Cahe Memory*

Nguyen Thi Hoang Lan

1 1. Cache Memory Overview

Cache memory is the fast and small memory. The basis idea of using a cache is simple. When CPU need a word, it first looks in the cache. Only if the word is not there does it go to main memory. If a substantial fraction of the words are in the cache, the average access time can be greatly reduced. So the success or faillure thus depends on what fraction of the words are in cache. The idea solution is that when a word is referenced, it and some of its neighbors are brought from the large slow memory into the cache, so that the next time it can be accessed quickly.

- Improving Performance

The memory mean access time, TM, is considered as follow:
TM = c + (1- h).m
Where:
c : the cache access time
m: the main memory access time
h: the hit ratio, which is the fraction of all references that can be satisfied out of cache (the success of cache), also the miss ratio is 1-h.

- Cache memory is a critical component of the memory hierarchy.

*Version 1.1: Jul 15, 2009 5:44 am -0500
†http://creativecommons.org/licenses/by/3.0/

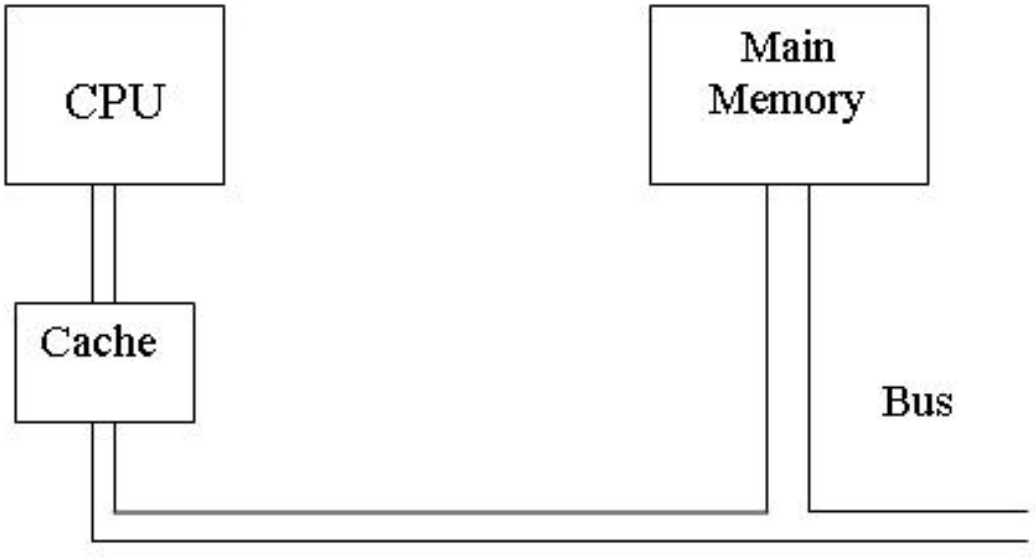

Figure 1

Figure 10.1. The cache is logically between the CPU and main memory in the the memory hierarchy. Physically, there are several possible places it could be located.

- Characteristics

– Compared to the size of main memory, cache is relatively small
– Operates at or near the speed of the processor
– Cache is very expensive compared to main memory
– Cache contains copies of sections of main memory

- Cache is considered as the main memory interface

– Assume an access to main memory causes a block of K words to be transferred to the cache
– The block transferred from main memory is stored in the cache as a single unit called a slot, line, or page
– Once copied to the cache, individual words within a line can be accessed by the CPU
– Because of the high speeds involved with the cache, management of the data transfer and storage in the cache is done in hardware
- If there are 2n words in the main memory, then there will be M=2n /K blocks in the memory
– M will be much greater than the number of lines, C, in the cache
– Every line of data in the cache must be tagged in some way to identify what main memory block it is. The line of data and its tag are stored in the cache.

- Factors in the cache design

- Mapping function between main memory and the cache
  - Line replacement algorithm
  - Write policy
  - Block size
  - Number and type of caches

2 2. Cache Organization and Operations

Mapping function organization: Mapping functions since M≫C, how are blocks mapped to specific lines in cache

2.1 2.1 Direct mapping

a/ Organization:

The simplest cache is known as a direct-mapped cache or direct mapping, it is shown in Figure 10.2.

