Lecture-04

৮০৮৫ আর্কিটেকচারের ইনস্ট্রাকশন সাইকেলের ধারা নিম্নরূপ:

১. প্রোগ্রাম কাউন্টার সূচনা: প্রোগ্রাম কাউন্টার প্রোগ্রাম সম্পাদনার জন্য পরবর্তী ঠিকানায় নির্দেশটি সংগ্রহ করার মাধ্যমে প্রক্রিয়া শুরু করে।

২. ঠিকানা/ডেটা বাস মাল্টিপ্লেক্সিং: মাল্টিপ্লেক্সড ঠিকানা/ডেটা বাস প্রথমে ঠিকানা বাস হিসেবে কাজ করে মেমরি থেকে নির্দেশ সংগ্রহ করে। নির্দেশ সংগ্রহ করার পরে এটি ডেটা বাস হিসেবে কাজ করে, নির্দিষ্ট মেমরি অবস্থান থেকে ডেটা এনে ৮-বিট অভ্যন্তরীণ বাসে পাঠায়। ঠিকানা এবং ডেটা বাসের মধ্যে এই পরিবর্তনকে নিয়ন্ত্রণ করতে Address Latch Enable (ALE) পিন ব্যবহার করা হয়; যদি ALE = ১ হয়, তবে এটি ঠিকানা বাস হিসেবে কাজ করে, অন্যথায় এটি ডেটা বাস হিসেবে কাজ করে।

৩. ইনস্ট্রাকশন সঞ্চয় ও ডিকোডিং: সংগৃহীত ডেটাটি Instruction Register-এ সাময়িকভাবে সংরক্ষিত হয় এবং Instruction Decoder-এ পাঠানো হয় যাতে এটি ডিকোডের জন্য প্রস্তুত থাকে।

৪. টাইমিং এবং কন্ট্রোল সিগনাল অপারেশন: টাইমিং এবং কন্ট্রোল সার্কিটগুলো সক্রিয় হয়, এবং মাইক্রোপ্রসেসরের ভেতরে সংকেত পাঠায় যা নির্দেশ করে যে নির্দেশটি READ/WRITE অপারেশনের জন্য কি না, বা MEMORY/I-O অ্যাক্সেসের জন্য কি না।

৫. অপারেশনের কার্যকরীকরণ: এই সংকেত অনুযায়ী মাইক্রোপ্রসেসর লজিকাল ও অ্যারিথমেটিক অপারেশনগুলো সম্পাদন করে, যেখানে Arithmetic Logic Unit (ALU) ব্যবহার করা হয়। প্রয়োজনীয় ডেটা বিভিন্ন রেজিস্টার থেকে সংগ্রহ করা হয় এবং ফলাফলের ভিত্তিতে Flag Register স্বয়ংক্রিয়ভাবে পরিবর্তিত হয়।

৬. সিরিয়াল **I/O** অপারেশন: Serial I/O ডেটা পিন (SID অথবা SOD) ব্যবহার করে বাইরের ডিভাইসের সাথে ইনপুট বা আউটপুটের মাধ্যমে ডেটা বিনিময় করা যায়।

৭. ইন্টারাপ্ট হ্যান্ডলিং: প্রক্রিয়াকালে কোনো ইন্টারাপ্ট শনাক্ত হলে বর্তমান প্রক্রিয়া সাময়িকভাবে থেমে যায়, এবং Interrupt Service Routine (ISR) সক্রিয় হয়। ইন্টারাপ্টের কাজ শেষ হলে স্বাভাবিক কার্যক্রম পুনরায় শুরু হয়।

****

The flow of an instruction cycle in the 8085 architecture is as follows:

1. **Program Counter Initialization**: The Program Counter begins the process by pointing to the next address for program execution, fetching an instruction from the specified memory location.
2. **Address/Data Bus Multiplexing**: The multiplexed address/data bus first acts as an address bus to fetch the instruction from memory. Once the instruction is fetched, it switches to a data bus role to retrieve data from the specified memory location, which it then sends through an 8-bit internal bus. The Address Latch Enable (ALE) pin controls this switching: if ALE = 1, the multiplexed bus is an address bus; otherwise, it functions as a data bus.
3. **Instruction Storage and Decoding**: The fetched data moves to the Instruction Register for temporary storage. It is then passed to the Instruction Decoder, which prepares it for decoding.
4. **Timing and Control Signal Operation**: The timing and control circuits activate, sending signals throughout the microprocessor to indicate whether the instruction requires a READ/WRITE operation or MEMORY/I-O access.
5. **Execution of Operations**: Based on these signals, the microprocessor performs logical and arithmetic operations using the Arithmetic Logic Unit (ALU). The required data is fetched from various registers, and the Flag register updates dynamically to reflect the results.
6. **Serial I/O Operations**: Using the Serial I/O data pins (SID or SOD), data can be exchanged with external devices, facilitating input/output operations.
7. **Interrupt Handling**: If an interrupt is detected during execution, the current process is paused, and the microprocessor invokes the Interrupt Service Routine (ISR) to address the interrupt. After handling the interrupt, normal execution resumes

Time Delay in the context of microprocessors like the 8085 refers to the amount of time an instruction takes to execute. It's measured in units called **T-States**, which represent the smallest time period for one clock cycle of the processor.

Key Points to Understand Time Delay:

- **T-State**: A single clock cycle in the microprocessor. Each T-State represents a fixed, short period of time during which a small part of an instruction is processed.
- **Instruction Delay**: The total time taken to complete an instruction, measured by the number of T-States required for that instruction.
- **Purpose**: Knowing the time delay of instructions helps in designing time-sensitive applications, like embedded systems, where precise timing is critical.

For example:

- **MVI C, FFH** instruction requires 7 T-States, which means the processor takes 7 clock cycles to complete this instruction.
- A more complex instruction like **CALL addr** takes 18 T-States due to additional operations, like fetching and storing the address.

In simple terms, the time delay tells us how long the processor takes to complete a specific instruction, which is crucial for accurate timing control in many applications.

টাইম ডিলে মাইক্রোপ্রসেসরের ক্ষেত্রে একটি নির্দেশনা সম্পন্ন করতে যে সময় লাগে, সেটাকে বোঝায়। এটি **T-**স্টেট এককে পরিমাপ করা হয়, যা প্রসেসরের একটি ক্লক সাইকেলের সবচেয়ে ক্ষুদ্র সময়কাল।

টাইম ডিলের গুরুত্বপূর্ণ দিকসমূহ**:**

- **T-**স্টেট: মাইক্রোপ্রসেসরের একটি ক্লক সাইকেল বা সময়ের ক্ষুদ্রতম একক। প্রতিটি T-স্টেট নির্দেশনাটি প্রক্রিয়াকরণের একটি ছোট অংশের জন্য সময় নির্ধারণ করে।
- ইনস্ট্রাকশন ডিলে: একটি নির্দেশ সম্পন্ন করতে যে মোট সময় লাগে, সেটি T-স্টেটের সংখ্যার মাধ্যমে পরিমাপ করা হয়।

- প্রয়োজনীয়তা: ইনস্ট্রাকশনগুলোর টাইম ডিলে জানার ফলে নির্ভুল টাইমিং প্রয়োজন এমন অ্যাপ্লিকেশন ডিজাইনে সহায়তা করে, যেমন এম্বেডেড সিস্টেম যেখানে সঠিক সময় খুব গুরুত্বপূর্ণ।

উদাহরণস্বরূপ:

- **MVI C, FFH** নির্দেশনার জন্য ৭টি T-স্টেট প্রয়োজন, অর্থাৎ এই নির্দেশটি সম্পন্ন করতে প্রসেসরের ৭টি ক্লক সাইকেল লাগে।
- **CALL addr** এর মত জটিল নির্দেশের জন্য ১৮টি T-স্টেট প্রয়োজন, কারণ এতে ঠিকানা ফেচ এবং সংরক্ষণের মতো অতিরিক্ত প্রক্রিয়া অন্তর্ভুক্ত থাকে।

