

## 1. Write down the different features of 8085 microprocessor.

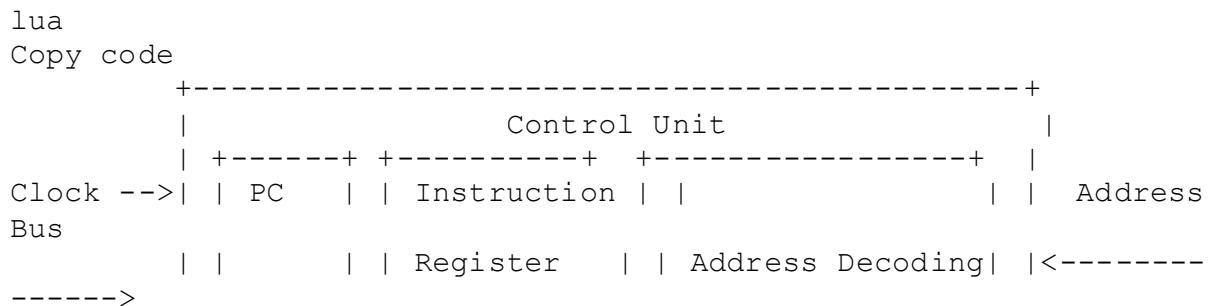
The Intel 8085 microprocessor, released in 1976, was a popular 8-bit microprocessor known for its versatility and widespread use in early computing and embedded systems. Here are the key features of the 8085 microprocessor:

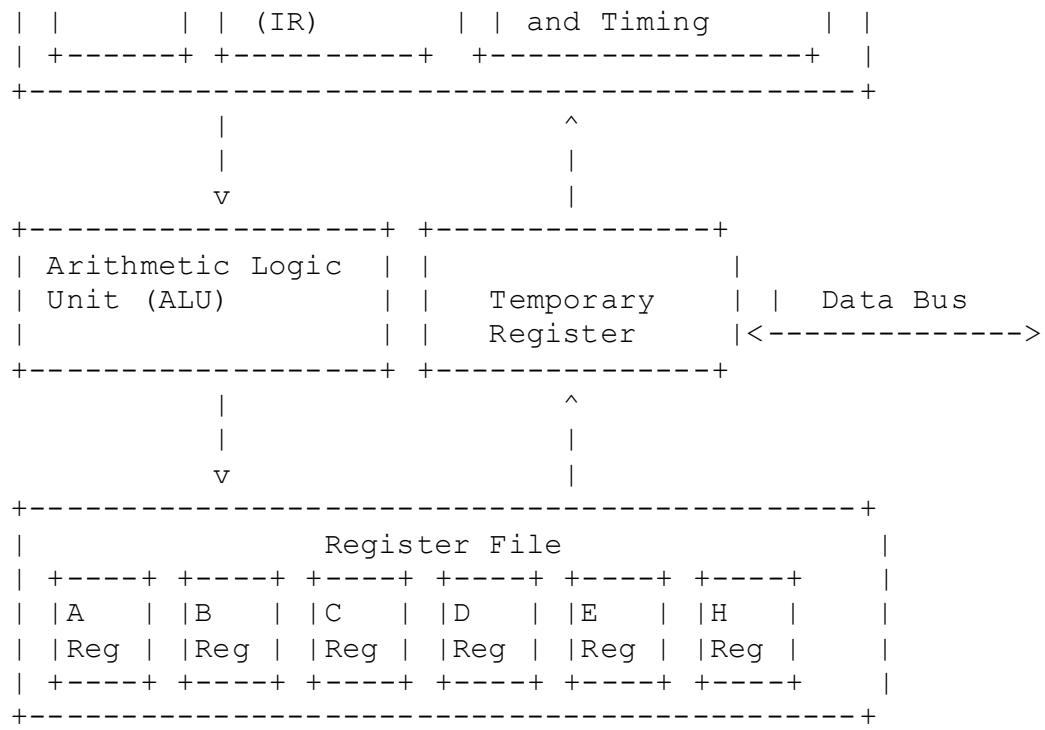
1. **Architecture:** It is an 8-bit microprocessor with a 16-bit address bus capable of addressing up to 64 KB of memory.
2. **Registers:** It has six primary registers:
  - Accumulator (A)
  - Temporary register (B)
  - Program Counter (PC)
  - Stack Pointer (SP)
  - Flag Register (F)
  - Memory Address Register (M)Additionally, it has two sets of 8-bit general-purpose registers (B, C; D, E; H, L) that can be used individually or paired for 16-bit operations.
3. **Clock Speed:** Typically operates at a clock frequency of 3 MHz.
4. **Instruction Set:** Includes a rich set of 74 instructions categorized into data transfer, arithmetic, logical, control transfer, and input/output operations.
5. **Addressing Modes:** Supports various addressing modes including direct addressing, indirect addressing, immediate addressing, and register indirect addressing.
6. **Interrupts:** Supports five interrupts: TRAP, RST 7.5, RST 6.5, RST 5.5, and INTR.
7. **Power Supply:** Requires a single +5V DC supply voltage.
8. **Data Bus:** It has an 8-bit bidirectional data bus.
9. **Pin Configuration:** The 8085 microprocessor has a 40-pin dual in-line package (DIP).
10. **Technology:** Fabricated using NMOS (N-channel Metal Oxide Semiconductor) technology.
11. **System Support:** Interfaces with standard memory and I/O devices, suitable for various embedded applications and early personal computers.
12. **Compatibility:** It was widely used in various systems and had significant software and hardware support during its prime.

These features made the 8085 microprocessor a cornerstone of early computing and embedded systems development, laying the groundwork for subsequent advancements in microprocessor technology.

## 2. Architecture of Intel 8085 Microprocessor with Block Diagram

The architecture of the Intel 8085 microprocessor can be illustrated with the following block diagram:





- **Control Unit (CU):** Responsible for managing the operations of the microprocessor. It controls the flow of data and instructions within the processor.
- **Instruction Register (IR):** Holds the current instruction being executed.
- **Arithmetic Logic Unit (ALU):** Performs arithmetic and logical operations such as addition, subtraction, AND, OR, etc.
- **Register File:** Contains various registers like the Accumulator (A), Temporary Registers (B, C, D, E, H, L), and special-purpose registers (Program Counter, Stack Pointer).
- **Address Bus:** Transfers addresses from the microprocessor to memory or peripherals.
- **Data Bus:** Transfers data between the microprocessor, memory, and peripherals.

