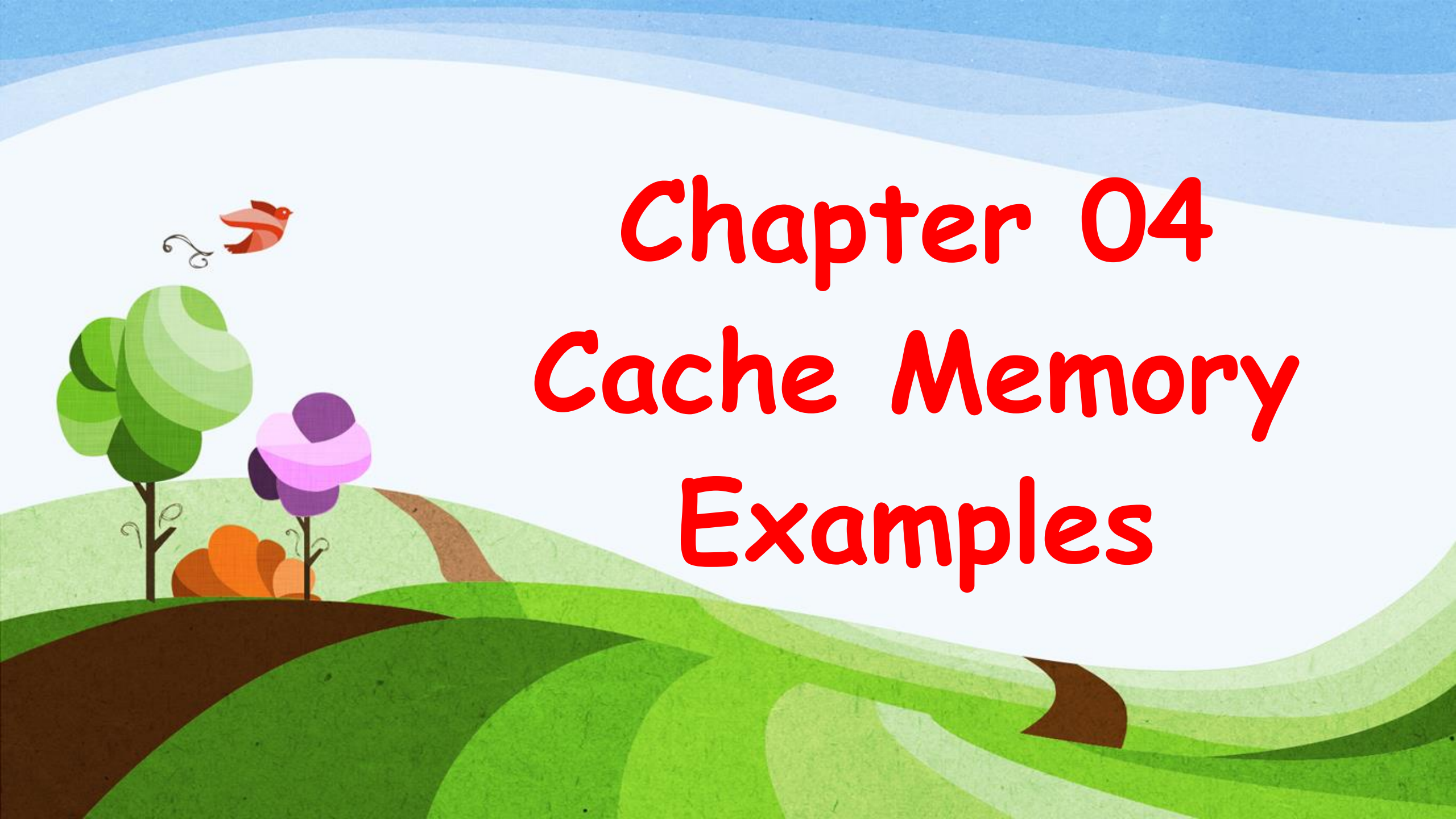


Chapter 04

Cache Memory

Examples



Access Methods

- **Sequential**

- Start at the beginning and read through in order
- Access time depends on location of data and previous location
- e.g. **tape**

- **Direct**

- Individual blocks have unique address
- Access is by jumping to vicinity plus sequential search
- Access time depends on location and previous location
- e.g. **disk**

Access Methods

- **Random**

- Individual addresses identify locations exactly
- Access time is independent of location or previous access
- e.g. **RAM**

- **Associative**

- to make a comparison of desired bit locations within a word for a specified match Thus, a word is retrieved based on a portion of its contents rather than its address
- Data is located by a comparison with contents of a portion of the store
- Access time is independent of location or previous access
- e.g. **cache**

From a user's point of view, the two most important characteristics of memory are **capacity** and **performance**.

Three performance parameters are used:

Access time (latency):

For RAM, this is the time it takes to perform a read or write operation, that is, the time from the instant that an address is presented to the memory to the instant that data have been stored or made available for use. For non-RAM, access time is the time it takes to position the read–write mechanism at the desired location.

Memory cycle time:

This concept is primarily applied to RAM and consists of the access time plus any additional time required before a second access can commence. This additional time may be required for transients to die out on signal lines. Note that memory cycle time is concerned with the system bus, not the processor.

Transfer rate:

This is the rate at which data can be transferred into or out of a memory unit. For RAM, it is equal to $1/(\text{cycle time})$.

The Memory Hierarchy

As might be expected, there is a trade-off among the three key characteristics of memory: namely, capacity, access time, and cost. A variety of technologies are used to implement memory systems, and across this spectrum of technologies, the following relationships hold:

- **Faster access time, greater cost per bit**
- **Greater capacity, smaller cost per bit**
- **Greater capacity, slower access time**