সহজভাবে, টাইম ডিলে বলতে বোঝায় প্রসেসরকে একটি নির্দিষ্ট নির্দেশ সম্পন্ন করতে কত সময় লাগে, যা সময়নির্ভর নিয়ন্ত্রণে গুরুত্বপূর্ণ।

Instruction Delay/Performance

Each instruction has a specific time it takes, measured in T-States:

- **MVI C, FFH**: 7 T-States
- **LOOP:**
  - **DCR C**: 4 T-States
  - **JNZ LOOP**: 7 or 10 T-States (depends on the condition)

T-States for Other Instructions:

- **ADD R**: 4 T-States
- **ADD M**: 7 T-States
- **CALL addr**: 18 T-States

Signal Types (F, S, R, W):

- **F = Fetch with 4 T-States**: Fetch operation using 4 T-States
- **S = Fetch with 6 T-States**: Fetch operation using 6 T-States
- **R = Memory Read**: Reading data from memory
- **W = Memory Write**: Writing data to memory

Here, each "T-State" is a small unit of time for processing instructions.

A Time Delay Loop is a simple programming technique used to create a specific delay or pause in a program. In microprocessors, it's often used to control the timing of actions, like blinking an LED or waiting before taking the next step.

How it Works:

- A loop repeats a set of instructions multiple times.
- The loop doesn't perform any task other than taking up time.
- Each instruction inside the loop takes a small amount of time, so by adjusting the number of repetitions, you can control the length of the delay.

Example:

In an 8085 microprocessor, a simple delay loop might look like this:

```
MVI C, FFH    ; Load the register C with a large value

LOOP: DCR C   ; Decrement C by 1

JNZ LOOP      ; Jump back to LOOP if C is not zero
```

Stack Pointer (SP) & Stack Memory

- **Stack Memory**: This is a specific area in memory used to store data temporarily during program execution.
- **LIFO Structure**: The stack works as a "Last In, First Out" system. The last item you put in is the first one to come out.

How the Stack Grows:

- The stack grows backwards in memory, moving from higher addresses to lower ones.
- **Bottom of the Stack**: The programmer defines where the stack starts (called the bottom). The stack then grows into lower memory addresses as it fills.

Stack Memory and Its Location:

- It's better to place the bottom of the stack at the end of the memory to keep it separate from the main program.
- **Setting the Stack Pointer (SP)**: The Stack Pointer (SP) register keeps track of the stack's current position. To set it, use an instruction like `LXI SP, FFFFH`, which sets the SP to the last memory address (FFFFH) for 8085.

Saving and Retrieving Information:

- **PUSH**: Saves data onto the stack.
- **POP**: Retrieves data from the stack.
- Both `PUSH` and `POP` work with register pairs (like B-C or D-E).

Examples:

- **PUSH B**:
  - SP decreases, then B is saved at the address SP points to.
  - SP decreases again, then C is saved in memory.
- **POP B**:
  - Data from the address SP points to is copied to C, then SP increases.
  - The next data is copied to B, then SP increases again.

.

Subroutines

What is a Subroutine?

A subroutine is a set of instructions that performs a specific task, and it can be reused in different parts of a program. Instead of writing the same code repeatedly, the instructions are grouped together in a subroutine, which can be called from various locations in the program.

How Subroutines Work:

- **CALL Instruction**: When the program reaches a `CALL` instruction, it temporarily stops its current work and "jumps" to the subroutine to execute its instructions.
- **RET Instruction**: Once the subroutine finishes its task, the `RET` (Return) instruction sends the program back to the point where it left off, continuing from where it stopped before the `CALL`.

Why Use Subroutines?

