Subject :
Computer Organization & Architecture

Designed by :
Mr. Shashank Sharma

Designation :
Assistant Professor

UNIT 4
Memory Organization

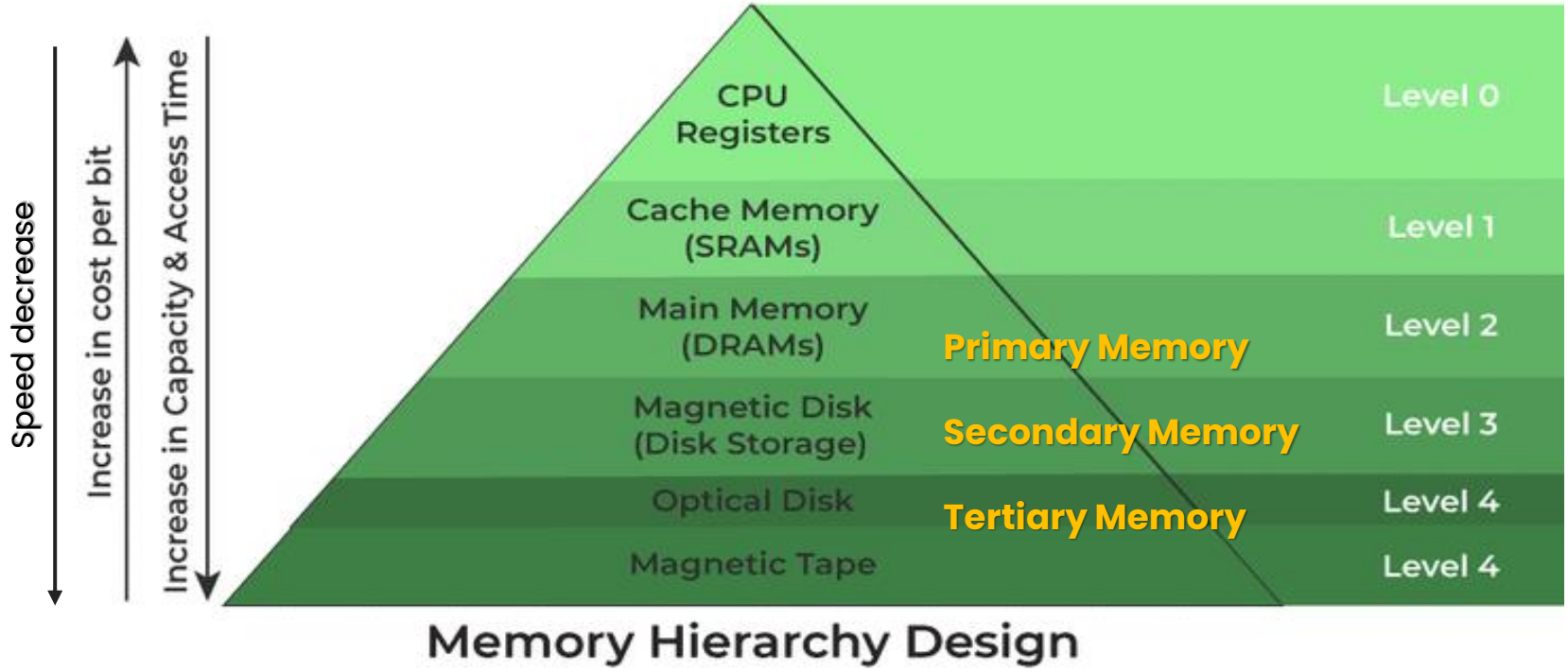
Syllabus

- **Main memory:** RAM, ROM
- **Secondary Memory:** Magnetic Tape, Disk, Optical Storage
- **Cache Memory:** Cache Structure and Design, Mapping Scheme, Replacement Algorithm, Improving Cache Performance, Virtual Memory, memory management hardware

Memory Hierarchy

Memory hierarchy in a computer system refers to the organization of various types of memory components based on their proximity to the CPU and their speed and size characteristics.
The memory hierarchy is designed to optimize the performance of the computer system by providing fast access to frequently used data and instructions while minimizing costs and power consumption.

Here's a typical memory hierarchy in a computer system, from the fastest and smallest to the slowest and largest:



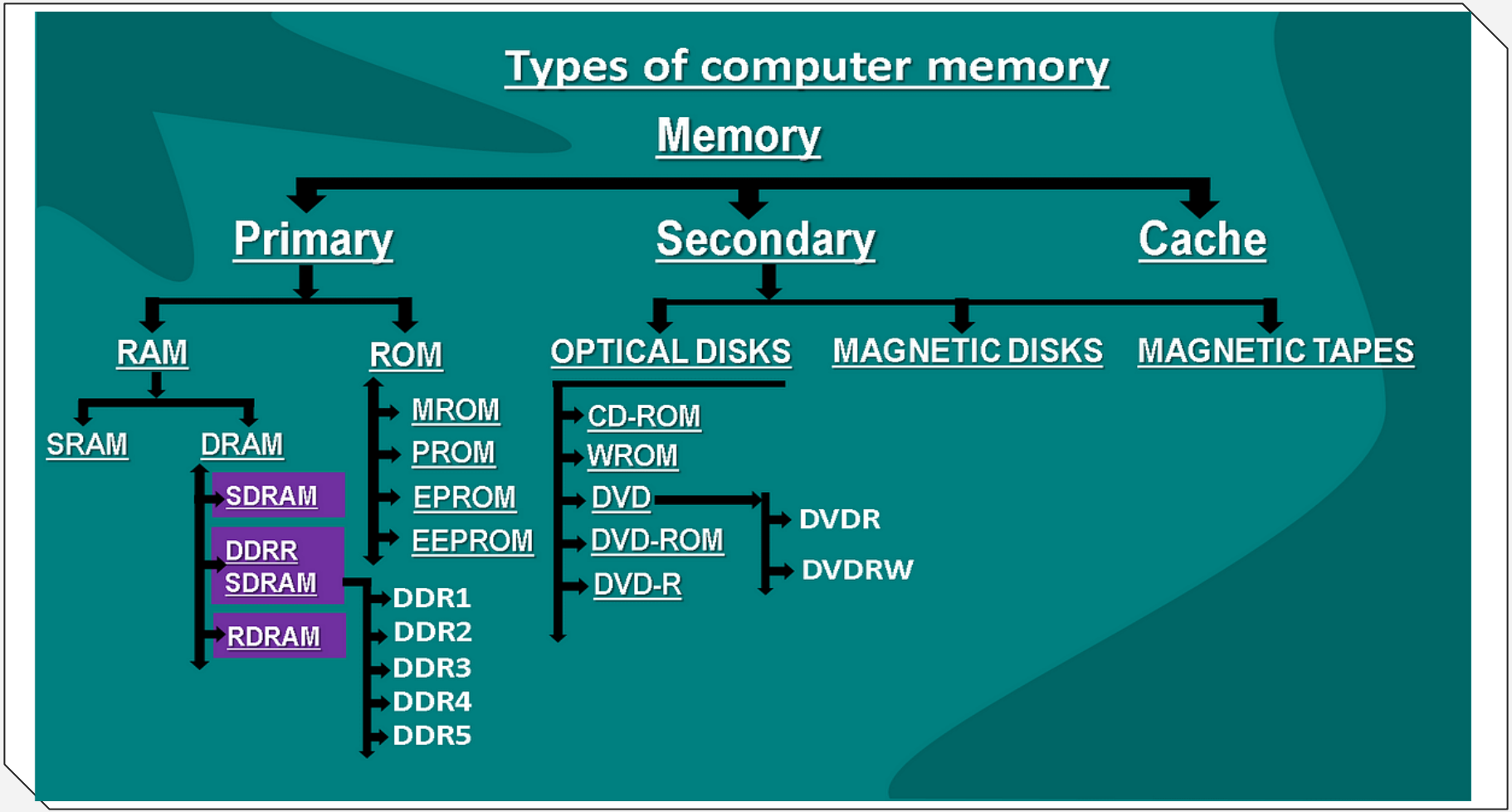
Registers: Registers are small, high-speed storage locations directly accessible by the CPU. They hold data and instructions that are currently being processed by the CPU. Registers have the fastest access time but are limited in size and expensive to manufacture.

