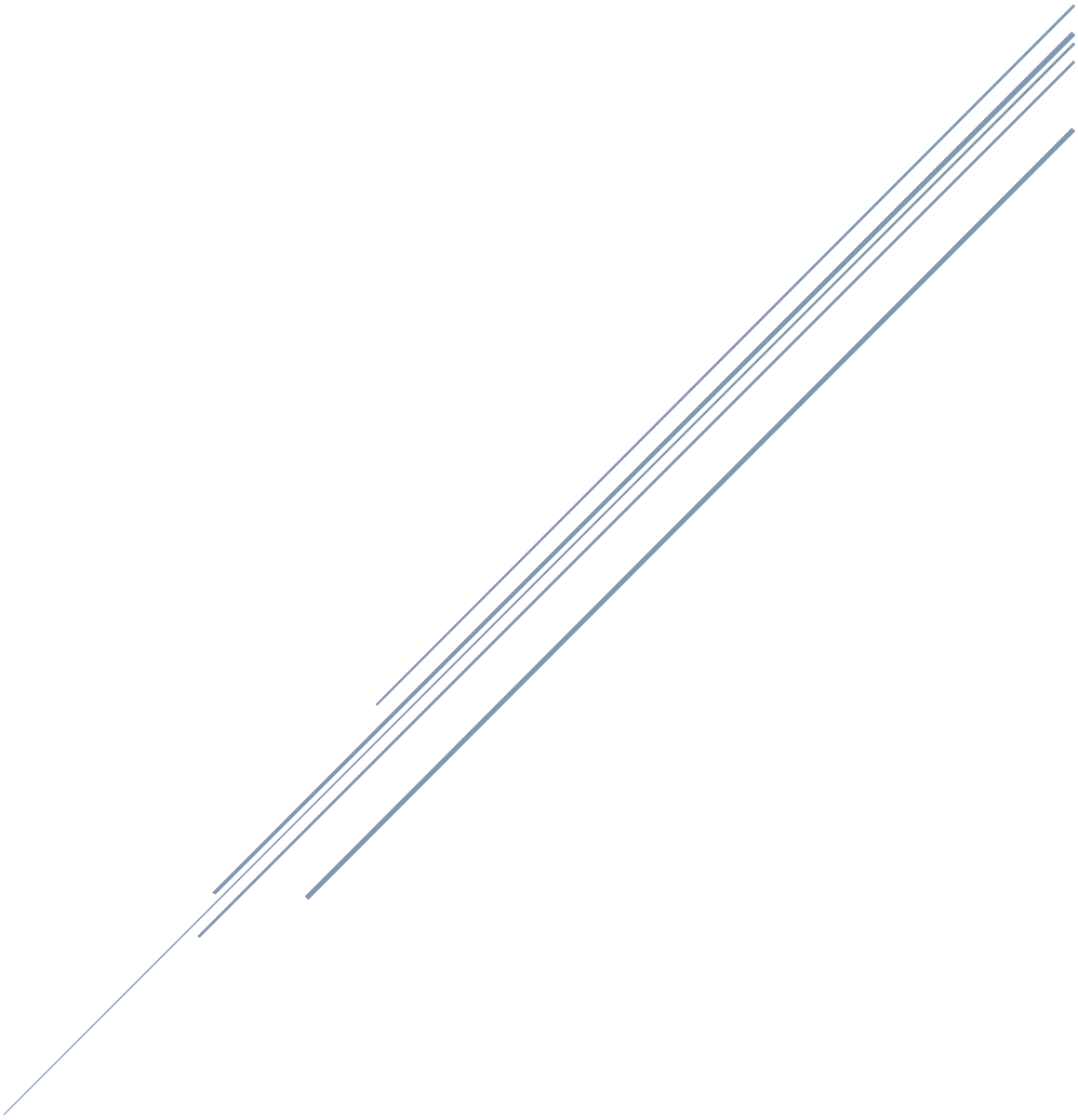
MICROPROCESSOR

Assignment



Mechi Multiple Campus
Bim Third Semester

QN. Define Microprocessor and Explain its Features .

Definition:

A **microprocessor** is a highly integrated circuit that functions as the **central processing unit (CPU)** of a computer or embedded system. It is responsible for executing program instructions, performing calculations, and controlling other parts of the system. It interprets instructions from memory and processes data accordingly.

Features of a Microprocessor:

- **Clock Speed:** The clock speed, measured in MHz or GHz, determines how many instructions the microprocessor can execute per second. A higher clock speed typically means faster performance.
- **Bit Width:** The bit width refers to the number of bits the microprocessor can process at a time. It can be 8-bit, 16-bit, 32-bit, or 64-bit, with larger widths allowing for faster and more efficient data processing.
- **Registers:** These are small, high-speed memory locations within the processor used to store data temporarily during processing.
- **Instruction Set:** This is the collection of instructions that the microprocessor can execute. It defines the operations the processor is capable of carrying out..
- **Multiprocessing Capability:** Some microprocessors have multiple cores, allowing them to handle more tasks simultaneously (parallel processing).
- **I/O Management:** The microprocessor controls the transfer of data to and from external devices, such as input devices (keyboards, mice) and output devices (monitors, printers).

