SUBJECT NAME: Computer Organization & Architecture

SUBJECT CODE: 203105253

UNIT 6

Prepared By
Trilok Suthar

MEMORY ORGANIZATION:

Memory organization: Memory interleaving, concept of hierarchical memory organization, cache memory, cache size vs. block size, mapping functions, replacement algorithms, write policies
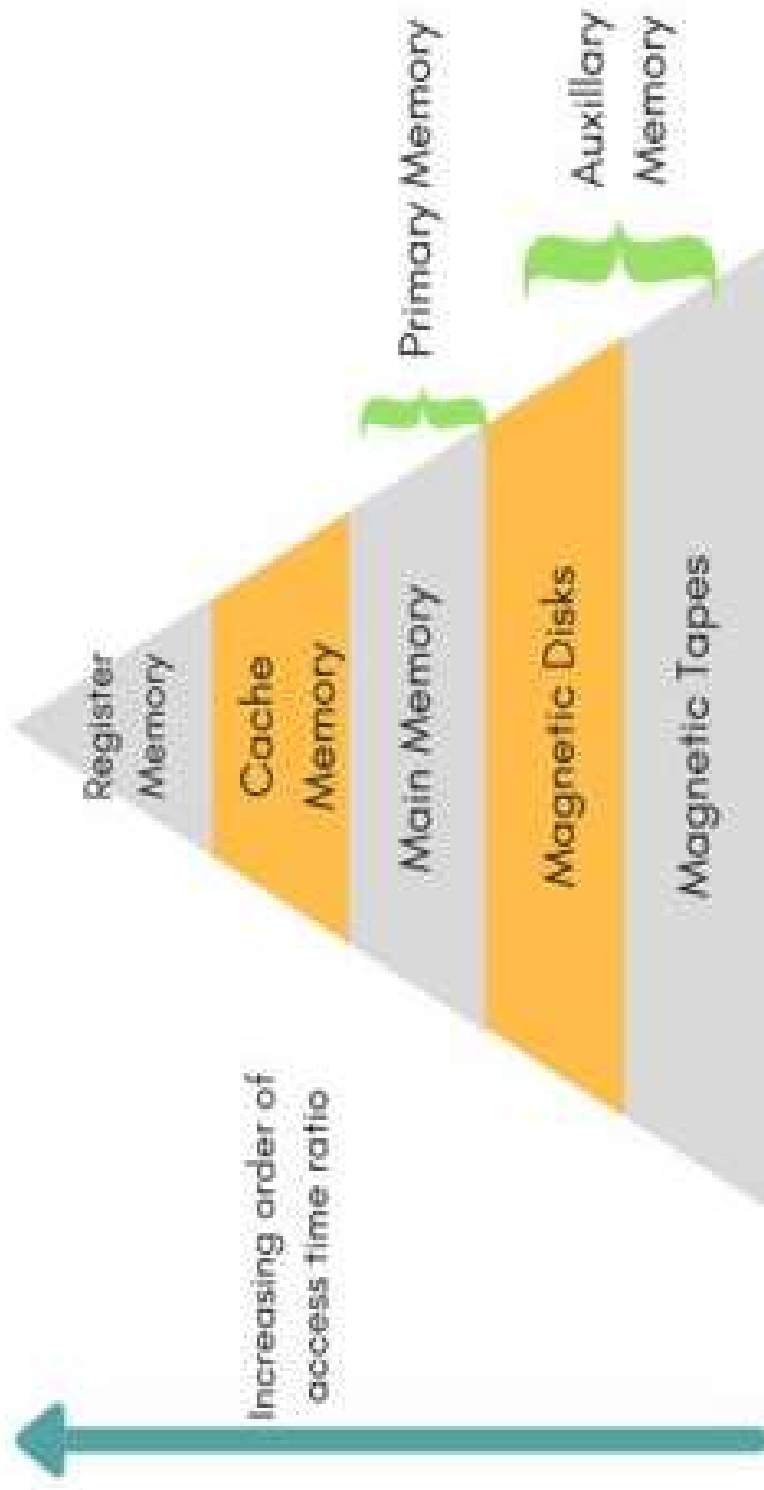
Memory interleaving

- Pipeline and vector processors often require simultaneous access to memory from two or more sources. An instruction pipeline may require the fetching of an instruction and an operand at the same time from two different segments.