***Cache Memory*:** Cache memory sits between the CPU and the main memory (RAM) and serves as a buffer to store frequently accessed data and instructions. It is faster than main memory but slower than registers. Modern CPUs typically have multiple levels of cache (L1, L2, L3), with L1 being the smallest and fastest and L3 being larger but slower. Cache memory helps reduce the average time to access data by storing copies of frequently used data closer to the CPU.

Main Memory (RAM): RAM (Random Access Memory) is the primary volatile memory of a computer system. It holds data and instructions that are actively being used by the CPU. While larger than cache memory, RAM is slower and has longer access times. However, it is still much faster than secondary storage devices like hard disk drives (HDDs) or solid-state drives (SSDs).

Secondary Storage: Secondary storage devices, such as HDDs and SSDs, provide long-term storage for data and programs that are not actively being used by the CPU. These storage devices offer much larger storage capacity compared to RAM but are significantly slower in terms of access times. Data needs to be transferred between secondary storage and RAM before it can be processed by the CPU.

Tertiary Storage: Tertiary storage refers to offline storage devices such as magnetic tapes or optical discs. These devices offer even larger storage capacities but are much slower than secondary storage devices. They are typically used for archival purposes rather than regular data access


Types of computer memory
Memory
Primary
Secondary
Cache
RAM
ROM
OPTICAL DISKS
MAGNETIC DISKS
MAGNETIC TAPES
SRAM
DRAM
SDRAM
DDRR SDRAM
RDRAM
MROM
PROM
EPROM
EEPROM
DDR1
DDR2
DDR3
DDR4
DDR5
CD-ROM
WROM
DVD
DVD-ROM
DVD-R
DVDR
DVDRW

Main Memory

Main memory, also known as ***primary memory*** or ***internal memory***, is a crucial component of a computer system. It is the primary storage area where data and instructions are temporarily stored during program execution. Main memory is directly accessible to the CPU (Central Processing Unit) and plays a vital role in the functioning of the computer.

Key characteristics of main memory include:

Accessibility: Main memory is directly accessible by the CPU. This allows for quick retrieval and storage of data and instructions needed for program execution.

Volatility: Main memory is volatile, meaning its contents are lost when the power is turned off. As a result, data stored in main memory needs to be periodically saved to non-volatile storage devices, such as hard drives or solid-state drives (SSDs), to prevent data loss.

Speed: Main memory is faster than secondary storage devices like hard drives or SSDs. This high-speed access allows for rapid data retrieval and execution of instructions by the CPU.

Capacity: Main memory typically has a limited capacity compared to secondary storage devices. However, advancements in technology have led to increases in main memory capacity over time.

Overall, main memory serves as a workspace for the CPU, providing the necessary storage for data and instructions required for program execution.

RAM (Random Access Memory):

RAM is a type of main memory that the CPU uses to store data and instructions temporarily while a computer is running.

It is called "random access" because the CPU can read from or write to any location in RAM, regardless of the current data stored there.

RAM is volatile, means its contents are lost when the power is turned off. It requires a constant power supply to maintain data.

***Role*:** RAM serves as the primary workspace for the CPU, providing temporary storage for data and instructions that are actively being used by programs.

Interaction: When a program is executed, its code and data are loaded into RAM from secondary storage devices like hard drives or SSDs. The CPU can then quickly access this data from RAM for processing. As programs run and new data is generated, RAM is continuously updated and accessed by the CPU.

Facilitating Data Processing: RAM's fast access speeds enable the CPU to quickly retrieve and manipulate data, facilitating smooth data processing and program execution.

Storage: RAM holds data temporarily; its contents are volatile and are lost when the power is turned off. To preserve data, it's periodically saved to secondary storage devices.

Types of RAM (Random Access Memory):

SRAM (Static RAM):

SRAM is a faster and more expensive type of RAM compared to DRAM.

Operation: SRAM stores each bit of data using a flip-flop circuit, which retains its state as long as power is supplied to the circuit. There is no need for periodic refreshing.

Characteristics: SRAM offers faster access speeds and consumes less power than DRAM. It is commonly used in cache memory due to its high-speed operation.

DRAM (Dynamic RAM):

DRAM is the most common type of RAM used in modern computer systems.
Operation: DRAM stores each bit of data in a separate capacitor within an integrated circuit. These capacitors need to be refreshed periodically to maintain the stored data, hence the term "dynamic.“

Characteristics: DRAM is less expensive and denser than other types of RAM but has slower access speeds. It requires more power and generates more heat due to the constant refreshing process.

DRAM further divided into:

- SDR RAM - Single Data Rate Random Access Memory
- DDR RAM - Double Data Rate Random Access Memory
- SDRAM - Synchronous Dynamic Random Access Memory
- (SDR) SDRAM – (Single Data Rate) Synchronous Dynamic Random Access Memory
- (DDR) SDRAM – (Double Data Rate) Synchronous Dynamic Random Access Memory
- RDRAM - Rambus Dynamic Random Access Memory.

ROM (Read-Only Memory)

ROM is another type of main memory that stores data and instructions permanently.

Unlike RAM, ROM is non-volatile, means its contents are retained even when the power is turned off.

