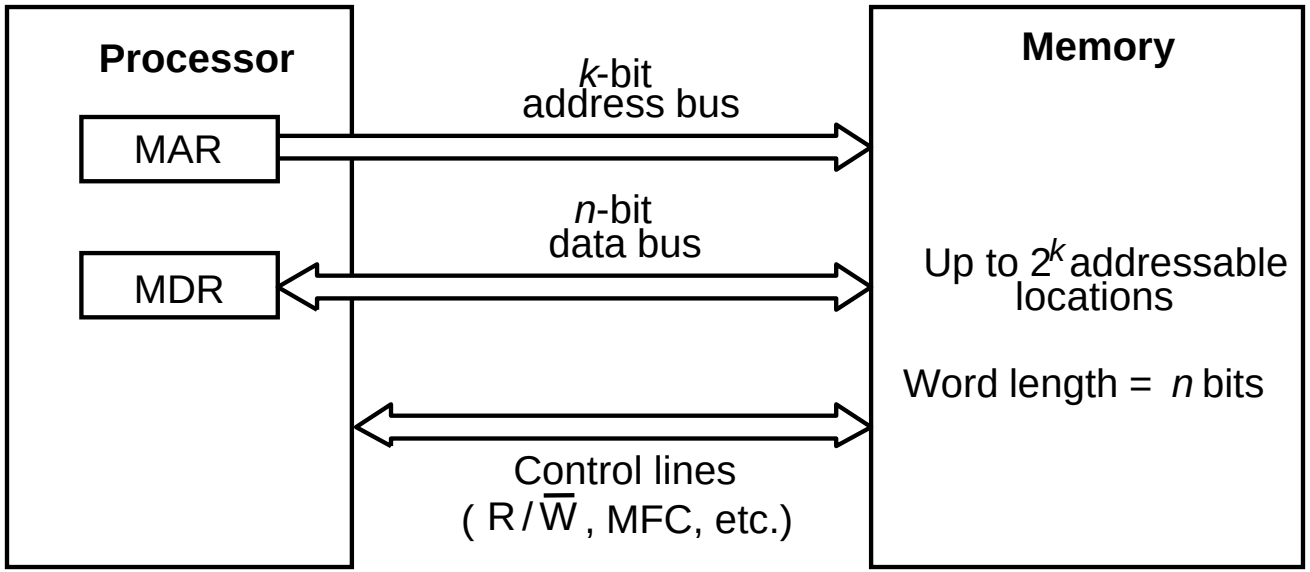
Module 4
Memory System Organization and Architecture

Introduction

- Programs and the data that processor operate are held in the main memory of the computer during execution, and the data with high storage requirement is stored in the secondary memories.

Memory Basic Concepts

- Maximum size of the Main Memory
- byte-addressable
- CPU-Main Memory Connection



Memory Basic Concepts

- Measures for the speed of a memory:
  - Memory access time.
  - Memory cycle time.
- An important design issue is to provide a computer system with as large and fast a memory as possible, within a given cost target.
- Several techniques to increase the effective size and speed of the memory:
  - Cache memory (to increase the effective speed).
  - Virtual memory (to increase the effective size).

Memory Hierarchy / Multilevel Memory

- Memory unit is essential component of digital computer since it is needed for storing programs and data.
- Memory unit that communicates directly with CPU is called Main memory.
- Devices that provide backup storage is called auxiliary memory.
- Only programs and data currently needed by processor reside in the main memory.
- All other information is stored in auxiliary memory and transferred to main memory when needed.

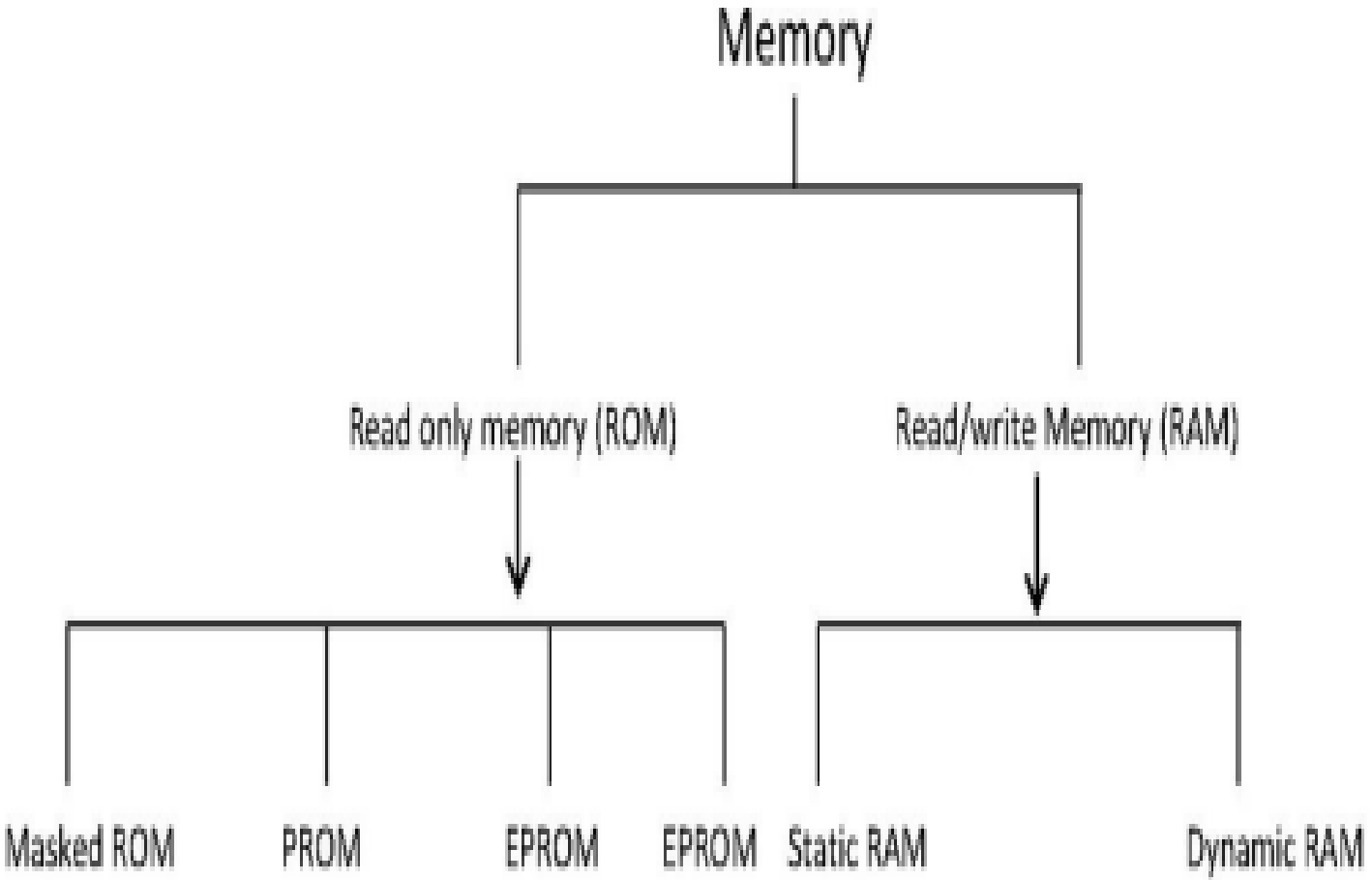
Main Memory

- It is the memory used to store programs and data during the computer operation.
- The principal technology is based on semiconductor integrated circuits.
- It consists of RAM and ROM chips.
- RAM chips are available in two form static and dynamic.

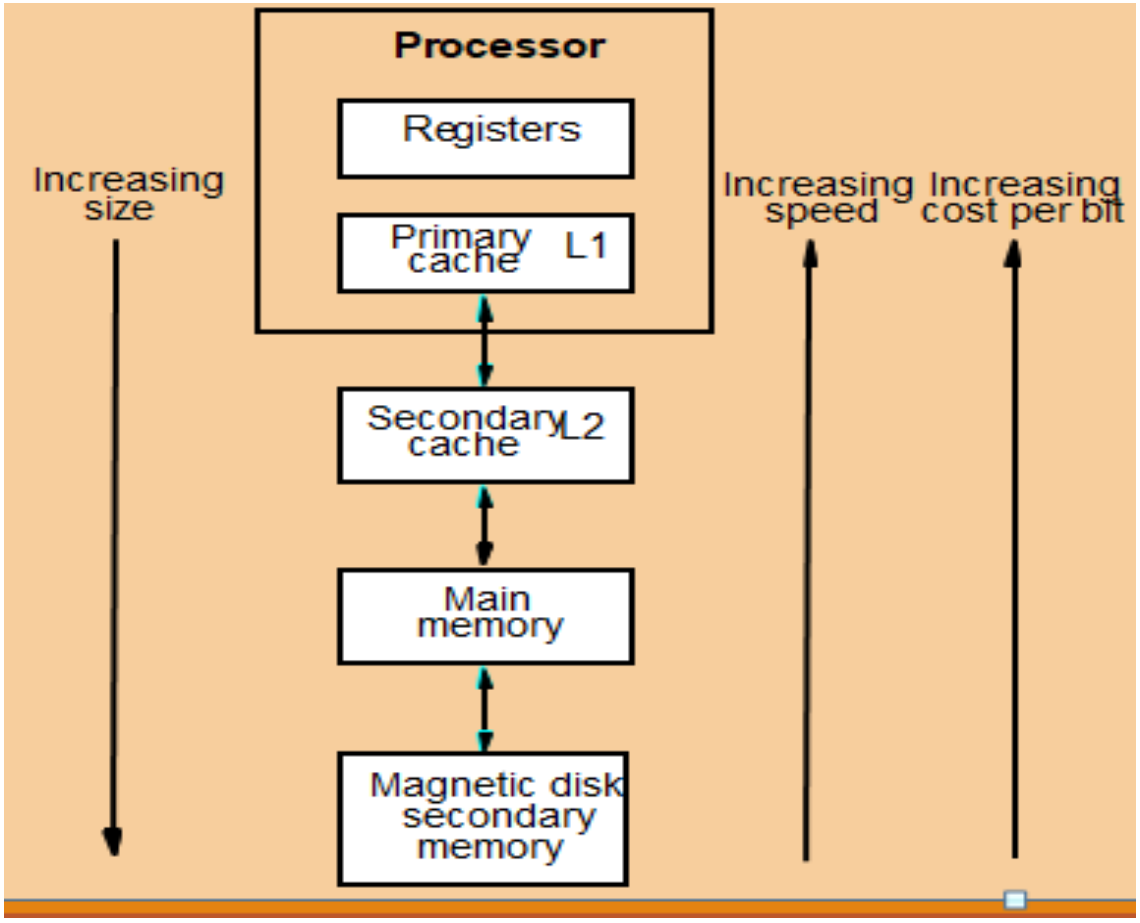
Static RAM	Dynamic RAM
➢ SRAM uses transistor to store a single bit of data	➢ DRAM uses a separate capacitor to store each bit of data
➢ SRAM does not need periodic refreshment to maintain data	➢ DRAM needs periodic refreshment to maintain the charge in the capacitors for data
➢ SRAM's structure is complex than DRAM	➢ DRAM's structure is simplex than SRAM
➢ SRAM are expensive as compared to DRAM	➢ DRAM's are less expensive as compared to SRAM
➢ SRAM are faster than DRAM	➢ DRAM's are slower than SRAM
➢ SRAM are used in Cache memory	➢ DRAM are used in Main memory

