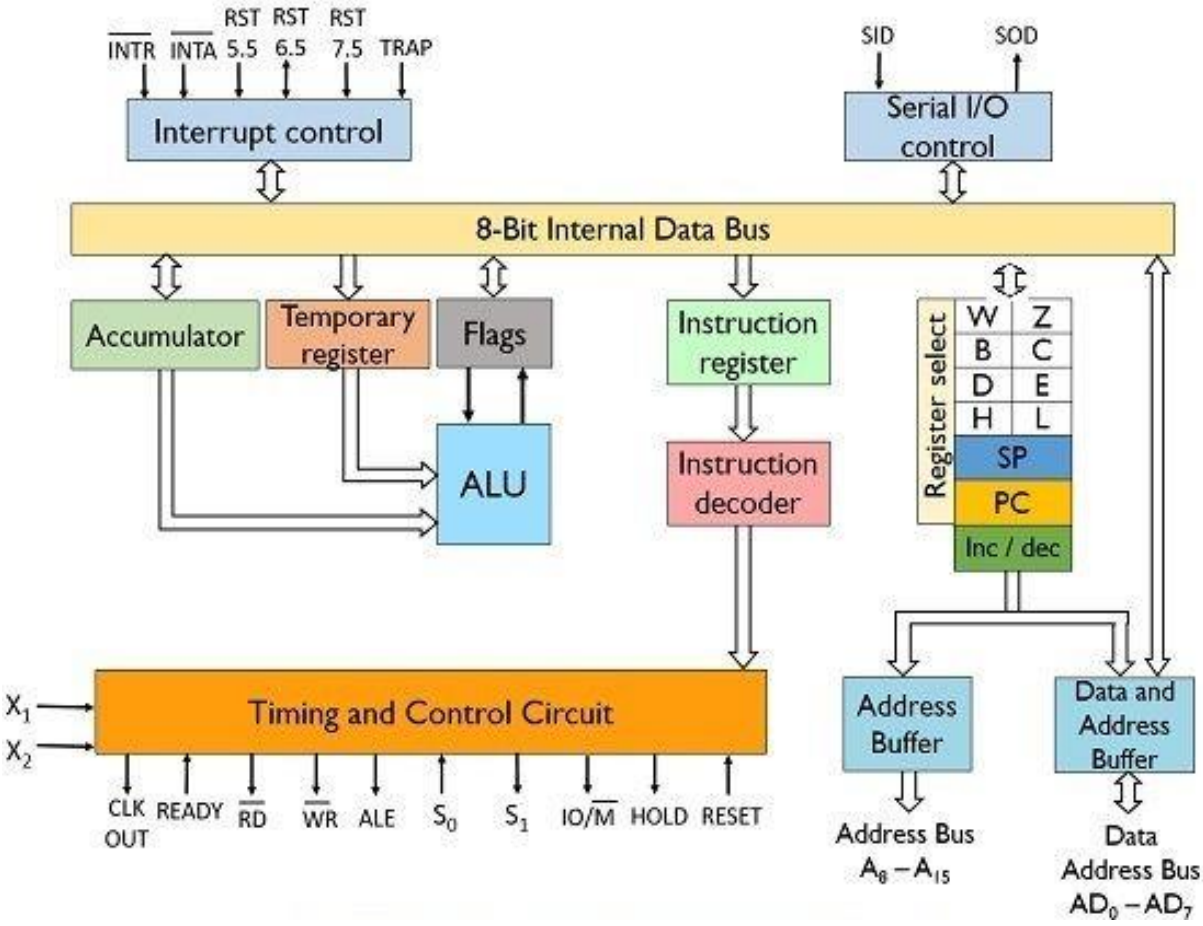
Applications of the 8085:

- **Early Personal Computers:** The 8085 was used in some early personal computers, including hobbyist kits.
- **Educational Trainer:** Due to its relative simplicity, the 8085 was a popular choice for teaching microprocessor fundamentals.
- **Embedded Systems:** In some cases, the 8085 was used in early embedded systems for simple control tasks.

While the 8085 is no longer a major player in the modern computing world, it holds historical significance and serves as a stepping stone to understand the evolution of more powerful microprocessors.

8085 Microprocessor Architecture

The Intel 8085 microprocessor, though not modern, serves as a great introduction to microprocessor architecture. Here's a breakdown of its components and how they work together:



Components of the 8085 Microprocessor:

1. **ALU (Arithmetic Logic Unit):** The heart of the 8085, it performs arithmetic operations (addition, subtraction) and logical operations (AND, OR, NOT) on data.
2. **Registers:** These are small, high-speed memory locations within the CPU that hold data temporarily during program execution. The 8085 has:
    - **Accumulator (A):** An 8-bit register for storing the main operand (data) during ALU operations.
    - **General Purpose Registers (B, C, D, E, H, L):** Six 8-bit registers that can be used individually or paired (B-C, D-E, H-L) to store 16-bit data.
3. **Control Unit (CU):** The brain of the 8085, it fetches instructions from memory, decodes them, and controls the execution process by directing other components.

4. **Program Counter (PC):** A 16-bit register that keeps track of the memory address of the next instruction to be executed. The CU increments the PC after fetching an instruction.
5. **Stack Pointer (SP):** Another 16-bit register that points to the top of a special area of memory called the stack. The stack is used for temporary data storage during subroutine calls and interrupts.
6. **Instruction Register (IR):** An 8-bit register that holds the instruction currently being fetched from memory.
7. **Status Flags:** A set of single-bit flags that reflect the outcome of ALU operations. These flags indicate conditions like Zero (result is 0), Carry (overflow during addition), etc.
8. **Data Bus (8-bit):** A pathway for transferring data between the 8085 and memory or I/O devices (8 bits at a time).
9. **Address Bus (16-bit):** Specifies the memory location for data or instructions (can access up to 64 KB of memory).

