SECR2033
Computer Organization and Architecture

Lecture slides prepared by “Computer Organization and Architecture”, 9/e, by William Stallings, 2013.

Module 6
Memory

Objectives:

- To master the concepts of hierarchical memory organization.
- To understand how each level of memory contributes to system performance, and how the performance is measured.
- To master the concepts behind cache memory and understands the basic concept of virtual memory.

Module 6
Memory

6.1 Introduction
6.2 Main Memory
6.3 Cache Memory
6.4 Virtual Memory
6.5 Summary

Module 6
Memory

6.1 Introduction
6.2 Main Memory
6.3 Cache Memory
6.4 Virtual Memory
6.6 Summary

- Overview
- Cache Mapping Schemes
- Replacement Policy
- Cache Performances

6 Overview

- **Cache memory* is designed to combine:
  - the memory access time of expensive, high-speed memory with
  - the large memory size of less expensive, lower-speed memory.
- *The cache contains a copy of portions of main memory.
- Unlike main memory, which is accessed by address, cache is typically accessed by content; hence, it is often called *content addressable memory*.

* William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.128

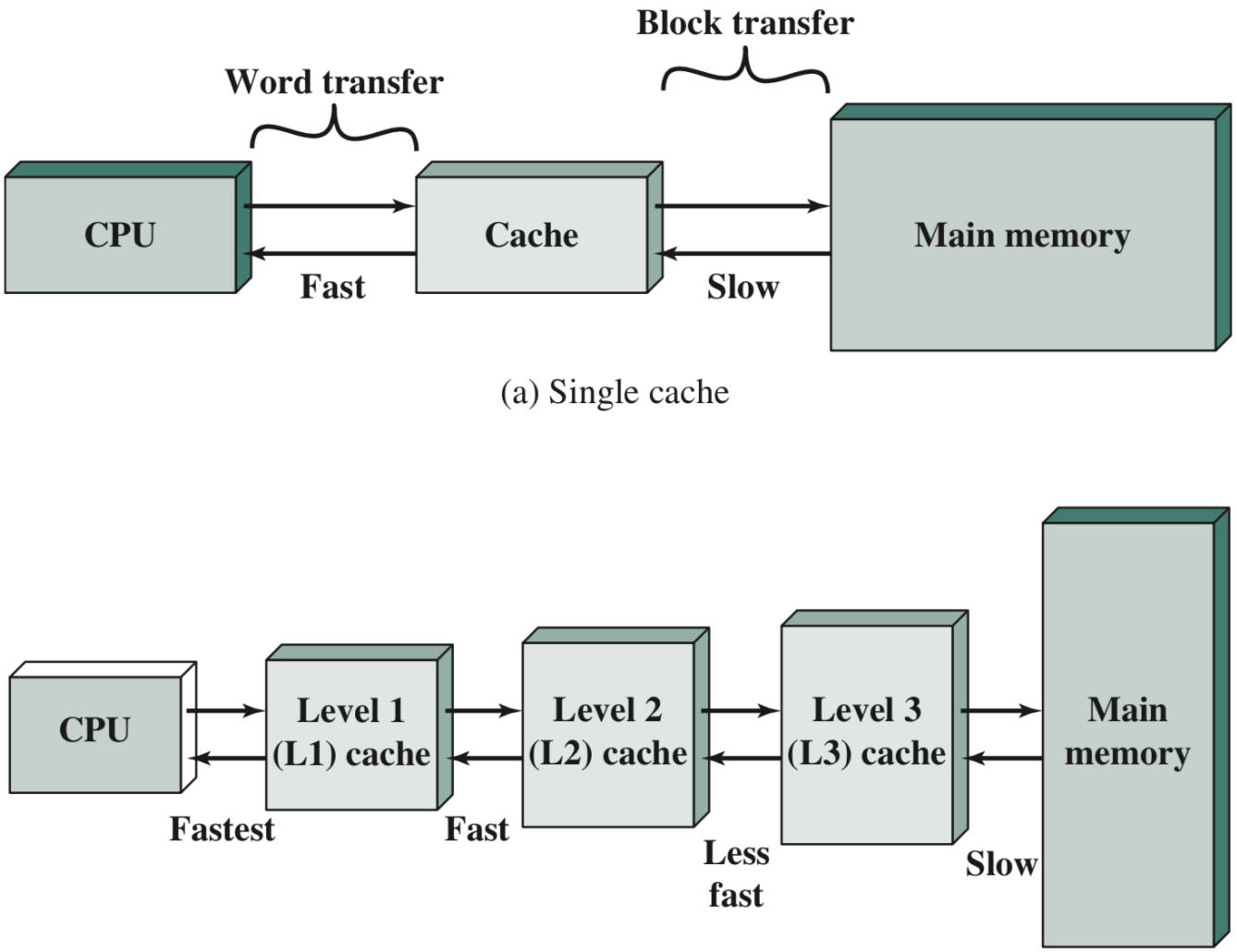


Figure: Cache and Main Memory

Cache Mapping Schemes

- The "content" that is addressed in *addressable cache memory* is a subset of the bits of a main memory address called a *field*.
- The fields into which a memory address is divided provide a many-to-one mapping between larger main memory and the smaller cache memory.

- Many *blocks* of main memory map to a single *block* of cache.
- A *tag* field in the cache block distinguishes one cached memory block from another.

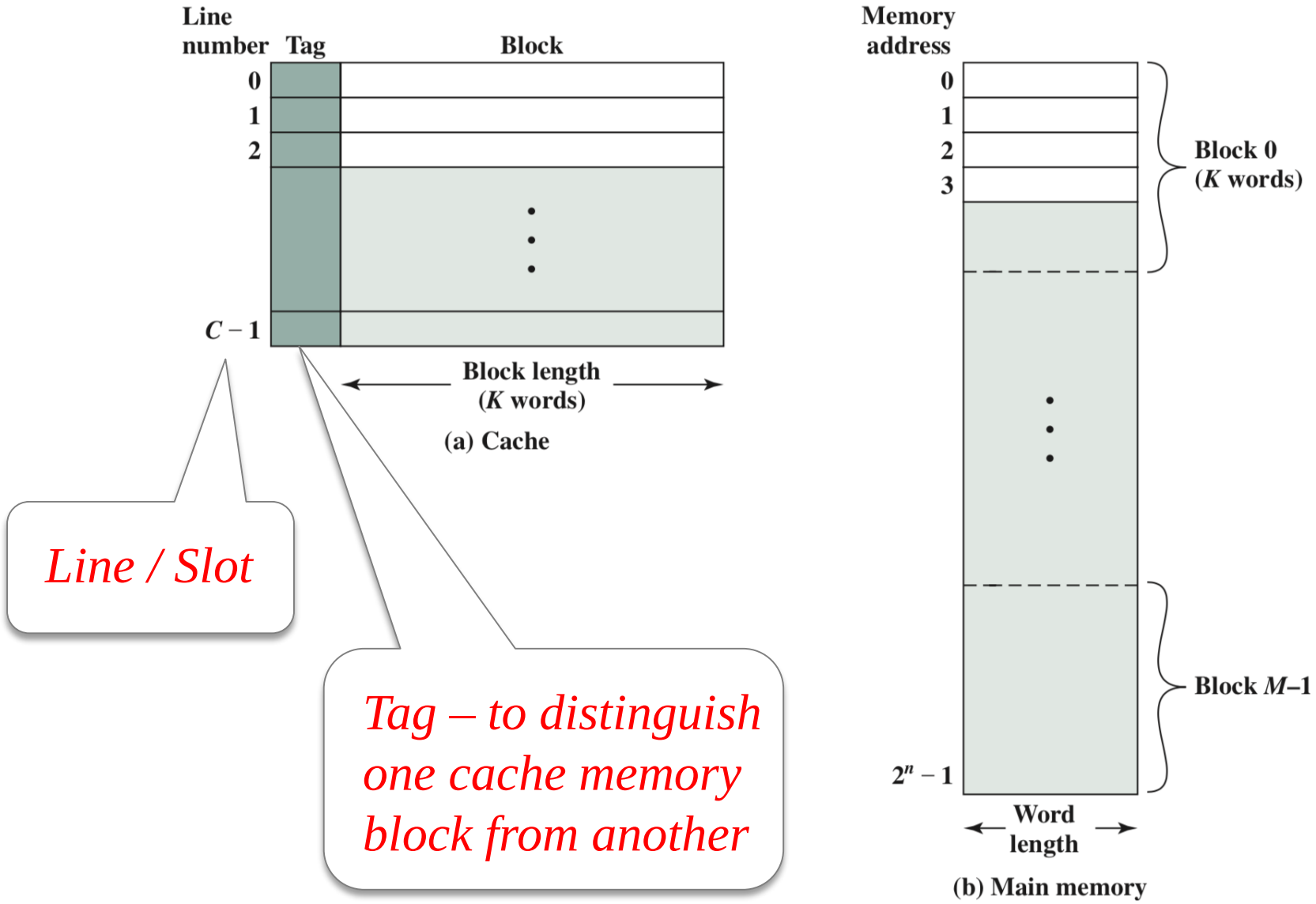
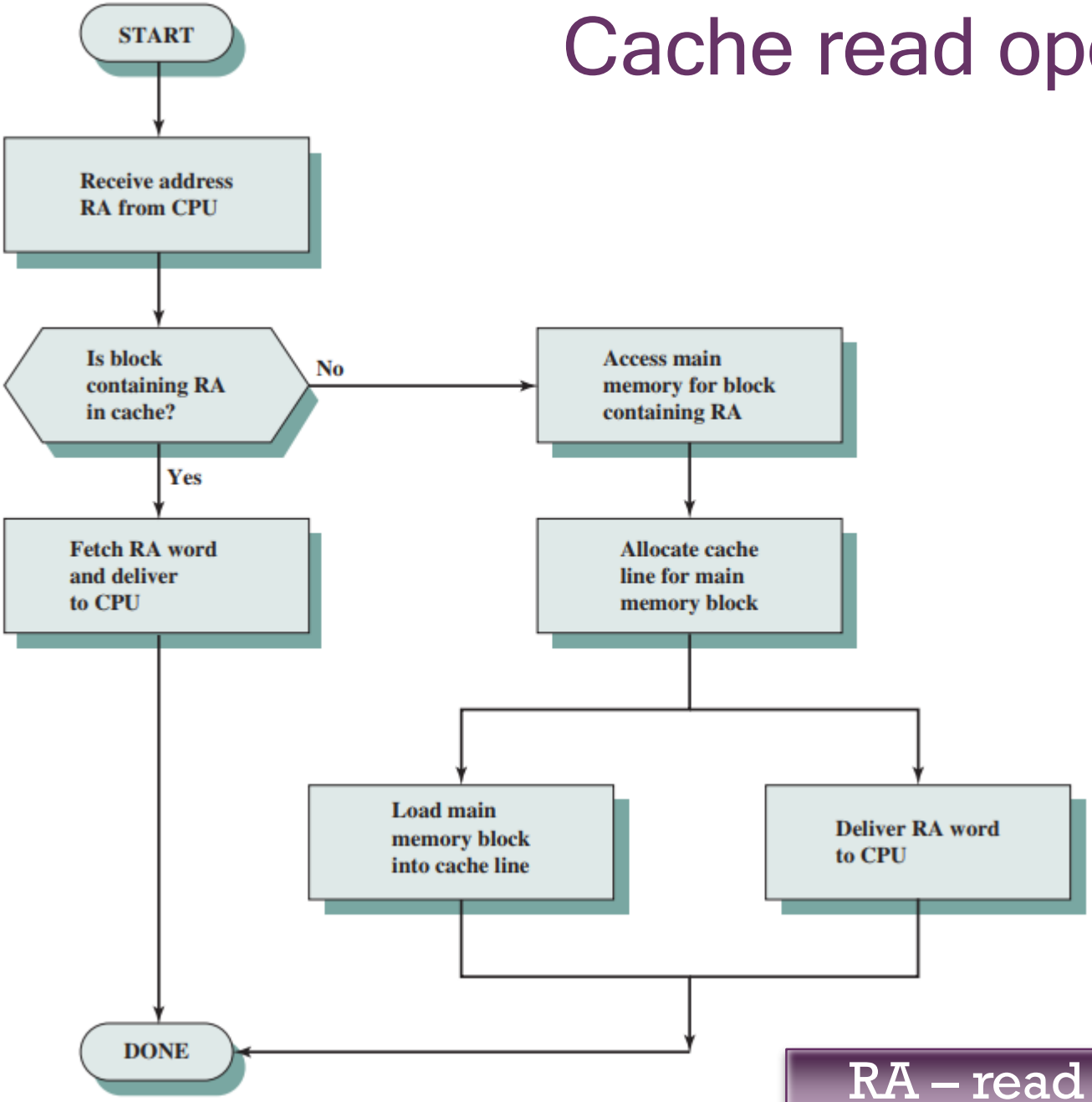


Figure: Cache/Main Memory Structure

Cache read operation

6



RA – read address

How to map main memory address to cache memory address?

- Depending on the mapping scheme, cache may has two or three fields.
- These fields depends on the particular mapping scheme being used to determine where the data is placed when it is originally copied into cache.

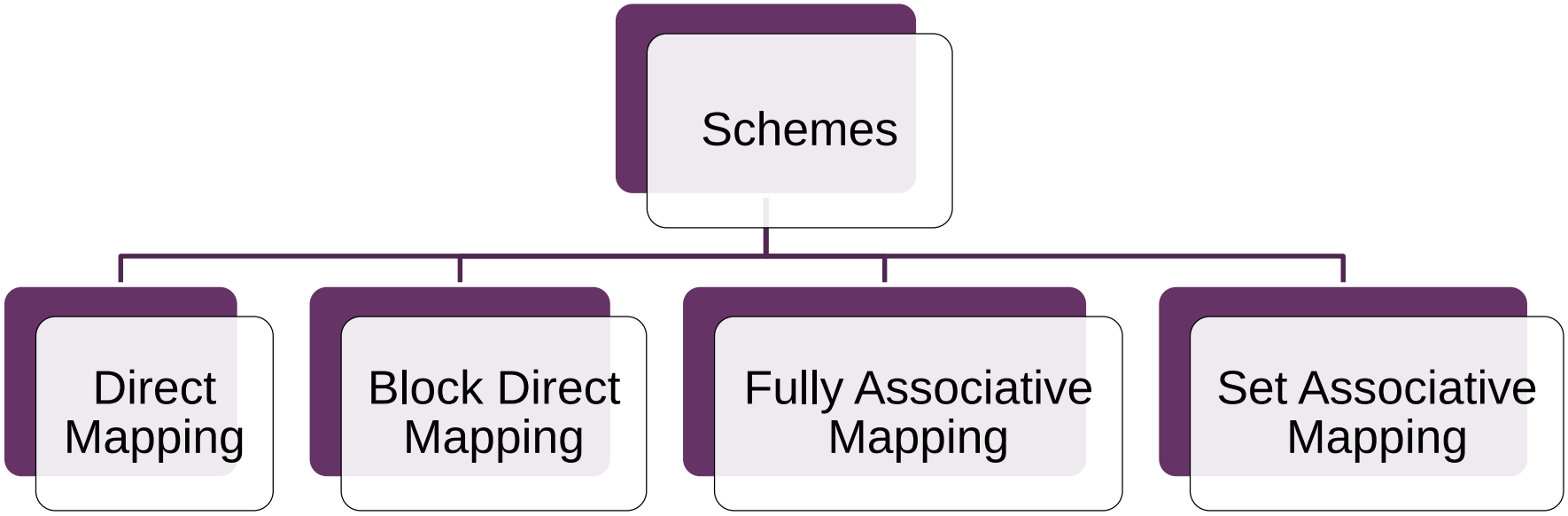


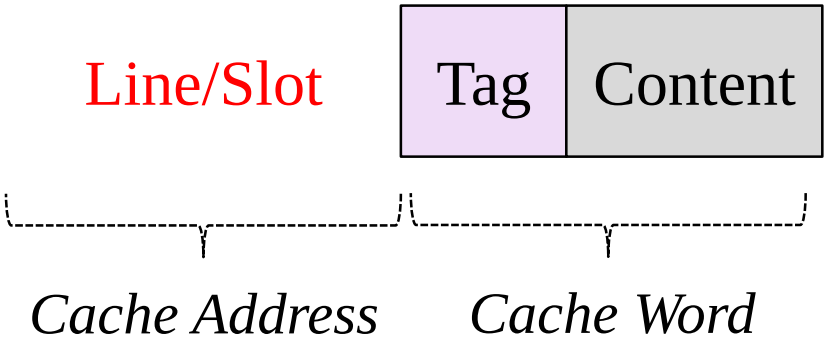
Figure: Cache mapping schemes

(a) Direct Mapping

- The simplest technique.
- *A cache structure in which each memory location is mapped to exactly one location in the cache.
- Address formats:

Main memory address :

Cache Memory :



*Patterson, D.A. and Hennessy, J.L. (2014). *Computer Organization and Design: The Hardware/Software Interface (5th Edition)*. United States: Elsevier, p.384

Example 7: Direct Mapping.

