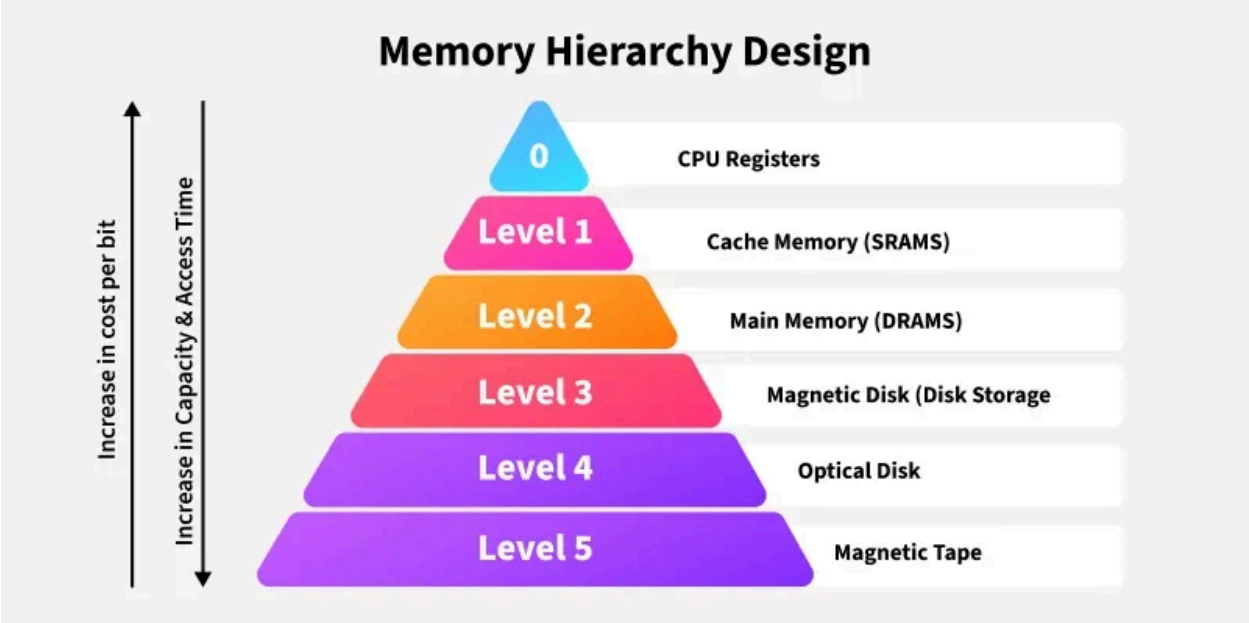
Module V – The Memory System

Basic Concepts of Memory System

The processor needs data and instructions to perform operations. These are stored in a hierarchy of memories, organized as follows:

Fastest, Smallest, Most Expensive



Slowest, Largest, Cheapest

This is called the **Memory Hierarchy**.

Why hierarchy?

Because:

- Fast memory is very expensive.
- Cheap memory is very slow.
- We need both speed (for CPU) and size (for programs).

Characteristics of Memory (Important)

(a) Access Time

Time taken to read/write a memory location.
Example:

- Register: < 1 ns
- Cache: 1–5 ns
- RAM: 50–100 ns
- Disk: milliseconds

(b) Cycle Time

Minimum time before the next operation can start.

(c) Bandwidth

Amount of data transferred per second.

(d) Capacity

Total amount of memory stored.

(e) Cost per bit

Price of storing one bit.

- Registers = highest cost
- Hard Disk = lowest cost

Types of Memories in Basic Concept

The module begins with basic classification:

(A) Primary Memory

Used by the processor directly.

- **Cache**
- **Main Memory (RAM)**

(B) Secondary Memory

Used for long-term storage.

- HDD
- SSD
- Pen drive

Secondary memory is NOT directly accessed by CPU; it goes through RAM.

Semiconductor Memories

These are made from semiconductor chips (ICs).
Two major types:

(A) RAM (Random Access Memory)

You can access any location equally fast.

Types of RAM:

1. **Static RAM (SRAM)**
2. **Dynamic RAM (DRAM)**

Static RAM (SRAM)

- Built using **6 transistors per bit**.
- Does **not** need refreshing.
- Very fast, very reliable.
- Used for **Cache memory**.
- BUT very expensive and large in size.

Example: L1, L2, L3 cache in CPUs.

Dynamic RAM (DRAM)

- Uses **1 transistor + 1 capacitor per bit**.
- Capacitor leaks charge → **needs refreshing every few ms**.
- Slower and less expensive.
- Used for **Main Memory (RAM)**.

Example: DDR3, DDR4, DDR5 RAM in laptops.

ROM (Read Only Memory)

Stores data permanently → used for system startup.

Types:

- **Masked ROM** – permanent factory-written
- **PROM** – programmable once
- **EPROM** – erasable by UV light

- **EEPROM / Flash** – electrical erase, used in pendrives, SSD

Used for:

- BIOS
- Firmware
- Embedded devices

Organization of Larger Memories

A single memory chip is small.
Example: 512K × 8 chip → holds 512K bytes.

To create bigger memories:

- Chips are arranged **in parallel** → increase word size
- Chips are arranged **in series** → increase memory size

Example:

To build 2M × 32 memory:

- Use 512K × 8 chips
- 4 chips arranged for 32-bit word (8 × 4 = 32)
- Then 4 groups in series to reach 2M rows

Speed, Size & Cost Trade-off (The Triangle)

You CANNOT get:

- **large size**
- **high speed**
- **low cost**

So architects combine SRAM + DRAM + Disk to get a balanced design → known as **memory hierarchy**.

Memory Type	Speed	Cost	Size
Registers	Fastest	Highest	Very small
Cache (SRAM)	Very fast	High	Small
RAM (DRAM)	Medium	Moderate	Medium
Disk	Slow	Low	Very large

Locality of Reference (Very Important Basic Concept)

Programs show predictable memory access patterns:

(a) Temporal Locality

If a location is accessed now, it will be accessed again soon.
Example: Loop variables.