QN. Explain Various Components of a Microprocessor .

A microprocessor consists of several integral components, each playing a crucial role in its operation:

a. Arithmetic and Logic Unit (ALU):

The **ALU** is responsible for performing **arithmetic operations** (such as addition, subtraction, multiplication, and division) and **logical operations** (such as AND, OR, NOT, XOR). It processes data from registers and stores results back in the registers or memory.

b. Control Unit (CU):

The **control unit** manages the flow of data between the different components of the microprocessor. It decodes instructions, sends control signals, and directs the operation of the ALU, registers, and memory. It ensures the correct sequence of operations is followed.

c. Registers:

Registers are small, high-speed memory locations used to store intermediate data temporarily. Examples of registers include the **Accumulator (A)** for storing results, **General-purpose registers (B, C, D, E, H, L)** for holding temporary data, **Stack Pointer (SP)** for tracking the stack's current position, and the **Program Counter (PC)** that stores the address of the next instruction to be fetched.

d. Cache Memory:

Cache memory is a small, fast memory unit that stores frequently accessed data to reduce the processor's access time to main memory. It is typically integrated within the processor or placed near it to maximize performance.

e. System Buses:

- **Address Bus:** This bus is used to carry addresses from the microprocessor to memory or I/O devices. It is **unidirectional**, meaning data flows in one direction, from the processor to memory.
- **Data Bus:** The data bus carries the actual data being processed or transferred. It is **bidirectional**, meaning it can send and receive data.
- **Control Bus:** The control bus transmits signals that control the flow of data and direct operations, such as **Read/Write signals** and **Interrupt signals**.

QN. Mention the Functions of Control Unit (CU) & Arithmetic Logic Unit (ALU) .

Functions of Control Unit (CU):

- Fetches instructions from memory.
- Decodes instructions into control signals.
- Directs data flow between ALU, registers, and memory.
- Generates timing and control signals.
- Coordinates execution of instructions.
- Manages communication with input/output devices.

Functions of Arithmetic Logic Unit (ALU):

- Performs arithmetic operations (addition, subtraction, multiplication, division).
- Executes logical operations (AND, OR, NOT, XOR).
- Handles comparison operations (greater than, less than, equal to).
- Updates status flags based on operation results (Zero flag, Carry flag, Sign flag, etc.).
- Works in coordination with registers and the control unit.

QN. Explain About Bus Organization in Microprocessor .

A **bus** is a collection of wires or lines that allow different components of a computer system to communicate with each other. Buses are categorized based on their function:

1. Address Bus:

- Unidirectional (data flows in one direction only, from CPU to memory or I/O devices).
- Carries the address of the memory location or I/O device being accessed.
- The width of the address bus determines the memory addressing capability of the processor. For example, an 8085 microprocessor has a 16-bit address bus, allowing it to access 64 KB of memory.

2. Data Bus:

- Bidirectional (data can flow both to and from the processor).
- Transfers actual data between the processor, memory, and I/O devices.
- The width of the data bus affects processing speed. An 8-bit data bus, like in the 8085, transfers 8 bits of data at a time.

3. Control Bus:

- Carries control signals that manage the operations of the microprocessor.
- Includes signals such as Read (RD), Write (WR), Memory Read (MEMR), Memory Write (MEMW), and Interrupt Request (INTR).
- Synchronizes data transfer and execution of instructions.

QN. Write About Control and Timing Signals in Microprocessor.