A main memory contains 8 words while the cache has only 4 words. Using direct address mapping, identify the fields of the main memory address.

Solution :

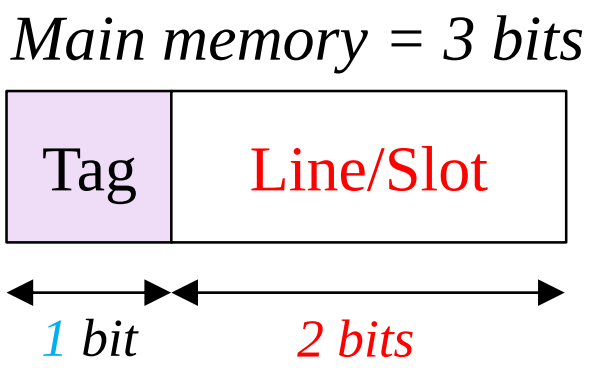
- Total memory words = 8 = 2^3

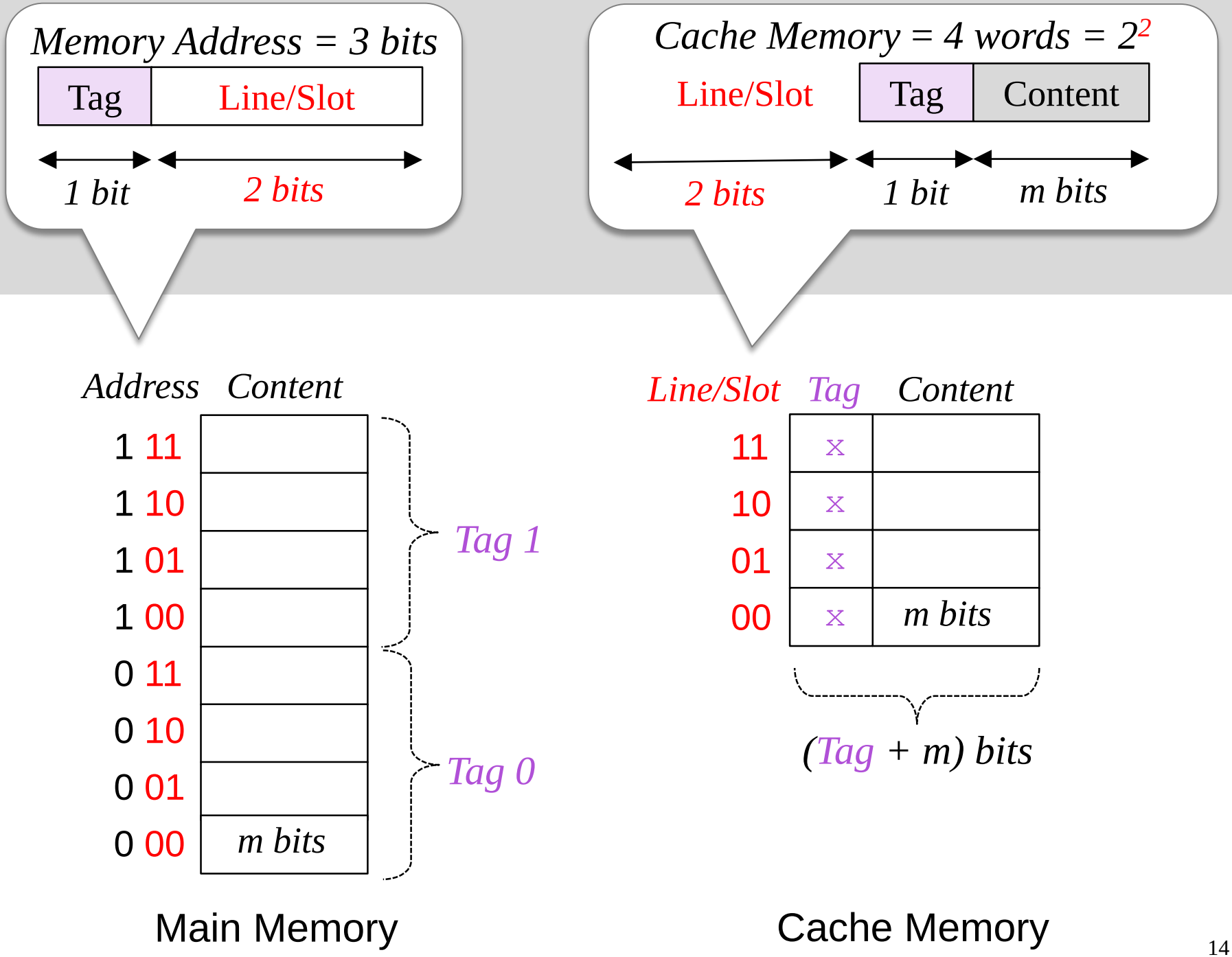
  → Require 3 bits for main memory address.

- Total cache words = 4 = 2^2

  → Require 2 bits for cache address (*Line/Slot*)

- Tag size = 3 – 2 = 1 bit




Memory Address = 3 bits
Tag
Line/Slot
1 bit
2 bits
Cache Memory = 4 words = 2^2
Line/Slot
Tag
Content
2 bits
1 bit
m bits
Address
Content
1 11
1 10
1 01
1 00
0 11
0 10
0 01
0 00
m bits
Tag 1
Tag 0
Line/Slot
Tag
Content
11
10
01
00
x
x
x
x
m bits
(Tag + m) bits
Main Memory
Cache Memory

Example 8: Direct Mapping.

A main memory contains 32 words while the cache has only 8 words. Using direct address mapping, identify the fields of the main memory address.

Solution :

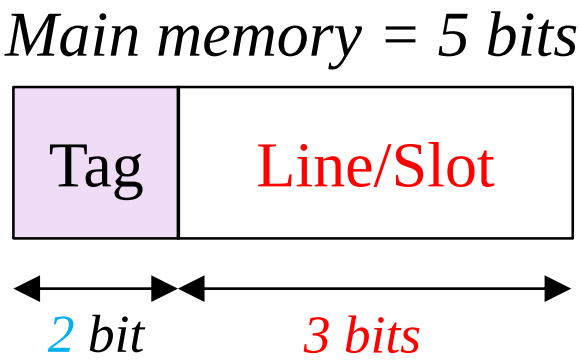
- Total memory words = 32 = 2^5

  → Require 5 bits for main memory address.

- Total cache words = 8 = 2^3

  → Require 3 bits for cache address (*Line/Slot*)

- Tag size = 5 – 3 = 2 bits



Example 9: Direct Mapping.

Size of cache memory is 64*Kword* and the size of main memory is 64*M* × 8 *bit* word. Determine the word size of main memory, cache and the main memory address format.

Solution :

- Total words in main memory = 64*M* = $2^6 \times 2^{20} = 2^{26}$

  → Require 26 bits for main memory address.

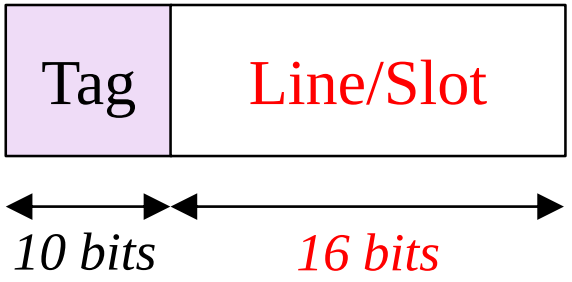
- Total cache words = 64*K* = $2^6 \times 2^{10} = 2^{16}$ → 16 bits for *Line/Slot*
- Tag size = 26 – 16 = 10 bits
- Size of main memory word = 8 *bits*

  → Size of cache word = Tag + (No. words in cache × size)

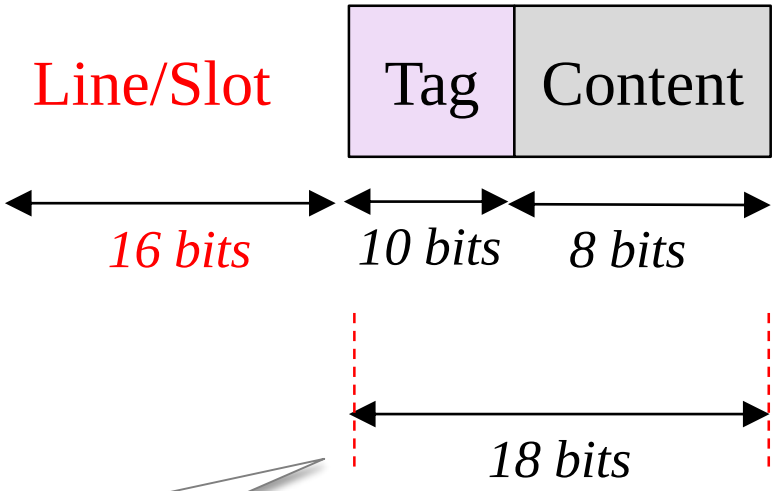
  = 10 + (1 × 8) = 18 *bits*

6

Main memory = 26 bits

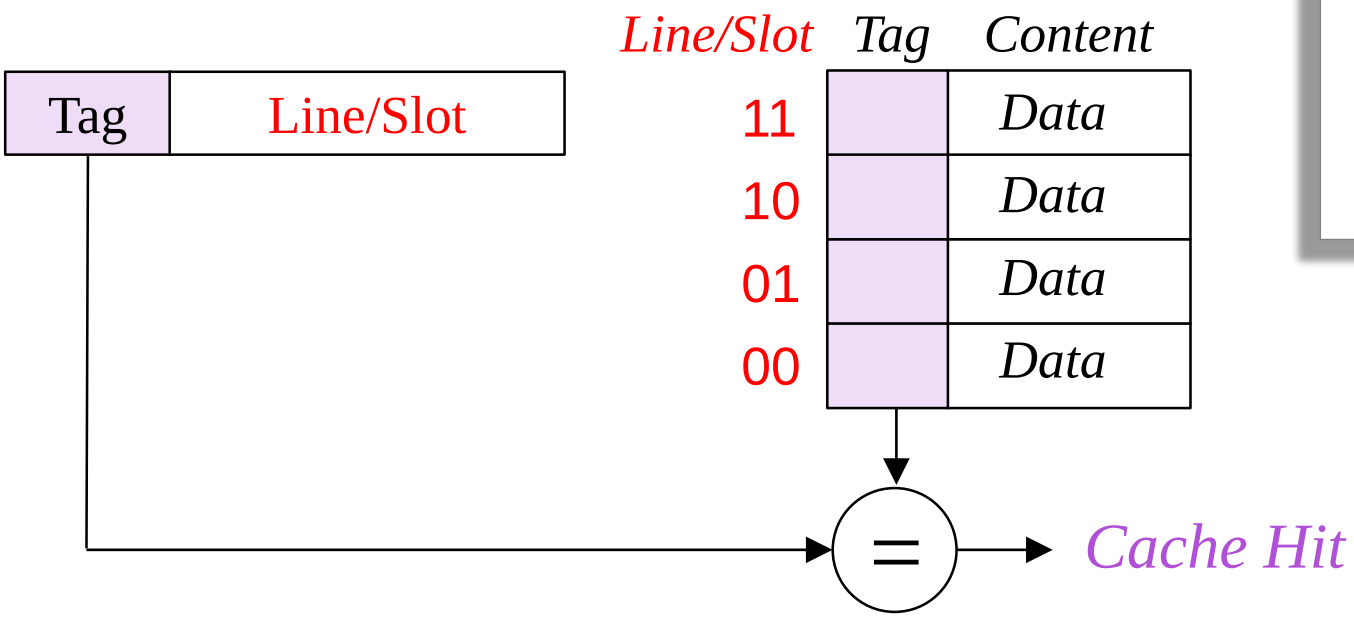


Cache Memory = 64K = 2^{16}



Cache word
= (Tag + size of content)
= 10 + 8 = 18 bits

Cache Hit and Miss



*Cache Miss

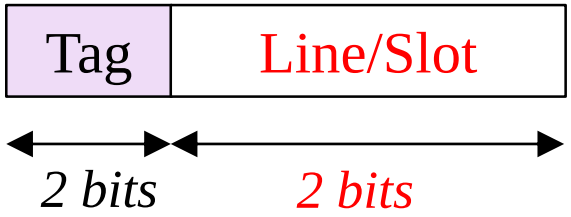
A request for data from the cache that cannot be filled because the data is not present in the cache.

Example 10:

Cache hit and miss.

A main memory contains 16 words while the cache has only 4 words. Using direct address mapping, identify the fields of the main memory address.

Main memory = 4 bits



Solution :

- Total memory words = 16 = 2^4
  - → Require 4 bits for main memory address.
- Total cache words = 4 = 2^2
  - → Require 2 bits for cache address (*Line/Slot*)
- Tag size = 4 – 2 = 2 bits

Dec/Hex.	*Line/Slot*	*Tag*	*Content*
3	11	xx	
2	10	xx	
1	01	xx	
0	00	xx	

Cache Memory

Based on previous example, consider the following main memory with contents.

Memory address:
→ (*Tag* | *Line/Slot*)

Dec/Hex.	*Line/Slot*	*Tag*	*Content*
3	11	xx	xx
2	10	xx	xx
1	01	xx	xx
0	00	xx	xx

Cache Memory

Hex.	*Address*	*Content (Hex)*
F	11 11	11
E	11 10	10
D	11 01	01
C	11 00	A1
B	10 11	F5
A	10 10	F4
9	10 01	F3
8	10 00	F1
7	01 11	FF
6	01 10	91
5	01 01	10
4	01 00	19
3	00 11	13
2	00 10	02
1	00 01	01
0	00 00	EE

Main Memory

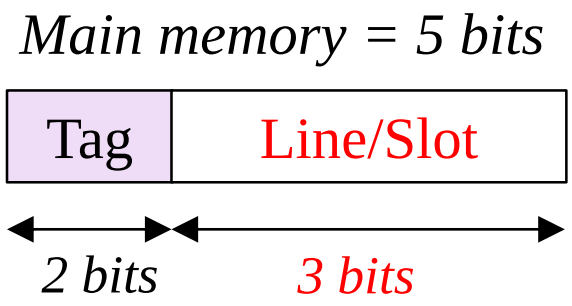
Example 11: Cache hit and miss operation.

(Read Process)

A main memory contains 32 words while the cache has only 8 words. Using direct address mapping, identify the fields of the main memory address.

