Introduction to Computer Systems

Hardware Engineer – a professional who designs, develops, tests, and produces **computer systems** and **various physical components** related to computer systems. They **work on the physical components** of computer systems, ensuring that they function properly and are compatible with other components.

Software Engineer - a professional who designs, develops, tests, and maintains **computer software**. They work on a variety of software projects, from small web applications to large enterprise software systems. Software engineers **use a variety of programming languages** and **tools** to create software that is efficient, reliable, and user-friendly.

Computer Organization

This is the study of the way in which the **hardware components** of a computer system are **arranged and interconnected.**

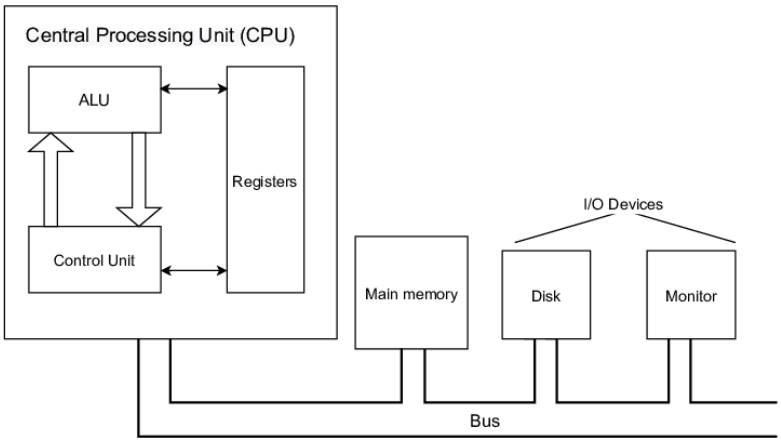

Figure 1: Computer Organization

The **main aspects** of computer organization include:

- Central Processing Unit (CPU): Bain of the computer responsible for executing instructions and performing calculations.

- Memory: Computers use different types of memory to store data and instructions. The primary memory, like RAM (Random Access Memory), stores data that the CPU is currently working on. The secondary memory, like hard drives or solid-state drives, provides long-term storage for programs and data even when the power is off.

- Input/Output (I/O) Devices: These devices facilitate communication between the computer and the external world. (Ex: keyboards, mice, monitors, printers, and network cards.)

- Bus System: a communication pathway that allows data and instructions to flow between the CPU, memory, and I/O devices.
  Buses can be categorized as follows.
  - Data bus (for transferring data)
  - Address bus (for specifying memory locations)
  - Control bus (for managing data flow)

The CPU is typically made up of the following components:

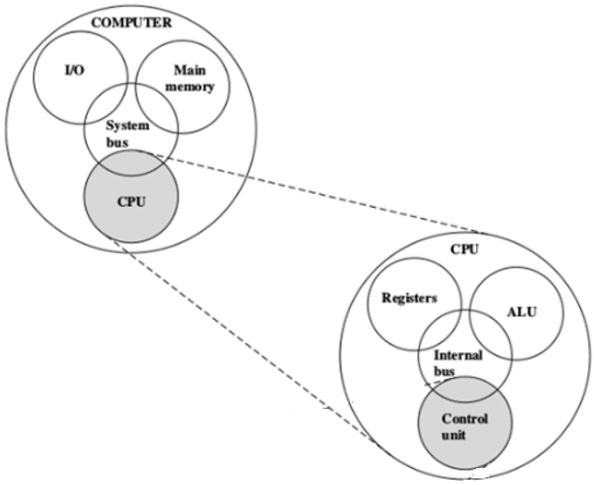

Figure 2: Elements of the CPU

- Arithmetic logic unit (ALU): The ALU performs the arithmetic and logical operations of a computer program. It can add, subtract, multiply, and divide numbers, and it can perform logical operations such as AND, OR, and NOT.
- Control unit: The control unit tells the ALU what to do and when to do it. It also controls the flow of data between the CPU and the rest of the computer system.
- Registers: Registers are small, high-speed memory units that are used to store data and instructions that are currently being processed by the CPU.

The CPU works by fetching an instruction from memory, decoding the instruction, and then executing the instruction. The instruction may tell the CPU to perform an arithmetic or logical operation, to load or store data, or to control the flow of the program.

8086 Architecture

The **8086** is **a 16-bit microprocessor** introduced by Intel in 1978 and is one of the early members of the x86 family of processors.

The **8086 architecture** is a complex instruction set computer (CISC) architecture. This means that the 8086 has a large number of instructions, many of which can perform multiple operations in a single instruction.