- Similarly, an arithmetic pipeline usually requires two or more operands to enter the pipeline at the same time. Instead of using two memory buses for simultaneous access, the memory can be partitioned into a number of modules connected to a common memory address and data buses.

- A memory module is a memory array together with its own address and data registers.

Address bus

Data bus
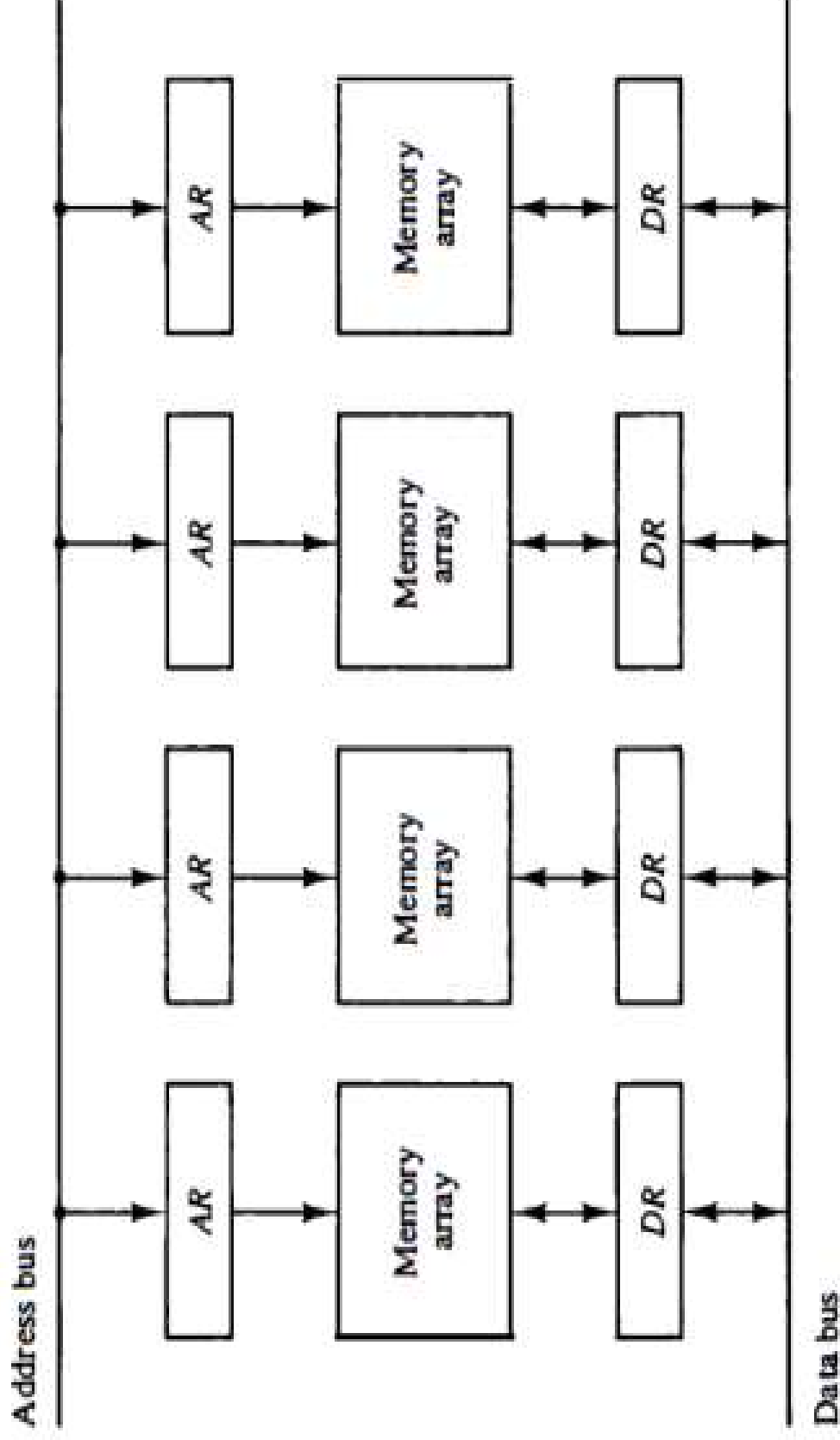
