Computer Organization & Architecture

Cybersecurity Department

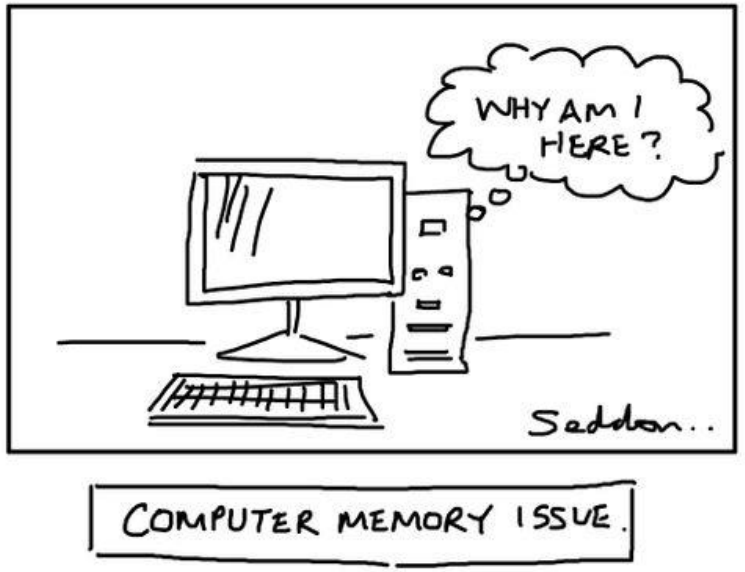
Course Code: ***CBS219***

Lecture 5: Addressing Modes - Memory Hierarchy

Halal Abdulrahman Ahmed

Lecture Outline

- What are Addressing Modes?
- Why CPUs need them
- Components of Instruction
- Addressing Mode Types
- Effective Address calculation
- Memory Hierarchy
- Purpose of Memory Hierarchy
- Types of Memory Hierarchy
- Characteristics of Memory Hierarchy



Lecture Outcomes

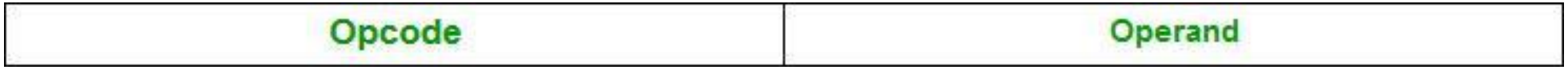
By the end of this lecture, students will be able to:

- Explain what **addressing modes** are and why CPUs use them.
- Identify and distinguish main **types of addressing modes**.
- Calculate the **effective address** in simple examples.
- Describe the purpose and structure of the **memory hierarchy**.
- Compare different **memory levels** in terms of speed, cost, and capacity.

Addressing Modes

Addressing modes are techniques used by the CPU to identify the location of the operand(s) needed for executing an instruction. They provide rules for interpreting the address field in an instruction, helping the CPU fetch operands correctly.

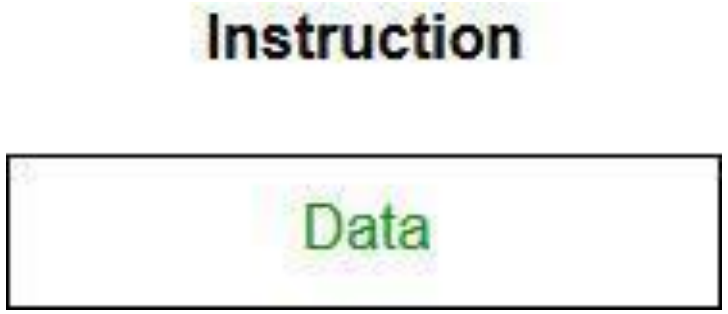
- **Opcode** – Tells the CPU what operation to perform (e.g., ADD, MOV).
- **Operands** – The data or addresses on which the operation is performed.