### 3. Functions of Internal Registers of 8085 Microprocessor

The internal registers of the 8085 microprocessor serve specific functions:

- **Accumulator (A):** Primary register for arithmetic and logical operations. Results of most operations are stored here.
- **General-Purpose Registers (B, C; D, E; H, L):** Used for storing intermediate data during calculations and for various other purposes.
- **Program Counter (PC):** Holds the address of the next instruction to be fetched from memory.
- **Stack Pointer (SP):** Points to the top of the stack in memory, used for managing subroutine calls and interrupts.
- **Flag Register (F):** Contains status flags (Sign, Zero, Auxiliary Carry, Parity, Carry) that reflect the outcome of arithmetic and logical operations.

### 4. Operation of Arithmetic and Logic Unit (ALU) of 8085 Microprocessor

The ALU in the 8085 microprocessor performs arithmetic and logical operations on data. It can:

- Add and subtract numbers stored in registers or memory.
- Perform logical operations such as AND, OR, XOR, and complement (NOT).

- Handle data transfer operations like moving data between registers.

The ALU processes data based on instructions fetched from memory. It uses the accumulator and other registers for operands and stores results back in the accumulator or designated registers.

## 5. Operation of Control Unit (CU) of 8085 Microprocessor

The Control Unit (CU) of the 8085 microprocessor manages the execution of instructions. Its functions include:

- Fetching instructions from memory into the Instruction Register (IR).
- Decoding instructions to determine the operation to be performed.
- Initiating and coordinating the operations of the ALU, register file, and other components.
- Generating control signals that regulate the flow of data and operations within the microprocessor.

## 6. Pin Diagram of 8085 Microprocessor

The pin diagram of the 8085 microprocessor typically includes the following important pins:

- Address Bus (A15-A0)**: Output lines for addresses.
- Data Bus (D7-D0)**: Bidirectional lines for data transfer.
- Control Bus (various control signals)**: Includes signals like RD, WR, IO/M, etc., for control of data and address buses.
- Power Supply and Clock Lines**: VCC, GND, and CLK.

## 7. Program Status Word (PSW) in 8085 Microprocessor

The Program Status Word (PSW) in the 8085 microprocessor refers to the combined state of the accumulator and flags. It consists of:

- Accumulator (A)**: 8-bit register for arithmetic and logical operations.
- Flags (Sign, Zero, Auxiliary Carry, Parity, Carry)**: Status bits that indicate various conditions after an operation.

Example: If after an arithmetic operation the result is zero, the Zero flag in the PSW will be set to 1, indicating that the result of the operation was zero.

## 8. Flag Register in 8085 Microprocessor

The Flag Register in the 8085 microprocessor is part of the PSW and contains 5 flags:

- Sign (S)**: Indicates the sign of the result after an arithmetic operation (1 for negative, 0 for positive).
- Zero (Z)**: Indicates if the result of an operation is zero (1 if result is zero, 0 otherwise).
- Auxiliary Carry (AC)**: Used for BCD (Binary Coded Decimal) arithmetic.
- Parity (P)**: Indicates the parity (even or odd) of the result.
- Carry (CY)**: Indicates if there was a carry-out from the most significant bit after an arithmetic operation.

Example: After an addition operation if the result is greater than 255 (in 8-bit arithmetic), the Carry flag will be set to 1 to indicate the carry-out.

## 9. Different Addressing Modes of 8085 Microprocessor

The 8085 microprocessor supports various addressing modes:

- **Immediate Addressing:** Operand is directly specified in the instruction. Example: MVI A, 05H.
- **Direct Addressing:** Operand is the memory location directly specified in the instruction. Example: LDA 2000H.
- **Register Addressing:** Operand is in a register specified in the instruction. Example: MOV A, B.
- **Register Indirect Addressing:** Operand is in a memory location addressed by a register pair. Example: MOV A, M.
- **Implicit Addressing:** Address is implied by the instruction. Example: HLT (Halts the processor).

## 10. DMA Related Signals Present in 8085

The Direct Memory Access (DMA) related signals in the 8085 microprocessor include:

- **Hold Acknowledge (HLDA):** Indicates that the 8085 is relinquishing control of the buses to allow DMA transfers.
- **Hold (HOLD):** Initiates a DMA request to pause the microprocessor.
- **DMA Acknowledge (DACK):** Acknowledges a DMA request from a DMA controller.
- **DMA Request (DREQ):** Requests control of the buses from the microprocessor for DMA transfers.

These signals facilitate efficient data transfer between memory and peripherals without involving the microprocessor in every transfer cycle.

These explanations provide a foundational understanding of the Intel 8085 microprocessor's architecture, operation, and features. If you have more specific questions or need further details on any topic, feel free to ask!

Let's delve into each of these topics related to the Intel 8085 microprocessor instructions:

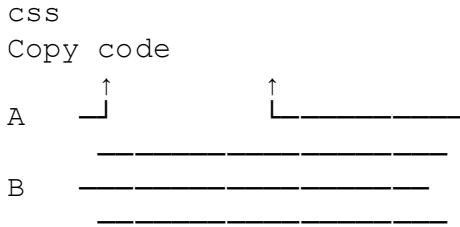
## 11. Instruction Cycle and Machine Cycle of Intel 8085

- **Instruction Cycle:** The execution of an instruction in the 8085 microprocessor typically involves fetching, decoding, executing, and optionally storing the results. It consists of the following phases:
  1. **Fetch:** The instruction is fetched from memory into the Instruction Register (IR).
  2. **Decode:** The control unit decodes the instruction to determine the operation to be performed and the operands involved.
  3. **Execute:** The ALU performs the actual operation specified by the instruction.
  4. **Store:** Results are stored back in registers or memory, depending on the instruction.
- **Machine Cycle:** The machine cycle is the basic operation performed by the microprocessor in one clock period. It is divided into:
  1. **Opcode Fetch Cycle:** Fetching the opcode (operation code) of the instruction from memory.
  2. **Memory Read Cycle:** If operands are involved, fetching data from memory.
  3. **Memory Write Cycle:** Storing results back into memory if required.