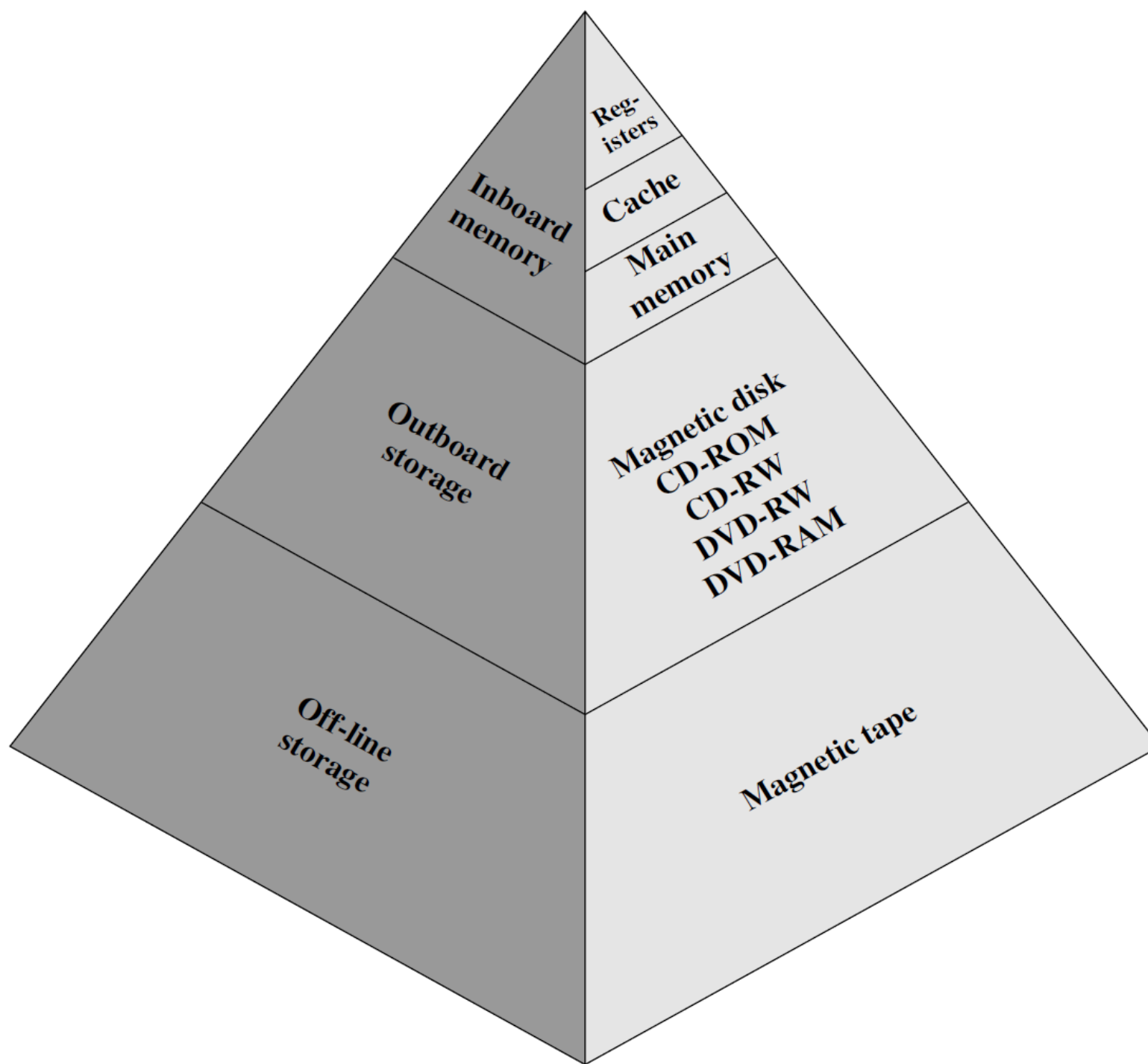
The dilemma facing the designer is clear. The designer would like to use memory technologies that provide for large-capacity memory, both because the capacity is needed and because the cost per bit is low. However, to meet performance requirements, the designer needs to use expensive, relatively lower-capacity memories with short access times

The way out of this dilemma is not to rely on a single memory component or technology, but to employ a memory hierarchy

As one goes down the hierarchy, the following occur:

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

Thus, smaller, more expensive, faster memories are supplemented by larger, cheaper, slower memories. The key to the success of this organization is item (d): decreasing frequency of access

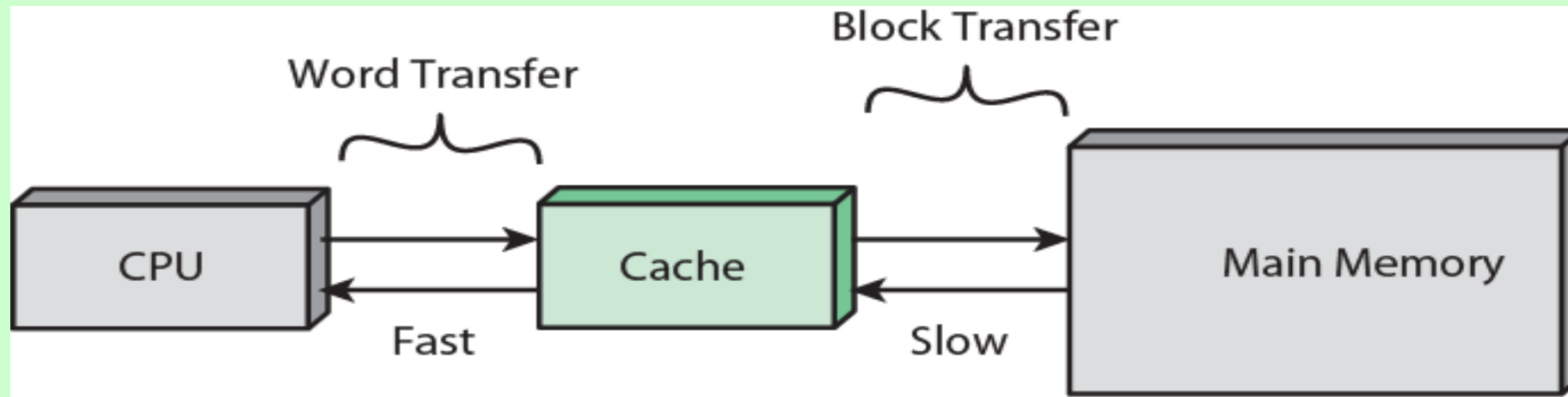


**Memory
Hierarchy
- Diagram**

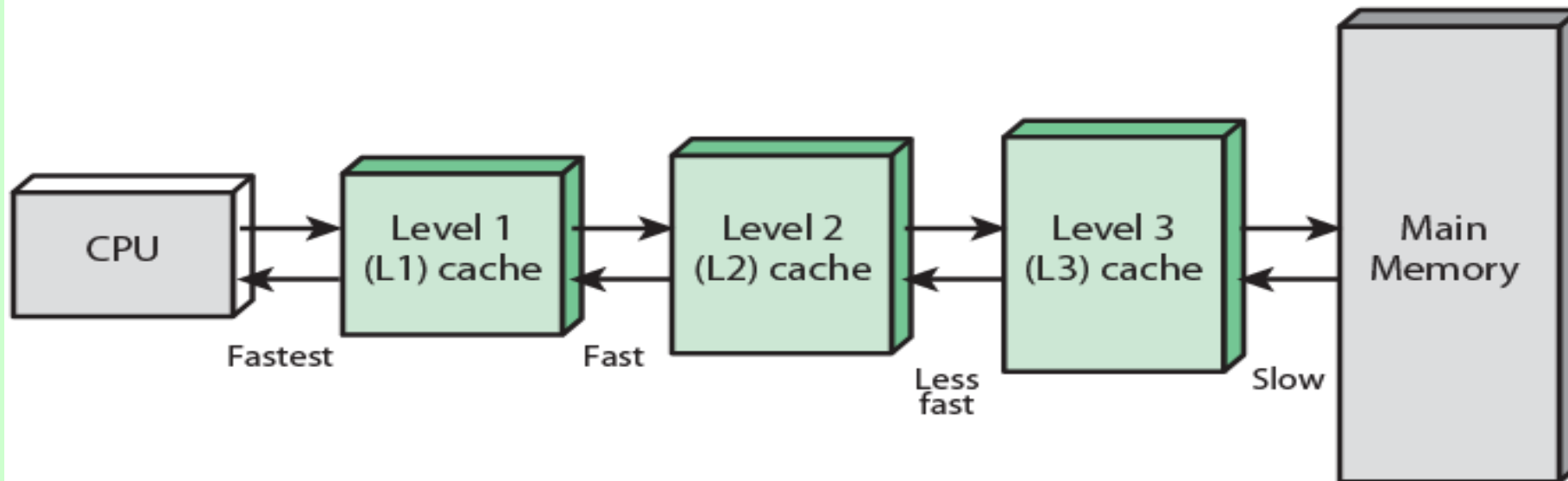
Hierarchy List

- Registers
- L1 Cache
- L2 Cache
- Main memory
- Disk cache
- Disk
- Optical
- Tape