Main Memory

- ROM uses Read Only Mode.
- It is used for storing programs that are permanent and the tables of constants that do not change.
- ROM store program called bootstrap loader whose function is to start the computer software when the power is turned on.
- When the power is turned on, the hardware of the computer sets the program counter to the first address of the bootstrap loader


Memory
Read only memory (ROM)
Read/write Memory (RAM)
Masked ROM
PROM
EPROM
EPROM
Static RAM
Dynamic RAM

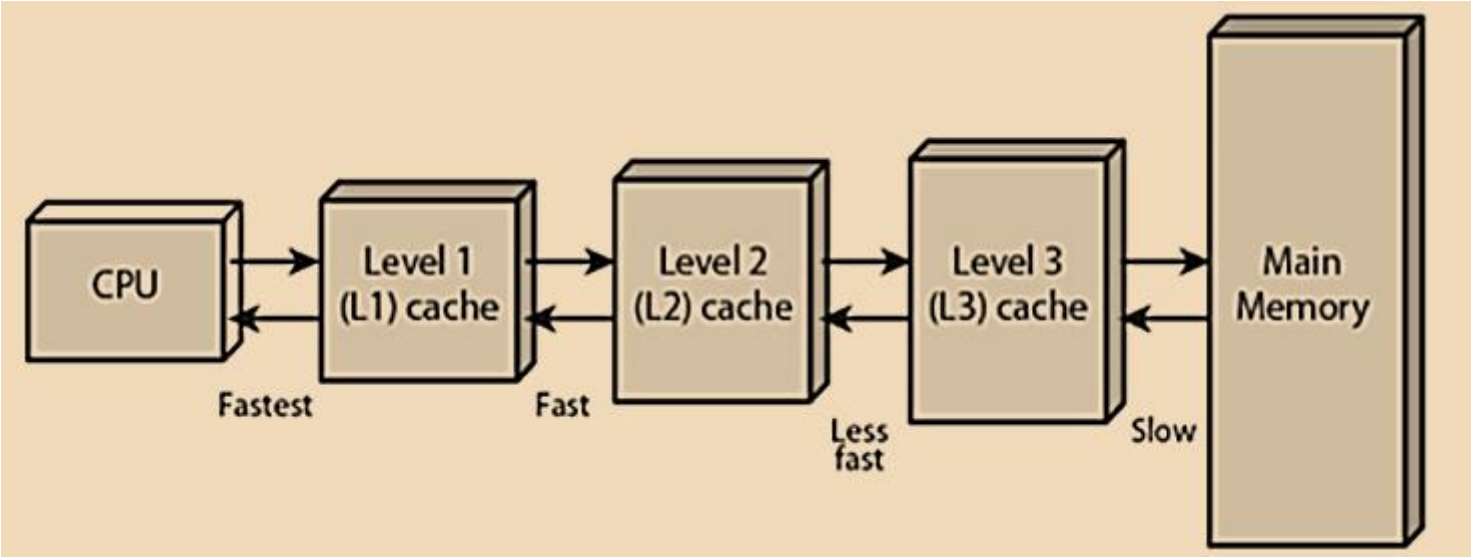
Memory Hierarchy



Memory Hierarchy(Levels of memory)

- **Level 1 or Register :** Data is stored and accepted that are immediately stored in CPU. Most commonly used registers are accumulator, Program counter, address register etc.
- **Level 2 or Cache memory:** Fastest memory which has faster access time where data is temporarily stored.
- **Level 3 or Main Memory: M**emory on which computer works currently. It is small in size and once power is off data no longer stays in this memory.
- **Level 4 or Secondary Memory: E**xternal memory which is not as fast as main memory but larger and data stays permanently in this memory.

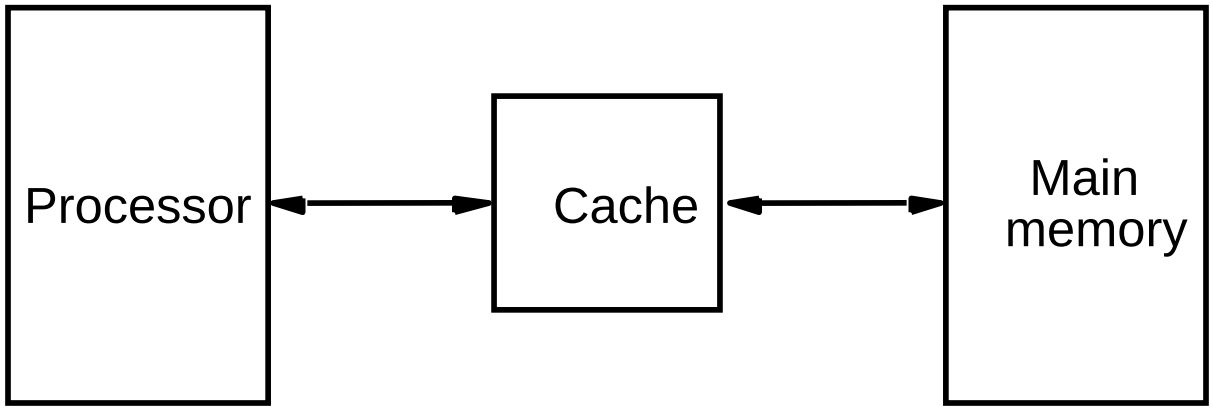
Cache Memory (Levels of memory)



Cache Memory

- A special very high-speed memory.
- Used to speed up and synchronizing with high-speed CPU.
- Costlier than main memory or disk memory but economical than CPU registers.
- Extremely fast memory type that acts as a buffer between RAM and the CPU. It holds frequently requested data and instructions so that they are immediately available to the CPU when needed.
- Used to reduce the average time to access data from the Main memory.
- Smaller and faster memory which stores copies of the data from frequently used main memory locations.
- Various different independent caches in a CPU, which store instructions and data.

Cache Memory Management



- *Processor issues a Read request, a block of words is transferred from the main memory to the cache, one word at a time.*
- *Subsequent references to the data in this block of words are found in the cache.*
- *At any given time, only some blocks in the main memory are held in the cache. Which blocks in the main memory are in the cache is determined by a "mapping function".*
- *When the cache is full, and a block of words needs to be transferred from the main memory, some block of words in the cache must be replaced. This is determined by a "replacement algorithm".*

Locality of Reference

- Analysis of programs indicates that many instructions in localized areas of a program are executed repeatedly during some period of time, while the others are accessed relatively less frequently.
- Temporal locality of reference:
  - Recently executed instruction is likely to be executed again very soon.
- Spatial locality of reference:
  - Instructions with addresses close to a recently instruction are likely to be executed soon.

Terminology

- Cache: memory closest to the processor in a memory hierarchy
- Caching: any storage management technique exploiting locality
- Upper/lower level: memory closer/further from the processor
- Block: unit of memory transfer (block of words) between two levels in a memory hierarchy. Also called a **cache line /index**

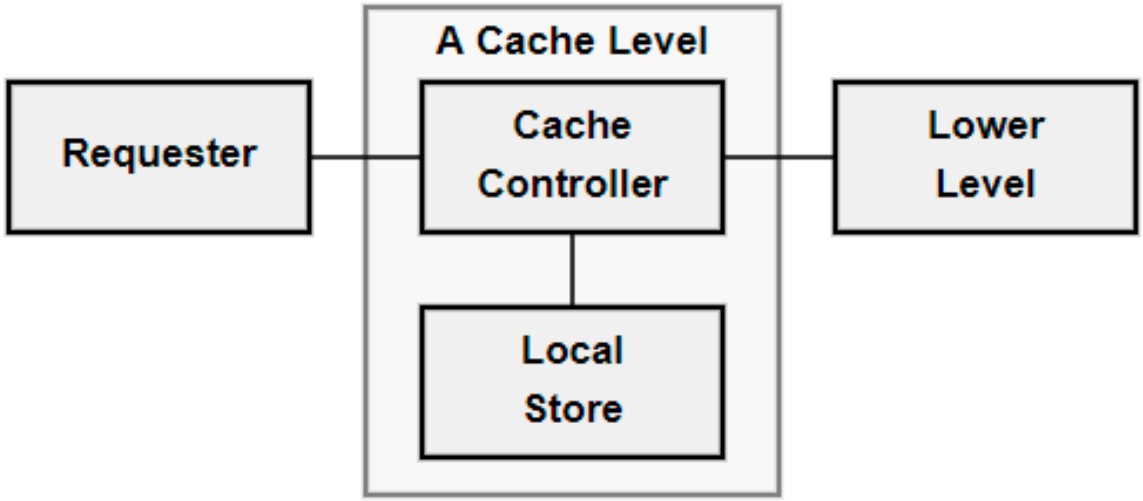
Cache Performance

- **Cache hit:** If the processor finds the memory location it is searching
- **Cache miss:** If the processor **does not** find the intended memory location
- The performance of cache memory is frequently measured in terms of a quantity called **Hit ratio.**

Hit ratio = hit / (hit + miss)= Number of hits/total accesses

- **Miss penalty :** time to move a block from a lower level in the hierarchy and satisfy the processor's request

Cache Access Time



Effective access time is a standard effective average.

effective-access-time = hit-rate * cache-access-time + miss-rate * lower-level-access-time

Miss penalty is defined as the difference between lower level access time and cache access time.

