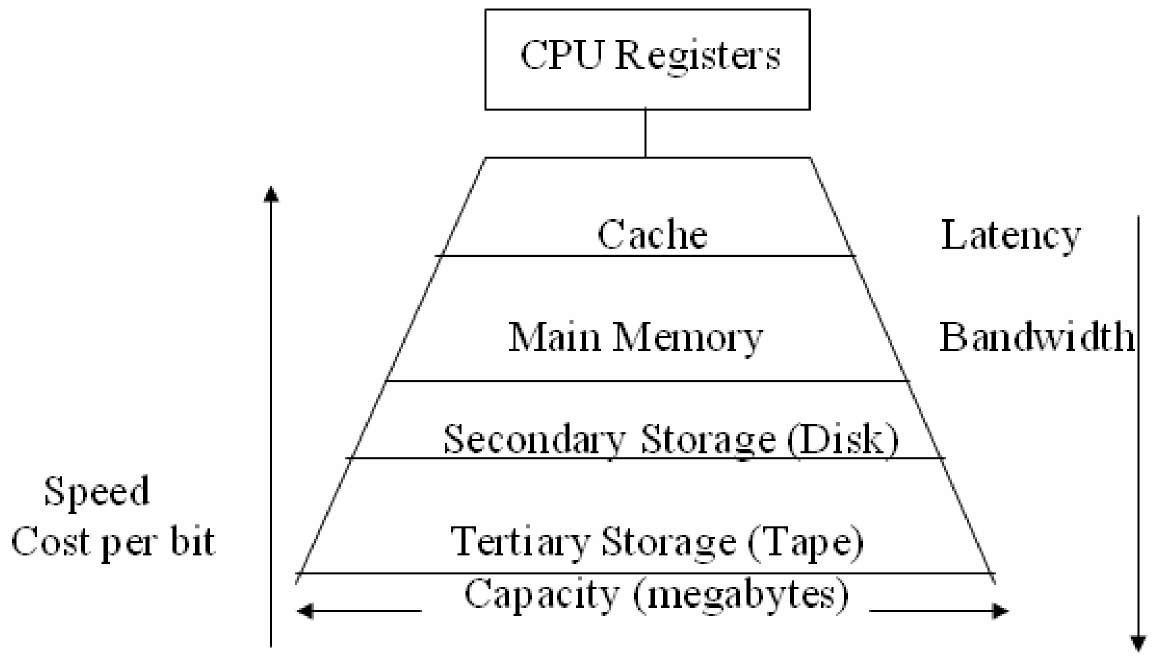
Chapter 6

Memory System Design I

6.1 Basic Concepts

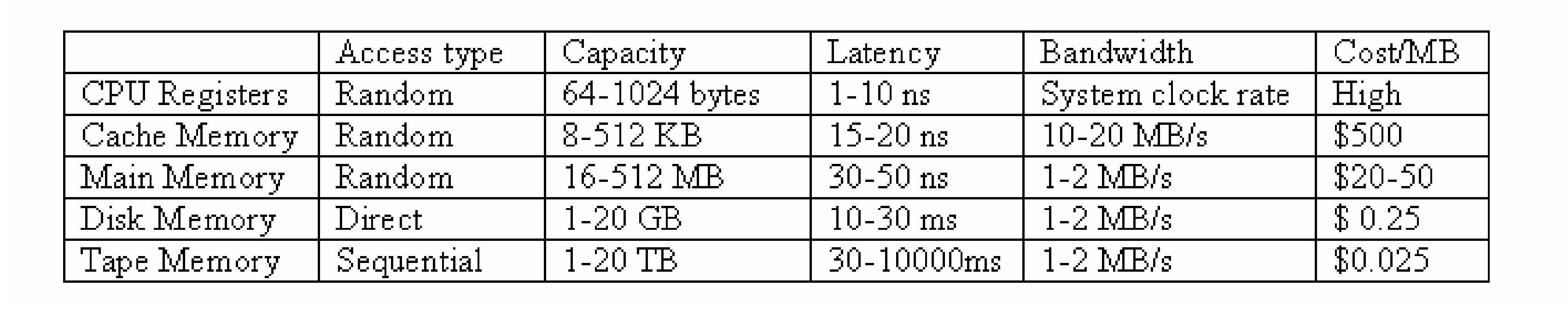
- Memory Hierarchy



A typical memory hierarchy

6.1 Basic Concepts

- Memory Hierarchy



	Access type	Capacity	Latency	Bandwidth	Cost/MB
CPU Registers	Random	64-1024 bytes	1-10 ns	System clock rate	High
Cache Memory	Random	8-512 KB	15-20 ns	10-20 MB/s	$500
Main Memory	Random	16-512 MB	30-50 ns	1-2 MB/s	$20-50
Disk Memory	Direct	1-20 GB	10-30 ms	1-2 MB/s	$ 0.25
Tape Memory	Sequential	1-20 TB	30-10000ms	1-2 MB/s	$0.025

Memory hierarch parameters

6.1 Basic Concepts

- Memory Hierarchy
  - The memory hierarchy can be characterized by a number of parameters.
  - Among these parameters are:
    - access type,
    - capacity,
    - cycle time,
    - latency,
    - bandwidth, and
    - cost.

6.1 Basic Concepts

- Memory Hierarchy
  - The effectiveness of a memory hierarchy depends on the principle of moving information into the fast memory infrequently and accessing it many times before replacing it with new information.
  - This principle is possible due to a phenomenon called *locality of reference*, i.e. within a given period of time, programs tend to reference relatively a confined area of memory repeatedly.

6.1 Basic Concepts

- Memory Hierarchy
  - There exist two forms of locality:
    - Spatial
      - *Spatial locality* refers to the phenomenon that when a given address has been referenced, it is most likely that addresses near it will be referenced within a short period of time, e.g., consecutive instructions in a straight line program.
    - Temporal
      - *Temporal locality*, on the other hand, refers to the phenomenon that once a particular memory item has been referenced, it is most likely that it will be referenced next, e.g., an instruction in a program loop.

6.1 Basic Concepts

- Memory Hierarchy
  - The sequence of events that takes place when the processor makes a request for an item is as follows:
    - First, the item is sought in the first memory level of the memory hierarchy.
      - The probability of finding the requested item in the first level is called the *hit ratio,* h_1
      - The probability of not finding (missing) the requested item in the first level of the memory hierarchy is called the *miss ratio,* $(1- h_1)$.
