Organisasi dan Arsitektur Komputer CII2A3

Cache Memory (bagian 1)

Tim Dosen COA

Fakultas Informatika
Universitas Telkom

Rencana Studi

Fakultas Informatika
School of Computing
Telkom University

Rencana Studi		Rincian Nilai Kuis																Rincian Nilai Tugas										Rincian Nilai Hasil Proyek										Bobot Tiap CLO (%)			
Pertemuan Ke-	Materi	CLO 1			CLO 2					CLO 3				CLO 4				CLO 1			CLO 2						CLO	CLO 3						CLO 4									
		Kuis (Kognitif) (20 %)																Tugas Partisipatif (35%)										Hasil Proyek (45%)													
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	1a	1b	1c	2a	2b	2c	2d	2e	3a	3b	1a	1b	2a	1c	2b	2c	3a	3b	4a	3c	4b	3d	4c	1	2	3	4
1	Sistem komputer	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-				
2	Input/Output	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	14	--	--	--
3	Sistem Bus	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-				
4	Organisasi memori	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-				
5	Cara kerja memori utama (RAM)	-	-	-		1	-	-	-	-	-	-	-	-	-	-		-	-	-		4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-				
6	Memori sekunder	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	--	26	--	--
7	Cara kerja *cache memory* (bag-1)	-	-	-	-	-	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-				
8	Cara kerja cache memory (bag-2)	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-				
9	Arsitektur SAP-1	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-				
10	Arsitektur SAP-2	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2	-	-	2	1	-	-		-	-	-	-	-	-	--	--	30	--
11	Arsitektur SAP-3	-	-	-	-	-	-	-	-	-	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	3	-	-	-	7	2		-	-	-	-	-	-				
12	Instruksi *Extended* dan *Indirect*	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	7	-	-	-	-	-	-				
13	Arsitektur MIPS	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-				
14	Instruksi MIPS	-	-	-	-	-	-	-	-	-	-	-	-	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2	1	-	-	-	--	--	--	30
15	Assembly MIPS (bag-1)	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	3	2	-				
16	Assembly MIPS (bag-2)	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	9	7				

Part 1: Bagaimana konsep *cache memory*?

Hirarki Memori

Fakultas Informatika
School of Computing
Telkom University

Registers

L1 Cache

L2 Cache

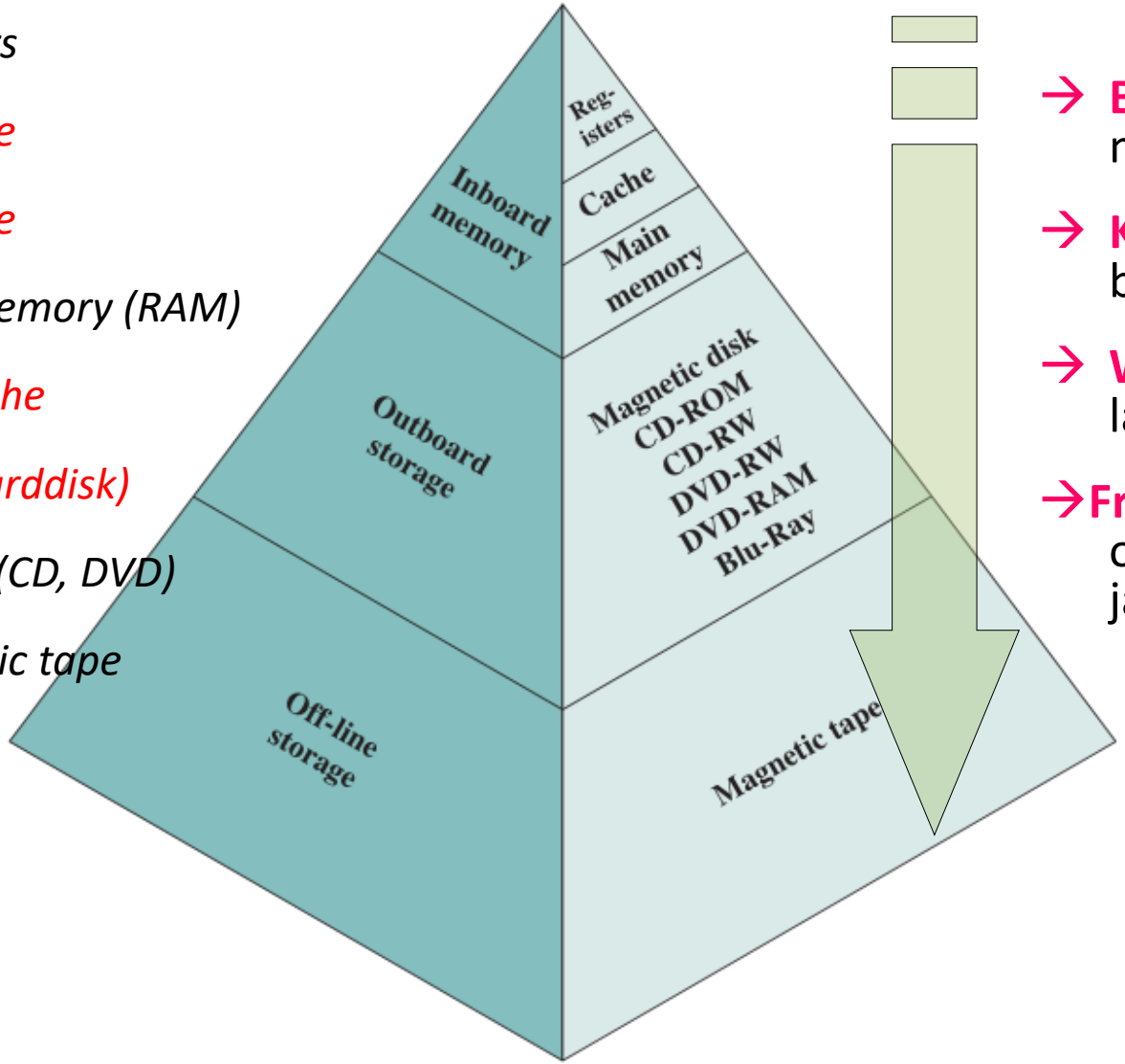
Main memory (RAM)

