Memory Hierarchy

DSC 315: Computer Organization & Operating Systems

Dr. Laltu Sardar

School of Data Science,
Indian Institute of Science Education and Research Thiruvananthapuram (IISER TVM)

February 5, 2026

Memory hierarchy

IISER THIRUVANANTHAPURAM

Why Do We Need Memory Hierarchy?

- Computers need memory that is
  - Very fast
  - Very large
  - Very cheap
- Unfortunately, no single memory technology satisfies all three
- Solution: organize memory in levels called **memory hierarchy**

Bookshelf Example: The Setup

Imagine you are studying for an exam.

- Some books are used very frequently
- Some books are used occasionally
- Some books are rarely used

To study efficiently, you organize books based on how often you need them.

Bookshelf Example: Different Storage Places

Place	Distance	Speed of Access
On your desk	Very close	Very fast
Nearby shelf	A little far	Fast
Cupboard or store room	Far away	Slow

You keep important books closer to reduce effort and time.

Mapping Bookshelf to Computer Memory

Bookshelf	Computer	Access Speed	Cost per Bit	Material / Technology
Desk	Registers	Fastest	Very high	Flip-flops (CMOS)
Nearby shelf	Cache	Very fast	High	SRAM (Static RAM)
Cupboard	Main Memory (RAM)	Slower	Moderate	DRAM (Dynamic RAM)
Store room	Secondary Storage (Disk)	Slowest	Very low	Magnetic disk / Flash memory

Mapping Bookshelf to Computer Memory

Bookshelf	Computer	Access Speed	Cost per Bit	Material / Technology
Desk	Registers	Fastest	Very high	Flip-flops (CMOS)
Nearby shelf	Cache	Very fast	High	SRAM (Static RAM)
Cupboard	Main Memory (RAM)	Slower	Moderate	DRAM (Dynamic RAM)
Store room	Secondary Storage (Disk)	Slowest	Very low	Magnetic disk / Flash memory

Cache Level	Location	Access Speed	Cost per Bit	Technology Used
L1 Cache	Inside CPU core	Fastest	Very high	SRAM (small, highly optimized)
L2 Cache	Inside CPU chip	Very fast	High	SRAM
L3 Cache	Shared on CPU chip	Fast	Moderate	SRAM (larger, slower)

Key Idea from the Bookshelf Example

Observation

You do not keep all books on your desk.

- Only frequently used books stay close
- Rarely used books stay farther away
- This saves space, cost, and effort

The same idea is used in memory hierarchy.

How Does the Computer Decide What to Keep Close?

- Programs do not access memory randomly
- They tend to reuse the same data and instructions
- They also access nearby memory locations

This behavior is explained by the **Principle of Locality**.

Principle of Locality

Definition

Programs tend to access a small portion of memory repeatedly over a short period of time.

There are two main types of locality:

- Temporal Locality
- Spatial Locality

Temporal Locality

Meaning

If a memory location is accessed once, it is likely to be accessed again soon.

- Loop variables
- Repeated instructions
- Frequently used data

Like keeping your most used book always on your desk.

Spatial Locality

Meaning

If a memory location is accessed, nearby locations are likely to be accessed soon.

- Array elements
- Sequential instruction execution

Like taking several pages from the same book at once.

Why Locality Makes Memory Hierarchy Effective

- Cache stores recently and nearby used data
- Most accesses hit fast memory levels
- Slow memory accesses are minimized

Result

High performance with low cost

block, hit, miss, hit rate, miss rate, hit time, and miss penalty

Why Do We Need These Cache Terms?

- Cache does not store data one word at a time
- Cache performance depends on how often data is found
- We need precise terms to measure efficiency

These terms help us analyze and compare memory systems.

Concept of a Block

Block (or Cache Line)

A block is the minimum unit of data transferred between main memory and cache.

- A block contains multiple contiguous bytes or words
- Example: 32 bytes or 64 bytes per block
- Exploits spatial locality

Why Blocks Instead of Single Words?

- Programs access nearby memory locations
- Fetching extra nearby data is usually beneficial
- Reduces number of memory accesses

Key Idea

One miss can bring many useful data items.

Cache Hit and Cache Miss

- **Cache Hit**: Requested data is found in cache
- **Cache Miss**: Requested data is not found in cache

Result

Hit is fast, miss is expensive.

