Module 1: Introduction to Computer Architecture

An Easy Guide to Understanding Functional Blocks, ISA, and More

What are Functional Blocks?

- A computer is made up of **different parts**, and each part has a unique job to do.
- These parts work together in harmony to perform all the tasks that we ask the computer to do, like running programs, storing data, and showing results.
- Without any one of these parts, the computer wouldn’t function properly.
- Each part plays an important role in making the computer work efficiently.

Overview of Functional Blocks

Functional blocks include:

- **CPU (Central Processing Unit)**
- **Memory (Primary and Secondary)**
- **Input-Output Subsystems**
- **Control Unit**

CPU: The Brain of the Computer

- The **CPU (Central Processing Unit)** is like the **brain** of the computer.
- It handles all the important tasks like performing calculations, making decisions, and controlling the other parts of the computer.
- It receives instructions, processes them, and then tells the rest of the computer what to do next.
- Without the CPU, the computer wouldn’t be able to function. It is the central part that ensures everything works smoothly.

Parts of the CPU

- The **CPU** has two main parts, each with its own important job:

1. **ALU (Arithmetic and Logic Unit):** This part handles all the **math** tasks like addition and subtraction, as well as **logic** tasks like comparing numbers or deciding if a condition is true or false.
2. **Control Unit:** This part is responsible for **directing the flow of operations**. It tells the CPU what to do next by sending out instructions to the other parts of the computer, making sure everything runs in the correct order.

Memory Overview

- Memory is like the storage of the computer.
- It holds data and instructions that the CPU needs in order to do its tasks.
- There are two main types of memory:
- **Primary Memory (RAM):** It stores data and programs that are currently in use. It's fast but temporary, meaning it loses the data when the computer is turned off.
- **Secondary Memory (like Hard Drives or SSDs):** It stores data long-term, such as files, programs, and the operating system, even when the computer is turned off.

Primary Memory (RAM)

- **Primary Memory** is the temporary storage of the computer.
- It holds the data and programs that the **CPU** is currently using while the computer is running.
- This type of memory is **fast**, allowing the CPU to quickly access the data it needs.
- However, once the computer is **switched off**, the data stored in primary memory is **lost** because it is temporary.
- Examples of primary memory include **RAM (Random Access Memory)**.

Secondary Memory

- **Secondary Memory** is the long-term storage of the computer.
- It stores **files**, **programs**, and other types of **data** that are not actively in use.
- Unlike primary memory, the data stored in secondary memory is **permanent** and is not lost when the computer is turned off.
- This type of memory is slower than primary memory, but it provides a much larger storage space.
- **Examples of secondary memory** include:

- **Hard Disk Drives (HDDs):** Traditional storage devices that use spinning disks to read and write data.
- **Solid State Drives (SSDs):** Faster, more durable storage devices that use flash memory to store data without moving parts.

Input-Output Subsystems

- **The Input-Output Subsystem** manages how the computer communicates with external devices, allowing data to be sent into and out of the computer.
- **Input devices** are the tools that send data into the computer. These include devices like the keyboard (for typing), mouse (for clicking), and scanner (for scanning documents).
- **Output devices** are the tools that display or produce results from the computer. These include devices like the monitor (for displaying images and text), printer (for printing documents), and speakers (for playing sound).

Instruction Set Architecture (ISA)

- The Instruction Set Architecture (ISA) is a **collection of commands or instructions** that the CPU can understand and execute.
- These instructions tell the CPU what to do, like performing calculations, moving data, or controlling the flow of the program.
- The ISA acts as a bridge between software (programs) and hardware (the CPU).
- It defines how the CPU communicates with memory, processes data, and carries out tasks, ensuring that programs run smoothly on the computer.

Components of ISA

- The Instruction Set Architecture (ISA) includes several important components that help the CPU understand and execute instructions. These components are:
- **Registers:** These are small, fast **storage locations** inside the CPU that **hold data** and instructions that the CPU is currently **working on.** Registers are used for quick access and processing, allowing the CPU to operate efficiently.

Registers

- **Data Registers:** These hold the data that the CPU is currently working on.
- **Address Registers:** These hold memory addresses. They store the locations in memory where the data is stored or needs to be retrieved from.
- **Program Counter:** This register keeps track of the next instruction that the CPU needs to execute. It stores the address of the next instruction in the program, ensuring the CPU follows the correct sequence of operations.

Components of ISA

- **Instruction Execution Cycle:** This is the process the CPU follows to execute commands. It typically involves three steps:
- **Fetch:** The CPU gets the instruction from memory.
- **Decode:** The CPU decodes or interprets what the instruction means.
- **Execute:** The CPU carries out the operation specified by the instruction (e.g., adding two numbers).

Components of ISA

- **RTL (Register Transfer Language):** RTL is a way to describe **how data moves** from one register to another in the CPU.
- It helps to explain the steps that happen during the execution of instructions, showing how information is **transferred and processed** within the CPU.
- These components together help the CPU understand and carry out instructions efficiently.

Addressing Modes

- Addressing modes are different methods the CPU uses to find or access data in memory or in registers.
- These modes help the CPU understand where to look for the data it needs for a specific operation.

Here are a few common addressing modes:

1. Immediate Addressing
2. Direct Addressing
3. Indirect Addressing

Immediate Addressing

- Immediate Addressing is a type of addressing mode where the data to be used is given directly in the instruction itself, rather than pointing to a memory location.
- **Example:** ADD 5: In this instruction, the number 5 is directly included in the instruction. The CPU will add 5 to a register, without needing to look in memory for the value.

Direct Addressing

- Direct Addressing is a type of addressing mode where the instruction contains the address of the data stored in memory. The CPU uses this address to directly access the data.
- **Example:** LOAD 1000: In this instruction, 1000 is the memory address where the data is stored. The CPU will load the data from memory location 1000 into a register.

Indirect Addressing

- Indirect Addressing is a type of addressing mode where the instruction provides the address of the address where the actual data is stored.
- The CPU first uses the provided address to find another address, and then it goes to that second address to retrieve the data.
- Example: LOAD (1000): In this instruction, 1000 is a memory address that contains another address. The CPU first goes to address 1000, reads the address stored there, and then uses that second address to load the actual data from memory.

Case Study: 8085 Processor

- The 8085 processor is a simple microprocessor that was widely used in early computer systems.
- It has its own set of instructions that allow it to perform basic operations.
- **This processor is still studied today for its simplicity and educational value in understanding how processors work.** Key Features of the 8085 Processor:
- **Basic Operations:** The 8085 processor can execute simple arithmetic operations such as adding (ADD), subtracting (SUB), and moving data (MOV) between registers or memory.

Case Study: 8085 Processor

- **Instruction Set:** The 8085 has its own instruction set that includes a variety of commands. Some examples are: ADD, SUB and MOV.
- **Registers and Memory:** The 8085 has several general-purpose registers and a few special-purpose registers like the Accumulator (for calculations) and Program Counter (which keeps track of the next instruction to execute).
- **Control of Program Flow:** The 8085 processor also has control instructions such as JMP (jump to a specific address) and CALL (call a subroutine), which help control the flow of the program and enable more complex operations.
- Despite being a simple processor by today's standards, the 8085 processor played a crucial role in the development of microprocessor technology and is a great example of how processors handle basic tasks through their instruction sets.

Recap and Closing

We covered:

- Functional blocks of a computer.
- Instruction Set Architecture.
- Addressing Modes.
- Instruction Set of the 8085 Processor.