Unit - 4

Cache Memory

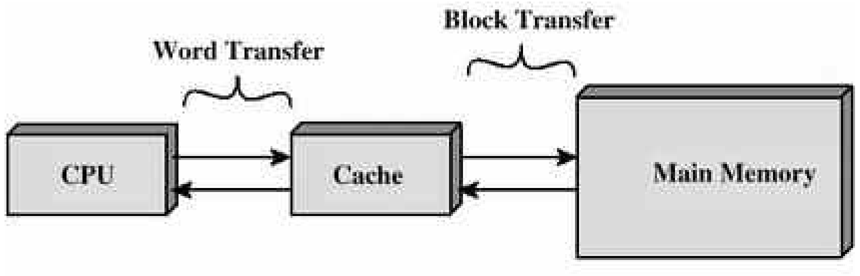
Cache Memory - Need Of Cache

The objective of cache memory is to reduce the access time.
Fundamental Idea :

- keep the frequently used data and instruction in the cache ,thus reducing the access time.
- When CPU needs to access memory, the cache is searched first for the data.
- If the word is found in the cache , it is accessed from the cache. But if the word is not in the cache , then access to main memory is done.
- And simultaneously word is copied in the cache.

Cache Memory - Need Of Cache

- The transfer of words from the main memory to cache is done in blocks.
- A block consists of many words.
- Since cache has small capacity , existing item has to be replaced if cache is being full. And the replacement is depends upon the algorithm.



Cache Memory - Need Of Cache

- If the active portions of the program and data are placed in a fast small memory, the average memory access time can be reduced.
- Thus reducing the total execution time of the program Such a fast small memory is referred to as cache memory
- The cache is the fastest component in the memory hierarchy and approaches the speed of CPU component

Cache Memory

- The main memory is divided into blocks,each consisting of fixed number of locations.
- While a location is accessed,the entire block is brought and stored into the cache memory.
- when the block is stored in the cache memory,it is referred to as line or block frame which is of the same size as the block in the main memory.
- hence any main memory block can be mapped to one of the cache lines.
- The mapping function decides that the block should be copied into which line number

Cache Hit and Cache Miss

- If data is in cache it is termed as "cache hit"
- If data is not present in the cache it is termed as "cache miss".
- The performance of the cache is measured in terms of hit ratio.
- Hit ratio is the ratio of the number of hits divided by the total number of references to the memory.
  Hit ratio = Numbers of hits / Total numbers of memory references

Cache Principle - Locality of reference

- The cache works on the principle of locality of reference.
- **What is locality of reference?**
- In computer science, locality of reference, also known as the principle of locality, is a term for the phenomenon in which the same values, or related storage locations, are frequently accessed.
- There are 2 types of Locality of reference
- 1. Spatial (Based on Space)
- 2. Temporal (Based on Time)
- If a word is accessed now then the word adjacent to it will be accessed next
- Keeping more words in a block affects spatial locality
- If a word is referenced now then the same word will be referenced again in future
- LRU (Least Recently Used) is used in temporal locality

Levels of Cache

- Level – 1 – Fastest , comes within processor ,Size from 8 kb to 64 kb
- Internal cache or Primary cache
- Level – 2 – Learger but slower in speed compared to L1
- Stores recently accessed information
- Known as secondary cache , comes between L1 and RAM
- Size 64 kb to 4 Mb

Levels of Cache

- Level – 3 enhanced form of memory present on motherboard
- Used to feed up L2 , Faster than system’s main memory but slower then L2.
- Size more then 3 MB
- Used to enhance the performance of L1 and L2

Replacement Algorithm

- Random Choice Algorithm:
- In this algorithm any block frame from the cache is selected randomly and deleted without any relation to previous page.
- First-in-First-out Algorithm :
- This algorithm selects the block frame which has been in the cache memory for a long time , i.e. the first block which entered the cache is one which has to be deleted.