Addressing Modes Types

Implicit (Implied) Addressing

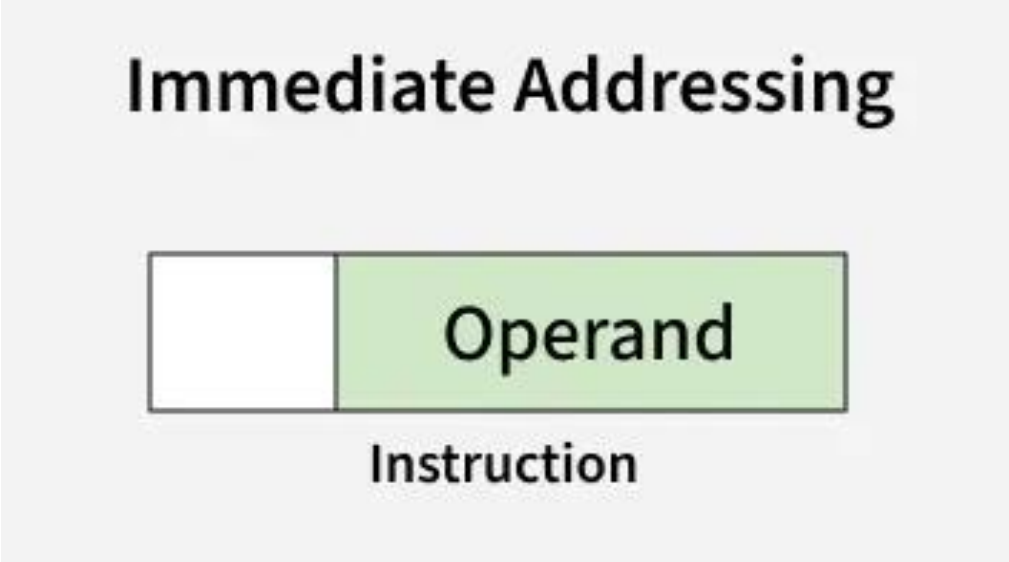
- The instruction does not mention the operand directly. The CPU knows what to use from the instruction itself, usually a special register like the accumulator or the stack.
- It is used for special instructions or control commands like CLA, PUSH, and RET, where the operand is automatically known from the instruction itself the instruction does not mention the data, the CPU already knows where the data is.
- The operand is **implied/automatic**, usually in **Accumulator** or **Stack**.



Immediate Addressing

The operand is the part of the instruction itself. It is used when the value is known while writing the program. Value is inside the instruction.

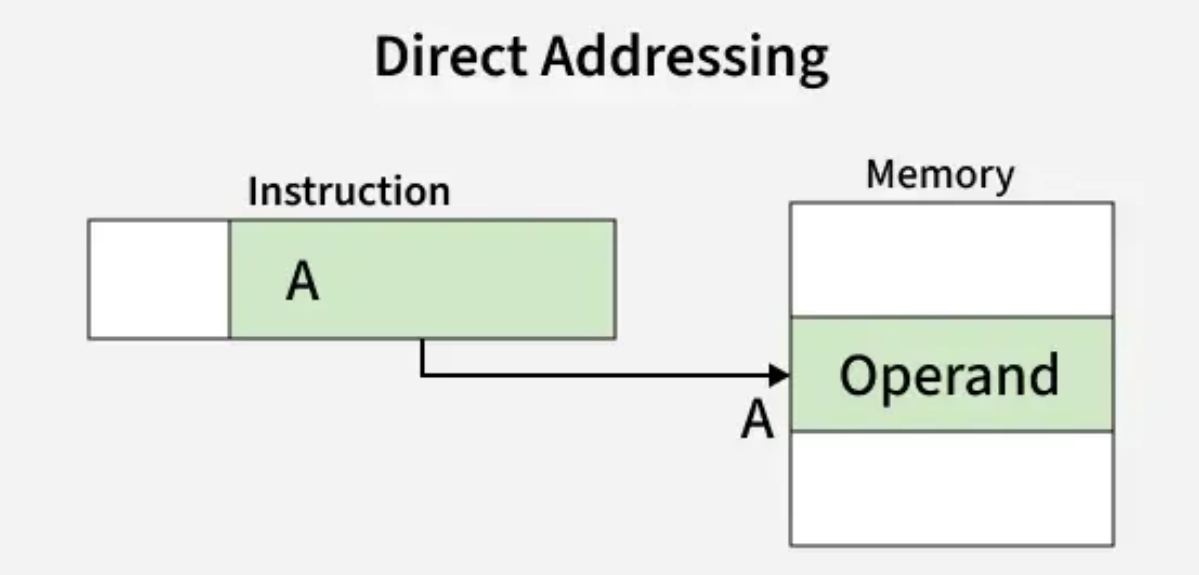
Example: MOV R1, #5 (moves the value 5 into register R1, where #5 is the immediate value).