Cache and Main Memory

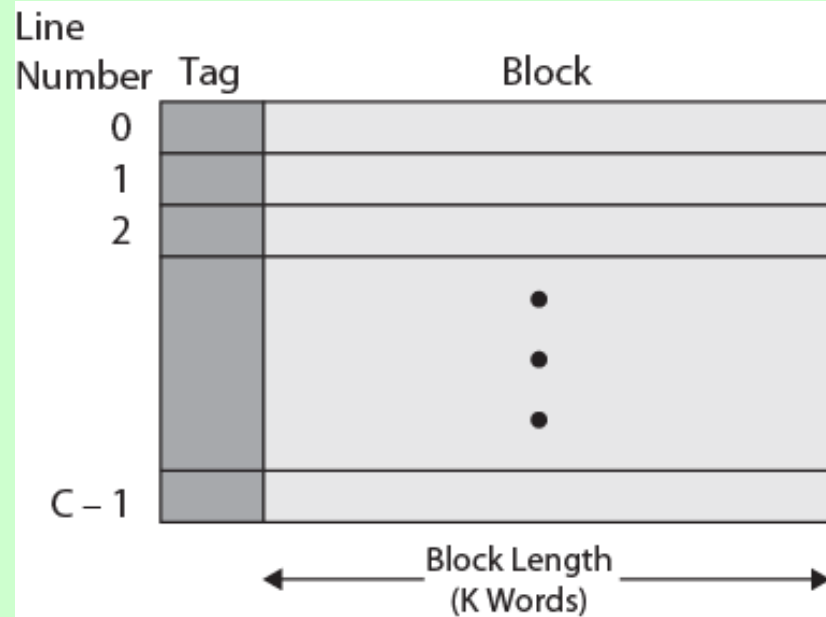


(a) Single cache

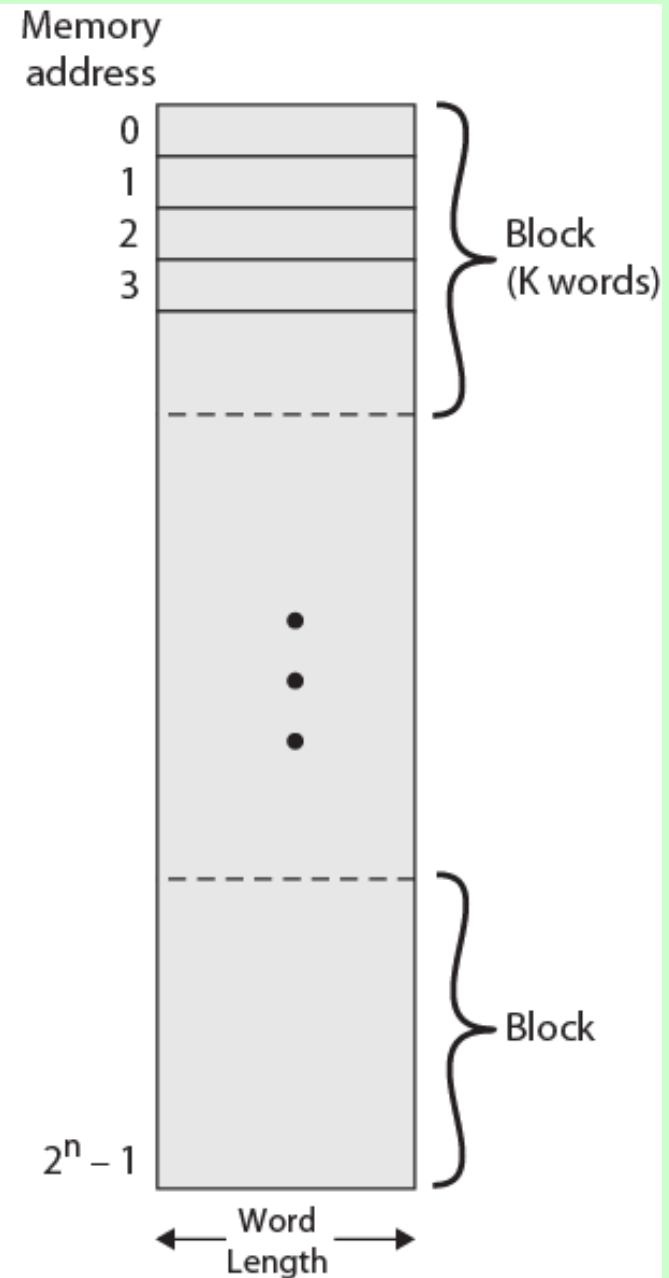


(b) Three-level cache organization

Cache/Main Memory Structure



(a) Cache



(b) Main memory

In referring to the **basic unit of the cache**, the term **line** is used, **rather than** the term **block**, for two reasons:

(1) to avoid confusion with a main memory block, which contains the same number of data words as a cache line; and

(2) because a cache line includes not only K words of data, just as a main memory block, but also include tag and control bits.

Cache operation – overview

- CPU requests contents of memory location
- Check cache for this data
- If present, get from cache (fast)
- If not present, read required block from main memory to cache
- Then deliver from cache to CPU
- **Cache includes tags to identify which block of main memory is in each cache slot**

Mapping Function

Because there are fewer cache lines than main memory blocks, an algorithm is needed for mapping main memory blocks into cache lines.

Further, a means is needed for determining which main memory block currently occupies a cache line. The choice of the mapping function dictates how the cache is organized. Three techniques can be used: direct, associative, and set associative

Mapping Function

- **Cache of 64kByte**
- **Cache block of 4 bytes**
 - i.e. cache is 16k (2^{14}) lines of 4 bytes
- **16MBytes main memory**
- **24 bit address**
 - ($2^{24}=16\text{M}$)

Direct Mapping

- 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
- Address is in two parts
- 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)

