Intro to Microprocessors

1.What are the differences between microprocessors and microcontrollers?

Microprocessor	Microcontroller
Used where intensive processing is required	Used where task is fixed and predefined
Only the CPU is in the chip. Memory, I/O port are connected externally	CPU, Memory, I/O port – all are connected on the same single chip
Higher Clock speed and external RAM used is also higher	Lower Clock speed and RAM used is also lower
The program for the microprocessor can be changed for different applications.	The program for the microcontroller is fixed once it is designed
Cost is comparatively higher	Cost is comparatively lower
Power consumption is higher	Power consumption is lower
Overall size of the system is large	Overall size of the system is smaller
Applications include personal computers	Applications include washing machines, cameras etc.

2.Explain the major components of a CPU and their functions.

- Control Unit & Instruction Decoder: To synchronize and control the overall operation of the CPU & To decode instruction and pass the necessary control signals to CU
- Arithmetic/Logic Unit: To perform shift and rotate operations that may either be arithmetic or logical in nature/ To perform the arithmetic and logical operations within the CPU
- Registers: A set of internal storage locations within the CPU

3.For each of the following devices, would you use a microcontroller or a microprocessor?

- **Keyboard** – Microcontroller
- **Mouse** – Microcontroller
- **Headphone** – Microcontroller
- **Computer** – Microprocessor
- **TV Remote** – Microcontroller
- **Smart TV** – Microprocessor
- **Regular fridge** – Microcontroller
- **Mobile phone** – Microprocessor

4.Imagine you are designing a smart home system. Would you use a microprocessor or a microcontroller? Justify your choice.

I would use a microcontroller.

Reason:
A smart home system needs to interact with sensors (temperature, motion, humidity), perform simple control tasks (turn lights on/off, adjust fan speed), and communicate over Wi-Fi/Bluetooth. These tasks require **real-time control**, **low power**, and **embedded I/O**, all of which a **microcontroller** provides efficiently. It is also cheaper and easier to integrate into home appliances.

5.Your friend claims that an 8-bit microprocessor is always faster than a 4-bit microprocessor. How would you explain the factors that influence processor performance?

My friend's claim is not always true. The bit-size alone does not determine speed. Here's how to explain it:

Factors that influence processor performance:

- Bit-size alone **does not determine** processor speed.
- **Clock speed** affects how many instructions can be executed per second.
- **Instruction set efficiency** can make a smaller-bit processor faster.

- **Architecture design** (pipelining, registers, ALU) impacts performance.
- **Memory access speed** influences how quickly data is handled.
- A 4-bit processor **can be faster** if it has better design or higher clock speed.

6.Suppose you are designing an embedded system for a washing machine. What features of a microcontroller make it a better choice than a microprocessor?

A microcontroller is a better choice for a washing machine because it provides the following features:

- **Built-in peripherals** (timers, ADCs, PWM, I/O ports) to control motors, sensors, water valves, etc.
- **Low power consumption**, ideal for always-on household appliances.
- **Real-time control capability** for handling wash cycles, spin speed, and water levels.
- **Lower cost** compared to using a microprocessor with external components.
- **Compact design** with CPU, memory, and I/O integrated on a single chip.
- **High reliability** for repetitive control tasks in appliances.

7.A company wants to build a high-performance computing system. What type of processor architecture (RISC or CISC) would you recommend and why?

Recommend **RISC** for high-performance computing.

- **Simple, fast instructions** allow quick execution.
- **Efficient pipelining** boosts performance.
- **Scales well** with multi-core architecture.
- **Lower power and heat** due to simpler design.
- Used in many **modern high-performance systems**.

8.A device needs high processing power for graphics, support for input devices, and the ability to update games. Power consumption should be balanced for longer use. As a consultant, help them choose whether to use a µP or a µC and give reason.

Use a microprocessor (µP).

- It supports high graphics performance, input devices, and game updates.
- µPs handle **complex graphics** and GPU tasks much better than microcontrollers.
- µPs support **OS, drivers, storage, and software updates** easily.
- They offer **more RAM and multitasking**, needed for games.
- Modern µP-based SoCs provide **good performance with balanced power use**.

9.Write an assembly program to perform the following operations:

- Store the value 7 in a register.
- Add 3 to it.
- Subtract 5 from the result.

