Computer Organization and Architecture

Chapter 04

CACHE MEMORY

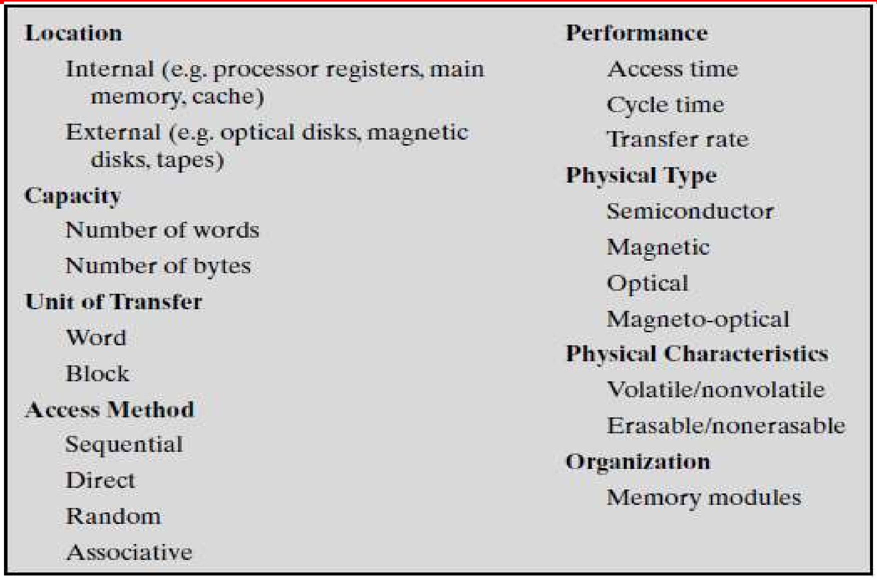
KEY POINTS

- Computer memory is organized into a hierarchy. At the highest level (closest to the processor) are the processor registers. Next comes one or more levels of cache. When multiple levels are used, they are denoted L1, L2, and so on.
- The hierarchy continues with external memory, with the next level typically being a fixed hard disk, and one or more levels below that consisting of removable media such as optical disks and tape.
- As one goes down the memory hierarchy, one finds decreasing cost/bit, increasing capacity, and slower access time.
- If the cache is designed properly, then most of the time the processor will request memory words that are already in the cache.

CACHE MEMORY

4.1. Computer Memory System Overview

4.1.1. Characteristics of Memory Systems



Location
- Internal (e.g. processor registers, main memory, cache)
- External (e.g. optical disks, magnetic disks, tapes)

Capacity
- Number of words
- Number of bytes

Unit of Transfer
- Word
- Block

Access Method
- Sequential
- Direct
- Random
- Associative

Performance
- Access time
- Cycle time
- Transfer rate

Physical Type
- Semiconductor
- Magnetic
- Optical
- Magneto-optical

Physical Characteristics
- Volatile/nonvolatile
- Erasable/nonerasable

Organization
- Memory modules

Figure 4.1. Key Characteristics of Computer Memory Systems

4.1.1. Characteristics of Memory Systems

- An obvious characteristic of memory is its **capacity.** For internal memory, this is typically expressed in terms of bytes (1 byte 8 bits) or words.
- Common word lengths are 8, 16, and 32 bits. External memory capacity is typically expressed in terms of bytes.
- A related concept is the **unit of transfer.** For internal memory, the unit of transfer is equal to the number of electrical lines into and out of the memory module.

4.1.1. Characteristics of Memory Systems

- The **unit of transfer** may be equal to the word length, but is often larger, such as 64, 128, or 256 bits.
- Consider three related concepts for internal memory:
  - **Word**: The "natural" unit of organization of memory. The size of the word is typically equal to the number of bits used to represent an integer and to the instruction length.
  - **Addressable units:** In some systems, the addressable unit is the word. However, many systems allow addressing at the byte level. In any case, the relationship between the length in bits A of an address and the number N of addressable units is $2^A = N$.

4.1.1. Characteristics of Memory Systems

- **Unit of transfer:** For main memory, this is the number of bits read out of or written into memory at a time.