## 12. Timing Diagram of Intel 8085 for the Instruction MOV A, B

For the instruction `MOV A, B`, which moves the content of register B into register A:

- **Timing Diagram:**



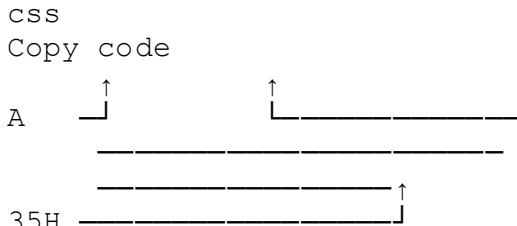
- **Explanation:**

- T1: Opcode fetch cycle. The opcode for `MOV A, B` is fetched from memory.
- T2: Execution cycle. The content of register B is moved to register A.
- The diagram shows the transfer of data from register B to register A during the execution phase (T2).

## 13. Timing Diagram of Intel 8085 for the Instruction MVI A, 35H

For the instruction `MVI A, 35H`, which loads immediate data `35H` into register A:

- **Timing Diagram:**



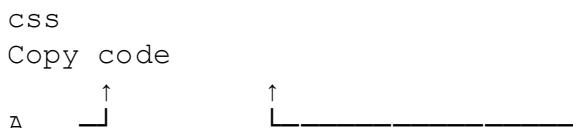
- **Explanation:**

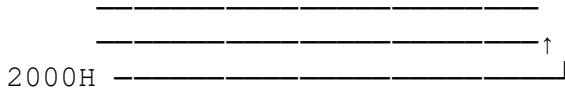
- T1: Opcode fetch cycle. The opcode for `MVI A, 35H` is fetched from memory.
- T2: Memory read cycle. The immediate data `35H` is fetched from memory.
- T3: Execution cycle. The immediate data `35H` is loaded into register A.
- The diagram shows the immediate data (`35H`) being transferred to register A during the execution phase (T3).

## 14. Timing Diagram of Intel 8085 for the Instruction LDA 2000H

For the instruction `LDA 2000H`, which loads the accumulator with data from memory location `2000H`:

- **Timing Diagram:**





- **Explanation:**
  - **T1:** Opcode fetch cycle. The opcode for LDA 2000H is fetched from memory.
  - **T2:** Memory read cycle. The data from memory location 2000H is fetched.
  - **T3:** Execution cycle. The data fetched from memory (at 2000H) is loaded into the accumulator (register A).
  - The diagram illustrates the data from memory (2000H) being transferred to register A during the execution phase (T3).

## 15. ADD, ADI, SUB, SUI Instructions with Examples

- **ADD A, B:** Adds the content of register B to the accumulator A. Example:

```

css
Copy code
MOV A, 25H ; A = 25H
MOV B, 35H ; B = 35H
ADD B       ; A = A + B = 25H + 35H = 5AH

```

- **ADI A, 35H:** Adds immediate data 35H to the accumulator A. Example:

```

css
Copy code
MOV A, 25H ; A = 25H
ADI 35H    ; A = A + 35H = 25H + 35H = 5AH

```

- **SUB A, B:** Subtracts the content of register B from the accumulator A. Example:

```

css
Copy code
MOV A, 50H ; A = 50H
MOV B, 30H ; B = 30H
SUB B      ; A = A - B = 50H - 30H = 20H

```

- **SUI A, 10H:** Subtracts immediate data 10H from the accumulator A. Example:

```

css
Copy code
MOV A, 50H ; A = 50H
SUI 10H    ; A = A - 10H = 50H - 10H = 40H

```

## 16. DAD and DAA Instructions with Examples

- **DAD RP:** Double Add instructions for register pairs (RP). Adds the content of the specified register pair to the HL register pair. Example:

```

css
Copy code
LXI H, 2050H ; HL = 2050H

```

```
LXI B, 1020H ; BC = 1020H  
DAD B ; HL = HL + BC = 2050H + 1020H = 3070H
```

- **DAA:** Decimal Adjust Accumulator. Adjusts the result of an addition operation in BCD (Binary Coded Decimal) format. Example:

```
css  
Copy code  
MOV A, 29H ; A = 29H (BCD)  
MOV B, 15H ; B = 15H (BCD)  
ADD B ; A = A + B = 29H + 15H = 3EH (BCD)  
DAA ; Adjust A to correct BCD format (after DAA, A  
= 44H)
```

## 17. MOV, LXI, MVI, LDA, and STA Instructions with Examples

- **MOV A, B:** Moves the content of register B to register A. Example:

```
css  
Copy code  
MOV B, 25H ; B = 25H  
MOV A, B ; A = B = 25H
```

- **LXI RP, DDDD:** Loads 16-bit immediate data (DDDD) into the specified register pair (RP). Example:

```
Copy code  
LXI H, 2050H ; HL = 2050H
```

- **MVI A, DD:** Loads immediate 8-bit data (DD) into register A. Example:

```
css  
Copy code  
MVI A, 35H ; A = 35H
```

- **LDA DDDD:** Loads the accumulator with data from memory location DDDD. Example:

```
css  
Copy code  
LDA 2000H ; A = [2000H]
```

- **STA DDDD:** Stores the accumulator content into memory location DDDD. Example:

```
css  
Copy code  
STA 3000H ; [3000H] = A
```

## 18. LHLD, SHLD, XCHG, SPHL Instructions with Examples

- **LHLD DDDD:** Loads HL register pair with data from memory locations DDDD and DDDD+1. Example:

```

css
Copy code
LHLD 2000H ; HL = [2001H][2000H]

```

- **SHLD DDDD:** Stores HL register pair content into memory locations DDDD and DDDD+1. Example:

```

css
Copy code
SHLD 3000H ; [3001H][3000H] = HL

```

- **XCHG:** Exchange the contents of HL register pair with DE register pair. Example:

```

mathematica
Copy code
LXI H, 2050H ; HL = 2050H
LXI D, 1020H ; DE = 1020H
XCHG          ; HL <-> DE (After XCHG, HL = 1020H, DE = 2050H)

```

- **SPHL:** Load stack pointer with HL register pair content. Example:

```

Copy code
LXI H, 3000H ; HL = 3000H
SPHL          ; SP = 3000H

```

## 19. ANA, ORA, XRA, CMA Instructions with Examples

- **ANA A, B:** Performs bitwise AND operation between accumulator A and register B. Example:

```

css
Copy code
MOV A, 25H ; A = 25H (0010 0101)
MOV B, 0AH ; B = 0AH (0000 1010)
ANA B      ; A = A & B = 0000 0000 (Result is 00H)