Direct Addressing

The instruction contains the memory address of the operand. The CPU accesses the data directly from that address. Instruction tells you *exact* memory location.

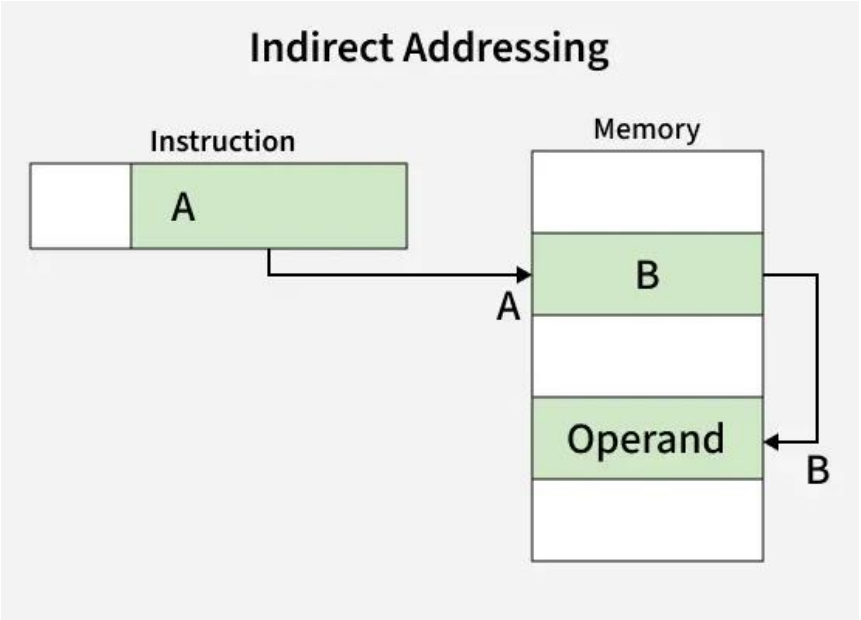
Example: LOAD R1, 1000 (loads data from memory address 1000 into register R1).



Indirect Addressing

The instruction contains the address of a register or memory location that holds the actual address of the operand. The CPU first fetches this address, then accesses the operand. Instruction gives you a location that contains another location.

Example: LOAD R1, (R2) (loads data from the memory location whose address is in register R2).



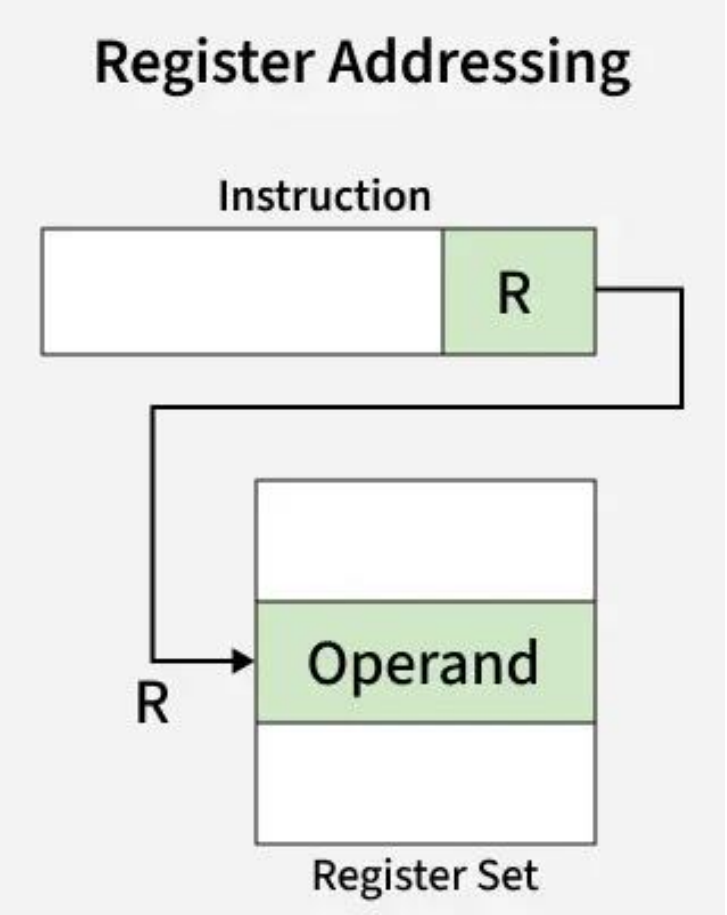
Register Addressing

The operand is located in a CPU register specified by the instruction. Data is already in a register.