10. **Control Bus:** A group of signals used by the CU to control external devices like memory and I/O peripherals.

How the Components Work Together:

1. The CU fetches an instruction from memory using the address pointed to by the PC.
2. The instruction is stored in the IR.
3. The CU decodes the instruction and determines the operation to be performed.
4. If data is needed, it is fetched from memory or registers and loaded into the ALU.
5. The ALU performs the required operation (arithmetic or logical) on the data.
6. The result is stored back in the accumulator or another register.
7. The CU updates the status flags based on the operation's outcome.

8. The PC is incremented to point to the next instruction.

9. Steps 1-8 repeat for each instruction in the program.

This fetch-decode-execute cycle continues until the program terminates. The flags register helps in conditional branching and decision-making within the program.

8085 Pin Diagram:

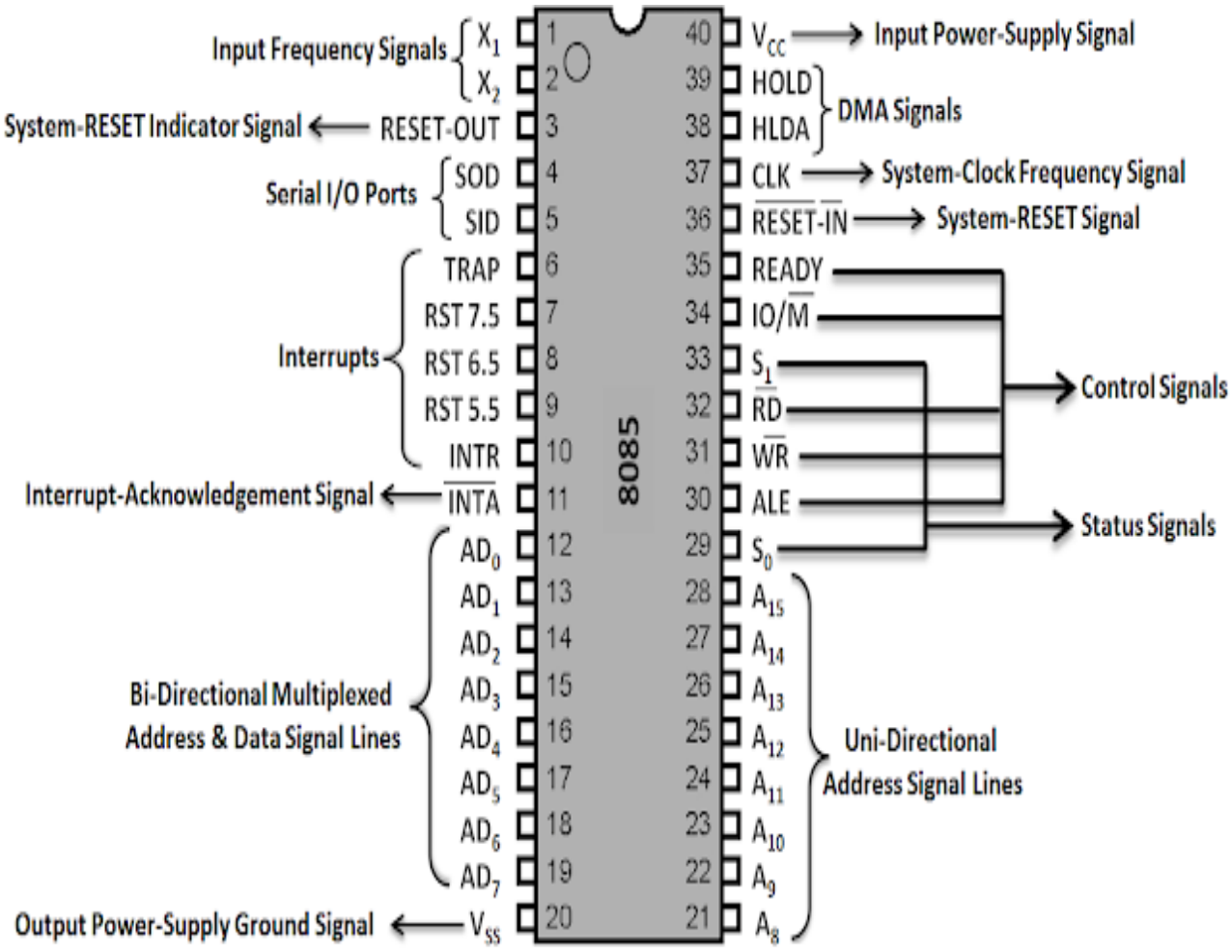


Fig.No.1:Pin-Diagram of 8085 Microprocessor

The 8085 microprocessor communicates with the outside world through a set of 40 pins, each with a specific function. Here's a breakdown of the pin categories and their roles:

Pin Categories and Functions:

1. **Address Bus (A0 - A15):**

- A group of 16 pins (A0 being the least significant and A15 the most significant) used to specify the memory location for data transfer or the address of an I/O device.

2. **Data Bus (AD0 - AD7):**

- An 8-bit bus (AD0 being the least significant and AD7 the most significant) that carries data between the 8085 and memory or peripherals during read/write operations.

3. **Control and Status Signals:**

- A set of pins that control the flow of data and indicate the status of the microprocessor. Some important ones include:

- **READ (RD):** Activated by the 8085 to initiate a read operation from memory or an I/O device.
- **WRITE (WR):** Activated by the 8085 to initiate a write operation to memory or an I/O device.
- **RESET (RESET IN):** An active-low signal that resets the 8085 to its initial state.
- **HOLD and HLDA:** Used for a peripheral device to request and acknowledge a temporary halt of the 8085's operation (useful for data transfer).
- **TRAP, INTR, and RST 5.5, RST 6.5:** Interrupt request lines for external devices to signal the 8085 for attention.