Replacement Algorithm

- Least Frequently Used Algorithm :
- This algorithm chooses the block frame that has been used very frequently by the CPU. The counter value of each block frame gives the details of least frequently used block.
- Least Recently Used Algorithm :
- This algorithm chooses the block frame that has been referenced by the CPU very less number of times from the time it was mapped onto the cache memory. The counter value gives the details of how many times the block has been referenced by the CPU.

Cache Coherence - Problem

- Cache coherence is a concern in a multi core environment because of distributed L1 and L2 caches. Since each core has its own cache, the copy of the data in that cache may not always be the most up-to-date version.
- Incorrect execution could occur if two or more copies of a given cache block exist, in two processors' caches, and one of these blocks is modified. Figure shows the case where core 1 and core 2 both have a copy of variable "a" in their caches (steps 1 and 2). The variable was read from main memory (or an L2 cache) that both processors share, so both copies of the variable contain the same values. If core 1 or core 2 performs a read instruction to read a value in variable "a," correct execution occurs. However, if core 1 performs a store instruction (step 3) that modifies variable "a," and core 2 subsequently performs a load instruction from that variable, the read instruction must see the new value. Therefore, the new value must somehow be propagated to the copy of variable "a" in core 2. This is called the cache coherence problem.

Cache Coherence - Solution

- The cache coherence can be solved by two approaches : a. Software and hardware or b. Only Hardware
- In software approach , the compiler marks the data that are not valid or non cacheable. Only non shared and read only data are stored in caches.
- The hardware and operating system prevents non cacheable data to be cached.
- This scheme restricts the data stored in the caches and degrades performance by introducing an extra software overhead.
- In hardware approach, all cached attached to the bus , monitor the network consistently for any possible write operation.
- Depending on the method used , they either update or invalidate the data in their private cache when a match is detected.

Cache Coherence - Solution

- The hardware unit used for this scheme is known as snoopy cache controller.
- Snoopy cache controller designed to keep watch on the bus for any write operation in the cache.
- In snoopy cache controller the simplest method is to adopt write through policy.
- All the controllers watch the bus for any memory store operation.
- When a word in a cache is updated, the corresponding location in the main memory is also updated.
- The snoopy cache controllers check their memory to determine if they have a copy of the word that is updated. If a copy exists , that location is marked invalid.
- If the processor accesses the invalid word from the cache,it is equivalent to a cache miss and the updated word is transferred from the main memory

Write-Through

- In write-through, data is simultaneously updated to cache and memory. This process is simpler and more reliable.
- The advantage of the write-through cache is that the main memory always contains the same data as the cache contains.
- This characteristic is desirable in a system which uses direct memory access scheme of data transfer.
- The I/O devices communicating through DMA receive the most recent data.

Write-Back

- The data is updated only in the cache and updated into the memory at a later time. Data is updated in the memory only when the cache line is ready to be replaced
- The main memory gets updated only when the corresponding word is to be deleted from the cache memory.
- The updated locations in the cache memory are marked by a flag so that later on, when the word is removed from the cache, it is copied into the main memory.
- The words are removed from the cache time to time to make room for a new block of words.

Memory Mapping

The transformation of data from the main memory to cache memory is known as mapping.