ROM is used to store firmware or low-level software that is essential for booting up the computer and initializing hardware.

Examples of ROM include BIOS (Basic Input/Output System) and firmware embedded in devices like smartphones and routers.

Types of ROM (Read-Only Memory):

Mask ROM (MROM):

Description: Mask ROM is manufactured with data already stored in it during the fabrication process. The data is physically encoded onto the ROM chip during manufacturing.

Characteristics: The data in mask ROM cannot be modified or erased after fabrication, making it permanent and unchangeable.

Applications: Mask ROM is commonly used for storing firmware that does not need to be updated or modified frequently, such as boot firmware in embedded systems.

Programmable ROM (PROM):

Description: PROM is a type of ROM that can be programmed with user-defined data after manufacturing.

Characteristics: The programming of PROM is typically done by burning fuses or using a similar irreversible process, making the programmed data permanent.

Applications: PROM was commonly used in the past for storing firmware or software that might need occasional updates but not frequent changes.

Erasable Programmable ROM (EPROM):

Description: EPROM is a type of ROM that can be erased and reprogrammed multiple times using ultraviolet (UV) light.

Characteristics: EPROM chips have a transparent quartz window on top through which UV light can penetrate to erase the data. Once erased, the chip can be reprogrammed with new data.

Applications: EPROM was widely used for firmware storage in the 1970s and 1980s, particularly in early microcomputers and gaming consoles.

Electrically Erasable Programmable ROM (EEPROM or E²PROM):

Description: EEPROM is a type of ROM that can be electrically erased and reprogrammed.

Characteristics: Unlike EPROM, EEPROM does not require exposure to UV light for erasure. Instead, it can be erased and reprogrammed electrically, which makes it more convenient for updating data.

Applications: EEPROM is used in many modern electronic devices for storing configuration data, BIOS settings, and other types of data that may need to be updated periodically.

Flash Memory

Flash memory is a type of EEPROM that can be erased and reprogrammed in large blocks.

Flash memory allows for faster erasure and programming compared to traditional EEPROM.

It is commonly used in USB flash drives, memory cards, solid-state drives (SSDs), and other storage devices.

Flash memory is widely used for storing firmware, operating systems, and user data in various electronic devices.

Flash memory is also considered secondary storage

Secondary memory

Secondary memory, also referred to as secondary storage, is a type of computer memory that is used to store data and programs that can be accessed even after the computer is turned off.
Unlike primary memory, which is volatile and temporary (meaning it loses its contents when the power goes out), secondary memory is non-volatile and can store data for extended periods of time.

Think of it like this: primary memory (RAM) is like your short-term memory, while secondary memory (hard drive) is like your long-term memory. You use short-term memory to hold onto information you're currently working with, but you use long-term memory to store information for later retrieval.

Here are some of the key characteristics of secondary memory:

- ***Non-volatile:*** Data is not lost when the computer is turned off.
- ***Slower access time:*** It takes longer for the CPU to access data in secondary memory than in primary memory.
- ***Larger storage capacity:*** Secondary memory devices can store much more data than primary memory.
- ***Lower cost per unit of storage:*** Secondary memory is generally less expensive than primary memory.

Common examples of secondary memory devices include:

- Hard disk drives (HDDs)
- Solid-state drives (SSDs)
- Optical media (CDs, DVDs, Blu-ray discs)
- Magnetic tape
- Flash memory (USB drives, SD cards)

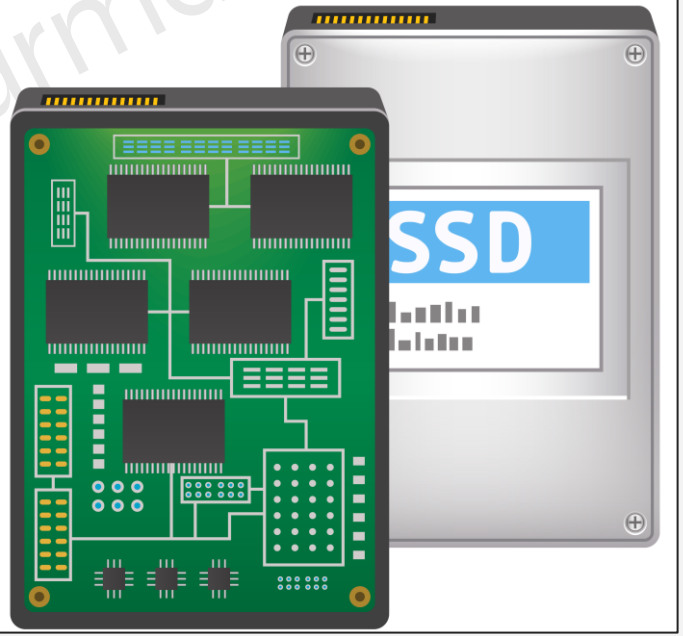
Secondary memory is essential for computers because it allows us to store data and programs that are too large or too important to keep in primary memory. For example, you would use secondary memory to store your operating system, applications, documents, photos, music, and videos.

Difference between magnetic disks and magnetic tapes

Features	Magnetic Disk	Magnetic Tape
Physical Form	Stack of spinning platters	Long, thin plastic ribbon
Data Access	Random access	Sequential access
Speed	Fast	Slow
Capacity	High (per unit volume)	Very high
Cost	High	Low (per unit volume)
Reliability	Susceptible to physical damage	Less prone to physical damage (but can degrade)
Applications	Primary & Secondary Storage	Backup & Archiving
Examples	Hard Disk Drive (HDD), SSD Hybrid Drive, External Hard Drive	Cassette Tape (audio), Video Cassette (VHS), Data Backup Tape

Hard Disk

SSD(Solid state drive)



Magnetic Tape:
Audio cassette

Magnetic Tape:
Video cassette

Data Backup Magnetic Tape

Optical storage

Optical storage refers to a method of storing digital data using light. It involves the use of optical discs, such as CDs (Compact Discs), DVDs (Digital Versatile Discs), and Blu-ray discs. These discs have a reflective surface that stores data in a binary format (0s and 1s).

1.<u>Basic Principle</u>: At its core, optical storage relies on the principle of light reflection. Data is stored on a special type of disc, such as a CD, DVD, or Blu-ray disc, which has a reflective surface. The data is represented as a series of microscopic pits and flat areas called lands.

2.<u>Data Encoding</u>: In optical storage, digital data is encoded onto the surface of the disc in binary form (0s and 1s). This encoding is achieved by physically altering the reflective surface to create patterns of pits and lands. These patterns represent the binary data.

3. ***Reading Mechanism***: To read the data stored on an optical disc, a laser beam is used. The laser is directed onto the surface of the disc, and its reflection is detected by a sensor. As the laser beam encounters the pits and lands on the disc's surface, it reflects differently. This difference in reflection is interpreted by the sensor as binary data.