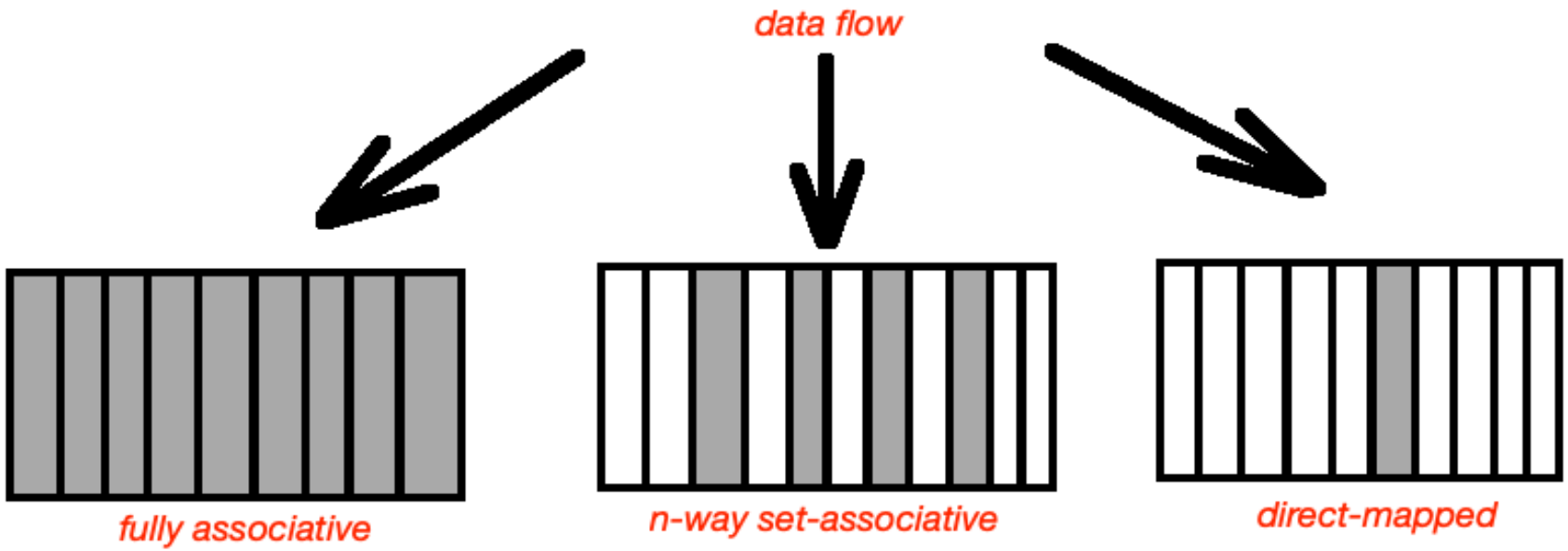
effective-access-time = hit rate * cache-access-time + miss-rate * miss-penalty

Average memory access time = Hit ratio x access time in cache memory + (1 – Hit ratio) x Access time in main memory

Cache Mapping

- Direct mapping, Associative mapping, and Set-Associative mapping

Three types of cache memory mapping

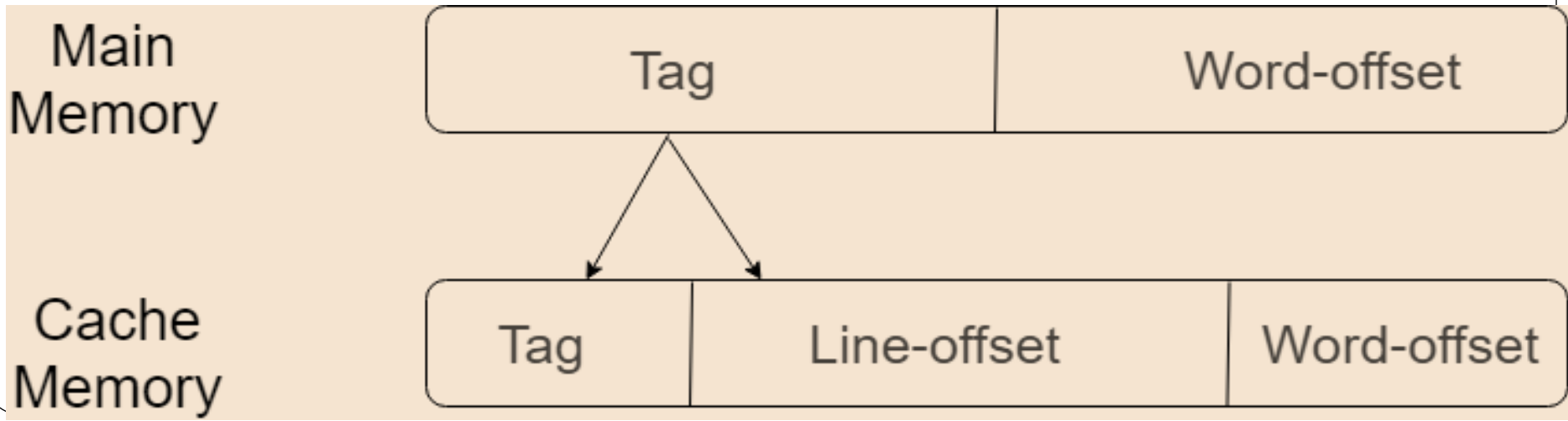


Three types of cache memory mapping

- In a fully associative cache every memory location can be cached in any cache line. This memory type significantly decreases amount of cache-line misses, considered as complex type of cache memory implementation.
- In direct-mapped cache memory location maps to a single cache line. It can be used once per address per amount of time. Performance of this cache memory type is lower than others.
- In N-way-set-associative cache, the most common cache implementation, memory address can be stored in any N lines of cache.

Cache Mapping: Direct mapping

- Maps each block of main memory into only one possible cache line.
- Index =Line number = block number
  - A simple processor example:
    - Cache consisting of 128 blocks of 16 words each.
    - Total size of cache is 2048 (2K) words.
    - Main memory is addressable by a 16-bit address.
    - Main memory has 64K words.
    - Main memory has 4K blocks of 16 words each.



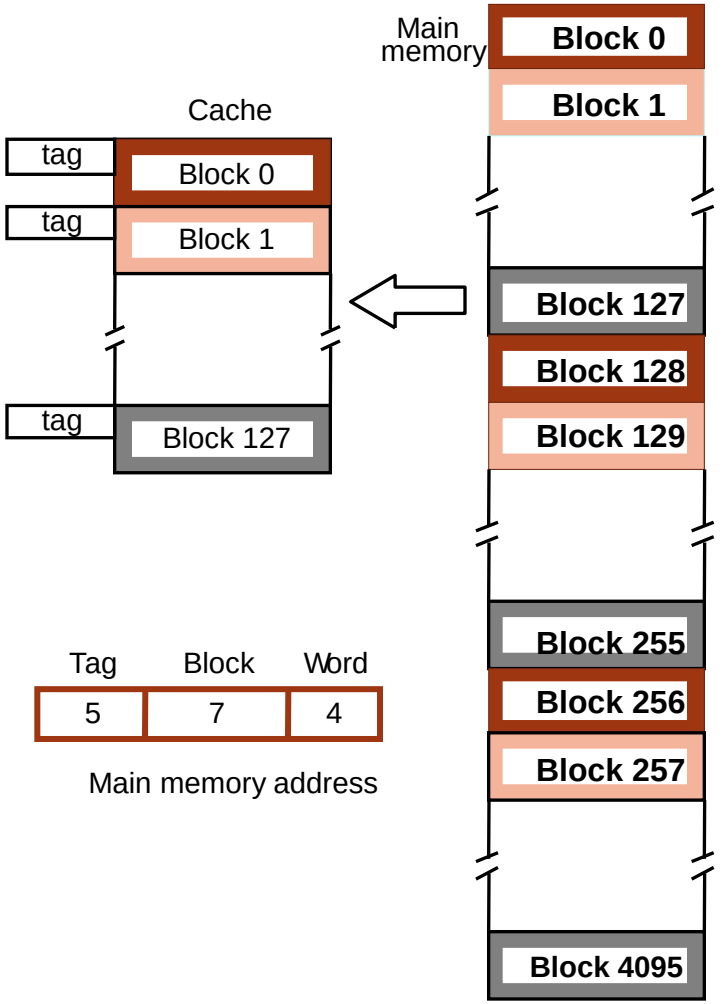
Direct-Mapped cache

- A particular block of main memory can be mapped to one particular cache line only.
- Block ‘j’ of main memory will map to line number (j mod number of cache lines) of the cache.
- There is no need of any replacement algorithm.

```
i = j modulo m
where
i=cache line number
j= main memory block number
m=number of lines in the cache
```