Hit Rate and Miss Rate

Hit Rate

Fraction of memory accesses that result in a cache hit.

$$\text{Hit Rate} = \frac{\text{Number of Hits}}{\text{Total Memory Accesses}}$$

Miss Rate

Fraction of memory accesses that result in a cache miss.

$$\text{Miss Rate} = 1 - \text{Hit Rate}$$

Hit Time

Hit Time

Time required to access data from cache when there is a hit.

- Includes cache access time
- Includes tag comparison time
- Typically very small

Miss Penalty

Miss Penalty

Extra time required to fetch the block from the next lower memory level and deliver it to the CPU.

- Access main memory
- Transfer block to cache
- Resume CPU execution

Why Miss Penalty Is Large

- Main memory is much slower than cache
- Entire block must be transferred
- CPU may have to stall

Observation

Reducing miss penalty is as important as improving hit rate.

Putting It All Together

Average Memory Access Time (AMAT)

$$\text{AMAT} = \text{Hit Time} + (\text{Miss Rate} \times \text{Miss Penalty})$$

This equation summarizes cache performance.

Average Memory Access Time and Required Hit Ratio

- Cache access time = 4 cycles
- Main memory access time = 100 cycles
- Processor always accesses cache first

Problem: Find required Hit Ratio to get AMAT 5 cycles

Average Memory Access Time

$$\text{AMAT} = \text{Hit Time} + (1 - \text{Hit Ratio}) \times \text{Miss Penalty}$$

Average Memory Access Time and Required Hit Ratio

Break-Even Analysis

Main memory only access gives:

$$\text{AMAT} = 100$$

With cache:

$$100 = 4 + (1 - \text{HR}) \times 100$$

$$\Rightarrow \text{HR} = 0.04 = 4\%$$

Achieving AMAT = 5 Cycles

$5 = 4 + (1 - \text{HR}) \times 100 \implies \Rightarrow \text{HR} = 0.99 = 99\%$

Memory Technologies

Section 5.2

Memory Technologies

Different levels of memory hierarchy use different technologies.

- Each technology is optimized for a specific goal
- Trade offs exist between speed, cost, density, and power
- No single technology can satisfy all requirements

We now study the major memory technologies used in computers.

Register Technology

Registers

Registers are the fastest storage elements inside the CPU.

- Implemented using flip flops
- Located inside the processor core
- Very small in size
- Extremely fast access

Usage

Operands, addresses, and intermediate results

SRAM Technology

Static Random Access Memory (SRAM)

SRAM stores data as long as power is supplied.

- Built using bistable circuits
- No refresh required
- Faster than DRAM
- Lower density and higher cost

Usage

Cache memory (L1, L2, L3)

DRAM Technology

Dynamic Random Access Memory (DRAM)

DRAM stores data as charge in capacitors.

- Requires periodic refresh
- Slower than SRAM
- Higher density
- Lower cost per bit

Usage

Main memory (RAM)

Flash Memory Technology

Flash Memory

Flash is a non volatile semiconductor memory.

- Retains data without power
- Faster than magnetic disks
- Limited write endurance
- Used in solid state storage

Usage

SSDs, pen drives, memory cards

Magnetic Disk Technology

Magnetic Disk

Magnetic disks store data using magnetic coating.

- Mechanical moving parts
- Very large storage capacity
- Very slow compared to RAM
- Lowest cost per bit

Usage

Hard disk drives

Summary of Memory Technologies

Technology	Speed	Cost	Volatility	Usage
Registers	Fastest	Very high	Volatile	CPU storage
SRAM	Very fast	High	Volatile	Cache memory
DRAM	Moderate	Medium	Volatile	Main memory
Flash	Fast	Low	Non volatile	SSDs
Magnetic Disk	Slowest	Very low	Non volatile	Secondary storage

Basics of Cache

Chapter 5.3

What Is Cache Memory?

- Cache is a small, fast memory placed between CPU and main memory
- Stores recently used data and instructions
- Exploits temporal and spatial locality

Goal

Reduce average memory access time

Why Do We Need Cache?

- CPU speed is much higher than main memory speed
- Direct access to RAM slows down execution
- Cache bridges the speed gap

Observation

Most memory accesses go to a small part of the program

Direct Mapped Cache

Direct Mapped Cache

Each block of main memory maps to exactly one cache line.