1.Types of Optical Discs:

1. ***Compact Disc (CD)***: Introduced in the 1980s, CDs were the first widely adopted optical storage medium. They typically store around 700 MB of data and are commonly used for music, software, and data storage.

2. ***Digital Versatile Disc (DVD)***: DVDs have a higher storage capacity than CDs, typically ranging from 4.7 GB to 9 GB. They are often used for movies, software distribution, and data backup.

3. ***Blu-ray Disc***: Blu-ray discs offer even greater storage capacity, typically ranging from 25 GB to 100 GB. They are commonly used for high-definition movies, video games, and large data backups.

1.Advantages:

1. *Portability*: Optical discs are compact and easy to transport.

2. *Durability*: Optical discs are relatively resistant to physical damage and can last for many years if handled properly.

3. *Cost-effectiveness*: Optical discs are often inexpensive to produce, making them a cost-effective storage solution for distributing large amounts of data.

1.Applications:

1. *Distribution of Software*: Optical discs have long been used to distribute software, including operating systems, applications, and games.

2. *Entertainment*: DVDs and Blu-ray discs are widely used for storing movies, TV shows, and other multimedia content.

3. *Archiving*: Optical discs are used for long-term storage and archiving of important data, such as medical records, historical documents, and scientific research.

Cache Memory

Cache memory is a small, high-speed type of volatile computer memory that provides high-speed data access to a processor and stores frequently used computer programs, applications, and data.

It acts as a buffer between the CPU and the main memory (RAM).

The primary purpose of cache memory is to store copies of frequently accessed data from main memory to reduce the time it takes for the CPU to access that data.

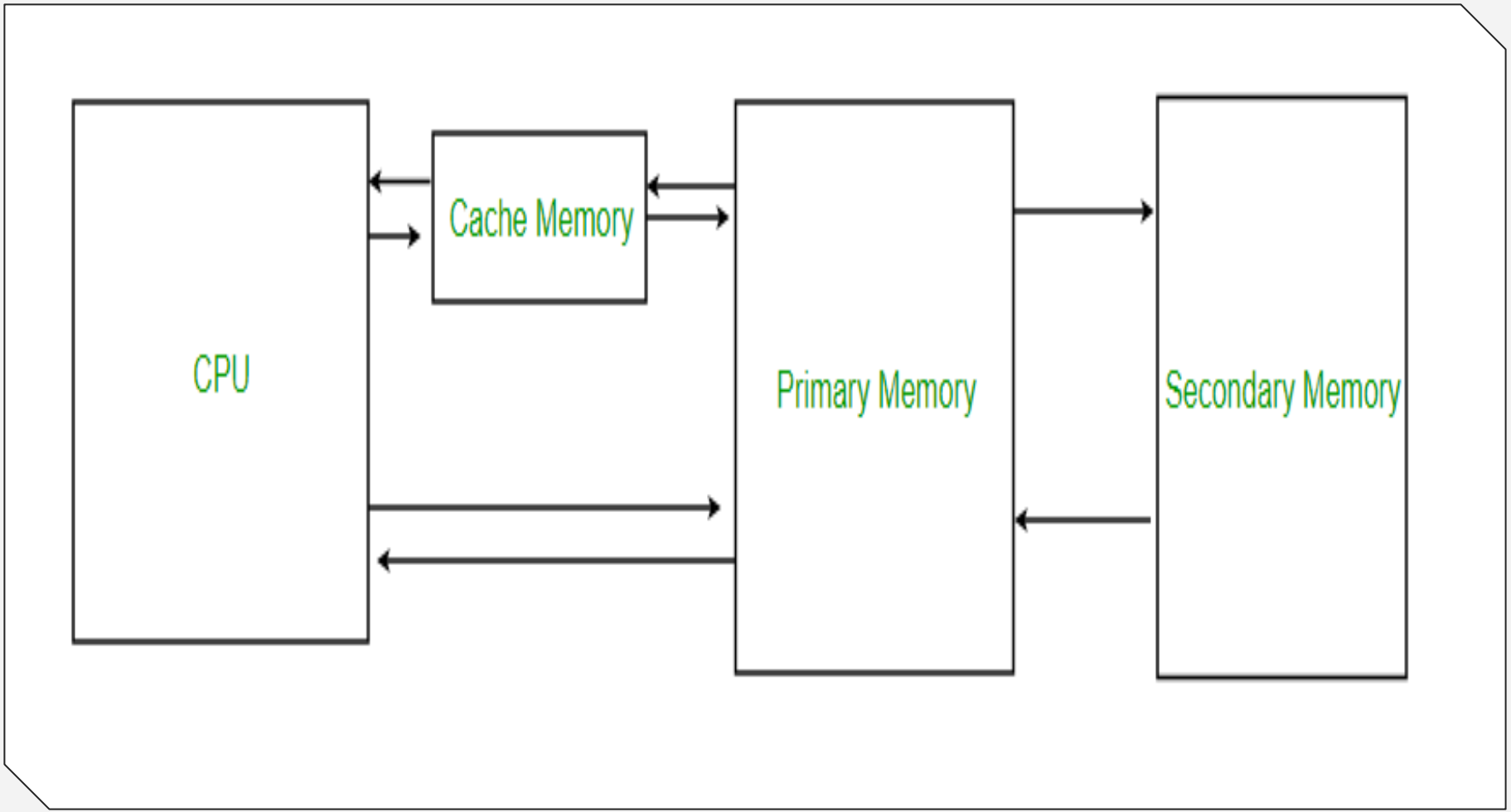
Cache memory is faster than main memory but smaller in size.

It typically comes in three levels: L1, L2, and L3 cache, with L1 being the smallest and fastest and L3 being the largest but slowest among them.

The CPU first checks the L1 cache for data, and if it's not found there, it checks the L2 cache, and so on, until the required data is located.

Cache memory helps improve overall system performance by reducing the time the CPU spends waiting for data from the slower main memory.

It exploits the principle of locality, which states that programs tend to access the same data or instructions repeatedly, making it advantageous to keep a copy of that data closer to the processor for quicker access.


CPU
Cache Memory
Primary Memory
Secondary Memory

Working of Cache Memory

- When the CPU needs to access memory, the cache is examined. If the word is found in the cache, it is read from the cache memory.
- If the word addressed by the CPU is not found in the cache, the main memory is accessed to read the word.
- A block of words one just accessed is then transferred from main memory to cache memory. The block size may vary from one word (the one just accessed) to about 16 words adjacent to the one just accessed.
- Next time, if the CPU needs to access the same data again, it will just retrieve the data from the cache instead of going through the whole loading process again.

