Parul® University
Vadodara, Gujarat
NAAC GRADE A++

Information and
Communication Technology

Computer Organization-2

Dr. M Krishnam Raju
Assistant Professor
Electronics & Communication Engineering

Content

1. **Assembler:** Machine Language, Assembly Language, Assembler, Program loops, Programming Arithmetic and logic operations, subroutines, I-O Programming.
2. **Memory Organization:** Memory hierarchy, Main memory, Auxiliary, memory, Flash memory, Associative memory, Cache memory, Virtual memory

Machine Language & Assembly Language & Assembler

Aspect	Machine Language	Assembly Language	Assembler
Definition	Binary code that the CPU directly understands.	Human-readable code using mnemonics that represent machine code instructions.	A program that translates assembly language into machine language.
Representation	Binary (0s and 1s).	Mnemonics (e.g., MOV, ADD, SUB).	Translates code line by line.
Ease of Use	Difficult for humans to read, write, or debug.	Easier to understand than machine language, but still low-level.	Does not interact with the user directly; just performs translation.
Control Over Hardware	Full control, as it is directly executed by the CPU.	Provides control over the hardware, but in a more readable form.	Converts assembly instructions to machine instructions.
Portability	Highly specific to CPU architecture (e.g., x86, ARM).	Also specific to CPU architecture.	Specific to the architecture for which it's built (e.g., x86 assembler for Intel CPUs).
Human-Readable?	No, it's in binary code.	Yes, but still low-level compared to high-level languages.	No, it's just a tool.
Error-Prone?	Very error-prone, since it's all raw binary and hard to debug.	Less error-prone than machine code but still requires care.	Not error-prone, but the assembly code must be correctly written for it to work.
Translation Process	Directly executed by the CPU, no translation needed.	Needs to be translated into machine language to be executed.	Translates assembly code into machine code for execution.
Example	10110000 01100001	MOV AL, 61h	Converts MOV AL, 61h into 10110000 01100001

Program loops

- Loops are used to repeat a certain set of instructions a specific number of times or until a certain condition is met. Loops are an essential part of most programs, especially when repetitive tasks need to be performed.
- Finite loop will repeat a certain number of times and then exit. We can implement this using a counter stored in a register (or memory location). Once the counter reaches zero, the loop terminates.

```
MVI B, 05      Load register B with the number 5 (loop count)
LOOP:
Your operation goes here (e.g., moving data, adding, etc.)
  DCR B        Decrement B (counter)
  JNZ LOOP     If B is not zero, jump back to LOOP
```

Program loops

In a conditional loop, the loop continues to execute as long as a certain condition is met. This can be based on flags like Zero, Carry, etc.

Example 1

```
MVI A, 30    ; Load A with 30
MVI B, 10    ; Load B with 10
LOOP:
   SUB B      ; Subtract the value in B from A
   JZ END     ; Jump to END if the Zero flag is set
   JMP LOOP   ; Otherwise, repeat the loop
END:
   NOP        ; No operation (end of the program)
```

Example 2

```
MVI A, 10    ; Load A with 10
MVI B, 20    ; Load B with 20
LOOP:
   ADD B      ; Add B to A
   JC CARRY   ; If carry occurs, jump to CARRY
   JMP LOOP   ; Otherwise, repeat the loop
CARRY:
   NOP        ; No operation (Carry has occurred)
```

Programming Arithmetic and logic operations

Arithmetic and logic operations are fundamental operations in programming, essential for performing calculations, comparisons, and decision-making within programs.

Example 1

```
MVI A, 05H     ; Load 05 into accumulator
MVI B, 03H     ; Load 03 into register B
ADD B          ; Add contents of register B to accumulator (A = A + B)
HLT            ; Halt execution
```

Example 2

```
MVI A, 08H     ; Load 08 into accumulator
MVI B, 03H     ; Load 03 into register B
SUB B          ; Subtract contents of register B from accumulator (A = A - B)
HLT            ; Halt execution
```

Example 3

```
MVI H, 30H     ; Load 30H into register H
MVI L, 40H     ; Load 40H into register L
INX H          ; Increment HL register pair (HL = HL + 1)
HLT            ; Halt execution
```

Example 4

```
MVI B, 02H     ; Load 02H into register B
MVI C, 00H     ; Load 00H into register C
DCX B          ; Decrement BC register pair (BC = BC - 1)
HLT            ; Halt execution
```

Subroutines

- Subroutines refer to a series of instructions that perform a specific task and can be reused multiple times within a program. They are analogous to functions or procedures in higher-level programming languages.
- A subroutine in 8085 can be called from the main program, and after it completes its task, control returns to the instruction following the subroutine call.