Step:

- The instruction specifies a register (R).
- The CPU takes operand directly from register R.

Example: MOV A, B operates between registers A and B.

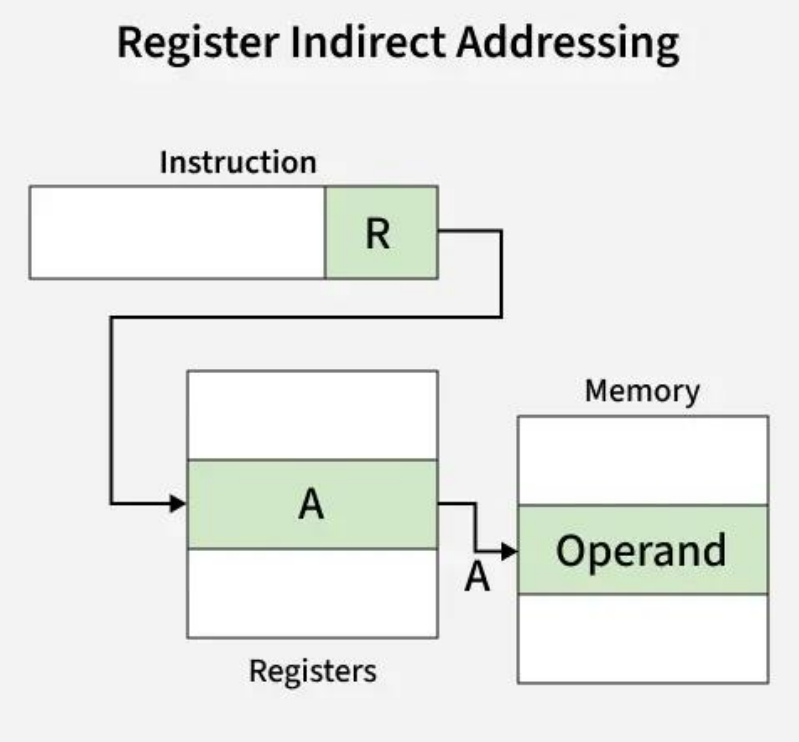


Register Indirect Addressing

The register specified in the instruction contains the memory address of the operand. Register contains memory address.

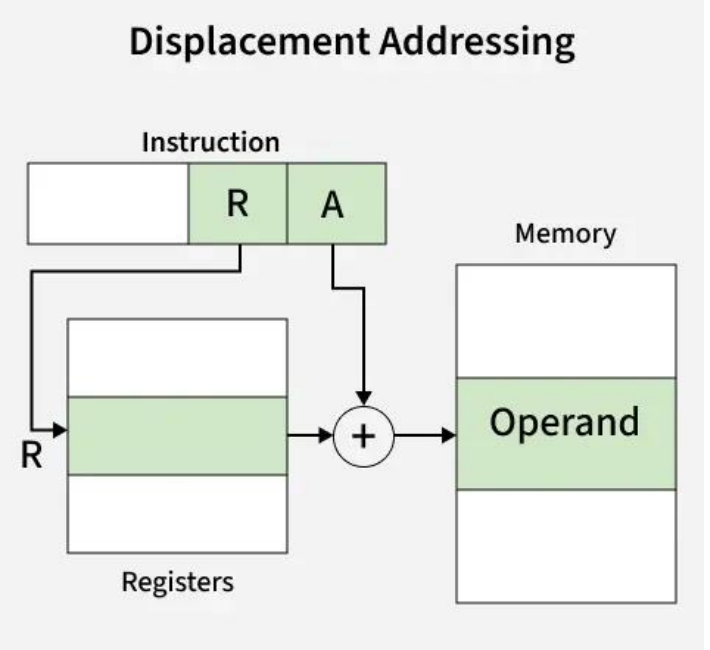
Step:

- The instruction specifies a register.
- This register holds the address (A).
- The CPU fetches the operand from memory location A.
- **Example:** MOV A, [R1] uses content of R1 as memory address.