- Figure shows a memory unit with four modules. Each memory array has its own address register AR and data register DR.
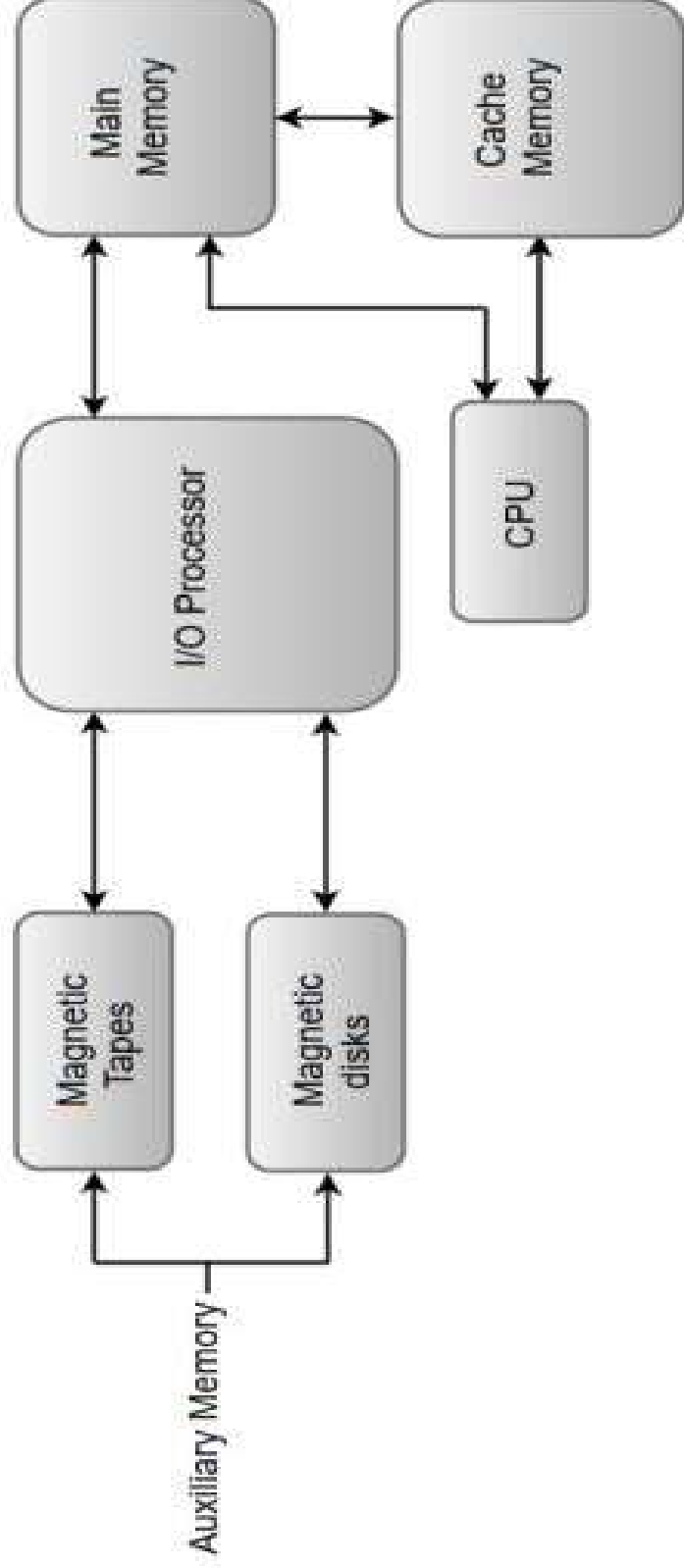
concept of hierarchical memory organization

- **This Memory Hierarchy Design is divided into 2 main types:**

- **External Memory or Secondary Memory**
  Comprising of Magnetic Disk, Optical Disk, Magnetic Tape i.e.
  peripheral storage devices which are accessible by the processor
  via I/O Module.