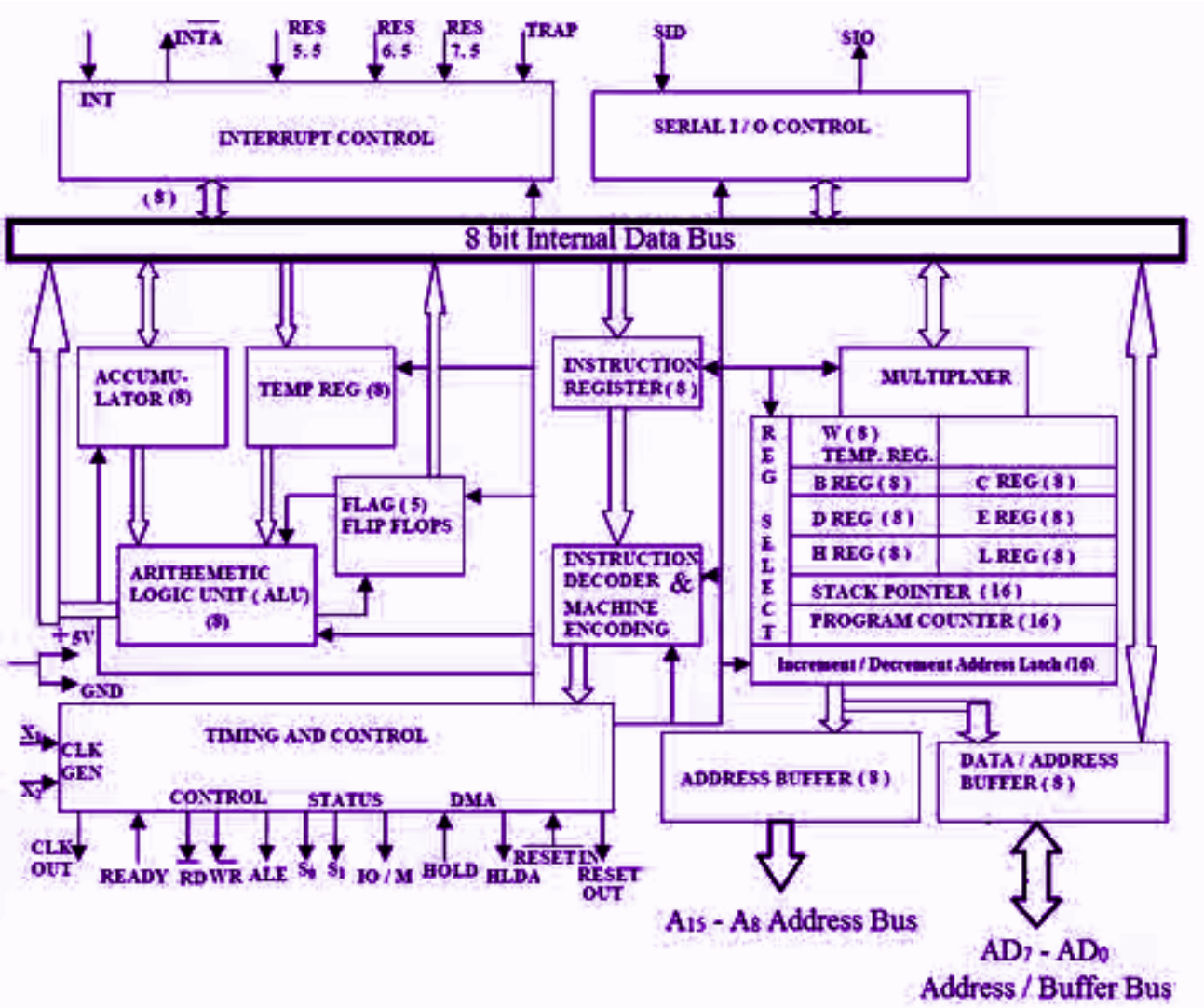
Control and timing signals are used to synchronize operations within the microprocessor and with other components. They ensure that instructions are fetched, decoded, and executed in the correct sequence and that data transfers occur correctly.

Key Signals:

- **Clock Signal:** This signal synchronizes all operations in the processor by providing a timing reference. It dictates the speed at which instructions are processed.
- **Read (RD) Signal:** The Read signal indicates that data is being read from memory or an I/O device.
- **Write (WR) Signal:** The Write signal indicates that data is being written to memory or an I/O device.
- **Interrupt Signals:** Interrupts are used to notify the processor of an external event or request, such as input from a user or completion of an I/O operation.

SAP 1	SAP 2
Data Bus is 8-bit	Data Bus is 16-bit
PC is 4-bit.	PC is 16-bit.
It does not have hexadecimal keyboard encoder.	It has hexadecimal keyboard encoder.
It has single input.	It has two input ports.
MAR receives 4-bit address from PC.	MAR receives 16-bit address from PC.
It does not have ROM.	It has 2 KB ROM.
It has 16 Byte memory.	It has 62 KB memory.
It does not have MDR.	It has MDR.
It has only adder/subtractor.	It has ALU.
It does not have flag.	It has 2 flags.
It does not have temporary register.	It has temporary register.
It has single register (B).	It has 2 registers (B and C).
It has single output port.	It has 2 output ports.
It has 5 instruction sets.	It has 42 instruction sets.

QN.Explain Internal Architecture of Intel 8085 .



The **internal architecture of the Intel 8085 microprocessor** consists of various components that work together to execute instructions efficiently. The main components include:

a. Arithmetic and Logic Unit (ALU):

- Performs arithmetic operations like addition, subtraction, multiplication, and division.
- Executes logical operations like AND, OR, XOR, and NOT.
- Sets or resets flags based on the operation results.
- Supports increment (INR) and decrement (DCR) operations to increase or decrease values stored in registers.

b. Control Unit (CU):

- Directs data flow within the microprocessor.
- Decodes instructions and generates control signals.
- Synchronizes operations using the system clock.

c. Registers:

- Temporary storage locations inside the CPU.
- Includes:
    - **Accumulator (A):** Stores intermediate results of arithmetic and logical operations.
    - **General-purpose registers (B, C, D, E, H, L):** Used for temporary data storage.
    - **Stack Pointer (SP):** Points to the top of the stack in memory.
    - **Program Counter (PC):** Holds the address of the next instruction to be executed.
    - **Instruction Register (IR):** Stores the current instruction being executed.
    - **Increment (INR) and Decrement (DCR) Registers:** Used to increase or decrease values stored in registers without affecting the carry flag.

d. Instruction Decoder:

- Converts the fetched instruction into control signals.
- Determines the operation type and operands.

e. Timing and Control Circuit:

- Generates the necessary clock signals to synchronize operations.
- Manages read and write control signals for memory and I/O operations.

f. Interrupt Control:

- Handles external and internal interrupts.
- Prioritizes interrupts and executes corresponding service routines.

g. Serial I/O Control:

- Manages serial communication through SID (Serial Input Data) and SOD (Serial Output Data) lines.

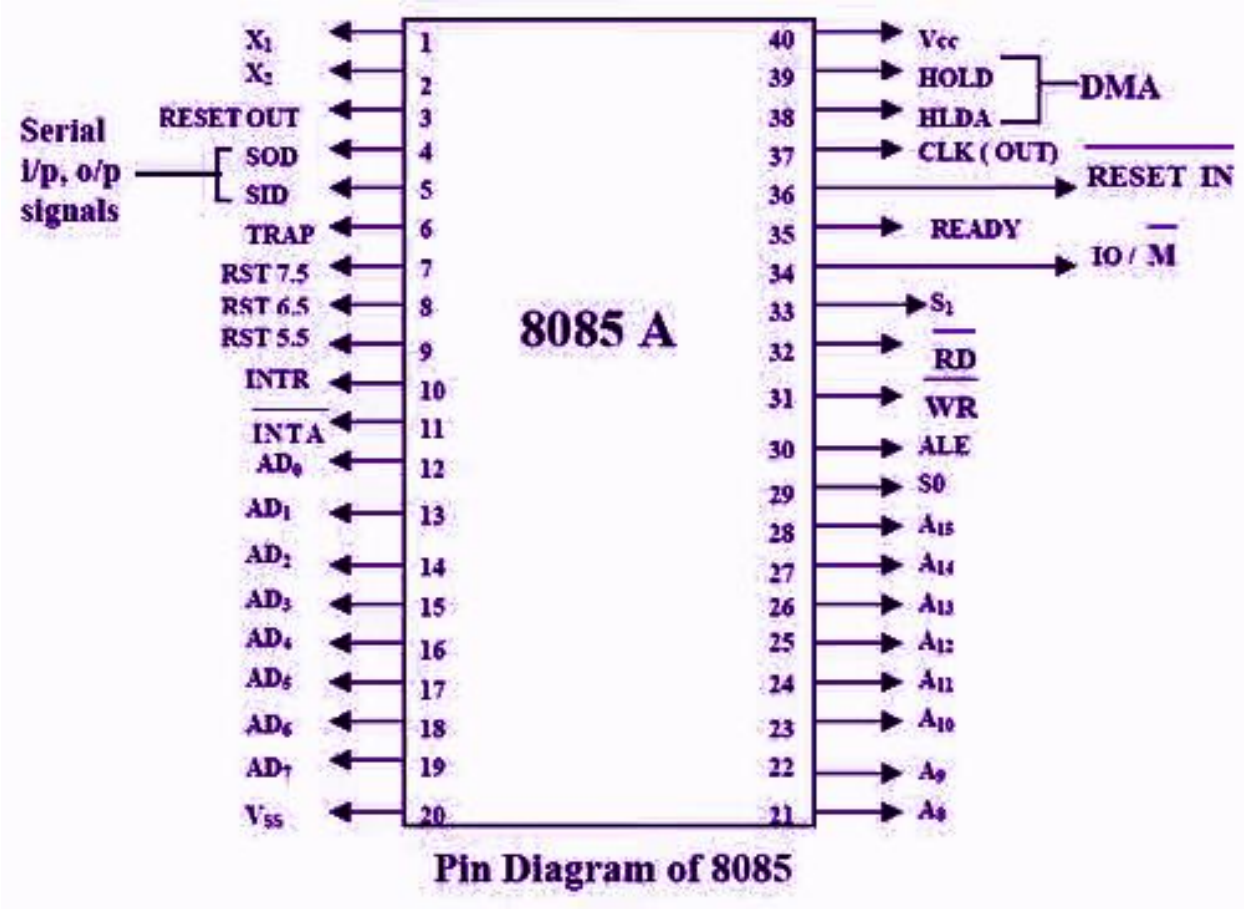
h. Address and Data Buses:

- **Address Bus:** 16-bit unidirectional bus used to specify memory locations.
- **Data Bus:** 8-bit bidirectional bus used to transfer data between the processor and memory or I/O devices.

i. Flag Register:

- Stores status flags (Zero, Sign, Carry, Parity, and Auxiliary Carry) that indicate the result of an operation.