Disk cache

Disk (Harddisk)

Optical (CD, DVD)

Magnetic tape



→ **Biaya** per bit makin murah

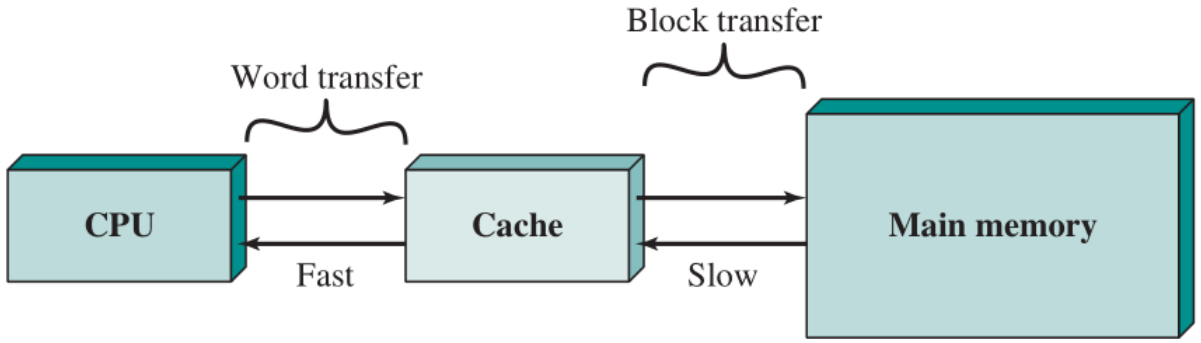
→ **Kapasitas** makin besar

→ **Waktu akses** makin lama

→ **Frekuensi** diakses oleh prosesor makin jarang

Cache Memory - Fungsi

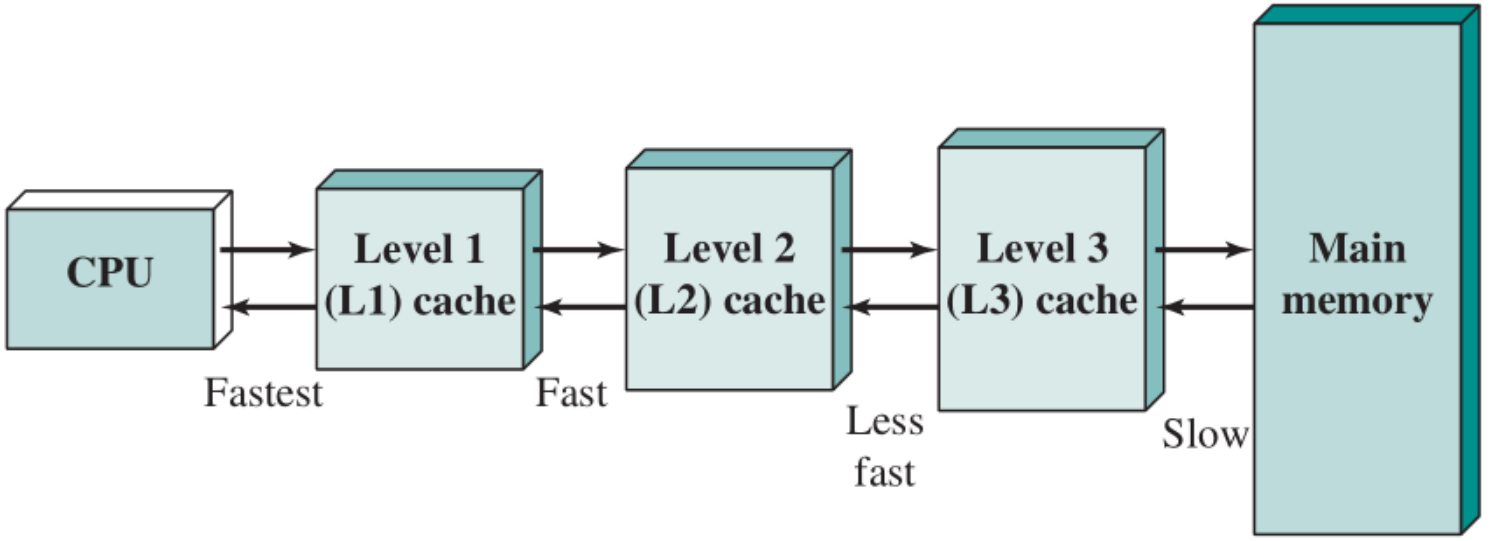
- Adalah *fast memory* berkapasitas kecil
- Tujuan:
  - Memberikan memori yang mempunyai kecepatan mendekati memori tercepat yang tersedia (*register*)
  - Memberikan memori semikonduktor dengan harga lebih murah
  - Untuk meng-*copy* sebagian isi *main memory* yang sering diakses