Solution :

- Total memory words = 32 = 2^5 → 5 bits
- Total cache words = 8 = 2^3 → 3 bits (*Line/Slot*)
- Tag size = 5 – 3 = 2 bits



Memory address:
→ (*Tag* | *Line/Slot*)

Dec/Hex.	Line/Slot	Tag	Content
7	111	xx	xx
6	110	xx	xx
5	101	xx	xx
4	100	xx	xx
3	011	xx	xx
2	010	xx	xx
1	001	xx	xx
0	000	xx	xx

Cache Memory

Hex./Dec	Address	Content (Hex)
1F	31	11 111	11
1E	30	11 110	10
1D	29	11 101	01
1C	28	11 100	A1
1B	27	11 011	F5
1A	26	11 010	F4
19	25	11 001	F3
18	24	11 000	F1
17	23	10 111	FF
16	22	10 110	91
15	21	10 101	10
14	20	10 100	19
13	19	10 011	13
12	18	10 010	02
11	17	10 001	01
10	16	10 000	EE
…	…	…	…
3	3	00 011	A3
2	2	00 010	A1
1	1	00 001	AB
0	0	00 000	01

Main Memory

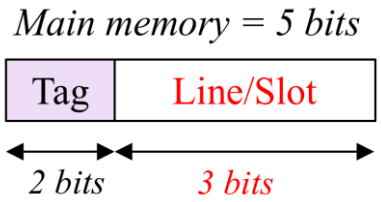
Based on the example, show the contents of the cache as it responds to a series of request (decimal address):

22, 26, 22, 26, 16, 3, 16, 18

Generated Address			Format Address	
Address		Content	Tag	Line/Slot
(Dec)	*(Binary)*	*(Hex)*		
22	10110	91	10	110
26	11010	F4	11	010
22	10110	91	10	110
26	11010	F4	11	010
16	10000	EE	10	000
3	00011	A3	00	011
16	10000	EE	10	000
18	10010	02	10	010

Main Memory

Complete the table by referring to previous slide



Based on the example, show the contents of the cache as it responds to a series of request (decimal address):

22, 26, 22, 26, 16, 3, 16, 18

Format Address		
Line/Slot	Tag	Content (Hex)
111	xx	xx
110	10	91
101	xx	xx
100	xx	xx
011	00	A3
010	~~11~~ 10	~~F4~~ 02
001	xx	xx
000	10	EE

Cache Memory

Cache memory after updating

Activity 3

Exercise 6.1:

Base on previous example, complete the following tables as they respond to a series of request (hexadecimal address):

1D, 14, 1E, 1D, 14, 17, 3, 1C

Format Address		
Line/Slot	Tag	Content (Hex)
111		
110		
101		
100		
011		
010		
001		
000		

Cache Memory

Format Address	
Tag	Line/Slot

Main Memory

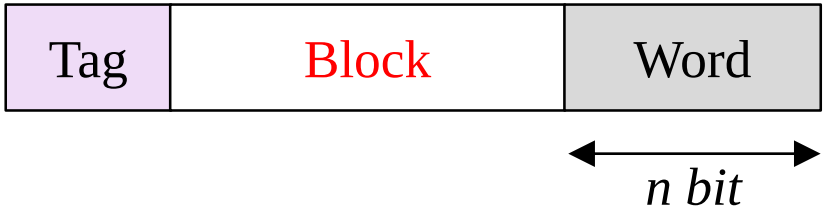
Read/Write operation cache				
Hit	Miss	Update cache	Read	Write

(b) Block Direct Mapping

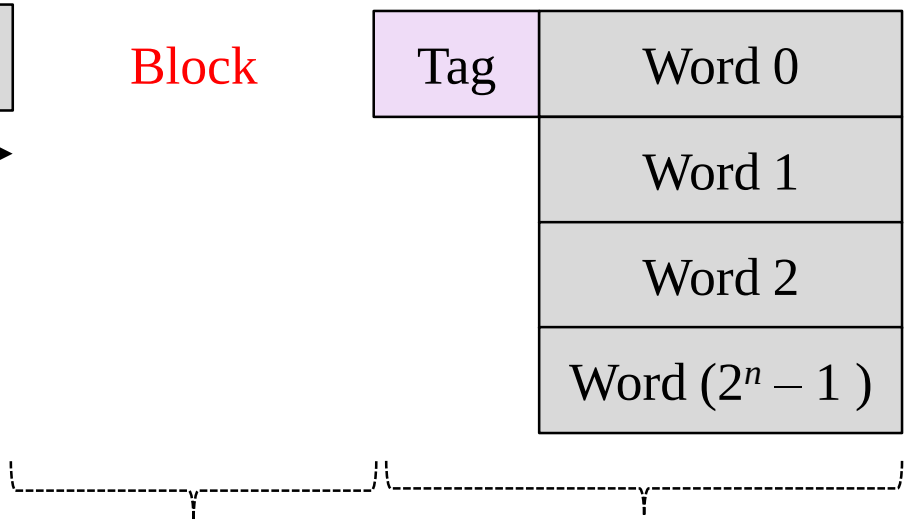
$$= 2^n - 1$$
$$= 2^2 - 1$$
$$= 4 - 1 = 3$$

- The simplest technique of direct mapping can maps each block of main memory into only one possible cache line.
- Address formats: (2^n words, *e.g.* n = 2)

Main memory address :



Cache Memory :



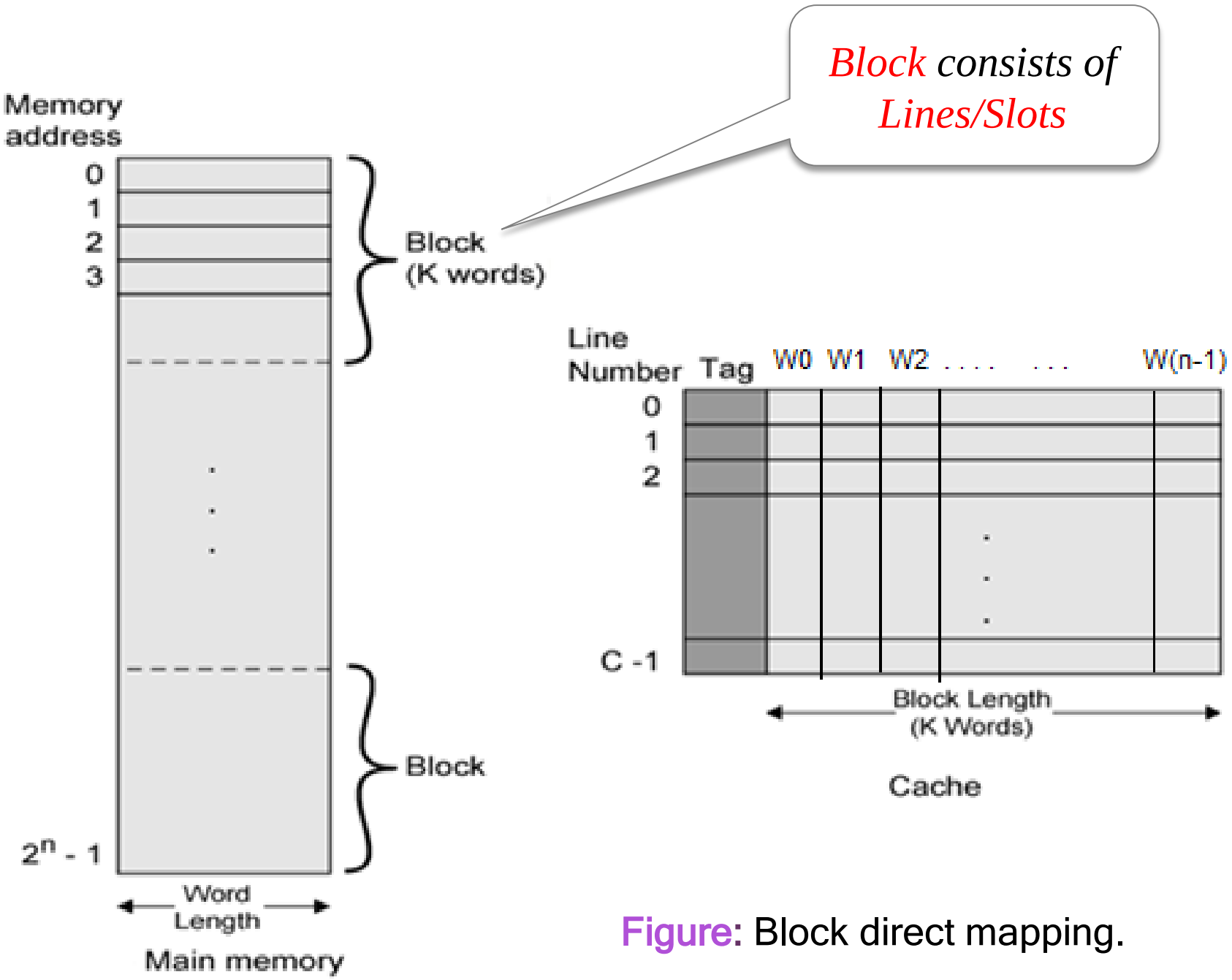
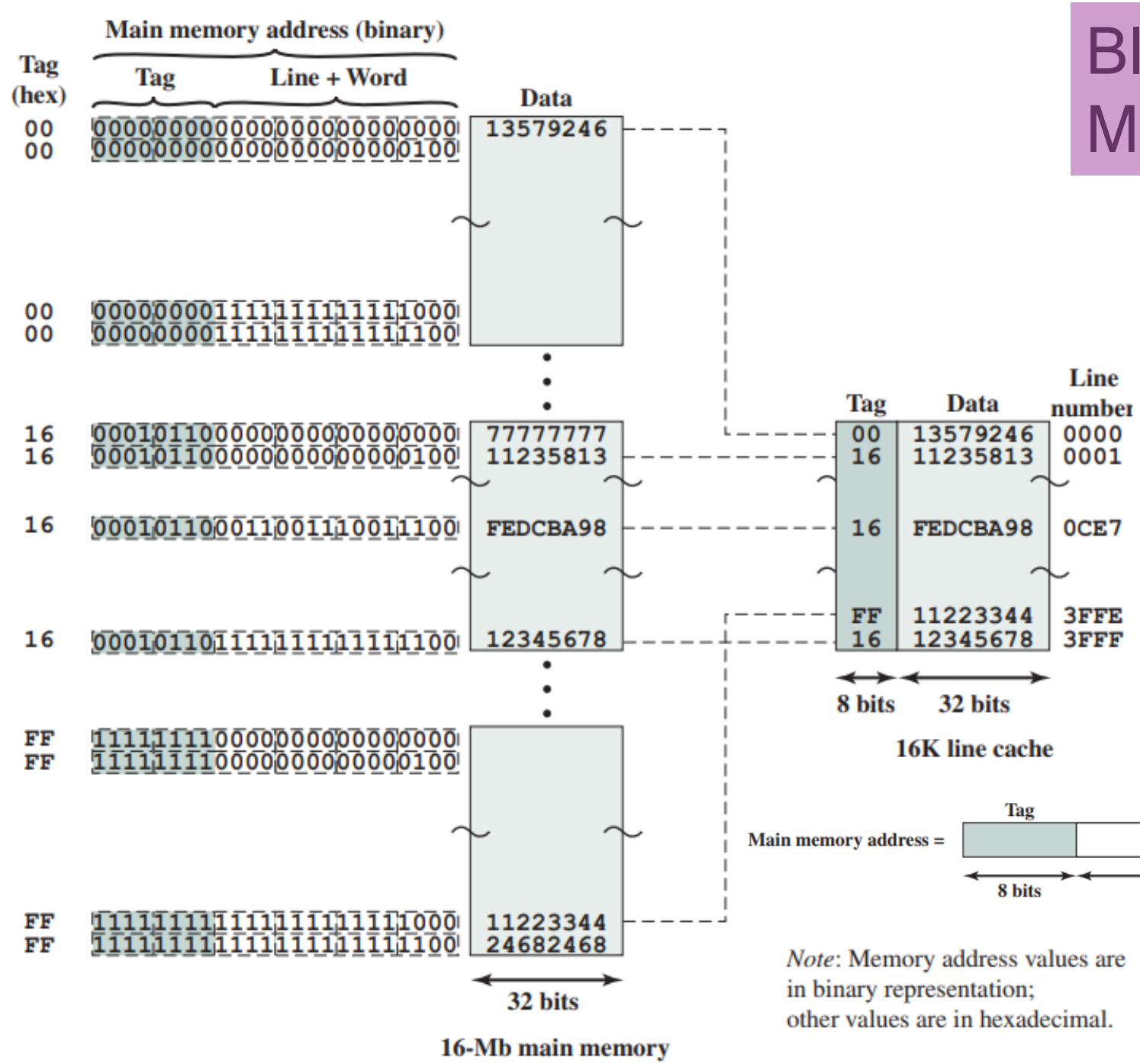


Figure: Block direct mapping.

Block Direct Mapping



Maps each block of main memory into only one possible cache line

22-bit (tag&line) is an index into the cache to access a particular line

2-bit word is to select one of the 4 bytes in that line

Main Memory 16-MByte

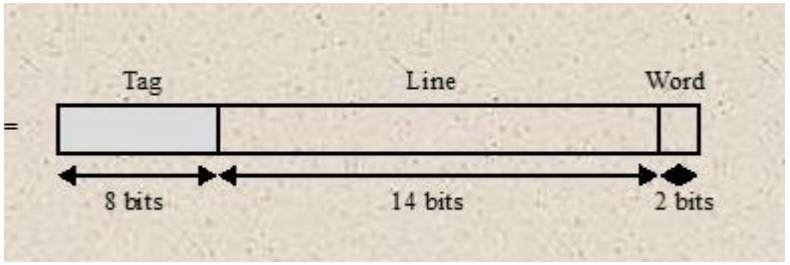
Main memory address	8 bits Data
0000 0000 0000 0000 0000 0000	13h
0000 0000 0000 0000 0000 0001	57h
0000 0000 0000 0000 0000 0010	92h
0000 0000 0000 0000 0000 0011	46h
0000 0000 0000 0000 0000 0100	24h
0000 0000 0000 0000 0000 0101	36h

1 block = 4 words

Main memory address	<-------- 32 bits --------> Data
0000 0000 0000 0000 0000 0000	13579246h
0000 0000 0000 0000 0000 0100	24364776h

Main memory address	WORD 00	01	10	11
0000 0000 0000 0000 0000 00	13h	57h	92h	46h
0000 0000 0000 0000 0000 01	24h	36h	47h	76h
0000 0000 0000 0000 0000 10				
0000 0000 0000 0000 0000 11				

- 16-MByte Main memory = $2^4 \times 2^{20} = 2^{24}$
  ➔ 24 bit address
- Each block is 4 words
  Word = 4 = 2^2
  ➔ 2 bits to represent every word



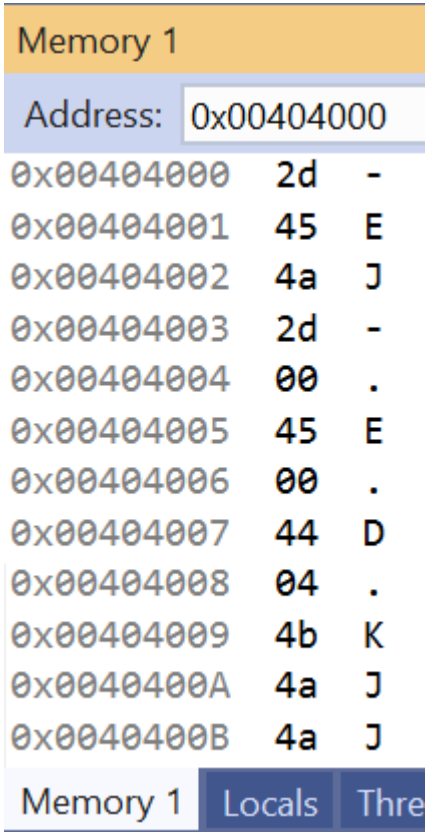
Main Memory 16-MByte

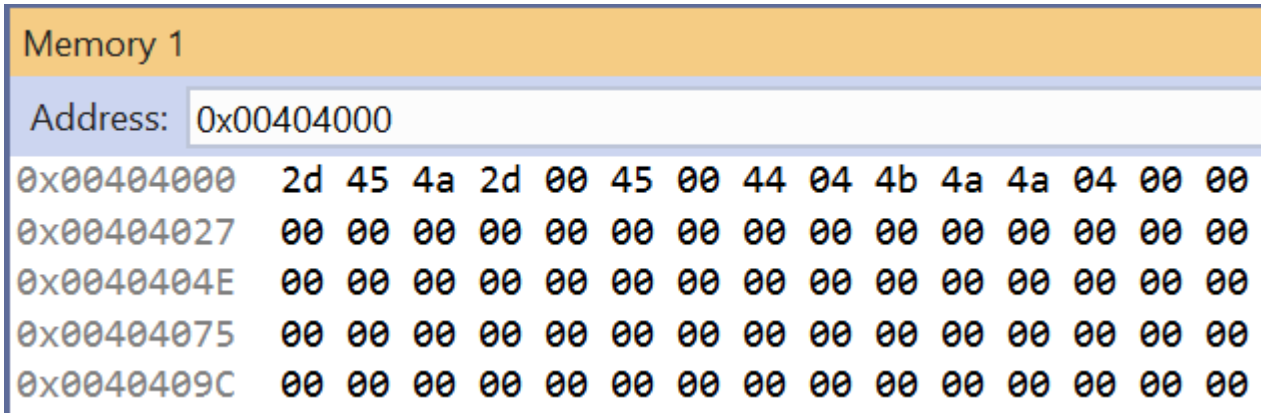
Main memory address	8 bits Data
0000 0000 0000 0000 0000 0000	13h
0000 0000 0000 0000 0000 0001	57h
0000 0000 0000 0000 0000 0010	92h
0000 0000 0000 0000 0000 0011	46h
0000 0000 0000 0000 0000 0100	24h
0000 0000 0000 0000 0000 0101	36h

