Cache Memories

Carl Hamacher, Zvonko Vranesic and Safwat Zaky, Computer Organization and Embedded Systems, (6e), McGraw Hill Publication, 2017.

Ch 8: 8.6, 8.6.1, 8.6.2,

Cache Memories

- What is cache?
- Why we need it?
- Locality of reference
- Cache block – cache line

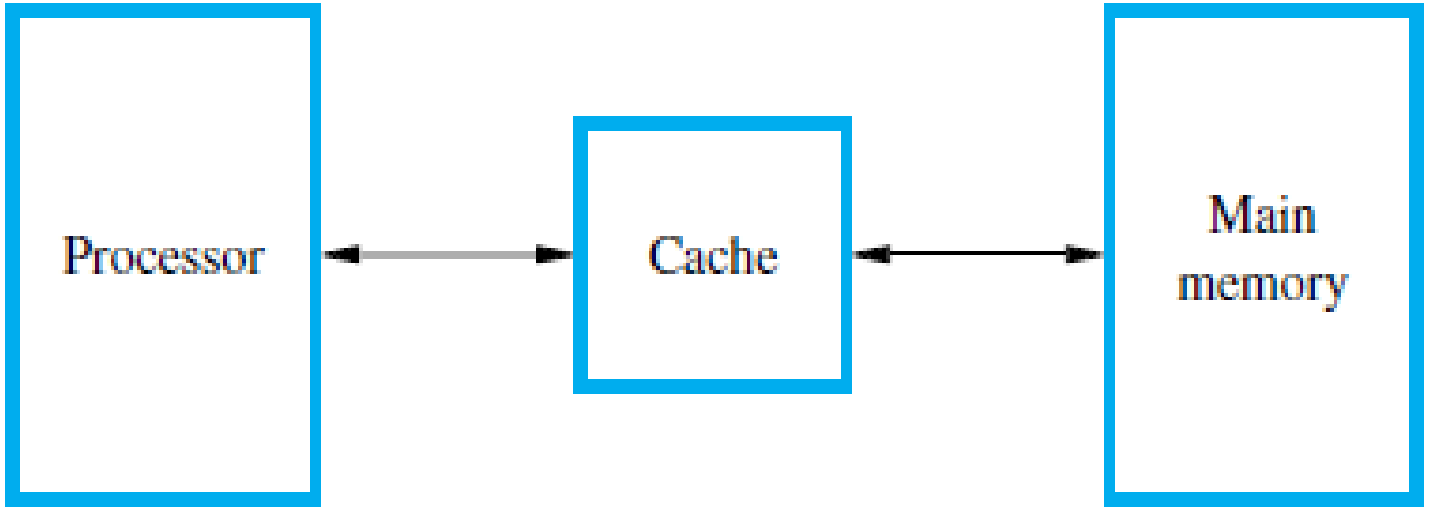


Figure 8.15 Use of a cache memory.

Cache Memories

- The cache is a small and very fast memory, interposed between the processor and the main memory.
- Its purpose is to make the main memory appear to the processor to be much faster than it actually is.
- The effectiveness of this approach is based on a property of computer programs called *locality of reference*.
  - temporal- a recently executed instruction is likely to be executed again very soon.
  - spatial- instructions close to a recently executed instruction are also likely to be executed soon
- Cache block – cache line
  - A set of contiguous address locations of some size

Cache Memories

- Mapping function
- Replacement algorithm
- Hit / miss
- Write-through / Write-back
- Load through