- Another distinction among memory types is the **method of accessing units of** data.These include the following:
  - **Sequential access:**
    - ✓Memory is organized into units of data, called records.
    - ✓Access must be made in a specific linear sequence.
    - ✓Access time depends on location of data and previous location
    - ✓e.g. tape

4.1.1. Characteristics of Memory Systems

- **Direct access:**
  - ✓Individual blocks have unique address
  - ✓Access is by jumping to vicinity plus sequential search
  - ✓Access time depends on location and previous location
  - ✓e.g. Disk
- **Random access:**
  - ✓Individual addresses identify locations exactly
  - ✓Access time is independent of location or previous access
  - ✓e.g. RAM

4.1.1. Characteristics of Memory Systems

- **Associative:**
  - ✓Data is located by a comparison with contents of a portion of the store
  - ✓Access time is independent of location or previous access
  - ✓e.g. cache
- From a user's point of view, the two most important characteristics of memory are **capacity** and **performance.**
- Three performance parameters are used:
  - **Access time:**
    - ✓ Time between presenting the address and getting the valid data

4.1.1. Characteristics of Memory Systems

- **Memory Cycle time:**
  - ✓Time may be required for the memory to "recover" before next access
  - ✓Cycle time is access + recovery
- **Transfer rate:**
  - ✓This is the rate at which data can be transferred into or out of a memory unit.

4.1.2. The Memory Hierarchy

- The design constraints on a computer's memory can be summed up by three questions: How much? How fast? How expensive?
- The way out of this dilemma is not to rely on a single memory component or technology, but to employ a memory hierarchy.
- A typical hierarchy is illustrated in Figure 4.1. As one goes down the hierarchy, the following occur:

4.1.2. The Memory Hierarchy

a. Decreasing cost per bit
b. Increasing capacity
c. Increasing access time
d. Decreasing frequency of access of the memory by the processor

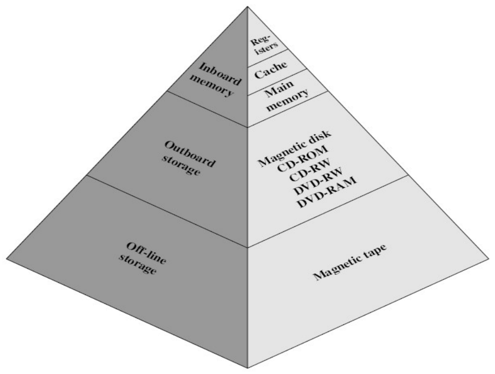


Figure 4.1 The Memory Hierarchy

4.2. Cache Memory Principles

- The concept is illustrated in Figure 4.3.
- There is a relatively large and slow main memory together with a smaller, faster cache memory.
- The cache contains a copy of portions of main memory. When the processor attempts to read a word of memory, a check is made to determine if the word is in the cache.
  - If so, the word is delivered to the processor.
  - If not, a block of main memory, consisting of some fixed number of words, is read into the cache and then the word is delivered to the processor