```

- **ORA A, B:** Performs bitwise OR operation between accumulator A and register B. Example:

```

css
Copy code
MOV A, 25H ; A = 25H (0010 0101)
MOV B, 0AH ; B = 0AH (0000 1010)
ORA B      ; A = A | B = 0010 1111 (Result is 2FH)

```

- **XRA A, B:** Performs bitwise XOR operation between accumulator A and register B. Example:

```

css
Copy code
MOV A, 25H ; A = 25H (0010 0101)

```

```

MOV B, 0AH      ; B = 0AH (0000 1010)
XRA B          ; A = A ^ B = 0010 1111 (Result is 2FH)

```

- **CMA:** Complement accumulator A (flip all bits). Example:

```

css
Copy code
MOV A, 25H      ; A = 25H (0010 0101)
CMA            ; A = NOT A = 1101 1010 (Result is DAh)

```

## 20. CMP, PUSH, POP Instructions with Examples

- **CMP A, B:** Compares accumulator A with register B (subtracts B from A but does not store the result). Example:

```

css
Copy code
MOV A, 30H      ; A = 30H
MOV B, 20H      ; B = 20H
CMP B          ; Compare A - B (Result is A - B = 30H - 20H =
10H, but result is not stored)

```

- **PUSH RP:** Pushes the content of the specified register pair (RP) onto the stack. Example:

```

arduino
Copy code
LXI SP, 3FFFH    ; Initialize stack pointer
LXI B, 1234H    ; BC = 1234H
PUSH B          ; Push BC onto the stack

```

- **POP RP:** Pops the top of the stack into the specified register pair (RP). Example:

```

arduino
Copy code
LXI SP, 3FFFH    ; Initialize stack pointer
POP B          ; Pop the top of the stack into BC

```

## 21. RLC, RAL, RRC, RAR Instructions

- **RLC (Rotate Left through Carry):** Rotates the bits in the accumulator left by one position. The carry flag is set from the most significant bit (MSB) of the accumulator and the MSB is shifted into the least significant bit (LSB).

Example:

```

yaml
Copy code
MOV A, 8AH      ; A = 1000 1010
RLC            ; A = 0001 0101 (Carry flag = 1)

```

After RLC, A becomes 15H (binary 0001 0101) and the Carry flag is set.

- **RAL (Rotate Left):** Rotates the bits in the accumulator left through the Carry flag. The Carry flag is shifted into the LSB, and the MSB is shifted into the Carry flag.

Example:

```
yaml
Copy code
MOV A, 8AH      ; A = 1000 1010
RAL             ; A = 0001 0101 (Carry flag = 1)
```

After RAL, A becomes 15H (binary 0001 0101) and the Carry flag holds the previous MSB.

- **RRC (Rotate Right through Carry):** Rotates the bits in the accumulator right by one position. The LSB is shifted into the MSB and the Carry flag, and the Carry flag is set from the LSB.

Example:

```
yaml
Copy code
MOV A, 8AH      ; A = 1000 1010
RRC             ; A = 1100 0101 (Carry flag = 0)
```

After RRC, A becomes C5H (binary 1100 0101) and the Carry flag is 0.

- **RAR (Rotate Right):** Rotates the bits in the accumulator right through the Carry flag. The Carry flag is shifted into the MSB, and the LSB is shifted into the Carry flag.

Example:

```
yaml
Copy code
MOV A, 8AH      ; A = 1000 1010
RAR             ; A = 1100 0101 (Carry flag = 0)
```

After RAR, A becomes C5H (binary 1100 0101) and the Carry flag holds the previous LSB.

## 22. Conditional and Unconditional Jump Instructions

- **Unconditional Jump (JMP):** Transfers control to the specified memory address unconditionally.

Example:

```
css
Copy code
JMP 3000H      ; Jump to the memory address 3000H
```

- **Conditional Jump:** Transfers control to the specified memory address based on the condition flags (Zero, Carry, Sign, etc.).

Examples:

```
vbnnet
Copy code
JC 3000H ; Jump to 3000H if Carry flag is set
JZ 3000H ; Jump to 3000H if Zero flag is set
JNZ 3000H ; Jump to 3000H if Zero flag is not set (Zero
flag is clear)
```

Conditional jumps provide flexibility in program flow based on the state of the flags.

### 23. Difference between SIM and RIM Instruction

- **SIM (Set Interrupt Mask):** The SIM instruction is used to mask (disable) or unmask (enable) interrupts in the 8085 microprocessor. It can mask either maskable interrupts (RIM and SIM) or non-maskable interrupts (RIM only).
- **RIM (Read Interrupt Mask):** The RIM instruction reads the status of interrupt-related signals and the state of interrupt enable flip-flops. It reads the status of maskable interrupts and the state of the interrupt enable flip-flops.

### 24. Difference between DI and EI Instruction

- **DI (Disable Interrupts):** The DI instruction disables maskable interrupts (including RST 7.5, RST 6.5, and RST 5.5) by clearing the interrupt enable flip-flop. The non-maskable interrupt (TRAP) is still recognized.
- **EI (Enable Interrupts):** The EI instruction enables maskable interrupts by setting the interrupt enable flip-flop. After EI, maskable interrupts (RST 7.5, RST 6.5, RST 5.5) can be recognized and serviced.

### 25. Memory Interfacing in 8085 Microprocessor

Memory interfacing in the 8085 microprocessor involves connecting external memory devices (such as RAM, ROM, and I/O devices) to the microprocessor. It allows the microprocessor to read from and write to these external memory locations. Key concepts include:

- **Address Bus:** Lines used by the microprocessor to specify the address of the memory location it wants to access.
- **Data Bus:** Lines used for bidirectional data transfer between the microprocessor and external memory or peripherals.
- **Control Signals:** Signals like RD (Read), WR (Write), and IO/M (Input/Output or Memory) control the direction of data flow and operation mode (memory or I/O).
- **Examples:**

- **Reading from Memory:**

```
csharp
Copy code
MOV A, M ; Load accumulator A with data from
memory location pointed by HL
```

- **Writing to Memory:**

```
csharp
Copy code
```

```
MOV M, A      ; Store contents of accumulator A into  
memory location pointed by HL
```

- **I/O Operations:**

```
css  
Copy code  
IN 00H      ; Read data from input port 00H  
OUT 01H     ; Output data to output port 01H
```

Memory interfacing enables the 8085 microprocessor to interact with external devices and store/retrieve data, expanding its capabilities beyond its internal registers and immediate data storage.