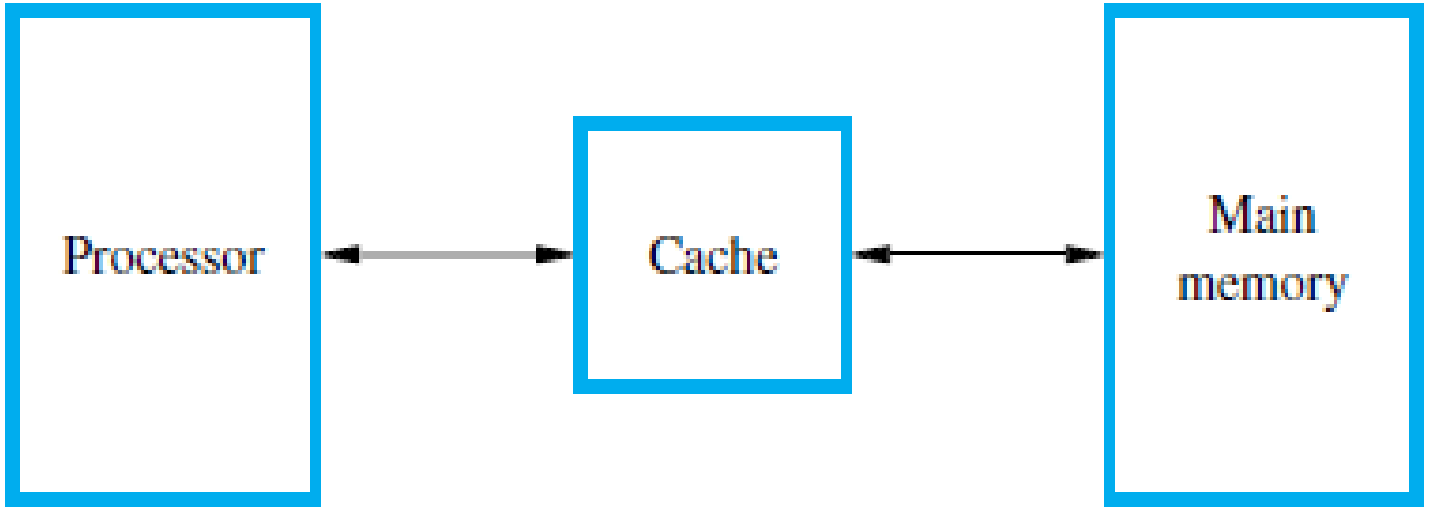


Figure 8.15 Use of a cache memory.

Cache Memories

- The correspondence between the main memory blocks and those in the cache is specified by a ***mapping function***.

- When the cache is full and a memory word (instruction or data) that is not in the cache is referenced, the cache control hardware must decide which block should be removed to create space for the new block that contains the referenced word.

- The collection of rules for making this decision constitutes the cache's ***replacement algorithm.***

Cache Hits

- The processor does not need to know explicitly about the existence of the cache.
- It simply issues Read and Write requests using addresses that refer to locations in the memory.
- The cache control circuitry determines whether the requested word currently exists in the cache.
- If it does, the Read or Write operation is performed on the appropriate cache location. –
  - In this case, a ***read or write hit*** is said to have occurred
- The main memory is not involved when there is a cache hit in a Read operation

Cache Hits

- Write Operation –two ways
  - **write-through protocol** - both the cache location and the main memory location are updated.
  - **write-back, or copy-back** - protocol-updates only the cache location and marks the block containing it with an associated flag bit, often called the dirty or modified bit.
    - The main memory location of the word is updated later, when the block containing this marked word is removed from the cache to make room for a new block..

write-through V/S write-back

- The write-through protocol is simpler than the write-back protocol
  - But it results in unnecessary Write operations in the main memory when a given cache word is updated several times during its cache residency.
- The write-back protocol also involves unnecessary Write operations
  - Because all words of the block are eventually written back, even if only a single word has been changed while the block was in the cache.
- The write-back protocol is used most often, to take advantage of the high speed with which data blocks can be transferred to memory chips.

Cache Misses

- A Read operation for a word that is not in the cache constitutes a Read miss.
- It causes the block of words containing the requested word to be copied from the main memory into the cache.
- After the entire block is loaded into the cache, the word requested is forwarded to the processor.
- Alternatively, this word may be sent to the processor as soon as it is read from the main memory. The latter approach, which is called load-through, or early restart, reduces the processor's waiting time somewhat, at the expense of more complex circuitry.
- When a Write miss occurs in a computer that uses the write-through protocol, the information is written directly into the main memory.
- For the write-back protocol, the block containing the addressed word is first brought into the cache, and then the desired word in the cache is overwritten with the new information.