- **Efficiency**: By grouping commonly used instructions in a subroutine, the code is shorter and easier to read.
- **Reusability**: You can reuse the same code across the program by calling the subroutine whenever needed, avoiding repetition.

Subroutines make programs organized, efficient, and modular, which helps with maintenance and readability.

General Purpose Registers

These are the main registers used in assembly language for various operations:

- **AX (Accumulator Register)**:
  - Often used in arithmetic, logic, and data transfer tasks.
  - Can be divided into two 8-bit parts, **AH** and **AL**.
  - Special functions:
    - Fast machine code generation.
    - Required for certain instructions like multiplication, division, and input/output.
- **BX (Base Register)**:
  - Holds addresses for data in memory, often used in addressing.
  - Can be divided into **BH** and **BL**.

- **CX (Count Register)**:
    - Mainly used for loops and repetitive tasks.
    - Can be divided into **CH** and **CL**.
    - Important for:
        - Looping with `LOOP` instruction.
        - Repeating string operations with `REP` instruction.
        - Setting bit count for shifts and rotations.
- **DX (Data Register)**:
    - Often works together with AX for large calculations.
    - Can be split into **DH** and **DL**.
    - Important for:
        - Some 32-bit multiplication and division operations.
        - Defining ports for input/output operations.

Segment Registers

These help the processor locate different parts of memory:

- **CS (Code Segment)**: Points to the part of memory containing the current program code.
- **DS (Data Segment)**: Generally used to point to where variables and data are stored.
- **ES (Extra Segment)**: Used for additional data, defined by the programmer.
- **SS (Stack Segment)**: Points to the area of memory where the stack (temporary data storage) is kept.

Instruction Pointer

- **IP (Instruction Pointer)**: Always points to the next instruction to execute, ensuring the program runs in order.

Pointer and Index Registers

These registers are often used for handling data locations and memory:

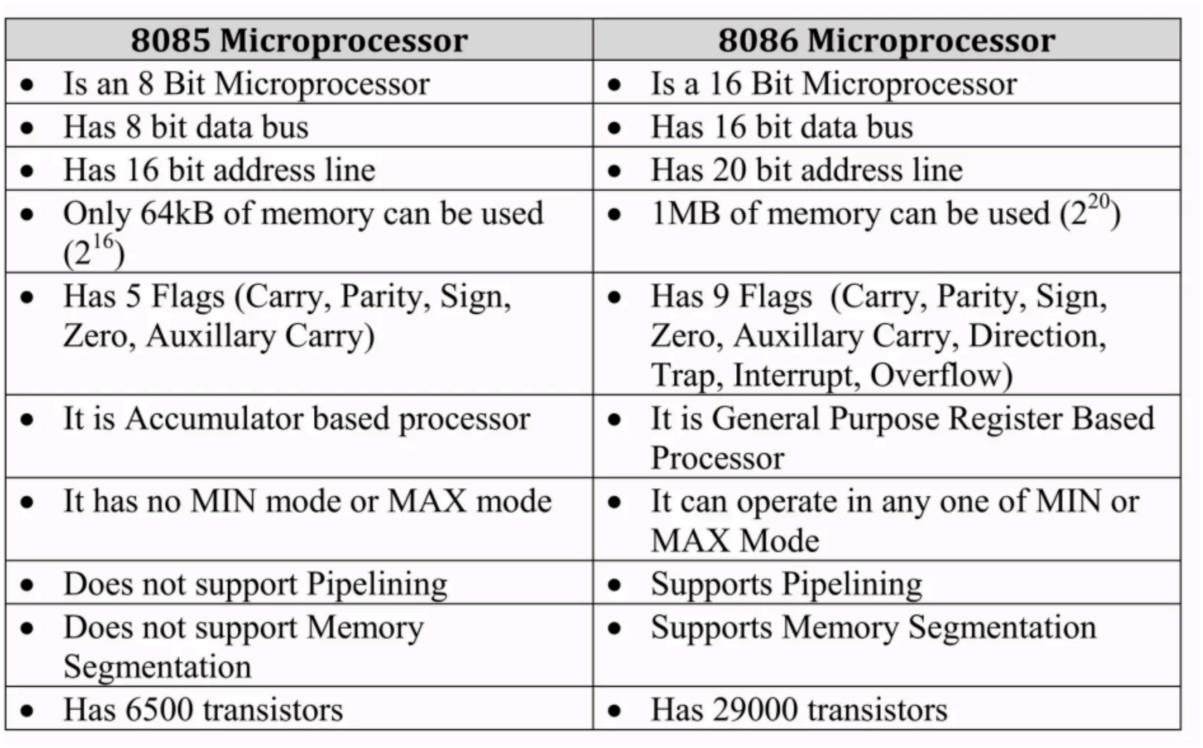
- **SI (Source Index)**:
  - Used as a pointer in string operations.
  - Points to source data for some instructions.
- **DI (Destination Index)**:
  - Points to destination data in string operations.
  - Often used as a destination pointer.
- **BP (Base Pointer)**:
  - Primarily for accessing data within the stack, especially parameters passed in functions.
- **SP (Stack Pointer)**:
  - Points to the top item on the stack.
  - If the stack is empty, SP holds the value **FFFEh**.