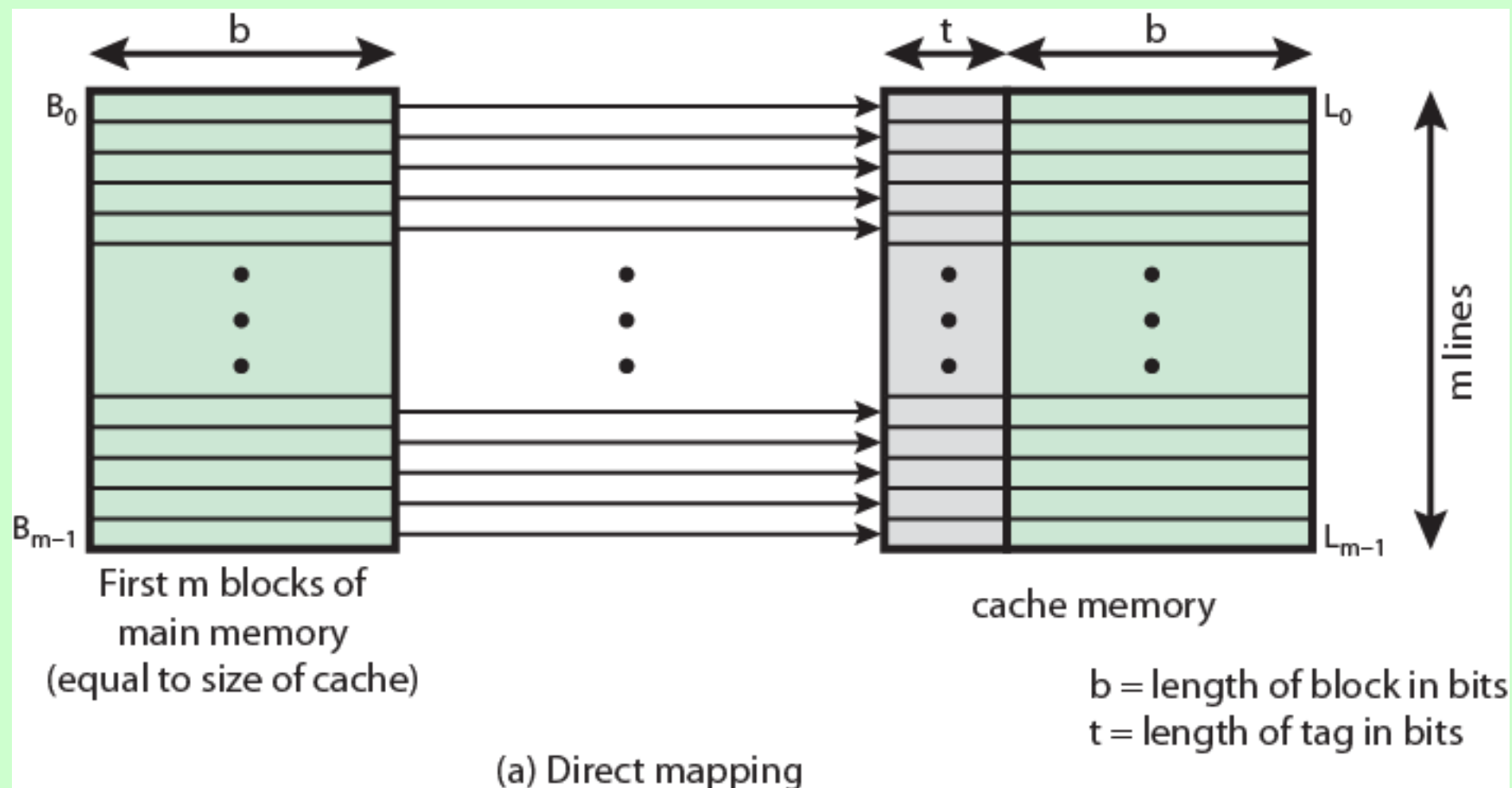
Direct Mapping

Address Structure

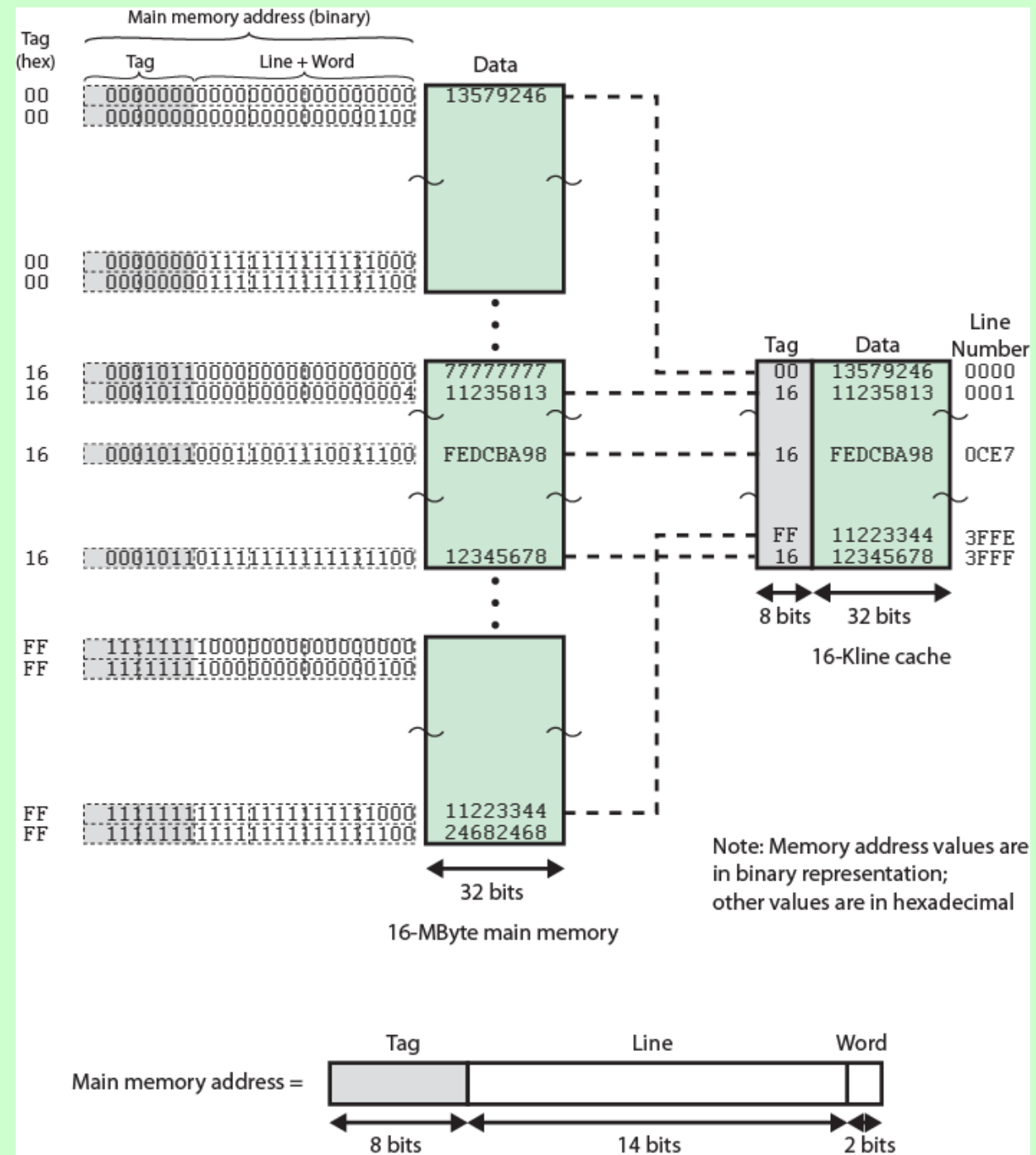
Tag s-r	Line or Slot r	Word w
8	14	2

- 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

Direct Mapping from Cache to Main Memory



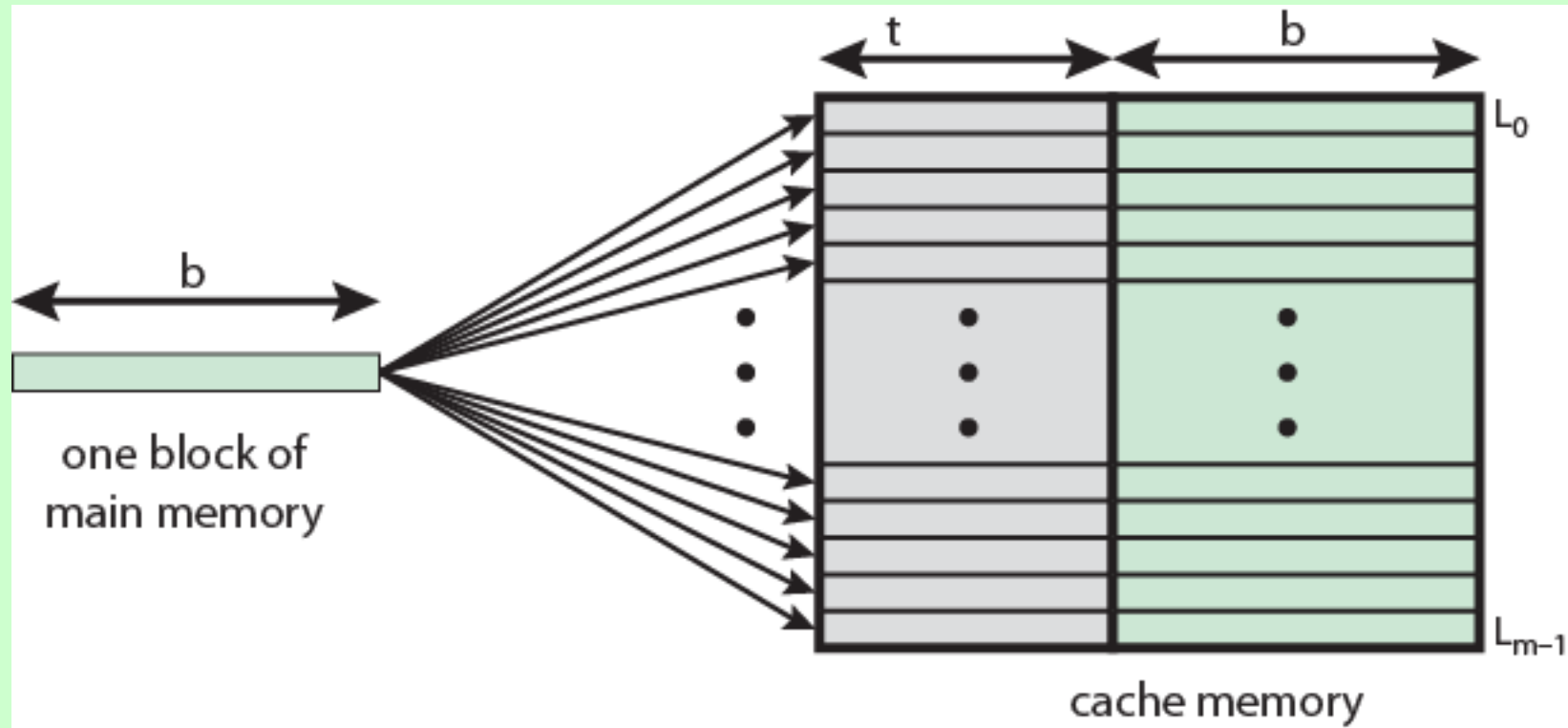
Direct Mapping Example