4. **Power Supply and Clock Signals:**

- **Vcc (+5V):** Provides the main power supply to the microprocessor.

- **X1 and X2:** These pins connect to a crystal oscillator circuit that generates the clock signal for the 8085. The internal clock frequency is half the crystal oscillator frequency.

5. **Serial I/O Signals:**

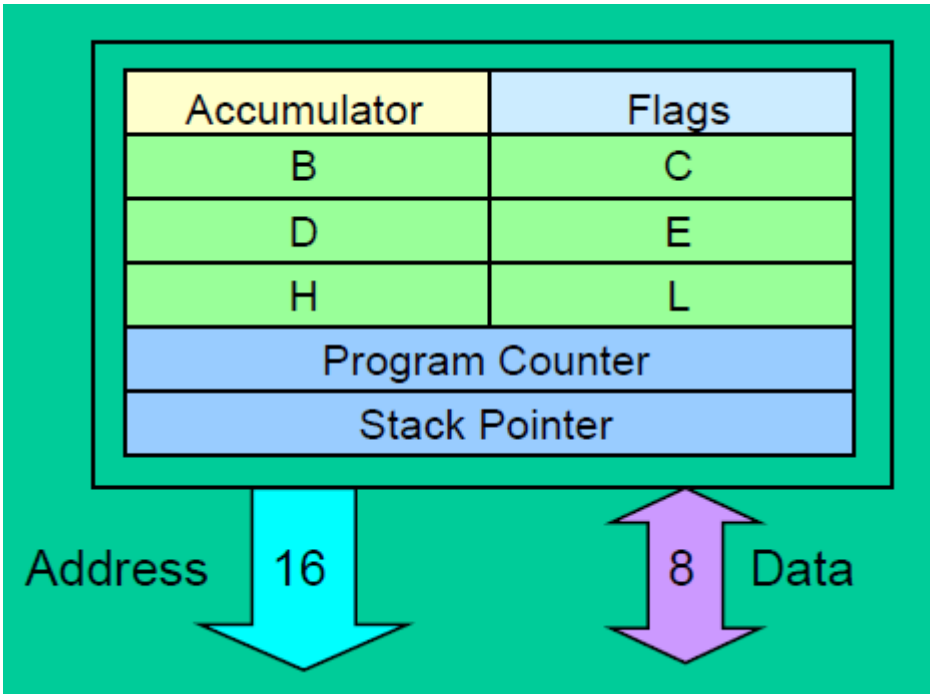
- A set of pins (SID and SOD) used for serial data input and output, allowing the 8085 to communicate with slower peripherals.

Benefits of the Pin Diagram:

- **Understanding Communication:** The diagram clarifies how the 8085 interacts with memory, peripherals, and external devices through specific pins.
- **Circuit Design:** It serves as a guide for connecting the 8085 to other components during circuit design for a microcomputer system.
- **Interrupt Handling:** The interrupt request and acknowledge pins are highlighted, aiding in understanding

how external devices can interrupt the 8085 for time-sensitive tasks.

Register Organization of 8085 Microprocessor:

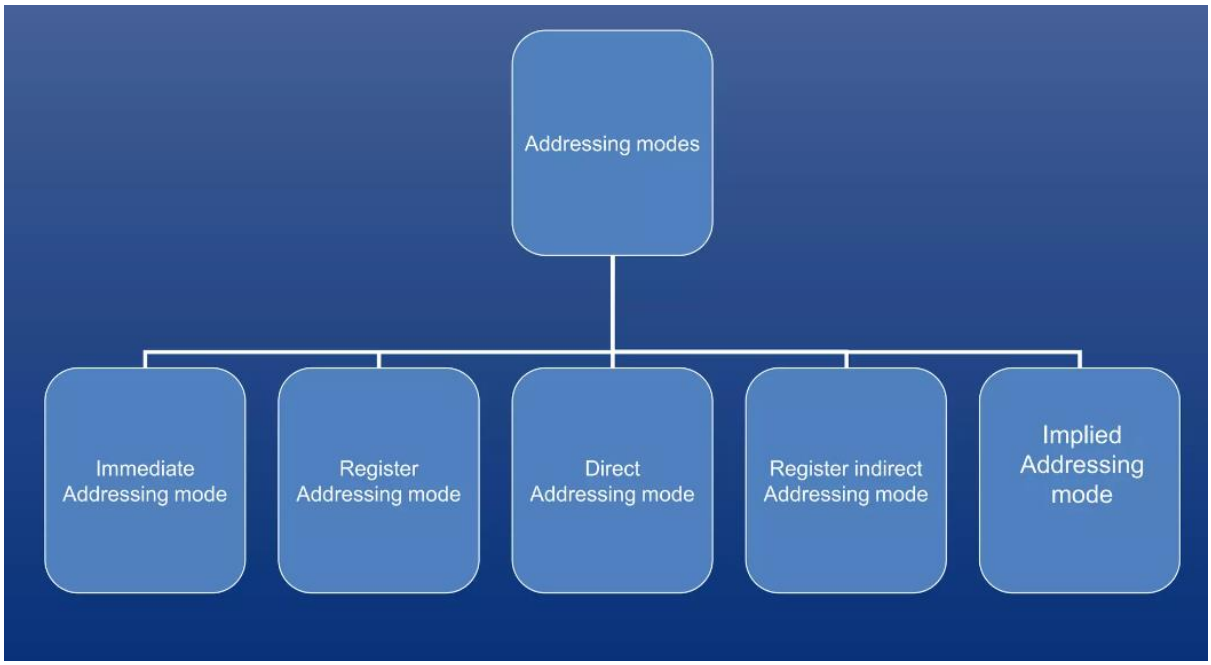


The 8085 microprocessor is an 8-bit microprocessor introduced by Intel in the mid-1970s. It's built on NMOS technology and typically used in embedded systems and computing tasks. Here's a detailed overview of its register organization:

- **Accumulator (A):** The 8-bit accumulator is used to store 8-bit data and in performing arithmetic and logic operations. The results of these operations are generally stored in the accumulator.
- **General Purpose Registers:** The 8085 has six general-purpose registers named B, C, D, E, H, and L. These can be used individually as 8-bit registers or paired together as BC, DE, and HL to perform as 16-bit registers. The HL pair is often used to store memory addresses, effectively making it act as a memory pointer.
- **Program Counter (PC):** This 16-bit register deals with sequencing the execution of instructions. It holds the memory address of the next instruction to be executed.
- **Stack Pointer (SP):** Also a 16-bit register, the stack pointer helps in memory management by pointing to the top of the current stack in memory.