Mapping Functions

There are several possible methods for determining where memory blocks are placed in the cache

- Direct Mapping
- Associative Mapping
- Set-Associative Mapping

  - Consider a cache consisting of 128 blocks of 16 words each, for a total of 2048 (2K) words
  - Assume that the main memory is addressable by a 16-bit address.
  - The main memory has 64K words, which we will view as 4K blocks of 16 words each.
  - For simplicity, we have assumed that consecutive addresses refer to consecutive words.

Direct Mapping

- Block j of main memory maps onto block j modulo 128 of the cache
  - Thus, whenever one of the main memory blocks 0, 128, 256, . . . is loaded into the cache, it is stored in cache block 0.
  - Blocks 1, 129, 257, . . . are stored in cache block 1, and so on.

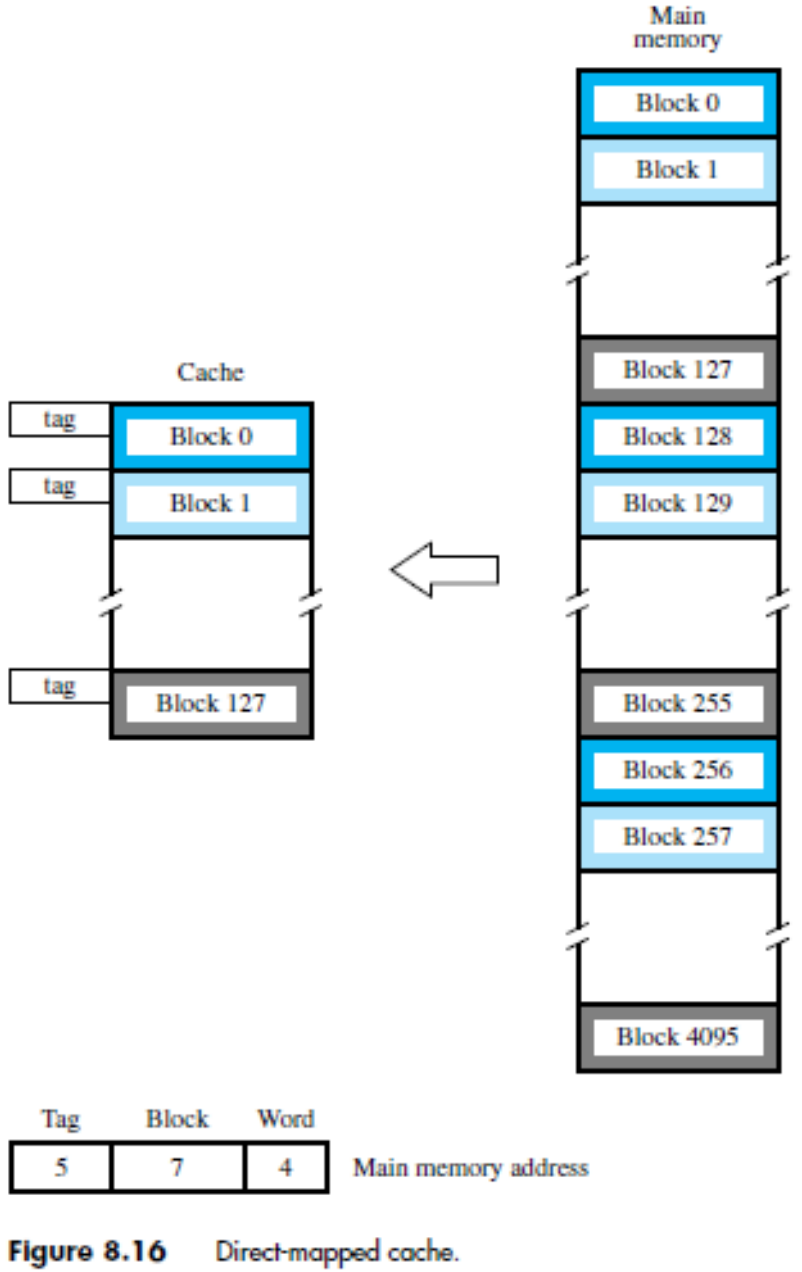


Figure 8.16 Direct-mapped cache.

Direct Mapping

The memory address can be divided into three fields, as shown in Figure 8.16.