(b) Spatial Locality

Nearby memory locations are often accessed.
Example: Array traversal.

These concepts are used to design:

- Cache memory

- Prefetching
- Virtual memory

Summary

Topic	Meaning
Memory hierarchy	Combines fast & slow memories
SRAM	Fast, expensive → cache
DRAM	Slow, cheap → main memory
ROM	Permanent storage
Memory organization	How chips are arranged
Locality	Explains why cache works
Speed/size/cost	Fundamental trade-off

Semiconductor RAM Memories – Organization – Static & Dynamic RAM

Semiconductor RAM is the **main memory technology** used in modern computers.
It is called *Random Access Memory* because **any memory location can be accessed directly**, without going through other locations.

There are **two main types** of semiconductor RAM:

1. **Static RAM (SRAM)**
2. **Dynamic RAM (DRAM)**

Before explaining them, let's look at **how RAM is organized internally**.

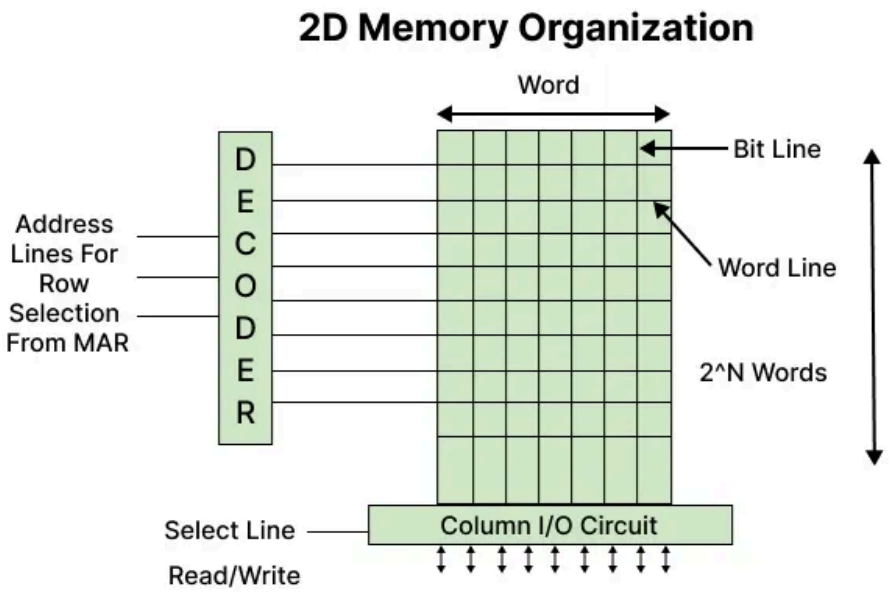
Organization of Semiconductor RAM

A RAM chip is organized as a **matrix (array) of memory cells**.

Example:
A memory chip labeled **64K × 8** means:

- 64K = 65,536 addresses
- Each address stores **8 bits = 1 byte**
- Total capacity = 65,536 × 8 bits

RAM Internal Structure



When the CPU wants to read/write:

1. **Address** is sent to RAM.
2. Address is split internally into:
    - Row address
    - Column address

3. Row decoder activates the correct row.
4. Column decoder picks the correct cell/byte.
5. Data is read/written.

This is how any RAM works (both SRAM & DRAM).

Static RAM (SRAM)

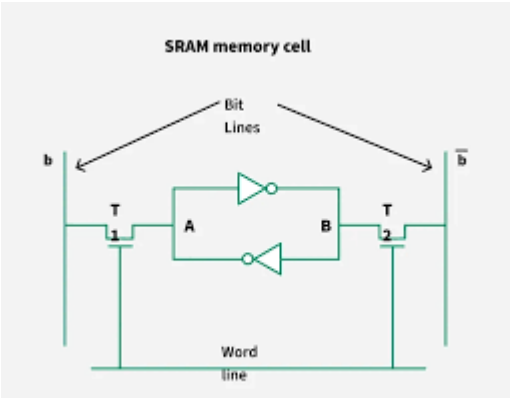
What is SRAM?

Static RAM stores each bit using **6 transistors** forming a bistable flip-flop.

6 Transistors → 1 memory bit

Because it is a flip-flop, it **holds data as long as power is ON** — no refreshing needed.

Internal Structure of an SRAM Cell



Properties of SRAM

Property	Description

Speed	Very FAST
Cost	Expensive (more transistors → bigger chip)
Power usage	Higher
Density	Lower (occupies more space)
Refreshing	❌ No refresh required
Used in	**Cache memory (L1, L2, L3)**

Advantages

- Fastest memory type
- No refresh circuitry → consistent speed

Disadvantages

- Larger size per bit
- High cost → too expensive for main memory

Where is SRAM used?

- **CPU Cache**
    - L1 cache (32KB–128KB)
    - L2 cache (512KB–2MB)
    - L3 cache (2MB–32MB)

Dynamic RAM (DRAM)

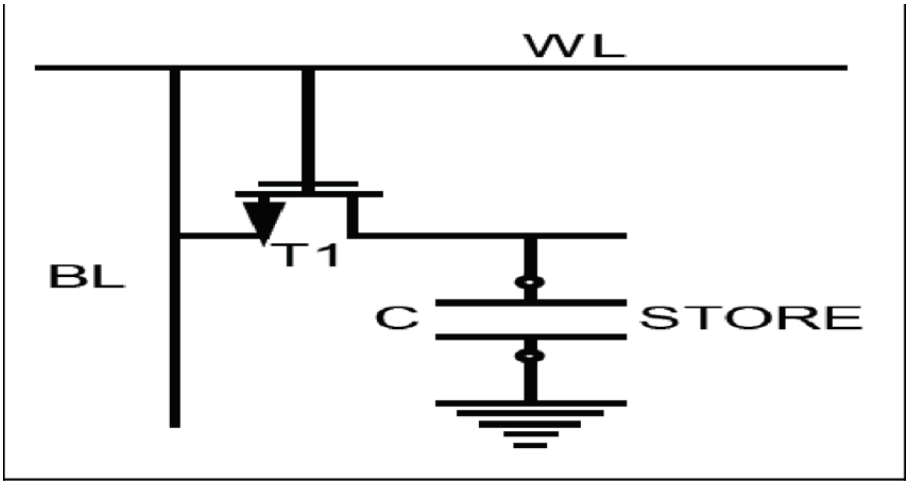
What is DRAM?