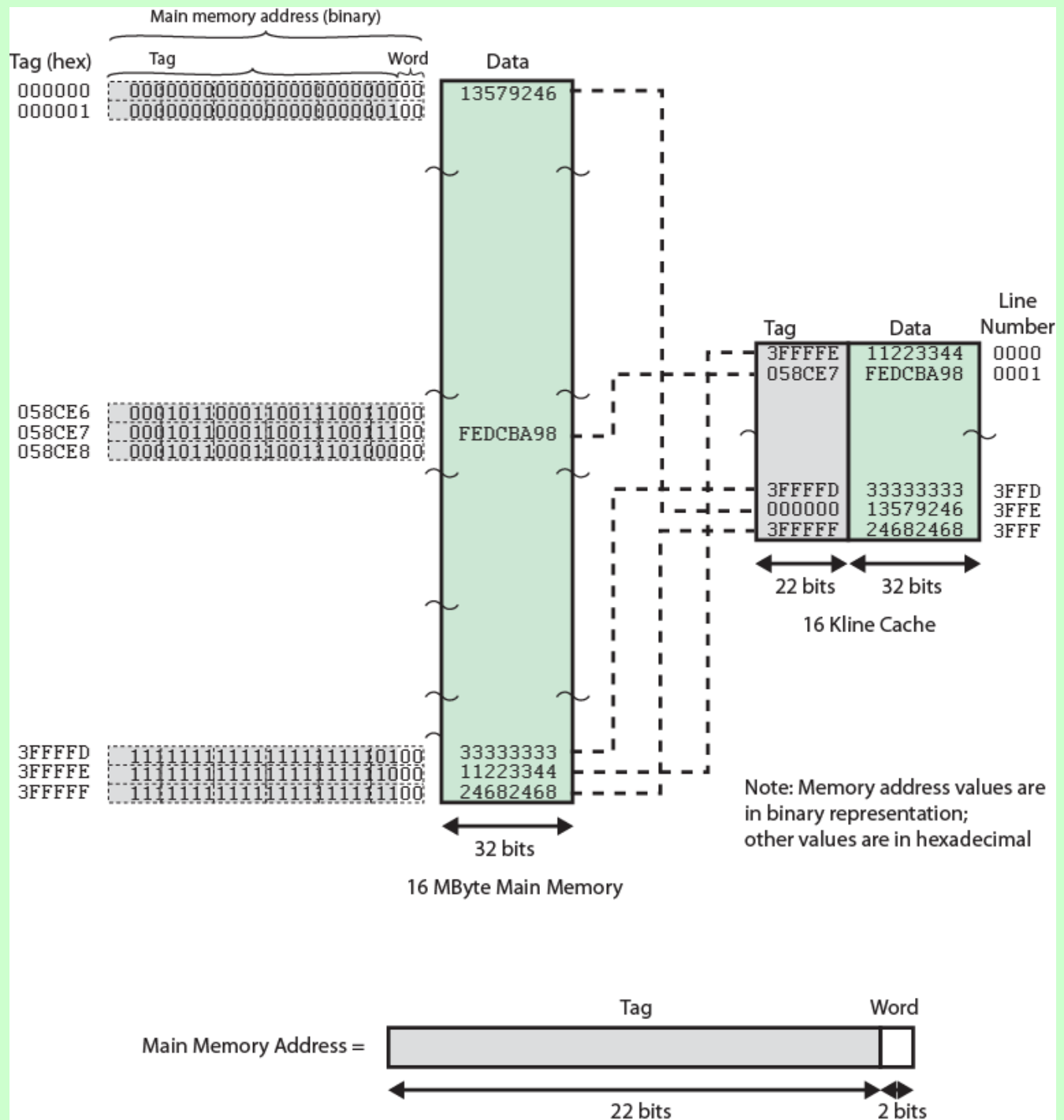
Associative Mapping

- A main memory **block can load into any line of cache**
- **Memory address** is interpreted as **tag and word**
- **Tag** uniquely identifies block of memory
- Every line's tag is examined for a match
- Cache searching gets expensive