Flag Register in Microprocessors

In 8085: Fg-8085

In 8086: Fg-8086

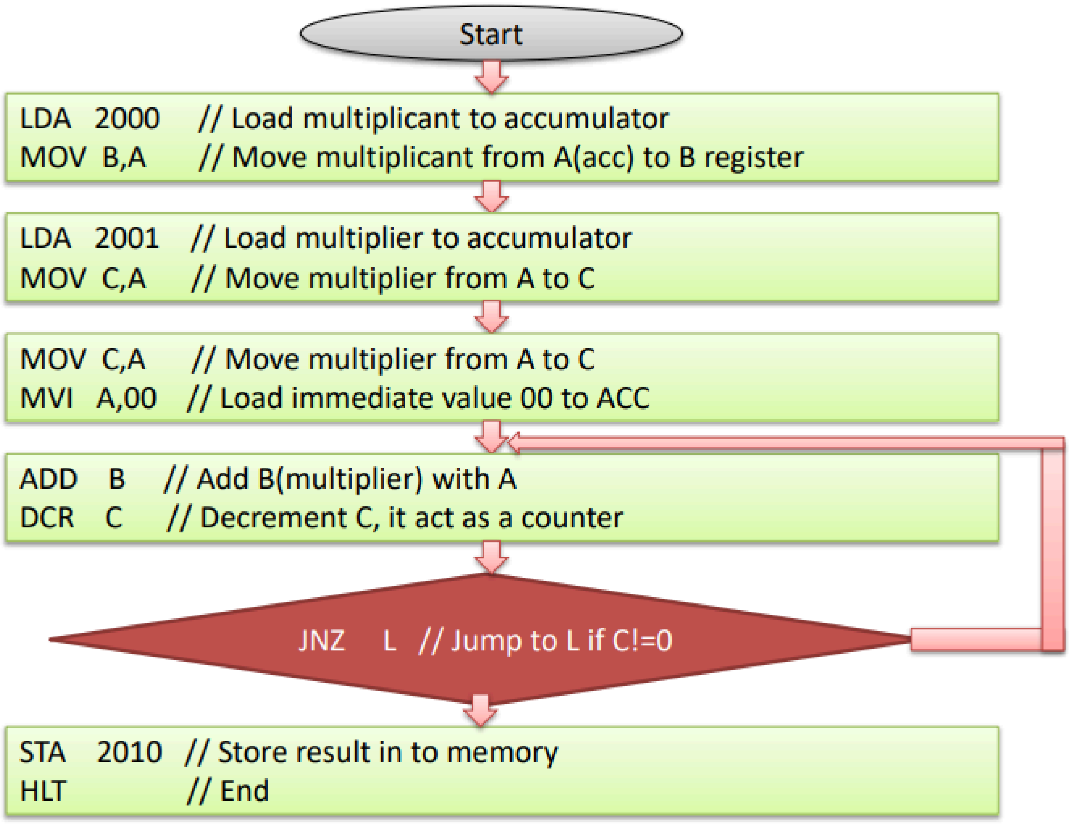
Difference Between 8085 and 8086 Microprocessor.