4.2. Cache Memory Principles

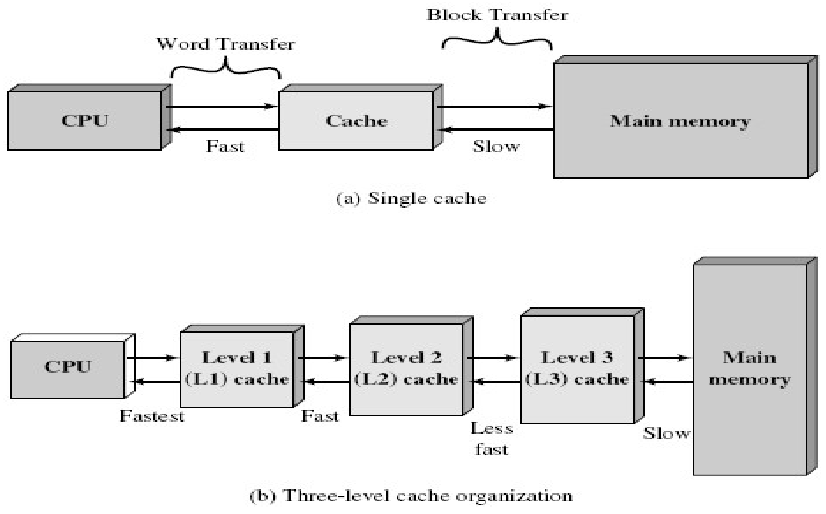


Figure 4.3. Cache and Main Memory

4.2. Cache Memory Principles

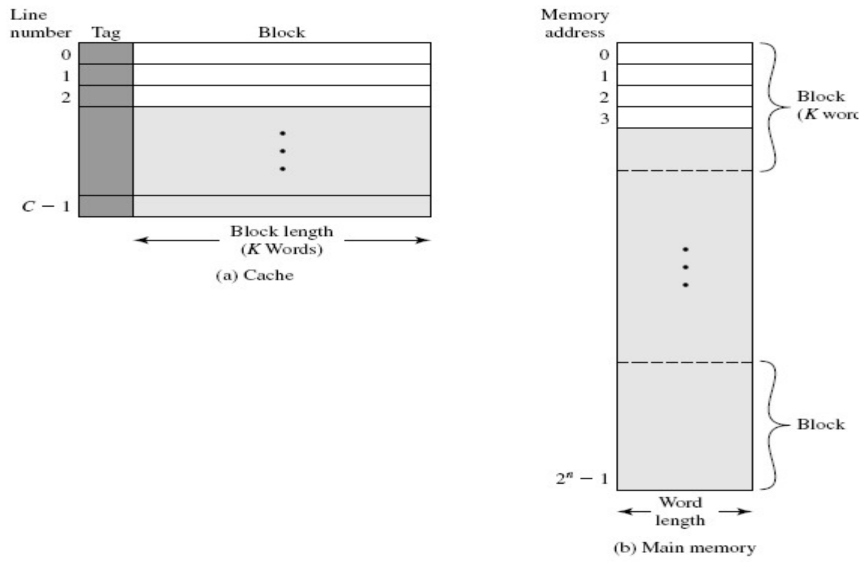


Figure 4.4 Cache/Main Memory Structure

4.2. Cache Memory Principles

- Main memory consists of up to **2^n** addressable words, with each word having a unique **n**-bit address.
- For mapping purposes, this memory is considered to consist of a number of fixed length blocks of **K** words each.
- That is, there are **$M=2^n/K$** blocks in main memory.
- The cache consists of **m** blocks, called lines. Each line contains **K** words, plus a tag of a few bits.

4.2. Cache Memory Principles

- Each line also includes control bits (not shown), such as a bit to indicate whether the line has been modified since being loaded into the cache.
  - In referring to the basic unit of the cache, the term *line is used, rather than the term block, for two reasons:*
    - ✓(1) to avoid confusion with a main memory block, which contains the same number of data words as a cache line;
    - ✓(2) because a cache line includes not only **K** words of data, just as a main memory block, but also include tag and control bits.

4.2. Cache Memory Principles

- The length of a line, not including tag and control bits, is the ***line size***.
- The number of lines is considerably less than the number of main memory blocks (**m < M**).
- At any time, some subset of the blocks of memory resides in lines in the cache.
- If a word in a block of memory is read, that block is transferred to one of the lines of the cache.
- Because there are more blocks than lines, an individual line cannot be uniquely and permanently dedicated to a particular block.
- Thus, each line includes a tag that identifies which particular block is currently being stored. The tag is usually a portion of the main memory address.

4.2. Cache Memory Principles

- Cache operation
  - CPU requests contents of memory location
  - Check cache for this data
  - If present, get from cache (fast)
  - If not present, read required block from main memory to cache
  - Then deliver from cache to CPU
  - Cache includes tags to identify which block of main memory is in each cache slot
- Figure 4.5 illustrates the read operation. The processor generates the read address (RA) of a word to be read.