A Simple Subroutine for Adding Two Numbers

Example 1: Main Program

```
MVI A, 05H        ; Load 5 into the accumulator (A)
MVI B, 03H        ; Load 3 into register B
CALL ADDITION     ; Call the subroutine
HLT               ; Halt the program

Subroutine to Add Two Numbers
ADDITION:
  MOV A, B        ; Move the value of B into A
  ADD C           ; Add the value of C (from register C) to A
  RET             ; Return from the subroutine
```

Subroutines

A Subroutine with Return Address on the Stack

Main Program

```
MVI A, 10H        ; Load 10 into accumulator A
MVI B, 05H        ; Load 5 into register B
CALL SUBROUTINE    ; Call the subroutine
HLT               ; Halt the program

Subroutine: Multiply A and B
SUBROUTINE:
  MOV A, B        ; Move the value of B into A
  MUL B           ; Perform the multiplication (just an example operation)
  RET             ; Return from the subroutine
```

I/O Programming

- I/O programming in the context of the 8085 microprocessor refers to how the processor interacts with external devices such as keyboards, displays, or sensors.
- These devices typically have their own set of input and output operations, which the 8085 processor manages through specific I/O instructions.

Program to read data from a keypad

```
START:  IN 01H       ; Read data from port 01H
(keypad)
       MOV B, A      ; Move the data into register B
(processing it)
       CALL PROCESS  ; Process the data as per your
application
       JMP START     ; Repeat the process
continuously
PROCESS:
Code to process the data in register B, could be
an action based on the key
       RET
```

I/O Programming

Program to output data to an LED display

```
START:  MVI A, 0FFH    ; Load A with 0FFH (turn on all LEDs)
        OUT 02H        ; Output the data to the LED connected at port 02H
        CALL DELAY     ; Delay for a short period
        MVI A, 00H     ; Load A with 00H (turn off all LEDs)
        OUT 02H        ; Output the data to the LED
        CALL DELAY     ; Delay again
        JMP START      ; Repeat the process

DELAY:  MVI B, 30      ; Set up a loop counter
DELAY_LOOP:
        NOP            ; No operation (just consumes time)
        NOP
        NOP
        NOP
        DCR B          ; Decrement B
        JNZ DELAY_LOOP  ; Repeat until B is zero
        RET
```