- The low-order 4 bits select one of 16 words in a block. (each block has $16=2^4$ words)
- When a new block enters the cache, the 7-bit cache block field determines the cache position in which this block must be stored. ($128=2^7$)
- The high-order 5 bits of the memory address of the block are stored in 5 tag bits associated with its location in the cache.
- The tag bits identify which of the 32 main memory blocks mapped into this cache position is currently resident in the cache. (Identify which of the 32 blocks that are resident in the cache (4096/128).

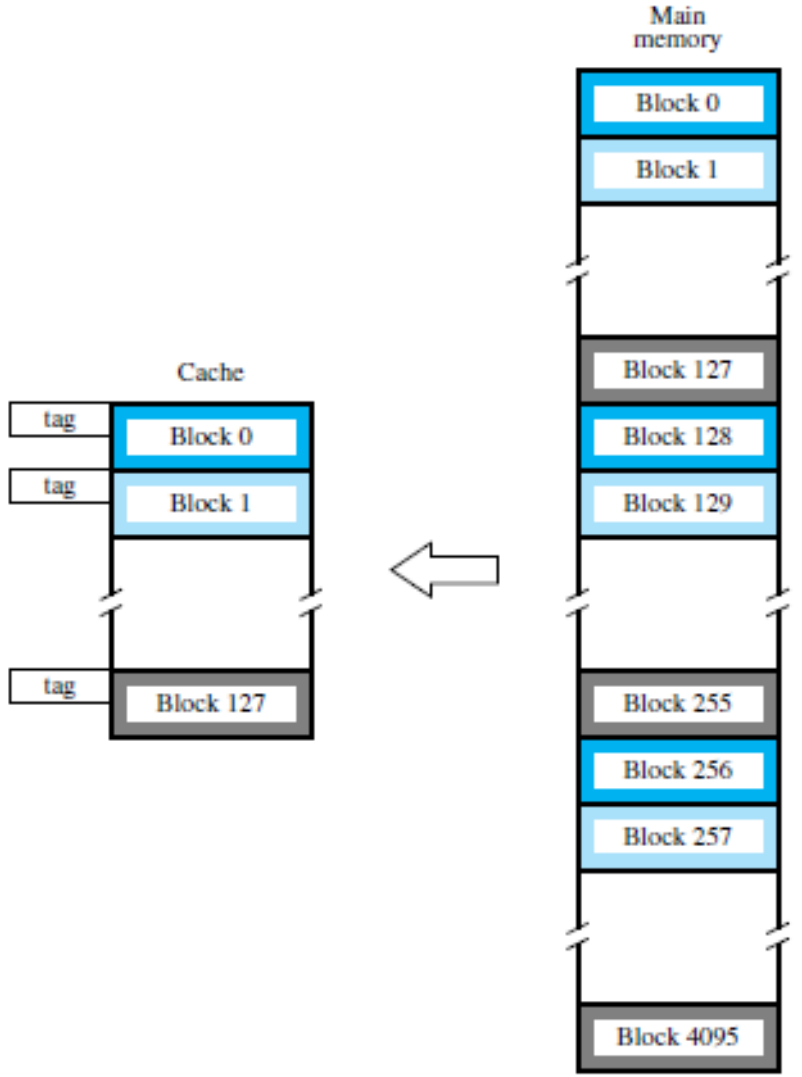


Figure 8.16 Direct-mapped cache.

- Example: 11101,1111111,1100
- Tag: 11101
- Block: 1111111=127, in the 127th block of the cache
- Word:1100=12, the 12th word of the 127th block in the cache

Associative Mapping

Main memory block can be placed into any cache block position

- 12 tag bits are required to identify a memory block when it is resident in the cache.
- The tag bits of an address received from the processor are compared to the tag bits of each block of the cache to see if the desired block is present.
- This is called the associative-mapping technique.

Main Memory address

- 4: one of 16 words. (each block has $16=2^4$ words)
- 12: 12 tag bits Identify which of the 4096 blocks that are resident in the cache $4096=2^{12}$.

