MICROPROCESSOR

Lecture 4
Microprocessor System

Dr Mohammed Fadhil
Email: mohammed.fadhil1@uomus.edu.iq

Microprocessor System

- A microprocessor system is a complete setup that allows the processor to receive data, process it, and produce output.
- It includes:
  - Microprocessor (CPU) → brain of the system
  - Memory (RAM, ROM) → stores data and instructions
  - Input Devices → keyboard, sensors, etc.
  - Output Devices → display, printer, etc.
  - System Bus → connects all parts together (Data, Address, Control buses)
- Example:
  - A computer, calculator, or embedded controller in a washing machine — all are microprocessor-based systems.

Need for Memory Segmentation

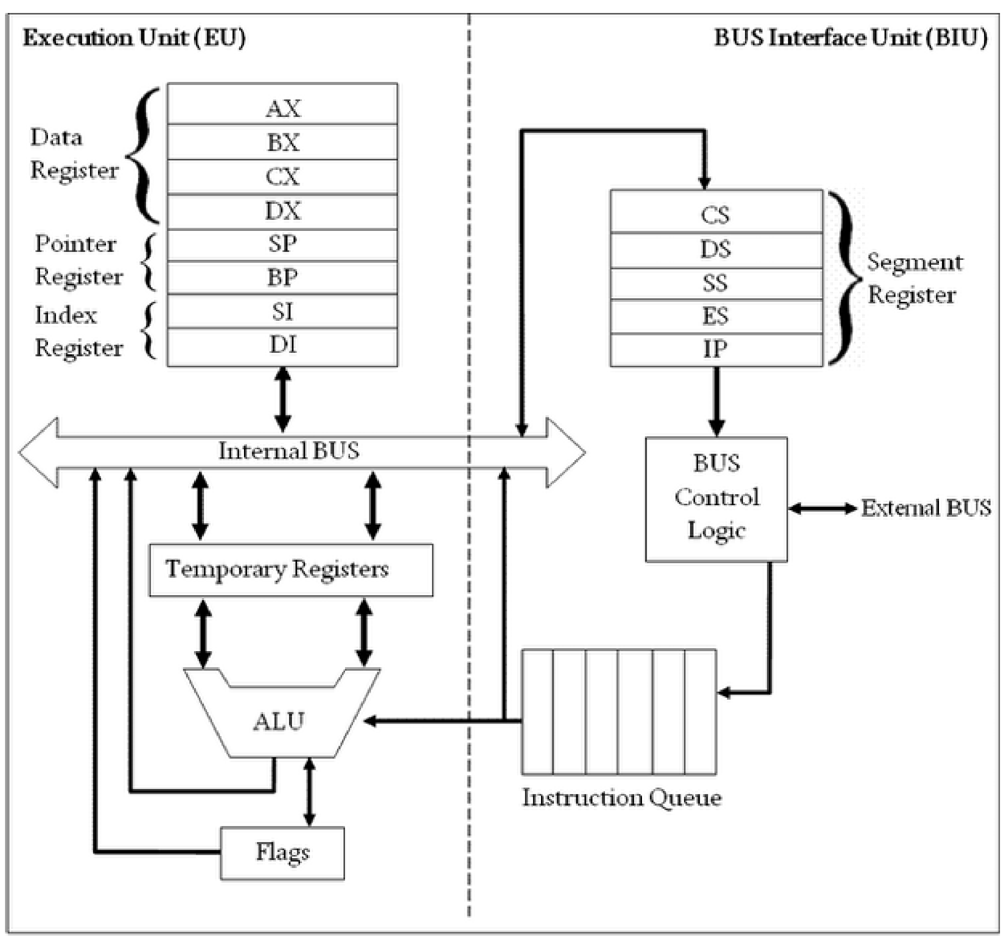
- In the 8086 microprocessor, memory segmentation was introduced to efficiently use the large memory (1 MB) and to simplify memory management.
- **Main Reasons for Segmentation:**
  - The 8086 has 16-bit registers, but it can access 1 MB (20-bit addresses) — segmentation makes this possible.
  - It helps organize memory into logical parts (code, data, stack, extra).
  - Makes programs modular and easier to manage.
  - Improves execution speed by allowing parallel processing between BIU and EU.
- **Example**:
  - If CS = 2000h and IP = 0000h,
  - the physical address = (CS × 10h) + IP = 20000h.
- 💡 **Each segment register** points to a specific part of memory, making addressing faster and organized.