Performance of Cache Memory

- The performance of the cache memory is frequently measured in terms of a quantity called **hit ratio**.
- When the CPU refers to memory and finds the word in cache, it is said to produce a **cache hit**.
- If the word is not found in the cache, it is in main memory and it counts as a **cache miss**.
- The ratio of the number of hits divided by the total CPU references to memory (*hit plus miss*) is the ***hit ratio***.

- **Hit Ratio** = no. of Hits / (Total no of CPU reference)

i.e. **Hit ratio (h)** = Hits / (Hits + Miss)

- **Miss Ratio** = no. Of Miss / (Total no of CPU reference)

i.e. **Miss Ratio(1-h)** = Miss / (Hits + Miss)

- **Cache Access Time(Cache Hit time)T_c** :- Time required to access word from the cache

- **Miss Penalty(Cache Miss time penalty)T_m** :- The time required to fetch required block from the Main Memory.

- **Average Access Time of CPU =**
  Hit Ratio x Cache Access Time + (1- hit ratio) x Miss Penalty

i.e. $= h \times T_c + (1-h) \times T_m$

Miss Penalty = Cache Memory Access Time + Main Memory Access Time

Level of Cache Memory

Cache memory in modern computer systems typically consists of multiple levels, usually referred to as L1, L2, and sometimes L3 cache. Here's a brief overview of each level:

L1 Cache (Level 1 Cache):

- *Location:* Located closest to the CPU.
- *Size:* Smallest in size, but also the fastest.
- *Purpose:* Designed to store frequently accessed data and instructions for quick retrieval by the CPU.
- *Speed:* Very fast access times, often accessed in just a few CPU clock cycles.
- *Split:* Often divided into separate caches for instructions (L1i) and data (L1d).

L2 Cache (Level 2 Cache):

- *Location*: Situated between the L1 cache and the main memory.
- *Size*: Larger than L1 cache but smaller than L3 cache.
- *Purpose*: Acts as a secondary cache, storing additional data and instructions that may not fit in the L1 cache.
- *Speed*: Slower access times compared to L1 cache but still faster than accessing data from the main memory.

L3 Cache (Level 3 Cache):

- *Location*: Positioned between the L2 cache and the main memory, or sometimes shared across multiple processor cores.
- *Size*: Largest among the cache levels.
- *Purpose*: Serves as a shared cache for multiple processor cores or as a larger cache for storing data that may not fit in the L1 and L2 caches.
- *Speed*: Slower access times compared to L1 and L2 caches but faster than accessing data from the main memory.

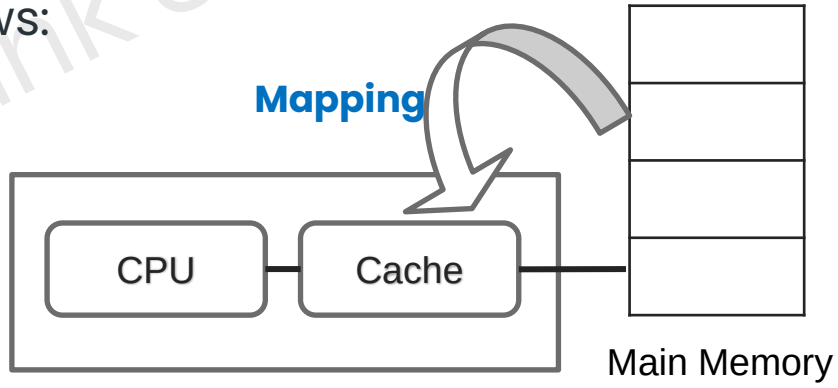
Cache Mapping

Cache mapping is a technique by which content of main memory is brought into the Cache memory.

It is the transformation of data from main memory to cache memory.

There are three different types of mapping used for the purpose of cache memory which is as follows:

- Direct Mapping
- Associative Mapping
- Set-Associative Mapping



Direct Mapping

Direct cache mapping is a straight forward and efficient method used to determine the location of data in a cache memory. It works by mapping each block of main memory to a single, specific cache line. Here's a detailed explanation of how it functions:

How Direct Cache Mapping Works

- *Structure of Cache Memory* : The cache is divided into a number of cache lines (or slots), each of which can hold one block of data from main memory. Each cache line has a unique index.

- *Mapping Process* : The main memory is also divided into blocks, which are the same size as the cache lines. Each block of main memory is mapped to a specific cache line using a simple formula.

The Mapping Formula :

To determine which cache line a block of memory maps to, the following formula is used:

Cache Line Index = (Block Address) mod (Number of Cache Lines)

i.e **K mod n,** where K is Block Address and n is Number of Cache Lines

- *Block Address(K)*: The address of the block in main memory.
- *Number of Cache Lines(n)*: The total number of lines in the cache.

Components of a Memory Address

A memory address is typically divided into three parts:

- *Tag*: Identifies which block of memory is currently stored in the cache line.
- *Index*: Determines the specific cache line where the block should be stored.
- *Offset*: Specifies the exact location within the block.

Example

Assume we have a cache with 4 lines (indexed L0 to L3) and a main memory with 32 blocks (indexed B0 to B31). Here's how blocks would map to cache lines:

As seen, multiple blocks (like block 0 and block 4) can map to the same cache line.

Note:
Block size = cache line size (it is 4 word in our example)

Main Memory

B0	W0,W1,W2,W3
B1	W4,W5,W6,W7
B2	W8,W9,W10,W11
.....	.....
.....	
B31	W124,W124,W126,W127

128 word(Memory size)

Cache Memory

L0	B0, B4 ..... , B28
L1	B1, B5 ..... , B29
L2	B2, B6 ..... , B30
L3	B3, B7 ..... , B31

16 word (Memory size)

- B0 maps to cache line (0 mod 4 = 0) i.e. L0
- B1 maps to cache line (1 mod 4 = 1) i.e. L1
- B2 maps to cache line (2 mod 4 = 2) i.e. L2
- B3 maps to cache line (3 mod 4 = 3) i.e. L3
- B4 maps to cache line (4 mod 4 = 0) i.e. L0
- B5 maps to cache line (5 mod 4 = 1) i.e. L1
- B6 maps to cache line (6 mod 4 = 2) i.e. L2
- B7 maps to cache line (7 mod 4 = 3) i.e. L3

.....

.....

.....

- B28 maps to cache line (28 mod 4 = 0) i.e. L0
- B29 maps to cache line (29 mod 4 = 1) i.e. L1
- B30 maps to cache line (30 mod 4 = 1) i.e. L2
- B31 maps to cache line (31 mod 4 = 1) i.e. L3

How CPU reference

- In Main Memory we generate physical address.
- Main Memory Size = 128 words, i.e 2^7 , so 7 bits is required to represent 128 words (w0 to w127) that's why Physical address is of 7bits.
- Block size = 4 words(4 words in a Block) i.e 2^2 , 2 bit is required to represent Block offset (Block size)i.e exact location, remaining 5 bit is to represent block no. (i.e 2^5 = 32 block{B0 to B31}).