QN. Pin Diagram of Intel 8085 Microprocessor: Detailed Explanation



Pin Diagram of 8085

The **Intel 8085** is an **8-bit microprocessor** with a **16-bit address bus**, which allows it to access up to **64KB of memory**. The 8085 microprocessor has a total of **40 pins** which serve various functions such as power supply, data transfer, control signals, interrupt handling, and more.

Let's break down the **40 pins** of the **8085 microprocessor** and explain their functions in detail.

Pin Diagram Overview

The 8085 microprocessor is organized in the following manner:

1. **Power Supply Pins (2 pins)**
2. **Clock Pins (2 pins)**
3. **Address Pins (16 pins)**
4. **Data Pins (8 pins)**
5. **Control and Status Pins (9 pins)**
6. **Interrupt Pins (5 pins)**
7. **Serial Input/Output Pins (2 pins)**

Explanation of Each Pin

1. Power Supply Pins

- **Vcc (Pin 40)**:
  - This is the **positive power supply** pin. It is connected to +5V (positive voltage) to power the microprocessor.
- **GND (Pin 20)**:
  - This is the **ground** pin. It is connected to the 0V (negative or ground voltage) to complete the circuit.

2. Clock Pins

- **X1 (Pin 1) and X2 (Pin 2)**:
  - These pins are used for the **clock oscillator** input. The 8085 requires an external clock to synchronize the operations of the microprocessor. A **crystal oscillator** is connected to these pins to generate the clock signal for the processor.
- **CLK OUT (Pin 6)**:
  - This pin outputs the **clock signal** generated by the internal oscillator. It is typically used to provide a clock signal to other components in the system.

3. Address Pins

The 8085 microprocessor has **16 address pins**, which are used to address memory and input/output (I/O) devices.

- **A0-A15 (Pins 19-17, 1-8)**:
    - These are the **address pins**. The 16-bit address bus allows the 8085 to access memory locations ranging from **0000H to FFFFH** (64KB of memory).
    - The 8085 uses these pins to send the address to memory or I/O devices, to read data from or write data to specific locations.

4. Data Pins

- **AD0-AD7 (Pins 11-18)**:
    - These are the **data pins**. These pins carry the data being transferred to or from the microprocessor.
    - **AD0-AD7** are multiplexed pins, meaning they act as both **address** and **data** pins. When the **address** is placed on these pins, it is used for addressing purposes. During the data transfer phase, the pins act as **data lines** for the exchange of data between the processor and memory or I/O devices.

5. Control and Status Pins

Control and status signals help manage operations like reading/writing data, memory or I/O access, and more.

- **IO/M (Pin 28)**:
    - This pin determines whether the operation involves **I/O** devices or **memory**.
    - If **IO/M = 0**, it indicates a **memory operation**.
    - If **IO/M = 1**, it indicates an **I/O operation**.

- **RD (Pin 32)**:
    - The **Read** signal is used to indicate that data is being **read** from memory or I/O.
    - If **RD = 0**, the microprocessor is reading data from memory or I/O devices.
- **WR (Pin 31)**:
    - The **Write** signal is used to indicate that data is being **written** to memory or I/O devices.
    - If **WR = 0**, the microprocessor is writing data to memory or I/O.
- **ALE (Pin 25)**:
    - The **Address Latch Enable** signal is used to latch the address into a buffer.
    - It ensures that the address is correctly stored before the data is accessed.
- **S0 and S1 (Pins 29 and 30)**:
    - These are **status pins** used to provide information about the current operation. The combination of these two pins helps in determining whether the operation is a **memory read**, **memory write**, **I/O read**, or **I/O write**.

6. Interrupt Pins

Interrupt pins allow external devices to interrupt the microprocessor's current task to provide higher priority services.

- **INTR (Pin 6)**:
    - **Interrupt Request** pin. It is used by external devices to request the microprocessor's attention.
    - The 8085 responds to this request by interrupting the current task.
- **TRAP (Pin 9)**:
    - **TRAP** is a **non-maskable interrupt** (cannot be disabled). It has the highest priority and is used for critical operations.

- **RST7.5, RST6.5, and RST5.5 (Pins 7, 8, 9)**:
    - These are **maskable interrupt** pins. They can be **enabled or disabled** by software.
    - These interrupts are used to temporarily halt the current task and allow external devices to perform important operations.
- **INTA (Pin 34)**:
    - **Interrupt Acknowledge** pin. It is used by the microprocessor to acknowledge the receipt of an interrupt request.

7. Serial Input/Output Pins

These pins are used for serial data communication.

- **SID (Pin 37)**:
    - **Serial Input Data** pin. This pin is used to receive serial data from external devices.
- **SOD (Pin 38)**:
    - **Serial Output Data** pin. This pin is used to transmit serial data to external devices.

QN. Define Interrupts and Explain its Types.