- Example: 111011111111,1100
- Tag: 111011111111
- Word:1100=12, the 12th word of a block in the cache

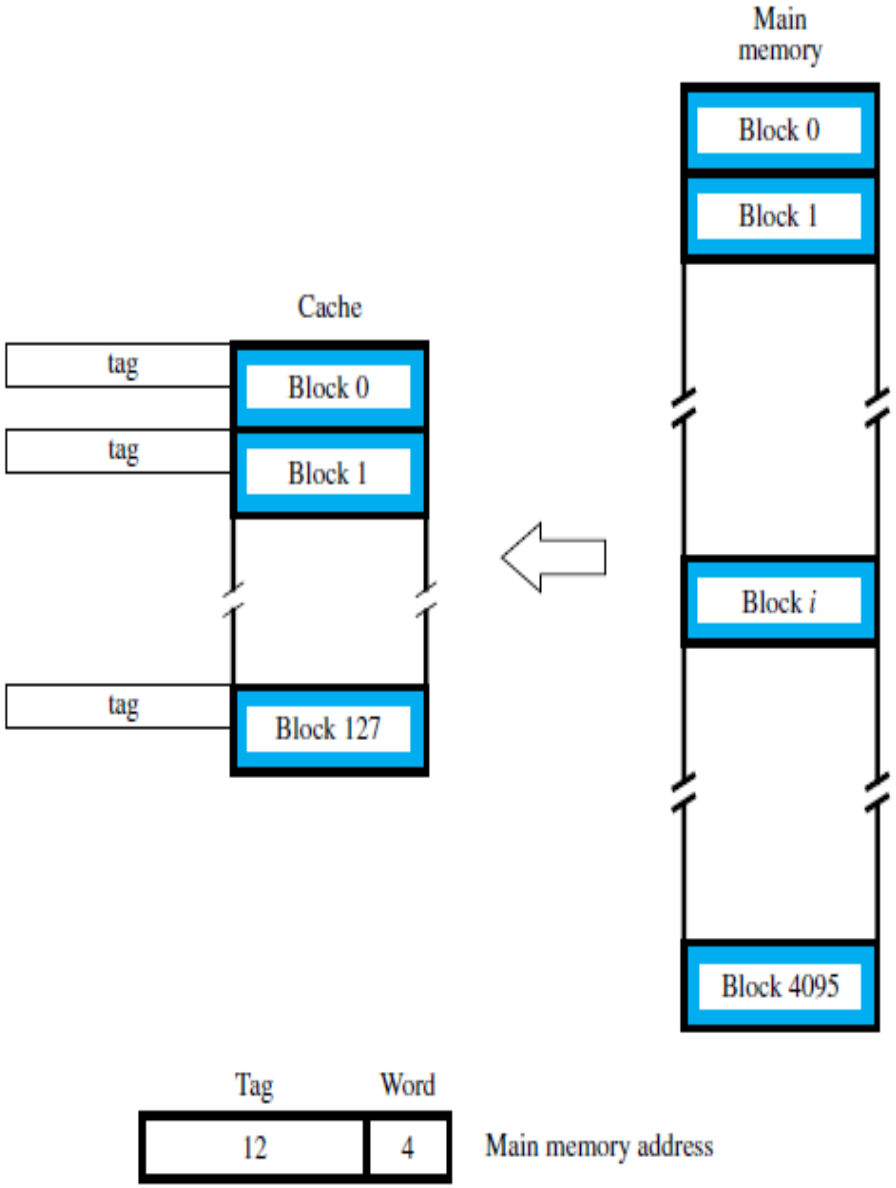


Figure 8.17 Associative-mapped cache.

SET Associative Mapping

- Set Associative mapping is a combination of the direct- and associative-mapping techniques.
- The blocks of the cache are grouped into sets, and the mapping allows a block of the main memory to reside in any block of a specific set.
- Hence, the contention problem of the direct method is eased by having a few choices for block placement.
- At the same time, the hardware cost is reduced by decreasing the size of the associative search.