4.2. Cache Memory Principles

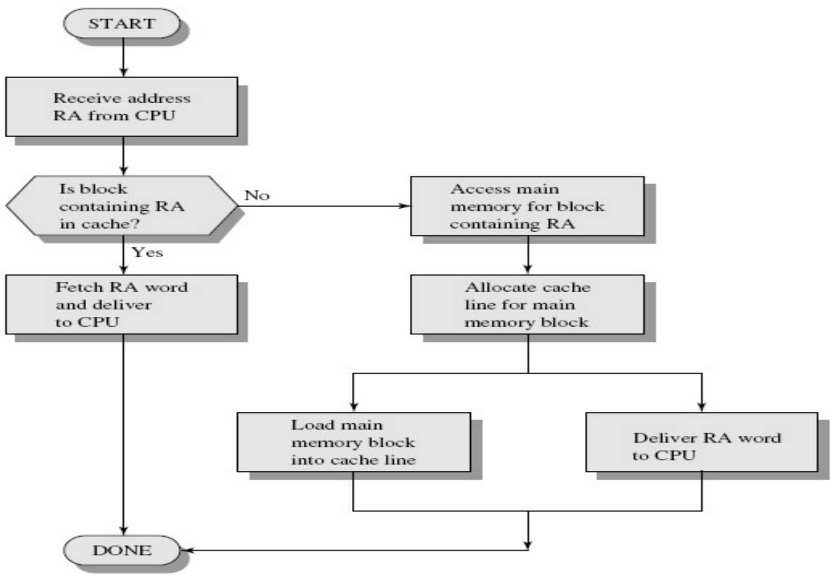

START
Receive address RA from CPU
Is block containing RA in cache?
No
Access main memory for block containing RA
Yes
Fetch RA word and deliver to CPU
Allocate cache line for main memory block
Load main memory block into cache line
Deliver RA word to CPU
DONE

Figure 4.5. Cache Read Operation

4.2. Cache Memory Principles

- Figure 4.6, which is typical of contemporary cache organizations.
  - The cache connects to the processor via data, control, and address lines.
  - The data and address lines also attach to data and address buffers, which attach to a system bus from which main memory is reached.
  - When a cache hit occurs, the data and address buffers are disabled and communication is only between processor and cache, with no system bus traffic.
  - When a cache miss occurs, the desired address is loaded onto the system bus and the data are returned through the data buffer to both the cache and the processor.

4.2. Cache Memory Principles

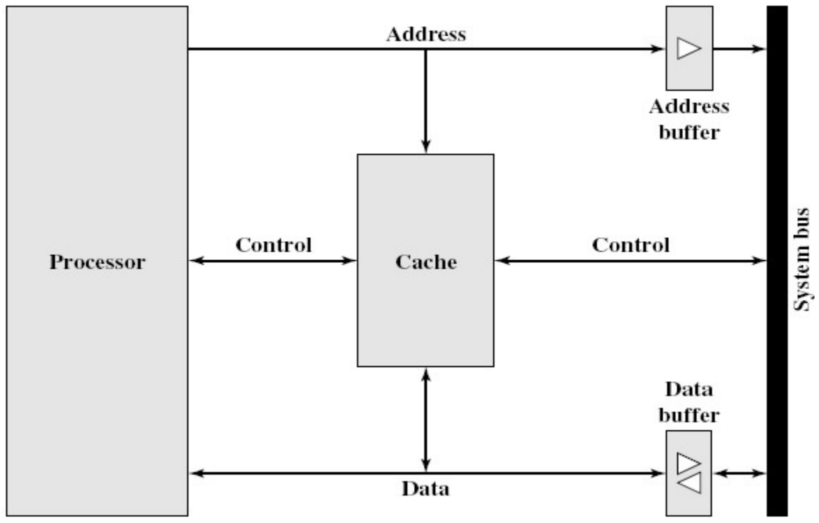
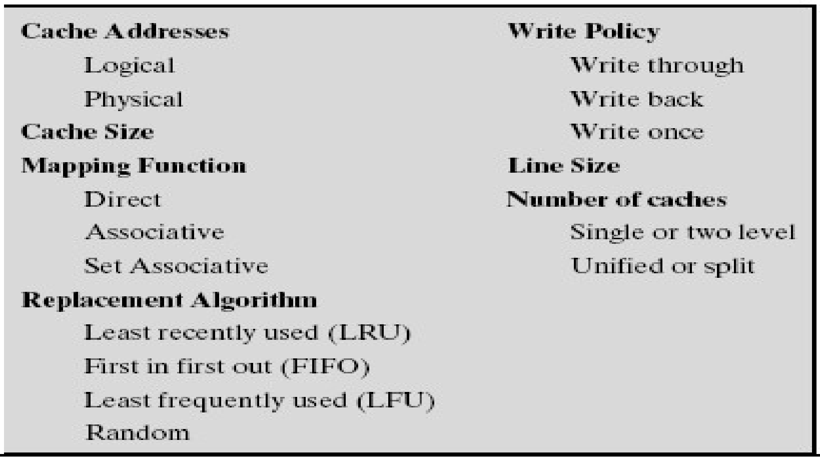