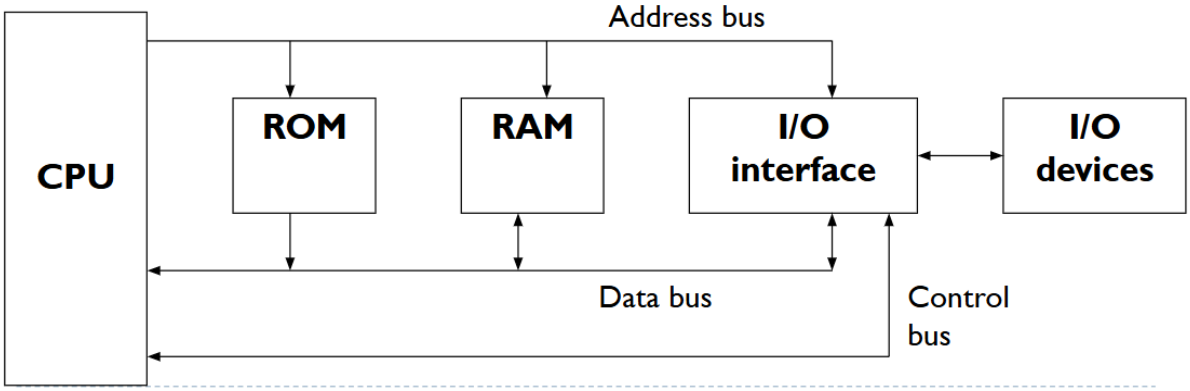
MOV AX, 7 ; Step 1: Store 7 in AX

ADD AX, 3 ; Step 2: Add 3 → AX = 10

SUB AX, 5 ; Step 3: Subtract 5 → AX = 5

Overview of Microcomputer (Structure and Operation)

1.Draw the block diagram of a Microcomputer.



2.Differentiate between RAM and ROM in terms of functionality and volatility.

Feature	RAM (Random Access Memory)	ROM (Read-Only Memory)
Functionality	Temporary storage for data and instructions that the CPU is currently using; read/write memory.	Permanent storage for firmware or fixed programs; mainly read-only.
Volatility	Volatile – loses data when power is turned off.	Non-volatile – retains data even when power is off.

3.Define the three types of system buses and their roles.

Three types of system buses and their roles:

1. **Data Bus**
    - **Role:** Transfers actual data between the CPU, memory, and I/O devices.
    - **Direction:** Bi-directional (data can flow both ways).
2. **Address Bus**
    - **Role:** Carries the address of the memory location or I/O device where data is to be read from or written to.
    - **Direction:** Unidirectional (from CPU to memory/I/O).
3. **Control Bus**
    - **Role:** Carries control signals to coordinate and manage operations (e.g., Read, Write, Clock, Interrupts).
    - **Direction:** Bi-directional or unidirectional depending on signal type.

4.Which part of the CPU would most likely handle calculation?

The **Arithmetic Logic Unit (ALU)** of the CPU handles all **calculations and logical operations**.

- Performs **arithmetic operations** like addition, subtraction, multiplication, division.
- Performs **logical operations** like AND, OR, NOT, XOR, comparisons.

It works closely with **registers** to store operands and results temporarily.

5.What is the primary role of the address bus in a CPU?

The **primary role of the address bus** is to **carry the address of a memory location or I/O device** that the CPU wants to read from or write to.

- It tells memory or I/O where to send or store data.
- **Direction:** Unidirectional (CPU → memory/I/O).

6.Which CPU component holds temporary data that is frequently accessed?

The **registers** in the CPU hold temporary data that is frequently accessed.

- They provide **very fast storage** compared to RAM.
- Used for **operands, intermediate results, and instruction addresses** during processing.

7.Describe the primary functions of the CPU and briefly explain them.

Primary functions of the CPU:

1. **Fetch**
    - Retrieves instructions or data from memory.
    - The **program counter (PC)** points to the next instruction to fetch.
2. **Decode**
    - Interprets the fetched instruction to determine the required operation.
    - The **control unit (CU)** generates signals to direct other CPU components.

3. **Execute**
    - Performs the operation specified by the instruction.
    - The **ALU** handles arithmetic and logical operations, while registers store intermediate results.

These steps together form the **instruction cycle** of the CPU.

8.Why is the address bus unidirectional while the data bus is bidirectional?