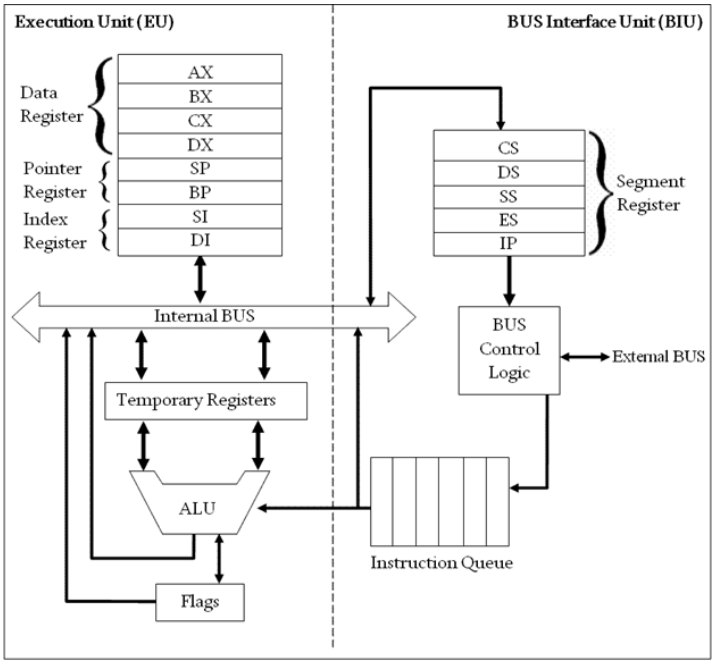

Figure 3: 8086 Architecture

Registers: a high-speed storage area within the CPU. All data must be represented in a register before it can be processed.

Ex: if two numbers are to be multiplied, **both numbers or their locations in the memory must be in registers**, and **the result** is also placed in a register.

The 8086 CPU has a set of registers used for various purposes:

- **General-Purpose Registers**: These include AX, BX, CX, DX, SI, DI, BP, and SP. They can be used for arithmetic operations, data manipulation, and memory addressing.
- **Segment Registers**: These include CS (Code Segment), DS (Data Segment), SS (Stack Segment), and ES (Extra Segment). They are used in memory addressing and to determine the starting addresses of different segments in memory.
- **Instruction Pointer (IP)**: This register holds the offset address of the next instruction to be fetched from the Code Segment (CS).
- **Flags Register**: The flags register contains individual flags that indicate the outcome of arithmetic and logical operations, such as carry, zero, sign, overflow, etc.

Instruction cycle

- Instruction register (IR):- holds the instructions that are currently being executed.
- Program counter (PC):- a specialized register that contains the memory address of the next instruction to be fetched and executed.

The instruction cycle is a fundamental process performed by a computer's Central Processing Unit (CPU) to execute instructions. It is a **continuous and repetitive** process that allows the

CPU to execute a series of instructions, **one after the other**, until the program's execution is complete.

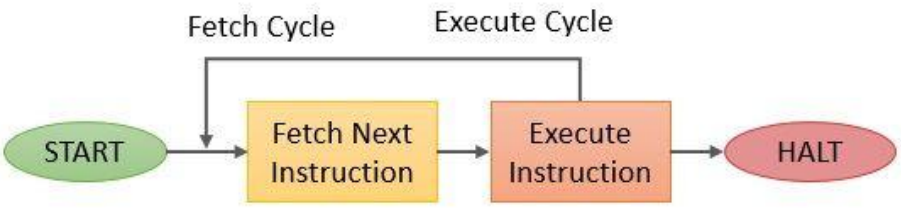

Figure 4: Instruction Cycle

Stages of instruction cycle;

Fetch: CPU fetches the next instruction from the memory location pointed to by the Instruction Pointer (IP) or Program Counter (PC).

Decode: Once the instruction is fetched, the CPU decodes the **binary representation of the instruction to determine** which operation needs to be performed and which operands are involved. The control unit interprets the instruction's opcode and identifies the necessary data paths and functional units required for execution.

Execute: In the execute stage, the CPU carries out the actual operation (ex: arithmetic operations, logical operations, data movement (e.g., load, store), etc…) specified by the instruction.

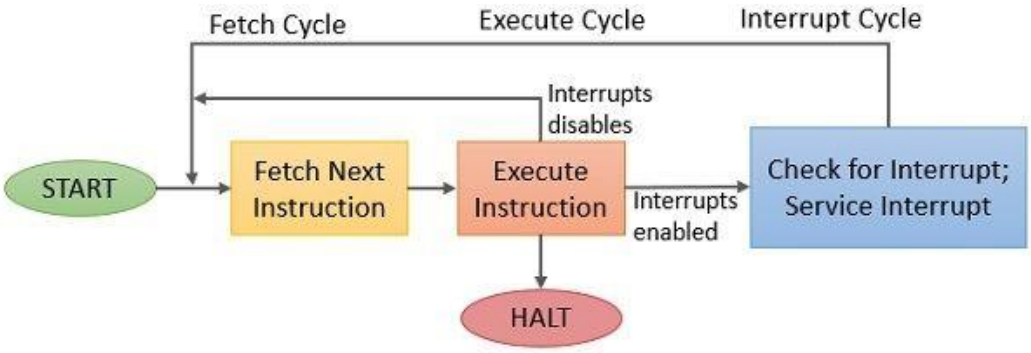

Figure 5: Instruction Cycle with Interrupts

- Normal execution of a program may be **temporarily interrupted**) if some devices require urgent servicing, to do this one device raises an Interrupt signal.
- An interrupt is a request signal from an I/O device for service by the processor.
- The processor provides the requested service by executing an appropriate interrupt service routine.
- The Diversion may change the internal stage of the processor its state must be saved
- in the memory location before interruption.
- When the interrupt-routine service is completed

- the state of the processor is restored so that the interrupted program may continue.