Dynamic RAM stores each bit using **1 transistor + 1 capacitor**.

1 transistor + 1 capacitor → 1 bit

The capacitor **leaks charge**, so it must be **refreshed periodically**.

Internal Structure of a DRAM Cell



Properties of DRAM

Property	Description
Speed	Slower than SRAM
Cost	Cheaper
Density	Higher (tiny cell → more bits/chip)
Refreshing	✔ Requires refresh (every few ms)
Used in	**Main Memory (RAM)**

Advantages

- High density → more memory per chip
- Lower cost → affordable for large RAM sizes
- Lower power than SRAM

Disadvantages

- Needs refresh → slower
- More complex controller
- Slower access compared to SRAM

Where is DRAM used?

- **Main Memory (system RAM)**
    - DDR3, DDR4, DDR5 modules
- Graphics RAM (GDDR)

Comparison: SRAM vs DRAM

This finds frequent place in exams.

Feature	SRAM	DRAM
Cell structure	6 transistors	1 transistor + 1 capacitor
Refresh	❌ No	✔ Yes
Speed	Very high	Medium
Density	Low	High
Cost	Very high	Low
Power consumption	High	Low
Used in	Cache memory	Main memory

Simple explanation:

- SRAM = fast but expensive → used for CACHE

- DRAM = cheaper but slower → used for MAIN MEMORY

Why both SRAM and DRAM are needed?

A CPU needs:

- **Small, very fast memory** → caches → SRAM
- **Large, cost-effective memory** → main memory → DRAM

Using only SRAM: computer would be very fast but extremely expensive.
Using only DRAM: computer would be slow because CPU waits for memory.

So designers combine them in a memory hierarchy.

SUMMARY

- Semiconductor RAM is organized internally as a **matrix** of rows × columns.
- **SRAM** uses flip-flop → 6 transistors → no refresh → used in cache.
- **DRAM** uses capacitor → needs refresh → used in main memory.
- SRAM is **fast & expensive**, DRAM is **slow & cheap**.
- Memory hierarchy uses both to balance speed and cost.

Structure of Larger Memories (Detailed Explanation)

Modern computers need large memory sizes like **1GB, 4GB, 16GB** etc.
But a **single memory chip** cannot store so much.

Therefore, computer architects **combine multiple small chips** to build a **larger memory unit**.

This is called **Memory Module Organization**.

Why We Need Larger Memory Structures?

A RAM chip comes in small sizes like:

- 64K × 4
- 128K × 8
- 512K × 8
- 1M × 16

But the CPU may require:

- 2M × 8
- 2M × 32
- 4M × 16
- 8M × 64

So smaller chips are **arranged together** to build bigger memories.

Two Ways to Build Larger Memory

There are **only two basic goals**:

(A) Increase the Word Size (Number of Bits per Address)

Example:
We need **8 bits**, but each chip gives only **4 bits**.
So we connect chips **in parallel**.

Chip 1 → 4 bits
Chip 2 → 4 bits

Total → 8 bits/word

This is called **bit-parallel connection**.

(B) Increase the Number of Addresses (Memory Depth)

Example:
We need **2M** addresses, but each chip has **512K** addresses.
We connect chips **in series** (one after another).

512K
512K
512K
512K

= 2M rows

This is called **address expansion**.

Putting Both Together: Word Size Expansion + Address Expansion

Real memory modules require both:

Example (very common exam question):

Design a 2M × 32 memory using 512K × 8 chips

Let's break it:

Step 1: Increase word size from 8 bits → 32 bits

One chip = 8 bits
To get 32 bits:

32 / 8 = 4 chips in parallel

So one row looks like this:

Chip1 (8 bits)
Chip2 (8 bits)
Chip3 (8 bits)
Chip4 (8 bits)

Total: 32-bit word

Step 2: Increase number of addresses from 512K → 2M

How many such groups needed?

2M / 512K = 4 groups

So 4 groups × 4 chips per group = **16 chips** total.

Final Structure:

Group 1: 4 chips → 32-bit word
Group 2: 4 chips → 32-bit word
Group 3: 4 chips → 32-bit word
Group 4: 4 chips → 32-bit word

CPU selects the group using a **chip select decoder**.

General Rules (Very Important)

Rule 1: To increase word size → connect chips in parallel

- All chips get same address input
- Data outputs combine to form a bigger word

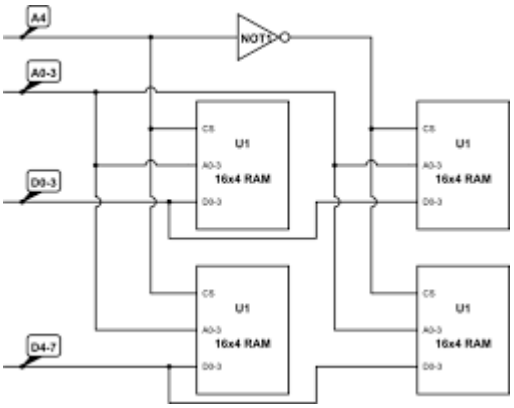
Rule 2: To increase memory depth → connect chips in series

- Chip Select decides which chip is active
- Decoder is used

For N chips: use a $\log_2(N)$-to-N decoder.