Microprocessor Architecture (8086)

- The **8086 Microprocessor** has two main internal units:
  - **Bus Interface Unit (BIU)**
  - **Execution Unit (EU)**
- Both work **simultaneously** to improve performance — while BIU fetches the next instruction, EU executes the current one.

Bus Interface Unit and Execution Unit

- The internal function of 8086 processor are partitioned logically into processing units ,Bus Interface Unit(BIU) and Execution Unit (EU).
- General block diagram of 8086 processor is shown in figure.



Execution Unit

- The **Execution Unit (EU)** is the part of the processor that **receives program instructions and data** from the **Bus Interface Unit (BIU)**. It **executes the instructions** and **stores the results** in its internal **registers**.
  After execution, the results can be:
  - Stored in memory, or
  - Sent to an input/output (I/O) device (through the BIU).
- **Note:** The **EU has no direct connection** with the system buses. It **receives and sends data only through the BIU**.

Bus Interface Unit

- Bus Interface Unit : Because the EU is not connected to the system buses, the **BIU** does that job.
- It connects the EU with the **memory** and **I/O devices**.
- The BIU is responsible for:
  - Transmitting **data**, **addresses**, and **control signals** on the system buses.
  - **Fetching instructions** from memory and placing them into an **instruction queue** for the EU.
- Both EU and BIU are connected by an **internal bus** inside the processor.

Bus Interface Unit

- Main functions of BIU:
  - Fetch instructions from memory and place them in the instruction queue (FIFO).
  - Manage the address, data, and control buses.
  - Calculate physical addresses using segment registers.
  - Transfer data between memory and EU.

How the Execution Unit Works

- The EU is mainly used to **execute instructions**.
- It contains:
  - An **Arithmetic and Logic Unit (ALU)** → performs all calculations and logical operations.
  - A set of **registers** → used to temporarily store data during execution.
- There are **eight general-purpose registers**: **AX, BX, CX, DX, SI, DI, BP, SP**, plus a **FLAGS register** that shows the status of operations.

Summary Table

Register	Main Use	Simple Example
AX	Arithmetic, logic	ADD AX, BX
BX	Base address	MOV BX, 1000
CX	Counter in loops	LOOP 5 times
DX	Data / I/O	MUL BX
SI	Source address	Copy data from SI
DI	Destination address	Copy data to DI
BP	Access stack variables	Used in subroutines
SP	Stack pointer	Points to top of stack
FLAGS	Status of operation	Shows if result = 0, negative, etc.

How EU and BIU Work Together

- The **BIU fetches** instructions from memory and puts them into the instruction queue (FIFO — First In, First Out).
- The **EU takes** the next instruction from the queue.
- The **EU decodes and executes** it using the ALU and registers.
- If needed, results go back to the **memory or I/O** through the BIU.

Simple Examples

- Addition Example
  - Suppose we want to add two numbers, 5 + 3.
  - The **BIU** fetches this instruction and data from memory.
  - The **EU** (specifically the ALU) performs the addition and stores the result 8 in register **AX**.
- Data Movement Example
  - Move the value 10 from memory to register **BX**.
  - The **BIU** reads the data (10) from memory.
  - The **EU** receives it and stores it into **BX**.

ALU (Arithmetic & Logic Unit)

- The **ALU** is the part of the **Execution Unit (EU)** that performs all **calculations** and **logical decisions** inside the processor.
- It works based on the **instruction** given by the program.