1 block
= 4 words

<-------- 32 bits -------->

Main memory address	Data
0000 0000 0000 0000 0000 0000	13579246h
0000 0000 0000 0000 0000 0100	24364776h





Data stored in memory as 8 bits vs 32 bits in VS2019

Cache Memory 16-Kline

- Cache Memory = 16-Kline
  = 2^4 x 2^{10} = 2^{14} Line

  ➔ 14 bit for Line

- Main memory size
  = Tag + Line + Word

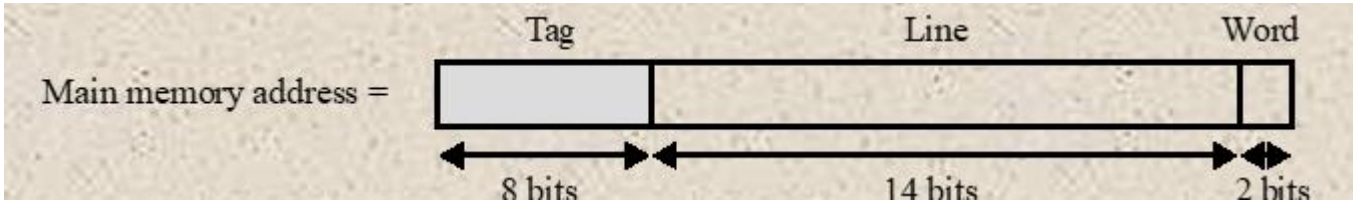
  Tag = 24 - 14 -2 = 8 bits

Main memory address		Data WORD			
Tag	Line	00	01	10	11
0000 0000	0000 0000 0000 00	13h	57h	92h	46h
0000 0000	0000 0000 0000 01	24h	36h	47h	76h
0000 0000	0000 0000 0000 10				
0000 0000	0000 0000 0000 11				

16-MByte Main Memory

Tag	Data WORD 00	01	10	11	Line Number	
0000 0000	13h	57h	92h	46h	0000 00 0000 0000	=0000h
0000 0000	24h	36h	47h	76h	0000 00 0000 0001	=0001h
					0000 00 0000 0010	=0010h
					0000 00 0000 0011	=0011h

16-Kline Cache



Addresses in Main Memory & Cache

8 bits

Main memory address	Data
0000 0000 0000 0000 0000 0000	13h
0000 0000 0000 0000 0000 0001	57h
0000 0000 0000 0000 0000 0010	92h
0000 0000 0000 0000 0000 0011	46h
0000 0000 0000 0000 0000 0100	24h
0000 0000 0000 0000 0000 0101	36h

- **Each address in main memory contains 1 word.**
- **Each address (line) in cache contains the whole block.**

-------- 32 bits --------

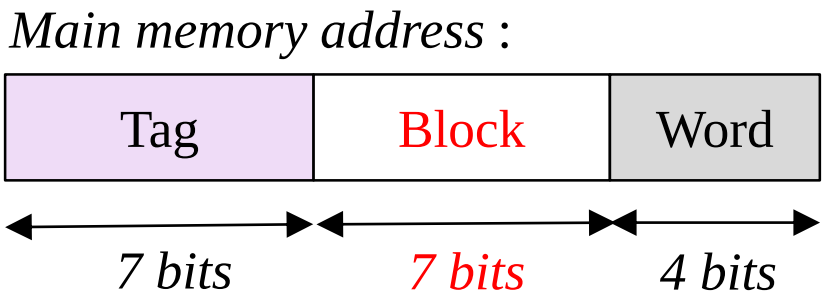
Main memory address	Data Block
0000 0000 0000 0000 0000 0000	13579246h
0000 0000 0000 0000 0000 0100	24364776h

16-MByte Main Memory

Tag	Data Block				Line Number
0000 0000	13h	57h	92h	46h	0000 00 0000 0000
0000 0000	24h	36h	47h	76h	0000 00 0000 0001
					0000 00 0000 0010
					0000 00 0000 0011

16-Kline Cache

Example 12:

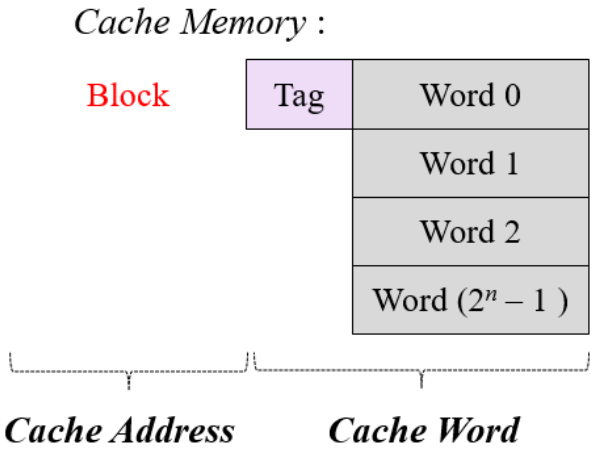


Given the main memory address format using block direct mapping, if each main memory word is 8 bits, calculate:

a) The main memory capacity.

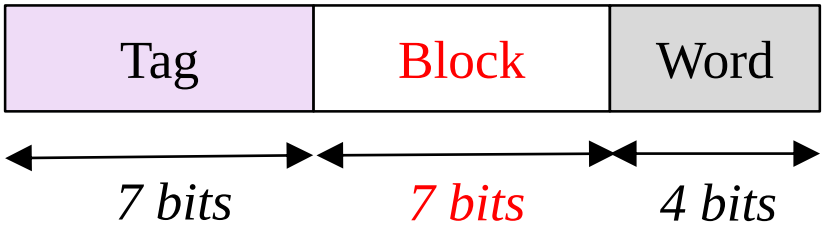
b) Total cache blocks.

c) The size of cache words.



Solution :

Main memory address :



a) Total main memory bits = 7 + 7 + 4 = 18 bits

→ Total memory words = 2^{18} = 256*Kword*

→ Total memory capacity = 256*Kword x* 8 *bits* = 2*Mbit*

b) Block = 7 bits

→ Total cache words = 2^7 → 128 words (*Block*)

c) Total words = 2^4 = 16 words

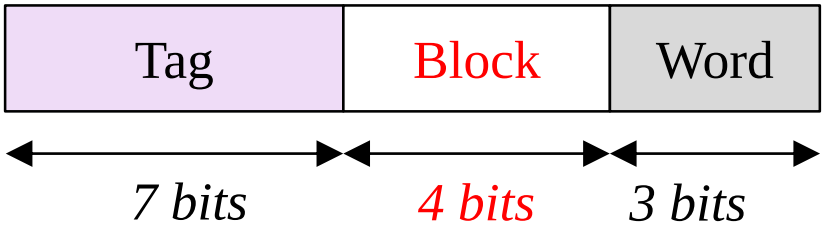
→ Cache word size = Tag + (No. of words in cache × size)

= 7 + (16 × 8)

= 132 bits

Example 13:

Main memory address :



Consider the main memory address.

Suppose a program generates the address 1AA.

→ In 14-bit binary, this number is: 00 0001 1010 1010

Padding with bit 0 at MSB to complete 14-bit

- ✓ The first 7 bits of the address go in the tag field.
- ✓ The next 4 bits go in the block field.
- ✓ The final 3 bits indicate the word within the block.

Main memory address :

0000011	0101	010

Main memory address :

1AA	0000011	0101	010
	Tag	*Block*	*Word*

Example 14: (Cache hit)

Supposed the program generates the subsequent address 1AB.

→ 1AB : 00 0001 1010 1011

It will find the data it is looking for in cache memory in block 0101, *word* 011.

Cache Memory :

0101	0000011	000	001	010	011	100	101	110	111

Block *Tag* *Words* ($2^3 = 8$)

Main memory address :

1AA	0000011	0101	010
	Tag	*Block*	*Word*

Example 15: (Cache miss)

Supposed the program generates the address 3AB.

→ In 14-bit binary, this number is: 00 0011 1010 1011

A miss occurred (tag is different).

The tag of cache memory for block 0101, must be replaced with 0000111.

Cache Memory :

0101	0000111	000	001	010	011	100	101	110	111

Block *Tag* *Words* ($2^3 = 8$)

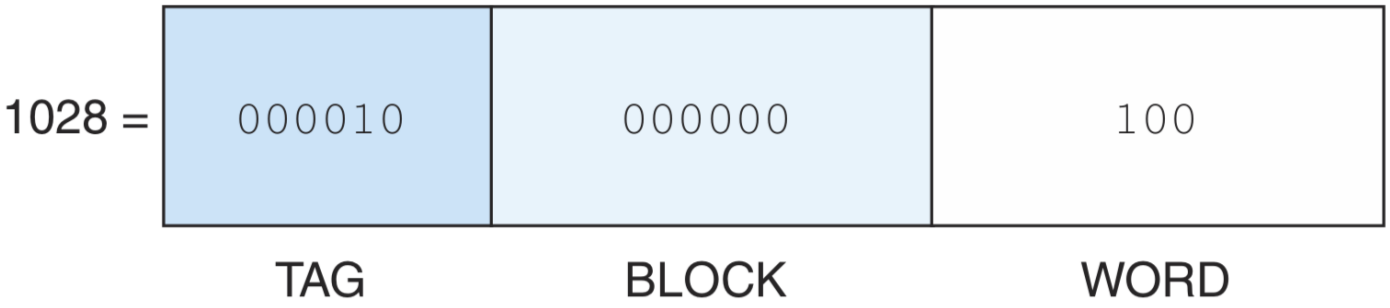
Activity 4

Exercise 6.2:

Suppose a system using 15-bit main memory addresses and 64 blocks of cache with each block contains 8 words. If the CPU generates the main memory address of 1028 (decimal), draw:

a) The main memory addressing format

b) The cache memory with all words fields.



Block Direct Mapping:

Advantages and Disadvantages

Other cache mapping schemes are designed to prevent this kind of thrashing.

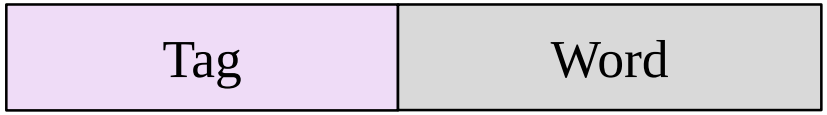
The technique is simple and inexpensive to implement.

- A fixed cache location for any given block. If a program accesses 2 line/slot/block that map to same line repeatedly, cache hit ratio will be low (Known as *thrashing*).
- Least effective in its utilization - that is, it may leave some cache lines unused because a given line/slot has fixed location.

(c) Fully Associative Mapping

- Overcomes the disadvantage of direct mapping by permitting each main memory block to be loaded into any location (line/slot/block) of the cache.
- A memory address is partitioned into only two fields:

Main memory address :



- With associative mapping, there is flexibility as to which block to replace when a new block is read into the cache.

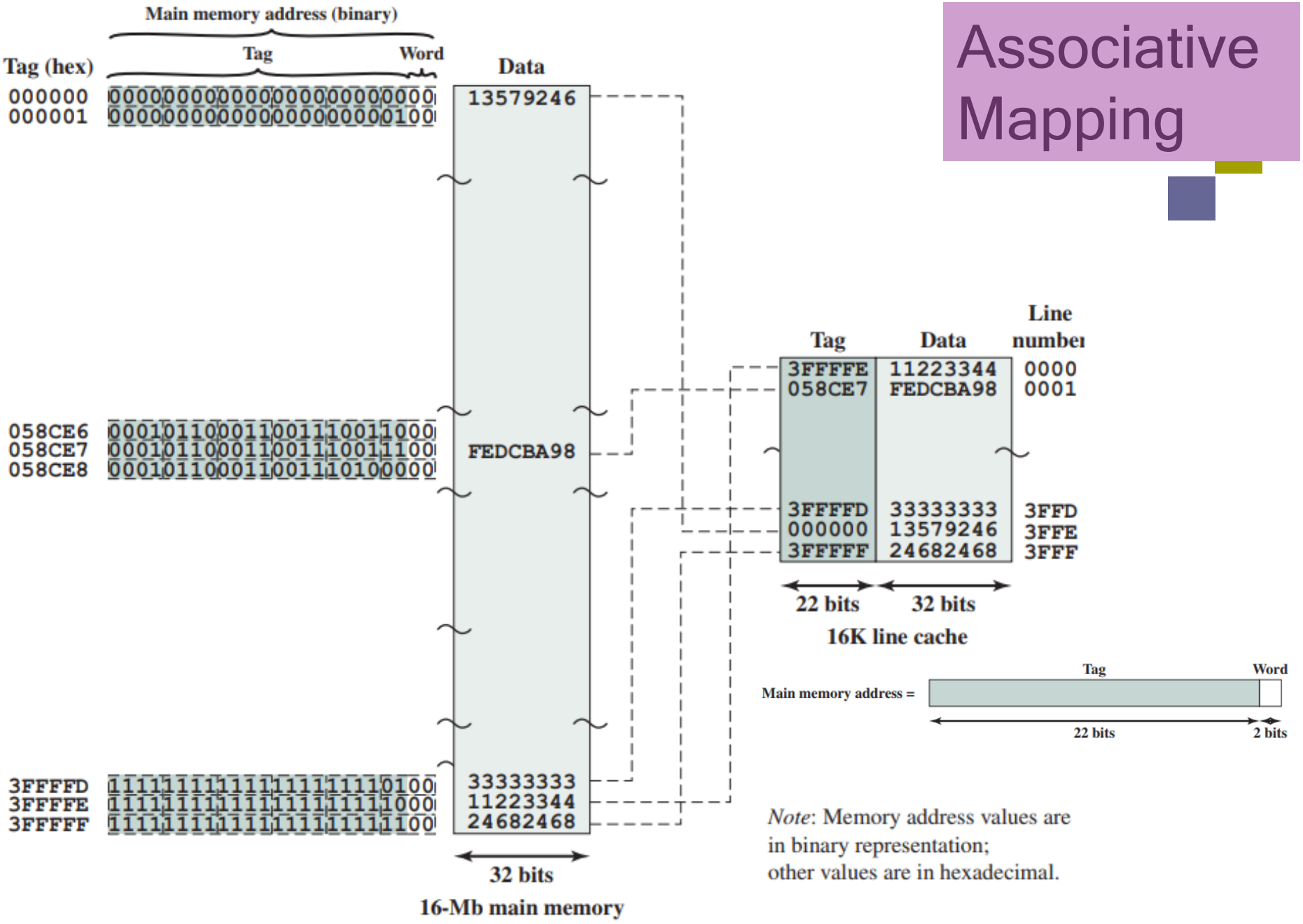
Linda Null and Julia Lobur (2003). *The Essentials of Computer Organization and Architecture.* United States: Jones and Bartlett Publishers. p.245
William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.138
*Patterson, D.A. and Hennessy, J.L. (2014). *Computer Organization and Design: The Hardware/Software Interface (*5th Edition). United States: Elsevier, p.403

6

- When the cache is searched for a specific main memory block, the tag field of the main memory address is compared to all the valid tag fields in cache;

  - ✓ If a match is found, cache hit.
  - ✓ If there is no match, we have a cache miss and the block must be transferred from main memory.