- **Internal Memory or Primary Memory**
  Comprising of Main Memory, Cache Memory & CPU registers. This
  is directly accessible by the processor

Register Memory

Cache Memory

Main Memory

Magnetic Disks

Magnetic Tapes

Primary Memory

Auxillary Memory

Increasing order of access time ratio

Memory Hierarchy in Computer system

Memory Hierarchy in a Computer System:

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:**

Earlier when the computer system was designed without Memory Hierarchy design, the speed gap increases between the CPU registers and Main Memory due to large difference in access time. This results in lower performance of the system and thus, enhancement was required. This enhancement was made in the form of Memory Hierarchy Design because of which the performance of the system increases. One of the most significant ways to increase system performance is minimizing how far down the memory hierarchy one has to go to manipulate data.

- **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.

Cache Memories:

- The cache is a small and very fast memory, interposed between the processor and the main memory. Its purpose is to make the main memory appear to the processor to be much faster than it actually is. The effectiveness of this approach is based on a property of computer programs called locality of reference.

- Analysis of programs shows that most of their execution time is spent in routines in which many instructions are executed repeatedly. These instructions may constitute a simple loop, nested loops, or a few procedures that repeatedly call each other.

CPU

Cache Memory

Primary Memory

Secondary Memory
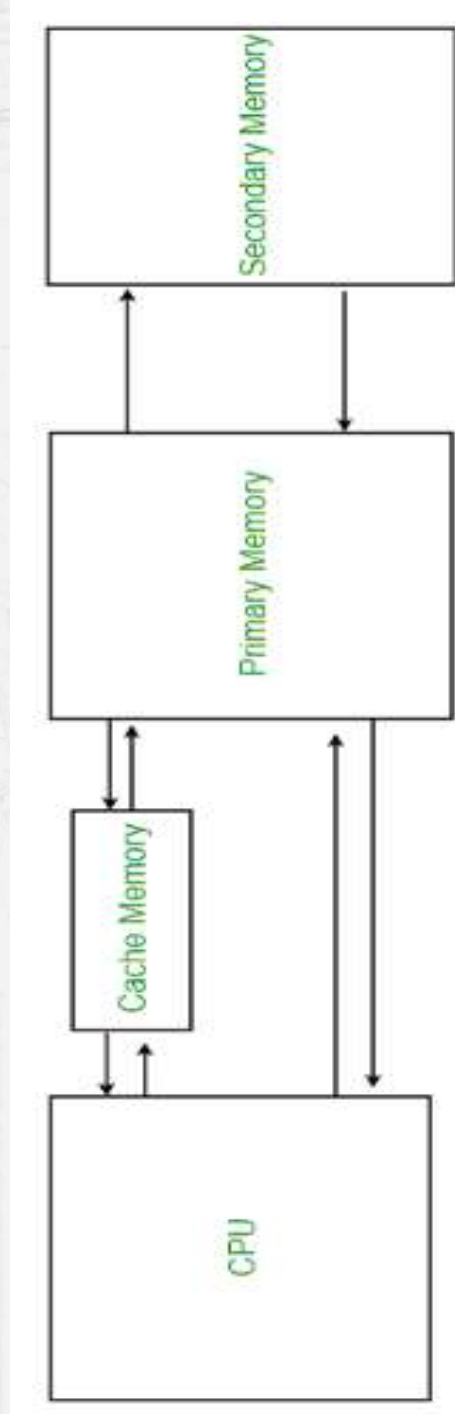
- The cache memory can store a reasonable number of blocks at any given time, but this number is small compared to the total number of blocks in the main memory. The correspondence between the main memory blocks and those in the cache is specified by a mapping function.

- When the cache is full and a memory word (instruction or data) that is not in the cache is referenced, the cache control hardware must decide which block should be removed to create space for the new block that contains the referenced word. The collection of rules for making this decision constitutes the cache's *replacement algorithm*.

Cache Hits

- The processor does not need to know explicitly about the existence of the cache.

- It simply issues Read and Write requests using addresses that refer to locations in the memory.