- **Address Bus (Unidirectional):**
    - It only carries the address from the CPU to memory or I/O devices.
    - The CPU specifies **where** to read/write, but memory/I/O **does not send addresses back**, so one-way direction is sufficient.
- **Data Bus (Bidirectional):**
    - It **transfers actual data** between the CPU, memory, and I/O devices.
    - Data can **flow both ways**: CPU can read from memory/I/O or write to memory/I/O, so bidirectional flow is necessary.

9.If a computer system needs to store startup instructions permanently, should it use RAM or ROM? Explain.

It should use **ROM (Read-Only Memory)**.

Explanation:

- Startup instructions, like the BIOS or firmware, need to be **permanently stored** and **retained even when power is off**.
- **ROM is non-volatile**, meaning it keeps data without power, unlike RAM, which is volatile and loses all data when the system is turned off.

10.What is the role of the program counter in instruction fetching?

The **program counter (PC)** holds the **address of the next instruction** to be fetched from memory.

- During **instruction fetching**, the CPU reads the instruction from the memory location pointed to by the PC.
- After fetching, the PC is **incremented** to point to the following instruction, ensuring sequential execution.

11.Differentiate between the Memory Address Register (MAR) and the Memory Data Register (MDR).

Feature	Memory Address Register (MAR)	Memory Data Register (MDR)
Function	Holds the **address** of the memory location to read from or write to.	Holds the **actual data** being read from or written to memory.
Direction	Unidirectional (CPU → Memory)	Bi-directional (CPU ↔ Memory)
Purpose	Specifies **where** in memory the operation will occur.	Stores the **data** involved in the memory operation.

12.Suppose a microprocessor has a 16-bit address bus. How many memory locations can it address?

A **16-bit address bus** can address (2^16) memory locations.
2^{16} = 65,536 locations

So, the microprocessor can access **65,536 memory locations** (or **64 KB** of memory, assuming each location stores 1 byte).

13.A CPU is executing a program, but you notice that some instructions take longer to execute than others. Which part of the fetch-decode-execute cycle could be causing this delay?

The **Execute** phase of the **fetch-decode-execute cycle** is most likely causing the delay.

- Some instructions, like **multiplication, division, or memory access**, require more **clock cycles** to complete.
- The **ALU** or other execution units may take longer for complex operations, while simple instructions (like addition or register moves) execute faster.

So, the variation in execution time is usually due to the **complexity of the operation in the Execute step**.

14.Imagine a data-intensive application that frequently transfers large amounts of data. Would increasing the size of the data bus improve performance? Why or why not?

Yes, increasing the size of the data bus would improve performance.

Reason:

- The **data bus width** determines how many bits can be transferred between the CPU, memory, and I/O in **one operation**.
- A wider data bus (e.g., 32-bit instead of 16-bit) allows **more data to move per cycle**, reducing the number of memory accesses required.
- This **reduces transfer time** for large amounts of data, improving overall system performance for data-intensive applications.

So, wider data buses **increase data throughput**, making the system faster for heavy data transfer tasks.

15.If a microprocessor system frequently communicates with external hardware devices, what role does the control bus play in ensuring smooth operation?

The **control bus** manages and coordinates communication between the microprocessor and external hardware devices.

Roles in ensuring smooth operation:

- **Read/Write Signals:** Indicates whether the CPU wants to **read data from** or **write data to** a device.
- **Timing and Synchronization:** Ensures data is transferred at the correct time, preventing conflicts.
- **Interrupts:** Allows devices to signal the CPU when they need attention, enabling responsive operation.
- **Control Signals:** Manages other operations like memory enable, I/O enable, and clocking.

So, the control bus **synchronizes actions and communicates intentions**, ensuring reliable and orderly interaction between the CPU and hardware.

16.BIOS is a special program that orchestrates loading the computer's operating system. Should it be stored in ROM or RAM ?

BIOS should be stored in ROM, not RAM.

Reason:

- **BIOS (Basic Input/Output System)** is needed **immediately when the computer is powered on**.
- At power-on, **RAM is empty** - it loses all data when the power is off.
- **ROM (Read-Only Memory)** is **non-volatile**, meaning it keeps its contents even when there is no power.

8086 Memory Address Space Partition

1.Deduce the size of the address bus if the total size of the memory is 4 MB.

log2(4000000)= 21.92=22

2.Determine the total memory size if the address bus size is 21 bits.

2^21 * 1B

3.Explain the difference between logical and physical addresses.