Associative Mapping



Associative Mapping Example

- Main memory size is 32 Bytes. Each block of main memory is 4 words. Cache size is 8 lines.

Part of main memory

00000	AABBCCDD
00100	EEFFGGHH
01000	ABCDABCD
01100	DDCCBBAA
10000	EEIITTRR
10100	FFFFFFFF
11000	GGAAHHTT

Current cache content

Tag	Data	Line
001	EEFFGGHH	000
		001
010	ABCDABCD	010
110	GGAAHHTT	011
		100
		101
		110
		111

Request from CPU

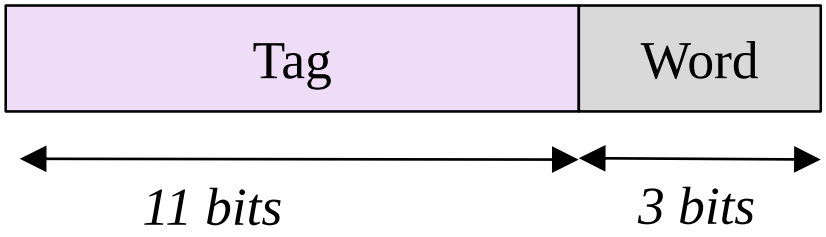
Request	Hit/Miss
00101	**Hit**
00110	**Hit**
00000	**Miss**
10100	**Miss**
00011	
00100	
00100	

Example 16:

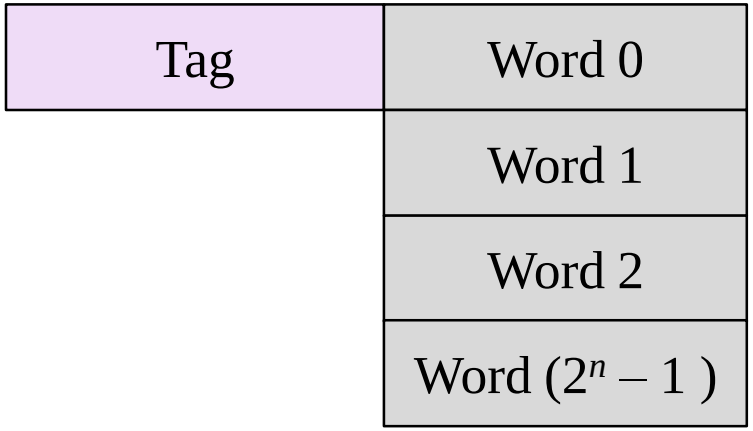
Consider a memory address configuration with 2^{14} words, a cache with 16 blocks, and blocks of 8 words.

- Each block = 8 words = 2^3 → 3 bits (*Word*)
- Tag size = 14 – 3 = 11 bits

Main memory address = 14 bits



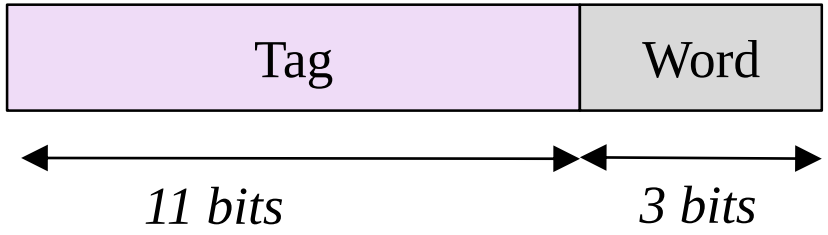
Cache Memory :



666h = 00 0110 0110 0110
0CCh = 00 0110 0110 0

Example 17:

Main memory address :

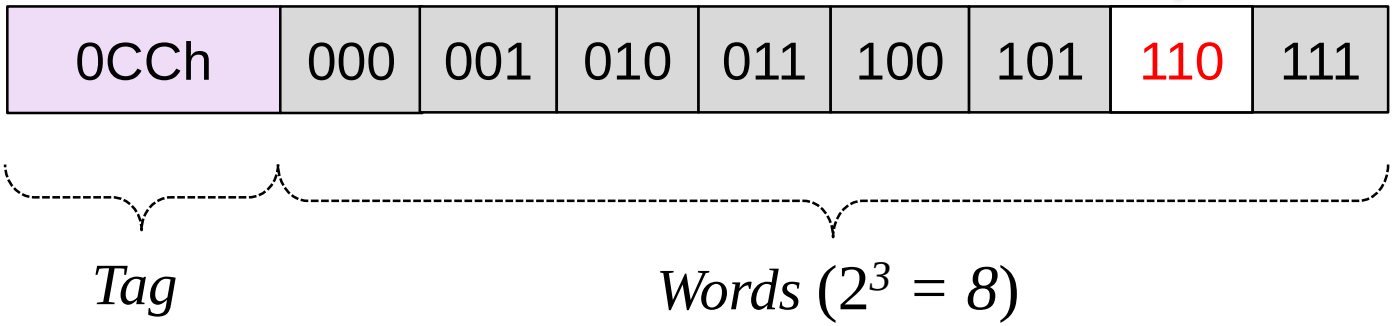


Supposed the program generates the memory address 666h.

→ In 14-bit binary, this number is: 00 0110 0110 0110

It will find the data it is looking in cache memory for in tag 0CCh, *word* 110.

Cache Memory :

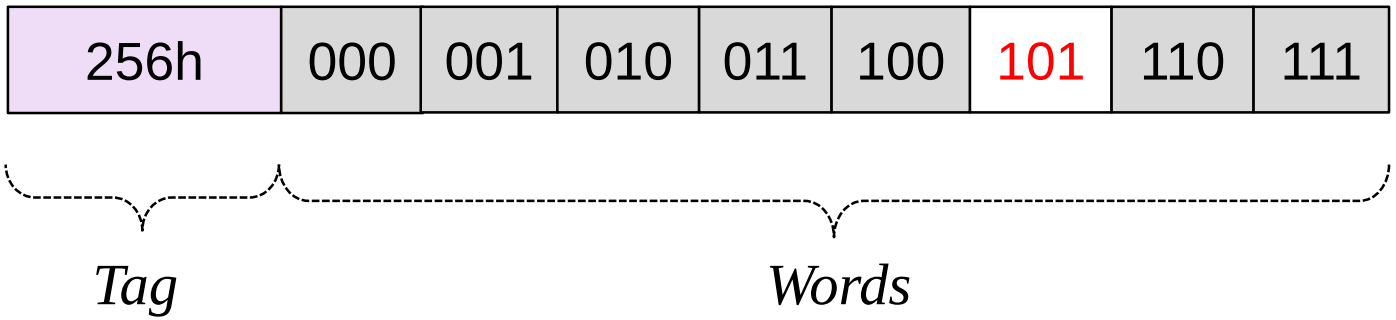


Activity 5

Example 18:

Consider a memory configuration with 2^{14} words, and blocks of 8 words. For data in cache as follows, what is the memory address (hexadecimal).

Cache Memory : (*Data*)



→ In 14-bit binary, the number 256h is: ?

→ The memory address = ?

Fully Associative Mapping:

Advantages and Disadvantages

Flexibility scheme in term of which block to be replaced when a new block is read into the cache.

Every line's tag is examined for a match in fully associative cache (associative mapping), cache searching gets expensive.

(d) Set Associative Mapping

6

N → number of lines in each set

- In this scheme, instead of mapping anywhere in the entire cache, a memory reference can map only to the subset of cache slots.
- Set Associative Mapping is a compromise that exhibits the strengths of both the direct mapping and fully associative mapping approaches while reducing their disadvantages.

Linda Null and Julia Lobur (2003). *The Essentials of Computer Organization and Architecture.* United States: Jones and Bartlett Publishers. p.246
William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.140
*Patterson, D.A. and Hennessy, J.L. (2014). *Computer Organization and Design: The Hardware/Software Interface (*5th Edition). United States: Elsevier, p.403

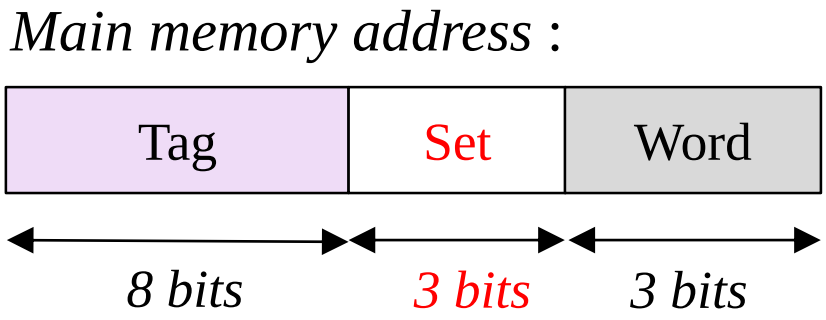
6

- A set-associative cache with *n* locations for a block is called an *n*-way set-associative cache.
- An *n*-way set-associative cache consists of a number of sets, each of which consists of *n* blocks.
- A block is directly mapped into a set, and then all the blocks in the set are searched for a match.
- In set-associative cache mapping, the main memory address is partitioned into three pieces:

Main memory address :

Linda Null and Julia Lobur (2003). *The Essentials of Computer Organization and Architecture.* United States: Jones and Bartlett Publishers. p.246
Patterson, D.A. and Hennessy, J.L. (2014). *Computer Organization and Design: The Hardware/Software Interface (5th Edition).* United States: Elsevier, p.403

Example 19: 2-way



- Suppose we have a main memory of 2^{14} bytes.
- It is mapped to a **2-way** set associative cache having 16 blocks where each block contains 8 words.

- Since this is a 2-way cache, **each set consists of 2 blocks**, and there are 8 sets. (8 words to differentiate 8 distinct sets)
- Thus, we need 3 bits for the set (3 bits to identify 8 unique sets) , 3 bits for the word, giving 8 leftover bits for the tag.

2-way = 2 blocks each set
e.g. first block is 3FFF
Second block is 3FFE
3FFEh = 1111 1111 1111 10

Cache memory (Hex.):

Set	Tag	Word	Tag	Word
7	FF	7	FF	6
7	FF	5	FF	4
7	FF	3	FF	2
7	FF	1	FF	0
6	FF	7	FF	6
6	FF	5	FF	4
6	FF	3	FF	2
6	FF	1	FF	0

(Hex.) Memory Address:

	Tag | Set | Word
3FFF	11111111111111
3FFE	11111111111110
3FFD	11111111111101
3FFC	11111111111100
3FFB	11111111111011
3FFA	11111111111010
3FF9	11111111111001
3FF8	11111111111000
3FF7	11111111110111
3FF6	11111111110110
3FF5	11111111110101
3FF4	11111111110100
3FF3	11111111110011
3FF2	11111111110010
3FF1	11111111110001
3FF0	11111111110000

Set Associative Mapping

- Compromise that exhibits the strengths of both the direct and associative approaches while reducing their disadvantages
- Cache consists of a number of sets
- Each set contains a number of lines
- A given block maps to any line in a given set
- e.g. 2 lines per set
  - 2-way associative mapping
  - A given block can be in one of 2 lines in only one set

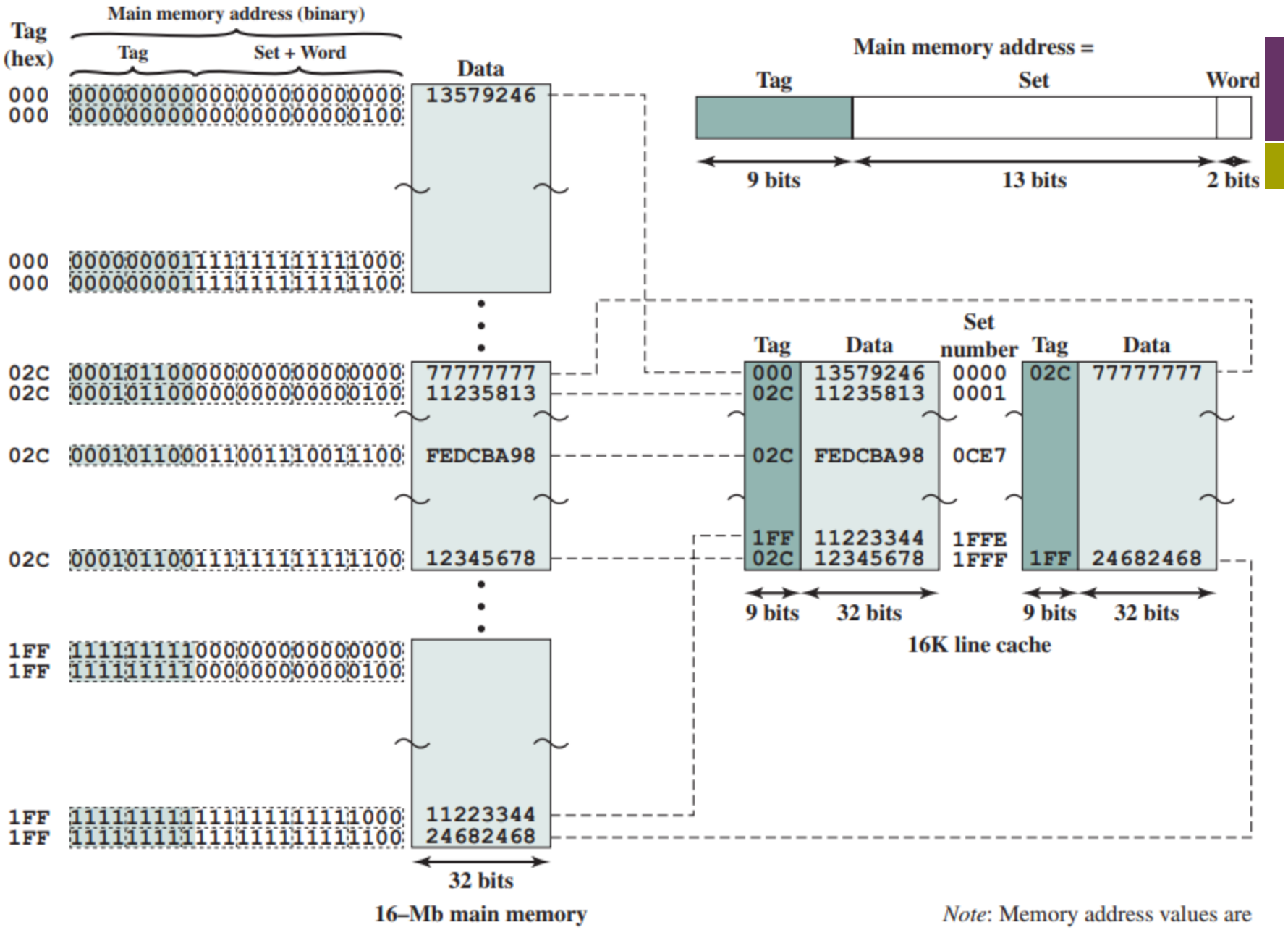
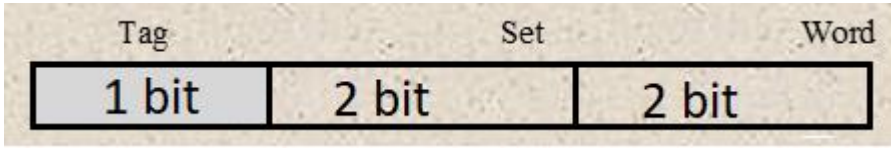


Figure 4.15 Two-Way Set-Associative Mapping Example

2-Way Set Associative Mapping Examp 6

- Main memory size is 32 Bytes. Each block of main memory is 4 words. Cache size is 8 lines.



