Chapter No. 04 – Cache Memory (Part – 02)

Lecture – 09

17-04-2018

Topics to Cover

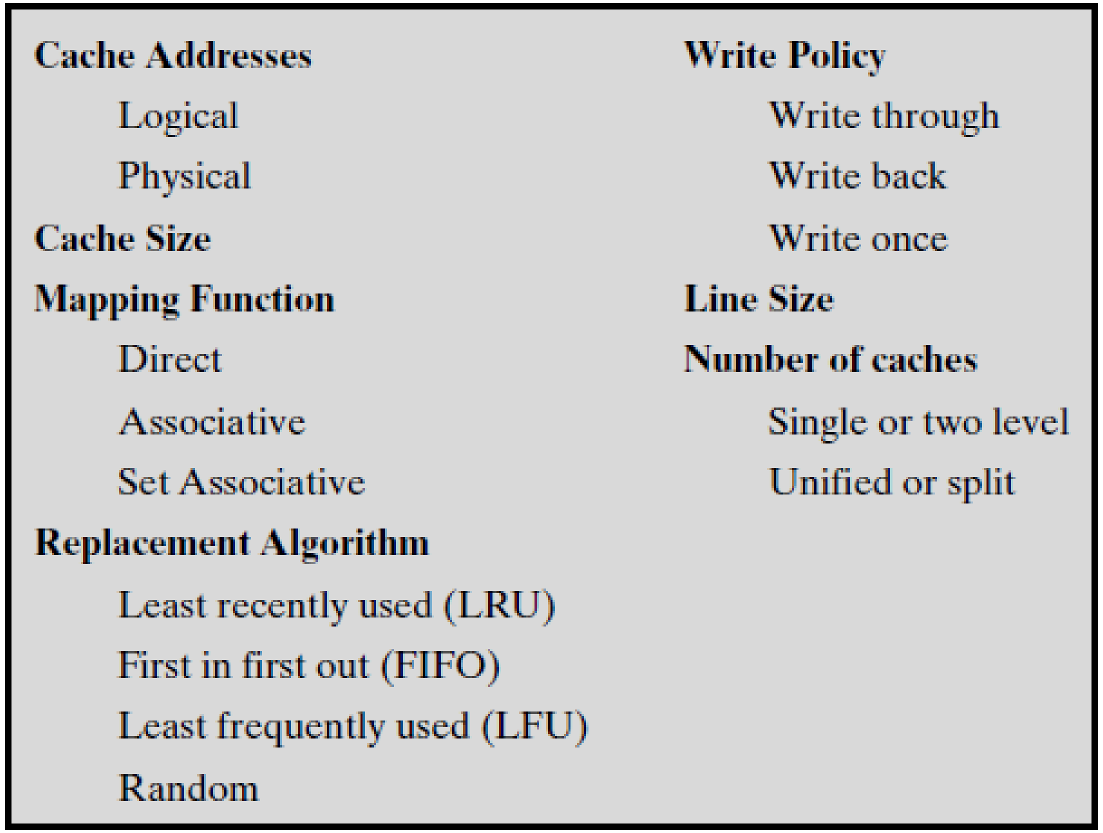
➢4.3 Elements of Cache Design

- Cache Addresses
- Cache Size
- Mapping Function
- Types of Mapping
- Replacement Algorithms
- Write Policy
- Line Size
- Number of Caches

Assignment-3
Due Next Week

4.3 Elements of Cache Design (Table 4.2)

- Although there are a large number of cache implementations, there are a few basic design elements that serve to classify and differentiate cache architectures. Table 4.2 below lists key elements.



Cache Addresses

- Logical
- Physical

Cache Size

Mapping Function

- Direct
- Associative
- Set Associative

Replacement Algorithm

- Least recently used (LRU)
- First in first out (FIFO)
- Least frequently used (LFU)
- Random

Write Policy

- Write through
- Write back
- Write once

Line Size

Number of caches

- Single or two level
- Unified or split

Cache Size

- We would like the size of the cache to be small enough so that the overall average cost per bit is close to that of main memory alone.
- We would like the cache to be large enough (up to a point) so that the overall average access time is close to that of the cache alone.
- It is observed, that the larger the cache, the larger the number of gates (bits) involved in addressing the cache.
- The result is that large caches tend to be slower than the smaller ones
- The available chip and board area also limits cache size.
- The performance of cache is very sensitive to the nature of workload, so it is impossible to arrive at a single ‘optimum’ cache size.

Mapping Function

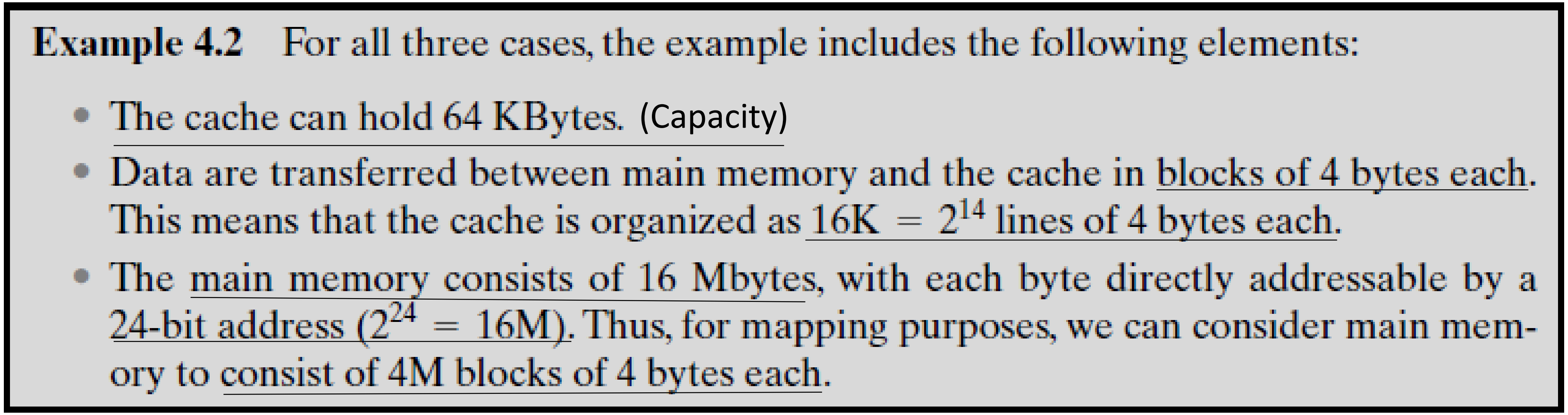
- Why do we need mapping?
- Because there are fewer cache lines than main memory blocks, an algorithm is needed for mapping main memory blocks into cache lines.
- Further, a means is needed for determining which main memory block currently occupies a cache line.
- Three mapping techniques can be used:

1. Direct 2. Associative 3. Set-Associative

- The choice of mapping function dictates how the cache is organized.

Example 4.2

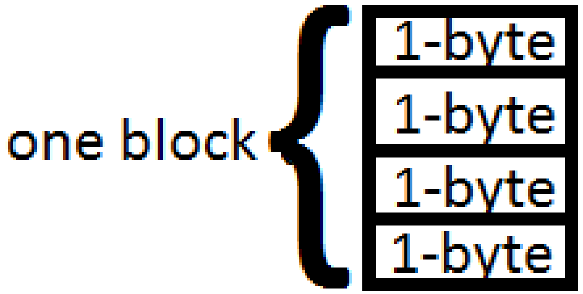
- We will examine each of these three mapping techniques. In each case, we look at the general structure and then a specific example.



Example 4.2 For all three cases, the example includes the following elements:

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

Example Explained (Cache)



one block

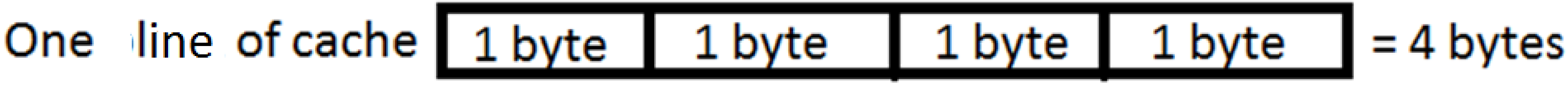
1-byte

1-byte

1-byte

1-byte

- Cache Capacity = 64-K bytes. (64*1024 bytes)
- One cache line = one block = 4 bytes.



One line of cache 1 byte 1 byte 1 byte 1 byte = 4 bytes

- So there are: 16-K blocks of 4 bytes each:

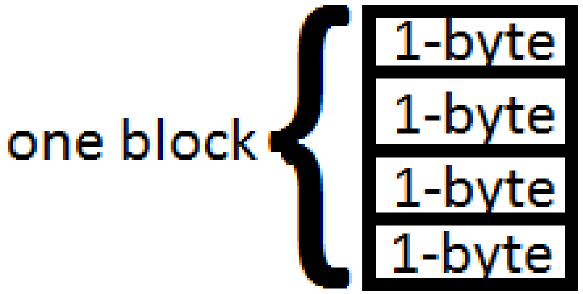
$$\frac{64 \times 1024\ bytes}{4\ bytes} = \text{16-K blocks}$$

- To address cache we need 2^{14} lines of 4 bytes each:

$$\text{16-K blocks} = 2^{4} * 2^{10} = 2^{4+10} = 2^{14}$$

- Thus cache needs a 14-bit address for each line (of 4-bytes).

Example Explained (Memory)


one block

1-byte

1-byte

1-byte

1-byte

- Main-memory capacity = 16-Mbytes (16*1024*1024 bytes)
- Main-memory addresses = 16 * 1024 * 1024 bytes

$$= 2^{4} * 2^{10} * 2^{10}$$

$$= 2^{4+10+10}$$

$$= 2^{24} \text{ (=> 24-bit address, } 2^{24} = 16\text{M)}$$

- For mapping purpose, we have:

$$\frac{16\times1024\times1024\ bytes}{(4\ bytes=1\ block)} = 4 * 1024 *1024 = 4\text{M}$$

- Thus, we have in main-memory = 4M blocks of (4-bytes each)

1. Direct Mapping

- **Direct mapping** maps each block of main memory into only one possible cache line.
- I.E. if a block is in cache, it must be in one specific place.
- The ‘direct mapping’ is expressed as:

$$\boxed{i = j \text{ modulo } m}$$

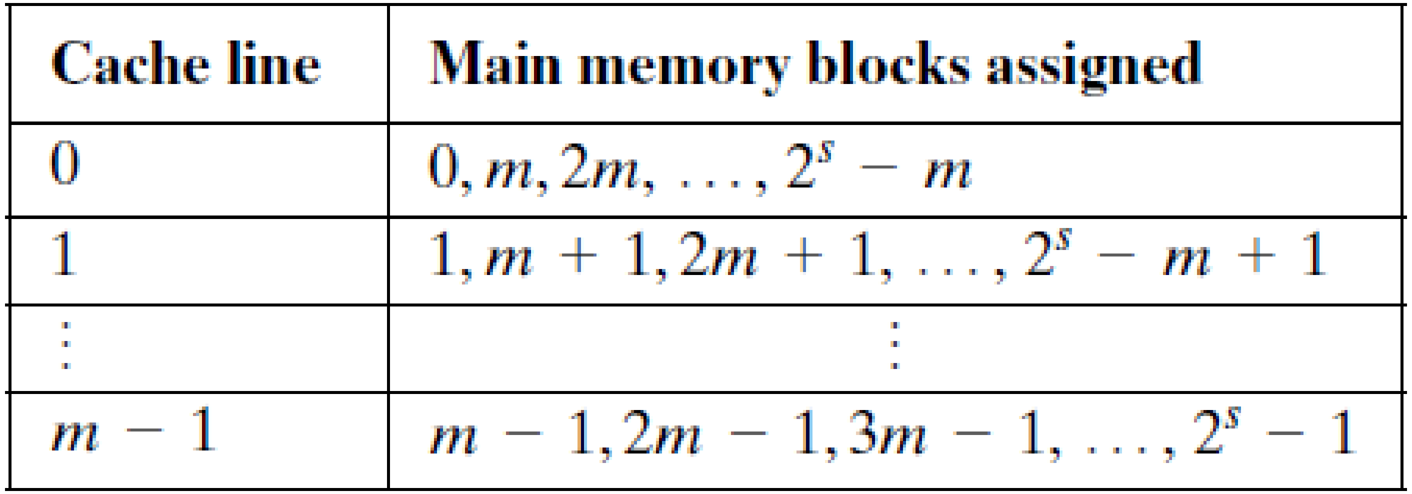
where

Modulus operator(%) returns remainder after integer division.