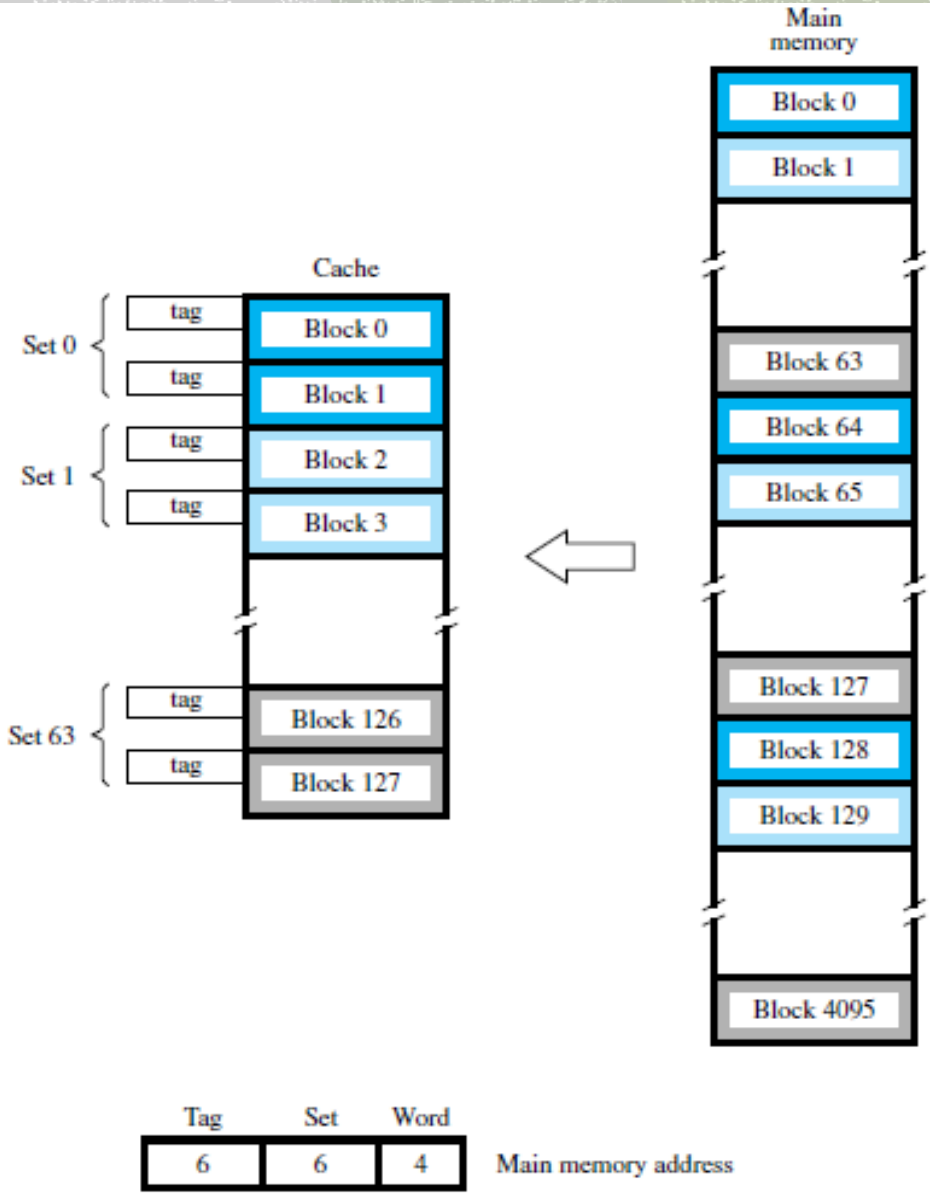


Figure 8.18 Set-associative-mapped cache with two blocks per set.

SET Associative Mapping

For a cache with two blocks per set, memory blocks 0, 64, 128, . . . , 4032 map into cache set 0, and they can occupy either of the two block positions within this set.

Main Memory address

- 4: one of 16 words. (each block has $16=2^4$ words)
- 6: points to a particular set in the cache ($128/2=64=2^6$)
- 6: 6 tag bits is used to check if the desired block is present ($4096/64= 64= 2^6$).

