Computer Organization & Architecture

Chapter 8 – Cache Memories

Zhang Yang 张杨

cszyang@scut.edu.cn

Autumn 2021

Contents of this lecture

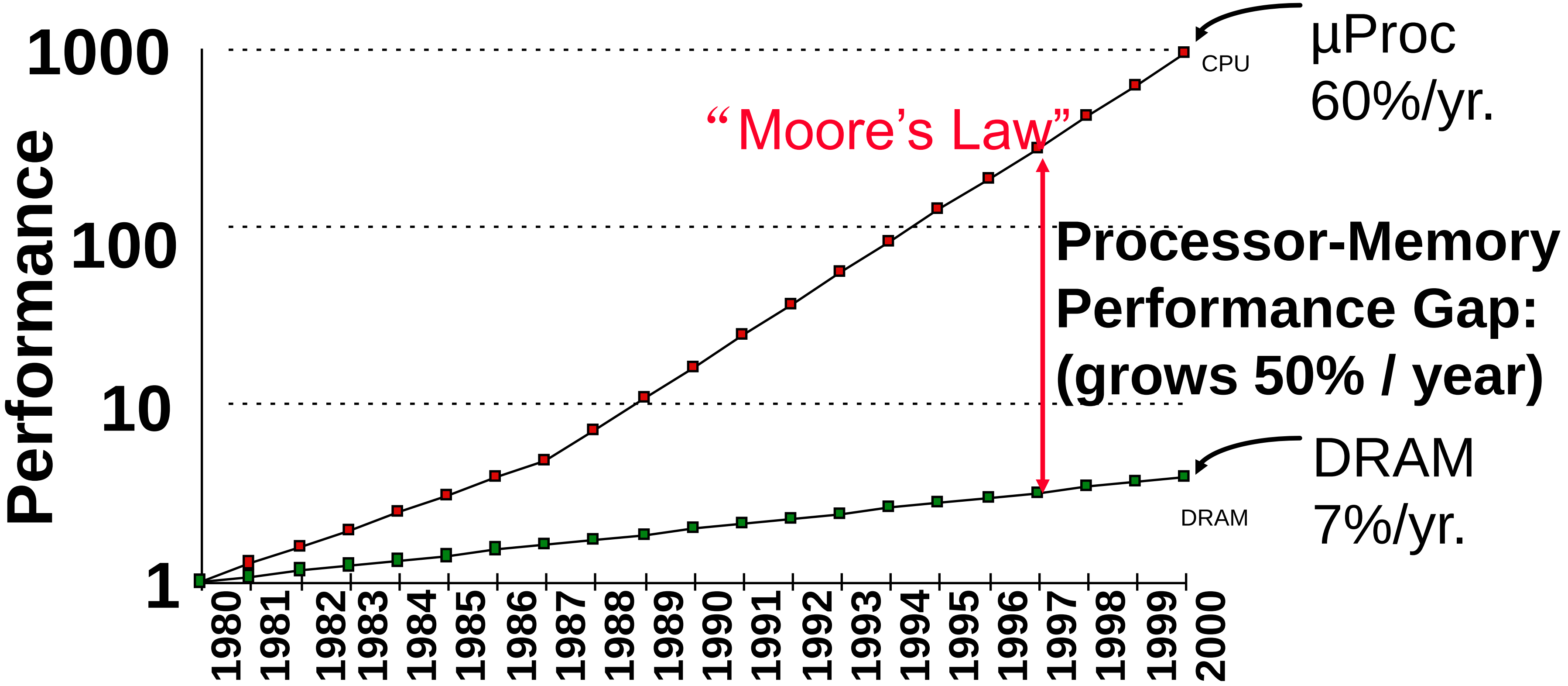
- 8.6 Cache Memories
  - Cache Principle & Policy
  - Mapping Schemes
  - Example of Mapping Technique
  - Solved Problems Example 8.3 (P324)

Cache Principles & Policy

- Purpose of Cache
- Organization of Cache/Main-Memory System
- Placement of Cache
- Cache Read Operation
- Valid Bit
- Hit and Miss
- Write Policy
- Cache Capacity (补充)
- Summary

Purpose of Cache Memory (1)

- CPU-DRAM Gap



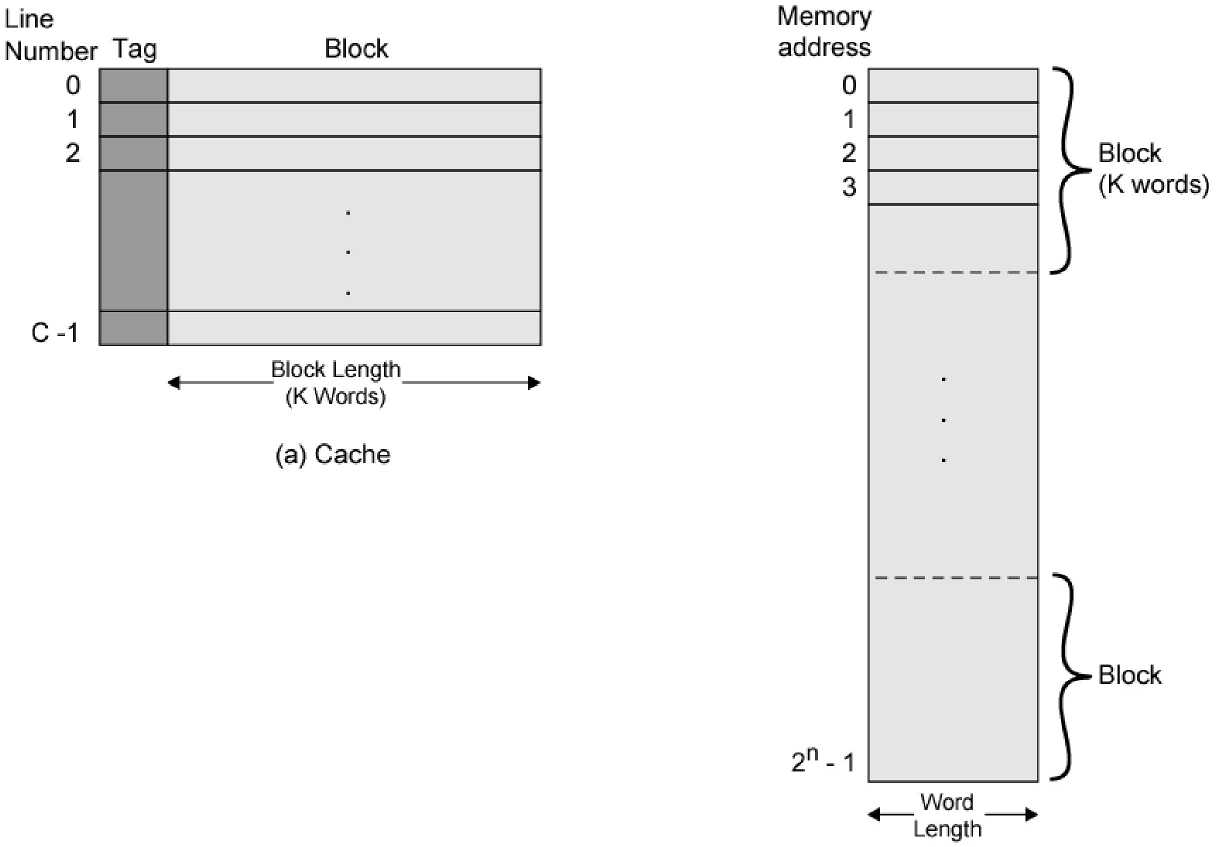
- □ 1980: no cache in µproc; 1995 2-level cache on chip (1989 first Intel µproc with a cache on chip)

Purpose of Cache Memory (2)

- Cache memories are small, fast SRAM-based memories managed automatically in hardware.
- Purpose of Cache Memory
  - Give main memory speed approaching that of the fastest memories available.
  - At the same time, provide a large memory size at the price of less expensive types of semiconductor memories.

Organization of Cache/Main-Memory System (1)

- Cache and Main Memory Organization Figure



(a) Cache

(b) Main memory

Number of Main Memory Block:

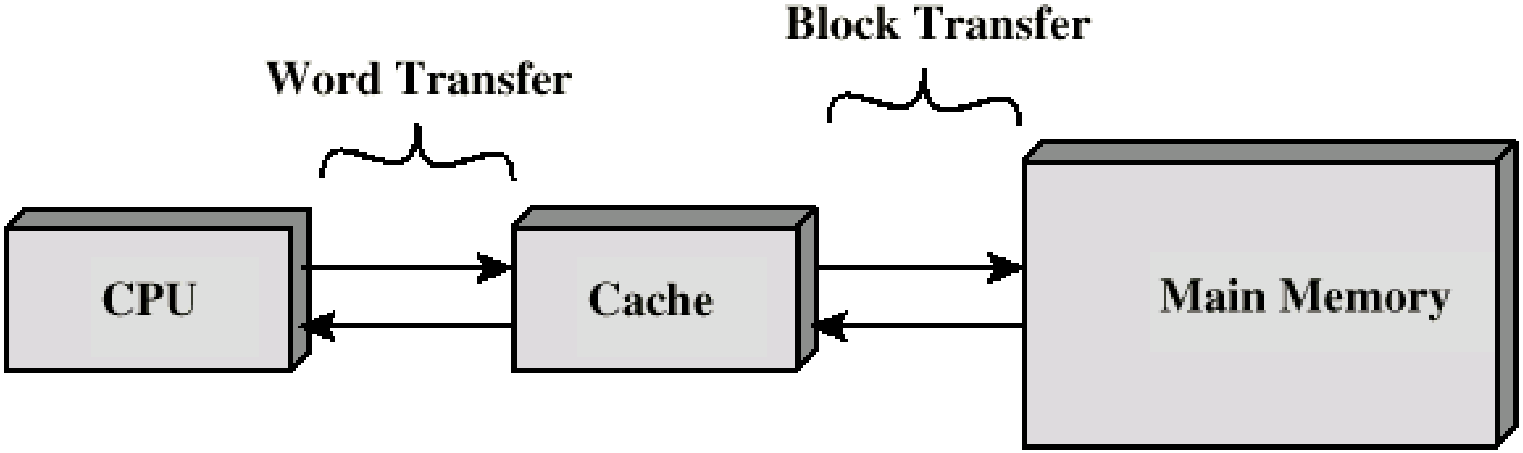
$2^n/K$

Organization of Cache/Main-Memory System (2)

- Each entry contains a block from main memory.
  - Copy of contents of several consecutive words/bytes in main memory.
  - Tag is also stored with each block-identifies addresses of these words/bytes.
- When one word/byte from a block is needed
  - If block not already in cache, entire block loaded.
- Block stayed in cache until entry must be reused.