Associative Mapping from Cache to Main Memory



Associative Mapping Example



Associative Mapping

Address Structure

Tag 22 bit	Word 2 bit
------------	---------------

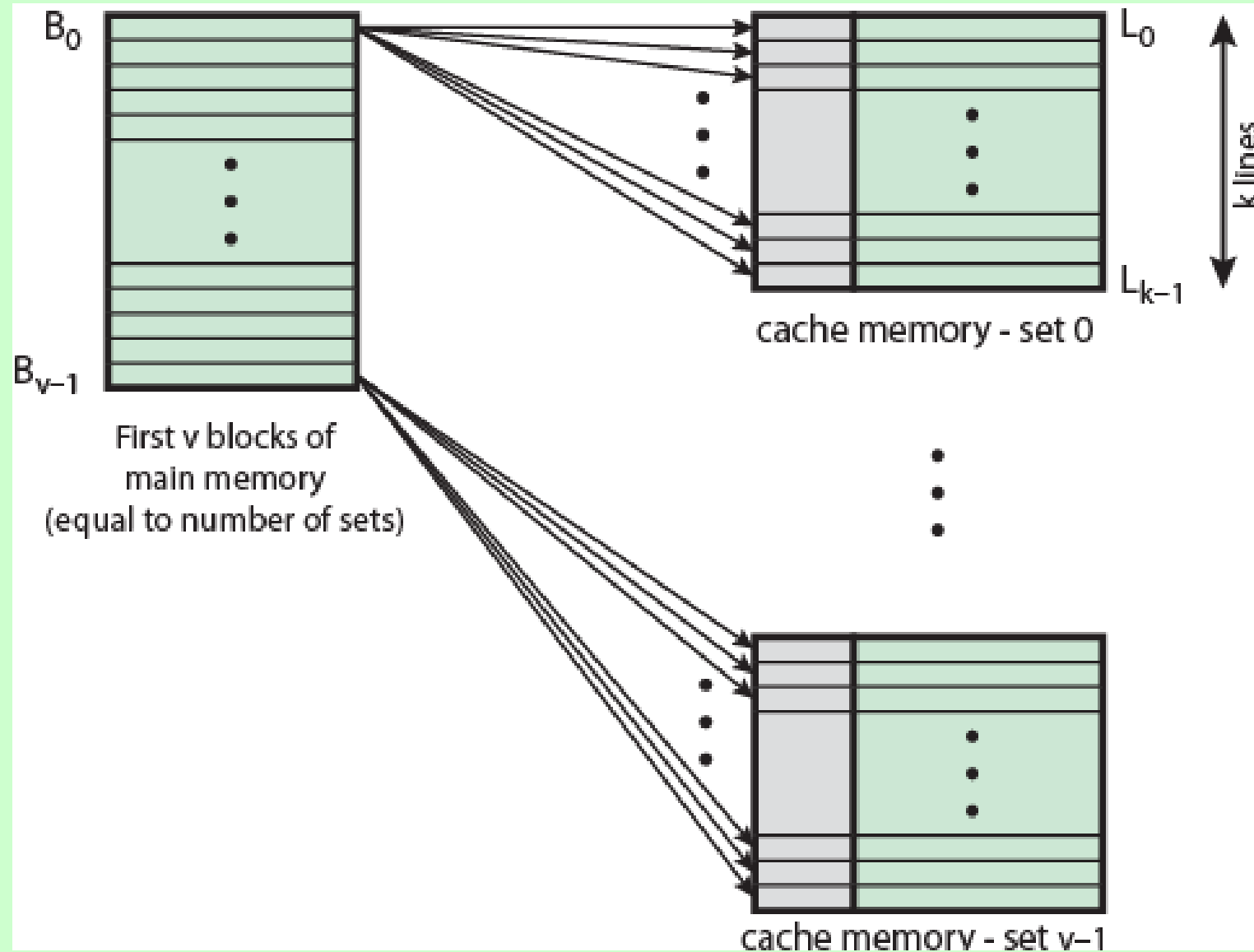
- **22 bit tag** stored with each 32 bit block of data
- Compare tag field with tag entry in cache to check for hit
- **Least significant 2 bits of address identify which 16 bit word is required from 32 bit data block**
- e.g.

— Address	Tag	Data	Cache line
— FFFFFC	FFFFFC24682468	3FFF	

Set Associative Mapping

- 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 B can be in any line of set i
- e.g. 2 lines per set
 - 2 way associative mapping
 - A given block can be in one of 2 lines in only one set

Mapping From Main Memory to Cache: v Associative



Set Associative Mapping

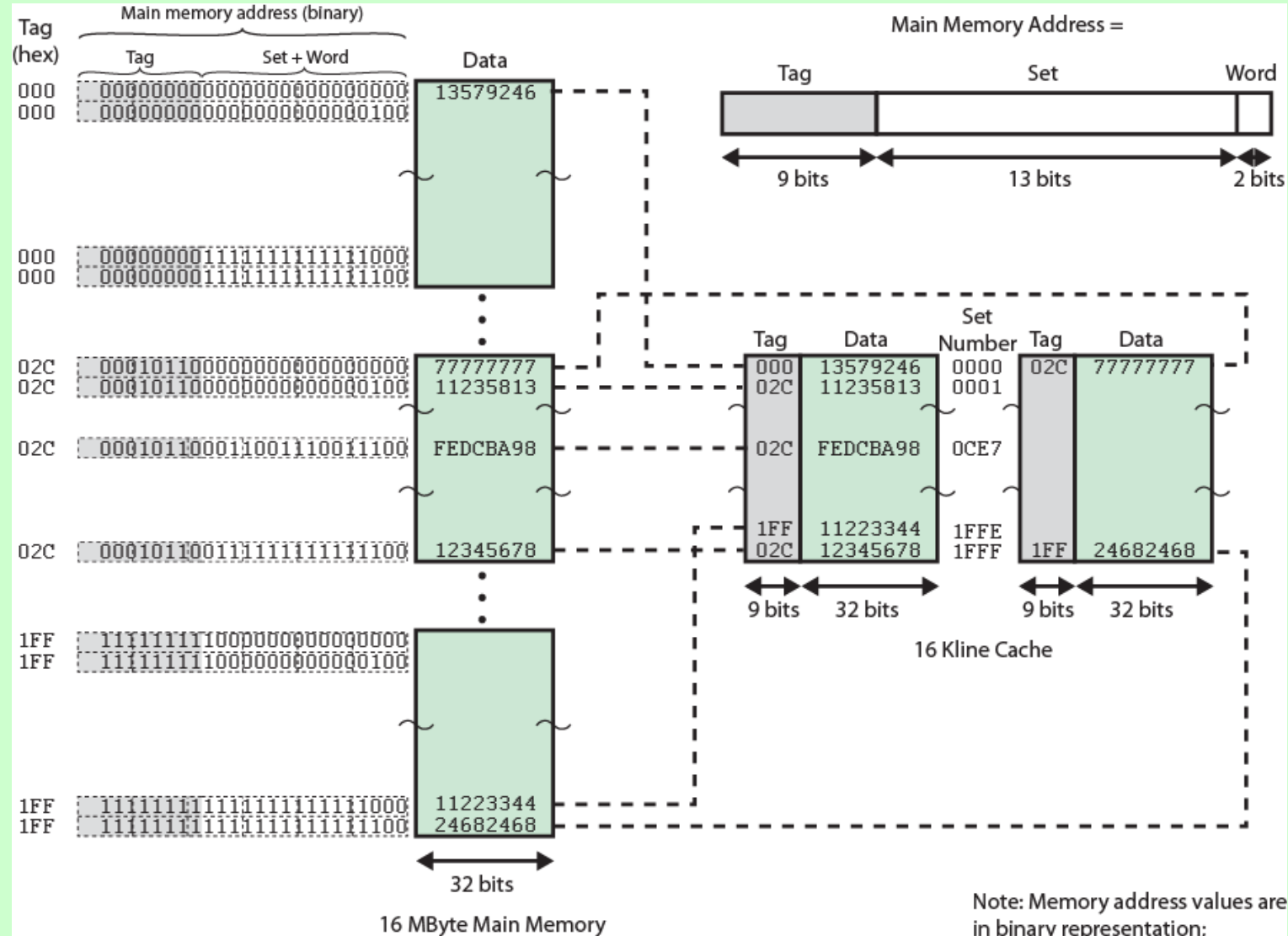
Address Structure

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

Two Way Set Associative Mapping Example



Note: Memory address values are in binary representation; other values are in hexadecimal

1. What are the differences among sequential access, direct access, and random access?

4.1 **Sequential access:** Memory is organized into units of data, called records. Access must be made in a specific linear sequence. **Direct access:** Individual blocks or records have a unique address based on physical location. Access is accomplished by direct access to reach a general vicinity plus sequential searching, counting, or waiting to reach the final location. **Random access:** Each addressable location in memory has a unique, physically wired-in addressing mechanism. The time to access a given location is independent of the sequence of prior accesses and is constant.