Request from CPU

Request
00100
10110
00000
11000
00011

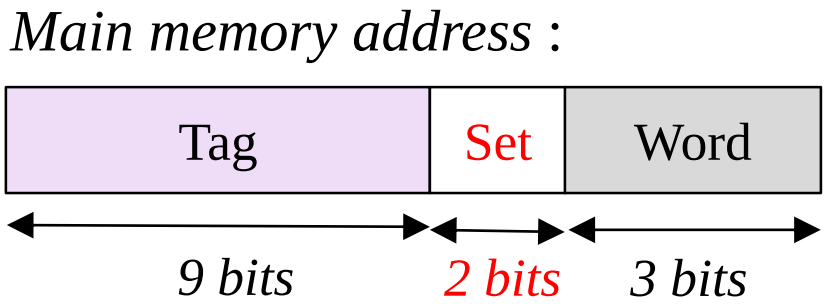
Part of main memory

00000	AABBCCDD
00100	EEFFGGHH
01000	ABCDABCD
01100	DDCCBBAA
10000	EEIITTRR
10100	FFFFFFFF
11000	GGAAHHTT

Current cache content

Tag	Data	Set Number	Tag	Data
0	AABBCCDD	**00**		
1	FFFFFFFF	**01**	1	EEFFGGHH
		10		
		11		

Example 20: 4-way



- Suppose we have a main memory of 2^{14} bytes.
- It is mapped to a 4-way set associative cache having 16 blocks where each block contains 8 words. (so 4 blocks in each set)
- Since this is a 4-way cache, each set consists of 4 blocks, and there are 4 sets altogether.
- Thus, we need 2 bits for the set, 3 bits for the word (to differentiate which block in the set) , giving 9 leftover bits for the tag.

(Hex.)	Memory Address:
3FFF	11111111111111
3FFE	11111111111110
3FFD	11111111111101
3FFC	11111111111100
3FFB	11111111111011
3FFA	11111111111010
3FF9	11111111111001
3FF8	11111111111000

Tag | Set | Word

(Hex.)	Memory Address:
3FF7	11111111110111
3FF6	11111111110110
3FF5	11111111110101
3FF4	11111111110100
3FF3	11111111110011
3FF2	11111111110010
3FF1	11111111110001
3FF0	11111111110000

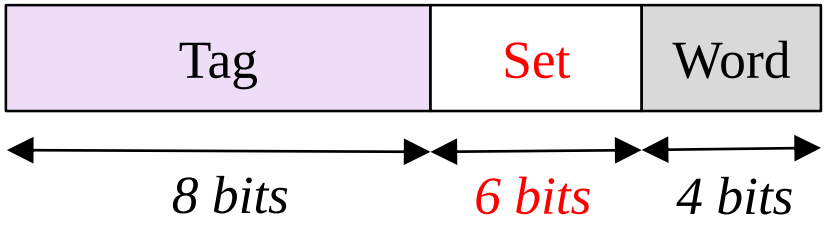
Tag | Set | Word

Cache memory (Hex.):

Set	Tag	Word	Tag	Word	Tag	Word	Tag	Word
3	FF	7	FF	6	FF	5	FF	4
3	FF	3	FF	2	FF	1	FF	0
2	FF	7	FF	6	FF	5	FF	4
2	FF	3	FF	2	FF	1	FF	0

Example 21:

Main memory address :



Suppose we want to read or write a word at the memory address 357A (hexadecimal)

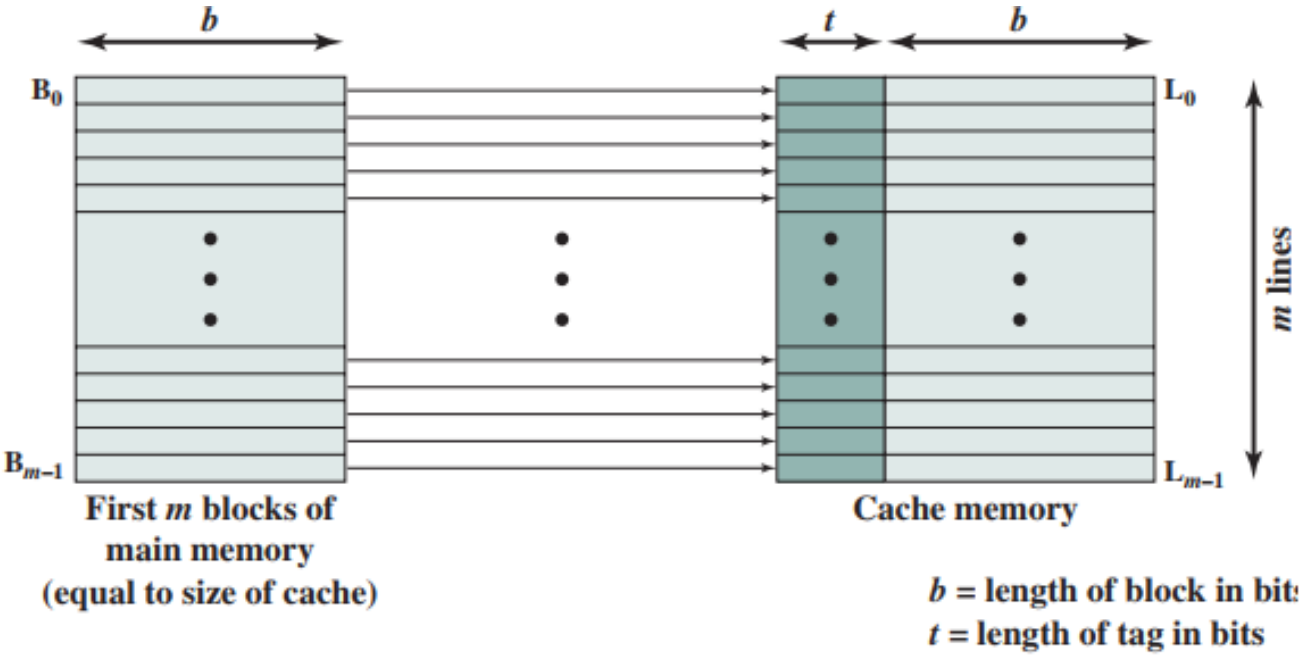
→ In 18-bit binary of the main memory, this number is:
00 0011 0101 0111 1010

Tag: 13 *Set: 23* *Word: 10*

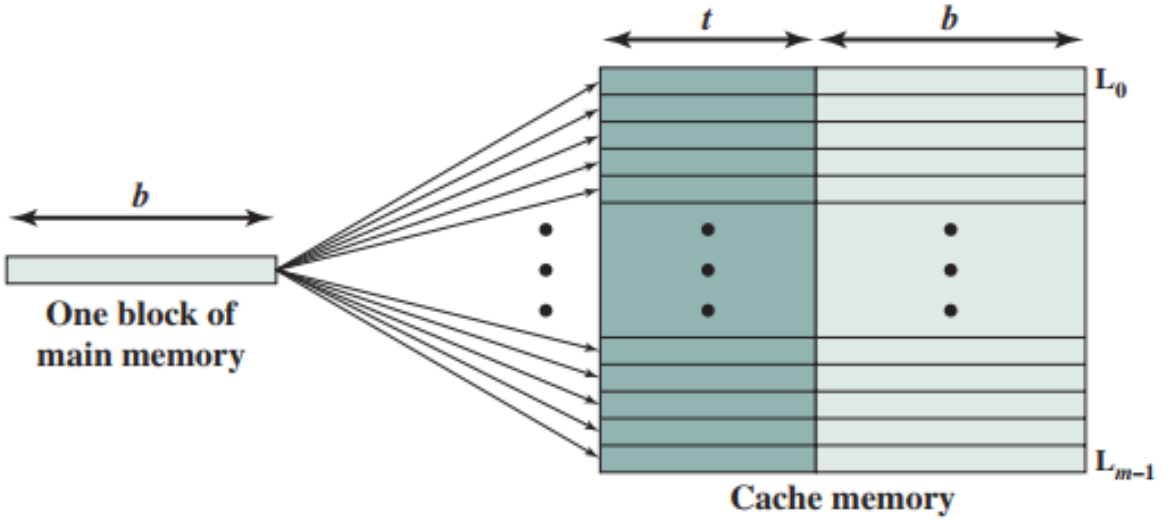
Under set-associative mapping, this translates to decimal

Search for set first then tag

→ So we search only the two tags in cache set 23 to see if either one matches tag 13. If so, we have a cache hit.



(a) Direct mapping



(b) Associative mapping

Comparison on Cache Mapping Schemes

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

Direct

- The simplest technique
- Maps each block of main memory into only one possible cache line

Associative

- Permits each main memory block to be loaded into any line of the cache
- The cache control logic interprets a memory address simply as a Tag and a Word field
- To determine whether a block is in the cache, the cache control logic must simultaneously examine every line's Tag for a match

Set Associative

- A compromise that exhibits the strengths of both the direct and associative approaches while reducing their disadvantages

Activity 6

Determine *n* for a), b), c) and d)

a) n-way set associative

Block	Tag	Data
0		000
1		001
2		010
3		011
4		100
5		101
6		110
7		111

b) n-way set associative

Set	Tag	Data	Tag	Data
0		000		001
1		010		011
2		100		101
3		110		111

c) n-way set associative

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0		000		001		010		011
1		100		101		110		111

d) n-way set associative

Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data
	000		001		010		011		100		101		110		111

6 Replacement Policy

- *Once the cache has been filled, when a new block is brought into the cache, one of the existing blocks must be replaced.

How do we determine which block in cache should be replaced?

- The algorithm for determining replacement is called the *replacement policy*.

Linda Null and Julia Lobur (2003). *The Essentials of Computer Organization and Architecture.* United States: Jones and Bartlett Publishers. p.247
* William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.145

6

- A replacement policy is invoked when it becomes necessary to evict a line/slot/block from cache.
- *There are several popular replacement policies:
  - → One that is not practical but that can be used as a benchmark by which to measure all others is the optimal algorithm.

- Although it is impossible to implement an optimal replacement algorithm, it is instructive to use it as a benchmark for assessing the efficiency of any other scheme.

* Linda Null and Julia Lobur (2003). *The Essentials of Computer Organization and Architecture.* United States: Jones and Bartlett Publishers. p.247
William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.145

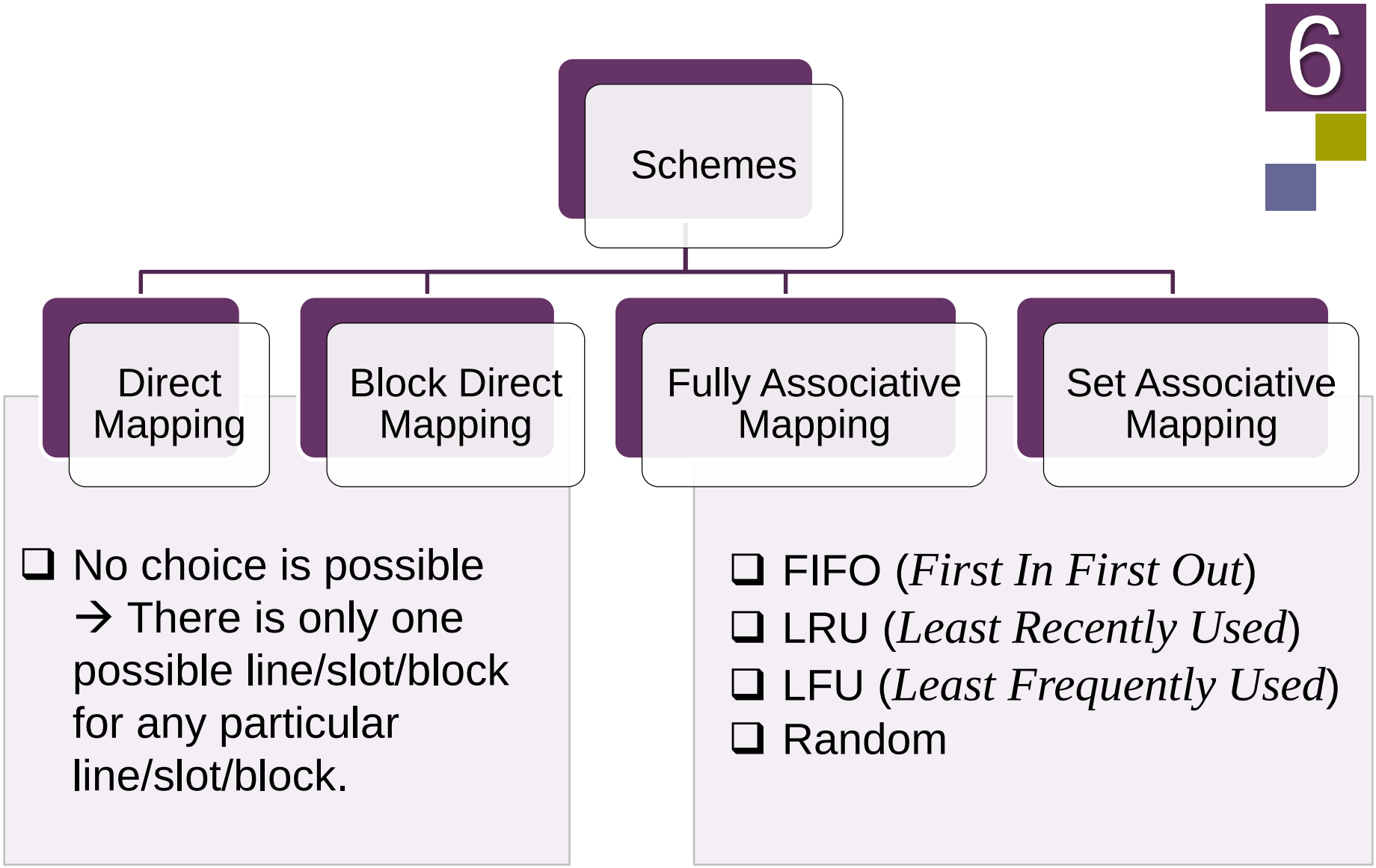


Figure: The replacement algorithms for cache memory.

Table: The replacement algorithms for cache memory.

Algorithms	Operational
FIFO *(First In First Out)*	Replace block that has been in cache longest.
LRU *(Least Recently Used)*	Keeps track of the last time that a block was assessed and evicts block that has been unused for the longest period of time.
LFU *(Least Frequently Used)*	Replace block which has had fewest hits.
Random	Picks a block at random and replaces it with a new block.

Linda Null and Julia Lobur (2003). *The Essentials of Computer Organization and Architecture.* United States: Jones and Bartlett Publishers. p.248
William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.145

Which replacement algorithm is the best?

- ✓ The algorithm selected often depends on how the system will be used.
- ✓ No single (practical) algorithm is best for all scenarios.
- ✓ For that reason, designers use algorithms that perform well under a wide variety of circumstances.

Cache Write Policies

6

- In addition to determining which victim to select for replacement, designers must also decide what to do with so-called *dirty blocks* of cache, or blocks that have been modified in cache.
- Dirty blocks must be written back to memory.
- A write policy determines how this will be done.

Linda Null and Julia Lobur (2003). *The Essentials of Computer Organization and Architecture.* United States: Jones and Bartlett Publishers. p.249
*William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.145

- *There are two basic write policies:

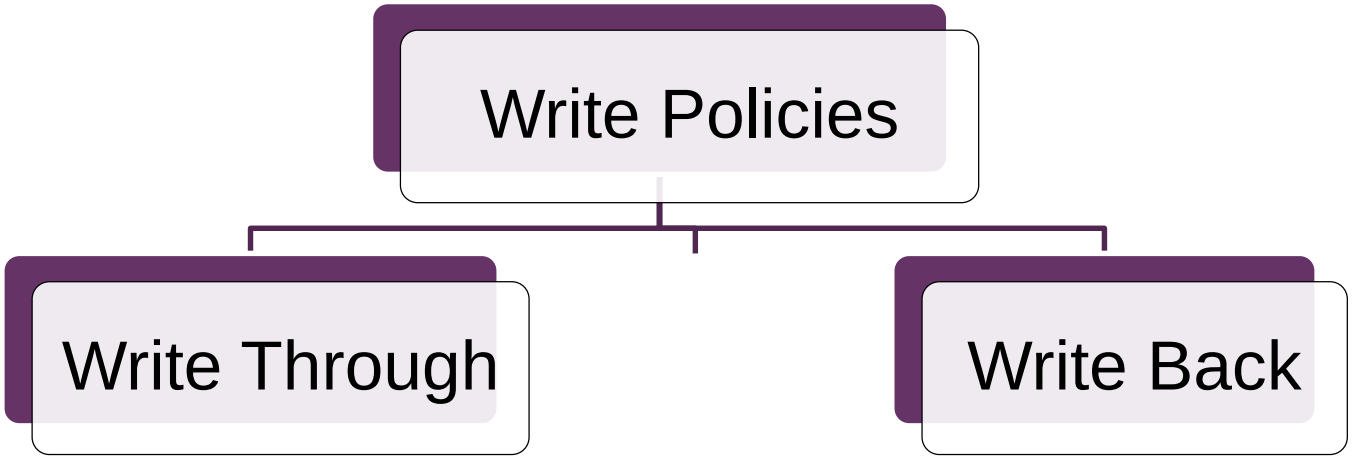


Figure: Cache write policy techniques

Linda Null and Julia Lobur (2003). *The Essentials of Computer Organization and Architecture.* United States: Jones and Bartlett Publishers. p.249
*William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.145
Patterson, D.A. and Hennessy, J.L. (2014). *Computer Organization and Design: The Hardware/Software Interface (5*th Edition). United States: Elsevier, p.393

Cache Write Policies:

(a) Write Through

- A write-through policy updates **both** the cache and the main memory simultaneously on every write.