These explanations cover the essential aspects of the specified topics for the Intel 8085 microprocessor. If you have more questions or need further clarification, feel free to ask!

You

26. Find out the outputs and number of T-states in the following programs: a) SUB A MOV B, A DCR B INR B b) MVI A, C5H ORI A7H RLC RRC c) MVI C, FFH LOOP: DCR C JNZ LOOP d) LHLD 2050H XCHG e) LXI H, 3050 MOV A, M CMA ADI 05 STA 3060 Assume input is FFH

ChatGPT

ChatGPT

Let's analyze each program segment to determine the outputs and count the number of T-states (timing states) for the Intel 8085 microprocessor:

## Program Analysis

### a) SUB A

```
assembly  
Copy code  
SUB A      ; Subtract accumulator A from itself (A - A)  
MOV B, A   ; Move contents of A to B (B = A)  
DCR B      ; Decrement B (B = A - 1)  
INR B      ; Increment B (B = A)
```

- **Outputs:**

- After SUB A: Accumulator A remains unchanged (A = FFH, assuming input is FFH).
- After MOV B, A: B = A = FFH.
- After DCR B: B = FFH - 1 = FEH.
- After INR B: B = FEH + 1 = FFH.

- **Number of T-states:**

- SUB A: 4 T-states
- MOV B, A: 5 T-states
- DCR B: 5 T-states
- INR B: 5 T-states
- **Total:** 19 T-states

## b) MVI A, C5H

```
assembly
Copy code
MVI A, C5H ; Load immediate data C5H into A
ORI A7H      ; Logical OR immediate data A7H with A
RLC          ; Rotate left through Carry
RRC          ; Rotate right through Carry
```

- **Outputs:**

- After MVI A, C5H: A = C5H.
- After ORI A7H: A = C5H | A7H = E7H.
- After RLC: A = C5H rotated left (bitwise).
- After RRC: A = E7H rotated right (bitwise).

- **Number of T-states:**

- MVI A, C5H: 7 T-states
- ORI A7H: 7 T-states
- RLC: 4 T-states
- RRC: 4 T-states
- **Total:** 22 T-states

## c) MVI C, FFH

```
assembly
Copy code
MVI C, FFH ; Load immediate data FFH into C
LOOP:
DCR C       ; Decrement C
JNZ LOOP    ; Jump to LOOP if C is not zero
```

- **Outputs:**

- MVI C, FFH: C = FFH.
- Loop executes from FFH to 00H.

- **Number of T-states:**

- MVI C, FFH: 7 T-states
- DCR C (inside loop, 5 T-states per iteration until C becomes 00H)
- JNZ LOOP (5 T-states if not taken, 11 T-states if taken)
- **Total:** Depends on the number of iterations; typically, multiple DCR and JNZ cycles until C becomes zero.

## d) LHLD 2050H

```
assembly
Copy code
LHLD 2050H ; Load HL pair with data from memory location 2050H
and 2051H
XCHG        ; Exchange contents of HL with DE register pair
```

- **Outputs:**

- LHLD 2050H: HL = Data at 2051H2050H.
- XCHG: HL <-> DE.

- **Number of T-states:**

- LHLD 2050H: 16 T-states
- XCHG: 4 T-states
- **Total:** 20 T-states

### e) LXI H, 3050H

```

assembly
Copy code
LXI H, 3050H ; Load HL pair with immediate data 3050H
MOV A, M      ; Move data from memory location HL to A
CMA          ; Complement accumulator A
ADI 05       ; Add immediate data 05H to A
STA 3060H    ; Store accumulator A at memory location 3060H

```

- **Assumption:** Assume input is FFH.
- **Outputs:**
  - LXI H, 3050H: HL = 3050H.
  - MOV A, M: A = Data at 3050H (assuming input is FFH).
  - CMA: A = NOT A = 00H.
  - ADI 05: A = 00H + 05H = 05H.
  - STA 3060H: Data at 3060H = 05H.
- **Number of T-states:**
  - LXI H, 3050H: 10 T-states
  - MOV A, M: 7 T-states
  - CMA: 4 T-states
  - ADI 05: 7 T-states
  - STA 3060H: 13 T-states
  - **Total:** 41 T-states

These analyses provide a breakdown of outputs and timing states for each program segment executed on the Intel 8085 microprocessor. Each instruction and operation incurs a specific number of T-states, which are crucial for understanding the timing and performance characteristics of the processor.

Let's address each of the questions step by step:

## 27. Addressing Range with EPROM and RAM interfaced with 8085

Given:

- Total memory space available: 64KB (64 \* 1024 bytes)
- Equally divided between EPROM and RAM

### Calculation:

- EPROM and RAM each get 32KB (32 \* 1024 bytes)

### Addressing Range:

- **EPROM:** Addresses range from 0000H to 7FFFH (0 to 32767 in decimal).
- **RAM:** Addresses range from 8000H to FFFFH (32768 to 65535 in decimal).

## **28. Interrupts in 8085 Microprocessor**

**Interrupts** in the 8085 microprocessor are mechanisms that temporarily halt the current program execution to handle urgent tasks or events. There are several types of interrupts:

- **Hardware Interrupts:**
  - **Maskable Interrupts:** Can be enabled or disabled by software using instructions like EI (Enable Interrupts) and DI (Disable Interrupts). Examples include RST 7.5, RST 6.5, and RST 5.5.
  - **Non-Maskable Interrupt (NMI):** Cannot be disabled by the EI instruction. It is used for critical tasks that require immediate attention, such as power failure detection (TRAP).
- **Software Interrupts:**
  - Initiated by software instructions like RST (Restart) or CALL.

## **29. Vectored Addresses of Interrupts in 8085 Microprocessor**

The vectored addresses for maskable interrupts (RST 7.5, RST 6.5, RST 5.5) are as follows:

- **RST 7.5:** Vectored to address 003C (hexadecimal).
- **RST 6.5:** Vectored to address 0034 (hexadecimal).
- **RST 5.5:** Vectored to address 002C (hexadecimal).

These addresses are predefined and hardcoded in the 8085 microprocessor.

## **30. Memory Mapped IO vs IO Mapped IO**

- **Memory Mapped IO:** In memory mapped IO, devices (like peripherals) are assigned addresses within the same address space as memory. The CPU communicates with these devices using memory read and write instructions.
- **IO Mapped IO:** In IO mapped IO, separate IO instructions are used to communicate with devices. These instructions are distinct from memory operations and allow the CPU to distinguish between accessing memory and accessing IO devices.