An **interrupt** is a mechanism that allows a **microprocessor** or **microcontroller** to temporarily halt its current task and divert its attention to a higher priority task. The process of handling an interrupt is known as **interrupt handling**, and it allows the microprocessor to respond to external or internal events in real-time, without having to constantly check the event status.

When an interrupt occurs, the processor stops executing its current instruction, saves the state (context) of its registers, and jumps to a predefined address (interrupt vector) where the interrupt service routine (ISR) is located. After the ISR finishes, the processor resumes executing the interrupted program from where it left off.

Types of Interrupts

Interrupts can be classified into several types based on their source, priority, and whether or not they can be disabled. Here are the key categories:

---

1. Based on Enable/Disable

- **Maskable Interrupts**:
    - These interrupts can be **enabled** or **disabled** by software. The processor can ignore (mask) these interrupts if the current task is critical and cannot be interrupted.
    - Examples in **Intel 8085**:
        - **RST5.5**
        - **RST6.5**
        - **RST7.5**
- **Non-Maskable Interrupts (NMI)**:
    - These interrupts **cannot be disabled** or masked, meaning the processor must respond to them immediately, regardless of the current task.
    - Example in **Intel 8085**:

- **TRAP** (Non-maskable interrupt)

2. Based on Source

- **Software Interrupts**:
  - Software interrupts are generated by executing a specific **software instruction**. These interrupts are typically used to call certain routines or handle system calls within a program.
  - Example in **Intel 8085**: The **CALL** instruction can trigger a software interrupt by executing the corresponding interrupt service routine.
- **Hardware Interrupts**:
  - Hardware interrupts are generated by **external hardware devices**, such as a timer, keyboard, or sensor. When the hardware device needs attention, it sends an interrupt signal to the processor.
  - Example in **Intel 8085**: The **INTR** pin is used to request an interrupt from an external device.

3. Based on Priority

- **Vectored Interrupts**:
  - In vectored interrupts, the processor is provided with an address (called the **interrupt vector**) where the interrupt service routine (ISR) starts. The processor directly jumps to this address when the interrupt occurs.
  - **Example in 8085**: The **RST5.5**, **RST6.5**, **RST7.5**, and **TRAP** are vectored interrupts. Each has its own fixed memory address where its ISR is located.
- **Non-Vectored Interrupts**:

- In non-vectored interrupts, the interrupting device must send the address of the interrupt service routine (ISR) to the processor. The processor doesn't know where to go for the ISR until it receives the address.
- **Example in 8085**: The **INTR** interrupt is non-vectored, meaning the processor does not know where the ISR is located until it is provided by the external device.

Interrupt Process Flow

Here is the step-by-step process that happens when an interrupt occurs:

1. **InterruptSignal**:
   The interrupt is generated by an external or internal device. For example, if an I/O device needs attention, it sends an interrupt signal to the processor.
2. **InterruptDetection**:
   The processor continuously monitors the interrupt lines. When an interrupt occurs, the processor checks whether it is enabled and if the interrupt has a higher priority than the current task.
3. **InterruptAcknowledgment**:
   The processor acknowledges the interrupt. For some types of interrupts, this acknowledgment is sent back to the external device (e.g., **INTA** signal in 8085).
4. **SavingtheContext**:
   The processor saves the **current state** (context) of the execution, including the program counter (PC) and the status registers. This is necessary to resume the task later.
5. **InterruptServiceRoutine(ISR)**:
   The processor jumps to the interrupt vector address (for vectored interrupts) or waits for the address to be provided (for non-vectored interrupts) and executes the corresponding **Interrupt Service Routine (ISR)**.
6. **ReturntoNormalExecution**:
   After the ISR is executed, the processor restores its state (context) from the saved registers and **resumes** the interrupted task from where it left off.

QN.Interrupts in Intel 8085

The **Intel 8085** microprocessor has several interrupt types, each with distinct characteristics:

1. Maskable Interrupts (Can be Disabled/Enabled by Software)

- **RST5.5 (Pin 5)**:
  - It is a maskable interrupt with a priority lower than TRAP. It is triggered by an external hardware device.
  - It is vectored to address **0024H**.
- **RST6.5 (Pin 6)**:
  - It is a maskable interrupt with a priority higher than RST5.5 but lower than TRAP.
  - It is vectored to address **0028H**.
- **RST7.5 (Pin 7)**:
  - It is a maskable interrupt with a higher priority than RST5.5 and RST6.5.
  - It is vectored to address **002CH**.

2. Non-Maskable Interrupt (Cannot be Disabled/Enabled)

- **TRAP (Pin 9)**:
  - TRAP is the highest priority interrupt in the 8085 microprocessor.
  - It is **non-maskable**, meaning it cannot be disabled by the software.
  - It is vectored to address **0024H** and is used for emergency processing.

3. Non-Vectored Interrupt

- **INTR (Pin 6)**:
  - This is a non-vectored interrupt. The processor must be given the address of the ISR by the external device.
  - It is vectored by external devices, unlike the other interrupts in the 8085.