Figure 4.6. Typical Cache Organization

4.3. Elements of Cache Design

- Although there are a large number of cache implementations, there are a few basic design elements that serve to classify and differentiate cache architectures.



Cache Addresses	**Write Policy**
Logical	Write through
Physical	Write back
Cache Size	Write once
Mapping Function	**Line Size**
Direct	**Number of caches**
Associative	Single or two level
Set Associative	Unified or split
Replacement Algorithm	
Least recently used (LRU)	
First in first out (FIFO)	
Least frequently used (LFU)	
Random	

4.3.1. Cache Addresses

- Almost all nonembedded processors, and many embedded processors, support virtual memory.
- In essence, virtual memory is a facility that allows programs to address memory from a logical point of view, without regard to the amount of main memory physically available.
- When virtual memory is used, the address fields of machine instructions contain virtual addresses.
- For reads to and writes from main memory, a hardware memory management unit (MMU) translates each virtual address into a physical address in main memory.

4.3.1. Cache Addresses

- When virtual addresses are used, the system designer may choose to place the cache as Figure 4.7
  - Between the processor and the MMU
  - Or between the MMU and main memory.
- A logical cache, also known as a virtual cache, stores data using virtual addresses.
- The processor accesses the cache directly, without going through the MMU.
- A physical cache stores data using main memory physical addresses.

4.3.1. Cache Addresses

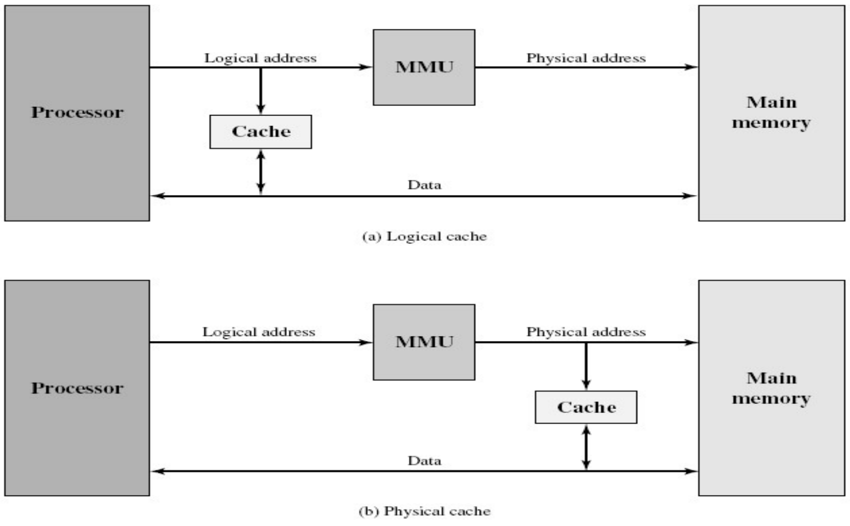
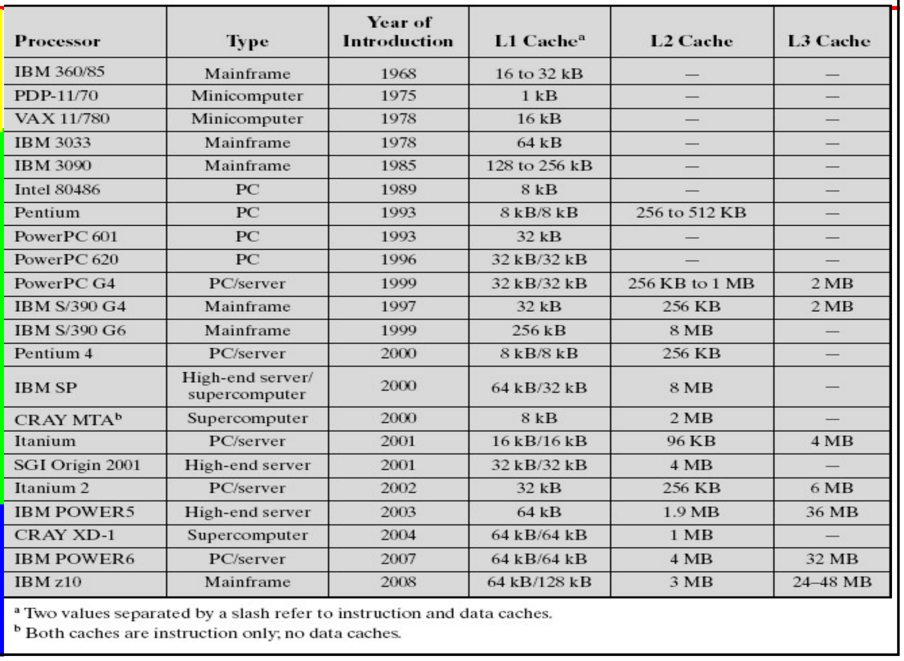