- The cache control circuitry determines whether the requested word currently exists in the cache.

- If it does, the Read or Write operation is performed on the appropriate cache location. In this case, a *read* or *write hit* is said to have occurred

- **Cache Misses**
  A Read operation for a word that is not in the cache constitutes a *Read miss*. It causes the block of words containing the requested word to be copied from the main memory into the cache.

- **Cache Mapping:**
  There are three different types of mapping used for the purpose of cache memory which are as follows: **Direct mapping, Associative mapping, and Set-Associative mapping.**

- **Direct mapping**
  The simplest way to determine cache locations in which to store memory blocks is the *direct mapping* technique.

- In this technique, block *j* of the main memory maps onto block

j modulo 128 of the cache, as depicted in Figure Thus, whenever one of the main memory blocks 0, 128, 256, . . . is loaded into the cache, it is stored in cache block 0. Blocks 1, 129, 257, . . . are stored in cache block 1, and so on. Since more than one memory block is mapped onto a given cache block position, contention may arise for that position even when the cache is not full.

Main memory

Block 0
Block 1
Block 127
Block 128
Block 129
Block 255
Block 256
Block 257
Block 4095

Cache

Block 0
Block 1
Block 127

tag
tag
tag

Main memory address

Tag	Block	Word
5	7	4
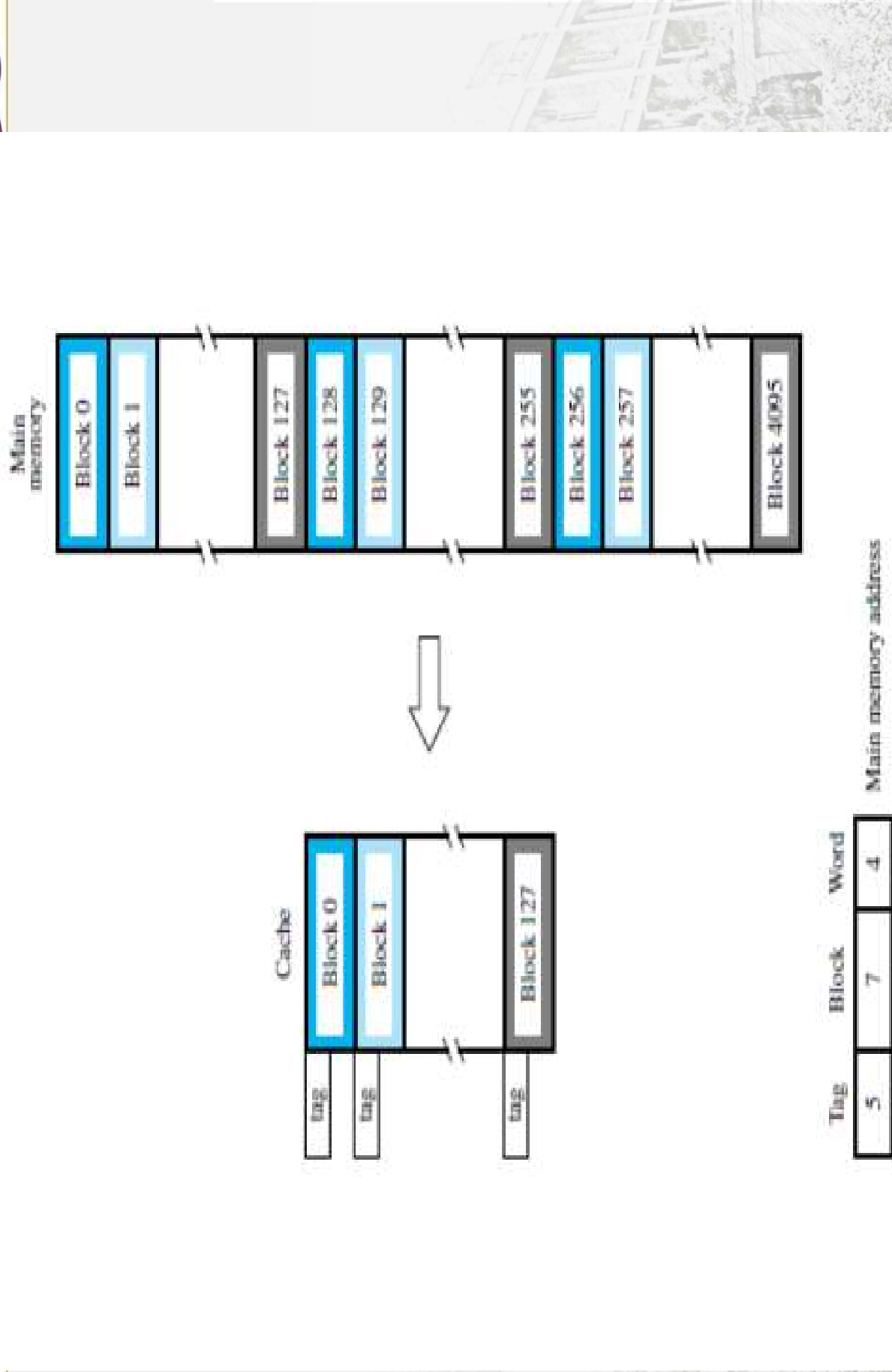
- For example, instructions of a program may start in block 1 and continue in block 129, possibly after a branch. As this program is executed, both of these blocks must be transferred to the block-1 position in the cache. Contention is resolved by allowing the new block to overwrite the currently resident block.