Instruction Set Architecture (ISA)

Instruction Set Architecture (ISA) refers to the set of instructions that a computer's Central Processing Unit (CPU) can understand and execute. It acts as an interface between the hardware and software, enabling programmers to write code that can be executed by the CPU.
Ex: **x86, ARM, MIPS**

The ISA defines the operations that the CPU can perform, the format of instructions, the number and types of registers available, and the memory addressing modes supported.

Assembly language

- A **low-level programming language** that is used to communicate directly with a computer's hardware. A **machine-dependent language**, meaning that each assembly language instruction is specific to a particular type of computer processor.
- Assembly language programs are typically written using a text editor. The program is then assembled into machine code by an assembler. The machine code can then be loaded into the computer's memory and executed.
- Assembly language uses mnemonic codes (short abbreviations or symbols) to represent machine instructions. These mnemonics are more understandable to humans, making it easier for programmers to read and write code.
  Ex:
  ADD - Add two numbers.
  CMP - Compare numbers.
  IN - Input from port into AL or AX. Second operand is a port number.
  JMP - Unconditional Jump. Transfers control to another part of the program.
  JNE - Short Jump if first operand is Not Equal to second operand.
  LOAD - Load information from RAM to the CPU.
  OUT - Output information to device, e.g. monitor.
  STORE - Store information to RAM.

Here is an example of an assembly language program that adds two numbers:

```
MOV AX, 10
MOV BX, 20
ADD AX, BX
```

1. This program first loads the value 10 into the AX register
2. Then loads the value 20 into the BX register
3. Next, ADD instruction adds the values in the AX and BX registers, and stores the result in the AX register.

Instruction format

- Computer instructions are the basic components of a machine language program.
- Each instruction initiates a sequence of micro-operations that fetch operands from registers or memory, possibly perform arithmetic, logic, or shift operations, and store results in registers or memory.

Figure 6: Instruction Format

Each instruction code contains of a operation code, or opcode, which designates the overall purpose of the instruction (e.g. add, subtract, move, input, etc.).
Many instructions also contain one or more operands, which indicate where in registers or memory the data required for the operation is located.

Ex: add instruction requires two operands, and a not instruction requires one.

The control unit is responsible for decoding the opcode and operand bits in the instruction register.

CISC & RISC

Reduced Instruction Set Architecture (RISC) – The main idea behind this is to make hardware simpler by using an instruction set composed of a **few basic steps** for loading, evaluating, and storing operations just like a load command will load data, a store command will store the data.

Complex Instruction Set Architecture (CISC) – The main idea is that **a single instruction will do all** loading, evaluating, and storing operations just like a multiplication command will do stuff like loading data, evaluating, and storing it, hence it's complex.

Both approaches try to increase the CPU performance

- **RISC:** Reduce the cycles per instruction at the cost of the number of instructions per program.

- **CISC:** The CISC approach attempts to minimize the number of instructions per program but at the cost of an increase in the number of cycles per instruction.

Example – Suppose we have to add two 8-bit numbers:

- **CISC approach:** There will be a single command or instruction for this like ADD which will perform the task.
- **RISC approach:** Here programmer will write the first load command to load data in registers then it will use a suitable operator and then it will store the result in the desired location.

So, add operation is divided into parts i.e. load, operate, store due to which RISC programs are longer and require more memory to get stored but require fewer transistors due to less complex command.

RISC	CISC
Focus on software	Focus on hardware
Uses only Hardwired control unit	Uses both hardwired and microprogrammed control unit
Fixed sized instructions	Variable sized instructions
Can perform only Register to Register Arithmetic operations	Can perform REG to REG or REG to MEM or MEM to MEM
Requires more number of registers	Requires less number of registers
Code size is large	Code size is small
An instruction executed in a single clock cycle	Instruction takes more than one clock cycle
An instruction fit in one word.	Instructions are larger than the size of one word
Simple and limited addressing modes.	Complex and more addressing modes.
RISC is Reduced Instruction Cycle.	CISC is Complex Instruction Cycle.
The number of instructions are less as compared to CISC.	The number of instructions are more as compared to RISC.
It consumes the low power.	It consumes more/high power.
RISC required more RAM.	CISC required less RAM.
Here, Addressing modes are less.	Here, Addressing modes are more.

Table 1: Comparison between RISC and CISC