Figure 4.7. Logical and Physical Caches

4.3.1. Cache Addresses

- One obvious advantage of the logical cache is that cache access speed is faster than for a physical cache, because the cache can respond before the MMU performs an address translation.
- The disadvantage has to do with the fact that most virtual memory systems supply each application with the same virtual memory address space.
  - That is, each application sees a virtual memory that starts at address 0.
  - Thus, the same virtual address in two different applications refers to two different physical addresses.
  - The cache memory must therefore be completely flushed with each application context switch, or extra bits must be added to each line of the cache to identify which virtual address space this address refers to.

4.3.2. Cache Size



Processor	Type	Year of Introduction	L1 Cache[a]	L2 Cache	L3 Cache
IBM 360/85	Mainframe	1968	16 to 32 kB	—	—
PDP-11/70	Minicomputer	1975	1 kB	—	—
VAX 11/780	Minicomputer	1978	16 kB	—	—
IBM 3033	Mainframe	1978	64 kB	—	—
IBM 3090	Mainframe	1985	128 to 256 kB	—	—
Intel 80486	PC	1989	8 kB	—	—
Pentium	PC	1993	8 kB/8 kB	256 to 512 KB	—
PowerPC 601	PC	1993	32 kB	—	—
PowerPC 620	PC	1996	32 kB/32 kB	—	—
PowerPC G4	PC/server	1999	32 kB/32 kB	256 KB to 1 MB	2 MB
IBM S/390 G4	Mainframe	1997	32 kB	256 KB	2 MB
IBM S/390 G6	Mainframe	1999	256 kB	8 MB	—
Pentium 4	PC/server	2000	8 kB/8 kB	256 KB	—
IBM SP	High-end server/ supercomputer	2000	64 kB/32 kB	8 MB	—
CRAY MTA[b]	Supercomputer	2000	8 kB	2 MB	—
Itanium	PC/server	2001	16 kB/16 kB	96 KB	4 MB
SGI Origin 2001	High-end server	2001	32 kB/32 kB	4 MB	—
Itanium 2	PC/server	2002	32 kB	256 KB	6 MB
IBM POWER5	High-end server	2003	64 kB	1.9 MB	36 MB
CRAY XD-1	Supercomputer	2004	64 kB/64 kB	1 MB	—
IBM POWER6	PC/server	2007	64 kB/64 kB	4 MB	32 MB
IBM z10	Mainframe	2008	64 kB/128 kB	3 MB	24–48 MB

[a] Two values separated by a slash refer to instruction and data caches.
[b] Both caches are instruction only; no data caches.