    - When the requested item causes a “*miss”,* it is sought in the next subsequent memory level.
      - The probability of finding the requested item in the second memory level, the hit ratio of the second level, is h_2
      - The miss ratio of the second memory level is $(1- h_2)$
    - The process is repeated until the item is found. Upon finding the requested item, it is brought and sent to the processor.

6.2 Cache Memory

- Impact of Temporal Locality
  - In this case, it is assumed that instructions in program loops, which are executed many times, e.g. *n* times, once loaded into the cache, are used more than once before they are replaced by new instructions. The average access time, t_{av}, is given by:

$$t_{av} = \frac{nt_c + t_m}{n} = t_c + \frac{t_m}{n}$$

  - The above expression reveals that as the number of repeated access, *n,* increases, the average access time decreases, a desirable feature of the memory hierarchy.

6.2 Cache Memory

- Impact of Spatial Locality
  - In this case, it is assumed that the size of the block transferred from the main memory to the cache, upon a cache miss, is *m* elements. It is also assumed that due to spatial locality, all *m* elements were requested, one at a time, by the processor. Based on these assumptions, the average access time, t_{av}, is given by:

$$t_{av} = \frac{mt_c + t_m}{m} = t_c + \frac{t_m}{m}$$

  - The above expression reveals that as the number of elements in a block, *m,* increases, the average access time decreases, a desirable feature of the memory hierarchy.

6.2 Cache Memory

- Impact of Combined Temporal and Spatial Locality
  - In this case, it is assumed that the element requested by the processor created a cache miss leading to the transfer of a block, consisting of *m elements,* to the cache (that take t_m).
  - Due to spatial locality, all *m* elements constituting a block were requested, one at a time, by the processor (requiring mt_c).
  - Following that, the originally requested element was accessed *(n-1)* times (temporal locality), i.e., a total of *n* times access to that element.
  - The average access time, t_{av} , is given by the following:

$$t_{av} = \frac{\frac{mt_c + t_m}{m} + (n-1)t_c}{n} = \frac{t_c + \frac{t_m}{m} + (n-1)t_c}{n} = \frac{t_m}{nm} + t_c$$