– Direct mapping cache treats a main memory address as 3 distinct fields
+ Tag identifier
+ Line number identifier
+ Word identifier (offset)
– Word identifier specifies the specific word (or addressable unit) in a cache line that is to be read
– Line identifier specifies the physical line in cache that will hold the referenced address
– The tag is stored in the cache along with the data words of the line

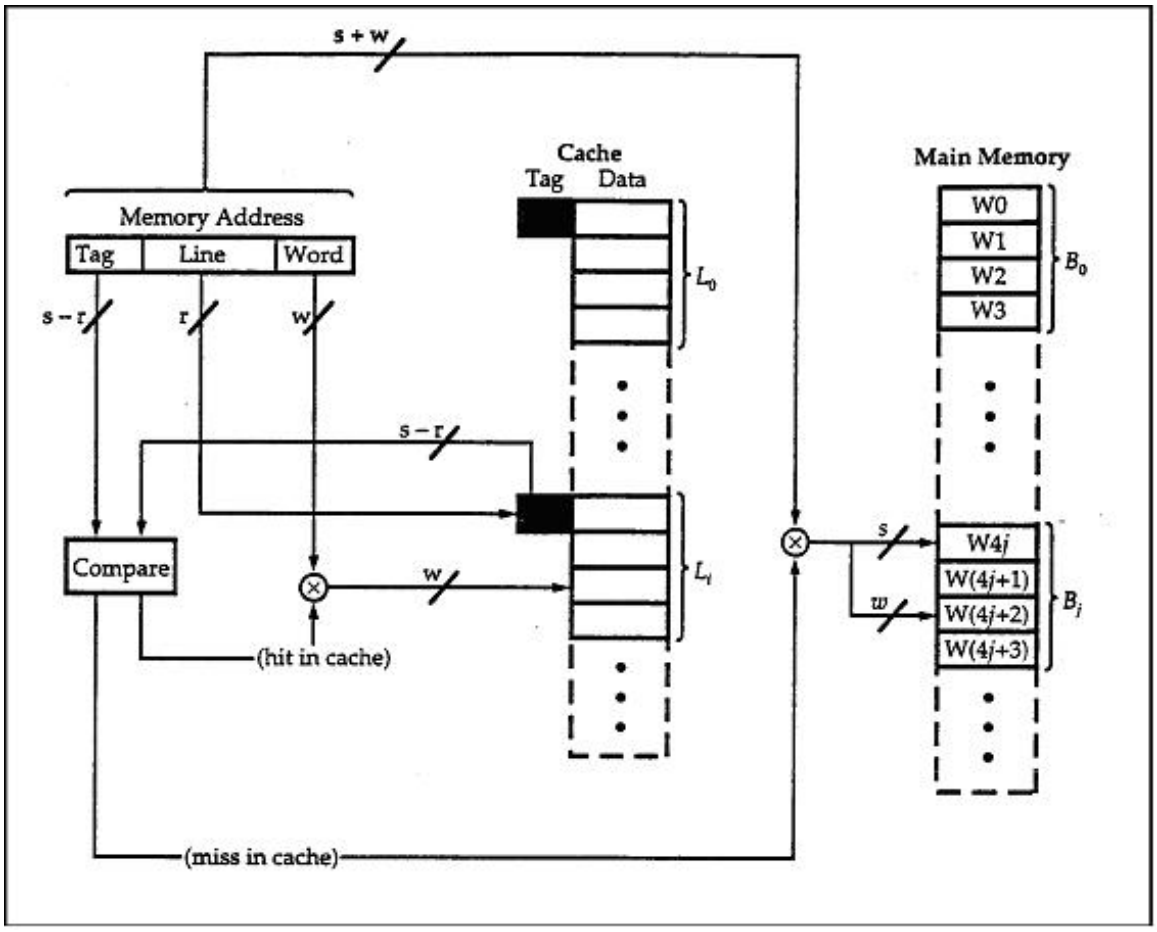

Figure 2

Figure 10.2. Direct mapping cache Organization

For every memory reference that the CPU makes, the specific line that would hold the reference (if it is has already been copied into the cache) is determined. The tag held in that line is checked to see if the correct block is in the cache

b/ Operations

- Each main memory block is assigned to a specific line in the cache:

i = j modulo C, where i is the cache line number assigned to main memory block j

– If M=64, C=4:

Line 0 can hold blocks 0, 4, 8, 12, ...