Main Memory

128 word(Memory size)

B0	W0,W1,W2,W3
B1	W4,W5,W6,W7
B2	W8,W9,W10,W11
.....	.....
.....	
B31	W124,W124,W126,W127

- Let CPU generate address $\mathbf{(0001010)_2}$ = $(10)_{10}$ {i.e word10 –> w10}
- So, First 5 bit gives *Block No.* i.e 00010 and last 2 bit gives block offset(index no. of word) i.e $(10)_2 = (2)_{10}$

Physical Address(7 bits)

5	2
Block No.	Block offset (Block size)

- *$(00010)_2 = (2)_{10}$ i.e Block no. 2 -→ B2*
- *$(10)_2 = (2)_{10}$ i.e index no. 2 i.e word 10 -→ w10*

How CPU reference

- Now when any block come into cache then how address is generated in cache and how the word present in block is accessed?
- Physical address is divided into 3 parts –

1. Block offset (2 bit)
2. Line no. (2 bit){as there are 4 line in cache indexed from L0 to L3 i.e 2^2 so, 2 bit is required to represent line no.}
3. Tag(3 bit)

Main Memory

128 word(Memory size)

B0	W0,W1,W2,W3
B1	W4,W5,W6,W7
B2	W8,W9,W10,W11
.....	.....
B31	W124,W124,W126,W127

Cache Memory

L0	B0, B4 ..... , B28
L1	B1, B5 ..... , B29
L2	B2, B6 ....., B30
L3	B3, B7 ....., B31

Physical Address(7 bits)

3	2	2
Tag	Line No. (index no)	Block offset (Block size) i.e exact location

16 word (Memory size)

Example:

Cache Memory

Let,

Tag	Line no. (L0)	Block offset		
001	00	0	0	= W16
001	00	0	1	= W17
001	00	1	0	= W18
001	00	1	1	= W19

So, currently in cache there is B4 Block
&
words are
W16, W17, W18, W19

Tag(001)

L0
L1
L2
L3

Main Memory
128 word(Memory size)

B0	W0,W1,W2,W3
B1	W4,W5,W6,W7
B2	W8,W9,W10,W11
.....	.....
.....	
B31	W124,W124,W126,W127

Cache Memory

L0	B0, B4 ..... , B28
L1	B1, B5 ..... , B29
L2	B2, B6 ..... , B30
L3	B3, B7 ..... , B31

Physical Address(7 bits)

3	2	2
Tag	Line No. (index no)	Block offset (Block size) i.e exact location

16 word
(Memory size)

Advantages of Direct Cache Mapping

- *Simplicity:* The mapping process is straightforward and easy to implement.
- *Speed:* Cache line lookup is very fast due to the direct index calculation.

Disadvantages of Direct Cache Mapping

- *Conflict Misses*: Since each block can only map to one specific cache line, if two blocks that map to the same line are accessed alternately, it causes a lot of cache misses (called conflict misses).

- *Limited Flexibility:* Less flexible compared to other mapping techniques like associative or set-associative mapping, which can reduce conflict misses.

Practical Implications

In practical scenarios, while direct mapping is efficient in terms of speed and simplicity, it might not be the best choice for applications with high conflict misses.
To balance performance, systems might use set-associative or fully associative mappings which provide better utilization of cache memory by reducing conflict misses.

Associative cache mapping

Associative cache mapping, also known as fully associative mapping, is a cache memory management technique used in computer architecture. It determines how data from main memory is stored in the cache. Here's an overview of how associative cache mapping works:

Key Characteristics:

- No Fixed Location : Unlike direct-mapping, where each block of main memory maps to a specific cache line, in associative mapping, any block of main memory can be stored in any cache line.

- *Tag and Data* : Each cache entry is divided into two parts: the tag and the data. The tag holds the address information needed to identify which block of memory the data belongs to, while the data is the actual information being stored.

- *Tag Matching* : When the CPU needs to access data, the cache is searched by comparing the tags of all cache lines to the address tag of the requested data. If a match is found (cache hit), the data is read from the cache. If no match is found (cache miss), the data is fetched from main memory and placed into the cache.

- *Cache Replacement Policies* : Since any block can be placed in any cache line, when the cache is full, a replacement policy decides which block to evict. Common policies include Least Recently Used (LRU), First In First Out (FIFO), and Random Replacement.

Advantages:

- *Flexibility* : Any memory block can be loaded into any cache line, leading to better utilization of cache space and reduced conflict misses (when multiple memory blocks map to the same cache line).

- *Higher Hit Rate* : Due to the flexible placement of blocks, fully associative caches generally have higher hit rates compared to direct-mapped caches, especially when the cache size is small relative to the working set of the program.

Disadvantages:

- *Complexity* : The need to search through all cache lines to find a match makes associative mapping more complex and expensive in terms of hardware, especially for larger caches.

- *Speed* : The search operation for tag comparison can be slower than direct mapping, as it requires checking multiple entries simultaneously.

Example:

Consider a cache with 4 lines and a memory with 8 blocks. In a fully associative cache, any of the 8 memory blocks can be placed in any of the 4 cache lines. When a memory block needs to be accessed, the cache controller checks the tags of all 4 lines to see if the block is present.

Main Memory
128 word(Memory size)

B0	W0,W1,W2,W3
B1	W4,W5,W6,W7
B2	W8,W9,W10,W11
……	……
……	
B31	W124,W124,W126,W127

Cache Memory
16 word(Memory size)

L0	B0, B1 …… , B28
L1	B0, B1 …… , B29
L2	B0, B1 …… , B30
L3	B0, B1 …… , B31

Physical Address(7 bits)

5	2
Block No.	Block offset (Block size)

Main Memory
128 word(Memory size)

B0	W0,W1,W2,W3
B1	W4,W5,W6,W7
B2	W8,W9,W10,W11
.....	.....
B31	W124,W124,W126,W127

Cache Memory
16 word(Memory size)

L0	B0, B1 ..... , B28
L1	B0, B1 ..... , B29
L2	B0, B1 ..... , B30
L3	B0, B1 ..... , B31

Physical Address(7 bits)

Let,
Tag = 00100

Tag	Block offset		
00100	0	0	= W16
00100	0	1	= W17
00100	1	0	= W18
00100	1	1	= W19

i.e When Tag bit is $(00100)_2$ that means we are talking about B4 Block
&
B4 Block contain W16, W17, W18, W19

K-Way Set Associative Cache

K-way set associative mapping is a type of cache memory organization that balances the simplicity of direct-mapped cache with the flexibility of fully associative cache. Here's how it works:

Dividing the Cache into Sets : The cache is divided into multiple sets. The number of sets is determined by dividing the total number of cache lines by the associativity (K). For example, in a 2-way set associative cache with 4 cache lines, there would be 2 sets (4 / 2 = 2).