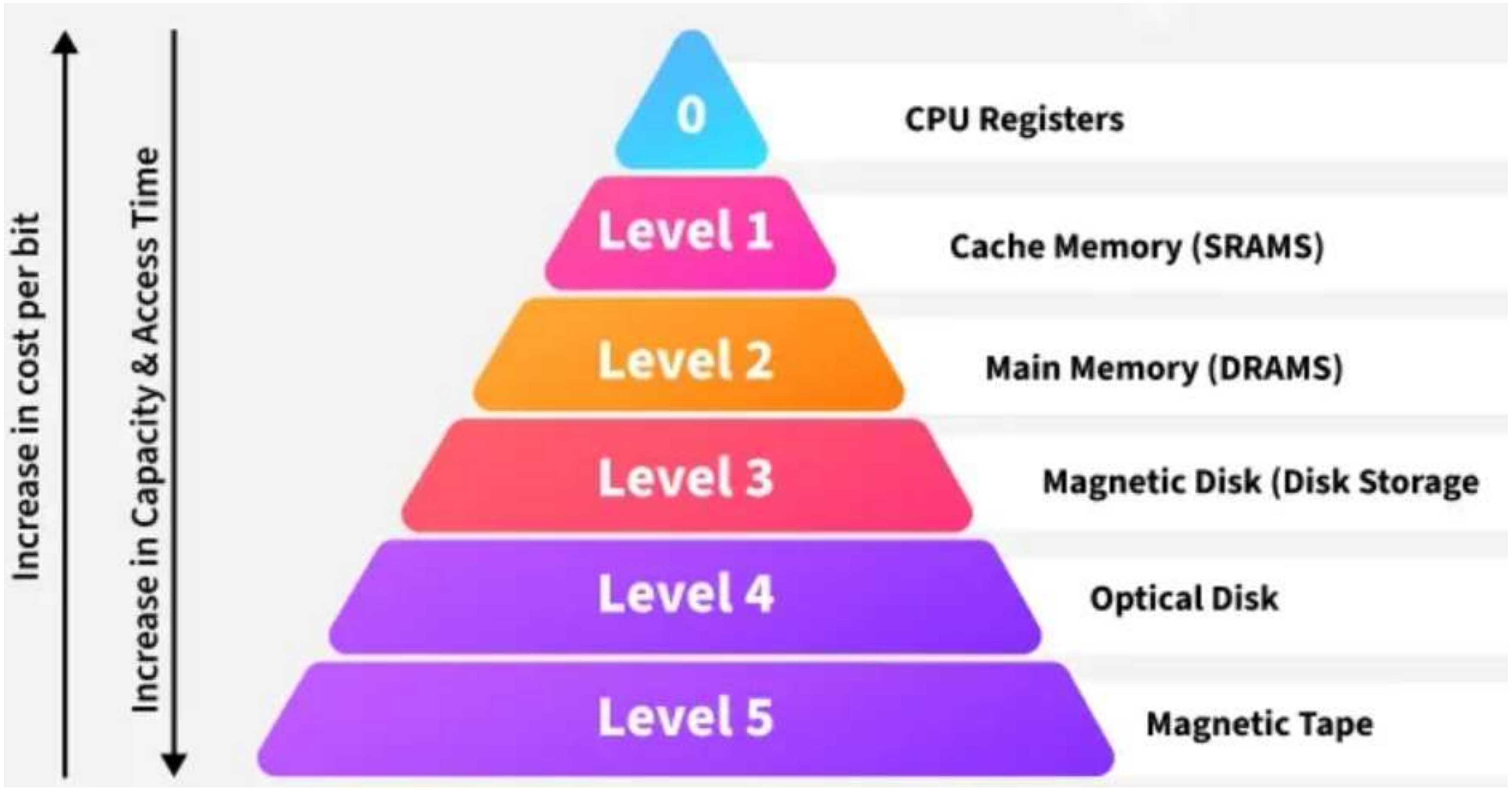
Memory Organization

- The memory organization of the 8085 microprocessor involves how the processor accesses and manages data and instructions in its memory system.
- The 8085 microprocessor has a 16-bit address bus, meaning it can address 2^16 (64KB) of memory locations. Here's a detailed breakdown:
- **Memory Addressing:**
  - The 8085 has a 16-bit address bus (A0 to A15), which can address a total of 64KB of memory (from address 0000H to FFFFH).
  - Each memory location in this range can hold one byte of data.
- **Memory Organization:**
  - **ROM (Read-Only Memory):** This is used to store the program code (instructions) that the microprocessor will execute. ROM is usually located at the lower part of memory in the 8085 system.
  - RAM (Random-Access Memory): This stores the data that the microprocessor operates on, as well as the stack and variables used during program execution. RAM is located at higher addresses.

Memory Organization

- ROM (Read-Only Memory): ROM is non-volatile memory, which means it retains data even when the power is turned off. ROM is used to store the permanent program code.
    - EPROM: Erasable programmable ROM, which can be erased and rewritten.
    - EPROM/EEPROM: Can be rewritten, but only under specific conditions (like exposure to UV light or electrical signals).
- RAM (Random Access Memory): RAM is volatile memory used for temporary storage during program execution.
    - Static RAM (SRAM): Faster and more reliable, but consumes more power.
    - Dynamic RAM (DRAM): Slower than SRAM but more cost-effective.
- Example:
    - 16KB ROM (0000H to 3FFFH)
    - 32KB RAM (4000H to 7FFFH)
    - I/O Devices (8000H to FFFFH)

Memory Hierarchy



Memory Hierarchy

Types of Memory Hierarchy

- **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.