$$\begin{aligned} i &= \text{cache line number} \\ j &= \text{main memory block number} \\ m &= \text{number of lines in the cache} \end{aligned}$$

Direct Mapping

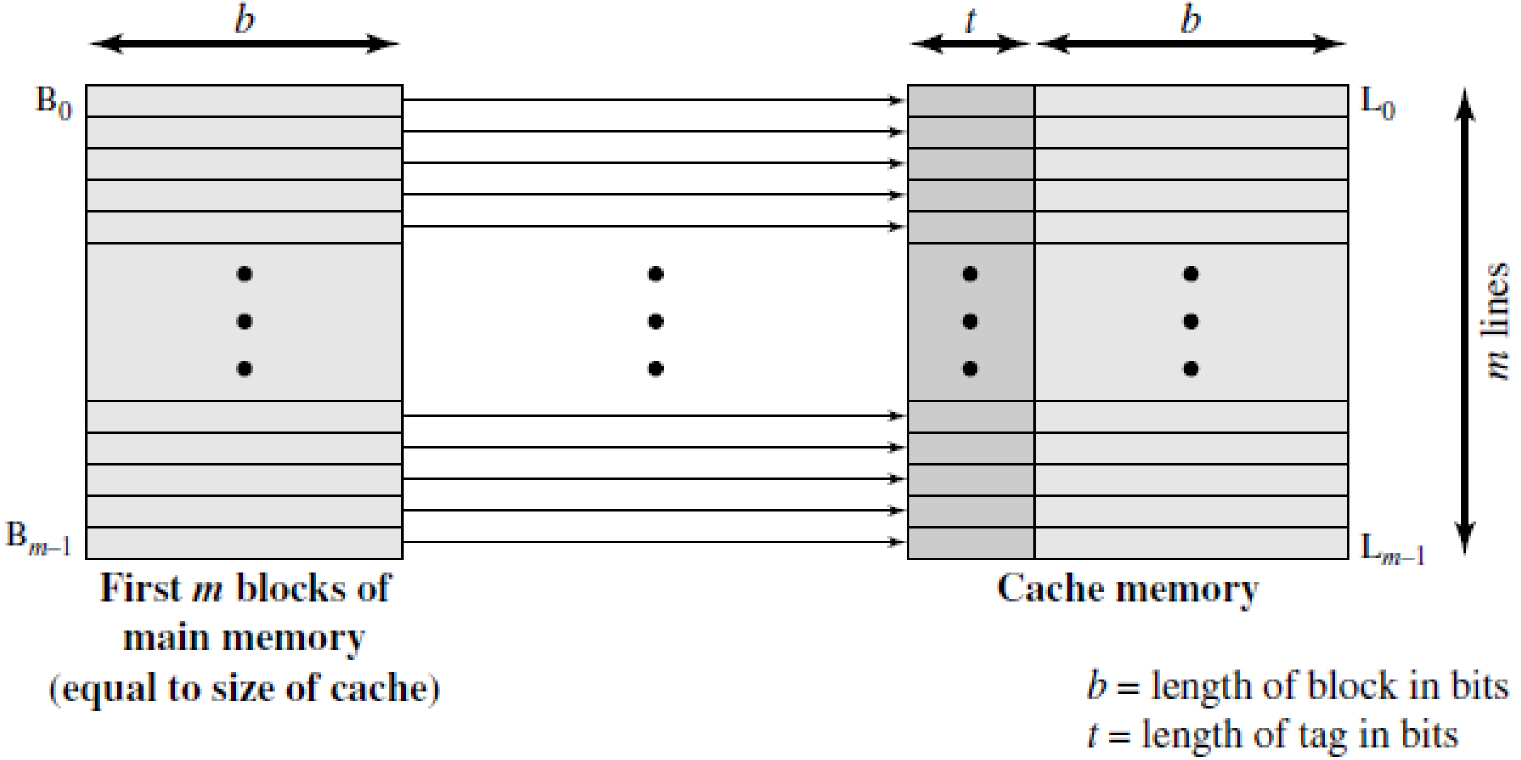
- In ‘Direct Mapping’, the blocks of main-memory are assigned to lines of the cache as follows:



Cache line	Main memory blocks assigned
0	$0, m, 2m, \ldots, 2^s - m$
1	$1, m + 1, 2m + 1, \ldots, 2^s - m + 1$
$\vdots$	$\vdots$
$m - 1$	$m - 1, 2m - 1, 3m - 1, \ldots, 2^s - 1$

- The use of a portion of the address (**r-bits**) as a line number provides a unique mapping of each block of main-memory into the cache.

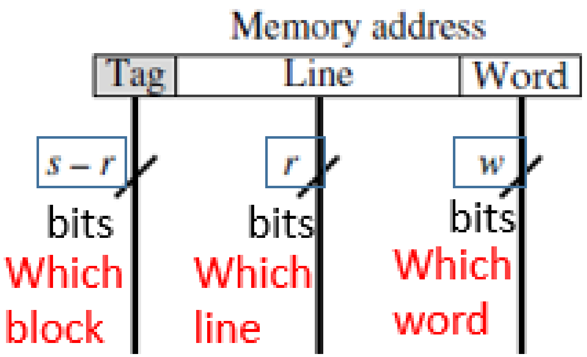
- Figure shows the mapping for the first **m**-blocks of main memory.
- Each block of main memory maps into one unique line of the cache.
- The next **m**-blocks of main memory map into the cache in the same fashion; that is, block $\mathbf{B_m}$ of main memory maps into line $\mathbf{L_0}$ of cache,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



b
t
b
B_0
B_{m-1}
L_0
L_{m-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

(a) Direct mapping

Direct Mapping (Figure Next Slide)



- For purpose of cache access, each main-memory address can be viewed as consisting of three fields.
- The least significant **w**-bits identify a unique **word** or **byte** within a *block* of main-memory. (The address is at the *byte* level)
- The remaining **s**-bits specify one of the **2**^{s}-blocks of main-memory.
- The cache logic interprets these **s**-bits as a **tag** of **s – r** bits (most significant portion) and a **line** field of **r**-bits.
- The 'line field' identifies one of the **m = 2**^{r} lines of the cache.

This square Box contains The answer to <u>Question 3</u>.

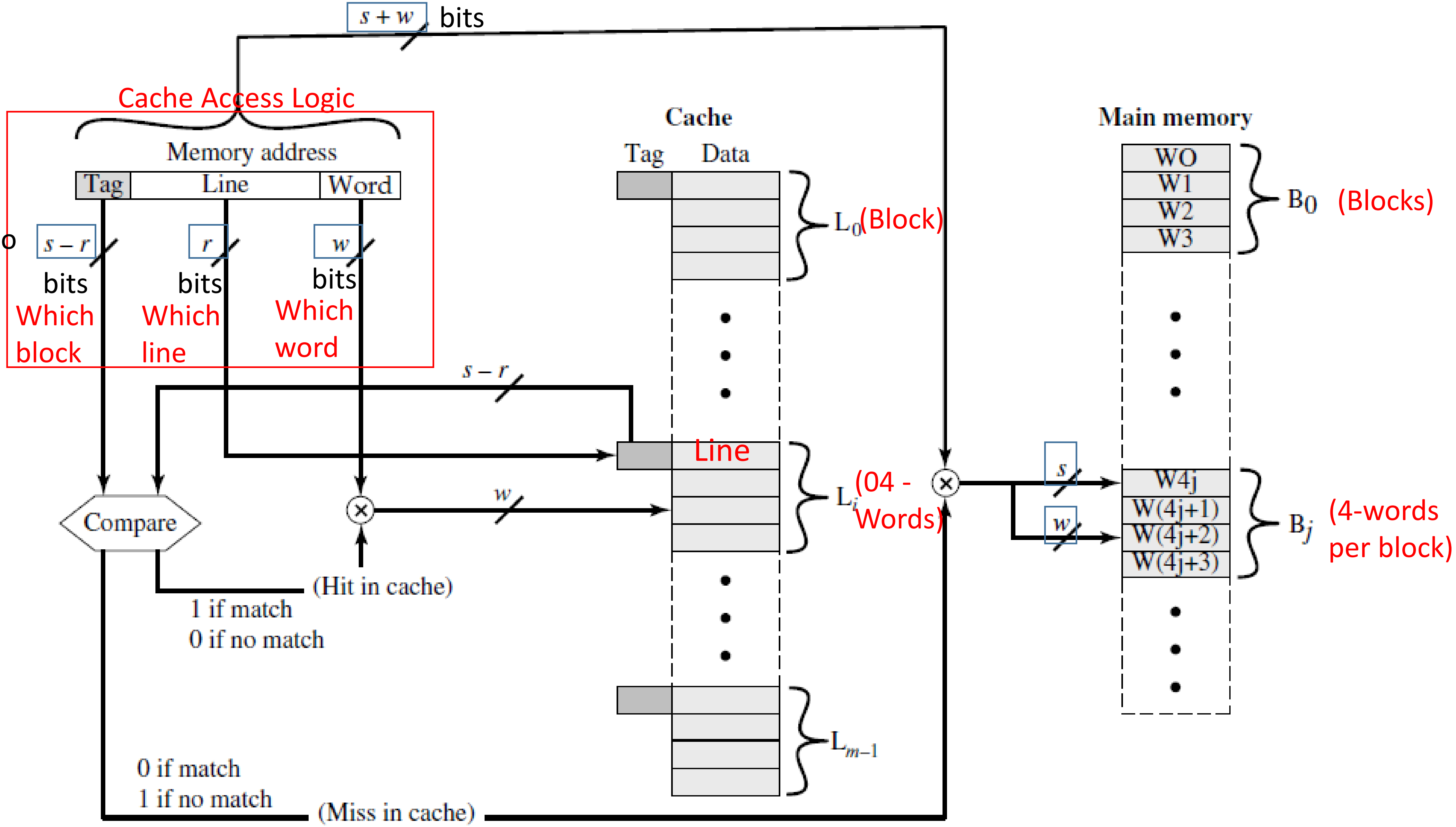


Figure 4.9 Direct-Mapping Cache Organization

Summary of ‘Direct Mapping’

- Address length $= (s + w)$ bits

  Where **s** is the number of blocks of main-memory and **w** is a word.

- Number of addressable units $= 2^{s+w}$ words or bytes

  In main-memory

- Block size $=$ line size $= 2^w$ words or bytes

  e.g. w =2-bits specifies 4-words in that block

- Number of blocks in main memory $= \dfrac{2^{s+w}}{2^w} = 2^s$

  (Blocks * words) / words = blocks

- Number of lines in cache $= m = 2^r$

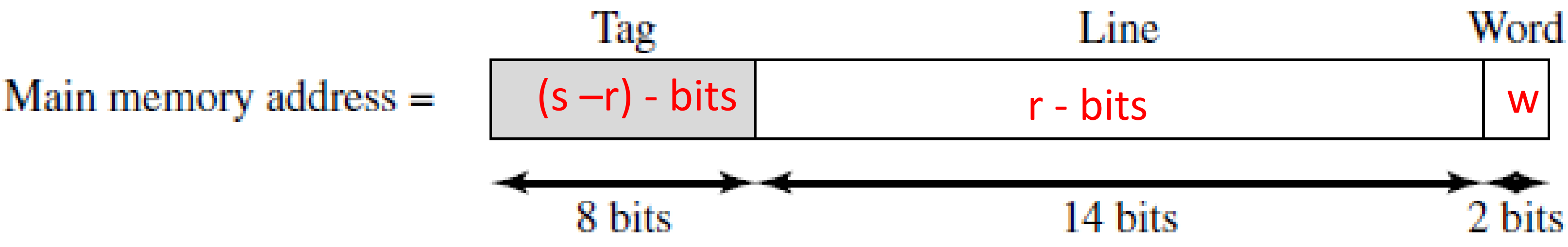
  r bits specify **m** lines of the cache

- Size of cache $= 2^{r+w}$ words or bytes

  Cache capacity = lines * words per line = **r + w** bits

- Size of tag $= (s - r)$ bits

  Note: **Tag** distinguishes a block from other blocks that can fit into that line.