Mapping Memory Blocks to Sets : Each memory block maps to a specific set, but within that set, it can be placed in any of the K lines. The set to which a memory block maps is determined by using the concept of Direct Mapping.

The Mapping Formula : To determine which set a block of memory maps to, the following formula is used:

Set Index = (Block Address) mod (Number of Set)

i.e **K mod n,** where K is Block Address and n is Number of Set

- *Block Address(K)*: The address of the block in main memory.
- *Number of Set(n)*: The total number of Set in which the cache is divided.

Note : This formula and concept is similar to Direct mapping the only difference is, in direct mapping *n* is *number of cache* lines and here *n* is *Number of sets*

- B0 maps to set (0 mod 2 = 0) i.e. S0
- B1 maps to set (1 mod 2 = 1) i.e. S1
- B2 maps to set (2 mod 2 = 0) i.e. S0
- B3 maps to set (3 mod 2 = 1) i.e. S1

.....

.....

.....

- B28 maps to set (28 mod 2 = 0) i.e. S0
- B29 maps to set (29 mod 2 = 1) i.e. S1
- B30 maps to set (30 mod 2 = 0) i.e. S0
- B31 maps to set (31 mod 2 = 1) i.e. S1

Main Memory
128 word(Memory size)

B0	W0,W1,W2,W3
B1	W4,W5,W6,W7
B2	W8,W9,W10,W11
.....	.....
B31	W124,W124,W126,W127

Cache Memory
16 word(Memory size)

L0	B0, B2, B4…,B30	S0
L1	B0, B2, B4…,B30	
L2	B1, B3, B5…,B31	S1
L3	B1, B3, B5…,B31	

Physical Address(7 bits)

5	2
Block No.	Block offset (Block size)

Physical Address(7 bits)

4	1	2
Tag	Set No.	Block offset (Block size) i.e exact location

For example, in a 2-way set associative cache with 4 cache lines, there would be 2 sets (4 / 2 = 2).
When we map any block in cache then,
Suppose, CPU generate address $(0010000)_2$ then first 4 is tag i.e 0010, next 1 bit is set no. and remaining 2 bit is offset.

Main Memory
128 word(Memory size)

B0	W0,W1,W2,W3
B1	W4,W5,W6,W7
B2	W8,W9,W10,W11
.....	.....
B31	W124,W124,W126,W127

Cache Memory
16 word(Memory size)

L0	B0, B2, B4…,B30	S0
L1	B0, B2, B4…,B30	
L2	B1, B3, B5…,B31	S1
L3	B1, B3, B5…,B31	

Tag	Set no. (S0)	Block offset		
0001	0	0	0	= W8
0001	0	0	1	= W9
0001	0	1	0	= W10
0001	0	1	1	= W11

So, currently CPU is referencing B2 Block in cache
&
words are
W16, W17, W18, W19

Tag Comparison : The remaining higher-order bits of the memory address form the tag. When accessing memory, the index is used to find the correct set, and then the tags of the lines in that set are compared to the tag of the requested block to check for a hit.

Replacement Policy : If all lines in a set are occupied and a new block needs to be loaded into the set, a replacement policy (such as Least Recently Used (LRU), First In First Out (FIFO), or Random) determines which block to evict.

Example

Consider a 2-way set associative cache with 8 total cache lines. This cache will have 4 sets (8 / 2 = 2). Suppose a memory block needs to be accessed:

Index Calculation: The block address is divided into three parts: tag, set index, and block offset.

If the total address is 16 bits, the block offset is determined by the block size (let's assume 4 bytes, needing 2 bits).

The remaining 14 bits are split between the set index and tag. With 4 sets, we need 2 bits for the index($2^{2=4}$)

Mapping:

Suppose the memory address is 0x1234.

The address in binary is 0001 0010 0011 0100.

Block offset: last 2 bits (`00` for this example).

Set Index: next 2 bits (`01` or `1` in decimal).

Tag: remaining higher-order bits (`0001 0010 0011` or `123` in decimal).

Accessing the Cache:

The set index 01 directs us to the second set.

The tag 123 is compared with the tags of the blocks in the lines of the second set.

If a match is found, it's a cache hit, and the data is accessed.

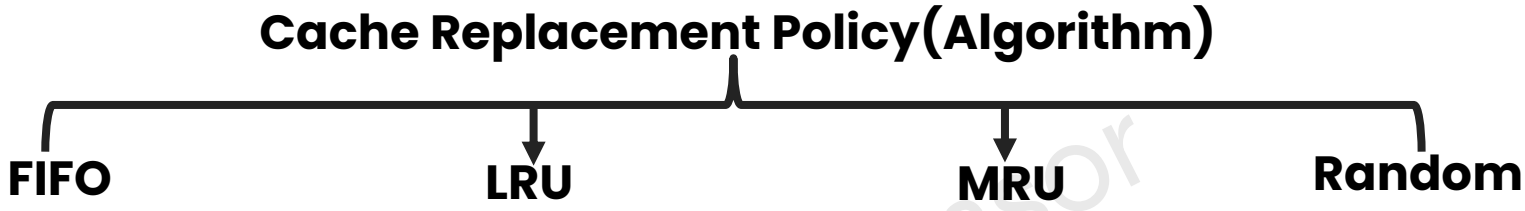
If no match, it's a cache miss, and the block must be loaded into one of the lines of the second set, possibly removing an existing block based on the replacement policy.

Advantages of K-Way Set Associative Cache

Reduced Conflict Misses: More flexibility than direct-mapped cache reduces conflicts.

Simpler Hardware: Less complex than fully associative cache, as only a small number of lines need to be searched.

Better Performance: Balances the hit rate and access time effectively for many applications.



A cache replacement policy is a strategy used by a cache memory system to determine which item to remove from the cache when the cache is full and a new item needs to be brought in.

The goal of a cache replacement policy is to maximize the cache hit rate, which is the proportion of memory accesses that are serviced by the cache rather than the slower main memory.

These policies balance factors like access patterns and memory constraints to optimize cache efficiency, thus improving overall system performance by minimizing accesses to slower main memory.

Note:

In every cache replacement algorithm, Initially, when cache is empty so we start putting the block in cache from Main memory (cache Miss) but when the block found in the cache then it is cache Hit.

When the cache line is full then only we use the concept of specific cache replacement algorithm to replace an old block and put a new block in a cache line.