8085 Microprocessor	8086 Microprocessor
• Is an 8 Bit Microprocessor	• Is a 16 Bit Microprocessor
• Has 8 bit data bus	• Has 16 bit data bus
• Has 16 bit address line	• Has 20 bit address line
• Only 64kB of memory can be used (2^{16})	• 1MB of memory can be used (2^{20})
• Has 5 Flags (Carry, Parity, Sign, Zero, Auxillary Carry)	• Has 9 Flags (Carry, Parity, Sign, Zero, Auxillary Carry, Direction, Trap, Interrupt, Overflow)
• It is Accumulator based processor	• It is General Purpose Register Based Processor
• It has no MIN mode or MAX mode	• It can operate in any one of MIN or MAX Mode
• Does not support Pipelining	• Supports Pipelining
• Does not support Memory Segmentation	• Supports Memory Segmentation
• Has 6500 transistors	• Has 29000 transistors

Mam blse Eita Dibe. So…

Flowchart to multiply two number


Start
LDA 2000 // Load multiplicant to accumulator
MOV B,A // Move multiplicant from A(acc) to B register
LDA 2001 // Load multiplier to accumulator
MOV C,A // Move multiplier from A to C
MOV C,A // Move multiplier from A to C
MVI A,00 // Load immediate value 00 to ACC
ADD B // Add B(multiplier) with A
DCR C // Decrement C, it act as a counter
JNZ L // Jump to L if C!=0
STA 2010 // Store result in to memory
HLT // End

Expalnation:

1. **Start**: The process begins.
2. **LDA 2000 // Load multiplicant to accumulator**
   - Load the multiplicand (the first number) from memory location 2000 into the accumulator (A register).
3. **MOV B, A // Move multiplicant from A (accumulator) to B register**
   - Copy the value from the accumulator (A register) to the B register. Now, the B register holds the multiplicand.
4. **LDA 2001 // Load multiplier to accumulator**
   - Load the multiplier (the second number) from memory location 2001 into the accumulator (A register).
5. **MOV C, A // Move multiplier from A to C**
   - Move the multiplier value from the accumulator (A) to the C register. The C register will now act as a counter to keep track of how many times the addition (multiplication) is done.
6. **MOV C, A // Move multiplier from A to C (again)**
   - Reaffirming that C holds the multiplier for later use in the counting process.
7. **MVI A, 00 // Load immediate value 00 to ACC**
   - Set the accumulator (A register) to 0, initializing it for accumulating the multiplication result.
8. **ADD B // Add B (multiplicand) with A**
   - Add the value in the B register (multiplicand) to the current value in the accumulator (A). This step adds the multiplicand to A for the first time in the loop.
9. **DCR C // Decrement C, it acts as a counter**
   - Decrement the C register by 1. The C register serves as a counter to track the number of additions (equal to the multiplier).
10. **JNZ L // Jump to L if C ≠ 0**
    - Jump back to the label "L" (the start of the loop) if the C register is not zero. This creates a loop that repeats the addition until C (the multiplier count) reaches zero.
11. **STA 2010 // Store result in memory**

   - Store the final result from the accumulator (A register) into memory location 2010 after the loop is complete.
12. **HLT // End**
   - Halt the program execution, indicating the end of the multiplication process.

In summary, this flowchart performs multiplication by repeatedly adding the multiplicand (B) to itself as many times as specified by the multiplier (C). Each addition is tracked by decrementing the C register until it reaches zero, at which point the result is stored in memory

Questions & Answers:

Set 1:

1. Describe the role and working of the Program Counter in the 8085 microprocessor's instruction cycle. (3 marks)
The Program Counter (PC) keeps track of the address of the next instruction that the CPU will execute. When an instruction is fetched, the PC is updated to point to the next instruction. This ensures the microprocessor executes instructions in the correct order.