Placement of Cache

- Cache sits between the CPU and main memory.
- Serves as a buffer between CPU and main memory.



- Hold frequently accessed blocks of main memory.

Hit and Miss (1)

- Cache Hit
  - A cache access finds data resident in the cache memory.
- Cache Miss
  - A cache access does not find data resident, forcing access to main memory.
- Hit Rate
  - The total cache hits divided by the total number of memory requests expressed as a percentage over a time interval.
  - High hit rates, well over 0.9, are essential for high-performance computers.

Hit and Miss (2)

- Miss Rate
  - The total cache misses divided by the total number of memory requests expressed as a percentage over a time interval.
  - Miss Rate = 1– (Hit Rate)

Cache Read Operation (1)

- 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.

Cache Read Operation (2)

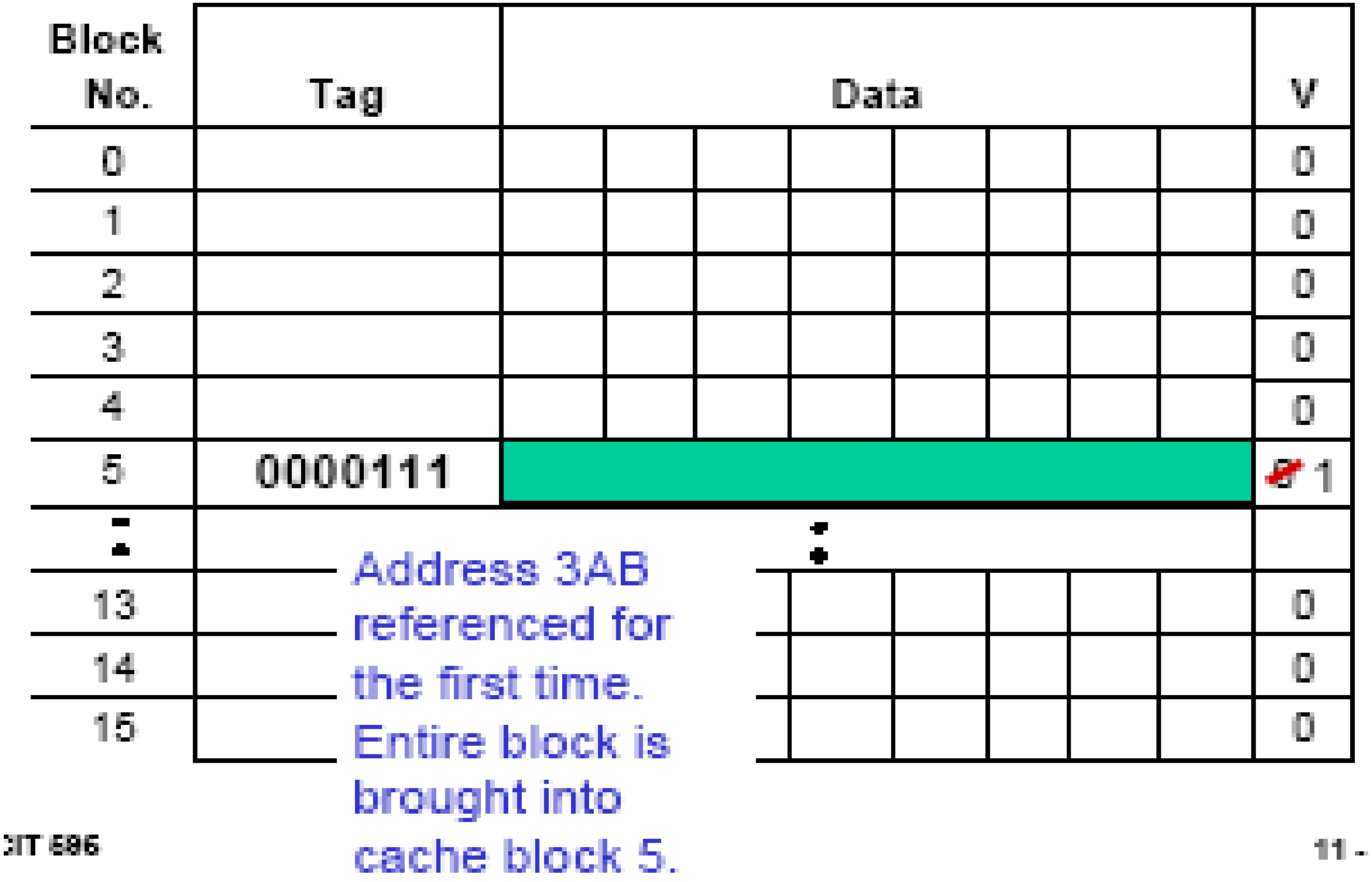
- Processor: Read Request
  - Cache Read Hit
    - Forward the requested word to the processor.
  - Cache Read Miss, Two Approaches
    - Copy the block of words containing the requested word from the main memory into the cache. Then forward the particular word requested is to the processor.
    - Load Through/Early Restart
      - Forward the requested word at the same time copy the block of words containing the requested word from the main memory into the cache.

Valid Bit (1)

- When power is first turned on, the cache contains *no* valid data.
- A control bit, usually called the *valid bit*, must be provided for each cache block to indicate whether the data in that block are valid.
- 0 – not valid, 1 – valid
  - The valid bits of all cache blocks are set to 0 when power is initially applied to the system.
  - If data from main memory is got into cache for a particular block, then valid bit for that field is set to 1.

Valid Bit (2)

- Example



Block No.	Tag	Data								V
0										0
1										0
2										0
3										0
4										0
5	0000111	🟩								~~0~~ 1
:		:								
13										0
14										0
15										0

Address 3AB referenced for the first time. Entire block is brought into cache block 5.

Revisiting Hit and Miss

- Hit means that we actually found data in the cache.
  - A hit occurs when valid bit = 1 *AND* tag in the cache matches the tag field of the address
- Miss
  - If both conditions don’t hold then we did not find the data in cache.
  - On a miss, the data is brought from main memory into the cache, and the valid bit is set.

Cache Design Issues

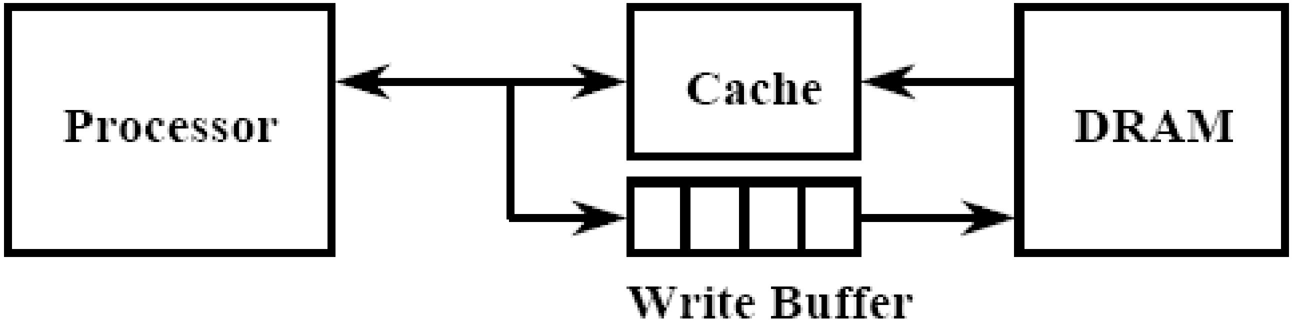
- Where can a block be placed in a cache? Mapping Scheme
- How is a block found if it is in a cache? Block Identification using Mapping Scheme
- Which block should be evicted on a cache miss? Replacement Algorithm
- What happens on write hit? What happens on write miss? Write Policy √

Write Policies (1)

- Write Hit
  - Write Through
    - The cache location and the main memory location are updated simultaneously.
    - Advantage
      - Keeps cache main memory consistent at the same time.
    - Disadvantages
      - All writes require main memory access (bus transaction).
      - Slows down the system - If the there is another read request for main memory due to miss in cache, the read request has to wait until the earlier write was serviced.

Write Policies (2)

- Write Hit (ctd.)
  - Write Through (ctd.)
    - How can we write to cache and memory at the same time? Isn’t memory too slow for this?
    - Solution: Write Buffer
      - Processor: write data into the cache and the write buffer
      - Memory controller: write contents of the buffer to memory.



Write Policies (3)

- Write Hit (ctd.)
  - Write Back
    - Update only the cache location and to mark it as updated with an associated flag bit (dirty or modified bit).
    - When the block containing the marked word is to be removed from the cache, update the main memory location of the word.
    - Advantages
      - Faster than write-through, time is not spent accessing main memory.
      - Writes to multiple words within a block require only one write to the main-memory.

Write Policies (4)

- Write Hit (ctd.)
  - □ Write Back (ctd.)
    - Disadvantages
      - □ Portions of main memory are invalid, and hence accesses by I/O modules can be allowed only through the cache.
      - □ Need extra bit in cache to indicate which block has been modified. 0 – Not Dirty, 1 – Dirty (modified). Adds to size of the cache.