Arithmetic Operations

- The ALU can perform different arithmetic operations, such as:
  - **Addition (ADD)** – adds two numbers.
    - Example: 5 + 3 = 8
  - **Subtraction (SUB)** – subtracts one number from another.
    - Example: 9 - 4 = 5
  - **Increment (INC)** – increases a number by 1.
    - Example: If AX = 5 → INC AX → AX = 6
  - **Decrement (DEC)** – decreases a number by 1.
    - Example: If BX = 10 → DEC BX → BX = 9
  - **Compare (CMP)** – compares two numbers to check if they are equal, greater, or smaller.
    - Example: CMP AX, BX → sets flags (Zero, Sign, etc.) based on comparison result.
  - **Convert Byte/Word** – used in special instructions when dealing with signed or larger numbers.

Logical Operations

- The ALU can also perform **logic-based operations** used in decision making and bit manipulation:
  - **AND** – checks if both bits are 1
    - Example: 1010 AND 1100 = 1000
  - **OR** – checks if any bit is 1.
    - Example: 1010 OR 1100 = 1110
  - **XOR (Exclusive OR)** – checks if bits are different.
    - Example: 1010 XOR 1100 = 0110
  - **Shift / Rotate** – moves bits left or right.
    - Example:
      - SHL (Shift Left): 0001 → 0010 (multiplies by 2)
      - SHR (Shift Right): 0100 → 0010 (divides by 2)
  - **TEST** – similar to AND, but does not change data — only updates the flags.
    - Example: checks if a number has certain bits set to 1.

Index Registers in the Microprocessor

1. **SP (Stack Pointer)**
   - **Meaning:** SP stands for *Stack Pointer*.
   - **Function:** It points to the **top of the stack** — a special area in memory used for temporary storage (like function return addresses, local variables, etc.).
   - It works together with the **SS (Stack Segment)** register to access the correct stack memory.

- **Example:**
  - If SP = 2000h and SS = 3000h → the stack top is at memory address 30000h + 2000h = 32000h.

Index Registers in the Microprocessor

2. BP (Base Pointer)

- **Meaning:** BP stands for *Base Pointer*.
- **Function:** It is used to **access data** in the **stack segment**.
  Unlike SP, BP can also access data from **other segments** such as data segment (DS) when needed.

- **Example:**
  - When working with procedure parameters stored in the stack, BP helps the program locate those values easily.

Index Registers in the Microprocessor

3. **SI (Source Index)**
   - **Meaning:** SI stands for *Source Index*.
   - **Function:** It points to **memory locations** in the **Data Segment (DS)**.
     It is usually used in **string and array operations** to identify the **source** of data to be copied or compared.

- **Example:**
  - If DS = 2000h and SI = 0500h, the data is located at memory address 20000h + 0500h = 20500h.
    When SI increases by 1, it moves to the next byte (next memory location).

Index Registers in the Microprocessor

4. **DI (Destination Index)**
   - **Meaning:** DI stands for *Destination Index*.
   - **Function:** It performs the same role as SI but points to the **destination** memory location.
     Used in **string operations** where data is moved from source (SI) → destination (DI).

- **Example:**
  - Instruction like MOVSB (Move String Byte) moves one byte from address in SI to address in DI, then both increase automatically to point to the next byte.

Segment Group

- The 8086 divides its 1 MB memory into **segments** (each of 64 KB).
- Each segment is pointed to by a **segment register**.
- It also contains 1 pointer register IP. IP contains the address of the next instruction to execute by the EU.

Segment	Register	Function
Code Segment (CS)	CS	Stores program instructions
Data Segment (DS)	DS	Stores variables and constants
Stack Segment (SS)	SS	Stores temporary data, function calls, return addresses
Extra Segment (ES)	ES	Used for string and extra data operations

DATA Group Registers

- These registers are part of the **Execution Unit (EU)**.
- They are used to **store data temporarily** during execution.

Register	Name	Function / Example
AX	Accumulator	Used for arithmetic, logic, and data transfer. Example: ADD AX, BX
BX	Base	Often used to hold the base address of data in memory.
CX	Counter	Used for loop and shift operations. Example: LOOP START
DX	Data	Used in multiplication, division, or I/O operations.

THANK YOU ☺