2. What is the function of the ALE (Address Latch Enable) pin in the 8085 architecture? Explain its role in data/address multiplexing. (4 marks)
The ALE pin helps separate the address and data on the same lines. In the 8085, the microprocessor uses the same pins to send both address and data, a technique called multiplexing. When the ALE pin is activated, it indicates that the current signals are addresses. This allows the system to correctly identify and use the address before switching to data transmission.

3. Explain the steps of an instruction cycle in the 8085, from fetching to execution. (3 marks)
The instruction cycle in the 8085 has three main steps:

1. **Fetch**: The CPU gets the instruction from memory using the address stored in the Program Counter.
2. **Decode**: The fetched instruction is then decoded to understand what action is needed.
3. **Execute**: Finally, the CPU performs the operation defined by the instruction.

4. How does the 8085 handle interrupts during program execution? (2 marks)
The 8085 can respond to interrupts, which are signals that temporarily stop the main program to handle urgent tasks. When an interrupt occurs, the microprocessor saves the current program state and jumps to a special routine to deal with the interrupt. After handling it, the microprocessor returns to the main program where it left off.

Set 2:

1. Define the purpose of the SID and SOD pins in the 8085 microprocessor. How are they used in data transmission? (3 marks)
The SID (Serial Input Data) pin is used to receive data, while the SOD (Serial Output Data) pin is used to send data. These pins allow the 8085 to communicate with other devices using serial data transmission, which means sending data one bit at a time over a single line.

2. Explain the function of the Instruction Decoder and how it interacts with other parts of the 8085 architecture. (4 marks)
The Instruction Decoder translates the instruction fetched from memory into signals that control other parts of the microprocessor. It tells the Arithmetic Logic Unit (ALU) and other components what operation to perform based on the instruction. This ensures the CPU executes the correct function.

3. Describe the roles of the ALU and Flag Register in executing instructions in the 8085. (3 marks)
The ALU (Arithmetic Logic Unit) performs all arithmetic and logical operations, like addition and subtraction. The Flag Register stores the status of the last operation (like whether it resulted in zero or a carry). This information helps the CPU make decisions for the next instructions based on the results.

4. What types of operations are impacted by the Flag Register? Provide examples. (2 marks)
The Flag Register affects operations like comparisons and conditional jumps. For example, if a subtraction results in zero, the Zero Flag is set, which can then be used in a conditional instruction to decide whether to jump to another part of the program.

Set 3:

1. Explain how data is stored and managed using the stack in the 8085 microprocessor, and describe the roles of PUSH and POP instructions. (4 marks)
The stack in the 8085 is a special memory area used to store data temporarily. The PUSH instruction adds data to the top of the stack, while the POP instruction removes data from the top. This is useful for saving return addresses during subroutine calls and for managing local variables.

---

2. What is the Program Status Word (PSW), and why is it important in the 8085 architecture? (3 marks)
The Program Status Word (PSW) is a collection of status flags that reflect the outcome of the last operation. It is important because it helps the CPU track conditions like whether the last operation was successful or if there was an overflow, allowing for appropriate responses in the program.

3. How should registers be handled before and after subroutine calls to preserve data integrity in 8085? (3 marks)
Before calling a subroutine, the values in registers should be saved to the stack to avoid losing data. After the subroutine finishes, the saved values can be restored from the stack, ensuring the main program continues with the correct data.

4. Define the purpose of the Stack Pointer in the 8085. (2 marks)
The Stack Pointer (SP) is a register that points to the top of the stack in memory. It helps the CPU know where to add or remove data in the stack during PUSH and POP operations.