Blocks in a Direct-Mapped cache

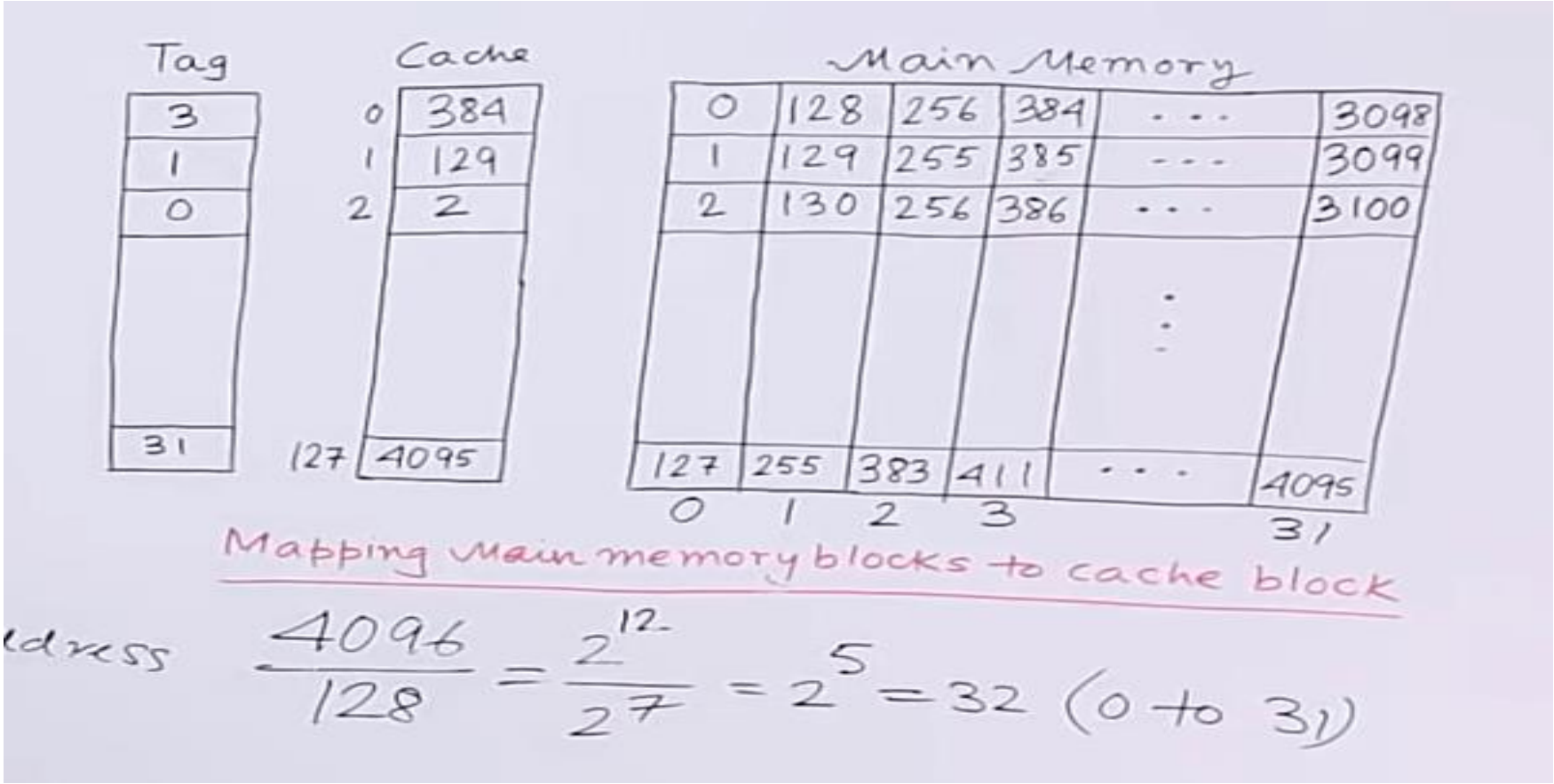
One block = the ***whole*** **cache**



- *Block j of the memory is mapped to block jmodulo128 of the cache*
- *Placement of block in cache is determined from the memory address which has 3 fields:*
- *Lower4bits—to select one of the16words in a block*
- *7 bits to determine the position of the block*
- *5-bit tag to determine which of the 32 blocks mapped to the same position are resident in the cache*

Partitioning of the memory into block in Direct-Mapped cache

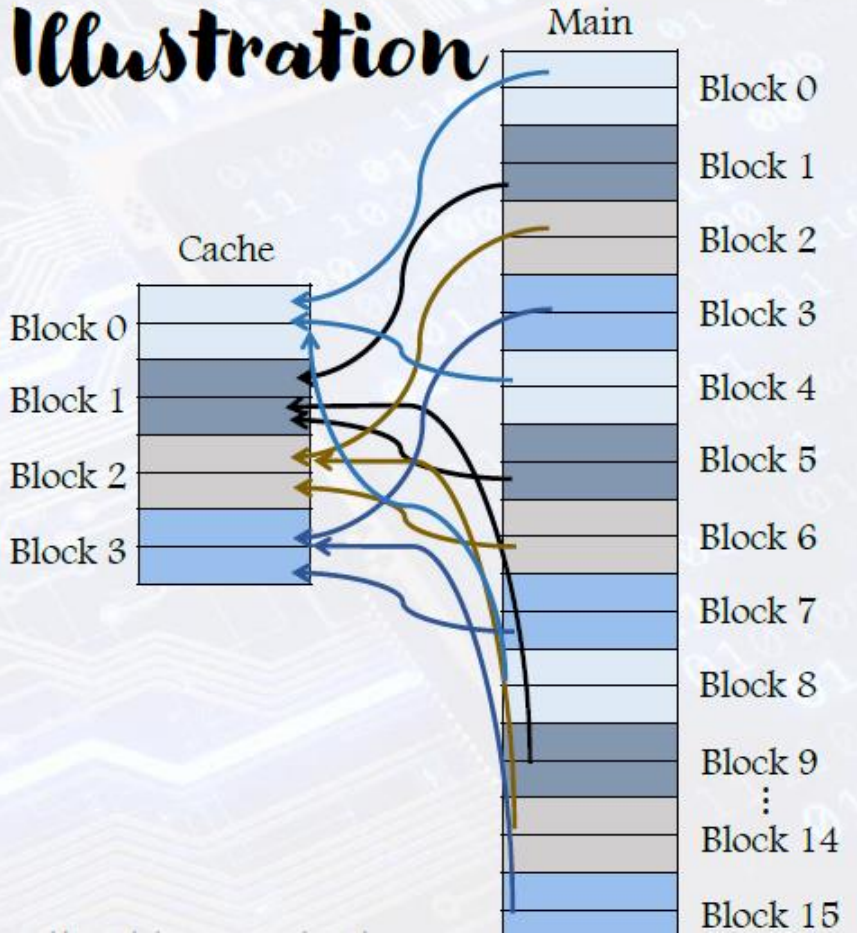
- **Block size** that are **equal** to the **size** of the **Direct-mapped cache**



Direct Mapping: Illustration

- Cache size = 8 bytes
- Block size = 2 bytes
- Memory size = 32 bytes

Tag	Block	Word
2	2	1



Consider a direct mapped cache of size 16 KB with block size 256 bytes. The size of main memory is 128 KB. Find the number of bits in tag and Tag directory size.

- **Solution:**

Given:-

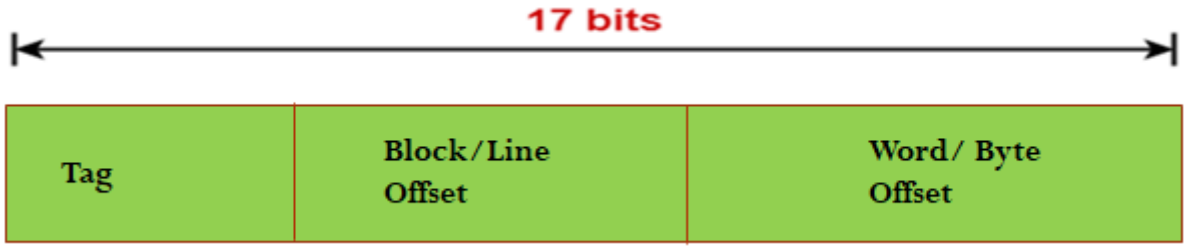
- Cache memory size = 16 KB
- Block size = Frame size = Line size = 256 bytes
- Main memory size = 128 KB

- Consider that the memory is byte addressable
- **Number of Bits in Physical Address-**
- Size of main memory= 128 KB = 2^{17} bytes
- Thus, Number of bits in physical address = 17 bits

Consider a direct mapped cache of size 16 KB with block size 256 bytes. The size of main memory is 128 KB. Find the number of bits in tag and Tag directory size.

- **Solution:**

Number of bits in physical address = 17 bits



Consider a direct mapped cache of size 16 KB with block size 256 bytes. The size of main memory is 128 KB. Find the number of bits in tag and Tag directory size.