Difference Between Logical and Physical Addresses:

Logical Address

- Formed by the segmentation mechanism of the 8086.
- Consists of Segment:Offset pair.
- Generated by the CPU while executing instructions.
- Does not directly correspond to an actual memory location.
- Must be translated into a physical address before accessing memory.

Physical Address

- The 20-bit real address is actually used to access memory.
- Produced by the 8086 address adder using:
  Physical Address = (Segment × 10h) + Offset
- Points to the actual location in RAM.
- Visible to hardware but not directly visible to the programmer.

Key Point

- Logical address = Segment:Offset
  Physical address = 20-bit real address after translation
- Logical → Physical conversion is done automatically by the 8086 segmentation hardware.

4.What are segment registers, and what role do they play in memory addressing?

The 8086 has four 16-bit segment registers:

- CS – Code Segment: holds segment number of the code segment.
- DS – Data Segment: holds segment number of the data segment.
- SS – Stack Segment: holds segment number of the stack segment and is used when a sub-program executes.
- ES – Extra Segment: holds alternate segment number of the data segment.

Role in Memory Addressing

- Segment registers define which 64 KB block of memory the CPU will use for instructions, data, and stack.
- Used to form logical addresses (Segment : Offset).
- CPU converts them to 20-bit physical addresses using:
  Physical = Segment × 10h + Offset
- They allow 8086 to access 1 MB memory using 16-bit registers.
- Separate segments help organize code, data, and stack efficiently.

5.Define overlapping and non-overlapping segments with examples.

Overlapping Segments:

- Two or more segments whose physical address ranges overlap in memory.
- This happens because segment base addresses differ by less than 16 bytes, causing their 64 KB ranges to share memory space.
- Used for memory sharing between code, data, or stack.

Example

- CS = 2000h, offset range = 0000h–FFFFh
- DS = 2001h, offset range = 0000h–FFFFh

Physical starts:

- CS starts at 2000h × 10h = 20000h
- DS starts at 2001h × 10h = 20010h

These differ by only 10h bytes, therefore overlap.

Non-Overlapping Segments:

- Segments whose physical ranges do NOT overlap.
- Their segment base addresses must differ by at least 1000h (4096 decimal), because:
  1000h × 10h = 10000h (64 KB)
- Completely separate memory areas for code, data, and stack.

Example

- CS = 1000h → physical start = 10000h
- DS = 2000h → physical start = 20000h

The difference is 10000h, which is exactly 64 KB, so no overlap.

Key Difference

- Overlapping segments share memory space; non-overlapping segments occupy completely separate 64 KB memory blocks.

6.What are the advantages of memory segmentation in 8086?

- Allows the CPU to access 1 MB memory with 16-bit registers.
- Organizes memory into code, data, and stack segments.
- Supports relocation of programs in memory.
- Helps in modular programming and better program structure.
- Provides protection and separation between code, data, and stack.

7.Describe the purpose and operation of the segment registers in the 8086 architecture. How are they different from general-purpose registers?

Purpose of Segment Registers (8086):

- Hold base addresses of 64 KB memory segments.
- Used to form logical addresses (Segment : Offset).
- Help generate 20-bit physical addresses.
- Organize memory into code, data, stack, extra segments.
- Allow access to 1 MB memory using 16-bit registers.

Operation of Segment Registers:

- CS: Base of code segment; used with IP for instruction fetch.
- DS: Base of data segment; used for memory data access.
- SS: Base of stack segment; used with SP/BP.
- ES: Extra segment; mainly used for string operations.
- Segment register + Offset → Physical address.

Difference from General-Purpose Registers:

- Segment registers control memory addressing, not data processing.
- Cannot be freely used for arithmetic or logic.
- General-purpose registers (AX, BX, CX, DX) store operands and results.
- Segment registers have fixed roles, general registers are flexible.

Microprocessor Internal Architecture

1.Illustrate using a block diagram, the Internal Architecture of Intel 8086 and label the individual components.