6.2 Cache Memory

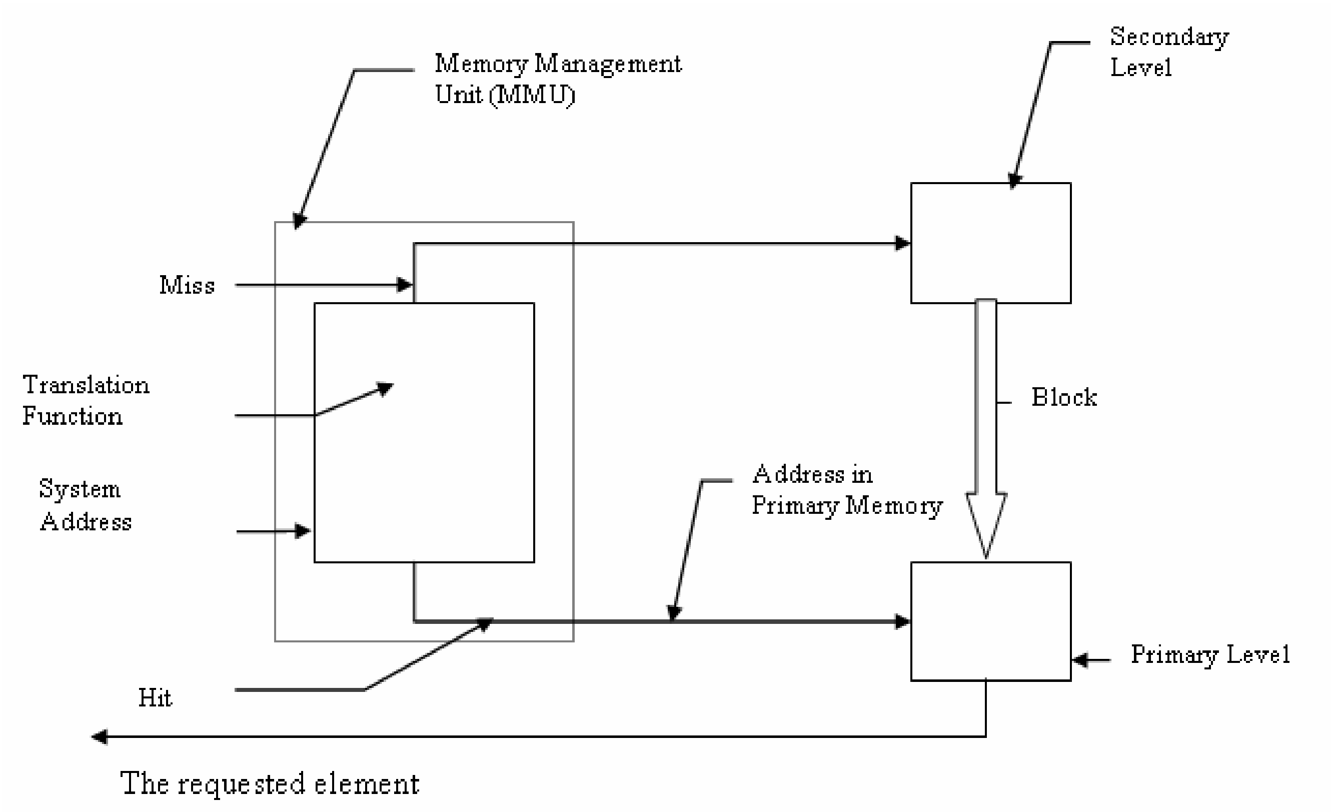
- Cache-Mapping Function
  - Cache-mapping function is presented taking into consideration the interface between two successive levels in the memory hierarchy:
    - Primary level and Secondary level.
  - If the focus is on the interface between the cache and main memory, then the cache represents the primary level, while the main memory represents the secondary level.

6.2 Cache Memory

- Cache-Mapping Function
  - A request for accessing a memory element is made by the processor through issuing the address of the requested element.
  - The address issued by the processor may correspond to that of an element that exits currently in the cache (cache hit); otherwise, it may correspond to an element that is currently residing in the main memory.
  - Therefore, address translation has to be made in order to determine the whereabouts of the requested element.
  - This is one of the functions performed by the Memory Management Unit (MMU).