4.3.3. Mapping Function

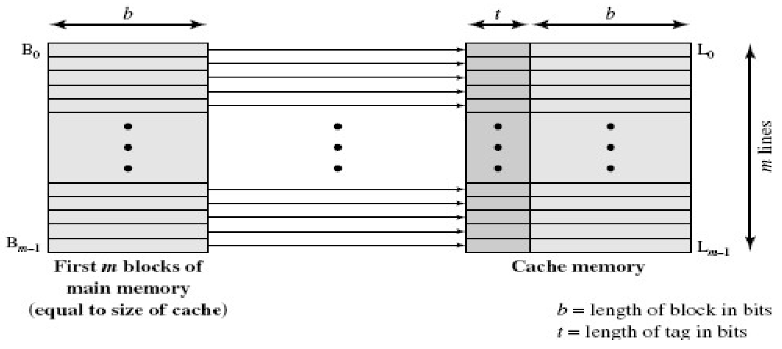
- Because there are fewer cache lines than main memory blocks, an algorithm is needed for mapping main memory blocks into cache lines.
- Further, a means is needed for determining which main memory block currently occupies a cache line.
- Three techniques can be used: direct, associative, and set associative.

4.3.3. Mapping Function

- For all three cases, the example includes the following elements:
  - The cache can hold 64 KBytes.
  - Data are transferred between main memory and the cache in blocks of 4 bytes each.
  - This means that the cache is organized as $16K=2^{14}$ lines of 4 bytes each.
  - The main memory consists of 16 Mbytes, with each byte directly addressable by a 24-bit address (2^{24} =16M).
  - Thus, for mapping purposes, we can consider main memory to consist of 4M blocks of 4 bytes each.

4.3.3. Mapping Function

- **DIRECT MAPPING** The simplest technique, known as direct mapping, maps each block of main memory into only one possible cache line.



b
t
b
B0
Bm–1
L0
Lm–1
m lines
First m blocks of main memory (equal to size of cache)
Cache memory
b = length of block in bits
t = length of tag in bits

Figure 4.8a. Direct mapping

4.3.3. Mapping Function

- The mapping is expressed as

  i = j modulo m

  - where
    - ✓i= cache line number
    - ✓J= main memory block number
    - ✓m= number of lines in the cache
- Figure 4.8a shows the mapping for the first **m** blocks of main memory.
- Each block of main memory maps into one unique line of the cache.The next **m** blocks of main memory map into the cache in the same fashion; that is, block $\mathbf{B_m}$ of main memory maps into line $\mathbf{L_0}$ of cache, block $\mathbf{B_{m+1}}$ maps into line $\mathbf{L_1}$, and so on.

4.3.3. Mapping Function

- The direct mapping technique is simple and inexpensive to implement.
- Its main disadvantage is that there is a fixed cache location for any given block.
  - Thus, if a program happens to reference words repeatedly from two different blocks that map into the same line, then the blocks will be continually swapped in the cache, and the hit ratio will be low.

4.3.3. Mapping Function

- ASSOCIATIVE MAPPING Associative mapping overcomes the disadvantage of direct mapping by permitting each main memory block to be loaded into any line of the cache (Figure 4.8b).
- In this case, the cache control logic interprets a memory address simply as a Tag and a Word field.
  - The Tag field uniquely identifies a block of main memory.
  - To determine whether a block is in the cache, the cache control logic must simultaneously examine every line's tag for a match.

4.3.3.Mapping Function

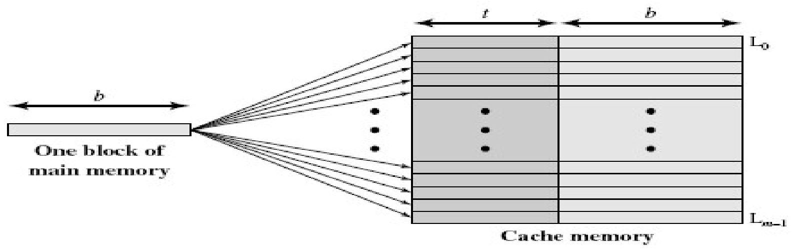


Figure 4.8b. Associative mapping

- With associative mapping, there is flexibility as to which block to replace when a new block is read into the cache.
- The principal disadvantage of associative mapping is the complex circuitry required to examine the tags of all cache lines in parallel.