- **Flag Register:** The flag register consists of 5 flip-flops that are set or reset after an arithmetic or logic operation according to data conditions in the accumulator or other registers. The flags are:
    - **Sign (S):** Set if the most significant bit (MSB) of the result is 1.
    - **Zero (Z):** Set if the result is 0.
    - **Auxiliary Carry (AC):** Set if there is a carry out from bit 3 to bit 4 of the result.
    - **Parity (P):** Set if the number of 1s in the result is even.
    - **Carry (CY):** Set if there is a carry during addition or borrow during subtraction/comparison.

Addressing modes of 8085 Microprocessor:



Addressing modes of a microprocessor define how the operand of an instruction is accessed. In 8085, there are five primary addressing modes:

- **Immediate Addressing**: The operand is specified in the instruction itself. For example, **MVI A, 32H** moves the hexadecimal value 32 into the accumulator.
- **Direct Addressing**: The 16-bit address of the operand is specified in the instruction. For example, **LDA 2500H** loads

the accumulator with the content of the memory location 2500H.

- **Register Addressing**: The operand is located in a register. For example, **MOV B, A** copies the content of the accumulator into register B.
- **Register Indirect Addressing**: The instruction specifies a register pair (like HL) which points to the memory location where the operand is stored. For example, **MOV A, M** moves the content of the memory location pointed by HL into the accumulator.
- **Implicit Addressing**: The operand is implied by the opcode itself. For example, the instruction **CMA** (complement accumulator) affects the accumulator directly.

1. Immediate Addressing (IMM):

- **Description:** The operand's value is included directly within the instruction itself. This is the simplest mode and eliminates the need for additional memory access.
- **Example:** MVI A, 30H (Move immediate value 30H to accumulator A)
    - Here, 30H is the operand (data) present within the instruction itself.

2. Register Addressing (REG):

- **Description:** The operand is located in one of the general-purpose registers (B, C, D, E, H, or L) of the 8085. This mode is efficient for data already stored in registers.
- **Example:** MOV B, C (Move data from register C to register B)
    - The operand is the data residing in register C, which is then copied to register B.

3. Direct Addressing (DIR):

- **Description:** The memory address of the operand is explicitly specified in the instruction itself. This mode allows direct access to a specific memory location.
- **Example:** LDA 2000H (Load data from memory location 2000H to accumulator A)
    - The instruction directly points to memory address 2000H, and the data located there is loaded into the accumulator (A).

4. Indirect Addressing (IND):

- **Description:** The address of the operand is not directly mentioned in the instruction but is instead stored in a register pair (HL, DE, or BC).
- **Example:** MVI A, [HL] (Move data from the memory location pointed to by the HL register pair to accumulator A)

- The HL register pair holds the memory address. The data from that memory location is then transferred to the accumulator.

5. Implied Addressing (IMP):

- **Description:** The operand is implicit in the instruction itself, typically referring to the accumulator (A). This mode is used for instructions that operate directly on the accumulator's content.
- **Example:** INR A (Increment the value in accumulator A by 1)
    - The operand (data) is the value in the accumulator, which is incremented by 1 in this case.

Instruction Set of 8085 Microprocessor:

The instruction set of the 8085 microprocessor is a comprehensive collection of commands that the processor can

execute. These instructions are divided into several categories based on their purpose and nature. Here's a detailed breakdown:

1. Data Transfer Instructions

These instructions are used for moving data between registers, from memory to registers, and vice versa without altering the data.

- **MOV**: Move data between registers.
- **MVI**: Move immediate data to register.
- **LXI**: Load register pair immediate.
- **STA**: Store accumulator direct.
- **LDA**: Load accumulator direct.
- **SHLD**: Store H and L direct.
- **LHLD**: Load H and L direct.
- **XCHG**: Exchange H and L with D and E.

2. Arithmetic Instructions

These instructions perform arithmetic operations involving the accumulator and other registers or memory locations.

- **ADD**: Add register or memory to accumulator.
- **ADI**: Add immediate to accumulator.
- **SUB**: Subtract register or memory from accumulator.
- **SUI**: Subtract immediate from accumulator.
- **INR**: Increment register or memory.
- **DCR**: Decrement register or memory.
- **INX**: Increment register pair.
- **DCX**: Decrement register pair.
- **DAD**: Double add register pair to H and L.
- **DAA**: Decimal adjust accumulator.

3. Logical Instructions

Logical operations on the accumulator and other registers or memory locations.

- **ANA**: Logical AND register or memory with accumulator.
- **ANI**: Logical AND immediate with accumulator.
- **XRA**: Logical XOR register or memory with accumulator.
- **XRI**: Logical XOR immediate with accumulator.
- **ORA**: Logical OR register or memory with accumulator.
- **ORI**: Logical OR immediate with accumulator.
- **CMP**: Compare register or memory with accumulator.
- **CPI**: Compare immediate with accumulator.
- **RLC**: Rotate accumulator left.
- **RRC**: Rotate accumulator right.
- **RAL**: Rotate accumulator left through carry.
- **RAR**: Rotate accumulator right through carry.

- **CMA**: Complement accumulator.
- **CMC**: Complement carry.
- **STC**: Set carry.