Displacement Addressing (Indexed, Base-Register, Relative)

The operand's effective address is calculated by adding a constant value (displacement) to the contents of one or more registers. Address = Register + offset (number)



- **Step:**
- The instruction provides a base register (R) and an address part (A).
- CPU adds the value of R and A to get the effective operand address.
- Operand is fetched from the calculated address in memory.

Example: Used for **arrays, loops, tables.**

Displacement Addressing (Indexed, Base-Register, Relative)

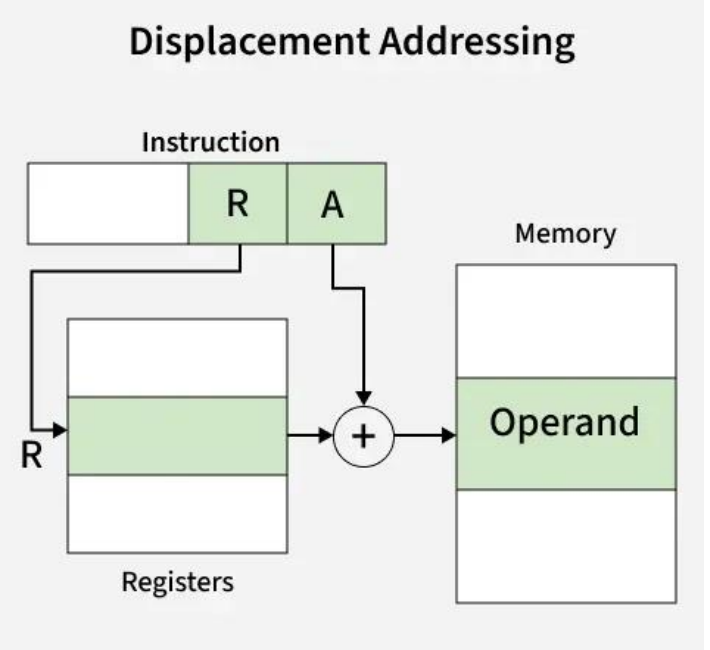
Example: MOV R1, 20(R2)

Effective Address = Register (Base) + offset

If R2 = 1000

Address = **1000** + **20** = **1020**

CPU gets data from memory[1020].

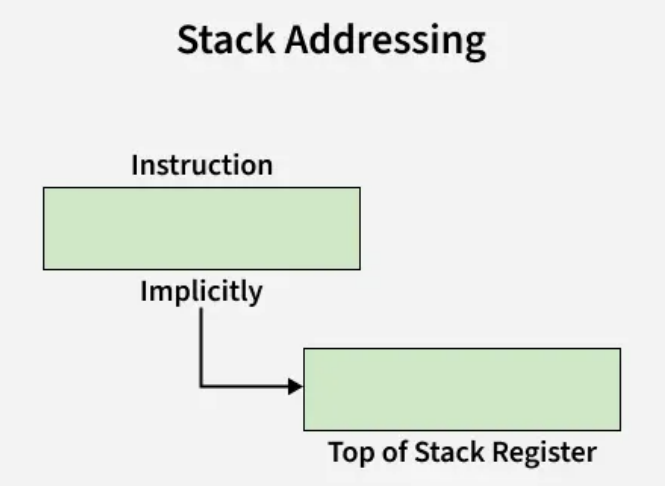


Stack Addressing

The operand is implicitly taken from the top of the stack, without being mentioned in the instruction. Data is taken from **top of stack** (Last in, first out)

Step:

- Operation is performed using the value at the stack's top (implied by instruction).
- No need for explicit operand field; CPU refers to stack pointer register by default.
- **Example:** POP and PUSH operations.