- **Number of Bits in Block Offset-**
- Block size = 256 bytes = 2^8 bytes
- Thus, Number of bits in block offset = 8 bits

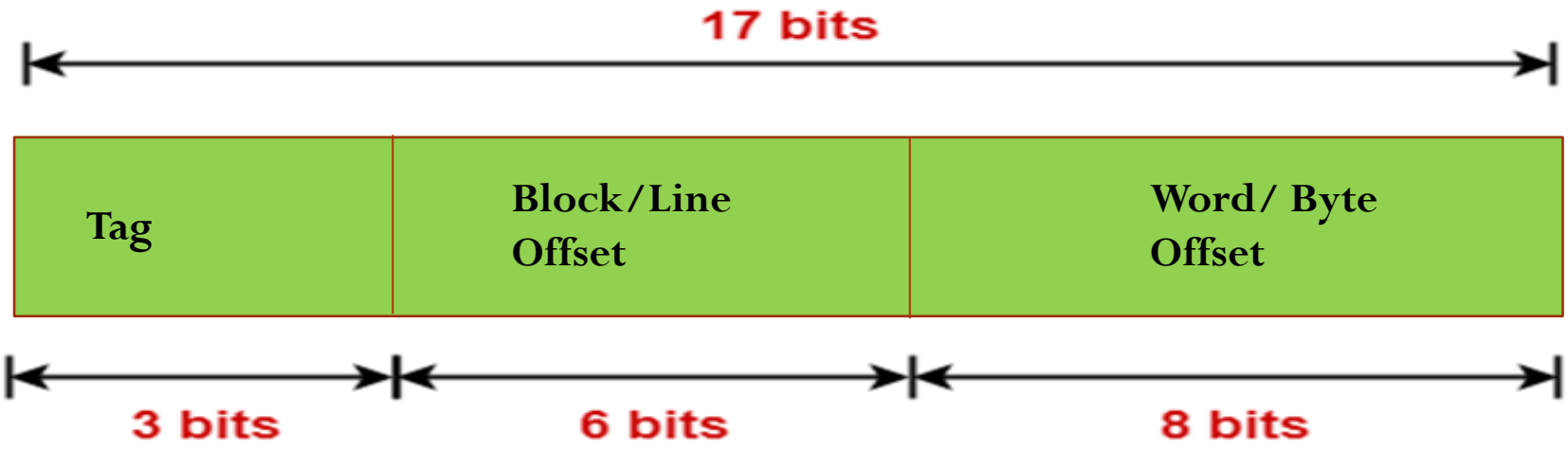
Consider a direct mapped cache of size 16 KB with block size 256 bytes. The size of main memory is 128 KB. Find the number of bits in tag and Tag directory size.

- **Number of Bits in Line Number-**

Total number of lines in cache = Cache size /Line size

- = 16 KB / 256 bytes
- = 2^{14} bytes / 2^{8} bytes = = 2^{6} lines
- Thus, Number of bits in line number = 6 bits

- **Number of Bits in Tag-**
- = Number of bits in physical address – (Number of bits in line number + Number of bits in block offset)
- = 17 bits – (6 bits + 8 bits) = 17 bits – 14 bits
- = 3 bits
- Thus, Number of bits in tag = 3 bits



- **Tag Directory Size**
- = Number of tags x Tag size
- = Number of lines in cache x Number of bits in tag
- = 2^6 x 3 bits
- = 192 bits
- = 24 bytes
- Thus, size of tag directory = 24 bytes

Problem-02: Consider a direct mapped cache of size 512 KB with block size 1 KB. There are 7 bits in the tag. Find Size of main memory, Tag directory size and find to which cache line block 127 will be mapped to?

- **Solution**
- Given:
- Cache memory size = 512 KB
- Block size = Frame size = Line size = 1 KB
- Number of bits in tag = 7 bits
- Consider that the memory is byte addressable.

- **Number of Bits needed to find Block Offset-**
- Block size = 1 KB = 2^{10} bytes
- Thus, Number of bits in word offset = 10 bits

Number of Bits in Line Number

- Cache size / Line size = 512 KB / 1 KB = 2^{9} lines
- Number of bits in line number = 9 bits

Number of bits in physical address

= Number of bits in tag + Number of bits in line number + Number of bits in block offset

- = 7 bits + 9 bits + 10 bits = 26 bits

Number of bits in physical address = 26 bits

Size of Main Memory-

- Number of bits in physical address = 26 bits
- Thus, Size of main memory = 2^{26} bytes = 64 MB
- **Tag Directory Size-**
- Tag directory size = Number of tags x Tag size

= Number of lines in cache x Number of bits in tag = = 2^9 x 7 bits

- = 3584 bits = 448 bytes
- Thus, size of tag directory = 448 bytes
- **Mapping**
- Cache lines = 2^9 = 512 lines

= 127 modulo 512 = 127th cache line

3. Consider a direct mapped cache with block size 4 KB. The size of main memory is 16 GB and there are 10 bits in the tag. Find the Size of cache memory and Tag directory size.

- Block size = Frame size = Line size = 4 KB
- Size of main memory = 16 GB
- Number of bits in tag = 10 bits

Number of Bits in Physical Address ?

Number of Bits in Block Offset ?

Number of Bits in Line Number ?

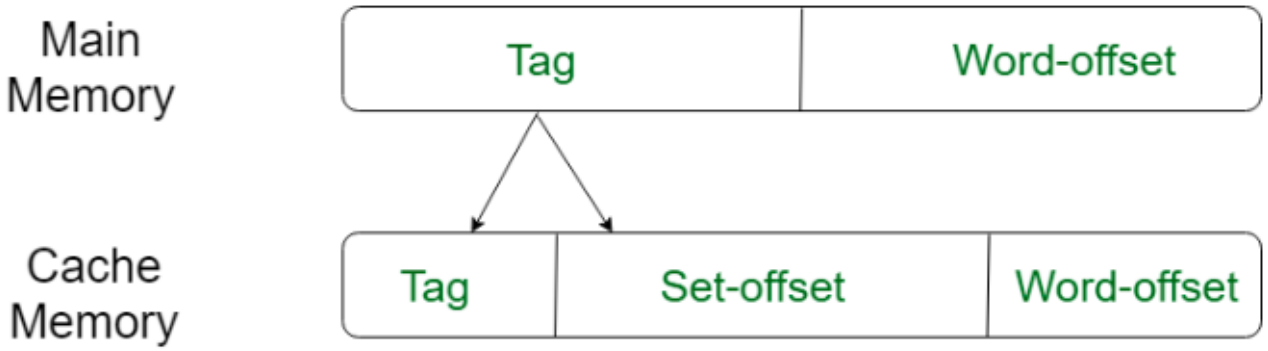
Number of Lines in Cache ?

Size of Cache Memory?

Tag Directory Size?

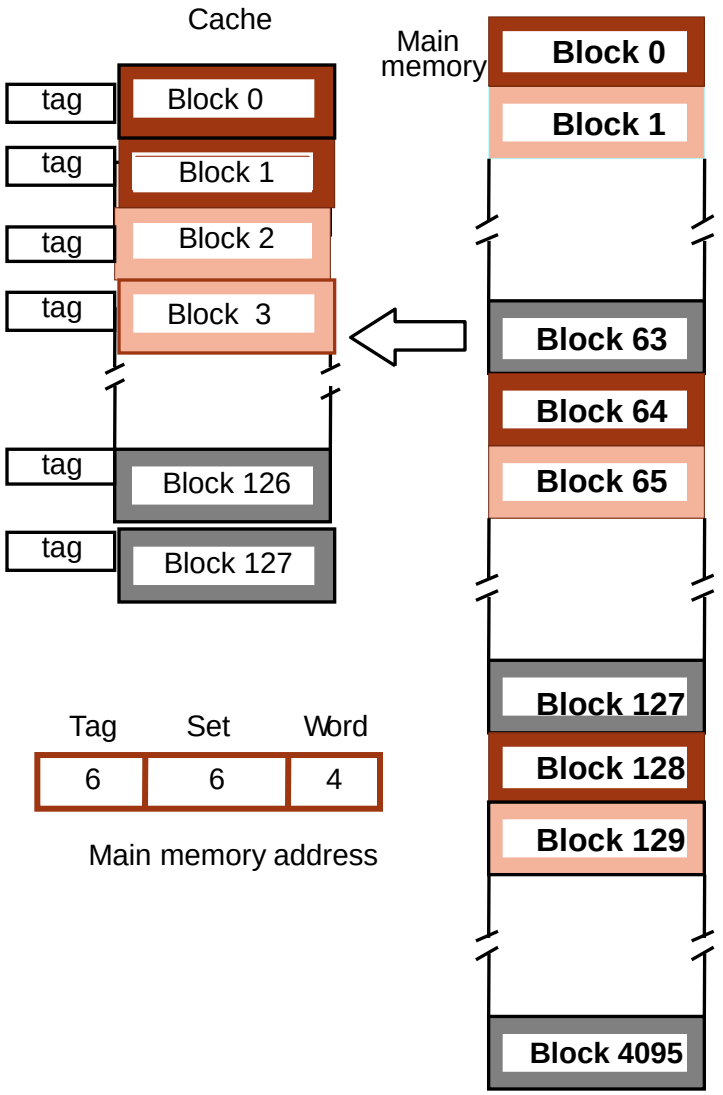
Set Associative Mapping

- A particular block of main memory can be mapped to one particular cache set only.
- Block 'j' of main memory will map to set number (j mod number of sets in cache) of the cache.
- Set-associative mapping combination of direct and associative mapping.
- A replacement algorithm is needed if the cache is full.
- Set number = number of blocks in the set= index



- *Number of blocks per set is a design parameter.*
  - *One extreme is to have all the blocks in one set, requiring no set bits (fully associative mapping).*
  - *Other extreme is to have one block per set, is the same as direct mapping*

Set-Associative mapping



- *Blocks of cache are grouped into sets.*
- *Mapping function allows a block of the main memory to reside in any block of a specific set.*
- *For a 2-way set associative,*
  Divide the cache into 64 sets, with two blocks per s
- *Memory block 0, 64, 128 etc. map to block 0, and t can occupy either of the two positions.*
- *Memory address is divided into three fields:*
  - *Lower order 4 bits to identify the word or byte*
  - *6 bit field determines the set number.*
  - *High order 6 bit fields are compared to the tag fields of the two blocks in a set.*
- *I = j modulo 64 (as the set size is 64)*
- *More choices for block placement than direct mappi*
- *Cost of associative search is reduced*