4. Branch Instructions

These instructions alter the flow of execution.

- **JMP**: Unconditional jump to specified address.
- **JC**: Jump on carry.
- **JNC**: Jump on no carry.
- **JZ**: Jump on zero.
- **JNZ**: Jump on not zero.
- **JP**: Jump on positive.
- **JM**: Jump on minus.
- **JPE**: Jump on parity even.
- **JPO**: Jump on parity odd.
- **CALL**: Call subroutine.

- **RET**: Return from subroutine.
- **RST**: Restart.

5. Control Instructions

Instructions for controlling the operation of the microprocessor.

- **NOP**: No operation.
- **HLT**: Halt.
- **DI**: Disable interrupts.
- **EI**: Enable interrupts.
- **SIM**: Set interrupt mask.
- **RIM**: Read interrupt mask.

Each of these instructions enables the 8085 microprocessors to perform a wide range of operations, from simple data transfers to complex arithmetic calculations, logical decisions, and controlling the sequence of operations. These instructions are fundamental to programming the microprocessor for various applications.

Program to Add Two Numbers

Objective: Add two numbers stored in memory and store the result in another memory location.

; Assuming numbers are stored at memory locations 2050H and 2051H

; Result will be stored at 2052H

LXI H, 2050H; Load HL pair with address 2050H

MOV A, M; Move the content of address 2050H into accumulator

INX H; Increment HL pair to point to 2051H

ADD M; Add content of address 2051H to the accumulator

INX H; Increment HL pair to point to 2052H

MOV M, A; Move the result from accumulator to address 2052H

HLT; Halt the program

Explanation:

- The **LXI H, 2050H** instruction loads the HL register pair with the starting address where the first number is stored.
- **MOV A, M** moves the number at this address into the accumulator.
- **INX H** increments the HL pair to point to the next memory location where the second number is stored.
- **ADD M** adds this second number to the accumulator.
- Another **INX H** moves the HL pair to the location where the result will be stored.
- **MOV M, A** moves the result from the accumulator to this memory location.
- **HLT** stops the program.

Program to Compare Two Numbers

Objective: Compare two numbers and set the zero flag if they are equal.

; Assuming numbers are stored at memory locations 3000H and 3001H

LXI H, 3000H; Load HL pair with address 3000H

MOV A, M; Move the content of address 3000H into accumulator

INX H; Increment HL pair to point to 3001H

CMP M; Compare content of address 3001H with accumulator

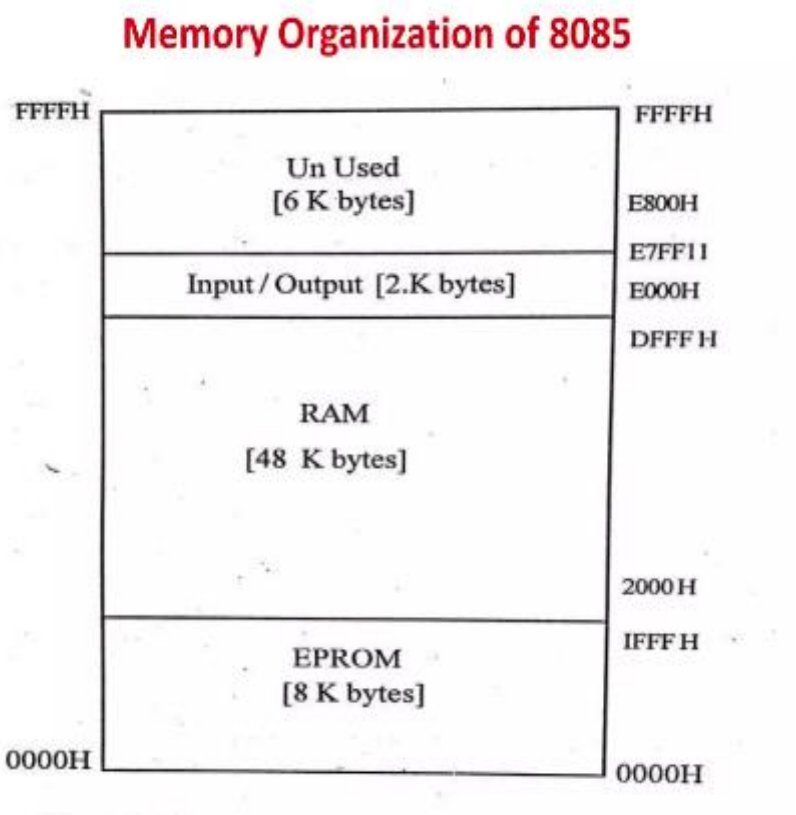
HLT; Halt the program

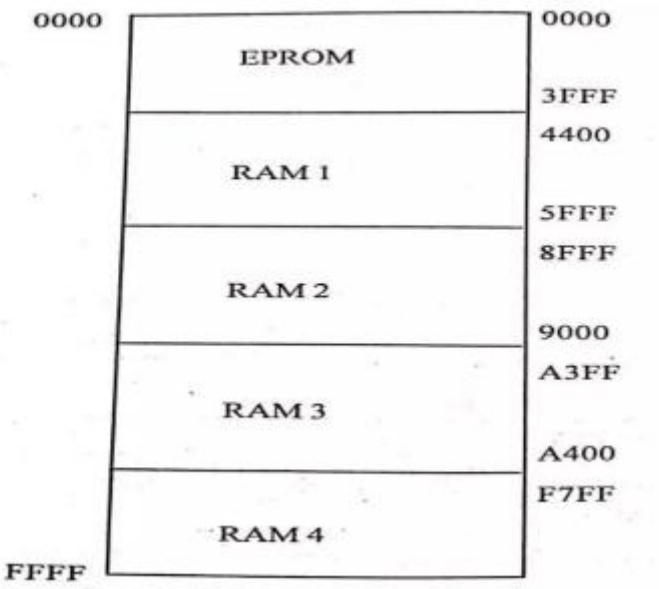
Explanation:

- **LXI H, 3000H** loads the HL pair with the address of the first number.
- **MOV A, M** moves the first number into the accumulator.
- **INX H** adjusts the HL pair to point to the second number.