Block No.	Tag	Data								V	D
0										0	0
1	xxxxx									1	0
2										0	0
3	xxxxx	🟥	🟥	🟥						1	1

🟥 *Dirty Words within one block*

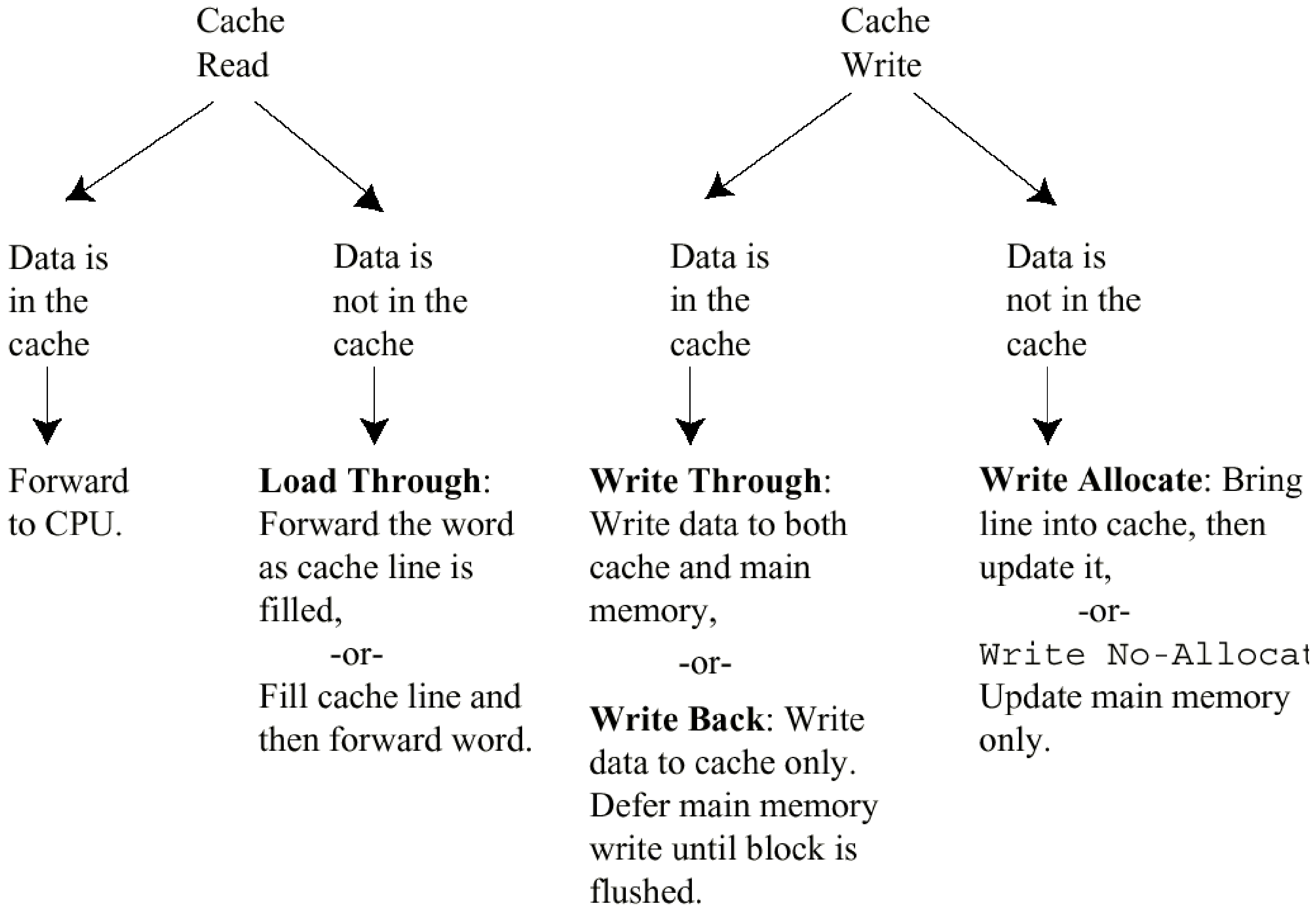
D – Dirty Bit   V- Valid Bit

Write Policies (5)

- Write Miss
  - No-Write Allocate
    - Write-through cache: The information is written directly into the main memory and not loaded into the cache.
  - Write Allocate
    - Write-back cache: 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.

Read & Write Policies

- Cache Read and Write Policies



Cache Read

Data is in the cache

Forward to CPU.

Data is not in the cache

Load Through: Forward the word as cache line is filled,

-or-

Fill cache line and then forward word.

Cache Write

Data is in the cache

Write Through: Write data to both cache and main memory,

-or-

Write Back: Write data to cache only. Defer main memory write until block is flushed.

Data is not in the cache

Write Allocate: Bring line into cache, then update it,

-or-

Write No-Allocat

Update main memory only.

Capacity of Cache （补充）

- The capacity of cache is simply the amount of data that can be stored in the cache, so a cache with a capacity of 32KB can store 32 kilobytes of data.
- But, such a cache require more than 32KB of memory to implement, because the storage in the tag, valid bit and dirty bit are not included in the capacity.

Summary

- 知识点: Cache Principle & Policy
  - Purpose of Cache
  - Organization of Cache/Main-Memory System
  - Cache Read Operation
    - Load Through/Early Restart
  - Valid Bit
  - Hit and Miss
  - Replacement Policy
  - Write Policy
    - Dirty Bit
  - Cache Capacity

Exercise (1)

- 1. The effectiveness of the cache mechanism is base on a property of computer programs called (   ).
  - □ A. parallelism
  - □ B. locality of reference
  - □ C. make the common case fast
  - □ D. forwarding

Exercise (2)

- 2. Which of the following manages the transfer of data between the cache and main memory?
  - A. compiler
  - B. registry
  - C. operating system
  - D. hardware

Exercise (3)

- 3. If a cache has 64-byte cache lines, it takes (   ) cycles to fetch a cache line if the main memory takes 20 cycles to respond to each memory request and returns 2 bytes of data in response to each request.
  - □ A. 128
  - □ B. 320
  - □ C. 640
  - □ D. 256

Contents of this lecture

- 8.6 Cache Memories
  - Cache Principle & Policy
  - Mapping Schemes
  - Example of Mapping Technique
  - Solved Problems Example 8.3 (P324)

Cache Design Issues

- Where can a block be placed in a cache? Mapping Scheme √
- How is a block found if it is in a cache? Block Identification using Mapping Scheme √
- Which block should be evicted on a cache miss? Replacement Algorithm
- What happens on write hit? What happens on write miss? Write Policy

Mapping Scheme (1)

- If the CPU generates an address for a particular *word* in main memory, and that data happens to be in the cache, the same main memory address cannot be used to access the cache.
- Hence a *mapping scheme* is required that converts the generated main memory address into a cache location.
- Mapping scheme also determines where the block will placed when it originally copied into the cache.

Mapping Scheme (2)

- Three Schemes
  - Direct Mapping
  - Associative Mapping
  - Set Associative Mapping
- Assume that
  - Cache line L_i $i = 0, 1, \ldots, m - 1$ $m = 2^r$
  - Memory block B_j $j = 0, 1, \ldots, n - 1$ $n = 2^s$
  - Each line or block consists of $k = 2^w$ consecutive words

Direct Mapping (1)

- Each block of main memory maps to only one cache line
  - i.e. if a block is in cache, it must be in one specific place.
- i = j modulo m, where
  - i = cache line number
  - j = main memory block number
  - m = number of lines in the cache

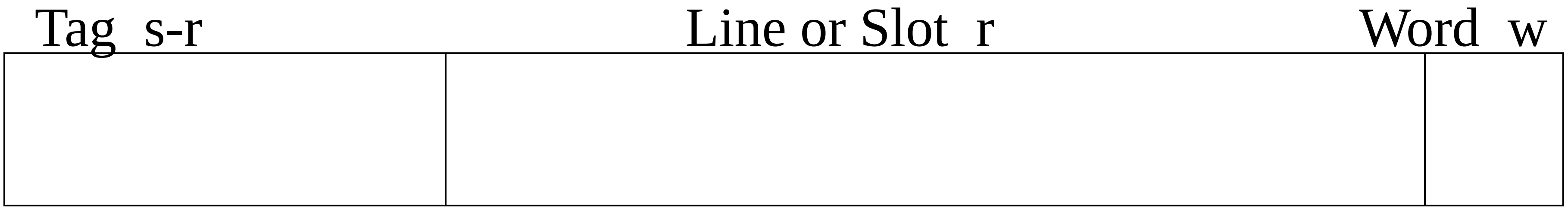
Direct Mapping (2)

- Cache Line Table

Cache line	Main Memory blocks assigned
0	0, m, 2m, …, 2^s- m
1	1, m + 1, 2m +1, …, 2^s- m+1
…	…
m – 1	m-1, 2m-1, 3m-1, …, 2^s-1

Direct Mapping (3)

- Main Memory Address Structure
  - Least Significant w bits identify unique word.
  - Most Significant s bits specify one memory block.
  - The MSBs are split into a cache line field r and a tag of s-r (most significant).