- With direct mapping, the replacement algorithm is trivial. Placement of a block in the cache is determined by its memory address. The memory address can be divided into three fields, as shown in Figure. The low-order 4 bits select one of 16 words in a block.

- When a new block enters the cache, the 7-bit cache block field determines the cache position in which this block must be stored. If they match, then the desired word is in that block of the cache. If there is no match, then the block containing the required word must first be read from the main memory and loaded into the cache.

- The direct-mapping technique is easy to implement, but it is not very flexible.

Associative Mapping

- In Associative mapping method, in which a main memory block can be placed into any cache block position. In this case, 12 tag bits are required to identify a memory block when it is resident in the cache.

- The tag bits of an address received from the processor are compared to the tag bits of each block of the cache to see if the desired block is present.

- This is called the *associative-mapping* technique.

Main memory

Block 0
Block 1
Block i
Block 4095

Cache

Block 0
Block 1
Block 127

tag
tag
tag

Tag   Word

12   4

Main memory address
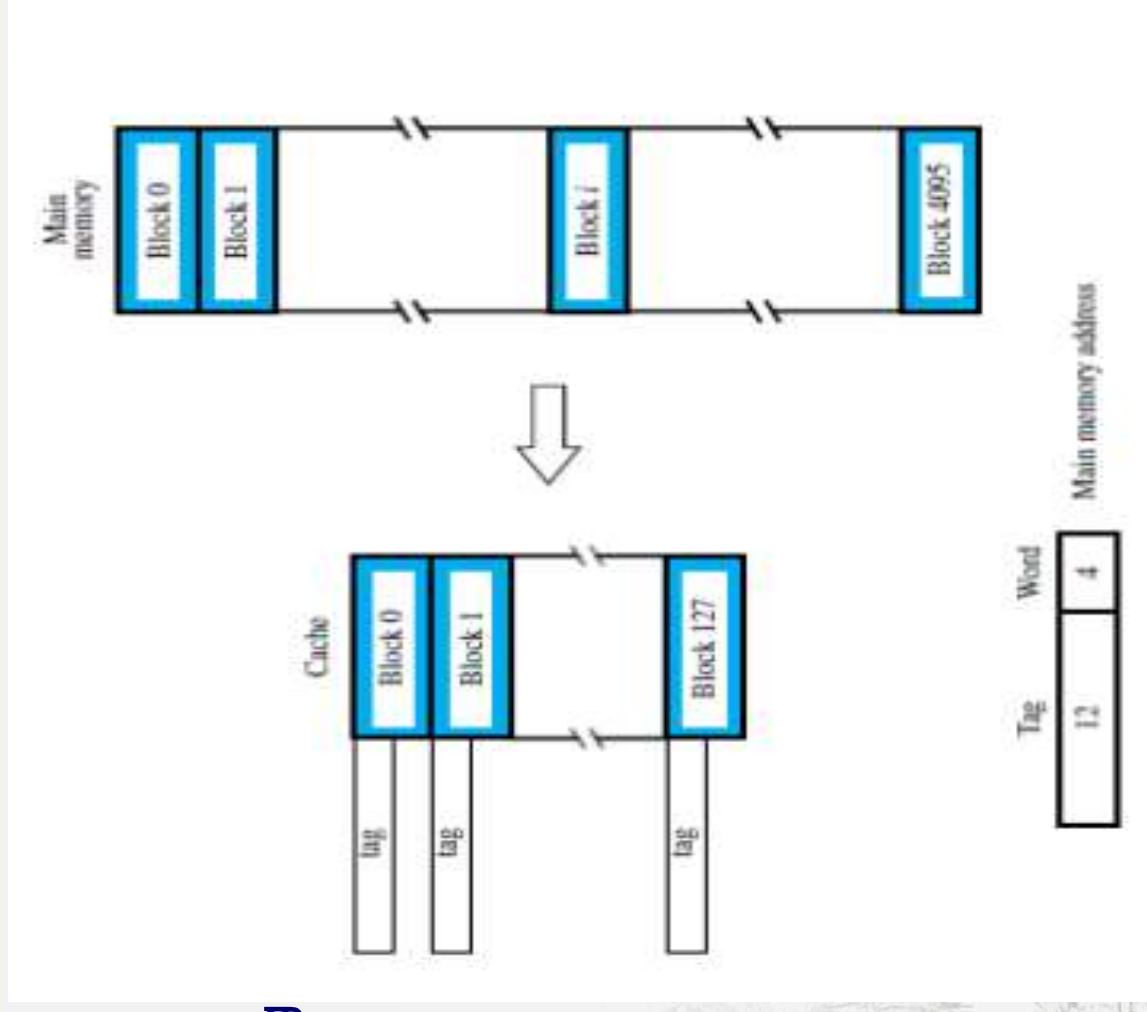
- It gives complete freedom in choosing the cache location in which to place the memory block, resulting in a more efficient use of the space in the cache.

- When a new block is brought into the cache, it replaces (ejects) an existing block only if the cache is full. In this case, we need an algorithm to select the block to be replaced

- To avoid a long delay, the tags must be searched in parallel. A search of this kind is called an *associative search*

Set-Associative Mapping

- Another approach is to use a combination of the direct- and associative-mapping techniques.

- The blocks of the cache are grouped into sets, and the mapping allows a block of the main memory to reside in any block of a specific set.

- Hence, the contention problem of the direct method is eased by having a few choices for block placement

- At the same time, the hardware cost is reduced by decreasing the size of the associative search

Cache

Block 0
Block 1
Block 2
Block 3
Block 126
Block 127

tag
tag
tag
tag
tag
tag

Set 0
Set 1
Set 63

Block 0
Block 1
Block 63
Block 64
Block 65
Block 127
Block 128
Block 129
Block 4095