## **31. 8255A Peripheral Interface with Intel 8085**

The 8255A is a popular programmable peripheral interface chip that can be interfaced with the Intel 8085 microprocessor. It provides versatile parallel I/O capabilities, allowing the microprocessor to control external devices.

Key features:

- 24 I/O lines grouped into three 8-bit ports (Port A, Port B, Port C).
- Mode selection for each port (Input, Output, or Bidirectional).
- Can be configured to operate in different modes such as simple I/O, strobed I/O, and handshake I/O.

## **32. Operation of DMA Controller**

- **DMA (Direct Memory Access)** Controller is a special device used to transfer data between external devices (such as disk drives, network cards) and memory without involving the CPU.

- The DMA controller operates by temporarily halting the CPU, gaining control of the system bus, and transferring data directly between memory and the external device.
- It significantly improves system performance by offloading data transfer tasks from the CPU.

### **33. Assembly Language Program to ADD Two 8-bit Numbers using 8085**

```

assembly
Copy code
; Program to add two 8-bit numbers
ORG 0000H ; Start address

MVI A, 23H ; Load first number into accumulator A (Example: 23H)
MVI B, 0AH ; Load second number into register B (Example: 0AH)

ADD B ; Add contents of B to A
MOV C, A ; Move result from A to C (C = A)

HLT ; Halt the program

END ; End of program

```

### **34. Assembly Language Program to SUBTRACT Two 8-bit Numbers using 8085**

```

assembly
Copy code
; Program to subtract two 8-bit numbers
ORG 0000H ; Start address

MVI A, 30H ; Load first number into accumulator A (Example: 30H)
MVI B, 0AH ; Load second number into register B (Example: 0AH)

SUB B ; Subtract contents of B from A
MOV C, A ; Move result from A to C (C = A)

HLT ; Halt the program

END ; End of program

```

### **35. Assembly Language Program to ADD Array of 10 Numbers using 8085**

```

assembly
Copy code
; Program to add array of 10 numbers
ORG 0000H ; Start address

; Initialize data
MVI B, 0AH ; B = 10 (number of elements)
LXI H, DATA ; Load HL with address of data array

; Loop to add numbers
LOOP:
MOV A, M ; Load current number from memory to A

```

```

ADD B      ; Add B to A
DCR B      ; Decrement B (number of elements)
JNZ LOOP   ; Jump to LOOP if B != 0

HLT       ; Halt the program

; Data section
DATA: DB 01H, 02H, 03H, 04H, 05H, 06H, 07H, 08H, 09H, 0AH ;
Example data array

END       ; End of program

```

### **36. Assembly Language Program to ADD Two 16-bit Numbers using 8085**

```

assembly
Copy code
; Program to add two 16-bit numbers
ORG 0000H ; Start address

LXI H, 3050H ; Load HL with first 16-bit number (Example: 3050H)
LXI D, 2040H ; Load DE with second 16-bit number (Example: 2040H)

MOV A, H      ; Move high byte of HL to accumulator A
ADD D        ; Add high byte of DE to A
MOV H, A      ; Move result to high byte of HL

MOV A, L      ; Move low byte of HL to accumulator A
ADC E        ; Add low byte of DE to A with carry
MOV L, A      ; Move result to low byte of HL

HLT       ; Halt the program

END       ; End of program

```

### **37. Assembly Language Program to Find Larger Between Two 8-bit Numbers using 8085**

```

assembly
Copy code
; Program to find larger between two 8-bit numbers
ORG 0000H ; Start address

MVI A, 40H ; Load first number into accumulator A (Example: 40H)
MVI B, 30H ; Load second number into register B (Example: 30H)

CMP B      ; Compare A and B
JNC SKIP   ; Jump to SKIP if no carry (A >= B)
MOV C, B    ; Move B to C (C = B)
JMP END    ; Jump to END

SKIP:
MOV C, A    ; Move A to C (C = A)

END:

```

```

HLT      ; Halt the program
END      ; End of program

```

### **38. Assembly Language Program to Find Smaller Between Two 8-bit Numbers using 8085**

```

assembly
Copy code
; Program to find smaller between two 8-bit numbers
ORG 0000H ; Start address

MVI A, 40H ; Load first number into accumulator A (Example: 40H)
MVI B, 30H ; Load second number into register B (Example: 30H)

CMP B      ; Compare A and B
JC SKIP    ; Jump to SKIP if carry (A < B)
MOV C, B   ; Move B to C (C = B)
JMP END    ; Jump to END

SKIP:
MOV C, A   ; Move A to C (C = A)

END:
HLT      ; Halt the program
END      ; End of program

```

### **39. Assembly Language Program to Find Largest Number from an Array of 10 Numbers using 8085**

```

assembly
Copy code
; Program to find largest number from an array of 10 numbers
ORG 0000H ; Start address

; Initialize data
MVI B, 0AH ; B = 10 (number of elements)
LXI H, DATA ; Load HL with address of data array

; Initialize max as first element of array
MOV A, M    ; Load first number from memory to A
MOV C, A    ; Move A to C (C = A)

; Loop to find largest number
LOOP:
MOV A, M    ; Load current number from memory to A
CMP C      ; Compare A and C
JNC SKIP    ; Jump to SKIP if no carry (A >= C)
MOV C, A    ; Move A to C (C = A)

SKIP:
DCR B      ; Decrement B (number of elements)
JNZ LOOP    ; Jump to LOOP if B != 0

```

```

HLT      ; Halt the program

; Data section
DATA: DB 01H, 02H, 03H, 04H, 05H, 06H, 07H, 08H, 09H, 0AH ;
Example data array

END      ; End of program

```

#### **40. Assembly Language Program to Find Smallest Number from an Array of 10 Numbers using 8085**

```

assembly
Copy code
; Program to find smallest number from an array of 10 numbers
ORG 0000H ; Start address

; Initialize data
MVI B, 0AH ; B = 10 (number of elements)
LXI H, DATA ; Load HL with address of data array

; Initialize min as first element of array
MOV A, M ; Load first number from memory to A
MOV C, A ; Move A to C (C = A)

; Loop to find smallest number
LOOP:
MOV A, M ; Load current number from memory to A
CMP C ; Compare A and C
JC SKIP ; Jump to SKIP if carry (A < C)
MOV C, A ; Move A to C (C = A)

SKIP:
DCR B ; Decrement B (number of elements)
JNZ LOOP ; Jump to LOOP if B != 0

HLT      ; Halt the program

; Data section
DATA: DB 01H, 02H, 03H, 04H, 05H, 06H, 07H, 08H, 09H, 0AH ;
Example data array

END      ; End of program