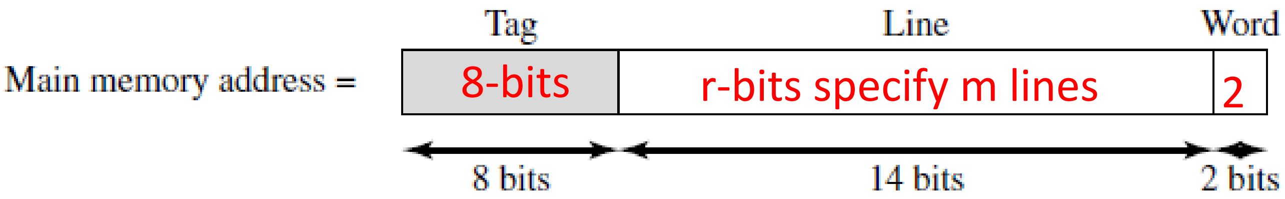
Main memory address =

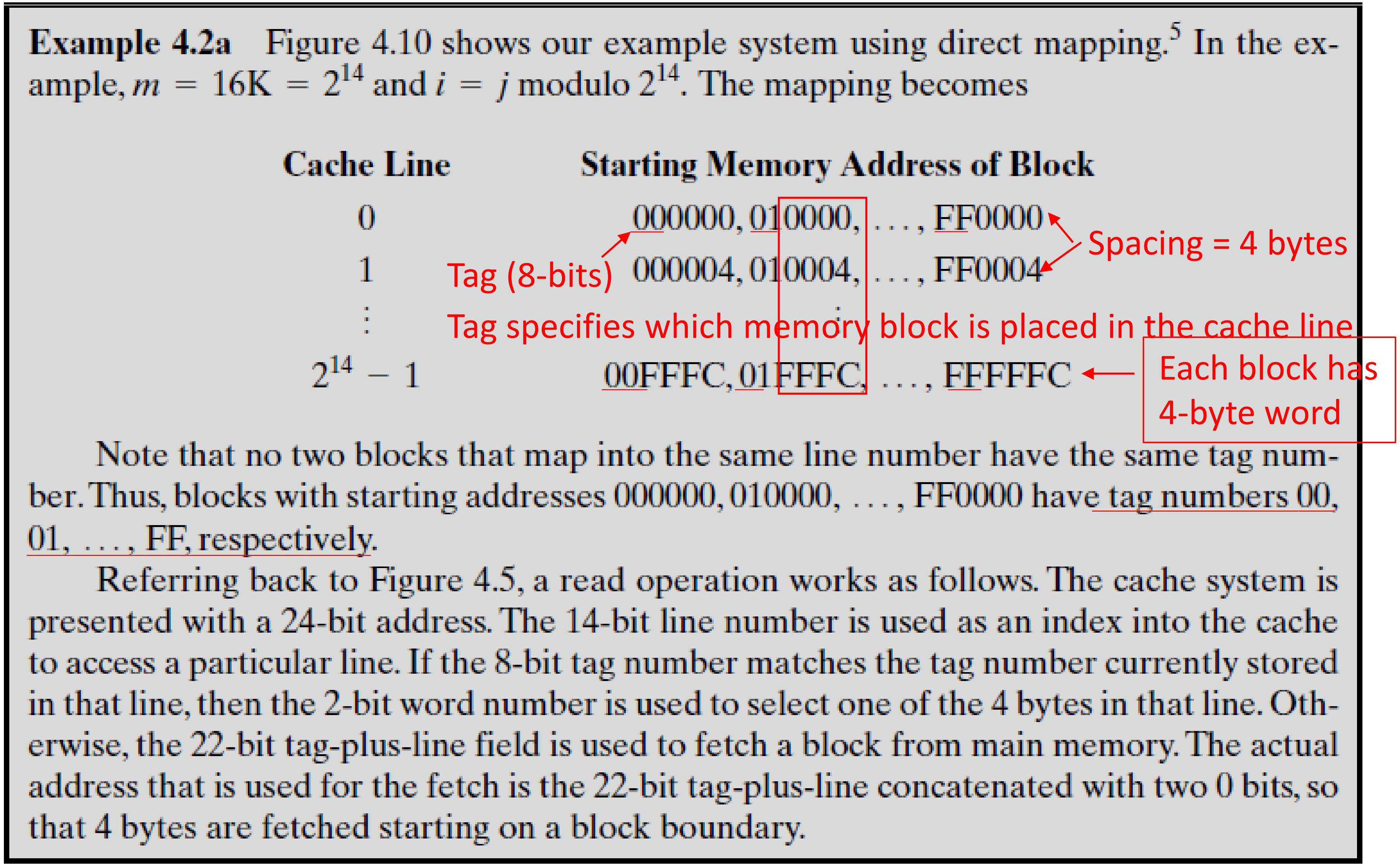
Tag	Line	Word
(s –r) - bits	r - bits	w
8 bits	14 bits	2 bits

Advantages and Disadvantages (Direct Mapping)

- **Advantages**
- The direct mapping technique is simple and inexpensive to implement
- **Disadvantage**
- 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, ‘cache misses’ are HIGH.
- Then the blocks will be continuously swapped in the cache, and a hit ratio will be low, this phenomenon is known as *thrashing*.

Example 4.2(a)





Example 4.2a Figure 4.10 shows our example system using direct mapping.[5] In the example, $m = 16\text{K} = 2^{14}$ and $i = j$ modulo 2^{14}. The mapping becomes

Cache Line	Starting Memory Address of Block
0	000000, 010000, …, FF0000
1	000004, 010004, …, FF0004
⋮	⋮
$2^{14} - 1$	00FFFC, 01FFFC, …, FFFFFC

Tag (8-bits)

Spacing = 4 bytes

Tag specifies which memory block is placed in the cache line

Each block has 4-byte word

Note that no two blocks that map into the same line number have the same tag number. Thus, blocks with starting addresses 000000, 010000, …, FF0000 have tag numbers 00, 01, …, FF, respectively.

Referring back to Figure 4.5, a read operation works as follows. The cache system is presented with a 24-bit address. The 14-bit line number is used as an index into the cache to access a particular line. If the 8-bit tag number matches the tag number currently stored in that line, then the 2-bit word number is used to select one of the 4 bytes in that line. Otherwise, the 22-bit tag-plus-line field is used to fetch a block from main memory. The actual address that is used for the fetch is the 22-bit tag-plus-line concatenated with two 0 bits, so that 4 bytes are fetched starting on a block boundary.

For Example

Tag s-r	Line or Slot r	Word w
8	14	2

- 24 bit address
- 2 bit word identifier (4 byte block)
- 22 bit block identifier
  - 8 bit tag (=22-14)
  - 14 bit slot or line
- No two blocks in the same line have the same Tag field
- Check contents of cache by finding line and checking Tag

Tag (s - r)	Line or Slot r	Word w
8	14	2

Cache Read Operation (Example)

1. The cache system is presented with a 24-bit address.
2. The 14-bit line number is used as an index into the cache to access a particular line. (containing 4 bytes/words)
3. If the 8-bit tag number (**is found**), matches the tag number currently stored in that line, then the 2-bit word number is used to select one of the 4 bytes in that line.
4. Otherwise (**if not found**), the 22-bit tag-plus-line field is used to fetch a block from main memory.
5. The actual address that is used for the fetch is the 22-bit tag-plus-line concatenated with two 0-bits, so that 4 bytes ($2^2 = 4$) are fetched starting on a block boundary.

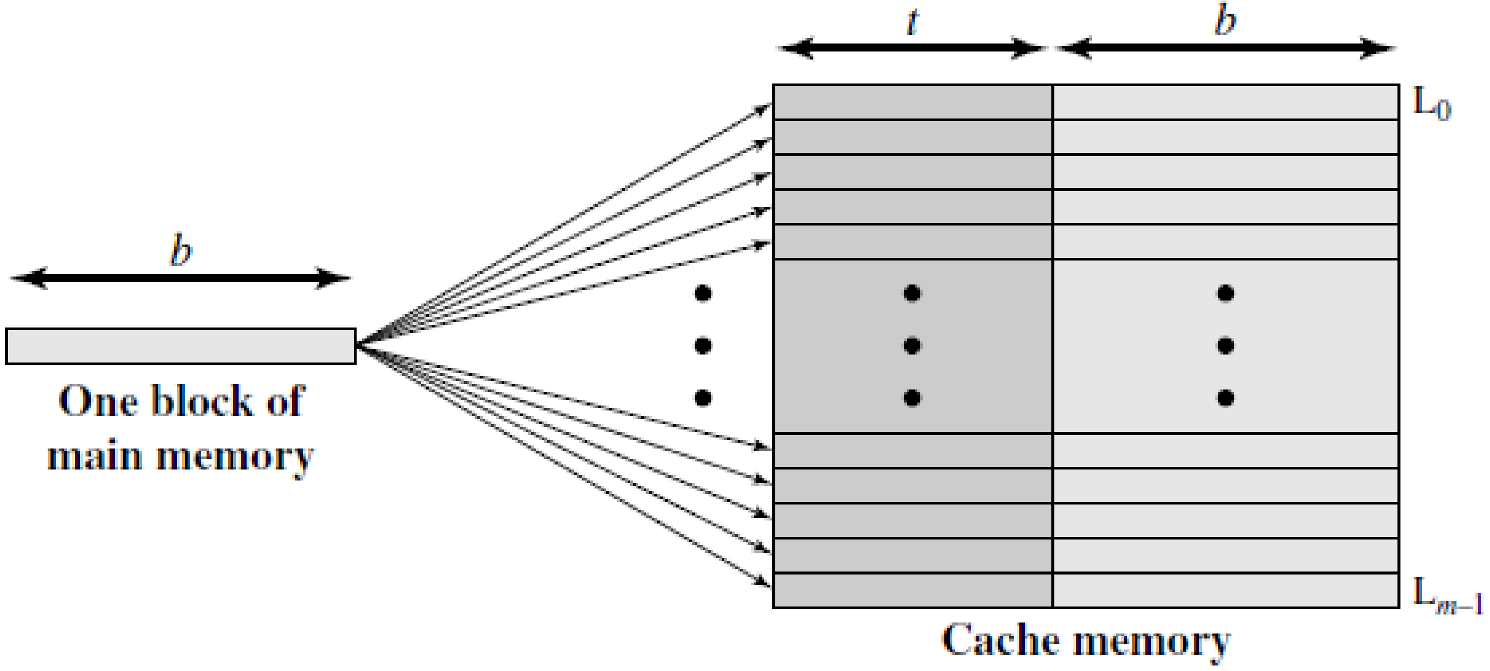
Victim Cache

- One approach to lower the miss penalty is to remember what was discarded in case it is needed again.
- Since the discarded data has already been fetched, it can be used again at a small cost.
- Such recycling is possible using a ‘Victim cache’.
- ‘Victim cache’ was originally proposed as an approach to reduce the conflict misses of direct mapped caches without affecting its fast access time.
- Victim cache is a fully associative cache, whose size is typically 4 to 16 cache lines, residing between a direct mapped L1 cache and memory.

(Break)

2. Associative Mapping

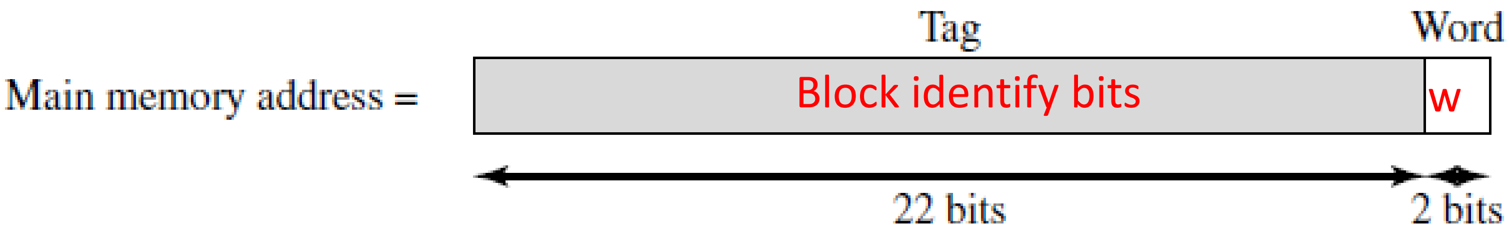
- **Associative mapping** permits each main-memory block to be loaded into any line of the cache.