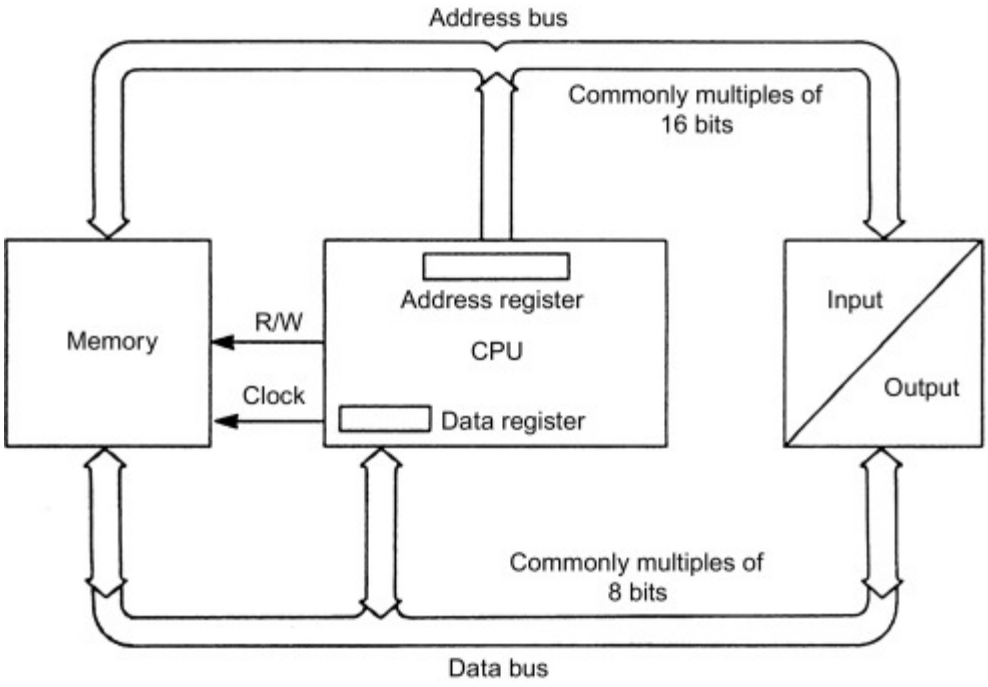
Block Diagram of Larger Memory Structure

Example: Building larger word size (8-bit from two 4-bit chips)



All chips receive the same Address
Parallel outputs combine into one data bus.

Example: Increasing address rows (series connection)

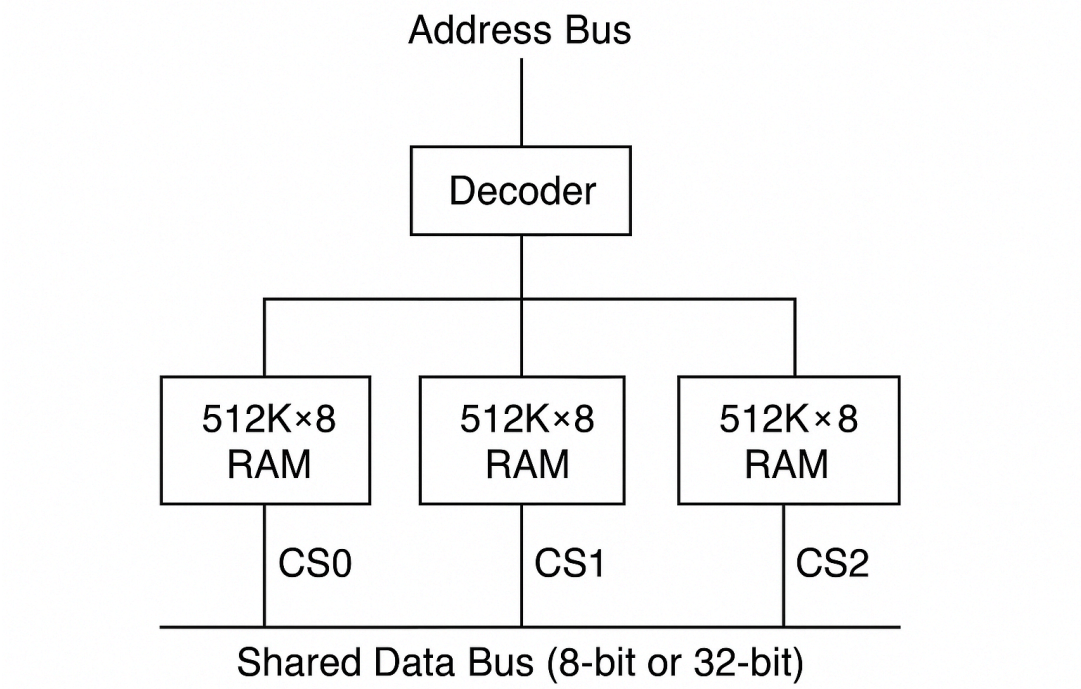


Decoder ensures only **one chip is active at a time**.

Why Larger Memory Structure is Needed? (Conceptual Points)

- A single chip cannot store large memory.
- Using parallel & series combinations allows flexible memory size design.
- Avoids manufacturing extremely large ICs.
- Reduces cost & improves reliability.
- Memory controller simplifies CPU interface.

Typical Exam Diagram



SUMMARY

To design large memory:

1. **Increase word size → chips in parallel**
2. **Increase address size → chips in series**
3. **Use decoder for chip selection**
4. **All chips share the same address and control lines**
5. **Data lines are merged to form required width**

Semiconductor ROM Memories

ROM stands for **Read Only Memory**.
It is a type of **non-volatile semiconductor memory**, which means:

- Data is **not lost** when power is switched OFF
- Contents are **permanently stored**

ROM is used mainly for:

- Bootstrapping (BIOS/UEFI)
- Firmware
- Microcontroller programs
- Embedded systems

Why ROM is Needed?

A computer must start executing instructions **immediately after power ON**.
But RAM is empty at startup.