QN.Define Flag and Explain its Types .

In microprocessors, **flags** are special-purpose **status bits** that provide information about the outcome of the last executed operation. These flags are typically stored in a **flag register**, which is a special register in the processor. Flags are used to monitor the results of arithmetic, logical, or control operations, and they help in decision-making processes during program execution.

1. Carry Flag (CY)

- **What it does:** Indicates whether there is a carry (in addition) or borrow (in subtraction).
- **Set to 1:** If the result of an operation generates a carry (in addition) or borrow (in subtraction).
- **Set to 0:** If no carry or borrow occurs.
- **Example:**
    - **Addition:** A = 250, B = 20 → A + B = 270 → Carry flag = 1.
    - **Subtraction:** A = 100, B = 50 → A - B = 50 → Carry flag = 0.

2. Zero Flag (Z)

- **What it does:** Indicates if the result of an operation is zero.
- **Set to 1:** If the result of the operation is zero.
- **Set to 0:** If the result is non-zero.
- **Example:**
    - **Subtraction:** A = 100, B = 100 → A - B = 0 → Zero flag = 1.
    - **Addition:** A = 50, B = 30 → A + B = 80 → Zero flag = 0.

3. Sign Flag (S)

- **What it does:** Indicates whether the result of an operation is positive or negative (in two's complement).
- **Set to 1:** If the result is negative (MSB = 1).
- **Set to 0:** If the result is positive (MSB = 0).
- **Example:**
  - **Addition/Subtraction:** A = 100, B = -200 → A + B = -100 → Sign flag = 1.
  - **Addition/Subtraction:** A = 50, B = 30 → A + B = 80 → Sign flag = 0.

4. Parity Flag (P)

- **What it does:** Indicates whether the result has an **even** or **odd** number of 1s.
- **Set to 1:** If the result has an even number of 1s (even parity).
- **Set to 0:** If the result has an odd number of 1s (odd parity).
- **Example:**
  - **Addition/Subtraction:** A = 3 (00000011), B = 5 (00000101) → A + B = 8 (00001000) → Parity flag = 0 (odd parity).
  - **Logical Operation:** A = 7 (00000111), B = 5 (00000101) → A OR B = 7 (00000111) → Parity flag = 1 (even parity).

5. Auxiliary Carry Flag (AC)

- **What it does:** Used for **BCD** (Binary-Coded Decimal) operations to indicate a carry from bit 3 to bit 4.
- **Set to 1:** If there is a carry from bit 3 to bit 4 during **BCD** addition.
- **Set to 0:** If there is no carry from bit 3 to bit 4.
- **Example:**
  - **BCD Addition:** A = 0x59, B = 0x37 → A + B = 0x90 → Auxiliary Carry flag = 1 (carry between nibbles).
  - **BCD Addition:** A = 0x25, B = 0x16 → A + B = 0x41 → Auxiliary Carry flag = 0 (no carry between nibbles).

QN. Define Addressing Modes and Explain its Types .

Addressing modes define the method by which an instruction locates its operand (data or memory location). Different addressing modes allow the microprocessor to access data in different ways, providing flexibility in how data is manipulated. In essence, addressing modes define how the instruction specifies the location of the operand (data), whether it's directly in the instruction, stored in a register, or retrieved from memory.

---

1. Immediate Addressing Mode

In **immediate addressing mode**, the operand (data) is specified directly within the instruction. The data is available immediately, without the need to fetch it from memory or a register.

Example:

MVI A, 45H

Advantages:

- **Speed:** Immediate operands do not require memory accesses, which makes the instruction execution faster.
- **Simplicity:** It's easy to use for loading specific values into registers or memory locations.
- **Compact:** The instruction is usually shorter, as the operand is directly part of the instruction.

Disadvantages:

- **Limited Flexibility:** You can only load small, fixed values directly into the register or memory location.
- **Space Limitation:** Since the operand is part of the instruction, large data values (beyond the size limit of the instruction set) can't be used in this mode.

---

2. Register Addressing Mode

In **register addressing mode**, the operand is located in one of the processor's registers. The operand's address is implied, and the operation is performed directly on the register.

Example:

MOV A, B

Advantages:

- **Fast Access:** Registers are high-speed memory, so operations are quicker compared to memory accesses.
- **Low Overhead:** No need to fetch data from memory, so fewer cycles are consumed.

Disadvantages:

- **Limited by Number of Registers:** Since only the processor's registers are used, the number of operands that can be accessed is limited.
- **Not Suitable for Large Data:** Larger data sets cannot be directly operated on in registers, as registers have limited size.

---

3. Direct Addressing Mode

In **direct addressing mode**, the operand's address is explicitly specified in the instruction. This mode allows the CPU to access data from a specific memory location.

Example:

LDA 3000H

Advantages:

- **Access to Memory:** Allows access to any specific memory location directly, useful for larger datasets.
- **Simple to Use:** It's straightforward to load or store data to/from a known memory address.