- Simple hardware design
- Fast access
- Higher conflict misses

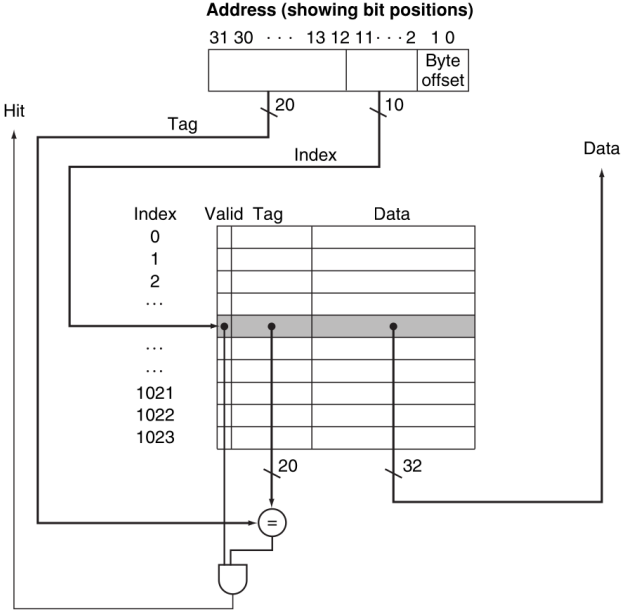
Structure of a Direct Mapped Cache

Each cache line contains:

- Valid bit
- Tag
- Data block

Key Idea

Index selects the cache line, tag verifies correctness


Address (showing bit positions)
31 30 · · · 13 12 11 · · · 2 1 0
Byte offset
20
10
Hit
Tag
Index
Data
Index
Valid
Tag
Data
0
1
2
· · ·
· · ·
· · ·
1021
1022
1023
20
32
=

Memory Address Format

A memory address is divided into:

- Tag
- Index
- Block offset

$$\text{Memory Address} = \text{Tag} \mid \text{Index} \mid \text{Offset}$$

Each field has a specific role in cache access.

Tag Size in a Direct-Mapped Cache

- 32-bit byte addresses
- A direct-mapped cache
- The cache size is 2^n blocks, so n bits are used for the index
- The block size is 2^m words (2^{m+2} bytes), so
  - m bits are used for the word within the block
  - 2 bits are used for the byte offset

Tag Field Size

The size of the tag field is

$$32 - (n + m + 2)$$

How Cache Access Works

1. CPU generates a memory address
2. Index bits select a cache line
3. Valid bit is checked
4. Tag is compared
5. If match, data is returned

Result

This is a cache hit

Cache Hit and Cache Miss

- Cache hit: valid bit is 1 and tag matches
- Cache miss: valid bit is 0 or tag mismatch

Consequence of Miss

Data must be fetched from main memory

Handling a Cache Miss

On a cache miss:

1. Required block is read from main memory
2. Block is placed in the indexed cache line
3. Tag is updated
4. Valid bit is set to 1
5. CPU resumes execution

Handling Write Operations

Cache must also handle write requests correctly.

Two common policies:

- Write through
- Write back

Write Through Policy

- Data is written to cache and main memory simultaneously
- Simple and consistent
- Higher memory traffic

Usage

Often combined with a write buffer

Write Back Policy

- Data is written only to cache
- Main memory updated later
- Requires a dirty bit
- Reduces memory traffic

Trade off

More complex hardware

Final Notes

- Cache improves performance by exploiting locality
- Direct mapped cache is simple and fast
- Address is divided into tag, index, and offset
- Cache misses and writes require careful handling

1. Complete the adding unsigned integer
2. While loop of A[i] == i; ??
3. Writing function ??
4. Instruction format; reason
5. Addressing modes ??
6. starting a program.??
7. Possible number of supporting memory; looking at the format.

Dr. Laltu Sardar, Assistant Professor, IISER Thiruvananthapuram
laltu.sardar@iisertvm.ac.in, laltu.sardar.crypto@gmail.com
Course webpage: https://laltu-sardar.github.io/courses/corgos_2026.html.


Carl Hamacher, Zvonko Vranesic, and Safwat Zaky.
Computer Organization.
McGraw-Hill, 6th edition, 2012.


John L. Hennessy and David A. Patterson.
Computer Architecture: A Quantitative Approach.
Morgan Kaufmann, 6th edition, 2017.


Vincent P. Heuring and Harry F. Jordan.
Computer System Design and Architecture.
Pearson, 2nd edition, 2007.


David A. Patterson and John L. Hennessy.
Computer Organization and Design: The Hardware/Software Interface.
Morgan Kaufmann, 4th edition, 2014.


William Stallings.
Computer Organization and Architecture: Designing for Performance.
Pearson, 10th edition, 2016.