Every write to the cache requires a main memory access, essentially slows the system down to main memory speed.

- However, in real applications, the majority of accesses are reads so this slow-down is negligible/insignificant.

Cache Write Policies:

(b) Write Back

- A write-back policy (also called *copyback*) only updates blocks in main memory when the cache block is selected as a victim and must be removed from cache.

- Normally faster than *write-through* because time is not wasted writing information to memory on each write to cache.
- Memory traffic is also reduced.

- Main memory & cache may not contain the same value at a given instant of time.
- Data in cache may be lost, if a process terminates (crashes) before the write to main memory is done.

6 Cache Performances

- To improve the performance of cache, one must increase the hit ratio by using :
  - a better mapping algorithm (up to roughly a 20% increase),
  - better strategies for write operations (potentially a 15% increase),
  - better replacement algorithms (up to a 10% increase), and
  - better coding practices (up to a 30% increase in hit ratio).
- Simply increasing the size of cache may improve the hit ratio by roughly 1– 4%, but is not guaranteed to do so.

Miss penalty = The time required to process a *miss*, including the time that it takes to replace a block of memory plus the time it takes to deliver the data to the processor

Average Memory Access Time (AMAT)

- To capture the fact that the time to access data for both hits and misses affects performance, designers sometime use AMAT as a way to examine alternative cache designs.

$$\text{AMAT} = \textit{Time for a hit} + (\textit{Miss rate} \times \textit{Miss penalty})$$

- AMAT is the average time to access memory considering both hits and misses and the frequency of different accesses.

Miss rate = The percentage (%) of memory accesses not found in a given level of memory → (1 – *hit rate*)

Example 22:

Assume that 33% of the instructions in a program are data accesses. The cache hit ratio is 97% and the hit time is one cycle, but the miss penalty is 20 cycles.

$$\begin{aligned} \text{AMAT} &= \textit{Time for a hit} + (\textit{Miss rate} \times \textit{Miss penalty}) \\ &= 1\ \textit{cycle} + (3\% \times 20\ \textit{cycles}) \\ &= 1\ \textit{cycle} + (0.03 \times 20\ \textit{cycles}) \\ &= 1.6 \text{ cycles} \end{aligned}$$

If the cache was perfect and never missed, the AMAT would be one cycle. But even with just a 3% miss rate, the AMAT here increases 1.6 times!

Activity 7

Exercise 6.3:

Find the AMAT for a processor with a 1*ns* clock cycle time, a miss penalty of 20 clock cycles, a miss rate of 0.05 misses per instruction, and a cache access time (including hit detection) of 1 clock cycle. Assume that the read and write miss penalties are the same and ignore other write stalls .

Effective Access Time (EAT)

- The performance of a hierarchical memory is measured by its EAT, or the average time per access.
- EAT is a weighted average that uses the hit ratio and the relative access times of the successive levels of the hierarchy.

$$EAT = (H \times Access_C) + ((1 - H) \times Access_{MM})$$

H : Cache hit rate
$Access_C$: Access time for cache
$Access_{MM}$: Access time for main memory

This formula can be extended to apply to three- or even four-level memories

Linda Null and Julia Lobur (2003). *The Essentials of Computer Organization and Architecture.* United States: Jones and Bartlett Publishers. p.248
William Stallings (2016). *Computer Organization and Architecture: Designing for Performance* (10th Edition). United States: Pearson Education Limited, p.147

Example 23:

Suppose the cache access time is 10ns, main memory access time is 200ns, and the cache hit rate is 99%.

The average time for the processor to access an item in this two-level memory would then be:

$$\begin{aligned} EAT &= (H \times Access_C) + ((1-H) \times Access_{MM}) \\ &= (0.99 \times 10ns) + ((1-0.99) \times 200ns) \\ &= 9.9ns + (0.01)\,200ns \\ &= 9.9ns + 2ns = 11ns \end{aligned}$$

Activity 7a

Exercise 6.4:

Suppose a single cache fronting a main memory, which has 80 nanosecond access time, and the cache memory has access time 10 nanoseconds. Calculate the effective access time if the hit rate are:

(a) 90%

(b) 99%

Review Questions

6

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

6.3.2. For a cache mapping, a main memory address is viewed as consisting of few fields. List and define the fields based on the following schemes:

(a) Direct mapping.

(b) Fully associative mapping.

(c) Set associative mapping.

6

6.3.3. 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

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

6.3.5. For the hexadecimal main memory addresses 111111h, 666666h, BBBBBBh, show the following information, in hexadecimal format :

(a) Direct mapping.

(b) Fully associative mapping.

(c) 2-way Set associative mapping.

6

6.3.6. Consider a 32-bit microprocessor that has an on-chip 16-kB four-way set-associative cache. Assume that the cache has a line size of four 32-bit words.

(a) Draw a block diagram of this cache showing its organization.

(b) Where in the cache is the word from memory location ABCDE8F8 mapped?

6.3.7. Suppose a computer using direct mapped cache has 2^{20} words of main memory and a cache of 32 blocks, where each cache block contains 16 words.

(a) How many blocks of main memory are there?

(b) What is the format of a memory address as seen by the cache, that is, what are the sizes of the tag, block, and word fields?

(c) To which cache block will the memory reference 0DB63h map?

6.3.8. Suppose a computer using direct mapped cache has 2^{32} words of main memory and a cache of 1024 blocks, where each cache block contains 32 words.

(a) How many blocks of main memory are there?

(b) What is the format of a memory address as seen by the cache, that is, what are the sizes of the tag, block, and word fields?

(c) To which cache block will the memory reference 000063FAh map?

6.3.9. Suppose a computer using fully associative cache has 2^{16} words of main memory and a cache of 64 blocks, where each cache block contains 32 words.

(a) How many blocks of main memory are there?

(b) What is the format of a memory address as seen by the cache, that is, what are the sizes of the tag, block, and word fields?

(c) To which cache block will the memory reference F8C9h map?

6.3.10. Suppose a computer using fully associative cache has 2^{24} words of main memory and a cache of 128 blocks, where each cache block contains 64 words.

(a) How many blocks of main memory are there?

(b) What is the format of a memory address as seen by the cache, that is, what are the sizes of the tag, block, and word fields?

(c) To which cache block will the memory reference 01D872h map?

6

6.3.11. Assume a system's memory has 128M words. Blocks are 64 words in length and the cache consists of 32K blocks. Show the format for a main memory address assuming a 2-way set associative cache mapping scheme. Be sure to include the fields as well as their sizes

6.3.12. Consider a byte-addressable computer with 24-bit addresses, a cache capable of storing a total of 64KB of data, and blocks of 32 bytes. Show the format of a 24-bit memory address for:

(a) Direct mapping.

(b) Fully associative mapping.

(c) Set associative mapping.

6

6.3.13. Consider a single-level cache with an access time of 2.5ns, a line size of 64 bytes, and a hit ratio of H = 0.95. Main memory uses a block transfer capability that has a first- word (4 bytes) access time of 50ns and an access time of 5ns for each word thereafter.

(a) What is the access time when there is a cache miss? Assume that the cache waits until the line has been fetched from main memory and then re-executes for a hit.

(b) Suppose that increasing the line size to 128 bytes increases the H to 0.97. Does this reduce the average memory access time?

Module 6
Memory

6.1 Introduction
6.2 Main Memory
6.3 Cache Memory
6.4 Virtual Memory
6.6 Summary

- Overview
- A Real-World Example: Pentium

+

Please see Pear Deck for Class Activity on Virtual Memory

The class activity is individual-based. Join link is shared on Telegram group

6 Overview

- We now know that caching allows a computer to access frequently used data from a smaller but faster cache memory. (Cache is found near the top of our memory hierarchy).
- Another important concept inherent in the hierarchy is *virtual memory*.

The purpose of virtual memory is to use the hard disk as an extension of RAM, thus increasing the available address space a process can use.

- Virtual memory can be implemented with different techniques:

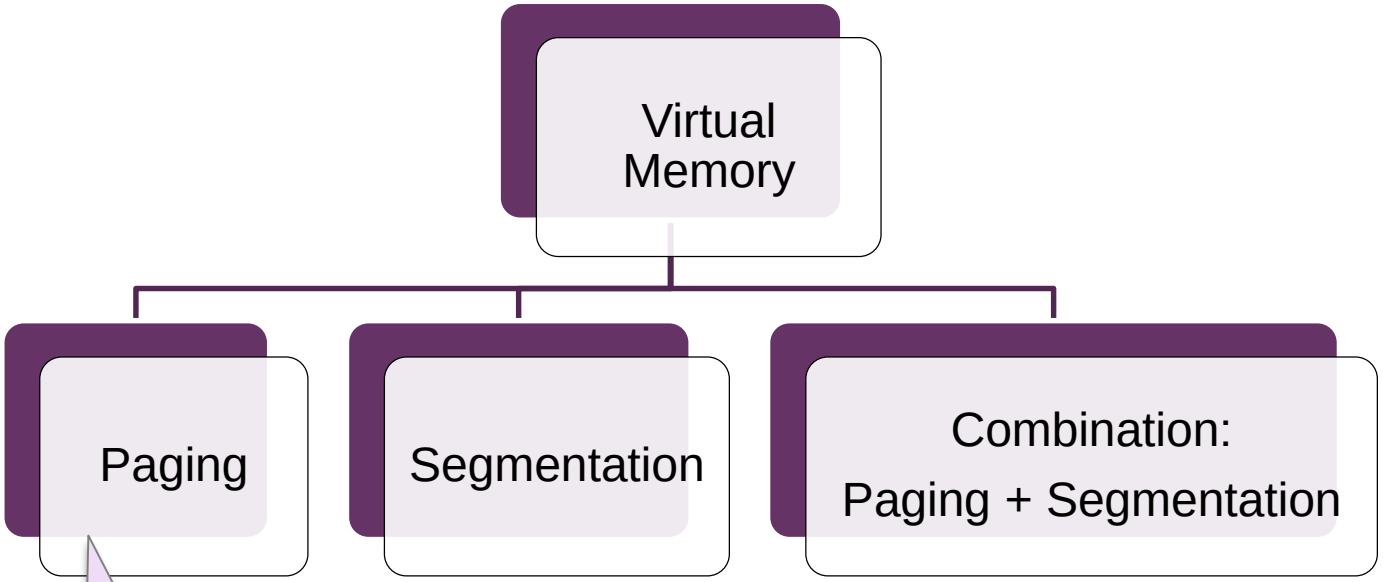


Figure: Virtual memory techniques

The most popular techniques !

Paging

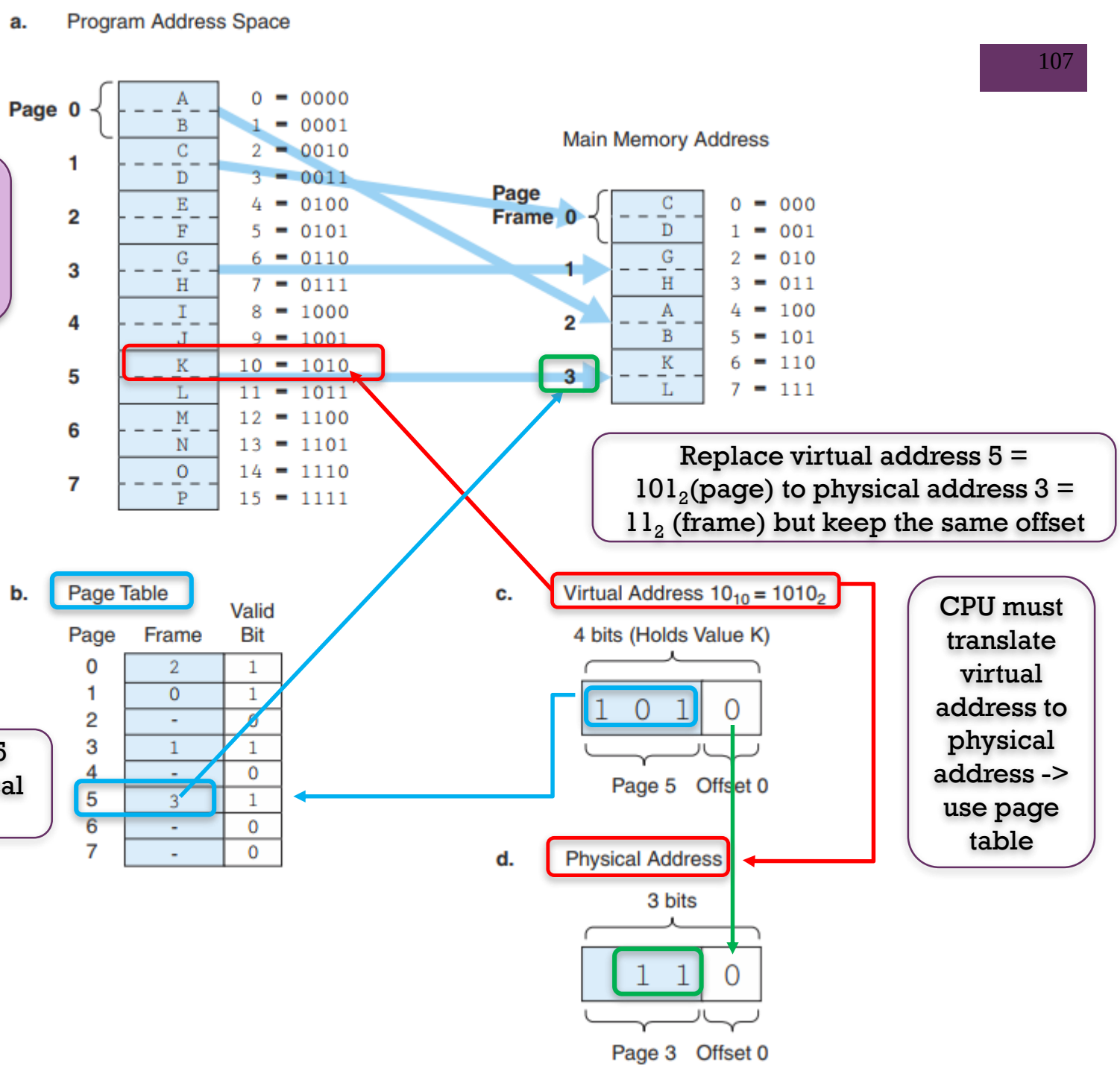
- The most common way to implement virtual memory is by using *paging*.
- Main memory is divided into fixed-size blocks and programs are divided into the same size blocks.
- Typically, chunks of the program are brought into memory as needed.
- It is not necessary to store contiguous chunks of the program in contiguous chunks of main memory.
- Every virtual address for a program that generated by CPU must be translated into a physical address.

Table: Some frequently used terms for virtual memory implemented through paging.

Terms	Description
Virtual address	The logical or program address generated by CPU.
Physical address	The real address in physical memory.
Mapping	Mechanism by which virtual addresses are translated into physical memory.
Page frames	The equal-size chunks in main memory.
Pages	The equal-size chunks in virtual memory.
Paging	The process of copying a virtual page from disk to a page frame in main memory.
Fragmentation	Memory that becomes unusable.
Page fault	An event when a requested page is not in main memory and must be copied into memory from disk.