Disadvantages:

- **Slower:** Memory accesses are slower compared to registers.

- **Limited Addressing Range:** The instruction can only access a limited range of memory (e.g., in the 8085, the memory address space is 64KB).

---

4. Indirect Addressing Mode

In **indirect addressing mode**, the operand's address is stored in a register pair (such as BC, DE, or HL). The instruction specifies which register pair holds the memory address of the operand, and the processor uses that address to access the operand.

Example:

LDAX B

Advantages:

- **Flexibility:** The address of the operand is dynamic, as it can be changed by modifying the register pair, allowing for more flexible addressing.
- **Support for Arrays and Tables:** Useful when accessing arrays or tables, as the register pair can point to different addresses during execution.

Disadvantages:

- **More Complex:** Requires an additional level of indirection (register pair) to access the operand, which can increase execution time.
- **Slower:** Memory access through indirect addressing takes longer because two memory operations are involved (one to get the address and another to fetch the data).

---

5. Implied Addressing Mode

In **implied addressing mode**, the operand is implicitly defined by the instruction itself. There is no need to specify the operand's location because the operation is defined by the opcode alone.

Example:

CMA

Advantages:

- **Efficiency:** No need for explicit operands in the instruction, which makes the instruction simpler and faster to execute.
- **Compact Code:** The instructions are very compact because the operand is not explicitly specified.

Disadvantages:

- **Limited Functionality:** The operations in implied addressing are typically more limited and are often used for simple tasks like rotating or complementing values in registers.
- **Less Flexibility:** You cannot directly use external data or memory locations with implied addressing.

QN. Define Microoperations and Explain its Types .

Microoperations are the operations executed on the data stored in the registers.

Types of **Microoperations** include :

---

1. Register Transfer Operations

Definition:
Register transfer operations involve transferring data between registers or between registers and memory. These operations are the fundamental building blocks of moving data around within the microprocessor.

Examples:

- **Register to Register Transfer:**
  R1 ← R2
  This means that the contents of **R2** are transferred to **R1**.
- **Memory to Register Transfer:**
  R1 ← M[100]
  This means that the data from memory location **100** is transferred to **R1**.
- **Register to Memory Transfer:**
  M[200] ← R2
  This means that the contents of **R2** are stored in memory location **200**.

---

2. Arithmetic Operations

Definition:
Arithmetic micro-operations are used to perform basic arithmetic operations such as addition, subtraction, multiplication, and division on the data stored in registers or memory.

Examples:

- **Addition:**
  R3 ← R1 + R2
  This means the contents of **R1** and **R2** are added together, and the result is stored in **R3**.

- **Subtraction:**
  R3 ← R1 - R2
  This means the contents of **R2** are subtracted from **R1**, and the result is stored in **R3**.
- **Increment:**
  R1 ← R1 + 1
  This means **R1** is incremented by 1.
- **Decrement:**
  R1 ← R1 - 1
  This means **R1** is decremented by 1.

---

3. Logical Operations

Definition:
Logical micro-operations perform logical operations on binary data, such as **AND**, **OR**, **NOT**, and **XOR**. These operations are used for decision-making and bit manipulation in microprocessor systems.

Examples:

- **AND Operation:**
  R1 ← R1 AND R2
  This means that the contents of **R1** and **R2** are logically **AND-ed**, and the result is stored back in **R1**.
- **OR Operation:**
  R1 ← R1 OR R2
  This means that the contents of **R1** and **R2** are logically **OR-ed**, and the result is stored back in **R1**.
- **NOT Operation (Complement):**
  R1 ← NOT R1
  This means that the contents of **R1** are complemented (flipped), and the result is stored back in **R1**.
- **XOR Operation:**
  R1 ← R1 XOR R2

This means that the contents of **R1** and **R2** are logically **XOR-ed**, and the result is stored back in **R1**.

---

4. Shift Operations

Definition:
Shift operations involve shifting the bits of data within a register, either to the left or right. These operations are important for tasks like multiplying or dividing numbers by powers of 2, or for bit manipulation tasks.

Types of Shift Operations:

- **Logical Shift Left (LSL):**
  R1 ← R1 << 1
  This means that the contents of **R1** are shifted one position to the left. A zero is filled in at the rightmost bit.
- **Logical Shift Right (LSR):**
  R1 ← R1 >> 1
  This means that the contents of **R1** are shifted one position to the right. A zero is filled in at the leftmost bit.
- **Arithmetic Shift Left (ASL):**
  R1 ← R1 << 1
  Similar to the logical shift left, but **ASL** preserves the sign bit in the case of signed numbers.
- **Arithmetic Shift Right (ASR):**
  R1 ← R1 >> 1
  Similar to the logical shift right, but **ASR** preserves the sign bit (the leftmost bit) in the case of signed numbers.

QN. Explain 5-Bit Adder-Subtractor Circuit .