- **CMP M** compares the second number with the first (in the accumulator). If they are equal, the zero flag (Z) is set.
- **HLT** halts the program.

Memory Organization of 8085 Microprocessor:





The 8085 microprocessor has a specific memory organization that defines how it interacts with memory and stores data. Here's a breakdown of the key points:

Memory Capacity:

- The 8085 has a 16-bit address bus, allowing it to theoretically address up to 2^16 (65,536) bytes of memory.

Memory Types:

- The 8085 can access two main types of memory:
    - **Program Memory (ROM):** Stores the program instructions that the 8085 executes. This memory is

typically read-only (ROM) and cannot be modified during normal operation.

- **Data Memory (RAM):** Stores data used by the program, such as variables, temporary results, and input/output (I/O) data. This memory is read-write (RAM), allowing data to be changed during program execution.

Memory Map:

- The entire memory address space is not solely dedicated to program or data.
- A memory map defines how the available addresses are allocated to different memory components.
- This map is crucial for ensuring that the 8085 can access instructions and data without conflicts.
- The specific memory map for a system using the 8085 depends on the design and the amount of memory

installed. However, it typically includes dedicated regions for program memory, data memory, and I/O devices.

Memory Addressing:

- The 8085 uses its 16-bit address bus to specify the memory location for data transfer operations (read or write).
- The addressing mode used in an instruction determines how the operand's (data's) memory address is specified.
- The 8085 supports various addressing modes, including immediate, register, direct, indirect, and implied, offering flexibility in accessing data from different locations.

Benefits of Understanding Memory Organization:

- **Effective Programming:** Programmers need to be aware of the memory map and addressing modes to write code that correctly accesses instructions and data.
- **System Design:** When designing a system using the 8085, understanding memory organization is crucial for

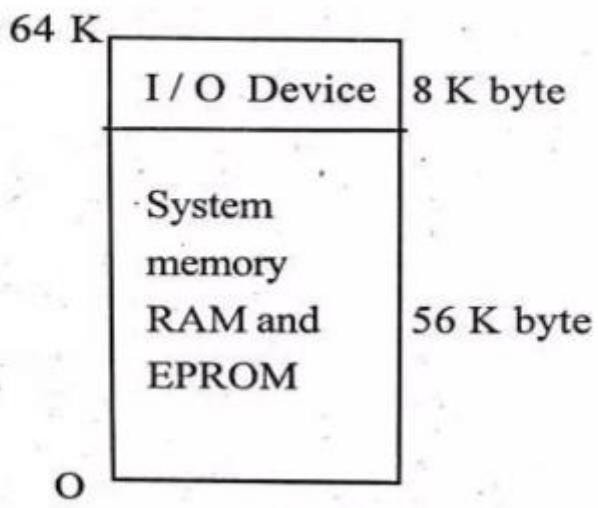
allocating memory resources and ensuring smooth interaction between the processor and memory components.

- **Appreciation for Modern Systems:** While the 8085 has a simpler memory organization, it lays the groundwork for understanding the more complex memory hierarchies and addressing mechanisms used in modern processors.

Memory-mapped I/O (Input/Output)

- In the 8085 microprocessor, memory-mapped I/O (Input/Output) is a technique for interfacing with external devices.
- It treats these devices as if they were memory locations. The processor can access and manipulate data to/from these devices using the same instructions it uses for memory access.

Memory Mapped I/O



Concept:

- Instead of dedicated instructions for I/O operations, the 8085 uses the same address space for both memory and I/O devices.
- Specific memory addresses are reserved for I/O devices like keyboards, displays, or sensors. These addresses are called I/O ports.
- The processor interacts with these I/O ports using special control signals (like I/O Read (RD*) and I/O Write (WR*))

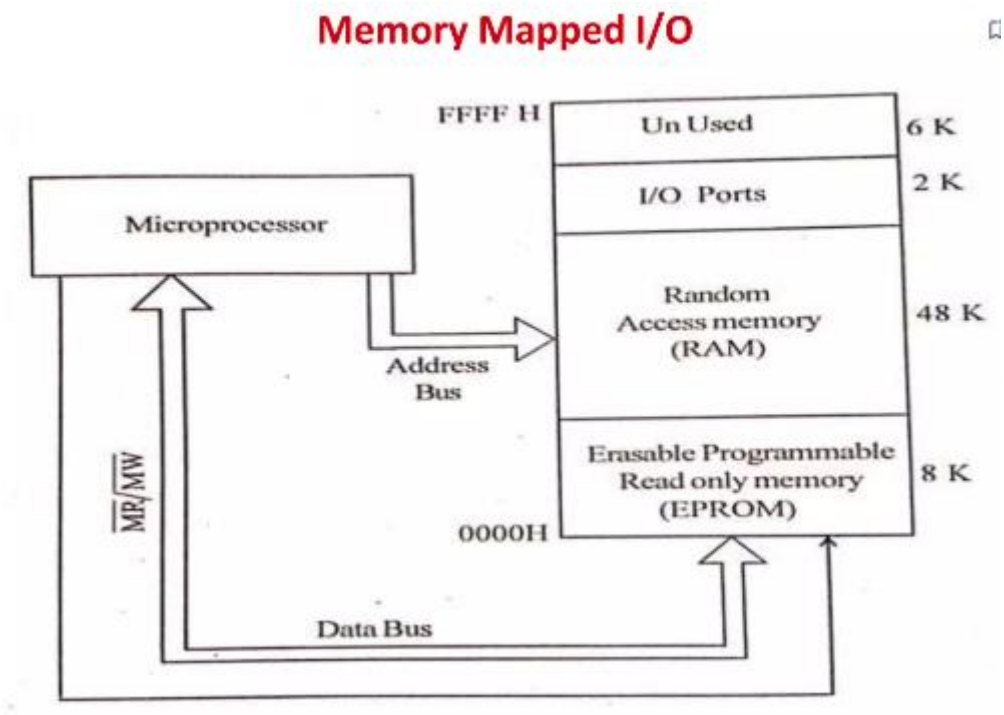
to indicate the type of operation (read or write) and data transfer.