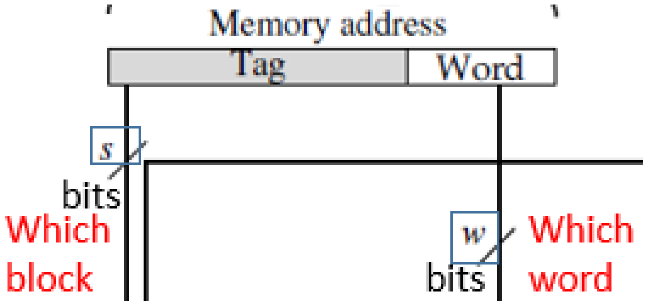


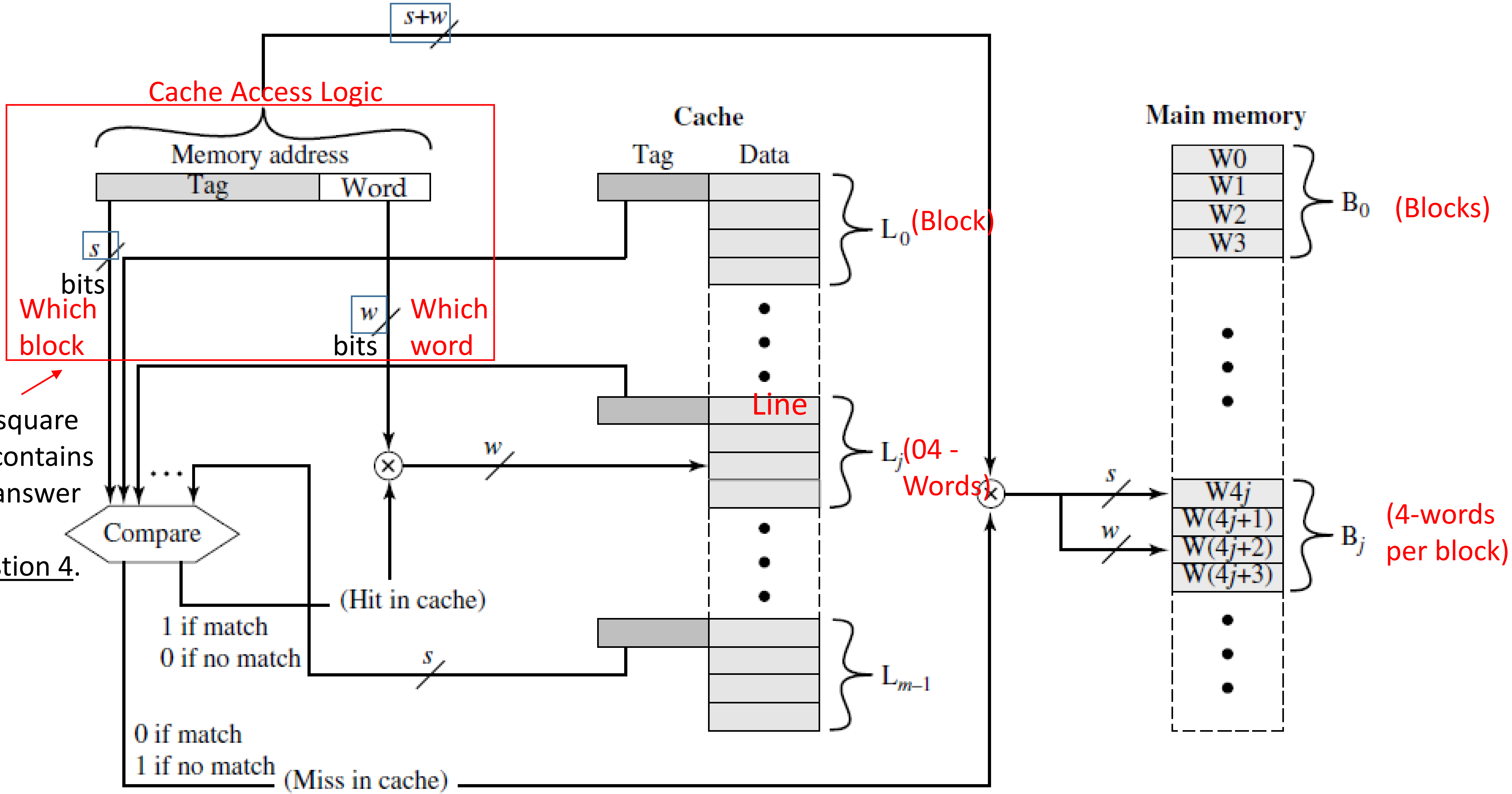
(b) Associative mapping

Fully Associate Cache Organization (Fig. Next)

- 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. **M** = 2^n/K blk
- To determine whether a block is in the cache, the ‘cache control logic’ must simultaneously examine every line’s tag for a match.
- Note that no field in the address corresponds to the line number ‘**r**’, so that the number of lines in the cache is not determined by the address format.





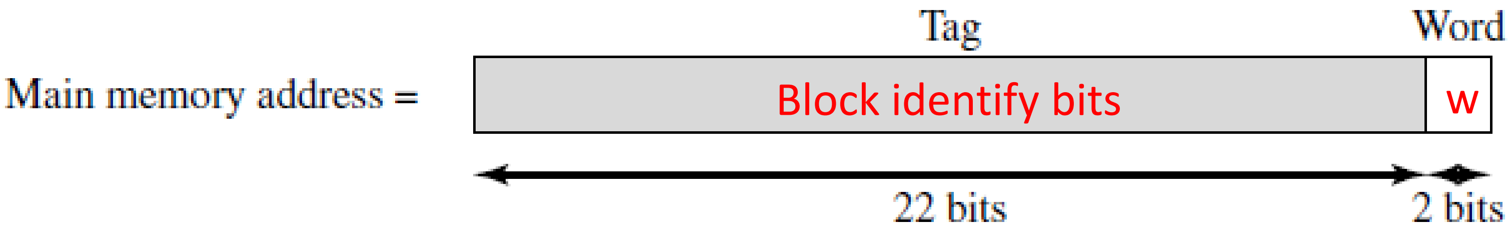


This square Box contains The answer to <u>Question 4</u>.

Figure 4.11 Fully Associative Cache Organization

Summary of ‘Associative Mapping’

- Address length $= (s + w)$ bits — Block of main memory = **s**, no. of words in each block = **w**
- Number of addressable units $= 2^{s+w}$ words or bytes — In main-memory
- Block size $=$ line size $= 2^{w}$ words or bytes — e.g. 2^2 = 4 bytes
- Number of blocks in main memory $= \dfrac{2^{s+w}}{2^{w}} = 2^{s}$ — (Blocks * words) / words = blocks
- Number of lines in cache $=$ undetermined — That is why we have NO **r**-bits in address
- Size of tag $= s$ bits — **Tag** distinguishes a block of main memory



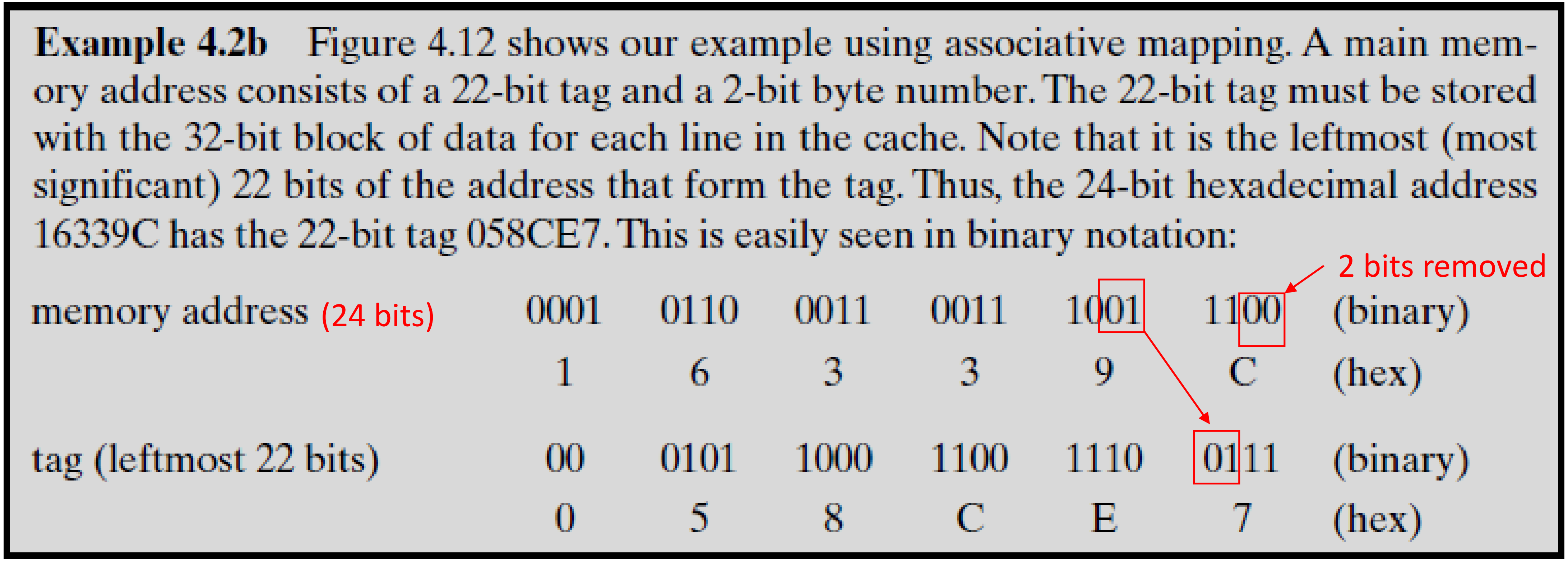
Main memory address =

Tag	Word
Block identify bits	w
22 bits	2 bits

Advantages and Disadvantages (Associative Mapping)

- **Advantages**
- With ‘Associative mapping’, there is flexibility as to which block to replace when a new block is read into the cache.
- **Replacement algorithms**, discussed later in this section, are designed to maximize the HIT-ratio.
- **Disadvantage**
- The principal disadvantage of ‘associative mapping’ is the complex circuitry required to examine the tags of all cache lines in parallel.
- This means that ‘cache searching gets expensive’.

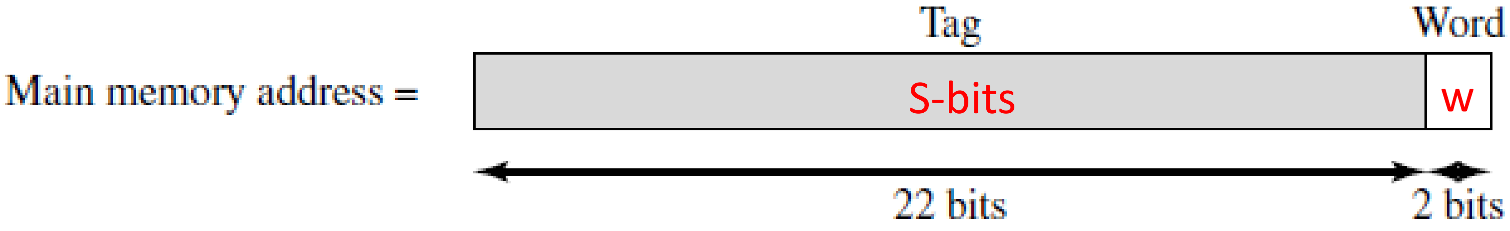
Example 4.2 (b)



Example 4.2b Figure 4.12 shows our example using associative mapping. A main memory address consists of a 22-bit tag and a 2-bit byte number. The 22-bit tag must be stored with the 32-bit block of data for each line in the cache. Note that it is the leftmost (most significant) 22 bits of the address that form the tag. Thus, the 24-bit hexadecimal address 16339C has the 22-bit tag 058CE7. This is easily seen in binary notation:

memory address (24 bits)	0001	0110	0011	0011	1001	1100	(binary)
	1	6	3	3	9	C	(hex)
tag (leftmost 22 bits)	00	0101	1000	1100	1110	0111	(binary)
	0	5	8	C	E	7	(hex)

2 bits removed



We have 4M blocks in Main-memory => 4*1024*1024 = $2^{2+10+10}$ = 2^{22} =>22-bits

For Example

Tag 22 bit	Word 2 bit

- 22 bit tag stored with each 32 bit block of data
- Compare tag field with tag entry in cache to check for hit
- Least significant 2 bits of address identify which 8 bit word is required from 32 bit data block
- e.g.