2. What is the general relationship among access time, memory cost, and capacity?

3. How does the principle of locality relate to the use of multiple memory levels?

4. What are the differences among direct mapping, associative mapping, and set-associative mapping?

- 4.2 Faster access time, greater cost per bit; greater capacity, smaller cost per bit; greater capacity, slower access time.
- 4.3 It is possible to organize data across a memory hierarchy such that the percentage of accesses to each successively lower level is substantially less than that of the level above. Because memory references tend to cluster, the data in the higher-level memory need not change very often to satisfy memory access requests.
- 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.

**5. For a direct-mapped cache, a main memory address is viewed as consisting of three fields.
List and define the three fields.**

**6. For an associative cache, a main memory address is viewed as consisting of two fields.
List and define the two fields.**

**7. For a set-associative cache, a main memory address is viewed as consisting of three fields.
List and define the three fields.**

- 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.

Problems

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.

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.

- 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 $8 \text{ kbytes} / 16 \text{ bytes} = 512$ 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 $64\text{-Mbyte} / 16 \text{ bytes} = 2^{22}$ 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

3. For the hexadecimal main memory addresses **111111**, **666666**, **BBBBBB**, show the following information, in hexadecimal format:

- a. **Tag**, **Line**, and **Word** values for a **direct-mapped** cache, using the format of **Figure 4.10**
- b. **Tag** and **Word** values for an **associative** cache, using the format of **Figure 4.12**
- c. **Tag**, **Set**, and **Word** values for a **two-way set-associative** cache, using the format of **Figure 4.15**

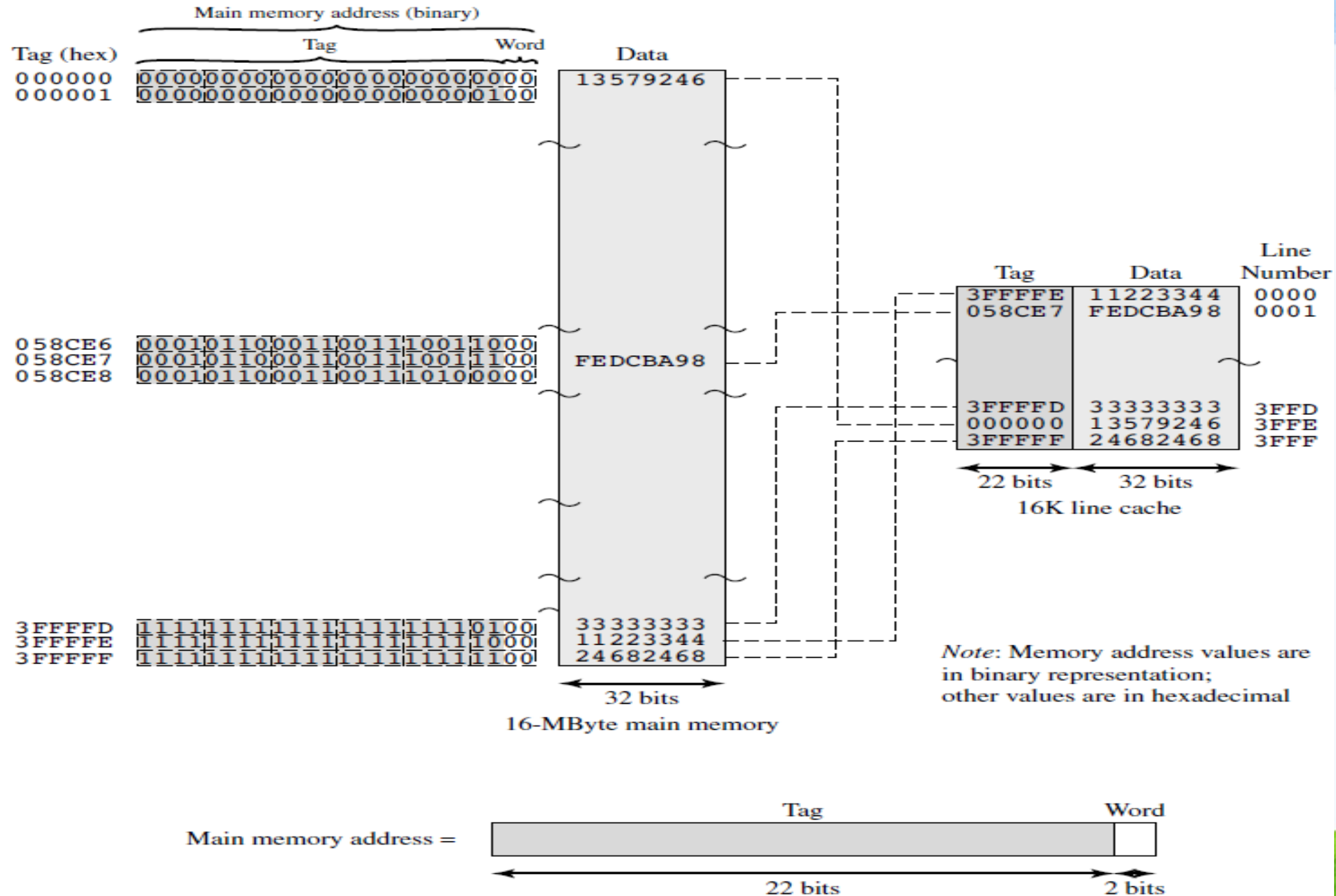


Figure 4.12 Associative Mapping Example

4.3

Address	111111	666666	BBBBBB
a. Tag/Line/Word	11/444/1	66/1999/2	BB/2EEE/3
b. Tag /Word	44444/1	199999/2	2EEEEEE/3
c. Tag/Set/Word	22/444/1	CC/1999/2	177/EEE/3

- 4. List the following values:**
- a. For the direct cache example of Figure 4.10:**
address length, number of addressable units, block size, number of blocks in main memory, number of lines in cache, size of tag
 - b. For the associative cache example of Figure 4.12:**
address length, number of addressable units, block size, number of blocks in main memory, number of lines in cache, size of tag
 - c. For the two-way set-associative cache example of Figure 4.15:** address length, number of addressable units, block size, number of blocks in main memory, number of lines in set, number of sets, number of lines in cache, size of tag

- 4.4
- a. Address length: 24; number of addressable units: 2^{24} ; block size: 4; number of blocks in main memory: 2^{22} ; number of lines in cache: 2^{14} ; size of tag: 8.
 - b. Address length: 24; number of addressable units: 2^{24} ; block size: 4; number of blocks in main memory: 2^{22} ; number of lines in cache: 4000 hex; size of tag: 22.
 - c. Address length: 24; number of addressable units: 2^{24} ; block size: 4; number of blocks in main memory: 2^{22} ; number of lines in set: 2; number of sets: 2^{13} ; number of lines in cache: 2^{14} ; size of tag: 9.

8. Consider a machine with a byte addressable main memory of 2^{16} bytes and block size of 8 bytes. Assume that a direct mapped cache consisting of 32 lines is used with this machine.