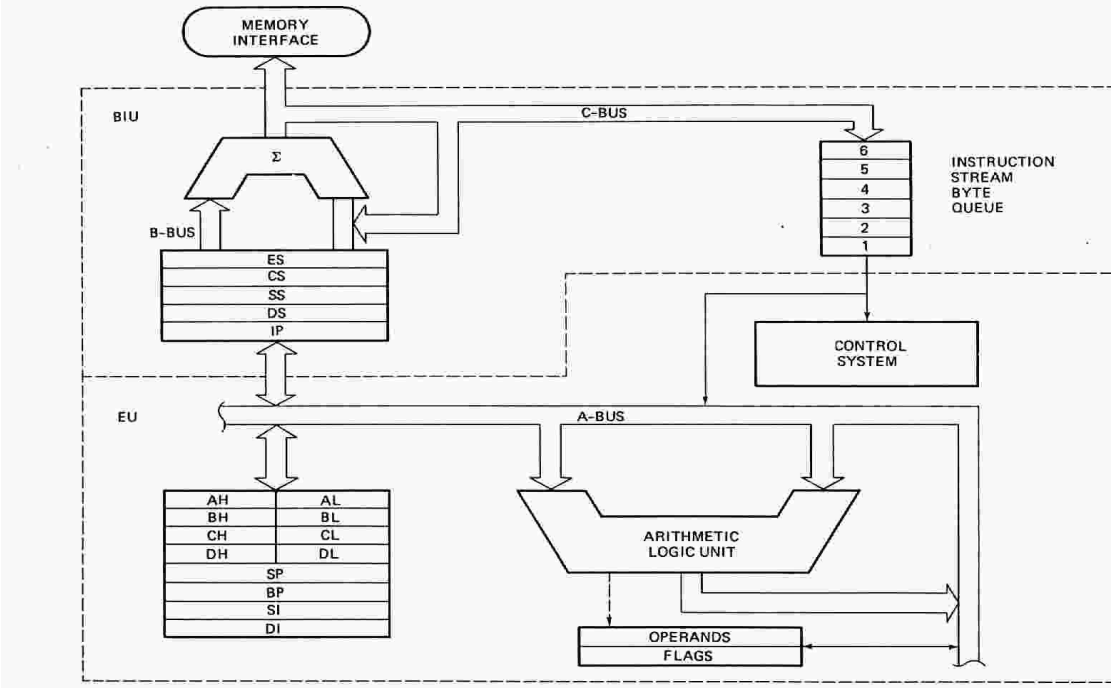


FIGURE 2-7 8086 internal block diagram. (*Intel Corp.*)

2.What are the two functional units of the 8086 microprocessor? Explain their roles.

1. Bus Interface Unit (BIU)

- It sends the address of the memory or I/O.
- It fetches instruction from memory.
- It reads data from port/memory.
- It writes data into port/memory.
- It supports instruction queuing.
- It provides the address relocation facility.

2. Execution Unit (EU)

- Fetches instructions from the Queue in BIU, decodes and executes arithmetic and logic operations using the ALU.
- Sends control signals for internal data transfer operations within the microprocessor.
- Sends request signals to the BIU to access the external module.
- It operates with respect to T-states (clock cycles) and not machine cycles.

Summary:

- BIU → fetches instructions & data, handles memory/I/O access.
- EU → executes instructions and performs operations.

3.What is the instruction queue, and how does it improve execution speed?

- Definition: A 6-byte FIFO buffer in the Bus Interface Unit (BIU) that prefetches instructions from memory.
- Purpose: Ensures the Execution Unit (EU) rarely waits for instructions.
- Improves Speed: Enables pipelining by overlapping instruction fetch (BIU) with instruction execution (EU).
- Efficiency: Reduces idle time of the EU, especially during sequential instruction execution.
- Key Feature: Basic mechanism for parallelism in 8086 architecture.

4.Briefly explain in which scenario the instruction queue fails.

When the Instruction Queue Fails

- Branching Instructions:
    - Instructions like JMP, CALL, RET, or conditional jumps change the flow of execution.
    - Prefetched instructions in the queue after the branch may become invalid.
- Queue Flush Required:
    - The BIU clears (flushes) the queue and refills it from the new address.

Key Point:

- The instruction queue fails to speed up execution when the program flow is non-sequential due to jumps, calls, or interrupts.

5.A program contains a JUMP instruction. How does this affect the instruction queue?

Effect of JUMP on Instruction Queue:

- Queue Flush: Prefetched instructions after the jump become invalid.
- Refill from New Address: BIU fetches instructions starting at the jump target.
- Temporary EU Delay: Execution Unit may wait until queue is refilled.
- Impact on Speed: Instruction queue benefits apply only to sequential execution; jumps temporarily reduce speed.