Memory Hierarchy Design:

- **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.
- **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.
- **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.

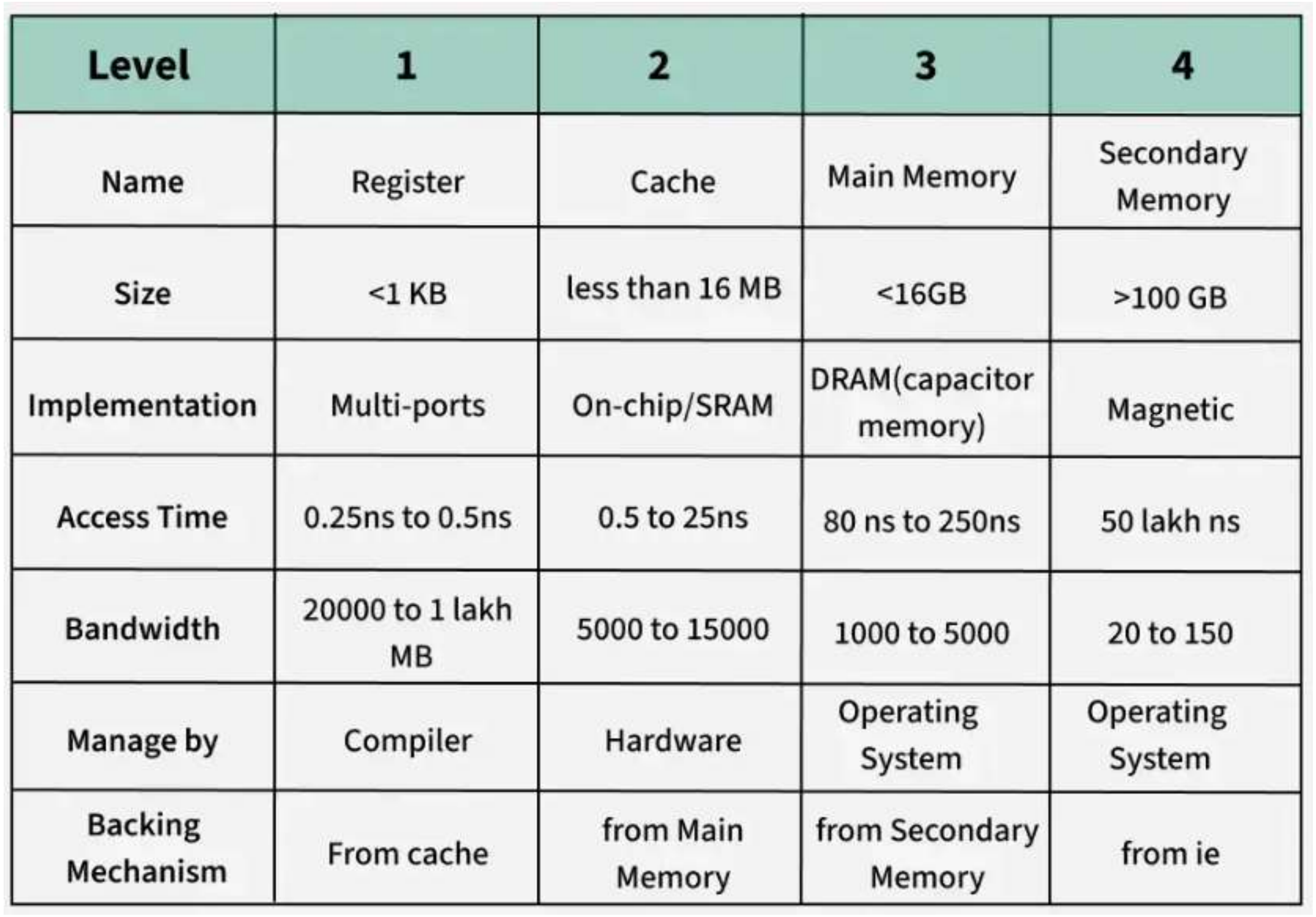
Memory Hierarchy

- **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.
- **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.