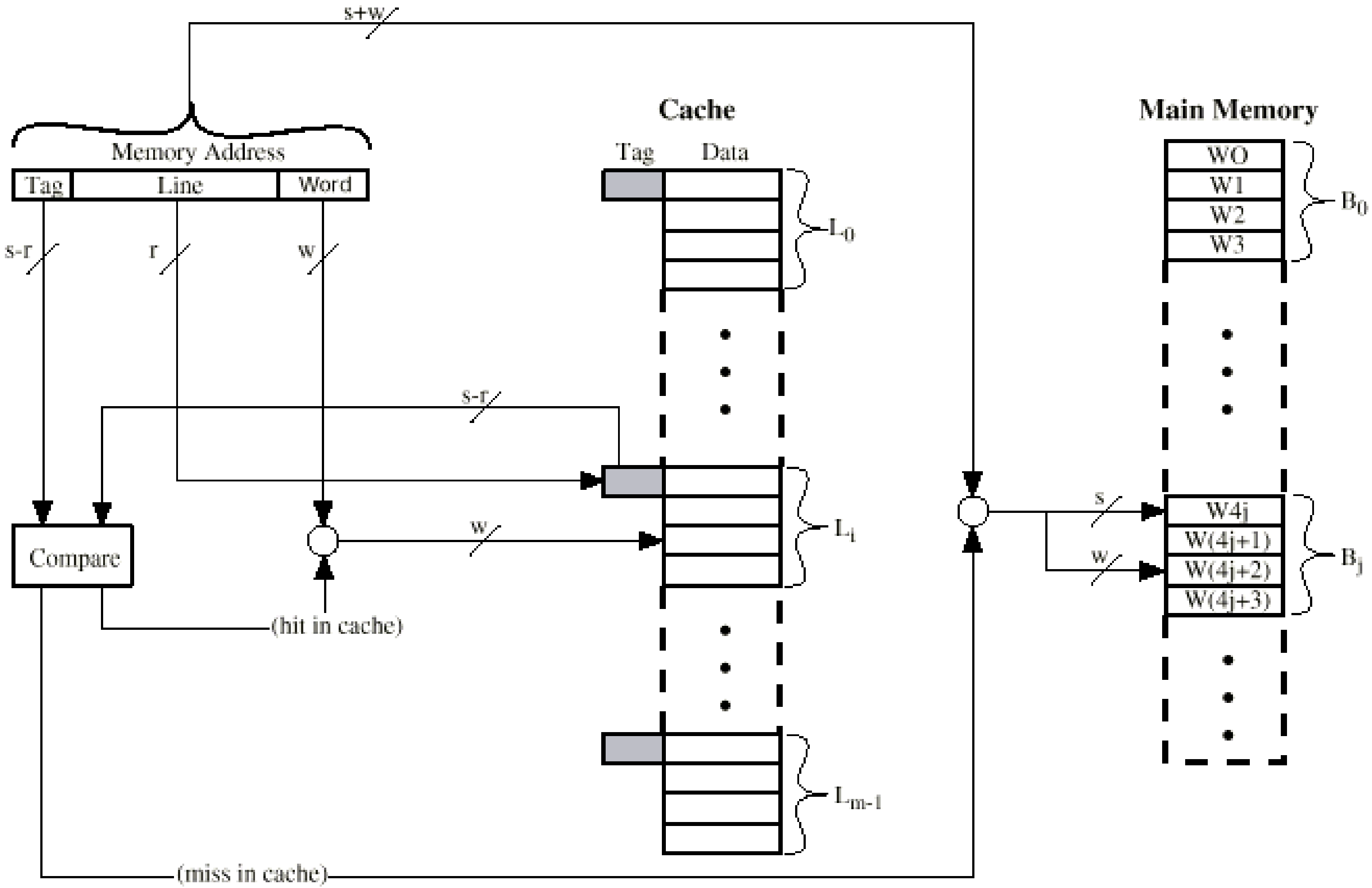
Tag s-r

Line or Slot r

Word w

Direct Mapping (4)

- Access Cache



Direct Mapping (5)

- Example: Consider a cache consisting of 128 lines of 16 words each. The main memory has 64K words. Assume that :
  - □ The main memory is addressable by a 16-bit address.
  - □ Consecutive addresses refer to consecutive words. Thus,
    - Total Main Memory Blocks =64K/16= 4K
    - i = j modulo 128

Direct Mapping (6)

- Example (ctd.)

Cache Line	Main Memory Blocks Assigned
0	0, 128, 256, …, 3968
1	1, 129, 257, …, 3969
…	…
127	127, 255, 383, …, 4095

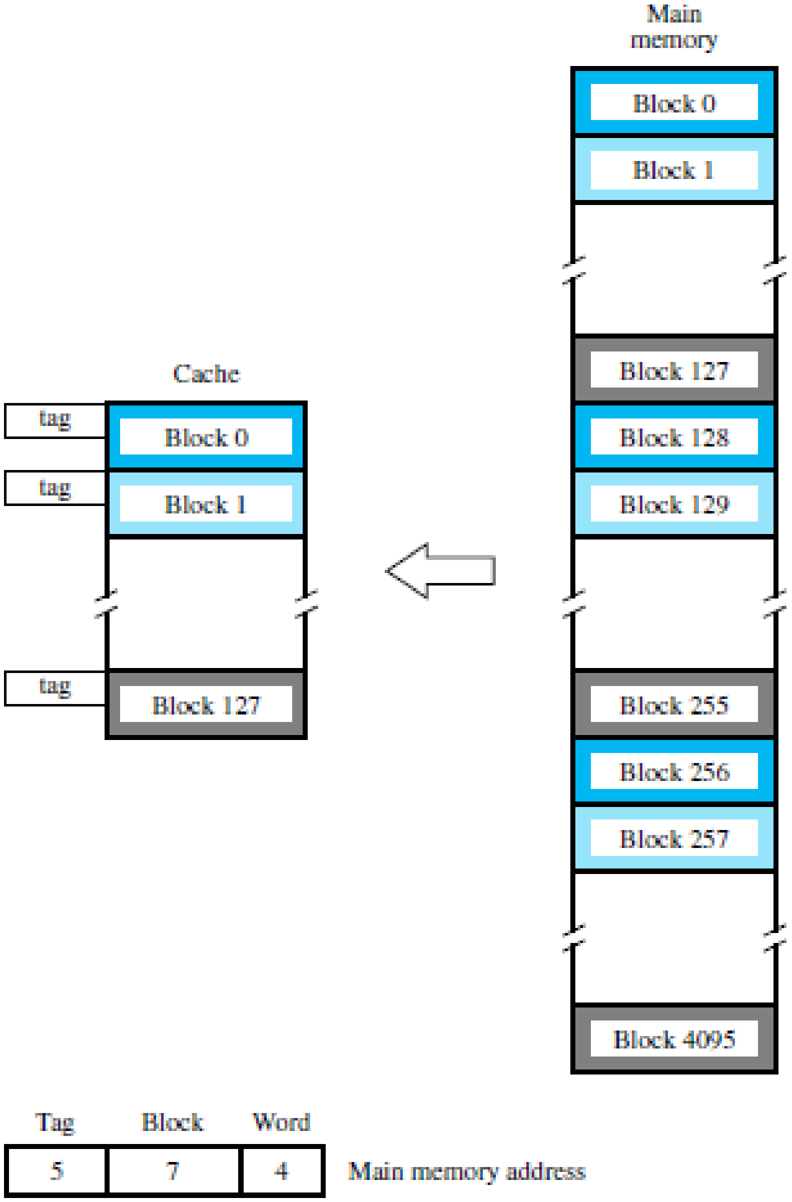


Figure 8.16 Direct-mapped cache.

Direct Mapping (7)

- Main Memory Address Conversion To Cache Location
  - Address conversion is done by giving special significance to the *bits of the main memory address.*
  - The address is split into distinct groups called *fields.*
  - The group fields are a way to find:
    - Which cache location?
    - Which word in the block?
    - Whether it is the right data are looking for? Some kind of unique identifier.

Direct Mapping (8)

- Example (ctd.)
  - □ How to divide 16-bits main memory address into tag, block and word fields?
    - *word* field = $log_2^{16} = 4\ bits$
    - *block* field = $log_2^{128} = 7\ bits$
    - *tag* field = 16-4-7 =5 bits

Direct Mapping (9)

- Advantages
  - Simple, easy to implement
  - Inexpensive
- Disadvantage
  - Fixed location for given block
    - If a program accesses 2 blocks that map to the same line repeatedly, cache misses are very high.

Associative Mapping (1)

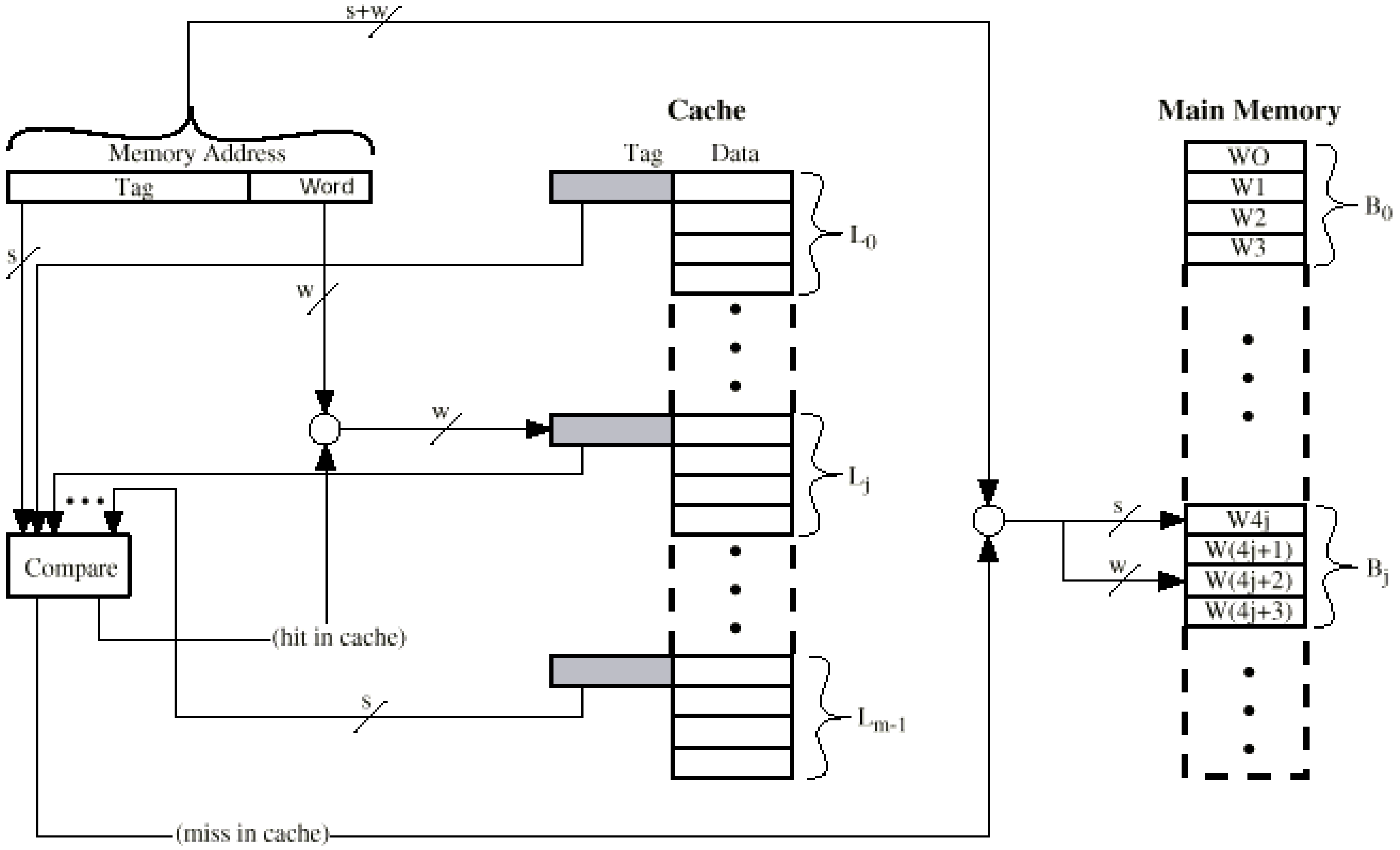
- No mapping functions. A main memory block can load into any line of cache.
- Main Memory Address Structure
  - Memory address is interpreted as tag and word.
  - Tag uniquely identifies block of memory.