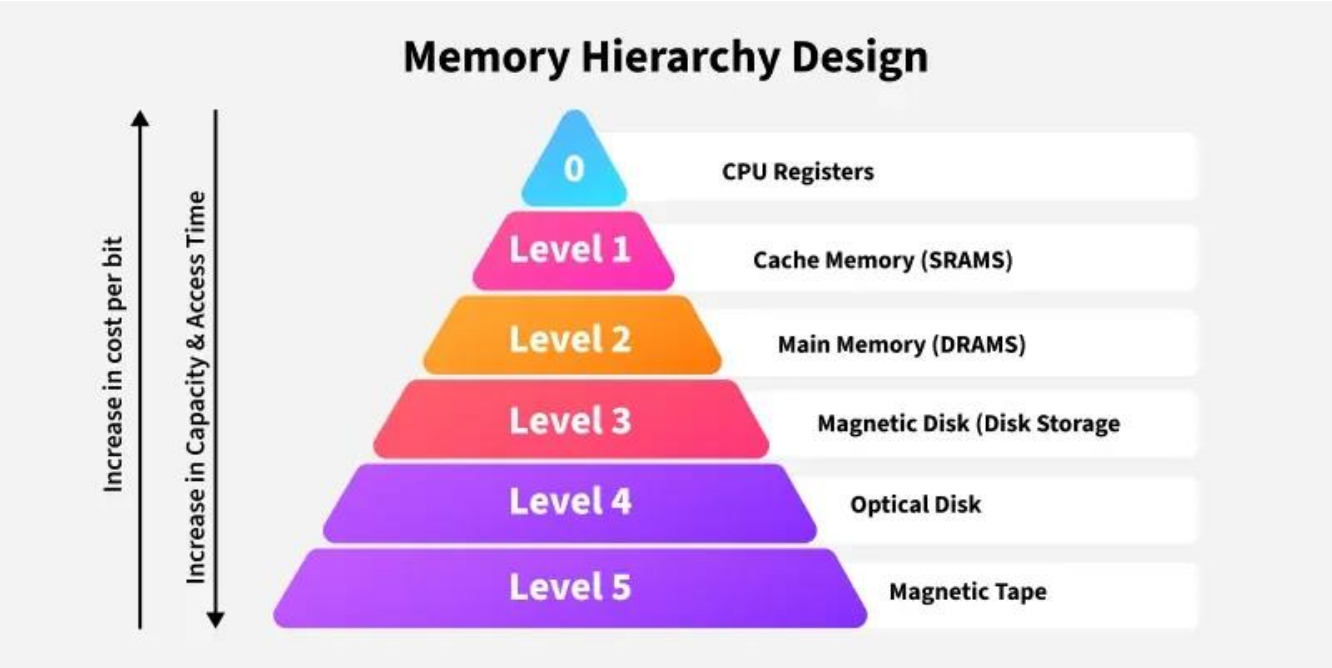


Memory Hierarchy

"My son's RAM just isn't satisfactory - can I part exchange for an upgrade?"

Memory Hierarchy

- The **memory hierarchy** is a system design technique that **organizes memory into multiple levels** (such as registers, cache, main memory, and secondary storage).
- Its main goal is to **reduce memory access time** and **improve overall system performance** by storing frequently used data in faster memory levels.



The design is based on the **principle of locality of reference**, which means:

a. **Temporal locality:** recently used data is likely to be used again soon.

b. **Spatial locality:** data near the recently used location is also likely to be accessed.

- The **higher levels** of the hierarchy (like registers and cache) are **smaller, faster, and more expensive**, while the **lower levels** (like hard disks) are **larger, slower, and cheaper**.