Address	Tag	Data	Cache line
16339C	058CE7	24682468	3FFF

3. Set-Associative Mapping

- **Set-Associative mapping**, each word maps into all the cache lines in a specific set, so that main-memory block B_0 maps into set 0, and so on.
- The cache consists of a number of sets (v), each of which consists of a number of lines (k). The relationships are

$$\boxed{\begin{aligned} m &= v \times k \\ i &= j \text{ modulo } v \end{aligned}}$$

where

i = cache set number (out of V-sets)

j = main memory block number

m = number of lines in the cache

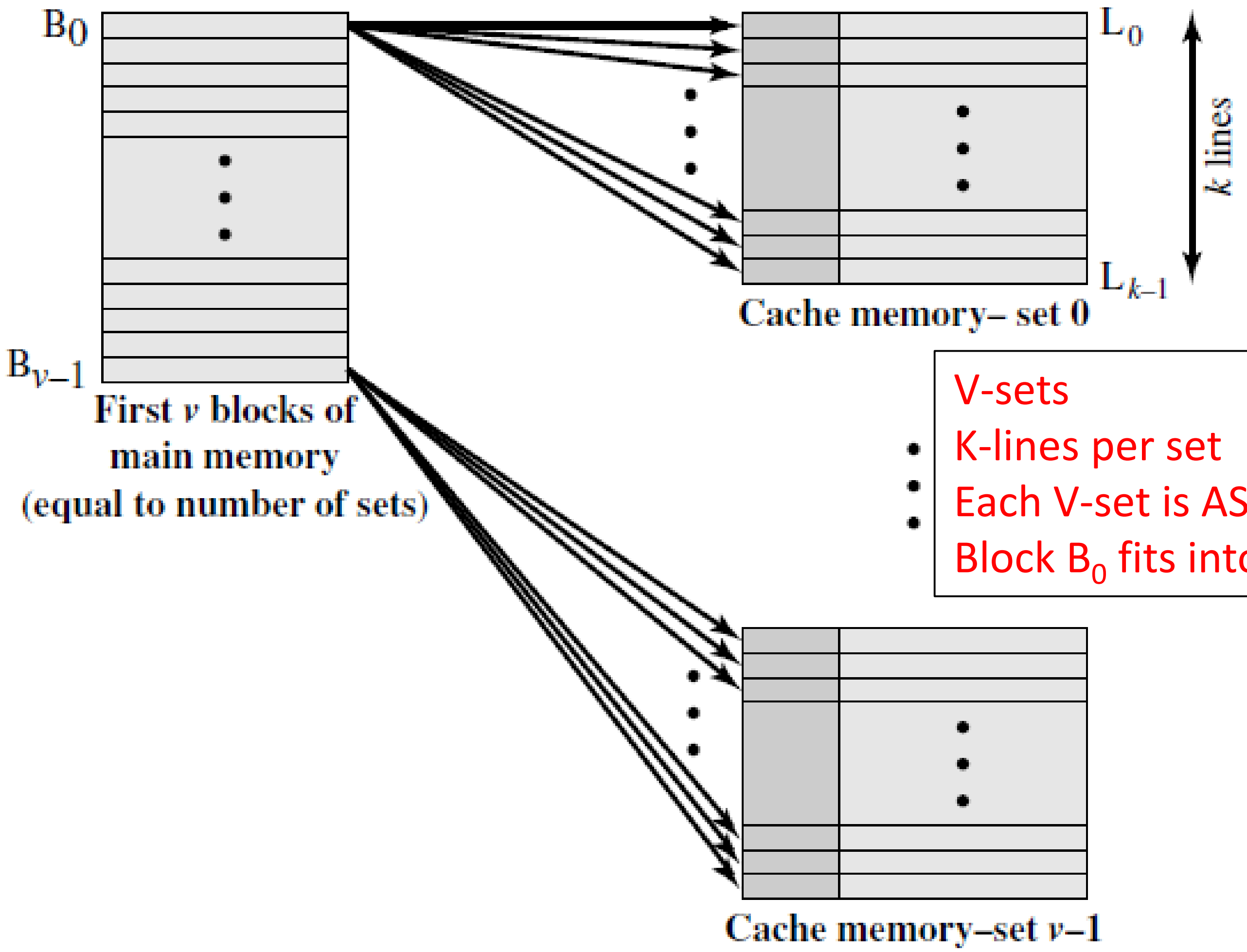
v = number of sets

k = number of lines in each set (K-way ‘Set-Associative’)

K-way Set-Associative Mapping (Fig. Next)

- With **k**-lines per set, this is referred to as k-way set-associative mapping.
- With set-associative mapping, block $\mathbf{B_j}$ can be mapped into any of the lines of set **j**.
- Thus set-associative cache can be implemented as v-associative cache
- E.g. in a '2-way set-associative mapping' (having 2 lines per set), a given block can be in one of the two-lines in any one set.
- Figure (next slide) illustrates this mapping for the first **v** blocks of main-memory.

‘V’ Associated–Mapped Caches



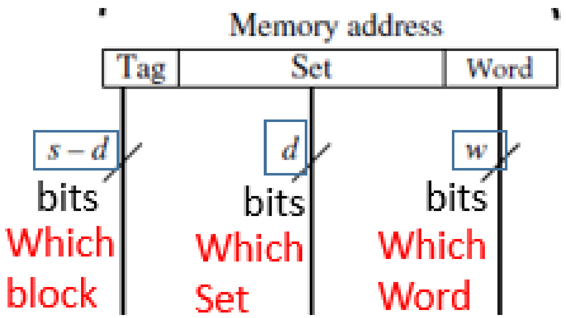
V-sets

K-lines per set

Each V-set is ASSOCIATIVE

Block B_0 fits into any line of set 0

Set-Associative 'Cache Control Logic'



- For 'Set-Associative mapping', the cache control logic interprets a memory address as three fields: **Tag**, **Set**, and **Word**.
- The **d** set bits specify one of **v = 2^{d}** sets.
- The **s** bits of the **Tag and Set** fields specify one of the **2^{s}** blocks of main-memory.
- **<u>Advantage</u>:** With '**k-way** set-associative mapping', the tag in a memory address is much smaller and is only compared to the **k** tags within a single set. (whereas associative-mapping tag is large, and compared with all lines).
- Figure (next slide) illustrates the 'cache control logic'.

This square Box contains The answer to Question 5.

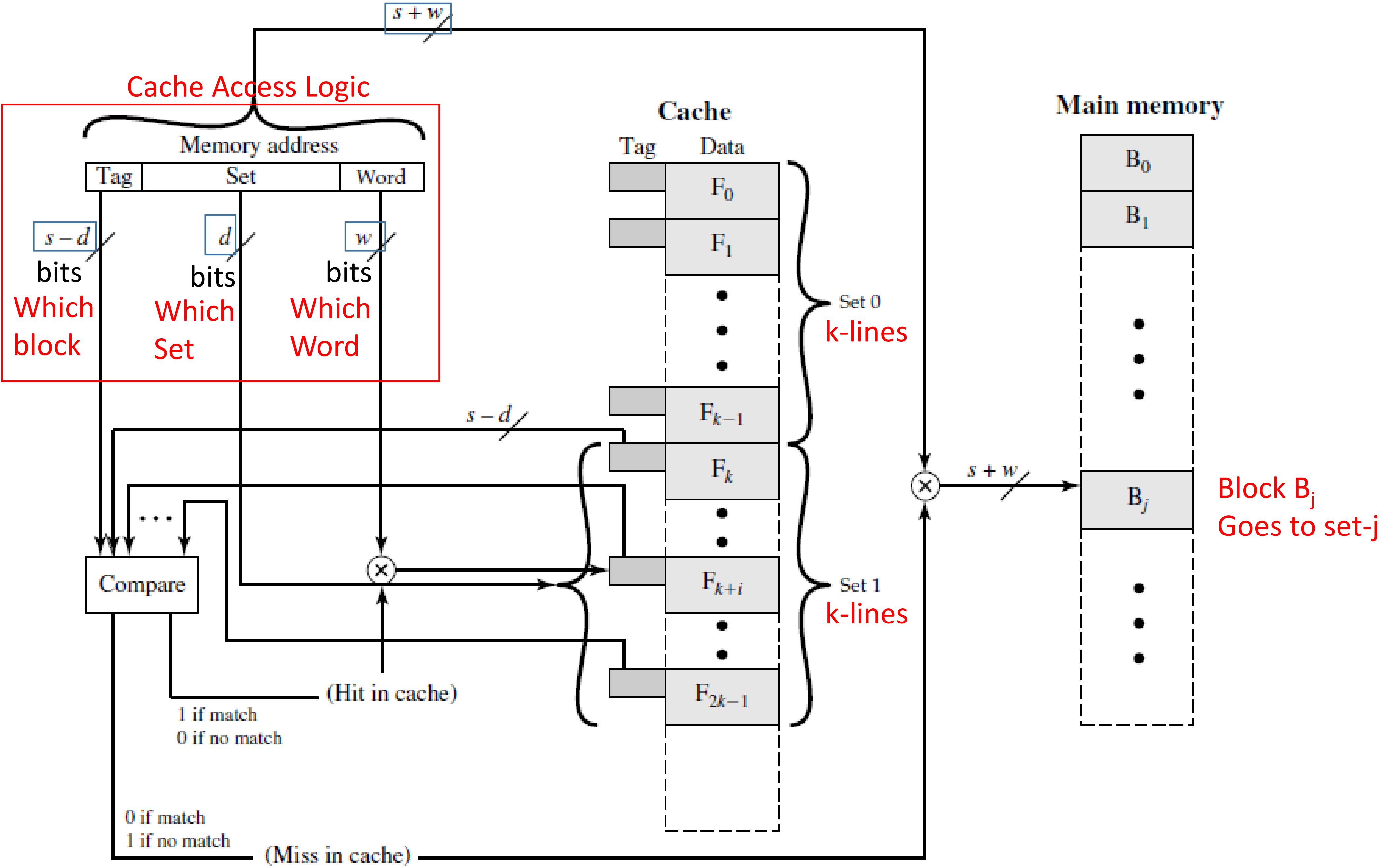
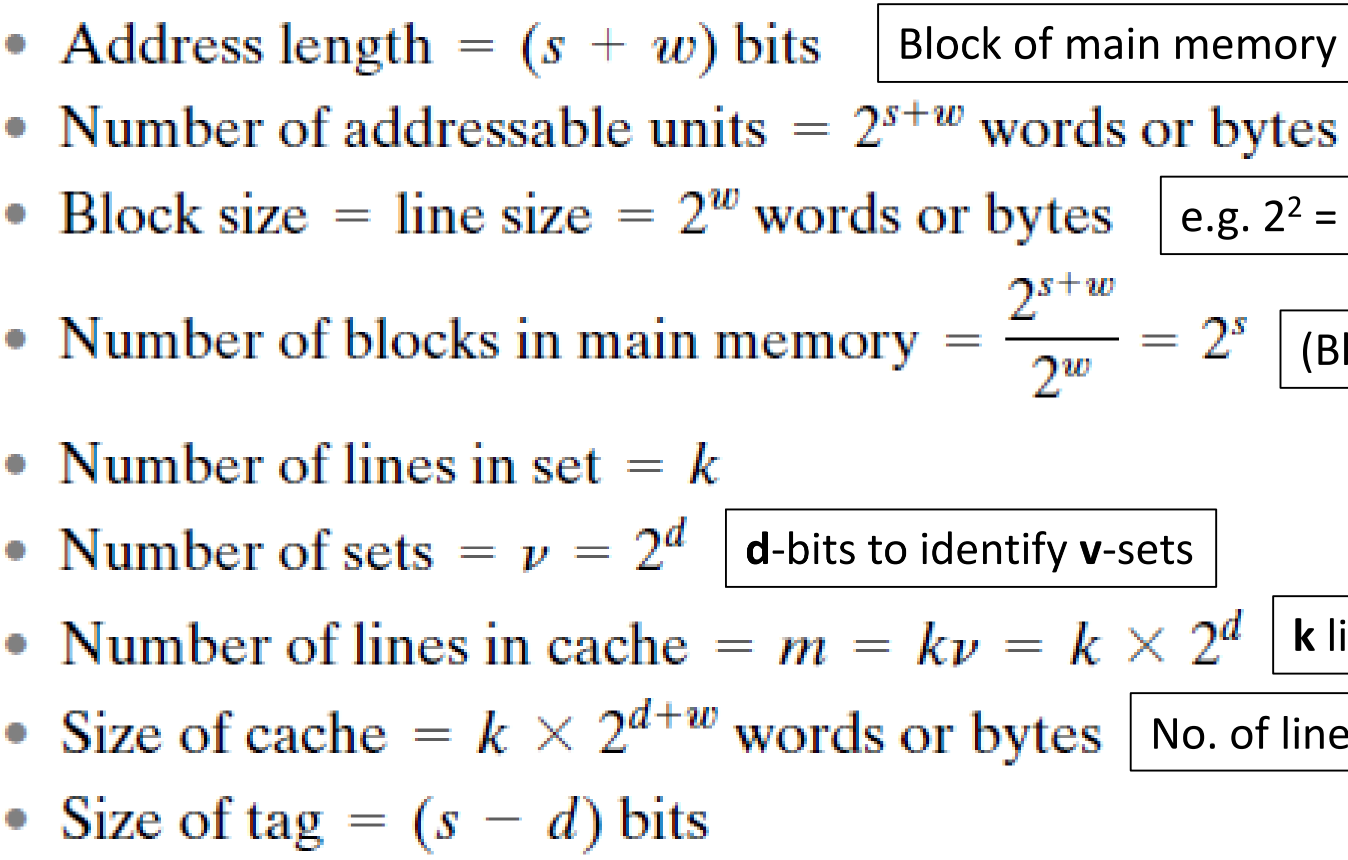