Tag s bit	Word w bit

  - Every line’s tag is examined for a match.
  - Cache searching gets expensive.

Associative Mapping (2)

- Access Cache



Associative Mapping (3)

- Example: Cache 128 lines, Main memory 4096 blocks

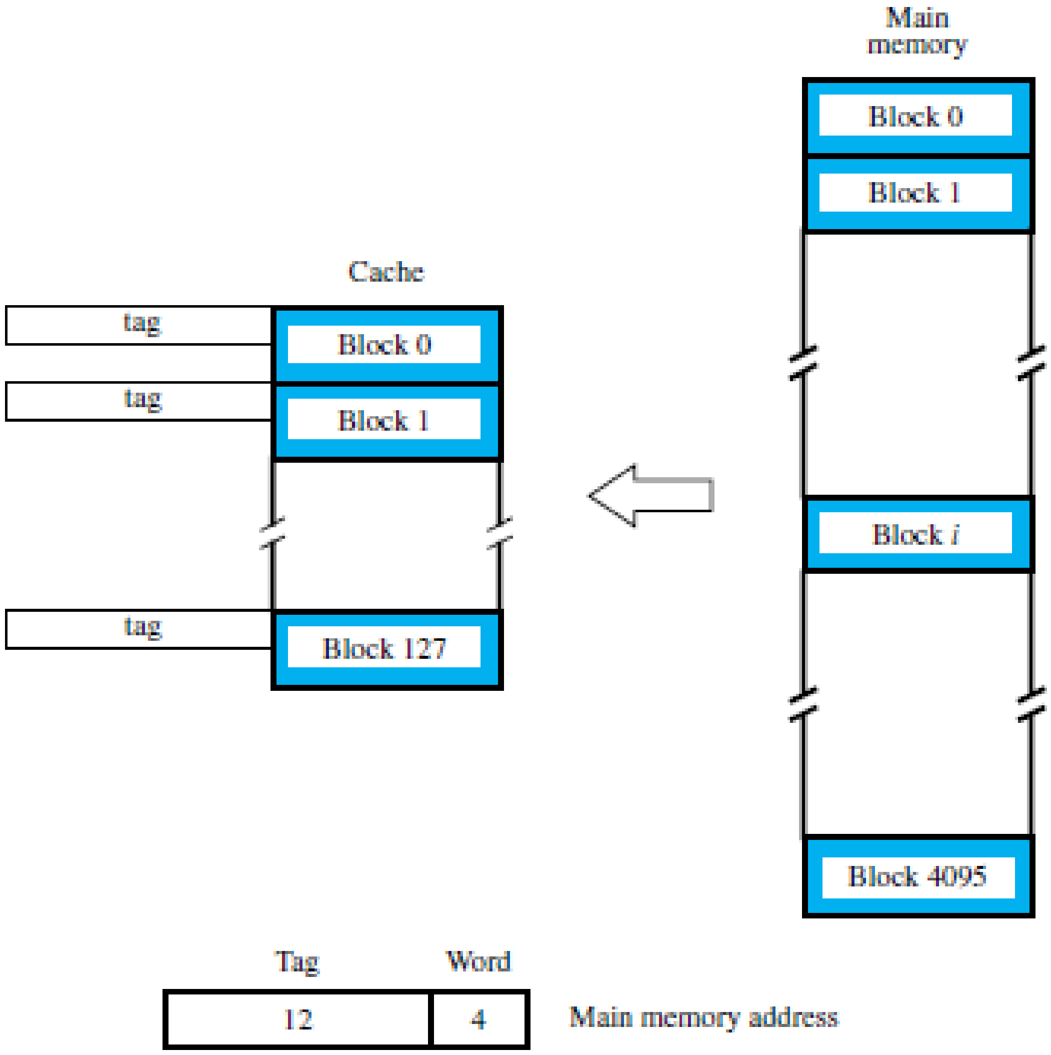


Figure 8.17 Associative-mapped cache.

Associative Mapping (4)

- Advantage
  - It gives complete freedom in choosing the cache location in which to place the memory blocks.
- Disadvantage
  - Complex circuitry required to examine the tags of all cache blocks in parallel.

Set Associative Mapping (1)

- A combination of the direct- and associative-mapping techniques.
- Cache is divided into a number of sets.
- Each set contains a number of lines.
- A given block maps to any line in a given set.
  - e.g. Block Bj can be in any line of set i
  - e.g. 2 lines per set
    - 2 way set associative mapping.
    - A given block can be in one of 2 lines in only one set.

Set Associative Mapping (2)

- The cache is divided into v ($=2^d$) sets, each of which consists of k lines. (k-way set associative mapping)
  - $m = v \times k$
  - i = j modulo v, where
    - i = cache set number
    - j = main memory block number
    - v = number of sets in the cache
  - Special Cases
    - v = m, k =1 Direct mapping
    - v = 1, k = m Fully Associative mapping
    - v = m/2, k = 2 2-way Set associative mapping
    - v = m/4, k = 4 4-way Set associative mapping

Set Associative Mapping (3)

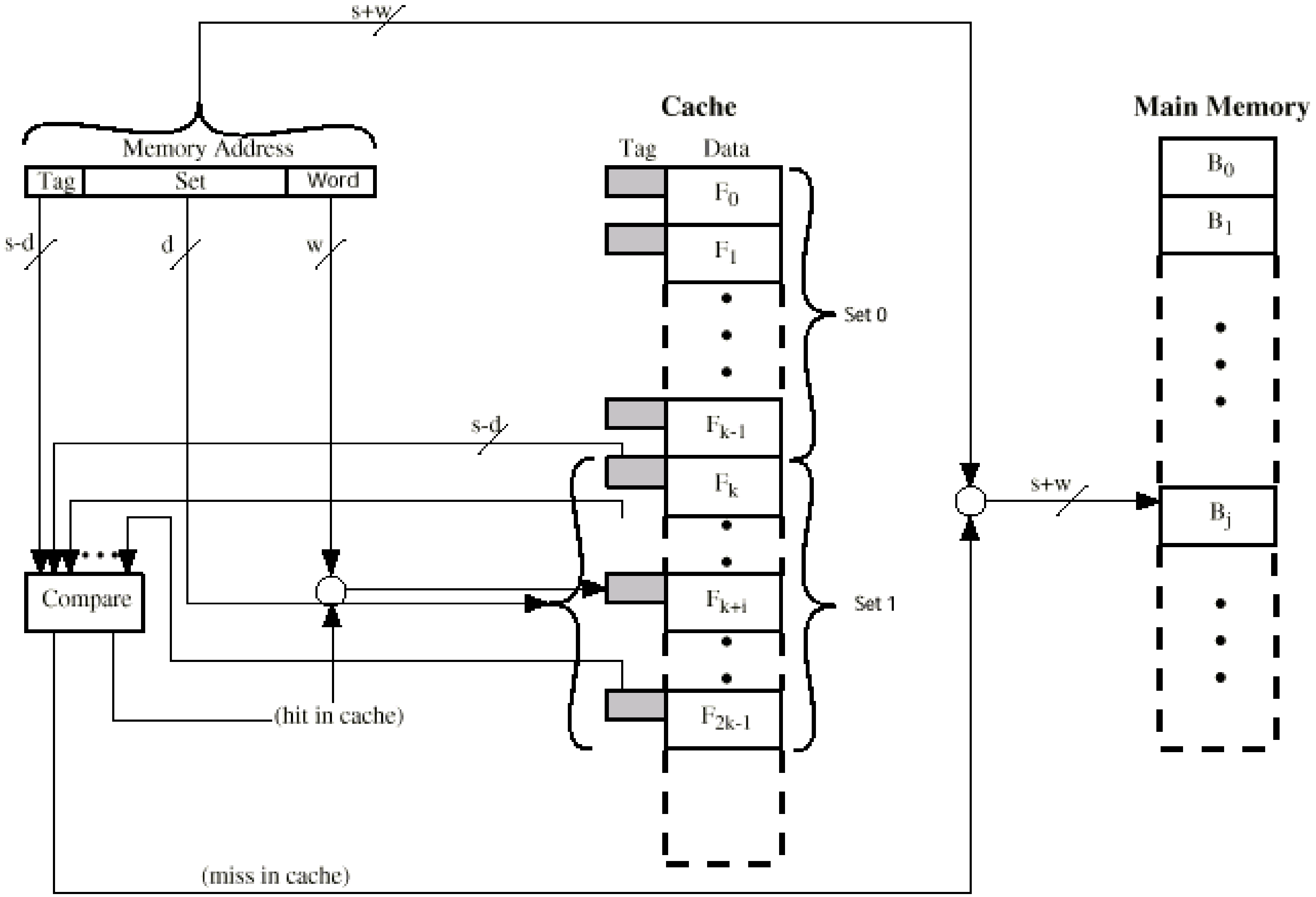
- Main Memory Address Structure

Tag (s-d) bit	Set d bit	Word w bit

- □ Set Field
  - Determine which set of the cache might contain the desired block.