Page table

- Allocate physical memory to processes in fixed size chunks (page frames) and keep track of where the various pages of the process reside by recording information in a *page table*.
- Every process has its own page table that typically resides in main memory, and the page table stores the physical location of each virtual page of the process.
- The page table has *N* rows, where *N* is the number of virtual pages in the process


a. Program Address Space
Page 0
1
2
3
4
5
6
7
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
0 = 0000
1 = 0001
2 = 0010
3 = 0011
4 = 0100
5 = 0101
6 = 0110
7 = 0111
8 = 1000
9 = 1001
10 = 1010
11 = 1011
12 = 1100
13 = 1101
14 = 1110
15 = 1111
Main Memory Address
Page Frame 0
1
2
3
C
D
G
H
A
B
K
L
0 = 000
1 = 001
2 = 010
3 = 011
4 = 100
5 = 101
6 = 110
7 = 111
Determine the physical address for virtual address 0000
Replace virtual address 5 = 101₂(page) to physical address 3 = 11₂ (frame) but keep the same offset
b. Page Table
Page
Frame
Valid Bit
0 2 1
1 0 1
2 - 0
3 1 1
4 - 0
5 3 1
6 - 0
7 - 0
Virtual page 5 maps to physical frame 3
c. Virtual Address 10₁₀ = 1010₂
4 bits (Holds Value K)
1 0 1 0
Page 5 Offset 0
CPU must translate virtual address -> physical address -> use page table
d. Physical Address
3 bits
1 1 0
Page 3 Offset 0

$$EAT = (H \times Access_C) + ((1 - H) \times Access_{MM})$$

6

Example 25:

Suppose a main memory access requires $200ns$ and that the page fault rate is 1% (99% of the time with the pages needed are in memory). Assume it costs about $10ms$ to access a page not in memory (this time of $10ms$ includes the time necessary to transfer the page into memory, update the page table, and access the data).

The effective access time for a memory access is now:

There is a time penalty associated with virtual memory:

For each memory access that the processor generates, there must now be *two* physical memory accesses —one to reference the page table and one to reference the actual data we wish to access

$$\begin{aligned} EAT &= 0.99\,(200ns + 200ns) + 0.01\,(10ms) \\ &= 0.99\,(400ns) + 0.1ms \\ &= 396ns + 10000ns \\ &= 10396ns \end{aligned}$$

Example 26:

Based on previous example, even if 100% of the pages were in main memory, the effective access time would be:

$$\begin{aligned} EAT &= 100\%\,(200ns + 200ns) + 0 \\ &= 1\,(400ns) \\ &= 400ns \end{aligned}$$

which is double the access time of memory.

Accessing the page table costs an additional memory access because the page table itself is stored in main memory.

TLB (translation look-aside buffer)

- *Accessing the page table costs an additional memory access because the page table itself is stored in main memory*
- We can speed up the page table lookup by storing the most recent page lookup values in a page table cache called a *translation look-aside buffer* (*TLB*).
- Each TLB entry consists of a virtual page number and its corresponding frame

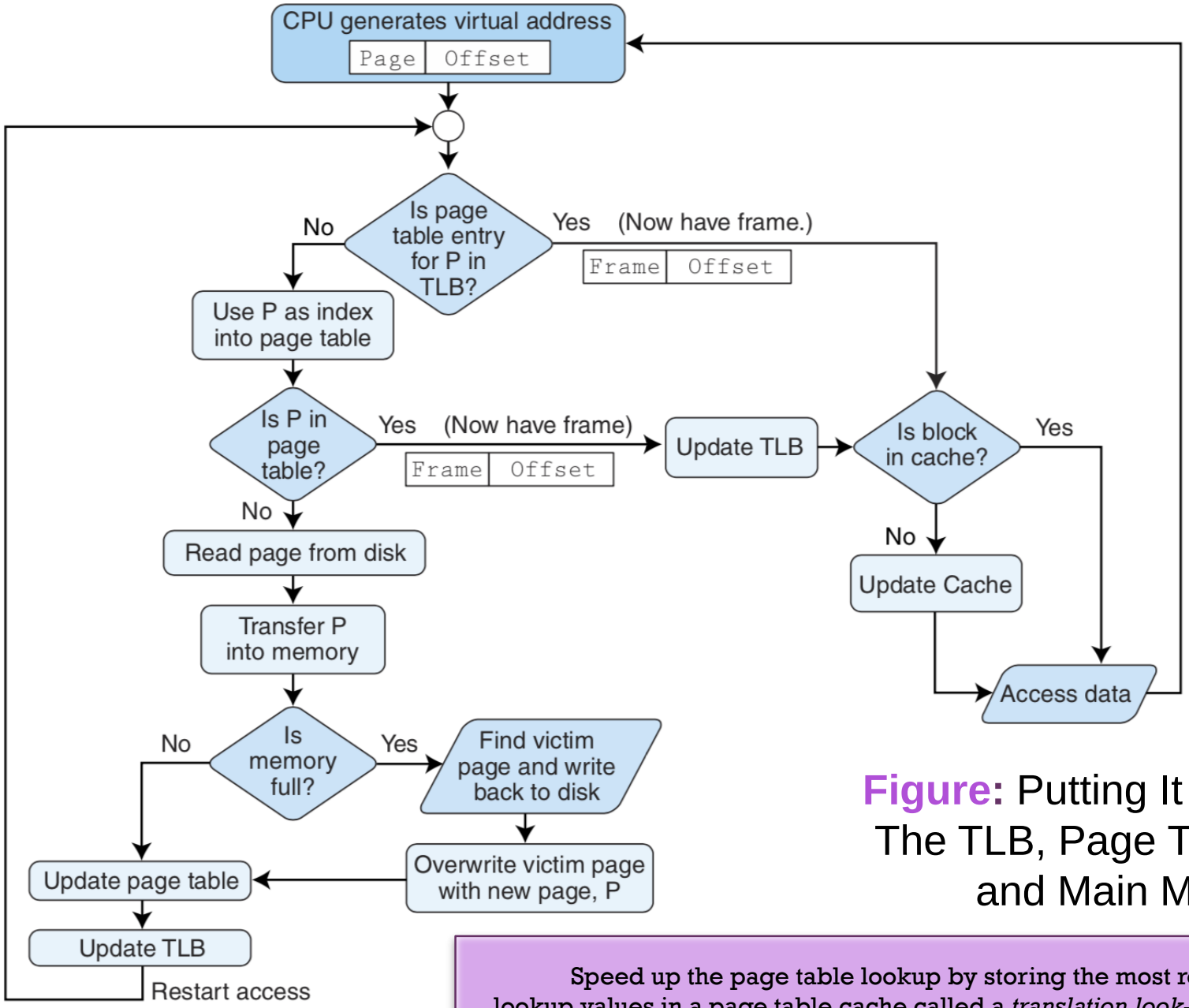


Figure: Putting It All Together: The TLB, Page Table, Cache and Main Memory

Speed up the page table lookup by storing the most recent page lookup values in a page table cache called a *translation look-aside buffer* (*TLB*)

Assume that block is in cache. From the previous video, determine the walkthrough line (A or B or C or D)

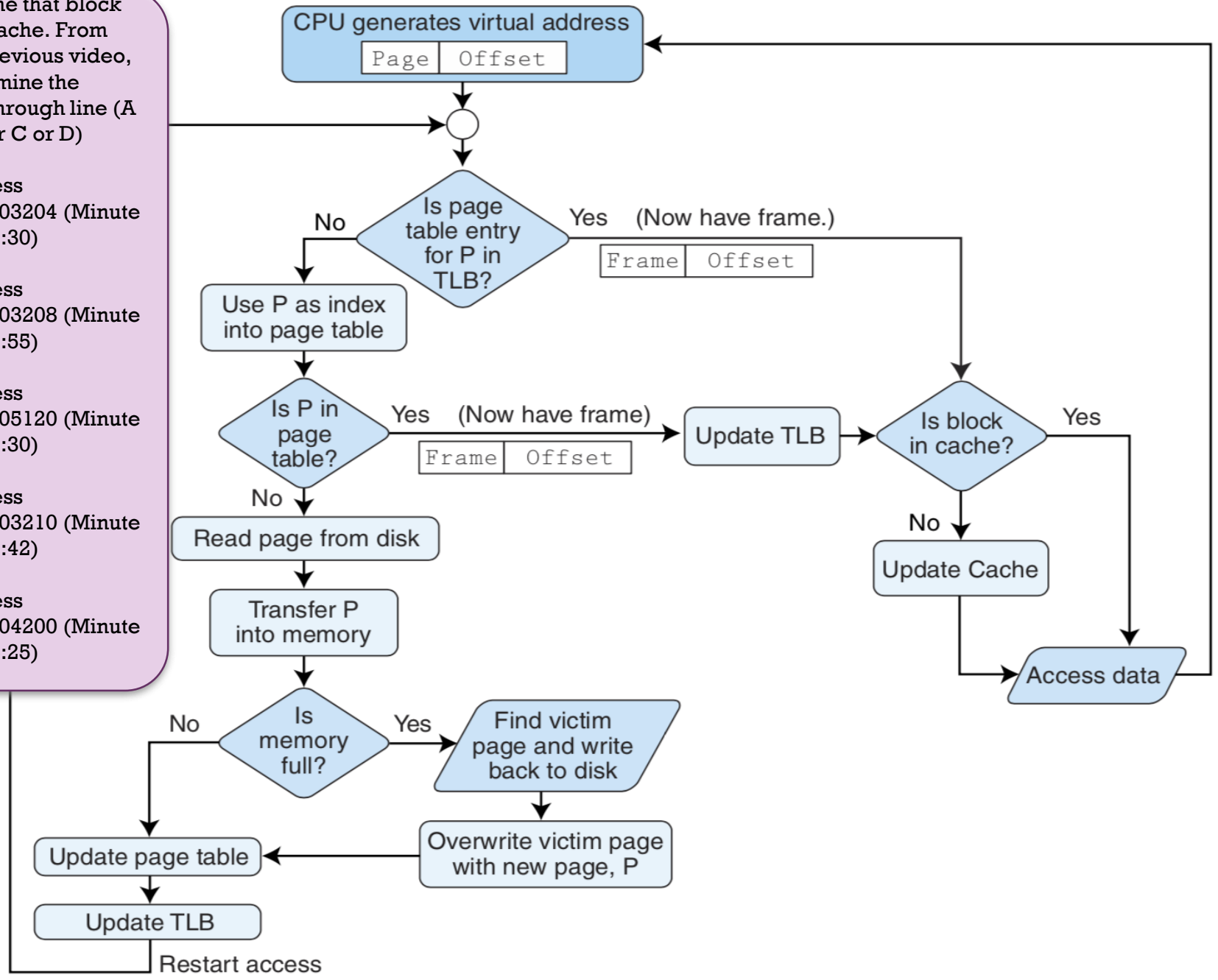
Address 0x00003204 (Minute 0:00-1:30)

Address 0x00003208 (Minute 1:31-1:55)

Address 0x00005120 (Minute 1:56-2:30)

Address 0x00003210 (Minute 2:31-2:42)

Address 0x00004200 (Minute 2:44-4:25)



A Real-World Example: Pentium

- The Pentium architecture exhibits fairly characteristic traits of modern memory management.
  - 32-bit virtual addresses and 32-bit physical addresses.
  - Uses either 4KB or 4MB page sizes, when using paging in different combinations of paging and segmentation.
  - Two caches, L1 and L2, both utilizing a 32-byte block size.
  - Both L1 caches (*I-cache* and *D-cache*) utilize an LRU (least recently used) bit for dealing with block replacement.
  - Each L1 cache has a TLB (*Translation Lookaside Buffer*).
  - The L2 cache can be from 512KB up to 1MB.

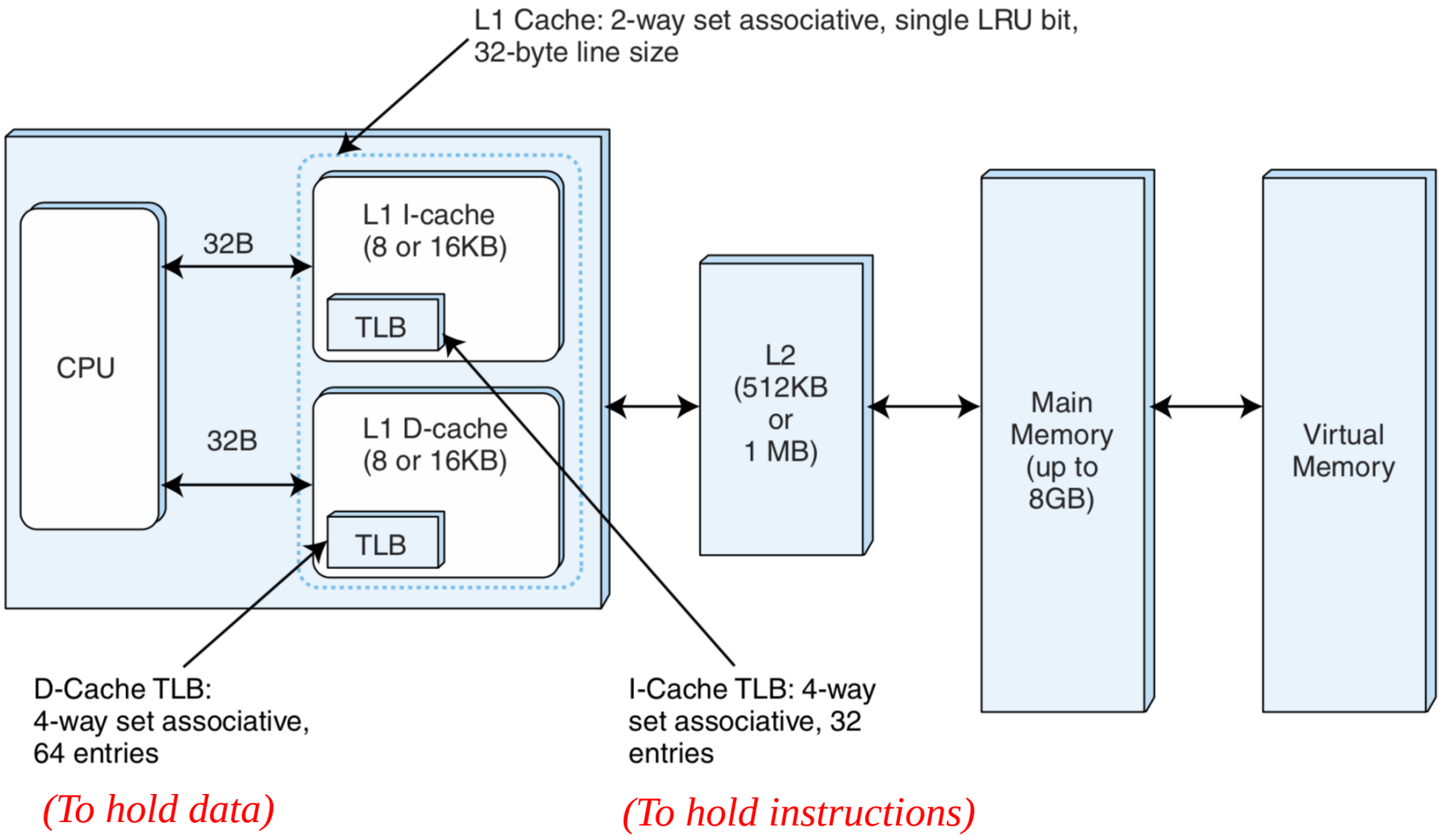


Figure: Pentium memory hierarchy

6.5 Summary

6

- Computer memory is organized in a hierarchy, with the smallest, fastest memory at the top and the largest, slowest memory at the bottom.
- Cache memory gives faster access to main memory, while virtual memory uses disk storage to give the illusion of having a large main memory.
- Cache maps blocks of main memory to blocks of cache memory. Virtual memory maps page frames to virtual pages.
- There are three general types of cache: Direct mapped, fully associative and set associative.

- With fully associative and set associative cache, as well as with virtual memory, replacement policies must be established.
- Replacement policies include LRU, FIFO, or LFU. These policies must also take into account what to do with dirty blocks.
- All virtual memory must deal with fragmentation, internal for paged memory, external for segmented memory.