So the computer loads its first instructions from **ROM**, which stores:

- Boot loader
- Initialization programs
- Low-level hardware control code

These must be **non-volatile**, hence ROM.

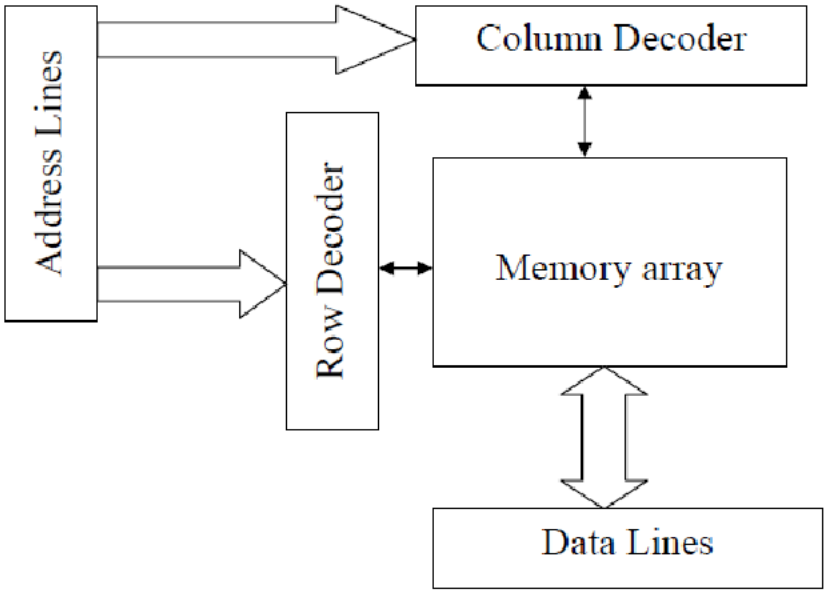
Basic Structure of ROM

A ROM chip consists of:

1. **Decoder** – selects memory location
2. **Memory Cell Array** – stores data (as 0s and 1s)

3. **Output buffers** – send data to the CPU

Basic block diagram:



The memory cell is typically a **diode/transistor connection** programmed to store either 0 or 1.

Types of Semiconductor ROM

There are **four major types**, and every exam expects a question on them.

(A) Masked ROM (MROM)

Permanent ROM manufactured with data burned into it using a mask.
Cannot be modified later.

How it works:

- During fabrication, metal connections are created or omitted → stores 0 or 1.
- Cheapest in bulk production.

Uses:

- Calculators
- Simple toys
- Old hardware

Advantages:

- Very cheap for mass production
- Very reliable

Disadvantages:

- Cannot be updated
- High cost for small quantity production

(B) PROM (Programmable ROM)

User can program it **once** using a PROM programmer.
Contains **fuse links**.

Programming is done by **blowing fuses**:

- Fuse intact → represents 1
- Fuse blown → represents 0

Advantages:

- Flexible (user can program once)

Disadvantages:

- Cannot rewrite
- Mistake = chip wasted

Used for:

- Small-scale device programming
- Microcontroller prototypes

(C) EPROM (Erasable Programmable ROM)

✔ Can be erased and reprogrammed.
✔ Erased by **ultraviolet (UV) light** for about 15–20 minutes.
✔ Has a **quartz window** on the chip.

Working:

- Stores bits using **floating-gate transistors**
- UV light removes charge → erases memory
- Can be reprogrammed electrically

Advantages:

- Reusable
- Good for development/testing

Disadvantages:

- Erasing takes long time
- Requires special UV eraser
- Cannot erase specific bytes (erases entire chip)

Used in:

- Early BIOS ROMs
- Educational microcontroller kits

(D) EEPROM (Electrically Erasable Programmable ROM)

✔ Can be erased **electrically**—no UV light needed
✔ Can erase **byte-by-byte**
✔ Very common today

Used in:

- BIOS/UEFI storage
- Microcontroller memory (Arduino, AVR, PIC)
- Smart cards

Advantages:

- Easy to rewrite
- No special UV equipment
- Partial erasing possible

Disadvantages:

- Limited write cycles (10^4–10^5)
- More expensive

(E) Flash Memory (EEPROM subtype)

Flash = modern improvement of EEPROM.

Erased in **blocks/pages** (not byte-by-byte)
MUCH faster than EEPROM
Used everywhere today:

- USB pen drives
- SSDs
- SD cards
- Mobile phone storage
- Camera memory

Comparison of ROM Types (Very Important Table)

Type	Programmable	Erasable	Physical Method	Reusability	Usage
Masked ROM	Factory only	No	Metal mask	No	Mass production
PROM	Once	No	Fuse links	No	Prototyping
EPROM	Yes	Yes	UV light	Yes	Early firmware
EEPROM	Yes	Yes (byte-wise)	Electrical	Yes	BIOS, controllers
Flash	Yes	Yes (block-wise)	Electrical	Yes	SSD, USB

ROM Cell Structure (Conceptual)

ROM uses a **diode matrix** or **transistor matrix**.

Example (diode matrix ROM):

```
      Columns (data bits)
        D0   D1   D2
     ---------------------
R0 |   X    0    X
R1 |   0    X    X
R2 |   X    X    0
```

"X" = diode present → output is 1
"0" = no diode → output is 0

A decoder selects a row; the column connections determine the data.

Applications of ROM

- Booting (BIOS, UEFI)
- Firmware in appliances
- Embedded system programs
- Microcontroller program storage
- Storing lookup tables (sine wave table, character font tables)
- Secure hardware (smart cards)

SUMMARY

- Semiconductor ROM is **non-volatile** memory.
- Used to store **firmware**, **boot code**, and **permanent programs**.
- Masked ROM is factory-programmed; PROM is programmable once.

- EPROM is UV-erasable; EEPROM is electrically erasable.
- Flash memory is a high-speed EEPROM used in SSD & USB drives.
- ROM stores bits using **fixed connections** (diode/transistor matrix).