Figure 4.14 *K*-Way Set Associative Cache Organization

Summary of 'Set-Associative Mapping'



- Address length $= (s + w)$ bits

Block of main memory = **s**, no. of words in each block = **w**

- Number of addressable units $= 2^{s+w}$ words or bytes

In main-memory

- Block size = line size $= 2^w$ words or bytes

e.g. $2^2 = 4$ bytes

- Number of blocks in main memory $= \dfrac{2^{s+w}}{2^w} = 2^s$

(Blocks * words) / words = blocks

- Number of lines in set $= k$
- Number of sets $= v = 2^d$

d-bits to identify **v**-sets

- Number of lines in cache $= m = kv = k \times 2^d$

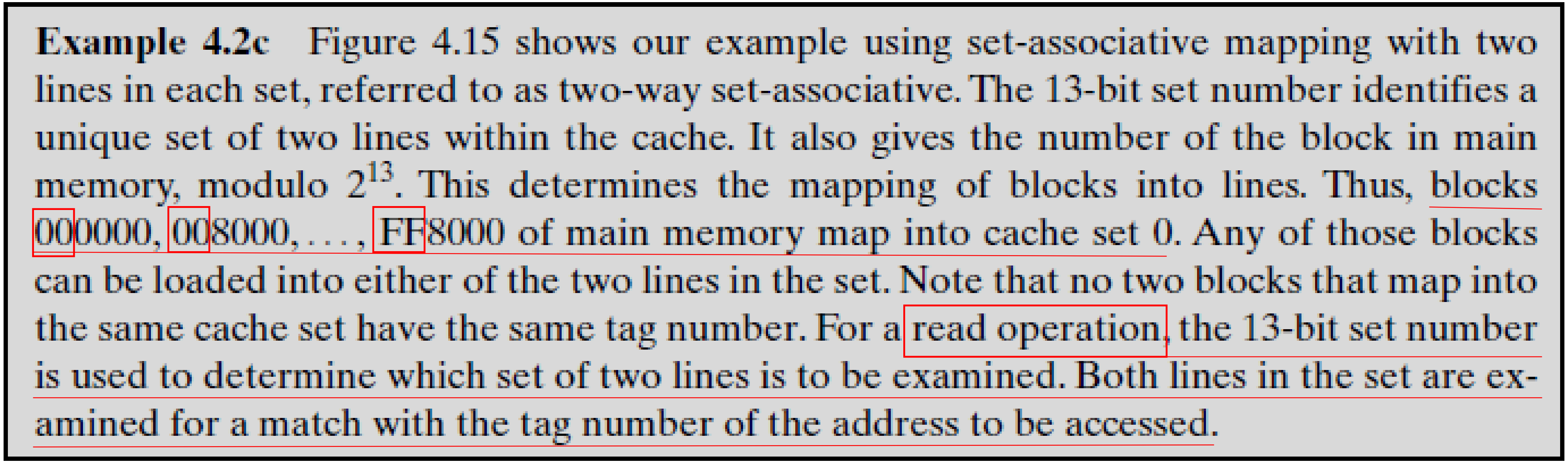
k lines per **v** sets

- Size of cache $= k \times 2^{d+w}$ words or bytes

No. of lines (k) **X** No. of sets (v) * No. of words (w)

- Size of tag $= (s - d)$ bits

Example 4.2 (c)



Example 4.2c Figure 4.15 shows our example using set-associative mapping with two lines in each set, referred to as two-way set-associative. The 13-bit set number identifies a unique set of two lines within the cache. It also gives the number of the block in main memory, modulo 2^{13}. This determines the mapping of blocks into lines. Thus, blocks 000000, 008000, . . . , FF8000 of main memory map into cache set 0. Any of those blocks can be loaded into either of the two lines in the set. Note that no two blocks that map into the same cache set have the same tag number. For a read operation, the 13-bit set number is used to determine which set of two lines is to be examined. Both lines in the set are examined for a match with the tag number of the address to be accessed.

- In a ‘2-way set-associative mapping’ (having 2 lines per set), a given block can be in one of the two-lines in any one set.
- 13 bit set number (identifies a unique set of two lines within the cache). Block number in main memory is modulo 2^{13}

For Example (Cache Read)

Tag 9 bit	Set 13 bit	Word 2 bit

- Use set field to determine cache set to look in
- Compare tag field to see if we have a hit
- e.g

Address	Tag	Data	Set number
1FF 7FFC	1FF	12345678	1FFF
001 7FFC	001	11223344	1FFF

Set-Associative Mapping (Extreme Cases)

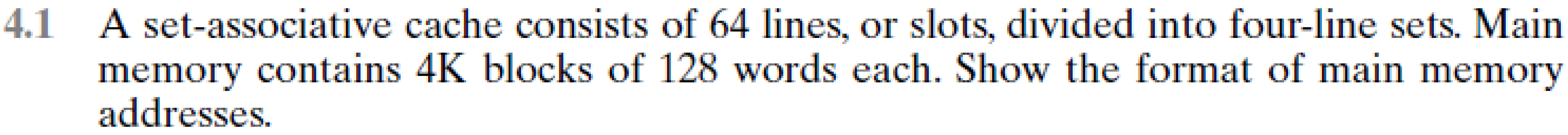
- In the extreme case of **v = m**, **k =1**, the set-associative technique reduces to **direct-mapping.**
- For **v =1**, **k = m**, it reduces to **associative-mapping.**

- The use of <u>two-lines per set</u> (**v = m/2** for **k = 2**) is the most common ‘set-associative’ organization.
- It significantly improves the HIT-ratio over ‘direct-mapping’.
- ‘Four-way set-associative’ (v = m/4, k = 4) makes a modest additional improvement. Further increase in No. of lines per set has little effect.

Problem 4-1

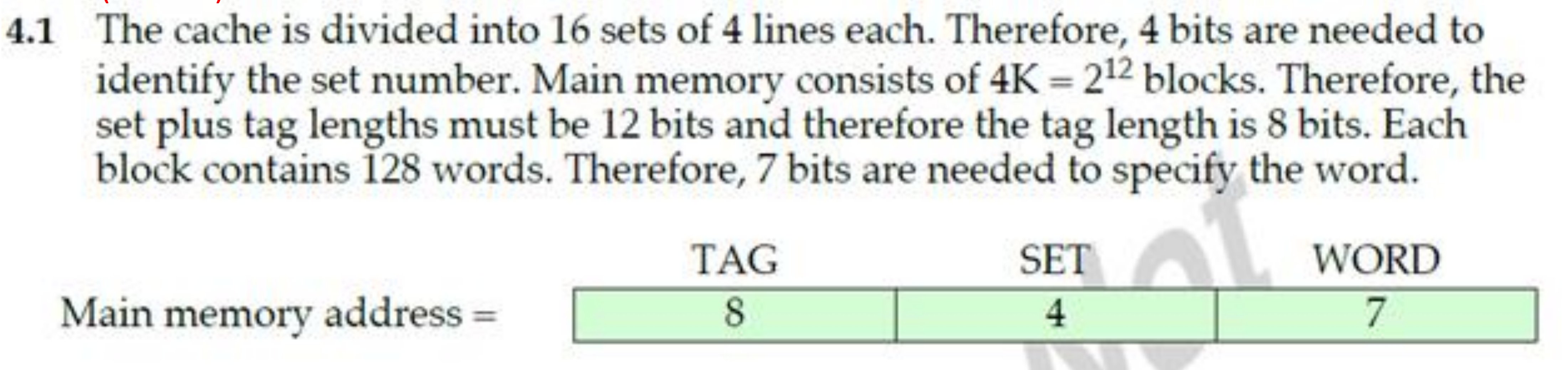
Assignment-3

See Last Questions



4.1 A set-associative cache consists of 64 lines, or slots, divided into four-line sets. Main memory contains 4K blocks of 128 words each. Show the format of main memory addresses.

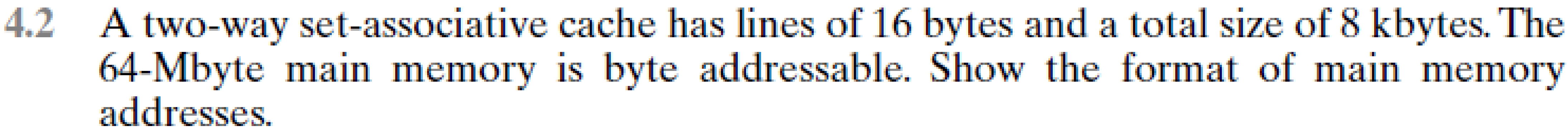
(Answer)



4.1 The cache is divided into 16 sets of 4 lines each. Therefore, 4 bits are needed to identify the set number. Main memory consists of 4K = 2^{12} blocks. Therefore, the set plus tag lengths must be 12 bits and therefore the tag length is 8 bits. Each block contains 128 words. Therefore, 7 bits are needed to specify the word.

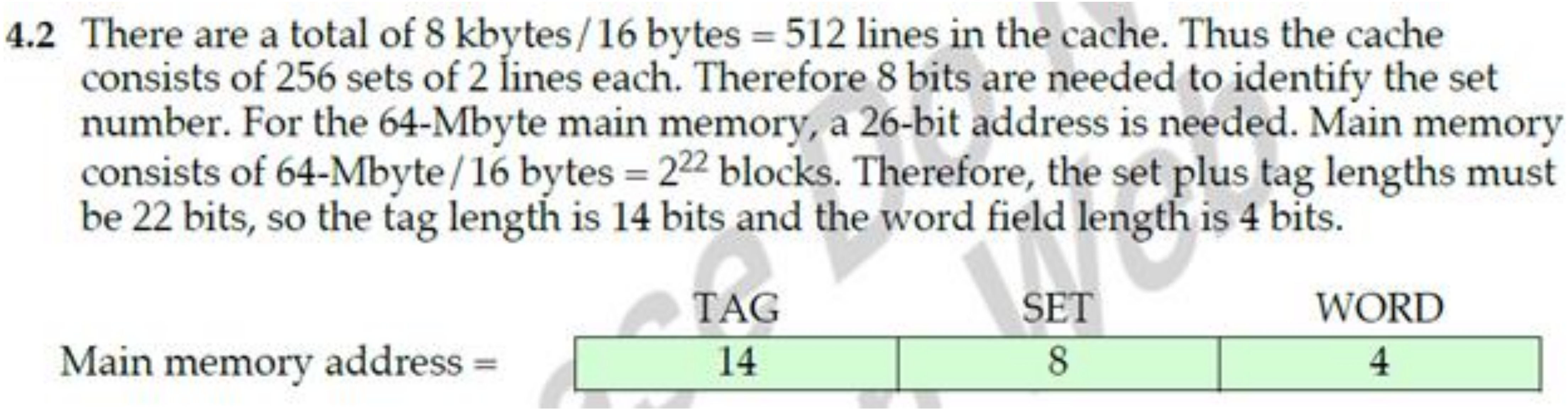
	TAG	SET	WORD
Main memory address =	8	4	7

Problem 4-2



4.2 A two-way set-associative cache has lines of 16 bytes and a total size of 8 kbytes. The 64-Mbyte main memory is byte addressable. Show the format of main memory addresses.