Consider a 2-way set associative mapped cache of size 16 KB with block size 256 bytes. The size of main memory is 128 KB. Find Number of bits in tag and Tag directory size

- Given: Set size = 2, Cache memory size = 16 K, Block size = Frame size = Line size = 256 bytes, Main memory size = 128 KB, consider that the memory is byte addressable.
- **Number of Bits in Physical Address-**
- Size of main memory= 128 KB = 2^{17} bytes
- Thus, Number of bits in physical address = 17 bits

Consider a 2-way set associative mapped cache of size 16 KB with block size 256 bytes. The size of main memory is 128 KB. Find Number of bits in tag and Tag directory size

Number of Bits in Block Offset-

- Block size = 256 bytes = 2^8 bytes
- Thus, Number of bits in block offset = 8 bits

- **Number of Lines in Cache-**

Total number of lines in cache = Cache size /Line size

= 16 KB / 256 bytes = 2^{14} bytes / 2^8 bytes = 64 lines

Consider a 2-way set associative mapped cache of size 16 KB with block size 256 bytes. The size of main memory is 128 KB. Find Number of bits in tag and Tag directory size

- **Number of Sets in Cache-**

Total number of sets in cache

- = Total number of lines in cache / Set size
- = 64 / 2 = 32 sets = 2^5 sets
- Thus, Number of bits in set number = 5 bits

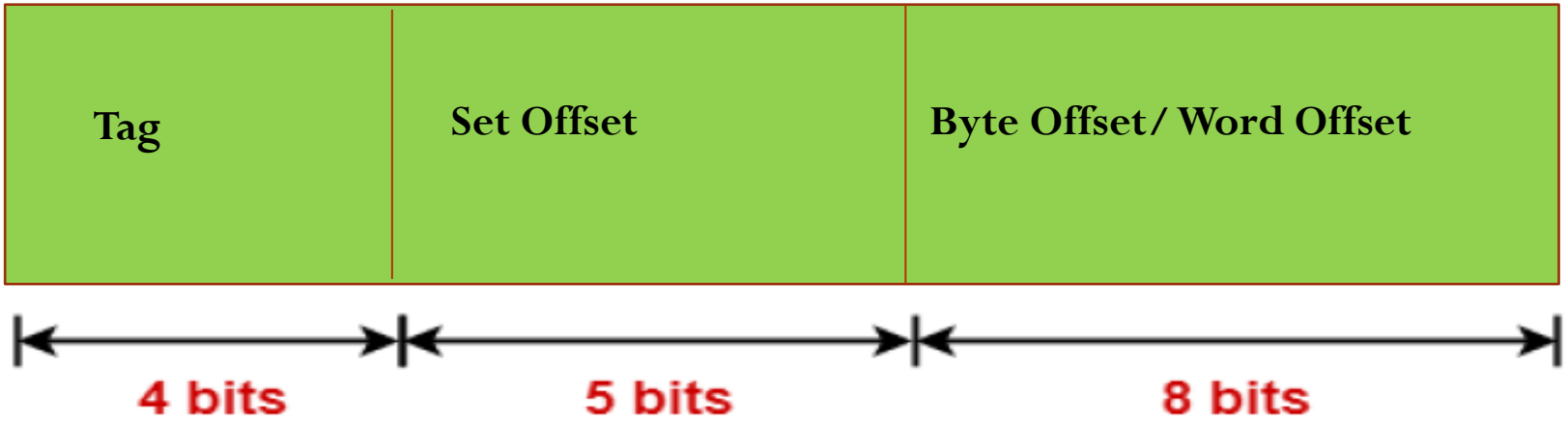
Consider a 2-way set associative mapped cache of size 16 KB with block size 256 bytes. The size of main memory is 128 KB. Find Number of bits in tag and Tag directory size

- **Number of Bits in Tag-**

= Number of bits in physical address – (Number of bits in set number + Number of bits in block offset)

= 17 bits – (5 bits + 8 bits) = 17 bits – 13 bits = 4 bits

- Thus, Number of bits in tag = 4 bits



Consider a 2-way set associative mapped cache of size 16 KB with block size 256 bytes. The size of main memory is 128 KB. Find Number of bits in tag and Tag directory size

- **Tag Directory Size-**

= Number of tags x Tag size

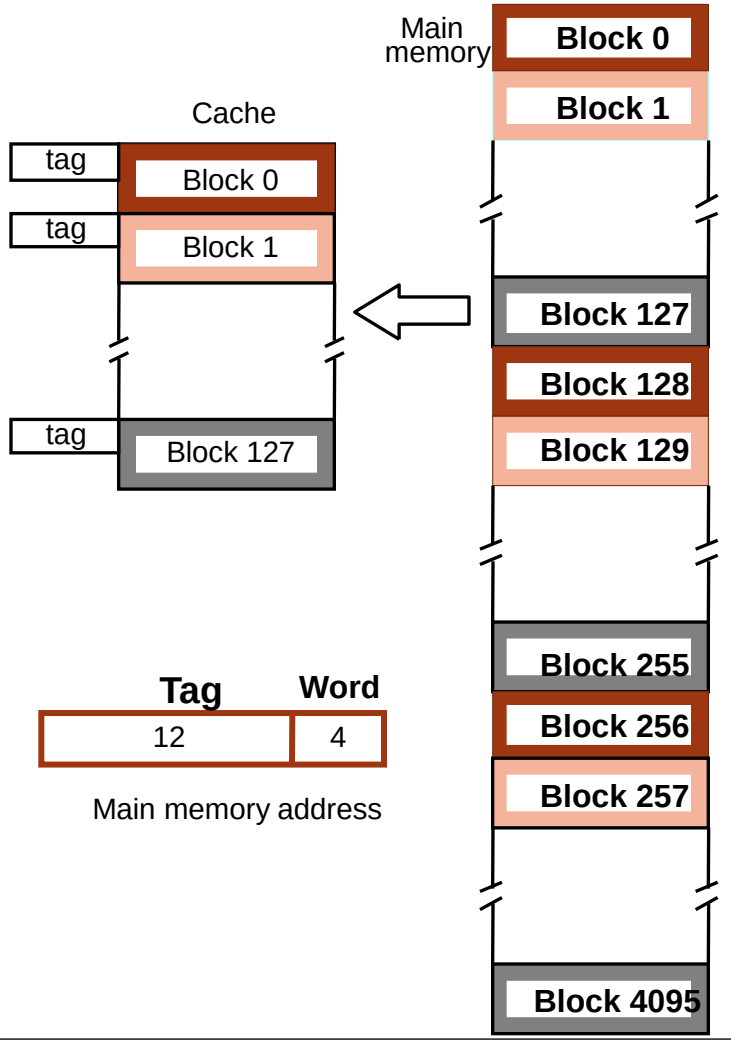
- = Number of lines in cache x Number of bits in tag
- = 64 x 4 bits = 256 bits = 32 bytes
- Thus, size of tag directory = 32 bytes

Fully Associative Mapping

- A block of main memory can be mapped to any freely available cache line.
- This makes fully associative mapping more flexible than direct mapping.
- A replacement algorithm is needed to replace a block if the cache is full.

Tag (Block number)	Word/ Byte Offset

Associative mapping



- *Main memory block can be placed into any cache position.*
- *Memory address is divided into two fields:*
  - *Low order 4 bits identify the word within a block.*
  - *High order 12 bits or tag bits identify a memory block when it is resident in the cache.*
- *Flexible, and uses cache space efficiently.*
- *Replacement algorithms can be used to replace an existing block in the cache when the cache is full.*
- *Cost is higher than direct-mapped cache because of the need to search all 128 patterns to determine whether a given block is in the cache.*

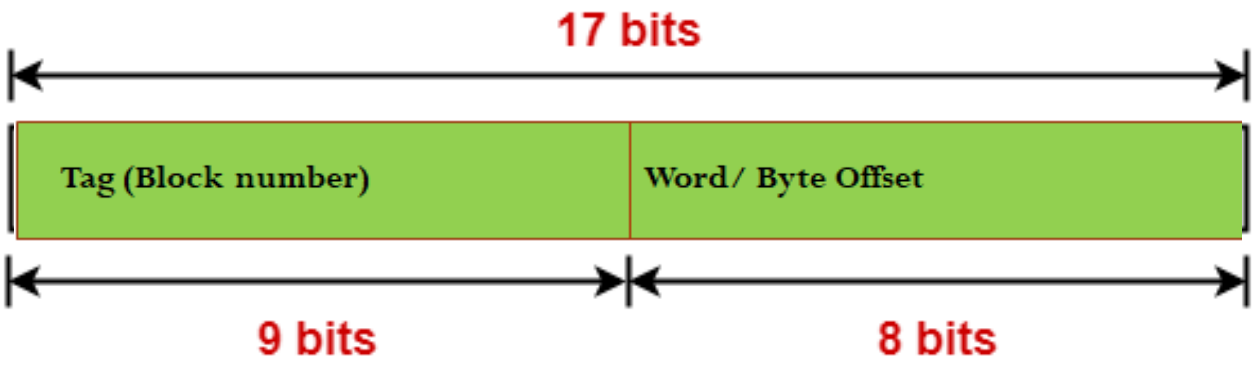
1.Consider a fully associative mapped cache of size 16 KB with block size 256 bytes. The size of main memory is 128 KB. Find the number of bits in tag, Tag directory size.