Speed, Size and Cost of Memory

(One of the most fundamental topics of memory architecture)

Different types of memory (Registers, Cache, RAM, ROM, Hard disk, SSD) have **different speed, size, and cost characteristics**.
Understanding this helps explain **why memory hierarchy exists**.

The relationship is:

Faster memory → Smaller size → Higher cost per bit

Slower memory → Larger size → Lower cost per bit

This concept is used to design the **Memory Hierarchy**.

Memory Speed

Speed refers to how fast the memory can be accessed and supply data to the CPU.

Measured as:

- **Access time** (ns)
- **Latency**
- **Bandwidth**

From fastest → slowest:

1. **CPU Registers**
2. **Cache Memory (L1, L2, L3)**
3. **Main Memory (DRAM)**
4. **Secondary Storage (SSD, HDD)**

Example Speed (Approx):

Memory	Access Time
Register	< 1 ns
L1 Cache	1–2 ns
L2 Cache	3–10 ns
DRAM (RAM)	50–100 ns
SSD	0.1 ms
HDD	5–10 ms

Observation:
As speed decreases, access time increases dramatically.

Memory Size

Memory size = how much data can be stored.

Small memories are fast; large memories are slow.

From smallest → largest:

1. **Registers** (few KB)
2. **Cache (SRAM)** (KB to MB)
3. **Main Memory (DRAM)** (GB)
4. **Secondary memory** (SSD/HDD terabytes)

Example Sizes:

Memory Type	Typical Size
CPU Registers	< 1 KB
L1 Cache	32–128 KB
L2 Cache	256 KB – 2 MB
Main Memory (DRAM)	4 GB – 32 GB

SSD	256 GB – 4 TB
HDD	1 TB – 14 TB

Memory Cost (Cost per bit)

Faster memories use more complex technology → cost increases.

Cost comparison (approx, for understanding):

Memory	Cost per MB
SRAM (Cache)	Very high
DRAM (RAM)	Moderate
SSD	Low
HDD	Very low

Why?

- SRAM uses **6 transistors per bit** → costly
- DRAM uses **1 transistor + 1 capacitor** → cheaper
- HDD/SSD uses mass storage mechanisms → cheapest per bit

Relationship Between Speed, Size, Cost

This is the most important exam point.

Fast memory → Small size → High cost

Medium memory → Medium size → Medium cost

Slow memory → Large size → Low cost

This relationship leads to the **memory hierarchy**:

CPU

↓ (fastest)

Registers

↓

Cache (SRAM)

↓

Main Memory (DRAM)

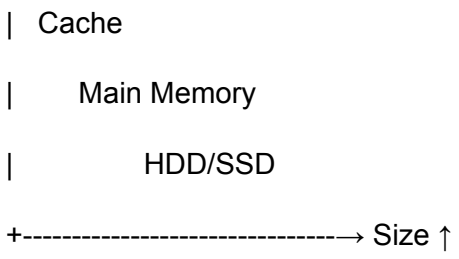
↓

SSD/HDD (slowest)

Graphical Representation

Speed ↑

| Registers



Faster memory is always near the CPU, but is small and expensive.

Why Memory Hierarchy Exists?

Because no single memory can provide:

- **Very fast speed**
- **Very large size**
- **Very low cost**

So computer designers combine different memories to get the best possible performance.

Example:

- Cache (fast but small) holds frequently used data
- RAM (big but slower) holds active programs
- SSD/HDD (very large, very cheap) holds long-term data

This gives:

- High speed
- Sufficient size
- Affordable cost

SUMMARY

- Memory systems vary in **speed**, **size**, and **cost per bit**.
- **Speed**: Registers > Cache (SRAM) > RAM (DRAM) > Secondary storage
- **Size**: Registers < Cache < RAM < SSD/HDD
- **Cost**: Registers > Cache (SRAM) > RAM (DRAM) > SSD/HDD
- Faster memories are **small and expensive**.
- Slower memories are **large and cheap**.
- Because no single memory can provide all three benefits, **memory hierarchy** is used.

CACHE MEMORY – DETAILED EXPLANATION

Cache memory is a **small, fast memory** placed between the CPU and main memory.

Purpose:
To reduce the average memory access time.

CPU first checks **cache** → if data is found → **Cache Hit**
If not found → **Cache Miss**, data is brought from RAM.

Cache Memory Basics

When CPU accesses memory, it references:

- **Main memory address** (large)
- Cache stores a **copy** of frequently used blocks

Cache works based on **locality of reference**:

- Temporal locality → recently used data is reused
- Spatial locality → nearby data is accessed

Cache Mapping Functions (VERY IMPORTANT)

Mapping = How a block of main memory is placed into cache.

There are **three mapping techniques**:

1. **Direct Mapping**
2. **Fully Associative Mapping**
3. **Set Associative Mapping**

All exams ask a question on this.
Let's explain each simply and clearly.

DIRECT MAPPING

Each block of main memory is mapped to **exactly one** cache line.

Formula:

Cache Line Number = (Main Memory Block Number) MOD (Number of Cache Lines)

Example:

If cache has 8 lines:
Block 0 → Line 0
Block 8 → Line 0
Block 16 → Line 0

Simple but causes more collisions.

Address Format:

| TAG | LINE | WORD |

Advantages:

- Simple hardware
- Fast
- Low cost

Disadvantages:

- More conflict misses
- Only one possible line → limited flexibility

FULLY ASSOCIATIVE MAPPING

A block can be placed **anywhere** in cache.

Address Format:

| TAG | WORD |

Features:

- No fixed location
- Cache controller searches **all** tags

Advantages:

- Very flexible

- Lowest chances of conflict misses

Disadvantages:

- Expensive hardware
- Slower tag comparison (needs associative search)

Used in:

- TLB (Translation Lookaside Buffer)
- Small caches

SET ASSOCIATIVE MAPPING

This is a **combination of Direct + Associative** mapping.
Most modern processors use this.