Purpose of Memory Hierarchy

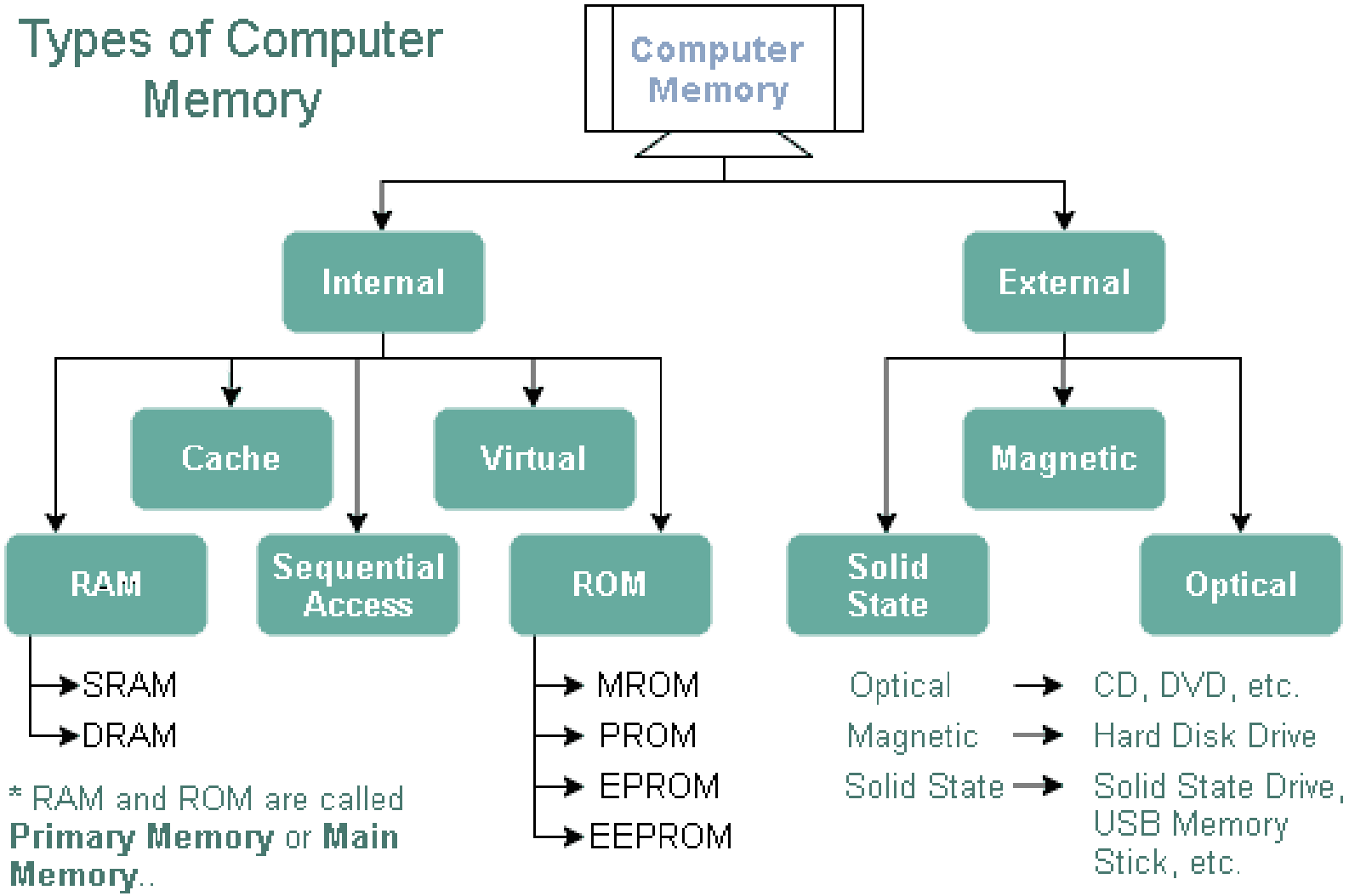
- The **Memory Hierarchy** helps to **use computer memory efficiently** by organizing it into several levels that differ in **speed, size, and cost**.
- **Faster memories** such as **Cache** and **Main Memory (RAM)** allow **quick data access**, but they are **smaller in capacity** and **more expensive**. In contrast, **slower memories** like **Hard Disks** or **Secondary Storage** provide **larger capacity** at a **lower cost**, but take **longer time** to access data.
- Since every type of memory operates at a different speed, the **CPU cannot access all data at the same rate**. The memory hierarchy ensures that the **most frequently used data** stays in the **fastest memory**, improving the overall **performance and efficiency** of the computer system.

Types of Memory Hierarchy

This Memory Hierarchy Design is divided into 2 main types:

- **External Memory or Secondary Memory:** Comprising of Magnetic Disk, Optical Disk, and Magnetic Tape i.e. peripheral storage devices which are accessible by the processor via an I/O Module.
- **Internal Memory or Primary Memory:** Comprising of Main Memory, Cache Memory & CPU registers. This is directly accessible by the processor.

Types of Computer Memory



* RAM and ROM are called **Primary Memory** or **Main Memory**..

Memory Hierarchy Design

1. Registers

Registers are small, high-speed memory units located in the CPU. They are used to store the most frequently used data and instructions. Registers have the fastest access time and the smallest storage capacity, typically ranging from 16 to 64 bits.

2. Cache Memory

Cache memory is a small, fast memory unit located close to the CPU. It stores frequently used data and instructions that have been recently accessed from the main memory. Cache memory is designed to minimize the time it takes to access data by providing the CPU with quick access to frequently used data.