Memory Hierarchy

- **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.
- **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.

Memory Hierarchy



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

Memory Hierarchy

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.

Main and Auxiliary Memory

Main Memory: It is the memory the CPU can access directly and quickly. It stores data and instructions that are currently in use.

- RAM (Random Access Memory): Temporary storage for programs while they are running.
- ROM (Read-Only Memory): Permanent storage for essential system instructions (like booting up).
- Main memory is fast but limited in size.
- Data is lost when the power is turned off (especially in RAM).

Auxiliary Memory: It is used for long-term storage of data and programs. It is not directly accessed by the CPU; the data must be first loaded into main memory.

- Hard Drives (HDD/SSD)
- USB Flash Drives
- CDs, DVDs
- Auxiliary memory is slower but much larger in size.
- Data is retained even when the power is off.

Main and Auxiliary Memory

Feature	**Main Memory**	**Auxiliary Memory**
Also called	Primary Memory	Secondary Memory
Speed	Very fast	Slower compared to main memory
Size	Limited (small)	Very large
Volatility	Volatile (data lost when power off)	Non-volatile (data stays)
Examples	RAM, ROM	Hard Drive, SSD, USB drive, CD/DVD
Direct CPU Access	Yes	No (data must be loaded first)
Purpose	Store currently used data/programs	Store data and programs long-term
Cost per MB	More expensive	Cheaper

Flash memory

- **Flash memory** is a type of non-volatile storage, meaning it keeps data even when the power is off. It's used in things like USB drives, SSDs (solid-state drives), memory cards, and smartphones.
- There are two main types:
  - **NAND flash** — mostly used in storage (SSDs, USB sticks).
  - **NOR flash** — often used where fast reading and reliable code storage is needed (like in embedded systems).
- Flash memory retains data without requiring a constant power supply. This is in contrast to volatile memory types like RAM, which lose their data once power is lost.
- Flash memory allows for fast read and write operations. Write speeds, however, are generally slower compared to read speeds.
- Flash memory can be erased and reprogrammed using electrical signals, which makes it more versatile than traditional magnetic storage like hard drives.

Flash memory

- Flash memory has a limited number of write/erase cycles. This means that after a certain number of write and erase operations (usually in the range of thousands to millions), the cells begin to degrade, and the memory can no longer reliably store data.
- Flash memory is organized into blocks, and data can only be erased at the block level. This makes random erasure of small sections of data more difficult.
- Flash memory is more durable and resistant to physical shock and vibration compared to traditional hard drives, as it has no moving parts.
- Flash memory consumes very little power, which is why it's used in portable devices like smartphones, USB drives, and SSDs.

Associative Memory

- Associative memory, also known as content-addressable memory (CAM), is a type of memory in which data can be retrieved based on its content rather than its specific address. This is different from traditional memory systems where data is accessed by its address.
- Associative memory is often used in artificial neural networks, pattern recognition, and other systems where the input is compared to stored data to find a match.
- **Content-Based Retrieval:** Instead of accessing memory locations through addresses, associative memory retrieves information based on the content provided as input. When a partial or noisy version of the input is given, the system can retrieve the most relevant stored data that matches it.
- **Parallel Search:** In associative memory, all stored patterns can be searched simultaneously. This is in contrast to traditional memory systems, which typically require sequential searching or direct addressing. This allows for faster retrieval in certain applications.
- **Fault Tolerance:** Associative memory can often retrieve the correct data even if the input is incomplete or noisy. This is because the system typically performs a "best match" search, allowing it to deal with errors or missing information in the input.

Associative Memory

- **Pattern Completion:** Associative memory is capable of completing or filling in incomplete patterns. If part of the input pattern is provided, the memory can often recall the full stored pattern that matches.
- **Biological Inspiration:** The concept of associative memory is inspired by the way the human brain works. In the brain, memories are not stored in specific locations but are distributed across neural networks. When you think of something, your brain associates it with related information, which can trigger the recall of other memories.
- **Associative Storage:** In this type of memory, the data is stored with a key that can be used to retrieve it. In some systems, the storage is binary, where each pattern is associated with a particular "label" or identifier.
- **Retrieval Time:** In an ideal associative memory system, the retrieval time is constant regardless of the amount of data stored. This can make associative memory very efficient, especially for applications where fast and content-based retrieval is necessary.
- **Associative Networks:** Associative memory can be represented as a network, where each node (memory unit) is connected to others in a way that allows for the retrieval of related information based on the network's structure.