Structure:

Cache is divided into **sets**, and each set has **multiple lines**.

Examples:

- 2-way set associative → 2 lines per set
- 4-way set associative → 4 lines per set
- 8-way set associative → 8 lines per set

Address Format:

| TAG | SET | WORD |

How it works:

- Memory block is mapped to **one set** (like direct mapping)

- Within that set, block can be placed in **any line** (like associative)

Advantages:

- Lower miss rate
- Good balance of cost and performance

Disadvantages:

- More complex than direct mapping

Comparison Table (Very Important)

Mapping	Flexibility	Conflict Misses	Hardware Cost	Speed
Direct	Low	High	Low	Fast
Associative	High	Very low	High	Slow
Set Associative	Medium	Low	Medium	Medium

This table is enough for 3–4 marks.

Cache Replacement Algorithms

Used **only in Associative and Set-Associative** mapping.
(Not needed in Direct Mapping because line is fixed.)

These algorithms decide **which block to remove from cache** when cache is full.

The most important algorithms are:

1. Least Recently Used (LRU)

Evicts the block that has **not been used for the longest time**.

Idea:

- Based on temporal locality
- The block least recently used is least likely to be needed soon

Advantage:

- Very effective in practice

Disadvantage:

- Expensive to maintain (hardware counters needed)

2. FIFO (First-In First-Out)

Evicts the block that has been **in cache the longest**, regardless of usage.

Advantage:

- Simple to implement

Disadvantage:

- Does not consider recent use → may remove frequently used block

3. LFU (Least Frequently Used)

Evicts block that is **used least number of times**.

Advantage:

- Keeps frequently used data in cache

Disadvantages:

- Needs counters
- May retain old frequently used blocks whose frequency gets outdated

4. Random Replacement

Evicts a random block.

Advantage:

- Very simple
- Requires no tracking

Disadvantage:

- Performance varies

Used in some hardware caches where simplicity is important.

Cache Hit and Miss

Cache Hit:

Data found in cache
→ Fast access
→ Hit ratio improves
→ Works well with CPU

Cache Miss:

Data not found in cache
→ Must fetch from main memory
→ Slower

Miss penalty = time difference between cache access and main memory access.

Types of Misses:

1. **Compulsory** – first access to block
2. **Capacity** – cache cannot hold everything
3. **Conflict** – multiple blocks map to same line (direct-mapped)

SUMMARY

Mapping Functions:

- **Direct Mapping**: One block → one line
- **Associative Mapping**: Block → any line
- **Set Associative Mapping**: Block → one set, any line within set

Replacement Algorithms:

- **LRU**: Remove least recently used
- **FIFO**: Remove oldest
- **LFU**: Remove least frequently used
- **Random**: Remove random line

Cache improves performance by:

- Using locality of reference

- Reducing average memory access time

VIRTUAL MEMORY (VM) – *What and Why?*

Virtual memory is a memory management technique where the **system gives an illusion of having more memory than physically present**.

Definition:

Virtual memory is a technique where secondary memory (disk/SSD) is used as an extension of main memory, allowing programs to use a larger logical address space than physical RAM.

Purpose of Virtual Memory:

- To run large programs on small RAM
- To allow MULTIPLE programs to run simultaneously
- To provide isolation/protection between processes
- To simplify memory allocation

How Virtual Memory Works?

- Each program thinks it has a large, continuous memory → **Virtual Address Space**
- Physical memory (RAM) contains only the needed parts
- Remaining parts are stored on the disk (swap space / page file)

When a page needed by CPU is not in RAM → **Page Fault** → It is loaded from disk.

Virtual vs Physical Address

Type	Meaning

Virtual Address (VA)	Generated by CPU; belongs to process
Physical Address (PA)	Actual location in RAM

A special hardware called **MMU (Memory Management Unit)** translates VA → PA.

Techniques of Virtual Memory

Virtual memory is implemented using either:

Paging

Segmentation

Combination (Paging + Segmentation: used by many modern systems)

PAGING (Important)

Paging divides both **virtual memory** and **physical memory** into **fixed-size blocks**.

In Virtual Memory → blocks are called Pages

In Physical Memory → blocks are called Frames

Page size = Frame size (4 KB, 8 KB commonly)

Basic Idea of Paging

- CPU generates a **virtual address**: (page number + offset)
- MMU translates page number → physical frame number using **Page Table**

Virtual Address Format:

| Page Number | Offset |

Physical Address Format:

| Frame Number | Offset |

Page Table

A page table maps:

Virtual Page Number → Physical Frame Number

Each entry contains:

- Frame number
- Valid/invalid bit
- Protection bits (read/write)
- Dirty bit (modified)

Page Fault

Occurs when:

- CPU accesses a page that is NOT in RAM

What happens?

1. Page fault happens
2. OS loads the page from disk into RAM
3. Page table updated
4. Instruction restarts

Page faults are **slow** because disk access is slow.

PART 2: SEGMENTATION

Segmentation divides memory based on **logical program structure**, NOT fixed size.

Segment = variable-sized memory block

Examples of segments:

- Code
- Data
- Stack
- Heap
- Functions, arrays, objects

Each segment can be of different length.

Segment Table

Contains:

- Base address (starting physical address)
- Limit (size of segment)

Virtual Address Format:

| Segment Number | Offset |

Physical Address:

Physical Address = Base + Offset

Segment table ensures the offset does not exceed the limit.
If offset > limit → segmentation fault (memory violation).