- Cache memory size = 16 KB
- Block size = Frame size = Line size = 256 bytes
- Main memory size = 128 KB
- Size of main memory= 128 KB = 2^{17} bytes
- Number of bits in physical address = 17 bits

Number of Bits in Block Offset

- Block size = 256 bytes = 2^8 bytes
- Number of bits in block offset = 8 bits

- **Number of Bits in Tag**



Number of Lines in Cache

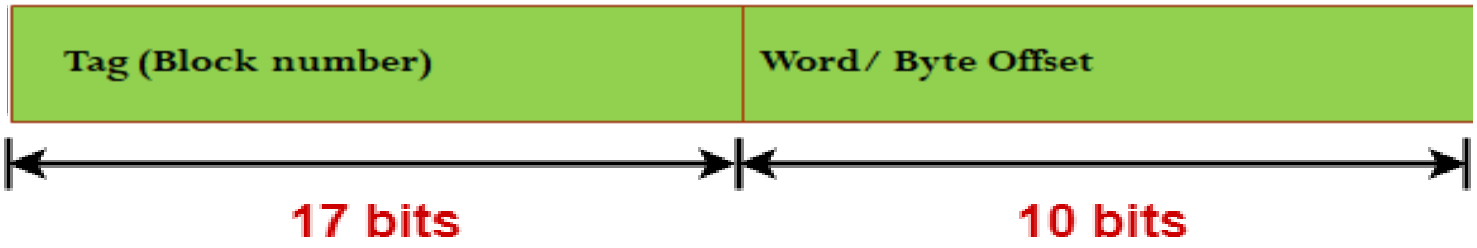
- Total number of lines in cache = Cache size/Line size
- = 16 KB / 256 bytes = 2^{14} bytes / 2^{8} bytes = 2^{6} lines
- **Tag Directory Size** = Number of tags x Tag size
- = Number of lines in cache x Number of bits in tag
- = 2^{6} x 9 bits = 576 bits = 72 bytes
- Size of tag directory = 72 bytes

2.Consider a fully associative mapped cache of size 512 KB with block size 1 KB. There are 17 bits in the tag. Find the size of main memory, Tag directory size

- Cache memory size = 512 KB
- Block size = Frame size = Line size = 1 KB
- Number of bits in tag = 17 bits

Consider a fully associative mapped cache of size 512 KB with block size 1 KB. There are 17 bits in the tag. Find the size of main memory, Tag directory size

- Cache memory size = 512 KB
- Block size = Frame size = Line size = 1 KB
- Number of bits in tag = 17 bits
- **Number of Bits in Block Offset**
- Block size = 1 KB = 2^{10} bytes
- Thus, Number of bits in block offset = 10 bits



- **Number of Bits in Physical Address**

Number of bits in physical address = Number of bits in tag + Number of bits in block offset

- = 17 bits + 10 bits = **27 bits**

- **Size of Main Memory=** Number of bits in physical address = 27 bits
- Size of main memory = 2^{27} bytes = 128 MB

- **Number of Lines in Cache-**

Total number of lines in cache = Cache size / Line size

- = 512 KB / 1 KB = 512 lines = 2^9 lines

- **Tag Directory Size-**
- = Number of tags x Tag size
- = Number of lines in cache x Number of bits in tag
- = 2^9 x 17 bits = 8704 bits = 1088 bytes
- Thus, size of tag directory = 1088 bytes

Basic Concepts- Replacement

When the cache is full a memory word not in the cache is referenced, an existing block should be cleared to make room–replacement algorithm

•If a word to be read/written is found in the cache–read hit or write hit

•In case of a write operation: Write-through–Cache and main memory updated simultaneously

Dirty or modified bit–Only cache is updated, and associated flag bit is updated. Main memory updated later–write-back/copy-back

•In the case of a read miss: The block of words that contains the requested word can be copied in to cache and then the word can be forwarded to the processor.

To reduce the processor's waiting period, the word can be directly sent to the processor–load through/early restart

Replacement Algorithms

• For direct mapping where there is only one possible line for a block of memory, no replacement algorithm is needed.

• For associative and set associative mapping, however, an algorithm is needed.

•For maximum speed, this algorithm is implemented in the hardware. Four of the most common algorithms are:

1. Least Recently Used:- This replaces the candidate line in cache memory that has been there the longest with no reference to it.
2. First In First Out:- This replaces the candidate line in the cache that has been there the longest.
3. Least Frequently Used:- This replaces the candidate line in the cache that has had the fewest references.
4. Random Replacement:- This algorithm randomly chooses a line to be replaced from among the candidate lines. This yields only slightly inferior performance than other algorithms.

FIFO (First In First Out)

- Pages in main memory are kept in a list
- First in first out is very easy to implement
- The FIFO algorithm select the page for replacement that has been in memory the longest time

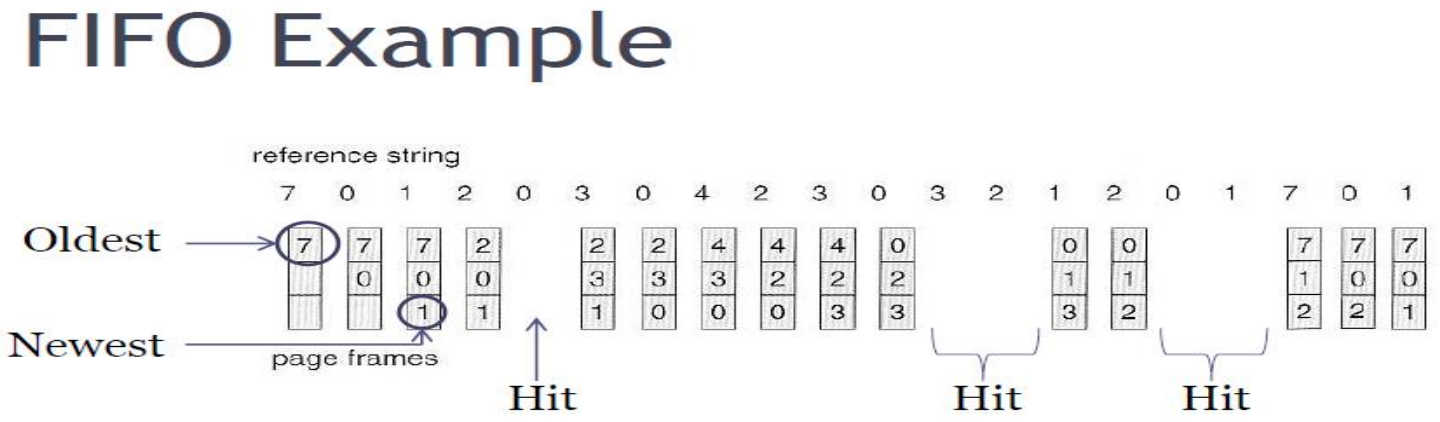


Fig: FIFO example

FIFO (First In First Out)

- Advantages:
  - FIFO is easy to understand.
  - It is very easy to implement.

- Disadvantages:
  - The oldest block in memory may be often used.

LRU (Least Recently Used)

- The least recently used page replacement algorithm keeps track page uses over a short period of time.

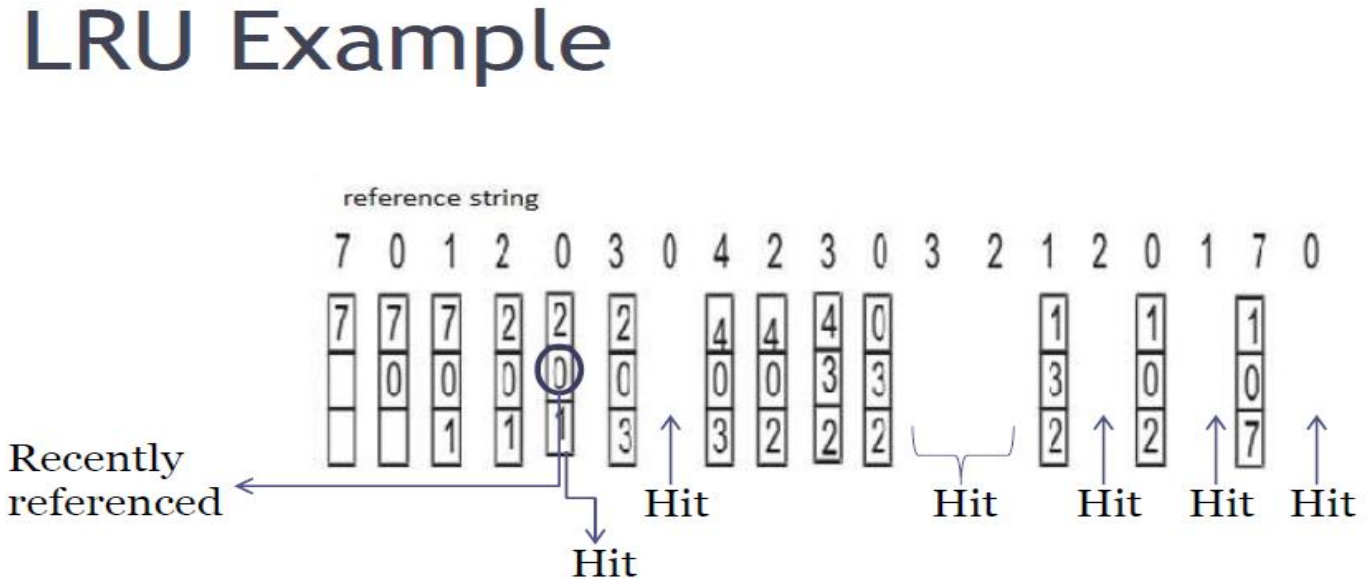


Fig: LRU example

LRU (Least Recently Used)

- Advantages:
  - LRU page replacement algorithm is quiet efficient.
- Disadvantages:
  - Implementation difficult. This algorithm requires keeping track of what was used when, which is expensive if one wants to make sure
  - the algorithm always discards the least recently used item.