Main memory address
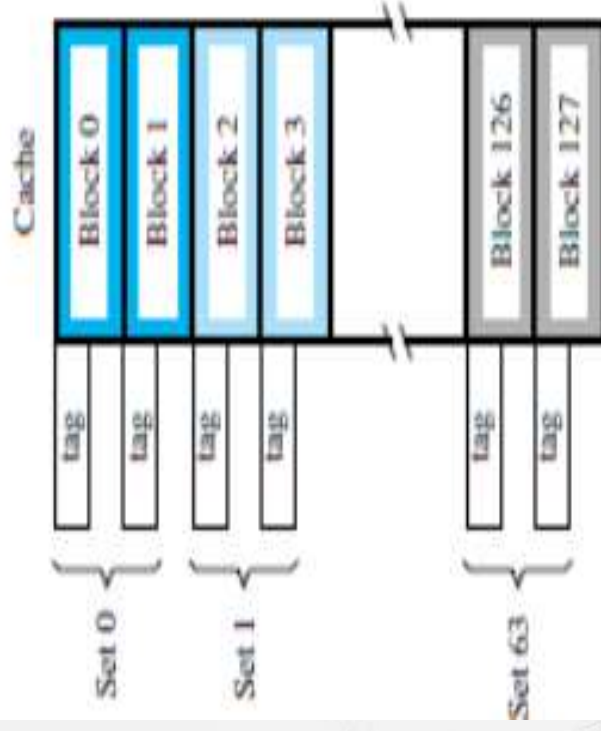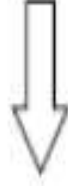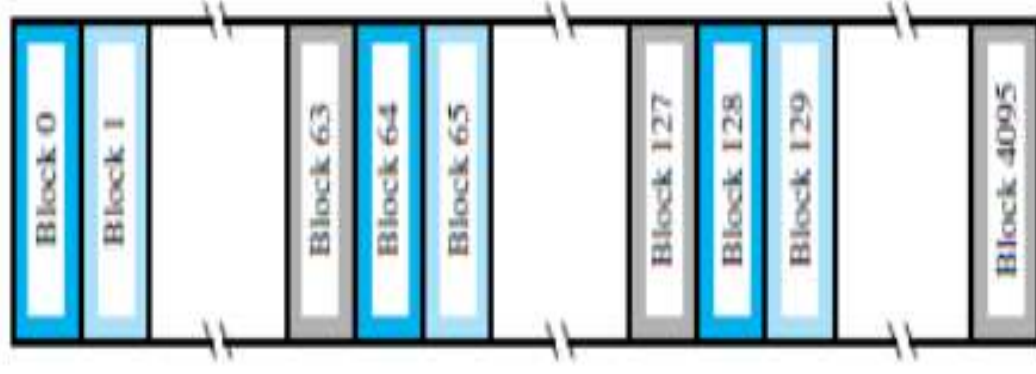
Tag	Set	Word
6	6	4

Replacement Algorithms

- In a direct-mapped cache, the position of each block is predetermined by its address; hence, the replacement strategy is trivial. In associative and set-associative caches there exists some flexibility. When a new block is to be brought into the cache and all the positions that it may occupy are full, the cache controller must decide which of the old blocks to overwrite.

- This is an important issue, because the decision can be a strong determining factor in system performance. In general, the objective is to keep blocks in the cache that are likely to be referenced in the near future. But, it is not easy to determine which blocks are about to be referenced.

- The property of locality of reference in programs gives a clue to a reasonable strategy. Because program execution usually stays in localized areas for reasonable periods of time, there is a high probability that the blocks that have been referenced recently will be referenced again soon.

- Therefore, when a block is to be overwritten, it is sensible to overwrite the one that has gone the longest time without being referenced.

- This block is called the *least recently used* (LRU) block, and the technique is called the *LRU replacement algorithm*.

- The LRU algorithm has been used extensively. Although it performs well for many access patterns, it can lead to poor performance in some cases

Least Recently Used (LRU)–

- In this algorithm page will be replaced which is least recently used.

- **Example-**Consider the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2 with 4 page frames. Find number of page faults.

Page reference   7,0,1,2,0,3,0,4,2,3,0,3,2,3   No. of Page frame - 4

7	0	1	2	0	3	0	4	2	3	0	3	2	3
			2	2	2	2	2	2	2	2	2	2	2
		1	1	1	1	1	4	4	4	4	4	4	4
	0	0	0	0	0	0	0	0	0	0	0	0	0
7	7	7	7	7	3	3	3	3	3	3	3	3	3

Miss Miss Miss Miss Hit Miss Hit Miss Hit Hit Hit Hit Hit Hit

Total Page Fault = 6

24

Write Policies

- The write operation is proceeding in 2 ways.
  1. Write-through protocol
  2. Write-back protocol

- **Write-through protocol:**
  Here the cache location and the main memory locations are updated simultaneously.

- **Write-back protocol:**
  ☐ This technique is to update only the cache location and to mark it as with associated flag bit called dirty/modified bit.
  ☐ The word in the main memory will be updated later, when the block containing this marked word is to be removed from the cache to make room for a new block.
  To overcome the read miss Load –through / Early restart protocol is used.