Set Associative Mapping (4)

- Access Cache



Set Associative Mapping (5)

- Example
  - □ 2-way set associative
  - □ i = j modulo 64

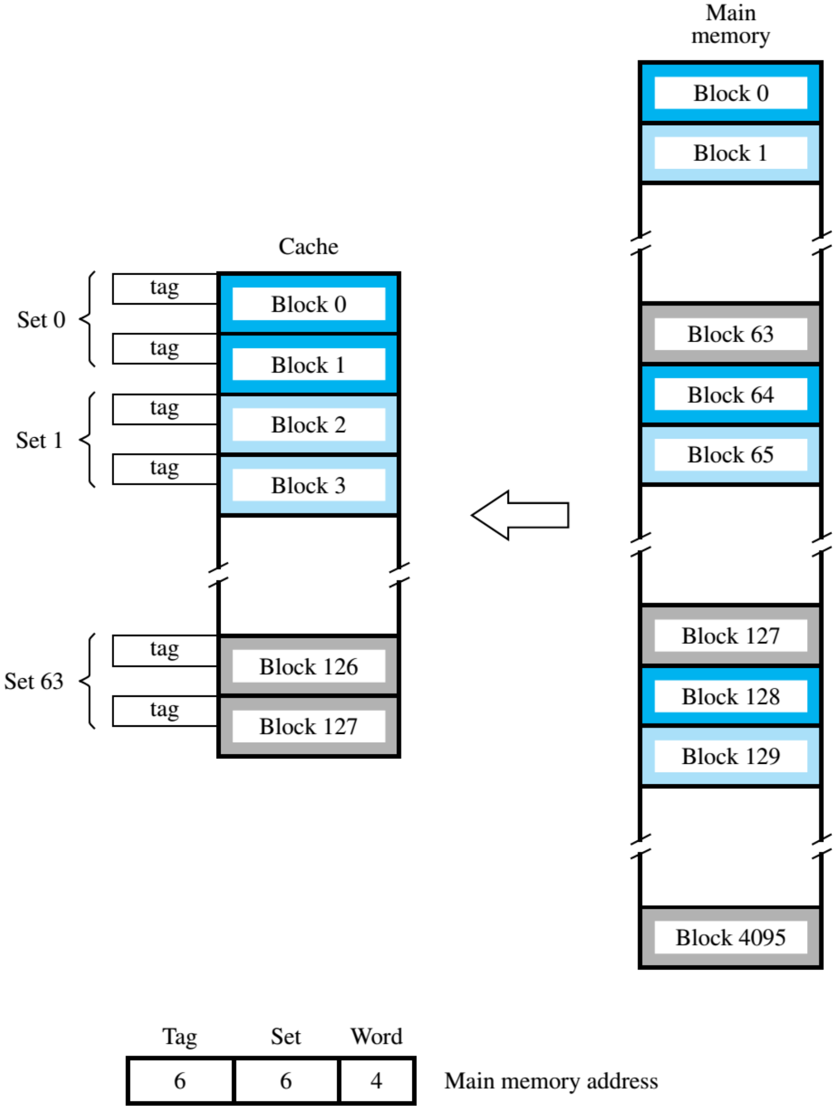


Figure 8.18 Set-associative-mapped cache with two blocks per set.

Set Associative Mapping (6)

- Example (ctd.)
  - How to divide 16-bits main memory address into tag, set and word fields?
    - *word* field = $log_2^{16} = 4\ bits$
    - *set* field = $log_2^{64} = 6\ bits$
    - *tag* field = 16-4-6 =6 bits

Set Associative Mapping (7)

- Advantages
  - The contention problem of the direct method is eased by having a few choices for block replacement.
  - The hardware cost is reduced by decreasing the size of the associative search.
- Disadvantages
  - Tags of each block in a set need to be matched (in parallel) to figure out whether the data is present in cache. Need k comparators.
  - Although, the hardware cost for matching is less than fully associative (need n comparators, where n = #blocks), but it is more than direct mapped (need only one comparator)

Cache Design Issues

- Where can a block be placed in a cache? Mapping Scheme
- How is a block found if it is in a cache? Block Identification
- Which block should be evicted on a cache miss? Replacement Algorithm √
- What happens on write hit? What happens on write miss? Write Policy

Replacement Algorithms (1)

- Direct Mapped Cache
  - No choice
  - Each main memory block only maps to one cache line
  - Replace that cache line

Replacement Algorithms (2)

- Associative and Set Associative Mapped Cache
  - Hardware implemented algorithm (speed)
  - LRU (Least Recently Used)
    - Replace block not accessed for longest time.
  - FIFO (First In First Out)
    - Push block onto queue when accessed.
    - Choose block to replace by popping queue.
  - Random
    - Replace block chosen at random.

Replacement Algorithms (3)

- LRU or Random?
  - □ Studies have shown that LRU replacement generally gives slightly higher hit rates than random replacement, but that the differences are very small for caches of reasonable size.
  - □ However, LRU replacement is relatively complex to implement.

Summary (1)

- 知识点: Cache Mapping Schemes
  - □ Direct Mapping
  - □ Associative Mapping
  - □ Set Associative Mapping
- 掌握程度
  - □ 掌握Direct Mapping的映射函数，使用主存地址访问cache的方法，熟练地将主存地址划分成fields，此种映射方式的优缺点。
  - □ 掌握Associative Mapping的映射方法，使用主存地址访问cache的方法，熟练地将主存地址划分成fields，此种映射方式的优缺点。

Summary (2)

- 掌握程度（续）
  - 掌握Set Associative Mapping的映射函数，使用主存地址访问cache的方法，熟练地将主存地址划分成fields，此种映射方式的优缺点。

Contents of this lecture

- 8.6 Cache Memories
  - Cache Principle & Policy
  - Mapping Schemes
  - Example of Mapping Technique
  - Solved Problems Example 8.3 (P324)

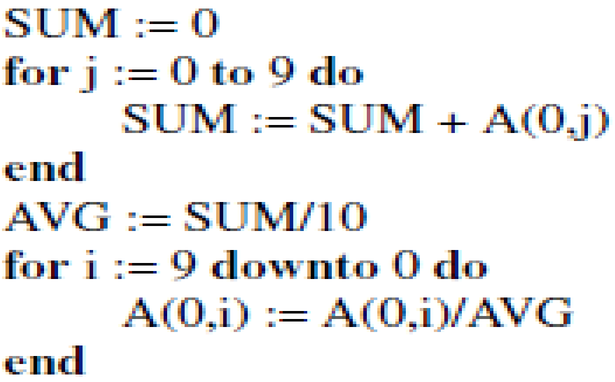
Example of Mapping Technique (1)

- Assume that
  - The processor has separate instruction and data caches.
  - The data cache has space for only 8 blocks of data.
  - Each block consists of only one 16-bit word of data and the memory is word-addressable with 16-bit addresses.
  - The LRU replacement algorithm is used for block replacement in the cache.
- Examine the changes in the data cache entries caused by running the following applications:
  - A 4×10 array of numbers, each occupying one word, is stored in memory locations 7A00 through 7A27.
  - The elements of this array, A, are stored in *column order*.

Example of Mapping Technique (2)

- Examine the changes in the data cache entries caused by running the following applications: (ctd.)
  - The application normalizes the elements of the first row of A with respect to the average value of the elements in the row. That is ,

$$A(0,i) \leftarrow \frac{A(0,i)}{(\sum_{j=0}^{9} A(0,j))/10} \qquad \text{for } i=0,1,\ldots,9$$

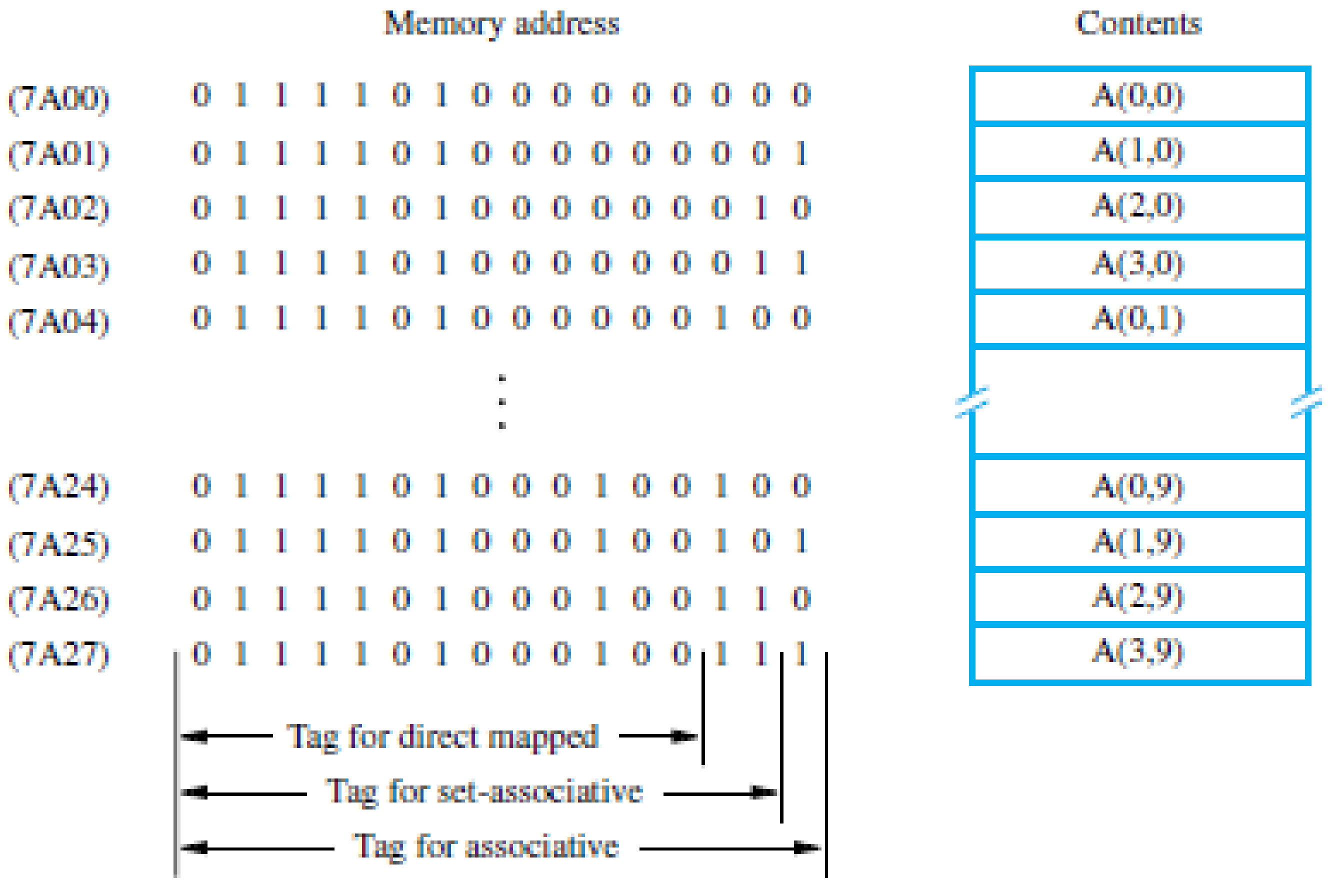


```
SUM := 0
for j := 0 to 9 do
        SUM := SUM + A(0,j)
end
AVG := SUM/10
for i := 9 downto 0 do
        A(0,i) := A(0,i)/AVG
end
```

Figure 8.20 Task for example in Section 8.6.3.