There are three types of mapping

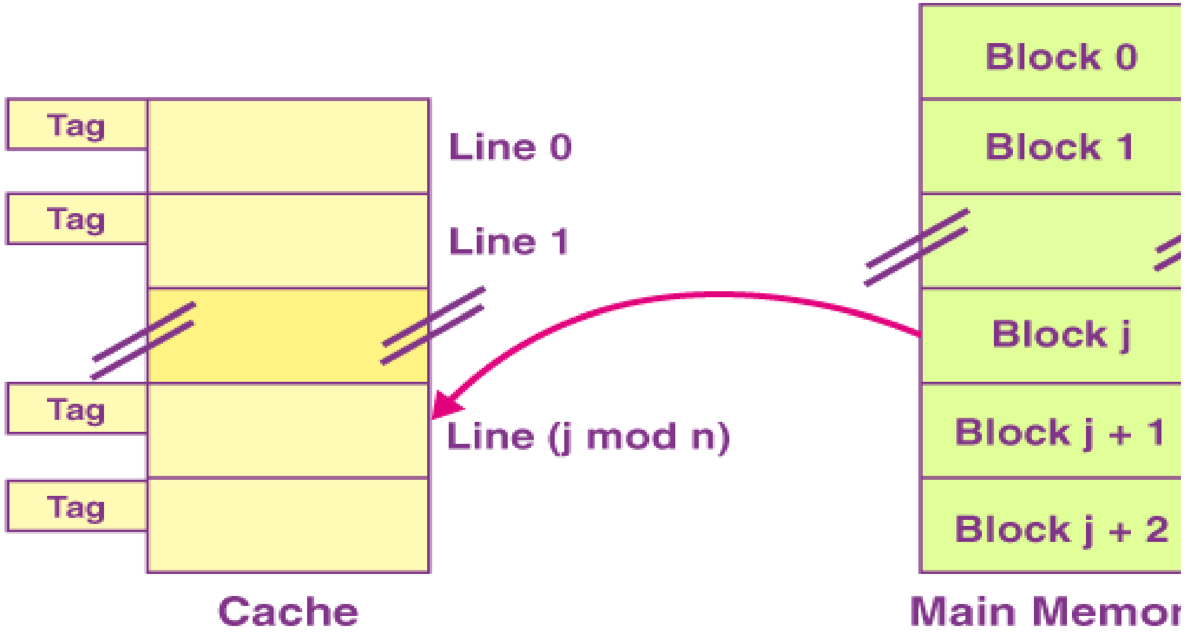
- Direct Mapping
- Associative Mapping
- Set-associative Mapping

Direct Mapping

- The simplest technique, known as direct mapping, maps each block of main memory into only one possible cache line. or In Direct mapping, assign each memory block to a specific line in the cache. If a line is previously taken up by a memory block when a new block needs to be loaded, the old block is trashed.
- Direct mapping`s performance is directly proportional to the Hit ratio.
- To assign any block from main memory to cache memory , the formula is K mod n

Where K = Block number and n = number of lines

Direct Mapping



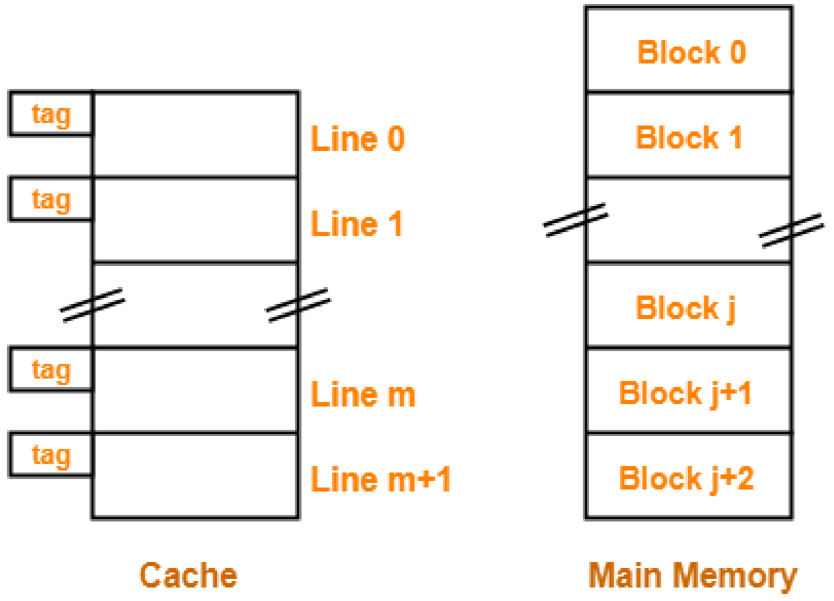
Direct Mapping

Associative Mapping

- Any block can go anywhere in cache i.e. Any empty block in cache can be used.
- When request comes for a block , all the entries are compared simultaneously to determine if the requested block is present or not.
- The mapping flexibility permits wide variety of replacement algorithms.
- It encounters longer access time because of associative search, it check all tags to check for a hit.
- It is more flexible and expensive than direct mapping.