3. Main Memory

- Main memory, also known as RAM (Random Access Memory), is the primary memory of a computer system. It has a larger storage capacity than cache memory, but it is slower. Main memory is used to store data and instructions that are currently in use by the CPU.

Types of Main Memory

- **Static RAM:** Static RAM stores the binary information in flip flops and information remains valid until power is supplied. Static RAM has a faster access time and is used in implementing cache memory.
- **Dynamic RAM:** It stores the binary information as a charge on the capacitor. It requires refreshing circuitry to maintain the charge on the capacitors after a few milliseconds. It contains more memory cells per unit area as compared to SRAM.

4. Secondary Storage

- Secondary storage, such as hard disk drives (HDD) and solid-state drives (SSD) , is a non-volatile memory unit that has a larger storage capacity than main memory. It is used to store data and instructions that are not currently in use by the CPU. Secondary storage has the slowest access time and is typically the least expensive type of memory in the memory hierarchy.

5. Magnetic Disk

- Magnetic Disks are simply circular plates that are fabricated with either a metal or a plastic or a magnetized material. The Magnetic disks work at a high speed inside the computer and these are frequently used.

6. Magnetic Tape

- Magnetic Tape is simply a magnetic recording device that is covered with a plastic film. Magnetic Tape is generally used for the backup of data. In the case of a magnetic tape, the access time for a computer is a little slower and therefore, it requires some amount of time for accessing the strip.

Characteristics of Memory Hierarchy

- **Capacity:** It is the global volume of information the memory can store. As we move from top to bottom in the Hierarchy, the capacity increases.
- **Access Time:** It is the time interval between the read/write request and the availability of the data. As we move from top to bottom in the Hierarchy, the access time increases.
- **Performance:** The Memory Hierarch design ensures that frequently accessed data is stored in faster memory to improve system performance.
- **Cost Per Bit:** As we move from bottom to top in the Hierarchy, the cost per bit increases i.e. Internal Memory is costlier than External Memory.

System-Supported Memory Standards

According to the memory Hierarchy, the system-supported memory standards are defined below:

Level	1	2	3	4
Name	Register	Cache	Main Memory	Secondary Memory
Size	<1 KB	less than 16 MB	<16GB	>100 GB
Implementation	Multi-ports	On-chip/SRAM	DRAM(capacitor memory)	Magnetic
Access Time	0.25ns to 0.5ns	0.5 to 25ns	80 ns to 250ns	50 lakh ns
Bandwidth	20000 to 1 lakh MB	5000 to 15000	1000 to 5000	20 to 150
Manage by	Compiler	Hardware	Operating System	Operating System
Backing Mechanism	From cache	from Main Memory	from Secondary Memory	from ie

Advantages of Memory Hierarchy

- **Performance:** Frequently used data is stored in faster memory (like cache), reducing access time and improving overall system performance.
- **Cost Efficiency:** By combining small, fast memory (like registers and cache) with larger, slower memory (like RAM and HDD), the system achieves a balance between cost and performance. It saves the consumer's price and time.
- **Optimized Resource Utilization:** Combines the benefits of small, fast memory and large, cost-effective storage to maximize system performance.
- **Efficient Data Management:** Frequently accessed data is kept closer to the CPU, while less frequently used data is stored in larger, slower memory, ensuring efficient data handling.

Disadvantages of Memory Hierarchy

- **Complex Design:** Managing and coordinating data across different levels of the hierarchy adds complexity to the system's design and operation.
- **Cost:** Faster memory components like registers and cache are expensive, limiting their size and increasing the overall cost of the system.
- **Latency:** Accessing data stored in slower memory (like secondary or tertiary storage) increases the latency and reduces system performance.
- **Maintenance Overhead:** Managing and maintaining different types of memory adds overhead in terms of hardware and software.

References

- Fox, C. (2022). Computer architecture: From the Stone Age to the quantum age. No Starch Press.
- Plantz, R. G. (2020). Introduction to computer organization: An under-the-hood look at hardware and ARM64 assembly (ARM ed.). No Starch Press.
- Stallings, W. (2022). Computer Organization and Architecture: Designing for Performance (11th ed.). Pearson.
- Boyd, C. (2020, January 1). How CPUs are designed and built: Fundamentals of computer architecture. TechSpot. https://www.techspot.com/article/1821-how-cpus-are-designed-and-built/


Any
Question