6.2 Cache Memory

- Cache-Mapping Function



6.2 Cache Memory

- Cache Memory Organization
  - There is basically three main different organization techniques used for cache memory:
    - Direct mapping
    - Fully associative mapping
    - Set-associative mapping
  - These techniques differ in two main aspects:
    - The criterion used to place, in the cache, an incoming block from the main memory.
    - The criterion used to replace a cache block by an incoming block (on cache full).

6.2 Cache Memory

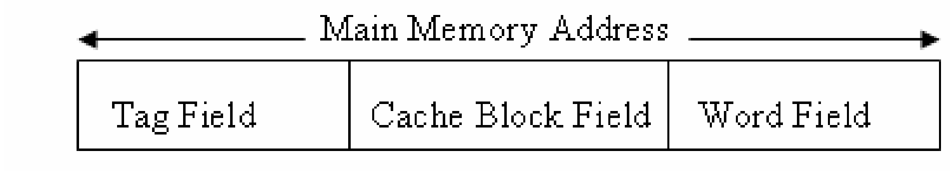
- Cache Memory Organization - Direct mapping
  - This is the simplest among the three techniques.
  - Its simplicity stems from the fact that it places an incoming main memory block into a specific fixed cache block location.
  - The placement is done based on a fixed relation between the incoming block number, *i,* the cache block number, *j,* and the number of cache blocks, *N,* as follows $j = i \bmod N$.

6.2 Cache Memory

- Cache Memory Organization - Direct mapping
  - The main advantage of the direct-mapping technique is its simplicity in determining the whereabouts to place an incoming main memory block in the cache.
  - Its main disadvantage is the inefficient use of the cache.
    - This is because according to this technique, a number of main memory blocks may compete for a given cache block even if there exists other empty cache block(s). This disadvantage should lead to achieving low cache hit ratio.

6.2 Cache Memory

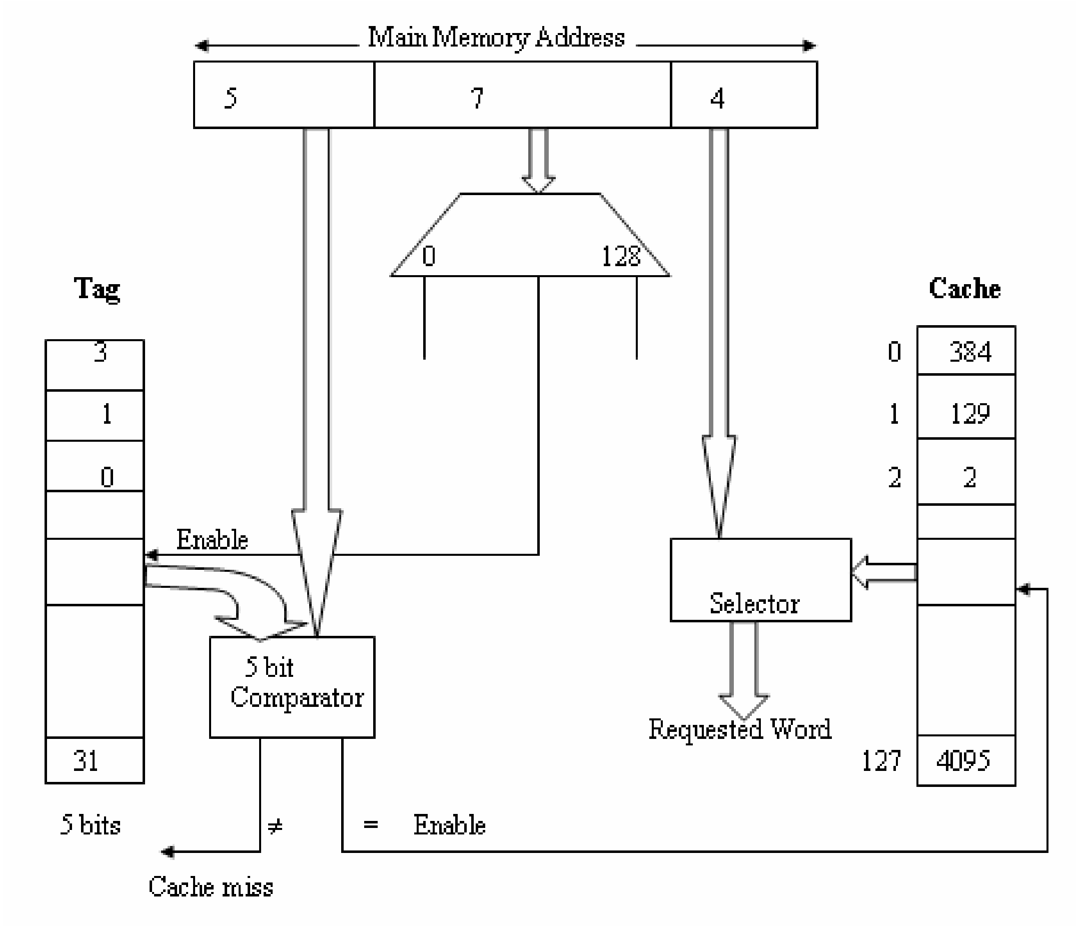
- Cache Memory Organization - Direct mapping
  - According to the direct-mapping technique, the memory management unit (MMU) interprets the address issued by the processor by dividing the address into three fields :