Associative Mapping

- Cache address contains only offset and tag.
- There is no need to add line number as line no has no significance , because any block can store in any line



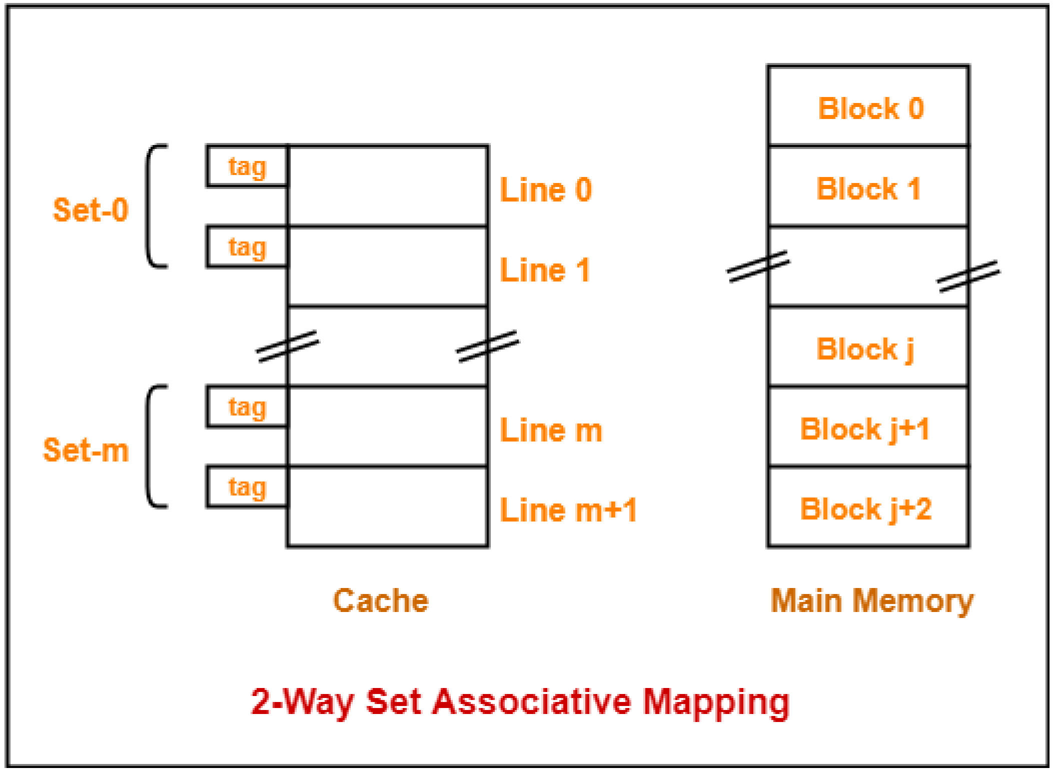
Set Associative Mapping

- Combines the simplicity of direct mapping with the flexibility of associative mapping.
- Set associative is an improvement over the direct mapping as here the cache is divided into sets.
- When an address is mapped to a set, the direct mapping scheme is used, and then associative mapping is used within a set.
- When miss occurs in set associative cache and set is full , one of the blocks has to be replaced according to replacement algorithm .

Set Associative Mapping

- Set can be any value in the power of 2 like 2 way set , 4 way 16 way etc.
- No. of set = no. of lines / value of k
- K = k way set

- Cache address will be generate
- Using tag , set and block offset


Set-0
tag
tag
Line 0
Line 1
Set-m
tag
tag
Line m
Line m+1
Block 0
Block 1
Block j
Block j+1
Block j+2
Cache
Main Memory
2-Way Set Associative Mapping