Line 1 can hold blocks 1, 5, 9, 13, ...

Line 2 can hold blocks 2, 6, 10, 14, ...

Line 3 can hold blocks 3, 7, 11, 15, ...

– Example:

Memory size of 1 MB (20 address bits) addressable to the individual byte

Cache size of 1 K lines, each holding 8 bytes:

Word id = 3 bits
Line id = 10 bits
Tag id = 7 bits
Where is the byte stored at main memory
location $ABCDE stored?
$ABCDE=1010101 1110011011 110
Cache line $39B, word offset $6, tag $55
c/ Remarks

- Advantages of direct mapping

+ Easy to implement
+ Relatively inexpensive to implement
+ Easy to determine where a main memory reference can be found in cache

- Disadvantage

+ Each main memory block is mapped to a specific cache line
+ Through locality of reference, it is possible to repeatedly reference to blocks that map to the same line number
+ These blocks will be constantly swapped in and out of cache, causing the hit ratio to be low.

2.2 2.2 Associative mapping

a/ Organization

A set-associative cache or associative mapping is inhenrently more complicated than a direct-mapped cache because although the correct cache entry to examine can be computed from the memory address being referenced, a set of n cache entries must be checked to see if the need line is present.

Associative mapping can overcomes direct mapping's main of the direct mapping, the associate cache organization is illustrated in Figure 10.3.