(Answer)



4.2 There are a total of 8 kbytes/16 bytes = 512 lines in the cache. Thus the cache consists of 256 sets of 2 lines each. Therefore 8 bits are needed to identify the set number. For the 64-Mbyte main memory, a 26-bit address is needed. Main memory consists of 64-Mbyte/16 bytes = 2^{22} blocks. Therefore, the set plus tag lengths must be 22 bits, so the tag length is 14 bits and the word field length is 4 bits.

	TAG	SET	WORD
Main memory address =	14	8	4

Summary of Three Mapping Techniques

Mapping Type	Address Length (bits)	No. of Addressable Units (words /bytes)	Block Size /Line Size (words /bytes)	No. of Blocks in Main-memory	No. of Lines in Cache (m)	Size of Cache (words)	Tag Size (bits)
Direct	(s + w)	2 ^ (s+ w)	2 ^ w	2 ^ s	2 ^ r (lines)	2 ^ (r + w)	(s – r)
Associative	(s + w)	2 ^ (s+ w)	2 ^ w	2 ^ s	undefined	unknown	s-bits
Set-Associative	(s + w)	2 ^ (s+ w)	2 ^ w	2 ^ s	k-lines per set	k * v * w (v = sets)	(s – d)

This is the difference in all 3 techniques

Note: **2 ^ w** means $\mathbf{2^w}$

(Break)

Cache ‘Replacement Algorithms’ (Associative)

- 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, there is only one possible line for a particular block, and no choice is possible. We have to replace that line (thrash).
- 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. We mention only four.

Types of ‘Cache Replacement Algorithms’

- Four of the most common cache replacement algorithms are:

1) Least Recently Used (LRU)
2) First-In-First-Out (FIFO)
3) Least Frequently Used (LFU)
4) Random Replacement (RR)

- ‘Least Recently Used (LRU)’ is the most effective and popular replacement algorithm.

Types of ‘Cache Replacement Algorithms’

1. **Least Recently Used (LRU):** Replace that block in the set that has been in the cache longest with no reference to it. (relates to time)

- We are assuming that more recently used memory locations are likely to be referenced again. This is called ‘Temporal locality’ and it should give the best HIT-ratio.
- The cache mechanism maintains a separate list of indexes to all the lines in the associative cache.
- When a line is referenced, it moves to the front of the list.
- For replacement, the line at the back of the list is used (discarded).

Types of ‘Cache Replacement Algorithms’

2. **First-In-First-Out (FIFO):** Replace that block in the set that has been in the cache longest. (first-come first-serve pattern)

- FIFO is easily implemented as a round-robin or circular buffer technique.

3. **Least Frequently Used (LFU):** Replace that block in the set that has experienced the fewest references. (relates to the use of data item)

- LFU could be implemented by associating a counter with each line.

4. **Random Replacement (RR):** Pick and replace a line at random from among the candidate lines.

Write Policy

- **Coherency property:** States that if a Word or a cache line (block) is modified in the lower level memory M1 (cache), then copies of that word must be updated immediately at all higher memory levels M2, M3 and so on, so as to maintain 'data coherency' among different memory levels.
- When a block that is resident in the cache is to be replaced, if it has not been altered (updated/overwritten), then it may directly be replaced without first writing out this block to the main memory.
- If at least one write operation has been performed on a word in that line of the cache, then main memory must be updated by writing that block to the main memory before bringing in the new block in cache.

Types of ‘Write Policies’

- If a Word has been altered only in the cache, then the corresponding memory Word is invalid.
- If the I/O device has altered the main-memory, then the cache Word is invalid.
- To maintain ‘data coherency’, two write policies have been adopted:

1) Write through 2) Write back

Types of ‘Write Policies’

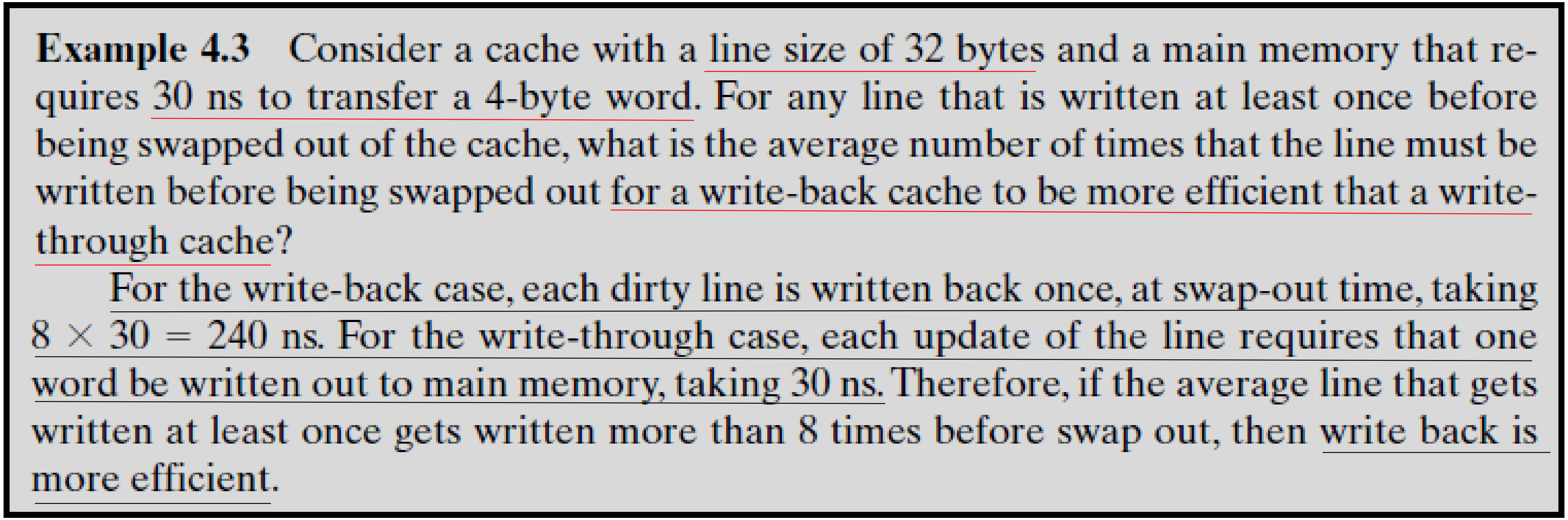
1. **Write through:** Using this technique, all write operations are made to main memory as well as to the cache simultaneously, ensuring that main-memory is always valid. (Advantage)

- ‘Write through’ does an ‘immediate update’ in RAM whenever a word is modified in cache.
- Its main disadvantage, it generates ‘memory traffic’ and bottleneck.

2. **Write back:** Minimizes memory writes, and it ‘delays an update’ in main-memory until that updated block is being replaced by a new block in that same line in the cache. (set ‘updated’ bit for a line)

- Before ‘write back’, the main-memory block is invalid (Disadvantage).

Example 4.3



Example 4.3 Consider a cache with a line size of 32 bytes and a main memory that requires 30 ns to transfer a 4-byte word. For any line that is written at least once before being swapped out of the cache, what is the average number of times that the line must be written before being swapped out for a write-back cache to be more efficient that a write-through cache?

For the write-back case, each dirty line is written back once, at swap-out time, taking 8 × 30 = 240 ns. For the write-through case, each update of the line requires that one word be written out to main memory, taking 30 ns. Therefore, if the average line that gets written at least once gets written more than 8 times before swap out, then write back is more efficient.

line is 32 bytes

Answer: line

4-	byte	X	8	=	32	byt	es

32/4 = 8 writes for a line
time = 8 * 30ns = 240 ns
for 'write back' at swap-out.

If average line gets written more than once, 'write-through' is inefficient.

Line Size

- When a block of data is retrieved and placed in the cache, not only the desired word but also some number of adjacent words are retrieved. (a block is a combination of words)
- As the block size increases, the hit ratio will at first increase, due to:
- **Principal of locality**, which states that data in the vicinity of a referenced word are likely to be referenced in the near future.
- As the block size increases further, the hit ratio will begin to decrease.
- The relationship between block size and hit ratio is complex, and no definitive optimum value has been found. 8-to-64 bytes is reasonable.

Number of Caches

- **Multilevel Caches / Cache Hiererchy**

1. **On-chip cache (L1):** is a cache on the same chip as the processor.

- The 'on-chip cache (Level-1)' reduces the processor's external bus activity and therefore speeds up execution times and increases overall system performance. It is often accessed in one cycle by CPU.

2. An **off-chip cache (L2)** is external to the processor and is designated as Level-2. It is made of static-RAM (SRAM).

- Due to the slow bus speed and slow memory access time, resulting in poor performance, an L2 cache is desirable & gives 'zero-wait state'.

- **L2 cache** uses a separate data path, so as to reduce the burden on the system bus.
- With the continued shrinking of the processor components (due to Moore’s law), a number of processors now incorporate the L2 cache on the processors chip, improving performance.
- A **Hit** is counted if the desired data appears in either the L1 or the L2 cache.
- L2 has little effect on the total number of cache hits until it is at least double the L1 cache size.
- With the increasing availability of on-chip area available for cache, most microprocessors have moved the L2 cache on the processor chip and added an **L3 cache**.
- The L3 cache is accessible over the external bus and gives a performance advantage to adding the third level. Cache size should be double.

‘Unified’ Versus ‘Split Caches’

- The on-chip cache first consisted of a single cache used to store references to both data and instructions.
- Now it has become common to split the cache into two: one dedicated to instruction and one dedicated to data, called **split cache.**
- These two caches both exist at same level, typically as two L1 caches.
- When the processor attempts to fetch and ‘instruction/data’ from main-memory, it first consults the ‘instruction/data’ L1 cache.
- **Unified cache** has only one cache that is used for holding both data and instructions.

Advantages of a ‘Unified Cache’

- There are two potential advantages of a unified cache over split cache:

1. For a given cache size, a unified cache has a higher hit rate than split caches because it balances the load between instruction and data fetches automatically.

- That is if an execution pattern involves many more instruction fetches than data fetches, then the cache will tend to fill up with instructions, and if an execution pattern involves relatively more data fetches, the opposite will occur.

2. Only one cache needs to be designed and implanted.

Advantages of a ‘Split Cache’

- The trend is toward split caches at the L1 and unified caches for higher levels.
- The key advantage of the split cache design is that it eliminates contention for the cache between the instruction fetch/decode unit and the execution unit. (useful for ‘pipelining’ of instructions)
- This enables the processor to fetch instructions ahead of time and fill a buffer, or pipeline, with instructions to be executed.

‘Spatial’ & ‘Temporal’ LOCALITY (Q 4.8 = Q12)

- **Principal of locality**, which states that data in the vicinity of a referenced word are likely to be referenced in the near future.
- The ‘principal of locality’ has two types:

1) Spatial locality 2) Temporal locality

1. **Spatial locality:** refers to the tendency of execution to involve a number of memory locations that are clustered. This means to ‘access neighbouring info items whose addresses are near one another’. Space locality.
2. **Temporal locality:** refers to the tendency for a processor to access memory locations that have been used recently. So the item that has a ‘probability of being used in future is kept’. (so don’t discard it) locality in time.