  - The length, in bits, of each of the above fields is given below:
    - (1) Word Field = $\log_2 B$, where *B* is the size of block in words.
    - (2) Block Field = $\log_2 N$, where *N* is the size of the cache in blocks.
    - (3) Tag Field = $\log_2 \frac{M}{N}$, where *M* is the size of the main memory in blocks.
    - (4) The number of bits in the main memory address = $\log_2 (B \times M)$

6.2 Cache Memory

- Cache Memory Organization - Direct mapping

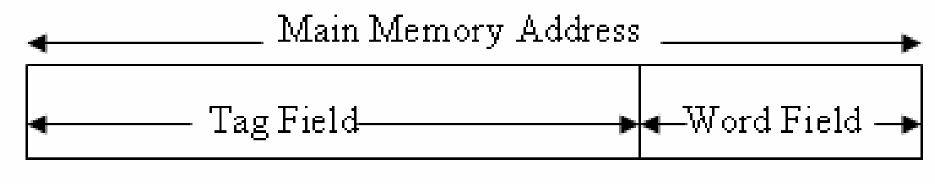


6.2 Cache Memory

- Cache Memory Organization – Fully Associative mapping
  - An incoming main memory block can be placed in any available cache block.
  - Therefore, the address issued by the processor need only to have two fields.
    - The *Tag:* uniquely identifies the block while residing in the cache.
    - The *Word:* identifies the element within the block that is requested by the processor.