LFU (Least Frequently Used)

- The Least-Frequently-Used (LFU) Replacement technique replaces the least-frequently block in use when an eviction must take place.
- Software counter associated with each block, initially zero is required in this algorithm.
- The operating system checks all the blocks in the cache at each clock interrupt.
- The R bit, which is '0' or '1', is added to the counter for each block. Consequently, the counters are an effort to keep track of the frequency of referencing each block.
- When a block must be replaced, the block that has the lowest counter is selected for the replacement.

LFU (Least Frequently Used)

Reference string:

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

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

Number of page faults = 12.
Number of page hits= 8.

LFU (Least Frequently Used)

- Advantages:
  - Frequently used block will stay longer than (fifo)
- Disadvantages:
  - Older blocks are less likely to be removed , even if they are on longer frequently used because this algorithm never forgets anything.
  - Newer blocks are more likely to be replaced even if they are frequently used.
  - Captures only frequency factor.

Random Replacement

- When we need to evict a page, choose one randomly
- Advantage:
  - Extremely simple
- Disadvantages:
  - Can easily make "bad" choices by swapping out pages right before they are needed.

Miss rate and Hit rate problems

Problem 1:

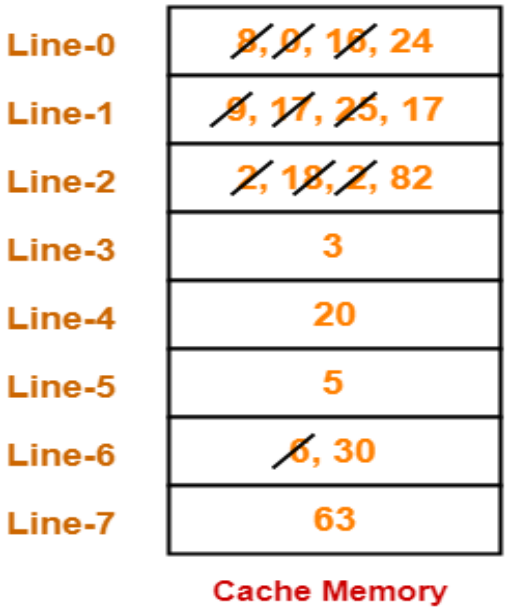
Consider a direct mapped cache with 8 cache blocks (0-7). If the memory block requests are in the order-
3, 5, 2, 8, 0, 6, 3, 9, 16, 20, 17, 25, 18, 30, 24, 2, 63, 5, 82, 17, 24
Which of the following memory blocks will be in the cache at the end of the sequence?

Also, calculate the hit ratio and miss ratio.

- Given,
- There are 8 blocks in cache memory numbered from 0 to 7.
- In direct mapping, a particular block of main memory is mapped to a particular line of cache memory.
- The line number is given by-
- Cache line number = Block address modulo Number of lines in cache

Solution-

- For the given sequence-
- Requests for memory blocks are generated one by one.
- The line number of the block is calculated using the above relation.
- Then, the block is placed in that particular line.
- If already there exists another block in that line, then it is replaced.



Cache Memory

Blocks requests in order
Calculation of line number

3 % 8 = 3	(Miss)
5 % 8 = 5	(Miss)
2 % 8 = 2	(Miss)
8 % 8 = 0	(Miss)
0 % 8 = 0	(Miss)
6 % 8 = 6	(Miss)
3 % 8 = 3	(Hit)
9 % 8 = 1	(Miss)
16 % 8 = 0	(Miss)
20 % 8 = 4	(Miss)
17 % 8 = 1	(Miss)
25 % 8 = 1	(Miss)
18 % 8 = 2	(Miss)
30 % 8 = 6	(Miss)
24 % 8 = 0	(Miss)
2 % 8 = 2	(Miss)
63 % 8 = 7	(Miss)
5 % 8 = 5	(Hit)
82 % 8 = 2	(Miss)
17 % 8 = 1	(Miss)
24 % 8 = 0	(Hit)

- Hit ratio = 3 / 20
- Miss ratio = 17 / 20

Problem2

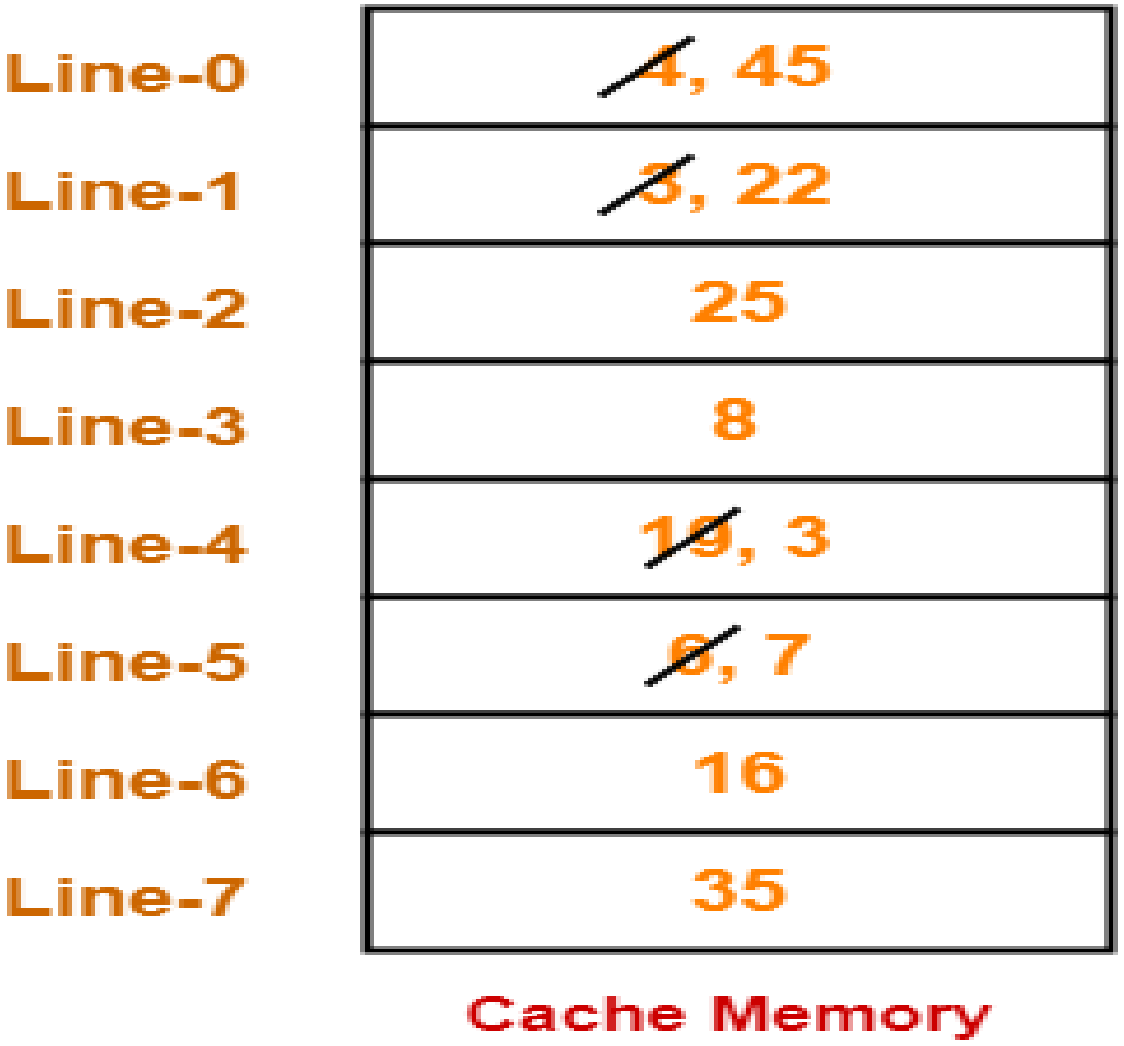
- Consider a fully associative cache with 8 cache blocks (0-7). The memory block requests are in the order-

4, 3, 25, 8, 19, 6, 25, 8, 16, 35, 45, 22, 8, 3, 16, 25, 7

- If LRU replacement policy is used, which cache block will have memory block 7?
- Also, calculate the hit ratio and miss ratio.

Solution:

- We have,
- There are 8 blocks in cache memory numbered from 0 to 7.
- In fully associative mapping, any block of main memory can be mapped to any line of the cache that is freely available.
- If all the cache lines are already occupied, then a block is replaced in accordance with the replacement policy.


Line-0
4, 45
Line-1
3, 22
Line-2
25
Line-3
8
Line-4
19, 3
Line-5
6, 7
Line-6
16
Line-7
35
Cache Memory

Thus,

- Line-5 contains the block-7.
- Hit ratio = 5 / 17
- Miss ratio = 12 / 17

Average Memory access time Problem

- A cache is having 60% hit ratio for read operation. Cache access time is 30 ns and main memory access time is 100 ns, 50% operations are read operation.
  What will be the average access time for read operation?

- Avg access time (for read operation) = Hit ratio * cache access time + miss ratio * (Cache access time + MM access time)
  Given, Cache access time is 30 ns and main memory access time is 100 ns.
  Hit ratio = 60% so Miss ratio = 40%
  so Avg access time = (0.6 * 30) + 0.4 * (30 + 100)
  = 18 + 52
  = 70
  so average access time for read operation is 70 ns.

Practice Problems

1. Interpret the main memory addresses FF010,12364,andC7691 considering direct, associative and 2 way set associative mapping if the main memory size is 1MB,word size is 16 bytes, and cache size is 64KB.

2. A byte-addressable computer has a small data cache capable of holding eight 32-bit words. Each cache block consists of one 32-bit word. When a given program is executed, the processor reads data from the following sequence of hex addresses:

200,204,208,20C,2F4,2F0,200,204,218,21C,24C,2F4

Show the contents of the cache and find the hit rate if the cache makes use of :

Direct mapping

Associative mapping

4-wayset-associative mapping