Preparatory Questions

Assignment-3

Attempt: Q1 to Q5

Q1. Why do we need 'cache mapping' techniques? What are the types of mapping? (Slide – 05)

Q2. What are the differences among 'direct mapping, associative mapping and set-associative mapping'? (Slide 9, 20 and 27)

Q3. For a 'direct-mapped cache', a main-memory address is viewed as consisting of three fields. List and define the three fields. (Slide – 13.Box, Attached Q5. Answer below)

Q4. For an 'associative cache', a main-memory address is viewed as consisting of two fields. List and define the two fields. (Slide – 22.Box, Attached Q6. Answer below)

Q5. For a 'set-associative cache', a main-memory address is viewed as consisting of three fields. List and define the three fields. (Slide – 31.Box, Attached Q7. Answer below)

Preparatory Questions

Q6. Write down the advantages and disadvantages of ‘direct mapping’. (Slide – 15)

Q7. Write down the advantages and disadvantages of ‘associative mapping’. (Slide – 24)

Q8. Write down the two ‘Extreme cases’ of ‘set-associative mapping’. (Slide – 35)

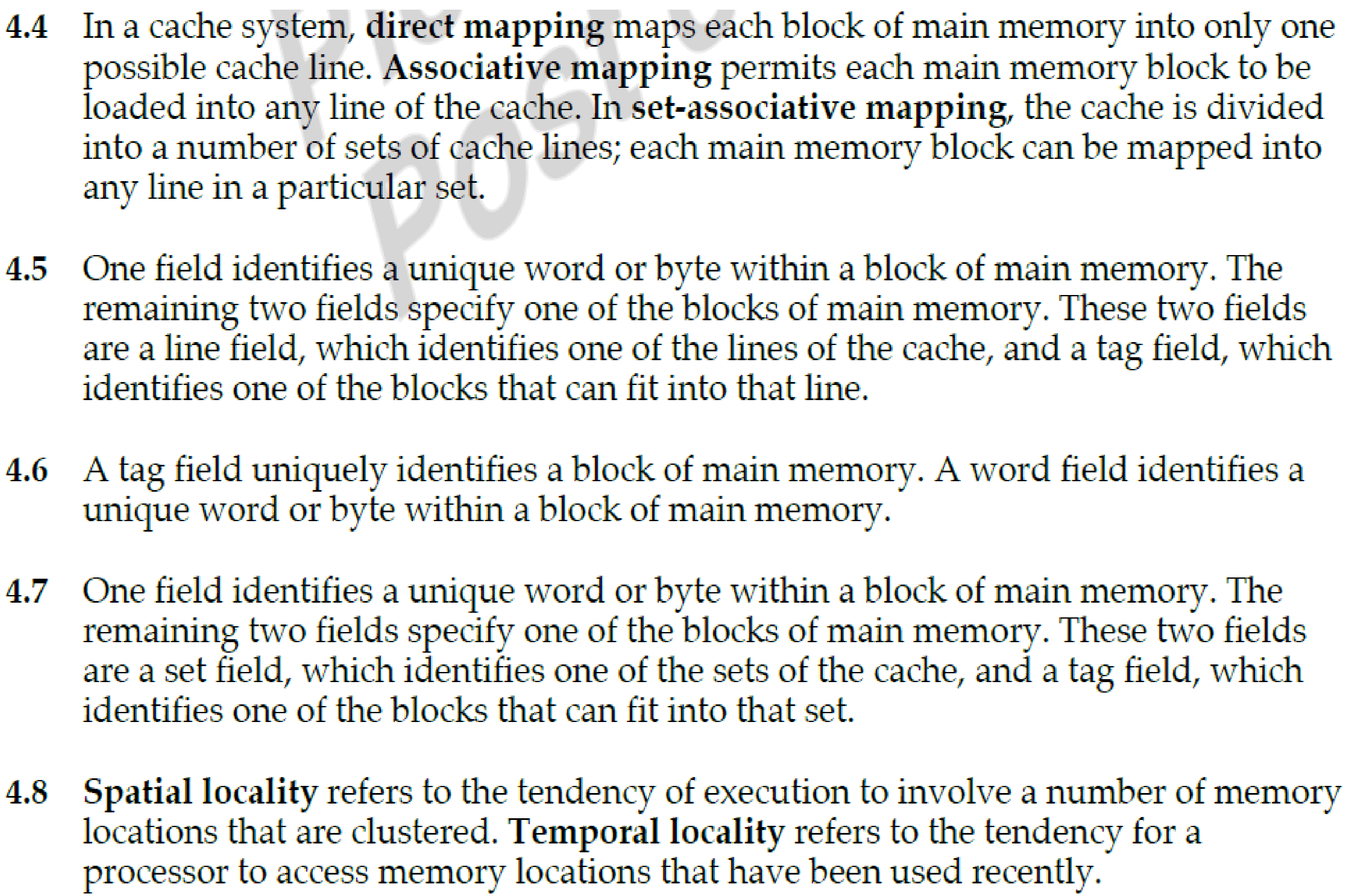
Q9. List and explain the various ‘replacement algorithms’ used in cache memory. (Slide – 39, 40, 41 and 42)

Q10. What is ‘data coherency’? To maintain data-coherency which two ‘write policies’ are adopted in caches? Explain. (Slide 43, 44 and 45)

Q11. State the advantages of ‘unified cache’ over ‘split cache’, and vice versa. (Slide – 51)

Q12. What is the distinction between ‘spatial locality’ and ‘temporal locality’? (Slide – 53)

Answers to Guess Questions: Q2. (4.4), Q3. (4.5), Q4. (4.6), Q5. (4.7), Q12. (4.8) below:



4.4 In a cache system, **direct mapping** maps each block of main memory into only one possible cache line. **Associative mapping** permits each main memory block to be loaded into any line of the cache. In **set-associative mapping**, the cache is divided into a number of sets of cache lines; each main memory block can be mapped into any line in a particular set.

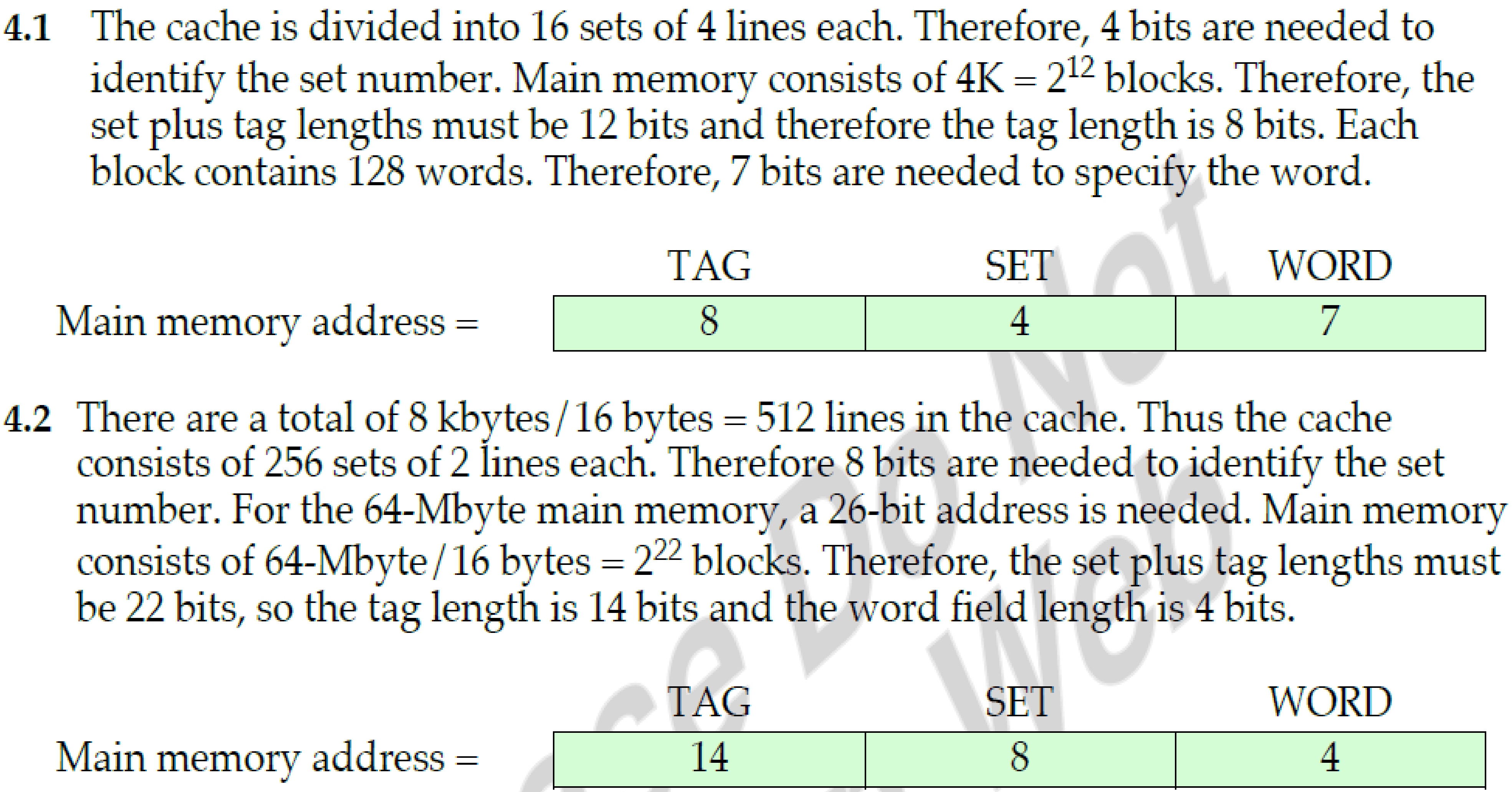
4.5 One field identifies a unique word or byte within a block of main memory. The remaining two fields specify one of the blocks of main memory. These two fields are a line field, which identifies one of the lines of the cache, and a tag field, which identifies one of the blocks that can fit into that line.

4.6 A tag field uniquely identifies a block of main memory. A word field identifies a unique word or byte within a block of main memory.

4.7 One field identifies a unique word or byte within a block of main memory. The remaining two fields specify one of the blocks of main memory. These two fields are a set field, which identifies one of the sets of the cache, and a tag field, which identifies one of the blocks that can fit into that set.

4.8 **Spatial locality** refers to the tendency of execution to involve a number of memory locations that are clustered. **Temporal locality** refers to the tendency for a processor to access memory locations that have been used recently.

Answers to Problems 4.1 and 4.2 below.



4.1 The cache is divided into 16 sets of 4 lines each. Therefore, 4 bits are needed to identify the set number. Main memory consists of 4K = 2^{12} blocks. Therefore, the set plus tag lengths must be 12 bits and therefore the tag length is 8 bits. Each block contains 128 words. Therefore, 7 bits are needed to specify the word.

	TAG	SET	WORD
Main memory address =	8	4	7

4.2 There are a total of 8 kbytes/16 bytes = 512 lines in the cache. Thus the cache consists of 256 sets of 2 lines each. Therefore 8 bits are needed to identify the set number. For the 64-Mbyte main memory, a 26-bit address is needed. Main memory consists of 64-Mbyte/16 bytes = 2^{22} blocks. Therefore, the set plus tag lengths must be 22 bits, so the tag length is 14 bits and the word field length is 4 bits.

	TAG	SET	WORD
Main memory address =	14	8	4

For Information

- **Problems 4.1** & **4.2** are part of your course.
- **Examples 4.1, 4.2(a), 4.2 (b), 4.2 (c),** and **4.3** are part of your course.
- Preparatory Questions are provided for this chapter.

Cache Addresses

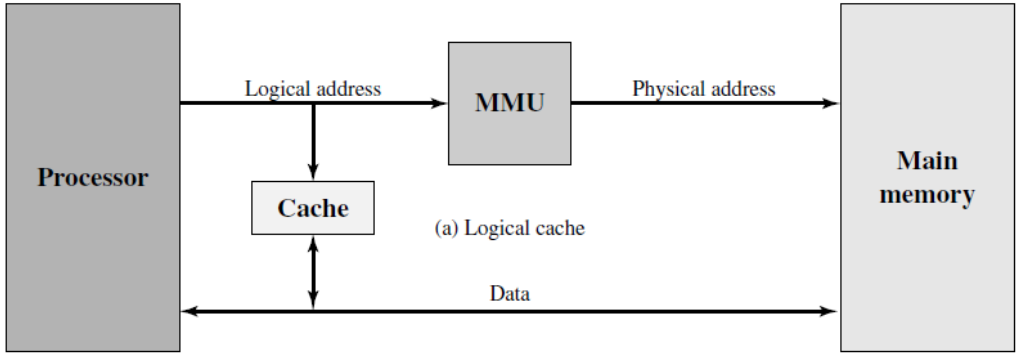
Out of Course

- Can be of two types

1) Logical addressing 2) Physical addressing

- Almost all processors support ‘Virtual Memory’.
- **Virtual memory** is a facility that allows programs to address memory locations higher than the processor’s addressing space from a logical point of view, without regard to the amount of main memory physically available. (This concept will be discussed in Chapter 8)
- Memory Management Unit (MMU) is a hardware unit that translates each virtual address into a physical address in main memory, to allow reads and writes to main memory.

Logical and Physical Caches



1. **Logical cache**, also known as **virtual cache**, stores data using **virtual addresses.**

- One obvious advantage of the logical cache is that 'the processor accesses the cache directly, without going through the MMU'. So the cache access speed is faster than for a physical cache. Because the cache can respond before the MMU performs an 'address translation'.
- The disadvantage is that most virtual memory systems supply each application with the same virtual memory address that starts at 0. Thus cache memory must be completely flushed for two applications.

Thus. the same virtual address in two different applications refers to two different physical addresses.

2. **Physical cache** stores data using main memory **physical addresses.**