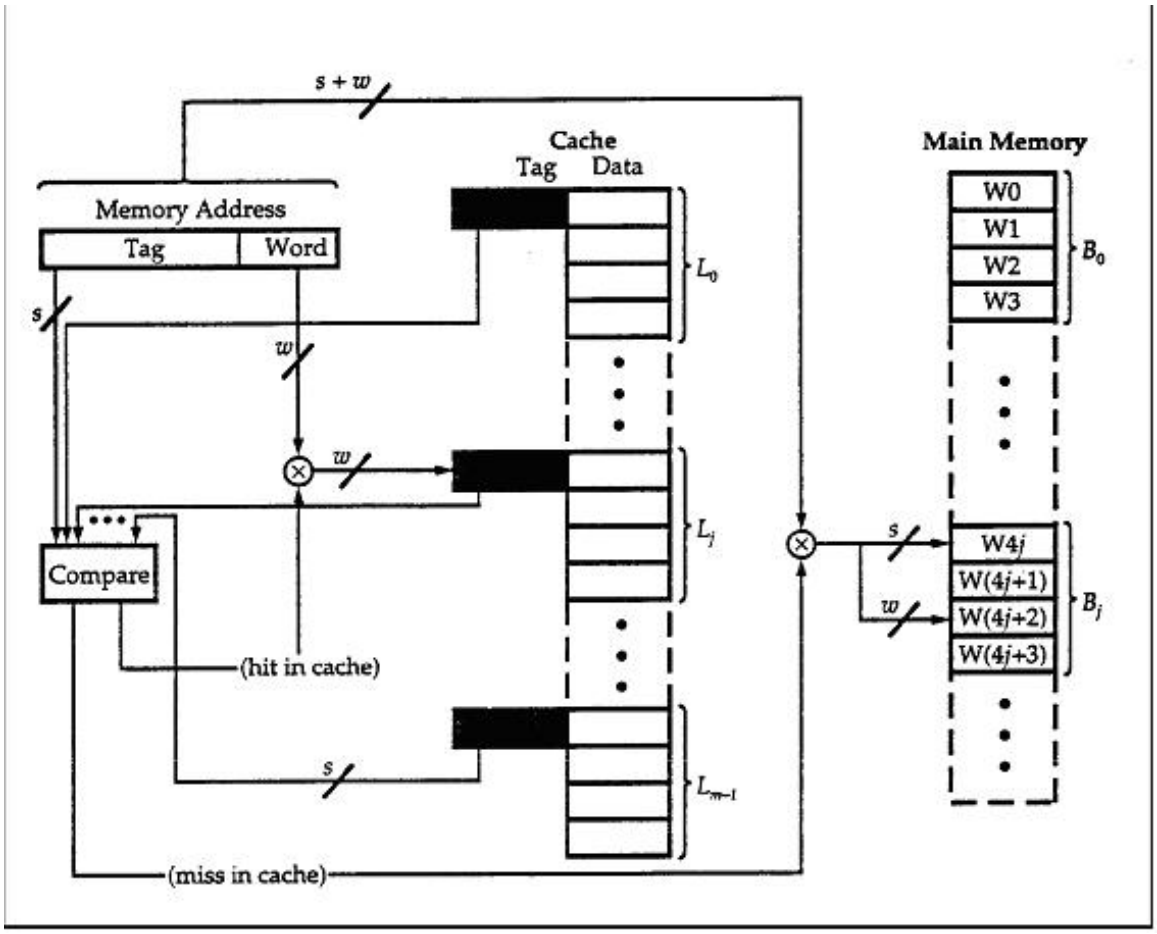

Figure 3

Figure 10.3. Associate Cache Organization

– Operation must examine each line in the cache to find the right memory block

+ Examine the line tag id for each line

+ Slow process for large caches!

– Line numbers (ids) have no meaning in the cache

+ Parse the main memory address into 2 fields (tag and word offset) rather than 3 as in direct mapping

– Implementation of cache in 2 parts:

+ Part SRAM: The lines themselves in SRAM

+ The tag storage in associative memory

– Perform an associative search over all tags

b/ Operation example

With the same example: Memory size of 1 MB (20 address bits) addressable to the individual byte. Cache size of 1 K lines, each holding 8 bytes:

Word id = 3 bits

Tag id = 17 bits

Where is the byte stored at main memory location $ABCDE stored?
$ABCDE=10101011110011011 110
Cache line unknown, word offset $6, tag $1579D.
c/ Remarks

- Advantages

- Fast
- Flexible

- Disadvantage
- Implementation cost

Example above has 8 KB cache and requires 1024 x 17 = 17,408 bits of
associative memory for the tags!

2.3 2.3 Set associative mapping

a/ Organization
The associative mapping is considered as compromise between direct and fully associative mappings that builds on the strengths of both
– Divide cache into a number of sets (v), each set holding a number of lines (k)
– A main memory block can be stored in any one of the k lines in a set such that
set number = j modulo v
– If a set can hold X lines, the cache is referred to as an X-way set associative cache
Most cache systems today that use set associative mapping are 2- or 4-way set
Associative.