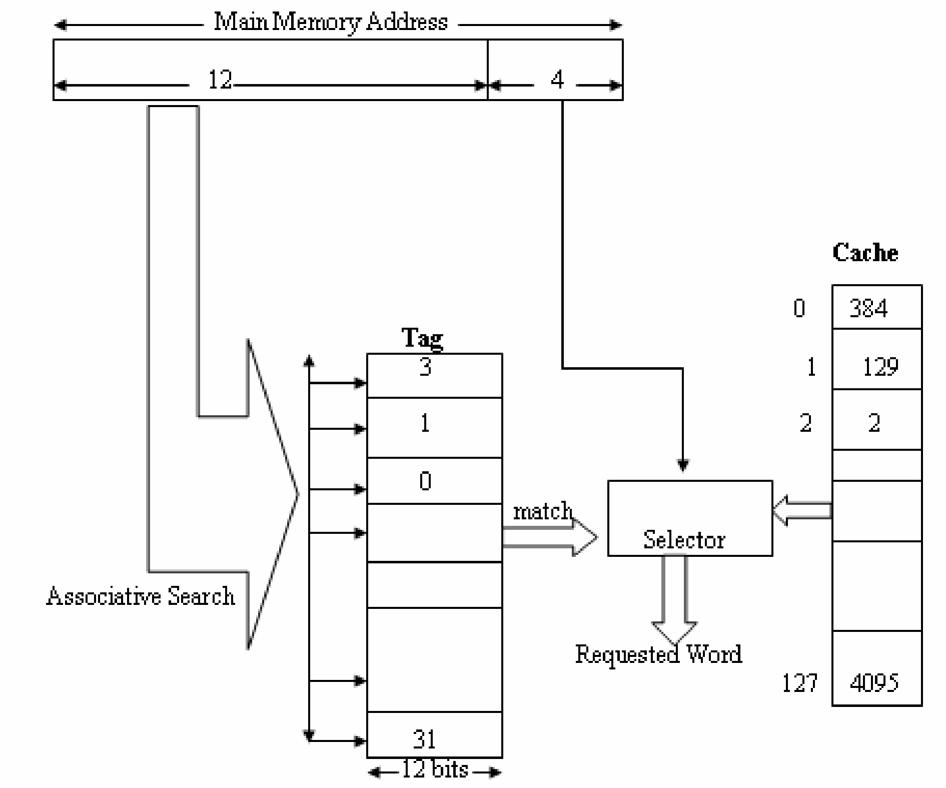
6.2 Cache Memory

- Cache Memory Organization – Fully Associative mapping
  - The memory management unit (MMU) interprets the address issued by the processor by dividing it into two fields.
  - The length, in bits, of each of the fields is given below:
    - (1) Word Field = $\log_2 B$, where *B* is the size of the block in words.
    - (2) Tag Field = $\log_2 M$, where *M* is the size of the main memory in blocks.
    - (3) The number of bits in the main memory address = $\log_2 (B \times M)$