```

#### **41. Assembly Language Program to Find 1's & 2's Complement of an 8-bit Number using 8085**

```

assembly
Copy code
; Program to find 1's & 2's complement of an 8-bit number
ORG 0000H ; Start address

MVI A, 53H ; Load number into accumulator A (Example: 53H)

```

```

; 1's complement
CMA           ; A = ~A (1's complement)

; 2's complement
ADI 01H       ; Add 1 to A (2's complement)

HLT           ; Halt the program

END           ; End of program

```

## 42. Assembly Language Program to Check Whether an 8-bit Number is Even or Odd using 8085

```

assembly
Copy code
; Program to check whether an 8-bit number is even or odd
ORG 0000H     ; Start address

MVI A, 56H    ; Load number into accumulator A (Example: 56H)

ANI 01H       ; Mask least significant bit (A & 0000 0001)
JZ EVEN        ; Jump to EVEN if result is zero (even)
JMP ODD        ; Jump to ODD (odd)

EVEN:
MVI B, 00H    ; Even number indicator (Example: 00H)
JMP END        ; Jump to END

ODD:
MVI B, 01H    ; Odd number indicator (Example: 01H)

END:
HLT           ; Halt the program

END           ; End of program

```

These assembly language programs demonstrate various operations and tasks that can be performed using the Intel 8085 microprocessor. Each program is structured to achieve a specific computational or control task efficiently within the constraints of the 8085 architecture.

You

Let's delve into each topic one by one:

## 43. Pipeline Architecture of 8086

The Intel 8086 microprocessor employs a pipelined architecture to improve instruction execution throughput. The pipeline stages in the 8086 are as follows:

1. **Instruction Prefetch:** Fetches the next instruction from memory.
2. **Instruction Decode:** Decodes the fetched instruction and determines the required operands.

3. **Execution:** Executes the instruction.
4. **Memory Access:** If the instruction involves memory access, data is read from or written to memory.
5. **Write-back:** The result of the execution is written back to the appropriate register or memory location.

#### 44. Operations of BIU and EU Sections in 8086 Microprocessor

- **Bus Interface Unit (BIU):**
  - Responsible for interfacing with the system bus.
  - Manages fetching instructions and operands from memory into the instruction queue.
  - Handles memory and I/O address generation.
  - Supports instruction prefetching to improve performance.
- **Execution Unit (EU):**
  - Executes instructions fetched by the BIU.
  - Includes the Arithmetic Logic Unit (ALU) which performs arithmetic and logical operations.
  - Handles data movement between registers, memory, and I/O devices.
  - Controls the flow of execution based on the decoded instructions.

#### 45. Difference between 8085 and 8086 Microprocessor

- **Architecture:**
  - **8085:** 8-bit microprocessor with a simpler instruction set and architecture suitable for small-scale applications.
  - **8086:** 16-bit microprocessor with a more advanced instruction set and architecture capable of handling larger data and more complex operations.
- **Memory Addressing:**
  - **8085:** 64 KB address space.
  - **8086:** 1 MB (20-bit) address space, segmented memory model.
- **Registers:**
  - **8085:** Accumulator (8-bit), 6 general-purpose registers (8-bit).
  - **8086:** 16-bit data and address registers, including AX, BX, CX, DX, SI, DI, BP, SP.
- **Instruction Set:**
  - **8085:** Smaller and simpler instruction set.
  - **8086:** Richer instruction set with more addressing modes and operations.

#### 46. Pin Diagram of 8086 Microprocessor

The pin diagram of the Intel 8086 microprocessor typically includes the following important pins:

- **Address Bus (A0-A19):** Outputs the address of the memory location to be accessed.
- **Data Bus (D0-D15):** Bi-directional bus for data transfer between the microprocessor, memory, and peripherals.
- **Control and Status Signals:** Includes signals like RD (Read), WR (Write), DT/R (Data Transmit/Receive), etc., for controlling data and status information.
- **Power and Clock Pins:** Vcc, GND, CLK (Clock), RESET, etc., for power supply and clock input.

#### 47. Different Modes of Operation in 8086 Microprocessor

The 8086 microprocessor operates in the following modes:

- **Real Mode:** Compatible with 8088, uses 20-bit segmented memory addressing (1 MB).
- **Protected Mode:** Provides multitasking, memory protection, and extended memory addressing (up to 1 GB).
- **Virtual 8086 Mode:** Allows multiple virtual 8086 environments within a protected mode operating system.
- **System Management Mode (SMM):** Power-saving mode for managing system events.

## 48. Flag Register in 8086 Microprocessor

The **Flag Register** in the 8086 microprocessor is a 16-bit register that contains status flags set or cleared based on the result of arithmetic and logical operations. The most commonly used flags include:

- **Carry (CF):** Set if an arithmetic operation generates a carry-out or a borrow for subtraction.
- **Zero (ZF):** Set if the result of an operation is zero.
- **Sign (SF):** Set if the result of a signed operation is negative (MSB is 1).
- **Overflow (OF):** Set if the signed result of an operation exceeds the capacity of a register.

**Example:**

```
assembly
Copy code
MOV AX, 0FFFH ; Load AX with a value
ADD AX, 1      ; Add 1 to AX
```

- \*\* Meaning

You

Let's cover each topic one by one:

## 49. Functions of Internal Registers of 8086 Microprocessor

The 8086 microprocessor has several internal registers that serve different functions:

- **General Purpose Registers:**
  - **AX, BX, CX, DX:** Used for general arithmetic and data manipulation.
  - **SI, DI:** Source and destination index registers for string operations.
  - **BP, SP:** Base pointer and stack pointer for addressing memory and managing the stack.
- **Segment Registers:**
  - **CS:** Code Segment register holds the segment address of the current program.
  - **DS:** Data Segment register points to the segment that contains data.
  - **SS:** Stack Segment register points to the segment holding the stack.
  - **ES:** Extra Segment register can be used for additional data segments.
- **Pointer and Index Registers:**
  - **IP:** Instruction Pointer holds the offset address within the current code segment.
  - **SP:** Stack Pointer points to the top of the stack.