Example of Mapping Technique (3)

- An Array Stored in the Main Memory



	Memory address	Contents
(7A00)	0 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0	A(0,0)
(7A01)	0 1 1 1 1 0 1 0 0 0 0 0 0 0 0 1	A(1,0)
(7A02)	0 1 1 1 1 0 1 0 0 0 0 0 0 0 1 0	A(2,0)
(7A03)	0 1 1 1 1 0 1 0 0 0 0 0 0 0 1 1	A(3,0)
(7A04)	0 1 1 1 1 0 1 0 0 0 0 0 0 1 0 0	A(0,1)
	⋮	
(7A24)	0 1 1 1 1 0 1 0 0 0 1 0 0 1 0 0	A(0,9)
(7A25)	0 1 1 1 1 0 1 0 0 0 1 0 0 1 0 1	A(1,9)
(7A26)	0 1 1 1 1 0 1 0 0 0 1 0 0 1 1 0	A(2,9)
(7A27)	0 1 1 1 1 0 1 0 0 0 1 0 0 1 1 1	A(3,9)

Figure 8.19 An array stored in the main memory.

Example of Mapping Technique (4)

- Memory Address of the 1st Row

	Memory Address	Contents
7A00	0 1 1 1 1 0 1 0 0 0 0 0 0 0 0 0	A(0,0)
7A04	0 1 1 1 1 0 1 0 0 0 0 0 0 1 0 0	A(0,1)
7A08	0 1 1 1 1 0 1 0 0 0 0 0 1 0 0 0	A(0,2)
7A0C	0 1 1 1 1 0 1 0 0 0 0 0 1 1 0 0	A(0,3)
7A10	0 1 1 1 1 0 1 0 0 0 0 1 0 0 0 0	A(0,4)
7A14	0 1 1 1 1 0 1 0 0 0 0 1 0 1 0 0	A(0,5)
7A18	0 1 1 1 1 0 1 0 0 0 0 1 1 0 0 0	A(0,6)
7A1C	0 1 1 1 1 0 1 0 0 0 0 1 1 1 0 0	A(0,7)
7A20	0 1 1 1 1 0 1 0 0 0 1 0 0 0 0 0	A(0,8)
7A24	0 1 1 1 1 0 1 0 0 0 1 0 0 1 0 0	A(0,9)

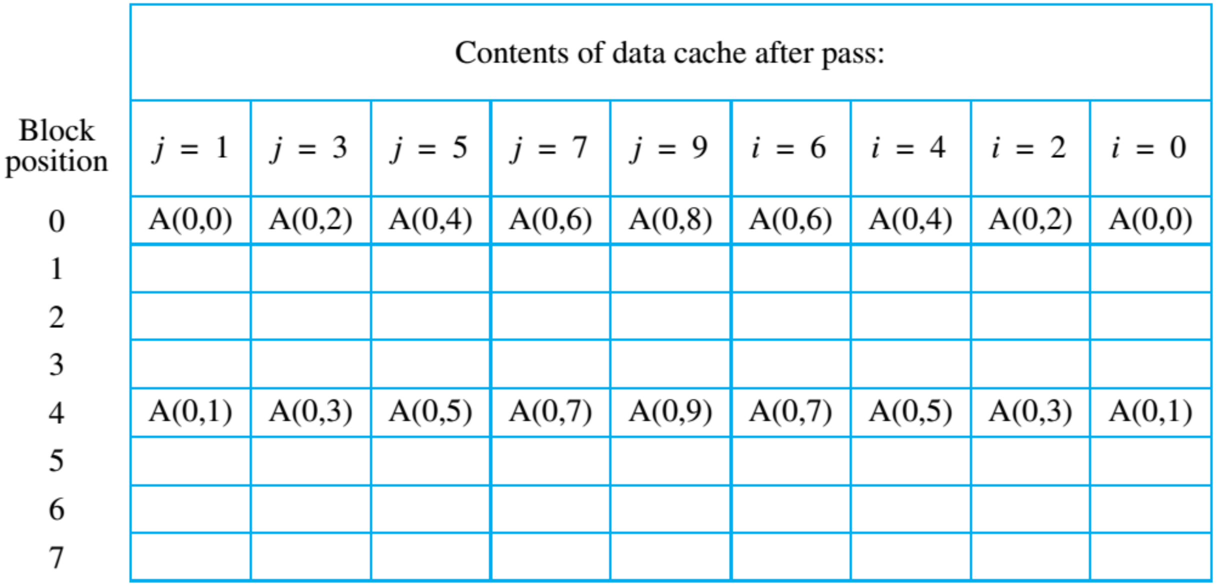
Example of Mapping Technique (5)

- Solution
  - □ Direct Mapped Cache

Tag	Block
13	3

Main memory address

Eight elements are replaced while the second loop is executed.

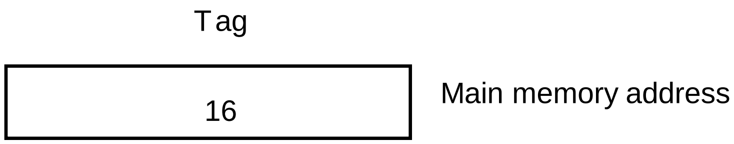


Block position	Contents of data cache after pass:								
	$j = 1$	$j = 3$	$j = 5$	$j = 7$	$j = 9$	$i = 6$	$i = 4$	$i = 2$	$i = 0$
0	A(0,0)	A(0,2)	A(0,4)	A(0,6)	A(0,8)	A(0,6)	A(0,4)	A(0,2)	A(0,0)
1									
2									
3									
4	A(0,1)	A(0,3)	A(0,5)	A(0,7)	A(0,9)	A(0,7)	A(0,5)	A(0,3)	A(0,1)
5									
6									
7									

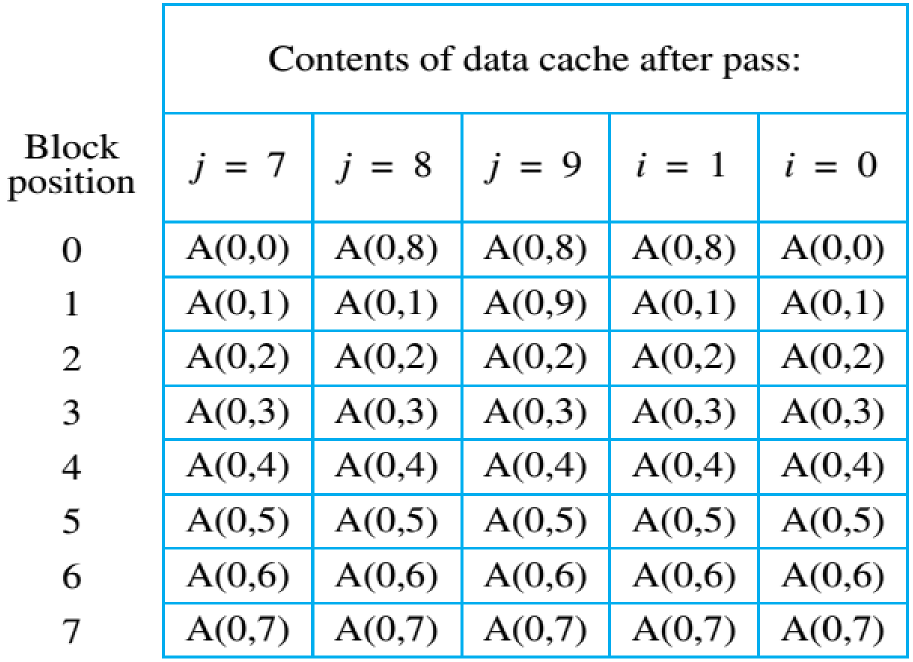
Figure 8.21 Contents of a direct-mapped data cache.

Example of Mapping Technique (6)

- Solution (ctd.)
  - Associative Mapped Cache



Two elements are replaced while the second loop is executed.