- Example: 111011,111111,1100
- Tag: 111011
- Set: 111111=63, in the 63th set of the cache
- Word:1100=12, the 12th word of the 63th set in the cache

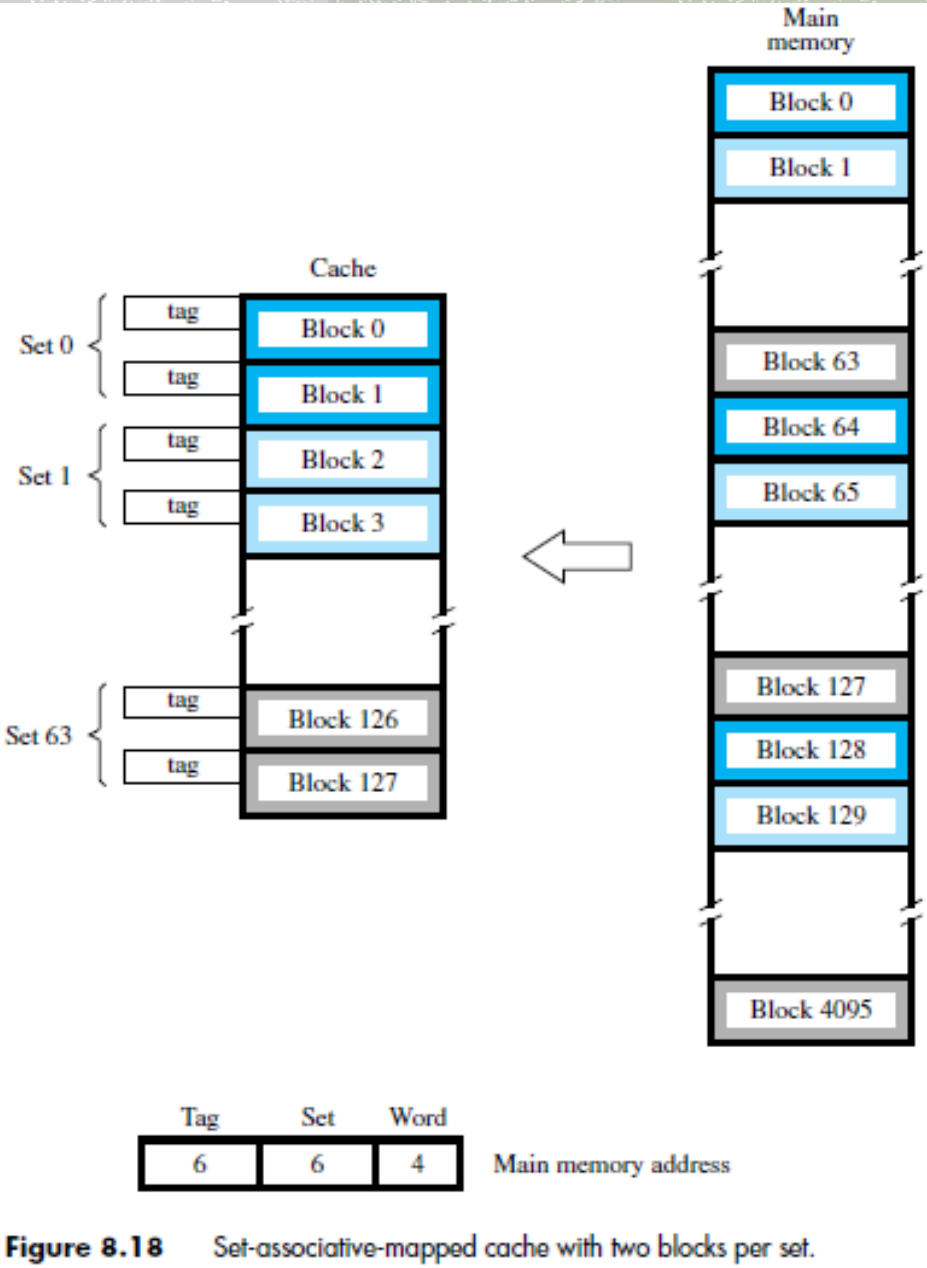


Figure 8.18 Set-associative-mapped cache with two blocks per set.

Exercise problem 1:

- 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 the size of tag, set and word.

- Solution:
  - No. of blocks in the set= 2
  - Size of the cache=16KB
  - No. of blocks in cache= 16KB/256= 16384B/ 256B= 64
  - No. of sets in the cache=64/2=32= 2^5
  - No. of bits in set field=5
  - No. of words(bytes) in a block=256=2^8
  - No. of bits in word field is 8
  - No. of bits in main memory address is 17 bits(log 128KB to base2)
  - No. of bits in the tag field is 4 (17- 5-8)

Exercise problem 2:

- A block-set-associative cache consists of a total of 64 blocks, divided into 4-block sets. The main memory contains 4096 blocks, each consisting of 32 words, how many bits are there in each of the Tag, Set, and Word fields?