- **BP:** Base Pointer is used for referencing parameters and local variables in procedures.
- **SI, DI:** Source and Destination Index registers are used for data movement.
- **Flag Register:**
  - Holds status flags such as Carry (CF), Zero (ZF), Sign (SF), Overflow (OF), etc., that reflect the result of arithmetic and logic operations.

## 50. Different Addressing Modes of Intel 8086 Microprocessor

The Intel 8086 microprocessor supports various addressing modes to provide flexibility in accessing operands and data. These addressing modes include:

- **Immediate Addressing:** Operand is specified directly in the instruction (e.g., MOV AX, 1234H).
- **Register Addressing:** Operand is in one of the CPU registers (e.g., MOV AX, BX).
- **Direct Addressing:** Operand is located at a specific memory address (e.g., MOV AX, [1234H]).
- **Register Indirect Addressing:** Operand is at the memory location specified by a register (e.g., MOV AX, [BX]).
- **Base Register Addressing:** Operand is at the memory location specified by a base register plus a displacement (e.g., MOV AX, [BX+SI]).
- **Indexed Addressing:** Operand is at the memory location specified by a base register plus an index register (e.g., MOV AX, [BX+DI]).
- **Based Indexed Addressing:** Operand is at the memory location specified by a base register plus an index register and a displacement (e.g., MOV AX, [BX+SI+1234H]).

## 51. Basic Features of 8051 Microcontroller

The 8051 microcontroller is a popular 8-bit microcontroller known for its simplicity and versatility. Some basic features include:

- **8-bit CPU:** Central Processing Unit with an 8-bit data bus.
- **On-chip RAM and ROM:** Typically includes 128 bytes of RAM and 4 KB of ROM (in some variants).
- **Peripheral Interface:** Includes I/O ports, timers/counters, serial communication ports (UART), and interrupt control.
- **Interrupt System:** Supports both external and internal interrupts for responding to events.
- **Clock Circuit:** Requires an external crystal oscillator for providing clock pulses.
- **Power Management:** Includes power-saving modes to conserve energy.

## 52. Difference between Microprocessor and Microcontroller

- **Microprocessor:**
  - Generally refers to a single-chip CPU used in computers.
  - Requires external components like memory, I/O devices, and peripherals for a complete system.
  - Designed for general-purpose computing and executing instructions from memory.
- **Microcontroller:**
  - Integrated circuit comprising a CPU, memory (RAM/ROM), I/O ports, timers, and other peripherals on a single chip.
  - Designed for specific tasks and embedded applications.

- Often used in devices requiring real-time control and monitoring, such as embedded systems.

### 53. Special Function Registers and Internal RAM Architecture of 8051

- **Special Function Registers (SFRs):**
  - These registers in the 8051 microcontroller control various on-chip peripherals and functionalities.
  - Examples include registers for GPIO (General Purpose I/O), timers/counters, UART (Universal Asynchronous Receiver/Transmitter), etc.
- **Internal RAM Architecture:**
  - The 8051 microcontroller typically has 128 bytes of on-chip RAM.
  - This RAM is used for storing data and variables during program execution.
  - It is directly accessible by the CPU and can be used for both data and stack operations.

### 54. Memory Organization of 8051 Microcontroller

The memory organization of the 8051 microcontroller is divided into several areas:

- **Program Memory (ROM):**
  - Holds the program code that the microcontroller executes.
  - Typically 4 KB (in standard variants) but can vary depending on the specific model.
- **Data Memory (RAM):**
  - Divided into Internal RAM (128 bytes) and External RAM (if interfaced).
  - Used for storing data, variables, and stack operations during program execution.
- **Special Function Registers (SFRs):**
  - Located in the memory-mapped I/O space.
  - Control and configure various on-chip peripherals and functionalities.

### 55. Different Addressing Modes of 8051 Microcontroller

The 8051 microcontroller supports several addressing modes similar to those found in the Intel 8085 and 8086 processors:

- **Immediate Addressing:** Operand is specified directly in the instruction.
- **Direct Addressing:** Operand is directly accessed from a specific memory location.
- **Indirect Addressing:** Operand is accessed indirectly through a register or pointer.
- **Register Addressing:** Operand is in one of the CPU registers.
- **Indexed Addressing:** Operand is accessed using an index or pointer to an array element.

### 56. Functions of Internal Registers of 8051 Microcontroller

The 8051 microcontroller has several internal registers that serve specific functions:

- **Accumulator (A):** Used for arithmetic and logic operations.
- **B Register:** Auxiliary register for some arithmetic and logic operations.
- **Data Pointer (DPTR):** Pointer used for accessing external memory (if present).
- **Program Status Word (PSW):** Holds status flags like Carry, Overflow, and auxiliary Carry.
- **Stack Pointer (SP):** Points to the top of the stack used for subroutine calls and interrupts.
- **Timer/Counter Registers:** Used for timing and counting operations.

- **Port Registers (P0 to P3):** Used for interfacing with external peripherals and I/O operations.

## 57. Flag Register in 8051 Microcontroller

The **Flag Register** in the 8051 microcontroller contains various status flags that reflect the outcome of arithmetic and logical operations. These flags include:

- **Carry Flag (C):** Set if the last arithmetic operation resulted in a carry out.
- **Auxiliary Carry Flag (AC):** Set if a carry occurred from bit 3 to bit 4 during arithmetic operations.
- **Overflow Flag (OV):** Set if the result of a signed operation exceeded the capacity of the register.
- **Register Bank Select (RS1, RS0):** Selects the register bank for accessing the general-purpose registers.

### Example:

```
assembly
Copy code
MOV A, #0FFH      ; Load accumulator with value 0FFH
ADD A, #01H       ; Add 1 to accumulator
JNC NO_CARRY     ; Jump if no carry flag set

; Carry flag is set here if addition caused carry

NO_CARRY:
...
```

## 58. Functions of PSEN and EA Pins in 8051 Microcontroller

- **PSEN (Program Store Enable):**
  - PSEN is an output pin used to enable external ROM (Program Memory) during fetch cycles.
  - When PSEN is active (low), the microcontroller fetches instructions from external ROM.
- **EA (External Access):**
  - EA is an input pin that controls whether the microcontroller fetches code from external ROM or internal ROM.
  - When EA is high, the microcontroller fetches code from external ROM (if present).
  - When EA is low, the microcontroller fetches code from internal ROM.

These pins are crucial for determining where the microcontroller fetches its program instructions from, enabling flexibility in system design and integration of external memory devices.