	Contents of data cache after pass:				
Block position	$j = 7$	$j = 8$	$j = 9$	$i = 1$	$i = 0$
0	A(0,0)	A(0,8)	A(0,8)	A(0,8)	A(0,0)
1	A(0,1)	A(0,1)	A(0,9)	A(0,1)	A(0,1)
2	A(0,2)	A(0,2)	A(0,2)	A(0,2)	A(0,2)
3	A(0,3)	A(0,3)	A(0,3)	A(0,3)	A(0,3)
4	A(0,4)	A(0,4)	A(0,4)	A(0,4)	A(0,4)
5	A(0,5)	A(0,5)	A(0,5)	A(0,5)	A(0,5)
6	A(0,6)	A(0,6)	A(0,6)	A(0,6)	A(0,6)
7	A(0,7)	A(0,7)	A(0,7)	A(0,7)	A(0,7)

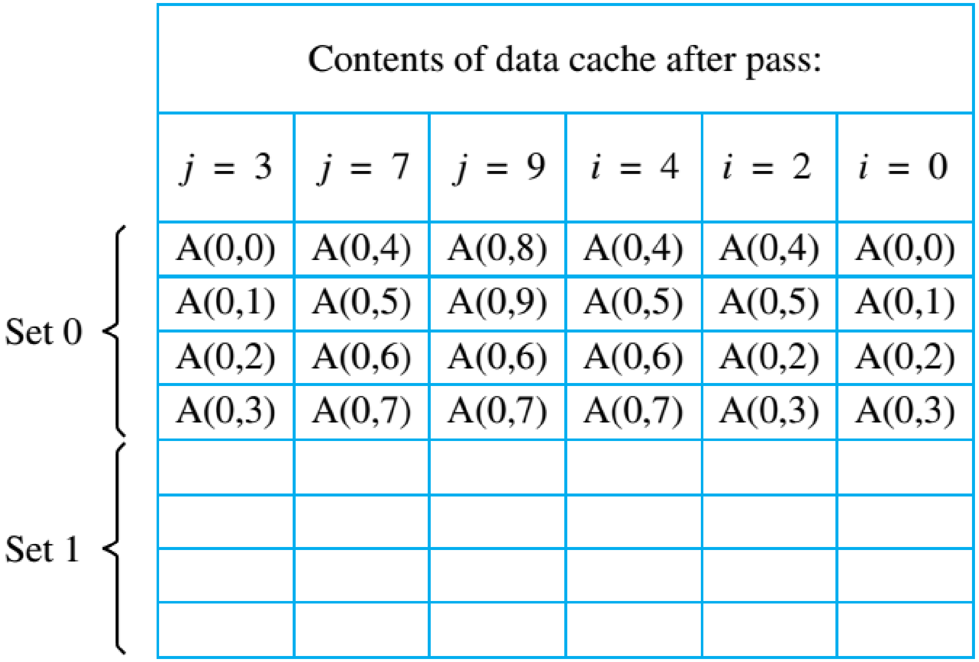
Figure 8.22 Contents of an associative-mapped data cache.

Example of Mapping Technique (7)

- Solution (ctd.)
  - Set Associative Mapped Cache

Tag	Set	
15	1	Main memory address

Six elements must be reloaded during execution of the second loop.



	Contents of data cache after pass:					
	$j = 3$	$j = 7$	$j = 9$	$i = 4$	$i = 2$	$i = 0$
Set 0	A(0,0)	A(0,4)	A(0,8)	A(0,4)	A(0,4)	A(0,0)
	A(0,1)	A(0,5)	A(0,9)	A(0,5)	A(0,5)	A(0,1)
	A(0,2)	A(0,6)	A(0,6)	A(0,6)	A(0,2)	A(0,2)
	A(0,3)	A(0,7)	A(0,7)	A(0,7)	A(0,3)	A(0,3)
Set 1						

Figure 8.23 Contents of a set-associative-mapped data cache.

Exercise (1)

- 1. By (   ) mapping technique, a main memory block can be placed into any cache block position.
  - A. direct
  - B. associative
  - C. set associative

Exercise (2)

- 2. If a cache has a capacity of 16KB and a line length of 128 bytes, how many sets does the cache have if it is 2-way, 4-way, or 8-way set associative?
  - □ A. 64, 32, 16
  - □ B. 64, 16, 8
  - □ C. 32, 16, 8
  - □ D. 128, 64, 32

Exercise (3)

- 3. A 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. How many bits are there in a main memory address?
  - A. 21
  - B. 24
  - C. 19
  - D. 32

Exercise (4)

- 4. A 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. How many bits are there in the main memory address of the TAG, SET, and WORD fields?
  - □ A. 2, 5, 12
  - □ B. 12, 2, 5
  - □ C. 4, 8, 7
  - □ D. 8, 4, 7

Exercise (5)

- 5. In cache system, when a block is to be overwritten, it is sensible to overwrite the one that has gone the longest time without being referenced. This technique is called the (   ) replacement algorithm.
  - A. FIFO
  - B. Random
  - C. LFU
  - D. LRU

Exercise (6)

- 6. A memory system uses 24-bit physical address. It has a direct-mapped cache with 64 entries. The block size is 4 words (word size is 4 bytes). Each cache entry has a “valid bit”, a “tag field”, and space for data. The number of bits reserved for the tag field is (   ).
    - A. 14
    - B. 10
    - C. 6
    - D. 12

Exercise (7)

- 7. A memory system uses 24-bit physical address. It has a direct-mapped cache with 64 entries. The block size is 4 words (word size is 4 bytes). Each cache entry has a “valid bit”, a “tag field”, and space for data. The total number of bits in the cache is ( ).
  - □A. $2^6 \times (143)$
  - □B. $2^6 \times (33)$
  - □C. $2^8 \times (145)$
  - □D. $2^{14} \times (33)$

Contents of this lecture

- 8.6 Cache Memories
    - □ Cache Principle & Policy
    - □ Mapping Schemes
    - □ Example of Mapping Technique
    - □ Solved Problems Example 8.3 (P324)

Solved Problems (1)

- Example 8.3 **Problem:** A computer system uses 32-bit memory addresses and it has a main memory consisting of 1Gbytes. It has a 4K-byte cache organized in the block-set-associative manner, with 4 blocks per set and 64 bytes per block.
  - □ (*a*) Calculate the number of bits in each of the Tag, Set, and Word fields of the memory address.
  - □ (*b*) Assume that the cache is initially empty. Suppose that the processor fetches 1088 words of four bytes each from successive word locations starting at location 0. It then repeats this fetch sequence nine more times. If the cache is 10 times faster than the memory, estimate the improvement factor resulting from the use of the cache. Assume that the LRU algorithm is used for block replacement.

Solved Problems (2)

- Example 8.3 **Solution:** Consecutive addresses refer to bytes.
  - □ (*a*) A block has 64 bytes; hence the Word field is 6 bits long.

    With 4 × 64 = 256 bytes in a set, there are 4K/256 = 16 sets, requiring a Set field of 4 bits. This leaves 32 − 4 − 6 = 22 bits for the Tag field.

Solved Problems (3)

- Example 8.3 **Solution: (ctd.)**
  - □ (*b*) The cache has space for 64 blocks.

  The 1088 words constitute 68 blocks, occupying blocks 0 to 67 in the memory.

    - Main memory blocks 0, 16, 32, 48 in cache set 0
    - Main memory blocks 1, 17, 33, 49 in cache set 1
    - Main memory blocks 2, 18, 34, 50 in cache set 2
    - Main memory blocks 3, 19, 35, 51 in cache set 3
    - ……
    - Main memory blocks 15, 31, 47, 63 in cache set 15

  Hence, after blocks 0, 1, 2, . . . , 63 have been read from the memory into the cache on the first pass, the cache is full.

  The next four blocks, numbered 64 to 67, map to sets 0, 1, 2, and 3. Each of them will replace the least recently used cache block in its set, which is block 0. (Memory block0 1,2,3 are replaced by memory block 64,65,66, 67.)

Solved Problems (4)

- Example 8.3 **Solution: (ctd.)**
  - (*b*) After first pass, the contents in cache are

Set 0	Set 1	Set 2	Set 3
(0)64	(1)65	(2)66	(3)67
16	17	18	19
32	33	34	35
48	49	50	51

Set 4		Set 15
4		15
20	……	31
36		47
52		63

Solved Problems (5)

- Example 8.3 **Solution: (ctd.)**
  - □ (*b*) During the second pass, memory block 0 has to be reloaded into set 0 of the cache, since it has been overwritten by block 64. It will be placed in the least recently used block of set 0 at that point, which is block 1. Next, memory blocks 1, 2, and 3 will replace block 1 of sets 1, 2 and 3 in the cache, respectively. (Memory blocks 16, 17, 18, 19 are replaced by memory blocks 0,1,2,3)

Memory blocks 4 to 15 will be found in the cache.

Memory blocks 16 to 19, which were in block location 1 of sets 0 to 3, have now been overwritten, and will be reloaded in block location 2 of these sets.

Memory blocks 20 to 31 will be found in the cache.

Memory blocks 32 to 35, which were in block location 2 of sets 0 to 3, have now been overwritten, and will be reloaded in block location 3 of these sets.

Memory blocks 36 to 47 will be found in the cache.

Memory blocks 48 to 51, which were in block location 3 of sets 0 to 3, have now been overwritten, and will be reloaded in block location 0 of these sets.

Solved Problems (6)

- Example 8.3 **Solution: (ctd.)**
  - □ (*b*) After second pass, the contents in cache are

Set 0	Set 1	Set 2	Set 3
48	49	50	51
64	65	66	67
16	17	18	19
32	33	34	35

Set 4		Set 15
4		15
20	……	31
36		47
52		63

Solved Problems (7)

- Example 8.3 **Solution: (ctd.)**
  - □ (*b*) As execution proceeds, all memory blocks that occupy the first four of the 16 cache sets are always overwritten before they can be used on a succeeding pass. Memory blocks 0, 16, 32, 48, and 64 continually displace each other as they compete for the 4 block positions in cache set 0. The same thing occurs in cache set 1 (memory blocks 1, 17, 33, 49, 65), cache set 2 (memory blocks 2, 18, 34, 50, 66), and cache set 3 (memory blocks 3, 19, 35, 51, 67).

Memory blocks that occupy the last 12 sets (sets 4 through 15) are fetched once on the first pass and remain in the cache for the next 9 passes.

Solved Problems (8)

- Example 8.3 **Solution: (ctd.)**
  - □ (*b*) In summary, on the first pass, all 68 blocks of the loop are fetched from the memory.

On each of the 9 successive passes, 48 blocks are found in sets 4 through 15 of the cache, and the remaining 20 blocks must be fetched from the memory. Let τ be the access time of the cache. Therefore,

$$\text{Improvement factor} = \frac{\text{Time without cache}}{\text{Time with cache}}$$

$$= \frac{10 \times 68 \times 10\tau}{1 \times 68 \times 11\tau + 9(20 \times 11\tau + 48\tau)}$$

$$= 2.15$$

Homework (1)

- P328 8.8, 8.10, 8.11, 8.13
- 补充题

3. A computer system has a cache organized in set associative manner, with 4 blocks per set and 64 words per block. The cache consists of 32 blocks and the main memory consists of 1024 blocks. The main memory is word-addressable.

(1)How many bits are there in main memory address?

(2)Calculate the number of bits in each of the TAG, SET, and WORD fields of the main memory address format.

(3)Assuming that the cache is initially empty. The processor fetches 3072 words sequentially from main memory locations 0, 1, …, 3071. Assume that the LRU algorithm is used for block replacement. Compute the hit rate of this cache.