6.Explain whether the BIU will begin prefetching instructions if 5 bytes of the 6-byte prefetch queue are already filled and 1 byte is still empty.

- Partial Queue: If the 6-byte prefetch queue is not full (e.g., 5 bytes filled, 1 byte empty), the BIU continues prefetching the next instruction byte.
- Goal: Keep the queue as full as possible to avoid EU delays.
- Prefetching: Happens in parallel with EU execution of current instructions.
- Key Point: The BIU always tries to maintain a full instruction queue unless a branch, jump, or interrupt occurs.

7.If an instruction requires accessing data stored in memory, which unit of the 8086 microprocessor is responsible for fetching it?

Memory Access –

- Responsible Unit: Bus Interface Unit (BIU)
- Reason:
    - The BIU handles all memory and I/O operations.
    - It fetches data from memory or I/O ports and supplies it to the Execution Unit (EU) for processing.

Key Point:

- EU executes instructions, but BIU fetches the data required from memory.

8.How many instructions can the instruction queue hold in the 8086 microprocessor?

Instruction Queue Size – 8086:

- Capacity: 6 bytes
- Unit: Located in the Bus Interface Unit (BIU)
- Purpose: Prefetches instructions to enable pipelined execution and reduce EU idle time.

9.Which functional unit does the following register belong to?

a. Segment Register
b. Instruction Pointer
c. Index Register
d. General Purpose Register
e. Instruction Queue
f. Flags

8086 Registers and Their Functional Units:

- Segment Register: Bus Interface Unit (BIU)
- Instruction Pointer (IP): Execution Unit (EU)
- Index Registers (SI, DI): Execution Unit (EU)
- General Purpose Registers (AX, BX, CX, DX): Execution Unit (EU)
- Instruction Queue: Bus Interface Unit (BIU)
- Flags Register: Execution Unit (EU)

Flag

1.How many bits are used in the 8086 flag register, and how are they categorized?

8086 Flag Register – Size:

- The 8086 flag register is 16 bits wide.

Flag Categories:

1. Status Flags (reflect results of arithmetic/logic operations)

- CF – Carry Flag (carry/borrow out of MSB)
- PF – Parity Flag (1 if result has even number of 1s)
- AF – Auxiliary Carry Flag
- ZF – Zero Flag (1 if result = 0)
- SF – Sign Flag (sign of result; MSB)
- OF – Overflow Flag (signed overflow)

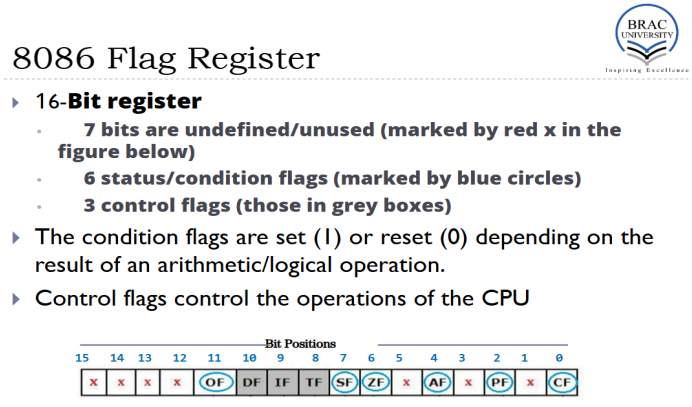
2. Control Flags (control CPU operations)

- TF – Trap Flag (single-step debugging)
- IF – Interrupt Flag (1 = enable maskable interrupts)
- DF – Direction Flag (string instruction direction: 0=increment, 1=decrement)

3. Unused/Undefined Flags

Total Active Flags:

- Out of 16 bits, 9 flags are active.
- Remaining bits are reserved/not used.



2.Explain the difference between status flags and control flags in the flag register.

Difference Between Status Flags and Control Flags (8086 Flag Register):

1. Status Flags

- Purpose: Show the result of arithmetic and logical operations.
- Effect: CPU sets or clears them automatically after ALU operations.
- Usage: Used by programs and conditional instructions to make decisions.
- Examples:
    - CF (Carry Flag)
    - PF (Parity Flag)
    - AF (Auxiliary Carry Flag)
    - ZF (Zero Flag)
    - SF (Sign Flag)
    - OF (Overflow Flag)

2. Control Flags

- Purpose: Control how the CPU operates, especially how certain instructions behave.
- Effect: Programmers can set/clear them using instructions (e.g., STI, CLD).
- Usage: Change processor modes such as interrupt handling and string direction.
- Examples:
    - TF (Trap Flag) – enables single-step execution
    - IF (Interrupt Flag) – enables/disables maskable interrupts
    - DF (Direction Flag) – sets auto-increment/decrement for string operations
- **Status flags = reflect ALU results.**
- **Control flags = control the CPU's operation mode.**

3.What does the Zero Flag (ZF) indicate, and how is it useful in programming?