Cache Memory

- Cache memory is a small, high-speed storage area located between the CPU and the main memory (RAM) in a computer system. It stores frequently accessed data and instructions, allowing the CPU to quickly retrieve them without needing to access the slower main memory.
- High Speed: Cache memory is much faster than main memory. It is built from static RAM (SRAM), which allows it to operate at speeds close to the CPU.
- Size: Cache memory is relatively small compared to main memory. It is designed to store only a small subset of data that is most likely to be used next, which is why it cannot hold all of the system's data.
- Levels of Cache: There are typically multiple levels of cache in modern computers:
  - L1 Cache: This is the smallest and fastest cache located closest to the CPU cores. It stores a small amount of data or instructions (usually 16-128 KB) that are most frequently used.
  - L2 Cache: Larger than L1, L2 cache can range from a few hundred KB to several MB. It serves as a buffer between the L1 cache and main memory, helping to further reduce access time.

Cache Memory

  - L3 Cache: Larger and slower than L2, L3 cache is shared by multiple CPU cores and can be several MB in size.
- Access Time: Cache memory significantly reduces the time it takes for the CPU to access data compared to fetching it from RAM. The access time for cache is measured in nanoseconds, much faster than the milliseconds for main memory.
- Hit and Miss: When the CPU looks for data in the cache:
  - Cache Hit: If the required data is found in the cache, it’s called a "hit," and the CPU retrieves it quickly.
  - Cache Miss: If the data is not in the cache, it’s called a "miss," and the CPU has to fetch the data from main memory, which takes longer.
- Associative Mapping: Cache memory uses various mapping schemes to determine where data is stored, such as:
  - Direct-mapped cache: Each block of main memory is mapped to exactly one line in the cache.
  - Set-associative cache: Cache is divided into sets, and each set can contain multiple blocks of data.
  - Fully associative cache: Data can be placed anywhere in the cache, offering the highest flexibility but requiring more complex control logic.

Virtual Memory

Virtual memory is a memory management technique that allows an operating system to compensate for physical memory limitations by using a portion of the hard drive or SSD as additional "virtual" memory. Here are the key properties of virtual memory:

- Each process is given the illusion of having its own contiguous block of memory, even though physical memory is fragmented. This helps protect processes from each other, as one process cannot directly access the memory of another.
- Virtual memory is divided into fixed-size blocks called "pages." Physical memory is divided into corresponding blocks known as "page frames." When a process accesses a page that is not in physical memory, a page fault occurs, and the operating system loads the required page from disk into RAM.
- In addition to paging, some systems use segmentation, where memory is divided into segments based on the logical divisions in a program (such as code, data, stack). This provides more flexibility than paging alone but is not as commonly used in modern operating systems.
- A page table maps virtual addresses to physical addresses. The operating system maintains this table to keep track of which virtual pages are stored in which physical page frames.

Virtual Memory

- When the system runs low on physical memory, it can move some of the pages that are not being actively used to a disk area called the "swap space." This process is called swapping or paging out. When those pages are needed again, they are swapped back into physical memory, which can slow down performance if excessive swapping occurs.
- This is a technique where a page is only loaded into physical memory when it is first needed (i.e., on demand). This helps to save memory resources and reduce load times for programs.
- Virtual memory ensures that a process cannot access the memory of other processes or the kernel memory, preventing one process from corrupting or interfering with another process or the operating system.
- By using virtual memory, operating systems can run larger applications than would otherwise fit into physical RAM. Only the most frequently accessed parts of programs need to be kept in RAM at any given time.
- Virtual memory involves translating a virtual address used by a program into a physical address in the computer's memory. This translation is done through a combination of hardware (the Memory Management Unit or MMU) and software (operating system).

Parul® University | NAAC GRADE A++

https://paruluniversity.ac.in/