Set 4:

1. Describe the basic structure and purpose of a subroutine in 8085 assembly language programming. (3 marks)
A subroutine is a set of instructions designed to perform a specific task that can be used multiple times in a program. It helps to organize code, making it easier to read and maintain. When a subroutine is called, the program jumps to its instructions and returns to the point where it was called once done.

2. Explain how the CALL and RET instructions work in the 8085 microprocessor, and how they relate to the stack. (4 marks)
The CALL instruction is used to jump to a subroutine. It saves the return address (the point to return to after the subroutine) onto the stack. The RET instruction is used to return from the subroutine, popping the return address from the stack so the program can continue from where it left off.

3. How can recursive functions be implemented in 8085 assembly? (3 marks)
Recursive functions can be implemented by having a subroutine that calls itself. Each call pushes a new return address onto the stack. The function continues to call itself until a base condition is met, at which point it starts returning, unwinding the stack and eventually returning to the original call.

4. Why is it recommended to place the bottom of the stack at the end of memory in the 8085? (2 marks)
Placing the bottom of the stack at the end of memory helps prevent stack overflow and ensures there is enough space for data to be added. This way, the stack grows downward, using memory efficiently without interfering with other program data.

Set 5:

1. Define the purpose of time delay loops in 8085 assembly programming and how they are implemented using MVI, DCR, and JNZ instructions. (3 marks)
Time delay loops are used to create pauses in program execution. They are implemented by loading a register with a value (MVI), decreasing that value (DCR), and using a jump instruction (JNZ) to loop back until the value reaches zero. This creates a delay based on the number of iterations.

---

2. Given a clock period and T-states for each instruction, calculate the total delay for a single loop. (4 marks)
To calculate the total delay, you multiply the number of T-states for each instruction in the loop by the clock period. For example, if an instruction takes 4 T-states and the clock period is 200 nanoseconds, the total delay for one instruction would be 4×2004 \times 2004×200 nanoseconds.

3. What role do T-states play in instruction timing, and why are they important in 8085 programming? (3 marks)
T-states are the basic units of time in the 8085 microprocessor. They measure how long each instruction takes to execute. Understanding T-states is important for programmers to optimize their code for speed and ensure timing meets the requirements of their applications.

4. Describe the factors that affect the speed of an 8085 assembly language program. (2 marks)
Factors affecting speed include the number of instructions, the complexity of operations, how efficiently the code is written, and the use of delays. Programs with fewer, simpler instructions generally run faster than those with many complex operations.

Set 6:

1. Compare the main architectural differences between the 8085 and 8086 microprocessors, particularly in terms of registers and data buses. (4 marks)
The 8085 is an 8-bit microprocessor with 8-bit registers and a 16-bit address bus, allowing it to access up to 64KB of memory. In contrast, the 8086 is a 16-bit microprocessor with 16-bit registers and a 20-bit address bus, enabling it to access up to 1MB of memory. This makes the 8086 more powerful than the 8085.

2. Describe the BIU (Bus Interface Unit) and EU (Execution Unit) in the 8086 microprocessor. How do they function independently? (3 marks)
The BIU manages data and address buses for the 8086, handling communication with memory and I/O devices. The EU executes instructions. They can work independently: while the EU processes instructions, the BIU can fetch the next instruction or data from memory, improving efficiency.

3. How do segment registers in the 8086 assist in memory management? (3 marks)
Segment registers in the 8086 (like CS, DS, ES, and SS) divide memory into segments, allowing the CPU to access different areas efficiently. This helps organize memory, making it easier to manage code, data, and stack areas, and supports larger programs.

4. Explain the roles of general-purpose registers AX, BX, CX, and DX in the 8086 microprocessor. (2 marks)
In the 8086, the general-purpose registers AX, BX, CX, and DX are used for various operations:

- **AX**: Used for arithmetic operations and as an accumulator.
- **BX**: Often used as a base pointer for memory access.
- **CX**: Used as a counter in loops and string operations.
- **DX**: Used for I/O operations and can store higher bits in multiplication and division.