Benefits:

- **Simpler Instruction Set:** Memory-mapped I/O eliminates the need for a separate set of I/O instructions, simplifying the 8085's architecture.
- **Flexibility:** Any memory access instruction can be used for I/O operations, offering flexibility in data transfer.

Drawbacks:

- **Limited Addressing Modes:** The 8085 can only use a limited set of addressing modes for I/O ports compared to memory access. This can restrict how the processor interacts with certain devices.
- **Potential Conflicts:** If memory addresses are not carefully allocated, there's a risk of conflicts between I/O ports and actual memory locations.



8085 Programming:

Program 1: Addition of Two Numbers (Data Transfer & Arithmetic Instructions)

; Program to add two 8-bit numbers and store the result

MVI A, 30H ; Load immediate value 30H into accumulator A (Immediate)

MVI B, 45H ; Load immediate value 45H into register B (Immediate)

ADD B ; Add the content of register B to accumulator A (Arithmetic)

STA 2000H ; Store the result (accumulator A) in memory location 2000H (Data Transfer)

HLT ; Halt the processor (Machine Control)

Explanation:

1. This program loads two 8-bit values (30H and 45H) into the accumulator (A) and register B, respectively, using immediate addressing.
2. The ADD B instruction performs addition, adding the value in B to the value in A and storing the result back in A.
3. The result in the accumulator is then stored in memory location 2000H using direct addressing.
4. Finally, the HLT instruction halts the processor.

Program 2: Looping and Increment (Branching & Arithmetic Instructions)

; Program to increment a value in memory 10 times using a loop

LXI H, 2000H ; Load immediate value 2000H (address) into HL pair (Data Transfer)

MVI C, 10H ; Load immediate value 10H (loop counter) into register C (Immediate)

LOOP: ; Label for the loop

INR A ; Increment the value in accumulator A (Arithmetic)

DCR C ; Decrement the loop counter in register C (Arithmetic)

JNZ LOOP ; Jump to LOOP if C is not zero (Branching)

HLT ; Halt the processor (Machine Control)

Explanation:

1. This program initializes a loop counter (C) to 10H and sets the HL register pair to point to memory location 2000H.
2. The LOOP label marks the beginning of the loop.

3. Inside the loop, the value in memory location 2000H (pointed to by HL) is incremented using INR A (assuming the value is initially stored in the accumulator).
4. The loop counter (C) is decremented using DCR C.
5. The JNZ LOOP instruction checks if the loop counter is zero. If not zero (NZ), it jumps back to the LOOP label, continuing the loop.
6. Once the loop counter reaches zero, the program halts using HLT.

3. Write an assembly language program for the 8085 microprocessor that multiplies two 8-bit numbers stored in memory locations 4200H and 4201H, and then stores the result (which could be a 16-bit number) in memory locations 4202H and 4203H.

```
; Multiplication of two 8-bit numbers stored at 4200H and 4201H
; Result will be stored in 4202H (low byte) and 4203H (high byte)

LXI H, 4200H    ; Load HL pair with the address 4200H
MOV A, M        ; Load the content of 4200H (first number) into Accumulator
MOV B, A        ; Copy it to B register for multiplication
INX H           ; Increment HL to point to 4201H
MOV C, M        ; Load the content of 4201H (second number) into C register
MVI D, 00H      ; Clear D register to use it for storing high byte of result
MVI E, 00H      ; Clear E register to use it for storing low byte of result

MULTIPLY:       ; Start multiplication loop
ORA C           ; OR C with accumulator to check if C is 0
RZ              ; If zero, multiplication is done
RAR             ; Rotate right through carry
JC ADDITION     ; If carry is 1 (LSB of C is 1), add B to DE
```

```
SHLD 4202H      ; Store intermediate low byte result to memory directly

DAD B           ; Double the B register (shift left or *2 operation)

JMP MULTIPLY    ; Repeat the loop until C is 0


ADDITION:       ; Addition routine if C's bit was 1

MOV A, E        ; Move E to A for addition

ADD B           ; Add B to A

MOV E, A        ; Store result back to E

MOV A, D        ; Adjust the carry to the high byte

ADC A, 00H      ; Add carry to D

MOV D, A        ; Store back to D

SHLD 4202H      ; Store the new intermediate result to memory

JMP MULTIPLY    ; Continue multiplication process


; Storing result

STAX D          ; Store the final result high byte to 4203H
```

```
MOV A, E        ; Move low byte to accumulator

STA 4202H       ; Store the final result low byte to 4202H

HLT             ; Halt the program
```

Explanation

1. **Initialize Pointers and Registers:**
    - The HL register pair is loaded with the address where the first number is stored (4200H).
    - The content of this address is moved into the accumulator and then copied to register B.
    - The HL pointer is incremented to point to the next memory address where the second number is stored (4201H), and this number is loaded into register C.
    - D and E registers are cleared to store the result of the multiplication.
2. **Multiplication Logic:**

- The program checks if the multiplier (C register) is zero. If zero, multiplication is complete.
- The program checks the least significant bit (LSB) of C to decide whether to add B to the result stored in DE. It does this by rotating C and checking the carry.
- If the LSB was 1 (carry is set after rotation), it adds B to the result stored in DE.
- B is then doubled (shifted left) for the next bit of the multiplier.
- This process repeats until all bits of the multiplier have been processed.