Paging vs Segmentation (Imp)

Feature	Paging	Segmentation
Size	Fixed-size pages	Variable-sized segments
Purpose	Avoid fragmentation	Reflect logical program structure
Address	Page# + Offset	Segment# + Offset
Fragmentation	Internal fragmentation	External fragmentation
Used in	OS memory management	Logical organization of program
Mapping table	Page table	Segment table

Combined Paging + Segmentation (Modern OS)

Most operating systems (Linux, Windows) use **both**:

- Segmentation for **logical view**
- Paging for **physical memory allocation**

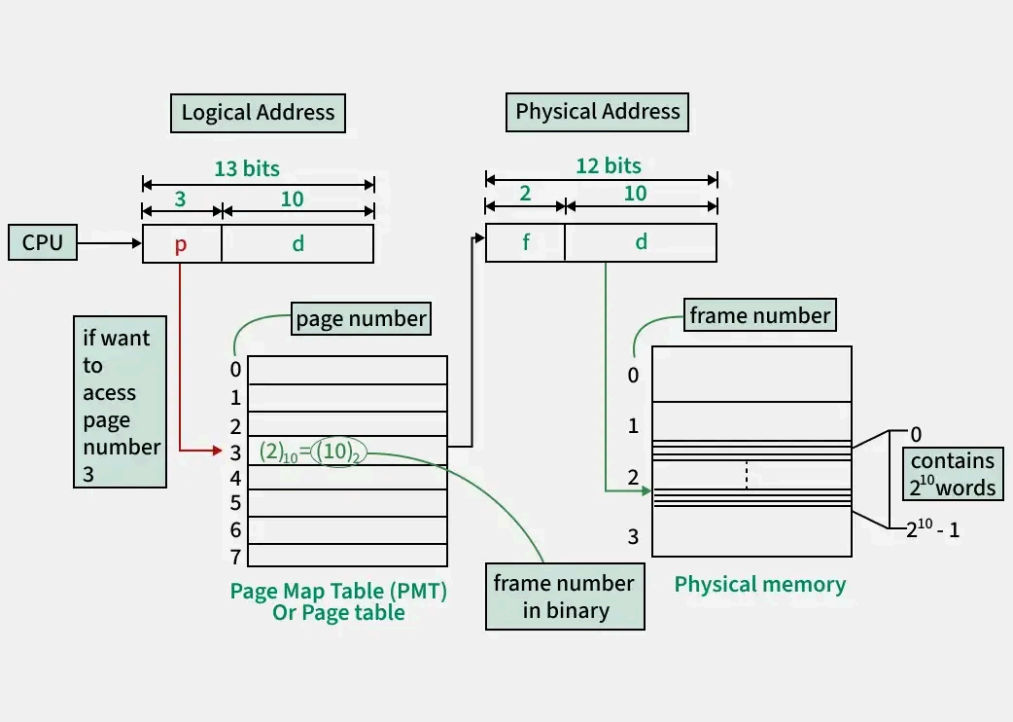
Steps:

1. CPU generates: Segment number + offset
2. Segment table gives **page table** of that segment
3. Paging works on those pages

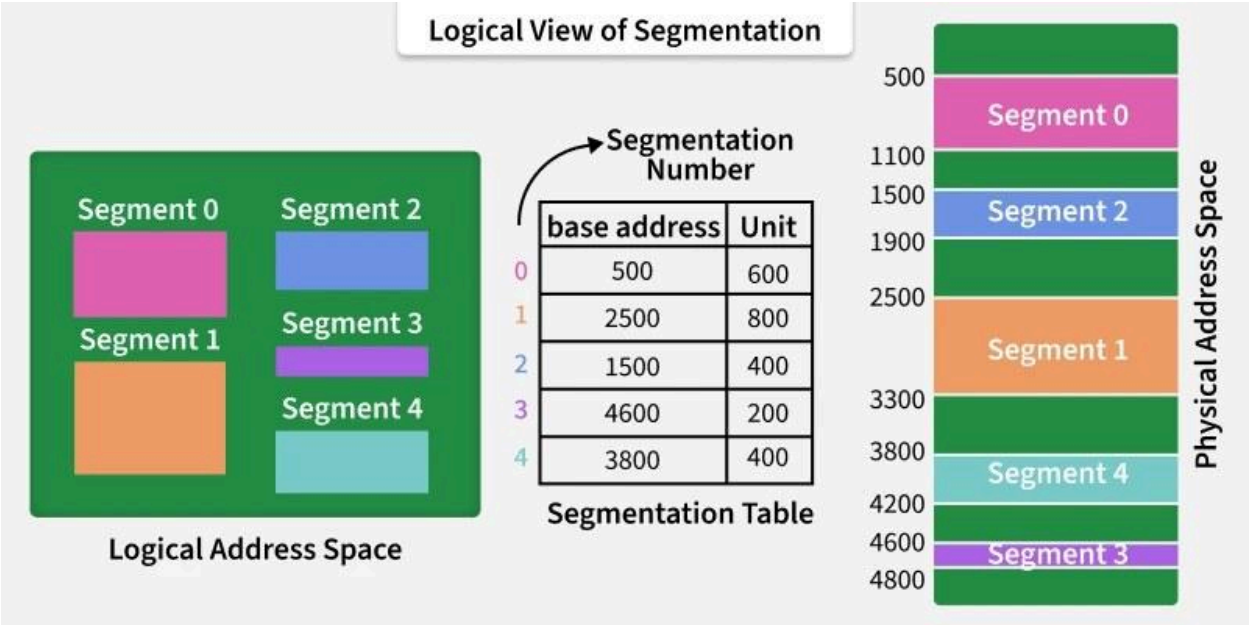
This provides:
✔ Logical protection (segments)
✔ No fragmentation (paging)

EXAM DIAGRAMS (Very Important)

1. Paging Hardware Diagram



2.Segmentation Hardware Diagram



SUMMARY

Virtual Memory

- Gives illusion of large memory
- Uses disk as extension of RAM
- Managed by OS and MMU

Paging

- Memory divided into **fixed-size pages/frames**
- Uses **page table** for translation
- Causes **page faults** if page not in RAM

Segmentation

- Memory divided into **logical segments**

- Uses **base** and **limit**
- Helps protection + modularity

Combined Paging + Segmentation

- Used in most modern OS
- Segmentation for logical structure
- Paging for physical allocation