4.3.3. Mapping Function

- SET-ASSOCIATIVE MAPPING Set-associative mapping is a compromise that exhibits the strengths of both the direct and associative approaches while reducing their disadvantages.
- In this case, the cache consists of a number sets, each of which consists of a number of lines
- The relationships are

  m= v *k

  i= j modulo v

4.3.3. Mapping Function

- where
  - ✓i =cache set number
  - ✓j =main memory block number
  - ✓m = number of lines in the cache
  - ✓v=number of sets
  - ✓K= number of lines in each set
- This is referred to as k-way set-associative mapping. With set-associative mapping, block **Bj** can be mapped into any of the lines of set **j**.

4.3.3. Mapping Function

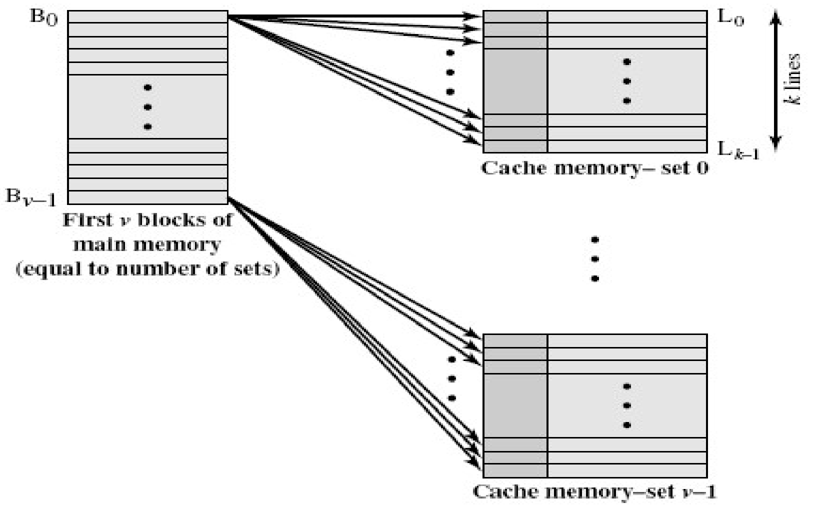


Figure 4.8c. v Associative–mapped caches

4.3.3. Mapping Function

- Figure 4.8c illustrates this mapping for the ***v*** first blocks of main memory.
  - As with associative mapping, each word maps into multiple cache lines. For set-associative mapping, each word maps into all the cache lines in a specific set, so that main memory block $\mathbf{B_0}$ maps into set **0**, and so on.
  - Thus, the set-associative cache can be physically implemented as ***v*** associative caches.

4.3.4. Replacement Algorithms

- Once the cache has been filled, when a new block is brought into the cache, one of the existing blocks must be replaced.
- For Direct mapping
  - No choice
  - Each block only maps to one line
  - Replace that line
- For the associative and set associative techniques, a replacement algorithm is needed. To achieve high speed, such an algorithm must be implemented in hardware. A number of algorithms have been tried.

4.3.4. Replacement Algorithms

- Probably the most effective is least recently used (LRU):
  - Replace that block in the set that has been in the cache longest with no reference to it.
  - LRU is the most popular replacement algorithm.
- Another possibility is first-in-first-out (FIFO):
  - Replace that block in the set that has been in the cache longest.
  - FIFO is easily implemented as a round-robin or circular buffer technique.

4.3.4. Replacement Algorithms

- Another possibility is least frequently used (LFU):
  - Replace that block in the set that has experienced the fewest references.
  - LFU could be implemented by associating a counter with each line.
- A technique not based on usage is to pick a line at random from among the candidate lines.
  - Simulation studies have shown that random replacement provides only slightly inferior performance to an algorithm based on usage

Computer Organization and Architecture

- ***Reference***: *Computer Organization and Architecture Designing for Performance (8th Edition), William Stallings, Prentice Hall, Upper Saddle River, NJ 07458.*