3. **Storing the Result:**
    - The final 16-bit product is stored across two memory locations, with the low byte in 4202H and the high byte in 4203H.

2. Write a 8085 assembly language program to add two numbers of 16-bit data stored in memory from 4200H to 4203H. The data is stored such that LSB first and then MSB. Store the results from 4204H to 4206H.

```
; Adding two 16-bit numbers stored in memory
; First number is stored at 4200H (LSB) and 4201H (MSB)
; Second number is stored at 4202H (LSB) and 4203H (MSB)
; Result to be stored at 4204H (LSB), 4205H, and 4206H (MSB of carry if any)

LXI H, 4200H      ; Load HL pair with address of first number's LSB
MOV A, M          ; Load LSB of the first number into A
INX H             ; Increment HL to point to MSB of the first number
MOV B, M          ; Load MSB of the first number into B
LXI H, 4202H      ; Load HL pair with address of second number's LSB
MOV C, M          ; Load LSB of the second number into C
```

```
INX H           ; Increment HL to point to MSB of the second number

MOV D, M        ; Load MSB of the second number into D


; Perform addition of LSBs

MOV A, C        ; Move LSB of the second number into A

ADD A, M        ; Add it to LSB of the first number

STA 4204H       ; Store the result (LSB of final result) at 4204H


; Perform addition of MSBs

MOV A, D        ; Move MSB of the second number into A

ADC B           ; Add it to MSB of the first number with carry

STA 4205H       ; Store the result (MSB of final result) at 4205H


; Check for any carry left after adding MSBs

MOV A, 00       ; Clear A to check for carry

ADC A, 00       ; Add carry (if any)
```

```
STA 4206H       ; Store the final carry (if any) at 4206H

HLT             ; Halt the program
```

Explanation

1. **Initialization and Memory Loading:**
    - The **LXI H, 4200H** instruction initializes the HL register pair to point to the address of the LSB of the first 16-bit number.
    - **MOV A, M** and **MOV B, M** load the LSB and MSB of the first number into registers A and B, respectively.
2. **Loading Second Number:**
    - The program then loads the second number similarly into registers C and D, using the same method as for the first number.
3. **Addition of LSBs:**
    - The accumulator is loaded with the LSB of the second number (**MOV A, C**), then added to the LSB of the first number. The

result is stored directly into memory location 4204H using **STA 4204H**.

4. **Addition of MSBs:**
    - Next, the MSB of the second number is added to the MSB of the first number, including any carry from the addition of LSBs (**ADC B**). The result is stored in memory location 4205H.
5. **Storing Potential Carry:**
    - Finally, any additional carry generated from the addition of the MSBs is stored at memory location 4206H. This is checked by adding zero to the accumulator with the carry bit considered (**ADC A, 00**).

4. Write a subroutine program to exchange the contents of BC-pair and DE-pair of 8085 microprocessor.

```
; Subroutine to exchange the contents of BC and DE register pairs

EXCHANGE_BC_DE:
```

```
    PUSH B       ; Save current contents of BC register pair on stack
    MOV B, D     ; Move D to B
    MOV C, E     ; Move E to C
    POP H        ; Retrieve original B into H (temporarily)
    MOV D, B     ; Move original B (now in H) to D
    MOV E, C     ; Move original C (now in L) to E
    XCHG         ; Exchange HL with DE to finalize moving original BC to DE
    RET          ; Return from subroutine

; Example usage in a main program
MAIN_PROGRAM:
    LXI B, 1234H  ; Load BC register pair with value 1234H
    LXI D, 5678H  ; Load DE register pair with value 5678H

    CALL EXCHANGE_BC_DE   ; Call the subroutine to exchange BC and DE

    HLT                   ; Halt the execution (for testing)
```

Explanation

1. **Pushing BC to Stack:**
    - **PUSH B**: This instruction saves the current contents of the BC register pair onto the stack. This is essential to free up the BC registers for swapping without losing their original contents.
2. **Copying DE to BC:**
    - **MOV B, D** and **MOV C, E**: These instructions move the contents of the DE register pair into the BC register pair. After this operation, BC will hold the original values of DE.
3. **Retrieving BC from Stack to DE:**
    - **POP H**: Pops the top of the stack into the H register. Because of the way the 8085's stack operations work, this instruction actually retrieves the original value of B into H and the original value of C into L.
    - **XCHG**: This exchanges the contents of the DE register pair with the HL register pair. Since HL now contains the original BC values and BC contains the original DE values, the exchange is effectively complete.
4. **Return from Subroutine:**

- **RET**: Returns control to the calling function or section of the program. This marks the end of the subroutine.

5. **Example Usage in Main Program:**

   - The example part **MAIN_PROGRAM** demonstrates initializing BC and DE with specific values, calling the **EXCHANGE_BC_DE** subroutine, and then halting the system to end the program. This part is crucial for testing and verifying that the subroutine works as intended.

5. Write an 8085 assemble language program to load registers A,B,C,D,E with value FFH.

```
; Program to load FFH into registers A, B, C, D, and E


MVI A, FFH   ; Load FFH into register A

MVI B, FFH   ; Load FFH into register B

MVI C, FFH   ; Load FFH into register C

MVI D, FFH   ; Load FFH into register D

MVI E, FFH   ; Load FFH into register E

HLT          ; Halt the program
```

Explanation:

1. The program uses the MVI (Move Immediate) instruction five times, each time loading the value FFH (hexadecimal for 255) into a different register.
2. The first MVI A, FFH instruction loads FFH into the accumulator (register A).
3. Subsequent MVI instructions with B, C, D, and E as operands load the same value into those registers, effectively setting them all to FFH.
4. Finally, the HLT instruction halts the processor.

This program is very simple and demonstrates the use of the MVI instruction for immediate data transfer.