FIFO (First-In, First-Out)

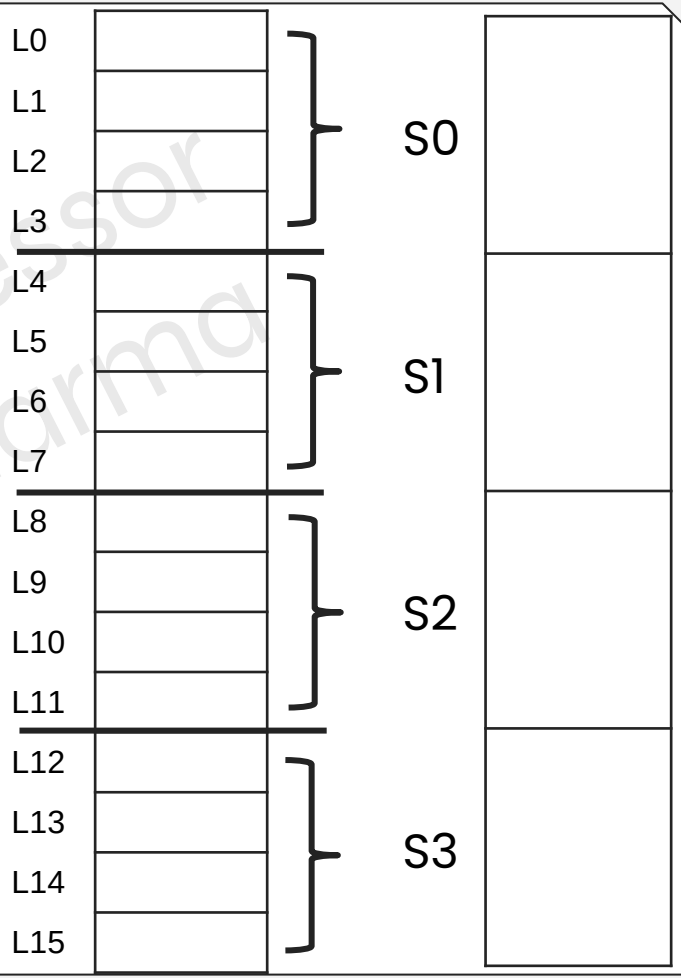
- The FIFO (First-In, First-Out) cache replacement policy operates on the principle of removing the oldest item from the cache when it becomes full and needs to accommodate a new item.
- It maintains a queue of items in the cache, and when removal is necessary, it removes the item that was first inserted into the cache, hence following a "first in, first out" approach.
- FIFO is straightforward to implement but may not always result in optimal cache performance, particularly if the oldest items are frequently accessed.

Example:-

Que. Consider a 4- way set associative cache with total 16 cache line. Main Memory block request are:

0, 255, 1, 4, 3, 8, 133, 159, 216, 129, 63, 8, 48, 32, 73, 92, 155, 2 *(Use FIFO)*

Sol. Given, 4 way set associativity
Total no of set = Total cache Line/associativity
= 16/4
= 4
Mapping formula = ***K mod n***, where *K*= Block No. and *n* = Total no of set



- 0 maps to set (0 mod 4 = 0) i.e. S0
- 255 maps to set (255 mod 4 = 3) i.e. S3
- 1 maps to set (1 mod 4 = 1) i.e. S1
- 4 maps to set (4 mod 4 = 0) i.e. S0
- 3 maps to set (3 mod 4 = 3) i.e. S3
- 8 maps to set (8 mod 4 = 0) i.e. S0
- 133 maps to set (133 mod 4 = 1) i.e. S1
- 159 maps to set (159 mod 4 = 3) i.e. S3
- 216 maps to set (216 mod 4 = 0) i.e. S0
- 129 maps to set (129 mod 4 = 1) i.e. S1
- 63 maps to set (63 mod 4 = 3) i.e. S3
- 8 maps to set (8 mod 4 = 0) i.e. S0 (Cache Hit)
- 48 maps to set (48 mod 4 = 0) i.e. S0
- 32 maps to set (32 mod 4 = 0) i.e. S0
- 73 maps to set (73 mod 4 = 1) i.e. S1
- 92 maps to set (92 mod 4 = 0) i.e. S0
- 155 maps to set (155 mod 4 = 3) i.e. S3
- 2 maps to set (2 mod 4 = 2) i.e. S2

S0	~~0~~, ~~4~~, ~~8~~, 216, 48, 32, 92
S1	1, 133, 129, 73
S2	2
S3	~~255~~, 3, 159, 63, 155

LRU (Least Recently Used)

The LRU (Least Recently Used) cache replacement algorithm remove the least recently accessed items from a cache when it's full, allowing space for new entries.

It maintains a record of when each item was last accessed. When the cache reaches its capacity, the algorithm removes the item that was least recently accessed, assuming it's less likely to be needed soon.

This strategy optimizes cache performance by prioritizing the retention of recently accessed items, which are more likely to be accessed again.

Implementation techniques vary, from using linked lists with timestamps to more efficient data structures like hash maps or priority queues.

Example:-

Que. Consider a fully associative cache with total 8 cache line. Main Memory block request are:

4, 3, 25, 8, 19, 6, 25, 8, 16, 35, 45, 22, 8, 3, 16, 25, 7 *(Use LRU)*

H H H H H

Sol. Given, fully associativity mapping
In fully associative mapping, any block can come at any place.

Cache

L0	~~4~~, 45
L1	~~3~~, 22
L2	25
L3	8
L4	~~19~~, 3
L5	~~6~~, 7
L6	16
L7	35

MRU (Most Recently Used)

The Most Recently Used (MRU) cache replacement policy prioritizes data based on its **last access time**. When the cache reaches its capacity and needs to make room for new data, it removes the data item that was accessed **most recently**.
This means that data that has been used recently is more likely to remain in the cache, potentially improving performance for short-term bursts of access.
However, MRU may not be ideal for situations where the same data is accessed repeatedly over time. In such cases, the recently accessed data might be constantly evicted and re-added to the cache, leading to unnecessary overhead and potentially worse performance. For these scenarios, other cache replacement policies like Least Recently Used (LRU) might be more suitable.

Example:-

Que. Consider a fully associative cache with total 8 cache line. Main Memory block request are:

4, 3, 25, 8, 19, 6, 25, 8, 16, 35, 45, 22, 8, 3, 16, 25, 7 *(Use MRU)*

H H H H H H H

Sol. Given, fully associativity mapping
In fully associative mapping, any block can come at any place.

Cache

L0	4
L1	3
L2	~~25~~, 7
L3	8
L4	19
L5	6
L6	16
L7	~~35~~, ~~45~~, 22

Thanks!

Do you have any questions?

Shashank.16596@gmail.com

+91-9827881405