Zero Flag (ZF) in 8086:

What ZF Indicates

- Zero Flag = 1 → The result of an arithmetic or logical operation is zero.
- Zero Flag = 0 → The result is non-zero.

The flag is automatically updated by the ALU after every operation.

Why ZF Is Useful in Programming:

- Used for conditional jumps, comparisons, loop control, and string matching.

4.What is the function of the Direction Flag (DF) in string operations?

Function of the Direction Flag (DF) in String Operations (8086):

- The Direction Flag (DF) controls the direction in which string instructions process memory.
- It determines whether the index registers (SI and DI) increment or decrement after each string operation.

How DF Works:

- DF = 0 → Forward direction
    - SI and DI increment
    - String operations move from lower memory to higher memory
    - (This is the most common mode)
- DF = 1 → Backward direction
    - SI and DI decrement
    - String operations move from higher memory to lower memory

Instructions to control DF:

- CLD → Clear DF (DF = 0, forward)
- STD → Set DF (DF = 1, backward)

DF decides whether string operations process memory forward (increment) or backward (decrement).

5.What is the Trap Flag (TF) used for in debugging?

Trap Flag (TF) – Use in Debugging (8086).

What TF Does:

- The Trap Flag (TF) enables single-step execution in the 8086.
- When TF = 1, the processor generates a debug interrupt (INT 1) after every instruction.

Why It's Useful:

- Allows the programmer to trace program execution one instruction at a time.
- Helps in:
    - Finding logical errors
    - Inspecting register/memory changes step-by-step
    - Monitoring program flow precisely

When TF = 0:

- Normal execution (no single stepping).

TF enables single-step debugging by generating an interrupt after each instruction, helping programmers trace code execution in detail.

6.A program needs to process a string of characters from the end to the beginning. How should the Direction Flag (DF) be set?

To process a string from the end to the beginning, the string instructions must move backward in memory.

Correct Setting of the Direction Flag (DF):

- DF = 1 → Process backward
- Set using: STD (Set Direction Flag)

Set DF = 1 (using STD) so that SI and DI decrement and the string is processed from end to beginning.

7.If an external hardware device sends an interrupt signal to the CPU, but the Interrupt Enable Flag (IF) is cleared (0), will the CPU respond to the interrupt? Explain why.

Answer: No, the CPU will not respond to the interrupt.

Explanation:

- The Interrupt Enable Flag (IF) controls maskable hardware interrupts.
- When IF = 0, maskable interrupts (INTR pin) are disabled.
- Even if an external device sends an interrupt request, the CPU ignores it because interrupts are not allowed at that moment.

Why?

- The CPU checks IF before accepting any maskable hardware interrupt.
- IF = 0 → Interrupts blocked
- IF = 1 → Interrupts allowed

If IF = 0, the CPU ignores maskable external interrupts because they are disabled.

8.When adding two 8-bit numbers, the sum causes a carry from the lower nibble to the upper nibble but not from the most significant bit. Which flag(s) will be set?

Flags Set When Carry Occurs from Lower Nibble Only.

If two 8-bit numbers are added and:

- There is a carry from bit 4 → bit 5 (lower nibble → upper nibble)
- No carry from the most significant bit (bit 7)

The flag that will be set:

- AF – Auxiliary Carry Flag (because it tracks carry from lower nibble to upper nibble)

The flag NOT set:

- CF – Carry Flag (because there is *no* carry out of bit 8)

Only the Auxiliary Carry Flag (AF) will be set.

9.A programmer enables single-step debugging on the 8086. Which flag is responsible for this, and how does it function?

Answer: Trap Flag (TF)

Function:

- TF = 1 → Enables single-step execution.
- After every instruction, the CPU generates a debug interrupt (INT 1).
- This allows the programmer to trace program execution one instruction at a time.

The Trap Flag (TF) controls single-step debugging by causing the CPU to interrupt after each instruction.