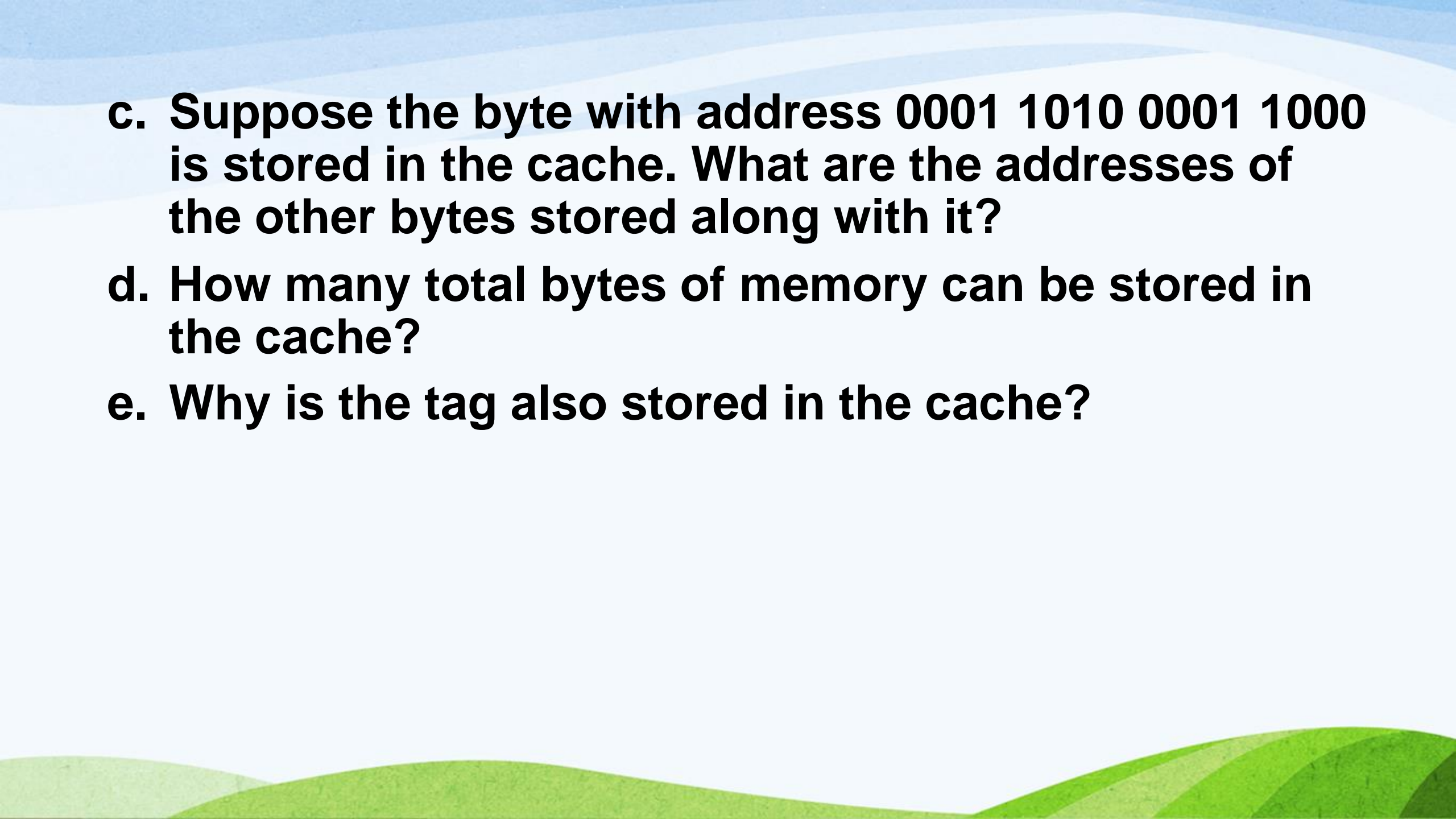
- a. How is a 16-bit memory address divided into tag, line number, and byte number?**
- b. Into what line would bytes with each of the following addresses be stored?**

0001 0001 0001 1011

1100 0011 0011 0100

1101 0000 0001 1101

1010 1010 1010 1010

- 
- c. Suppose the byte with address 0001 1010 0001 1000 is stored in the cache. What are the addresses of the other bytes stored along with it?**
 - d. How many total bytes of memory can be stored in the cache?**
 - e. Why is the tag also stored in the cache?**

- 4.8
- a. 8 leftmost bits = tag; 5 middle bits = line number; 3 rightmost bits = byte number
 - b. slot 3; slot 6; slot 3; slot 21
 - c. Bytes with addresses 0001 1010 0001 1000 through 0001 1010 0001 1111 are stored in the cache
 - d. 256 bytes
 - e. Because two items with two different memory addresses can be stored in the same place in the cache. The tag is used to distinguish between them.

12. Consider a computer with the following characteristics: total of 1Mbyte of main memory; word size of 1 byte; block size of 16 bytes; and cache size of 64 Kbytes.

- a. For the main memory addresses of F0010, 01234, and CABBE, give the corresponding tag, cache line address, and word offsets for a direct-mapped cache.**
- b. Give any two main memory addresses with different tags that map to the same cache slot for a direct-mapped cache.**
- c. For the main memory addresses of F0010 and CABBE, give the corresponding tag and offset values for a fully-associative cache.**
- d. For the main memory addresses of F0010 and CABBE, give the corresponding tag, cache set, and offset values for a two-way set-associative cache.**

4.12 a. Because the block size is 16 bytes and the word size is 1 byte, this means there are 16 words per block. We will need 4 bits to indicate which word we want out of a block. Each cache line / slot matches a memory block. That means each cache slot contains 16 bytes. If the cache is 64Kbytes then $64\text{Kbytes} / 16 = 4096$ cache slots. To address these 4096 cache slots, we need 12 bits ($2^{12} = 4096$). Consequently, given a 20 bit (1 MByte) main memory address:

Bits 0-3 indicate the word offset (4 bits)

Bits 4-15 indicate the cache slot (12 bits)

Bits 16-19 indicate the tag (remaining bits)

F0010 = 1111 0000 0000 0001 0000

Word offset = 0000 = 0

Slot = 0000 0000 0001 = 001

Tag = 1111 = F

01234 = 0000 0001 0010 0011 0100

Word offset = 0100 = 4

Slot = 0001 0010 0011 = 123

Tag = 0000 = 0

CABBE = 1100 1010 1011 1011 1110

Word offset = 1110 = E

Slot = 1010 1011 1011 = ABB

Tag = 1100 = C

- b. We need to pick any address where the slot is the same, but the tag (and optionally, the word offset) is different. Here are two examples where the slot is 1111 1111 1111

Address 1:

Word offset = 1111

Slot = 1111 1111 1111

Tag = 0000

Address = 0FFFF

Address 2:

Word offset = 0001

Slot = 1111 1111 1111

Tag = 0011

Address = 3FFF1

- c. With a fully associative cache, the cache is split up into a TAG and a WORDOFFSET field. We no longer need to identify which slot a memory block might map to, because a block can be in any slot and we will search each cache slot in parallel. The word-offset must be 4 bits to address each individual word in the 16-word block. This leaves 16 bits leftover for the tag.

F0010

Word offset = 0h

Tag = F001h

CABBE

Word offset = Eh

Tag = CABBh

- d. As computed in part a, we have 4096 cache slots. If we implement a two-way set associative cache, then it means that we put two cache slots into one set. Our cache now holds $4096/2 = 2048$ sets, where each set has two slots. To address these 2048 sets we need 11 bits ($2^{11} = 2048$). Once we address a set, we will simultaneously search both cache slots to see if one has a tag that matches the target. Our 20-bit address is now broken up as follows:

Bits 0-3 indicate the word offset

Bits 4-14 indicate the cache set

Bits 15-20 indicate the tag

F0010 = 1111 0000 0000 0001 0000

Word offset = 0000 = 0

Cache Set = 000 0000 0001 = 001

Tag = 11110 = 1 1110 = 1E

CABBE = 1100 1010 1011 1011 1110

Word offset = 1110 = E

Cache Set = 010 1011 1011 = 2BB

Tag = 11001 = 1 1001 = 19



Any Question?
Thank You!