6.2 Cache Memory

- Cache Memory Organization – Fully Associative mapping

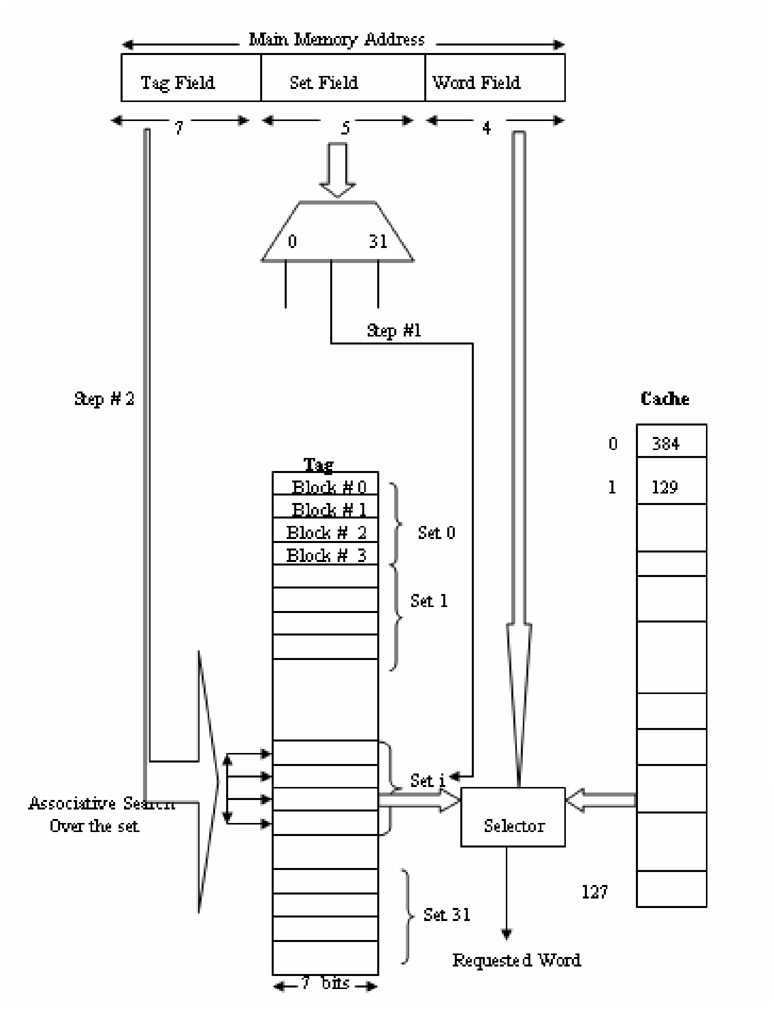


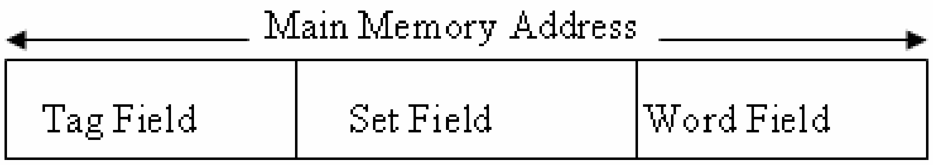
6.2 Cache Memory

- Cache Memory Organization – Set-Associative Mapping
  - The cache is divided into a number of sets.
  - Each set consists of a number of blocks.
  - A given main memory block maps to a specific cache set based on the equation $s = i \bmod S$, where S is the number of sets in the cache, i is the main memory block number, and s is the specific cache set to which block i maps.
  - However, an incoming block maps to any block in the assigned cache set.
  - Therefore, the address issued by the processor is divided into three distinct fields.
    - *Tag:* uniquely identifies the targeted block within the determined set.
    - *Set:* used to uniquely identify the specific cache set that ideally should hold the targeted block.
    - The *Word:* identifies the element (word) within the block that is requested by the processor.

6.2 Cache Memory

- Cache Memory Organization – Set-Associative Mapping





6.2 Cache Memory

- Qualitative comparison among cache mapping techniques

The mapping technique	Simplicity	Associative tag search	Expected cache utilization	Replacement technique
Direct	Yes	None	Low	Not needed
Associative	No	Involved	High	Yes
Set-associative	Moderate	Moderate	Moderate	Yes

6.2 Cache Memory

- Replacement Techniques
  - Random Selection: includes a randomly selected block.
  - First-in-First-out, FIFO: the block that has been in the cache the longest.
  - Least Recently Used, LRU*:* the block that has been used the least while residing in the cache.

6.2 Cache Memory

- Cache Write Policies
  - Coherence between a cache word and its copy in the main memory should be maintained at all times, if at all possible.
  - A number of policies (techniques) are used in performing write operations to the main memory blocks while residing in the cache.
  - These policies determine the degree of coherence that can be maintained between cache words and their counterparts in the main memory.

6.2 Cache Memory

- Cache Write Policies Upon a Cache Hit
  - There are basically two possible write policies upon a cache hit:
    - Write-through
      - every write operation to the cache is repeated to the main memory at the same time.
      - The write-through policy maintains coherence between the cache blocks and their counterparts in the main memory at the expense of the extra time needed to write to the main memory. This leads to an increase in the average access time.
    - Write-back
      - all writes are made to the cache.
      - A write to the main memory is postponed until a replacement is needed.
      - Every cache block is assigned a bit, called the *dirty bit,* to indicate that at least one write operation has been made to the block while residing in the cache.
      - At replacement time, the dirty bit is checked, if it is set, then the block is written back to the main memory; otherwise, it is simply overwritten by the incoming block.
      - Coherence is only guaranteed at the time of replacement.

6.2 Cache Memory

- Cache Write Policy Upon a Cache Miss
  - Two main schemes can be used:
    - Write-allocate whereby the main memory block is brought to the cache and then updated.
    - Write-no-allocate whereby the missed main memory block is updated, while in the main memory and not brought to the cache.
  - In general, write-through caches use write-no-allocate policy while write-back caches use write-allocate policy.

6.2 Cache Memory

- Cache Read Policy Upon a Cache Miss
  - Two possible strategies can be used.
    - In the first one, the main memory missed block is brought to the cache while the required word is forwarded immediately to the CPU as soon as it is available.
    - In the second strategy, the missed main memory block is entirely stored in the cache and the required word is then forwarded to the CPU.

6.2 Cache Memory

- Cache Write Through Policy
  - Write-allocate
    - The average access time for a memory system is given by:

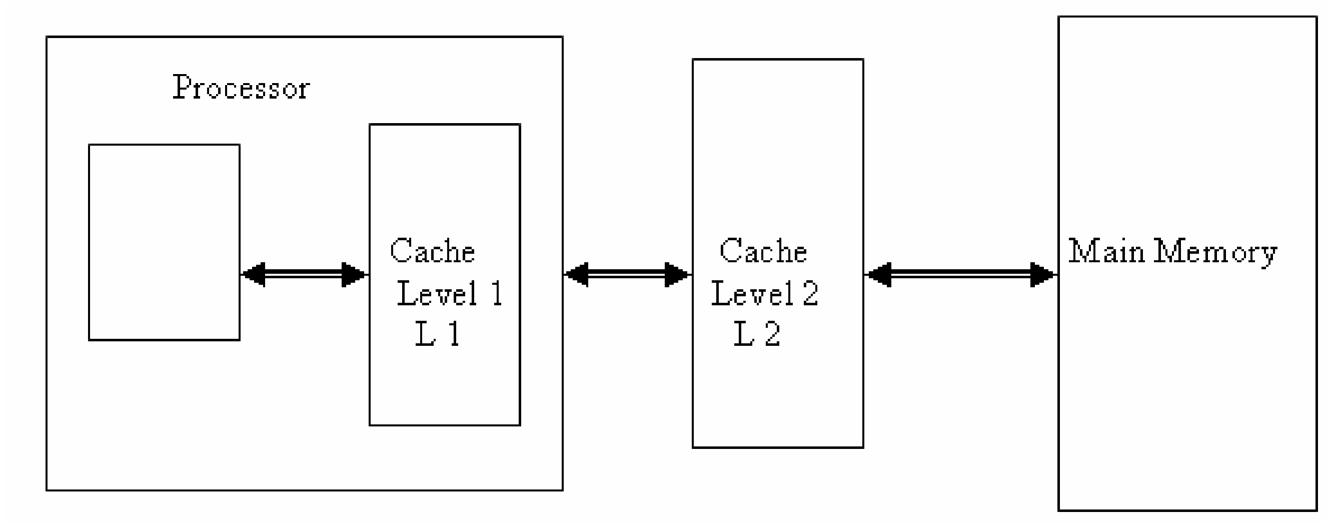
$$t_a = t_c + (1 - h)\, t_b + w\,(t_m - t_c)$$

    - t_b is the time required to transfer a block to the cache.
    - $(t_m - t_c)$ is the additional time incurred due to the write operations.
    - *w* is the fraction of write operations.
  - Write-no-allocate
    - The average access time for a memory system is given by:

$$t_a = t_c + (1 - w)(1 - h)\, t_b + w\,(t_m - t_c)$$

6.2 Cache Memory

- Real-Life Cache Organization Analysis
  - Intel's Pentium IV Processor Cache



6.2 Cache Memory

- Real-Life Cache Organization Analysis
  - PowerPC 604 Processor Cache
    - The PowerPC cache is divided into data and instruction caches, called *Harvard Organization*.
    - Both the instruction and the data caches are organized as 16-Kbyte four-way set associative.

Cache organization	Set-Associative
Block size	32 bytes
Main Memory size	4 GB (M = 128 Mega blocks)
Cache size	16 KB (N = 512 blocks)
Number of blocks per set	Four
Number of cache sets (S)	128 sets

6.2 Cache Memory

- Real-Life Cache Organization Analysis
  - PMC-Sierra RM7000A 64-bit MIPS RISC Processor
    - The RM7000 uses a different cache organization compared to the Intel's and the PowerPC. In this case, three separate caches are included. These are:
      - Primary Instruction cache: A 16 Kbytes, 4-way set associative cache with 32-byte block size (eight instructions).
      - Primary Data cache: A 16 Kbytes, 4-way set associative cache with 32 bytes block size (eight words).
      - Secondary Cache: A 256 Kbytes, 4-way set associative cache for both instructions and data.
      - In addition to the three on-chip caches, the RM7000 provides a dedicated tertiary cache interface, which supports tertiary cache sizes of 512 Kbytes, 2 Mbytes, and 8 Mbytes. This tertiary cache is only accessed after a secondary cache miss.

6.3 Summary

- In this chapter, the design and analysis of the first level of a memory hierarchy, i.e. the cache memory was considered.
- In this context, the locality issues were discussed and their effect on the average access time was explained.
- Three cache mapping techniques were analyzed and their performance measures were compared:
  - direct
  - associative, and
  - set-associative mappings
- Three replacement techniques were also introduced:
  - Random,
  - FIFO, and
  - LRU replacement.

6.3 Summary

- The impact of the three techniques on the cache hit ratio was analyzed.
- Cache writing policies were also introduced and analyzed.
- The discussion on cache ended up with a presentation of the cache memory organization and characteristics of three real-life examples:
  - Pentium-IV,
  - PowerPc, and
  - PMC-Sierra-RM-7000, processors.