- Solution:
  - Number of sets = 64/4 = 16
  - No. of bits in Set field = 4($2^4 = 16$)
  - Word bits = 5 bits ($2^5 = 32$)
  - Tag=17-(4+5)=8

Exercise problem 3:

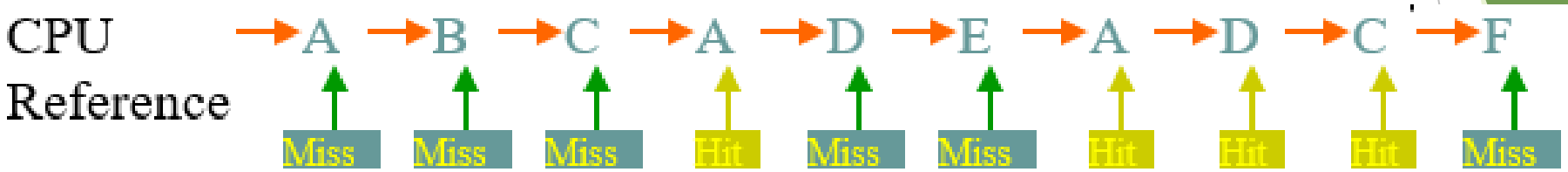
- A block-set-associative cache consists of a total of 64 blocks, divided into 4-block sets. The main memory contains 4096 blocks, each consisting of 128 words.
  a)How many bits are there in MM address?
  b)How many bits are there in each of the TAG, SET & WORD fields

- Solution:
  - Number of sets = 64/4 = 16
  - Set bits = 4($2^4 = 16$)
  - Number of words = 128
  - Word bits = 7 bits ($2^7 = 128$)
  - MM capacity : 4096 x 128 (2^{12} x $2^7 = 2^{19}$)
  - a)Number of bits in memory address = 19 bits
  - b) 8 4 7
    TAG SET WORD
  - TAG bits = 19 – (7+4) = 8 bits.

Replacement Algorithms

- In a direct-mapped cache, the position of each block is predetermined by its address; hence, the replacement strategy is trivial.
- In associative and set-associative caches there exists some flexibility.
- When a new block is to be brought into the cache and all the positions that
- it may occupy are full, the cache controller must decide which of the old blocks to overwrite.
- This is an important issue, because the decision can be a strong determining factor in system performance. (Difficult to determine which blocks to be removed)
- In general, the objective is to keep blocks in the cache that are likely to be referenced in the near future.
- Therefore, when a block is to be overwritten, it is sensible to overwrite the one that has gone the longest time without being referenced.
- This block is called the least recently used (LRU) block, and the technique is called the LRU replacement algorithm.
- To use the LRU algorithm, the cache controller must track references to all blocks as
- computation proceeds
- Increase / clear track counters when a hit/miss occurs

Replacement Algorithms



A	A	A	**A**	A	A	**A**	A	A	A
	B	B	B	B	E	E	E	E	F
		C	C	C	C	C	C	**C**	C
				D	D	D	**D**	D	D

Hit Ratio = 4 / 10 = 0.4

LRU Algorithm

- The cache controller must track references to all blocks as computation proceeds.
- Suppose it is required to track the LRU block of a four-block set in a set-associative cache.
- A 2-bit counter can be used for each block.
- When a hit occurs,
  - the counter of the block that is referenced is set to 0.
  - Counters with values originally lower than the referenced one are incremented by one, and
  - all others remain unchanged.
- When a miss occurs and the set is not full,
  - the counter associated with the new block loaded from the main memory is set to 0, and
  - the values of all other counters are increased by one.
- When a miss occurs and the set is full,
  - the block with the counter value 3 is removed,
  - the new block is put in its place, and its counter is set to 0.
  - The other three block counters are incremented by one.
- It can be easily verified that the counter values of occupied blocks are always distinct

LRU Algorithm

- Assume counter values as shown

00	Block0
01	Block1
10	Block2
11	Block3

- Suppose hit occurs for block 2

01	Block0
10	Block1
00	Block2
11	Block3