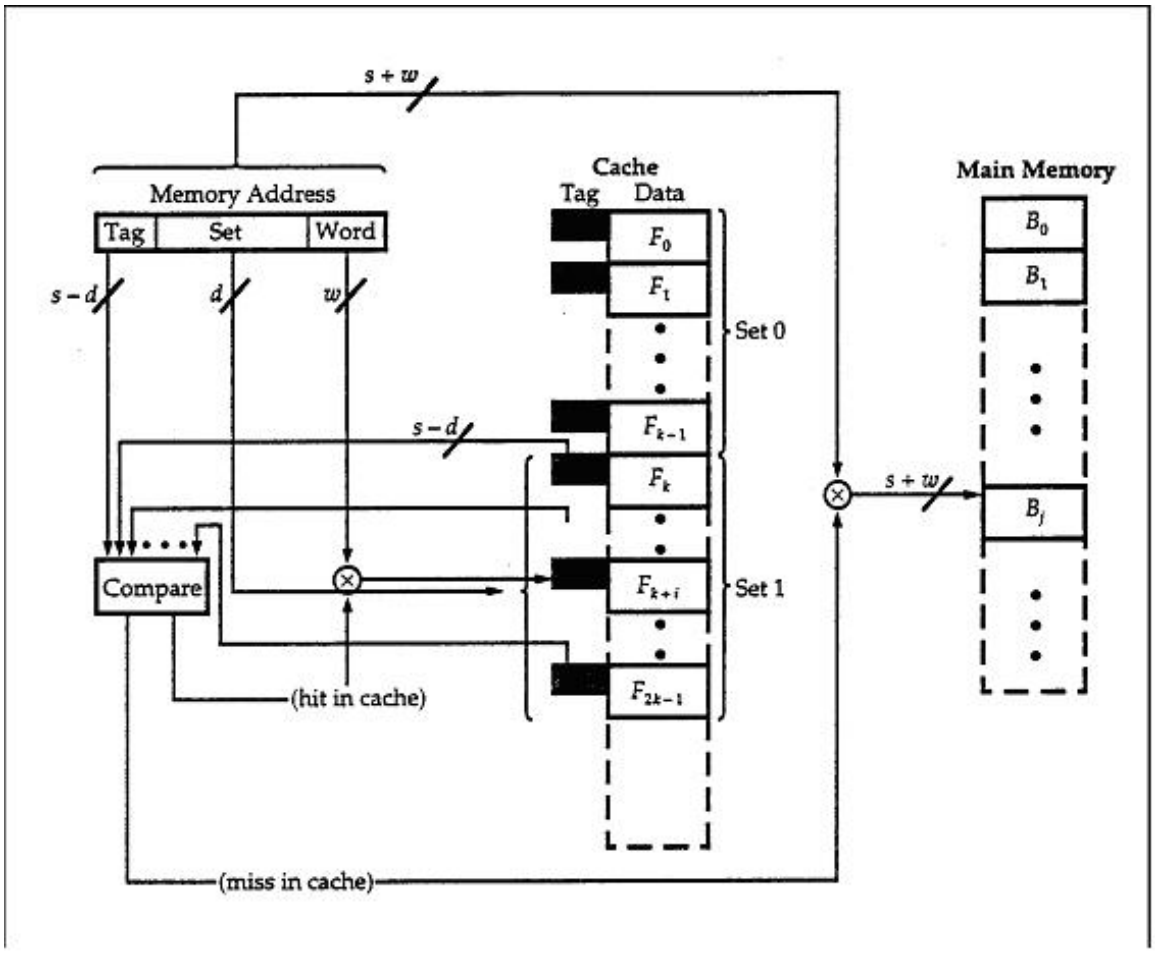

Figure 4

Figure. 10. 4 Set associative cache organization
b/ Example
Assume the 1024 lines are 4-way set associative:
1024/4 = 256 sets
Word id = 3 bits
Set id = 8 bits
Tag id = 9 bits
Where is the byte stored at main memory
Location $ABCDE stored?
$ABCDE=101010111 10011011 110
Cache set $9B, word offset $6, tag $157

3 3. Replacement Algorithms

As we known, cache is the fast and small memory. when new block is brough into the cache, one of the blocks existing must be replaced by the new block that is to be read from memory.

For direct mapping, there is only one possible line for any particular block and no choice is possible. For the associative cache or a set associative cache, a replacement algorithm is needed.

A number of algorithms can be tried:

– Least Recently Used (LRU)

– First In First Out (FIFO)

– Least Frequently Used (LFU)

– Random

Probably the most effective algorithm is least recently used (LRU): Replace that block in the set that has been in the cache longest with no refrence.

4 4. Performances

4.1 4.1 Write policy

– When a line is to be replaced, must update the original copy of the line in main memory if any addressable unit in the line has been changed

– Write through

+ Anytime a word in cache is changed, it is also changed in main memory

+ Both copies always agree

+ Generates lots of memory writes to main memory

– Write back

+ During a write, only change the contents of the cache

+ Update main memory only when the cache line is to be replaced

+Causes "cache coherency" problems – different values for the contents of an address are in the cache and the main memory

+ Complex circuitry to avoid this problem

4.2 4.2 Block / line sizes

– How much data should be transferred from main memory to the cache in a single memory reference

– Complex relationship between block size and hit ratio as well as the operation of the system bus itself

– As block size increases,

+ Locality of reference predicts that the additional information transferred will likely be used and thus increases the hit ratio (good)

+ Number of blocks in cache goes down, limiting the total number of blocks in the cache (bad)

+ As the block size gets big, the probability of referencing all the data in it goes down (hit ratio goes down) (bad)

+ Size of 4-8 addressable units seems about right for current systems

4.3 Number of caches

Single vs. 2-level cache:

- Modern CPU chips have on-board cache (Level 1 – L1):

L1 provides best performance gains

- Secondary, off-chip cache (Level 2) provides higher speed access to main memory

L2 is generally 512KB or less more than this is not cost-effective.