(a) Single cache

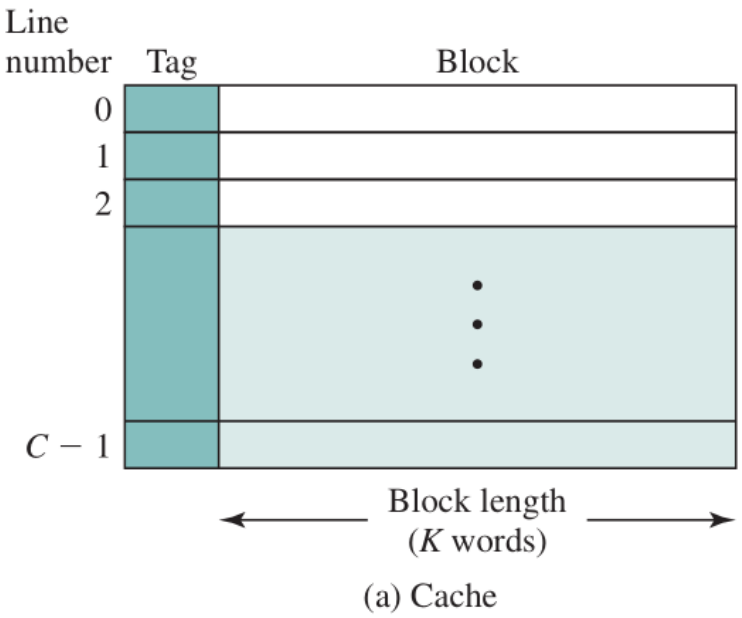
- Terletak antara CPU dan main memori
- Dapat terletak di dalam chip CPU atau di dalam modul tersendiri

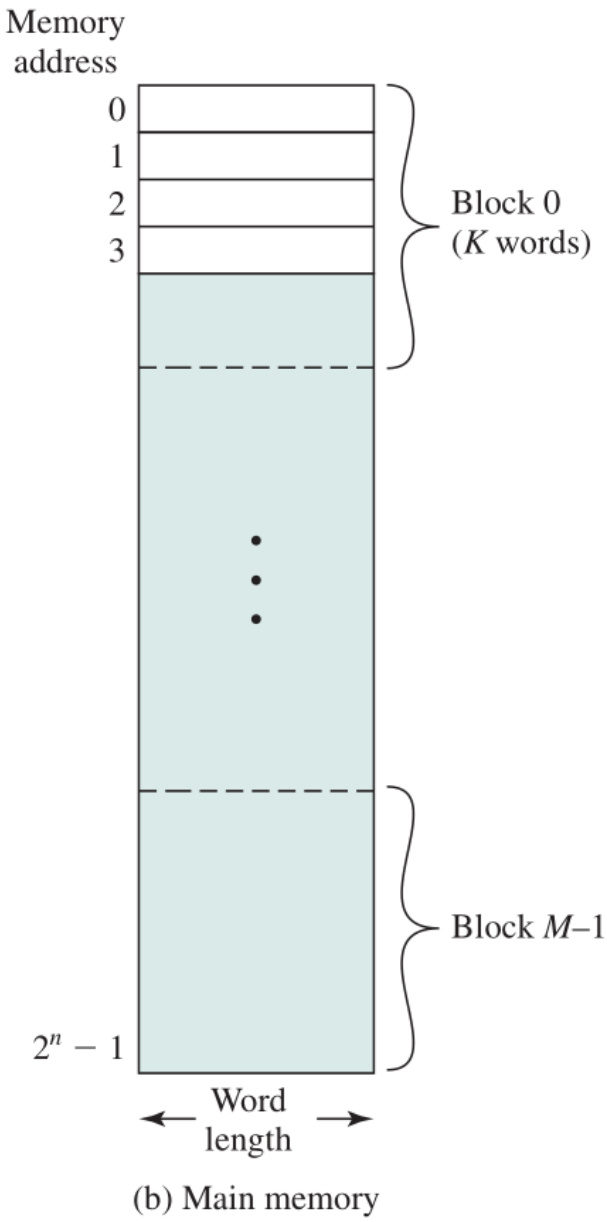
Cache Memory - Fungsi



(b) Three-level cache organization

Cache Memory - Struktur





--Satu blok terdiri dari beberapa *word*

--Tag:

- Sbg identitas blok yang mana yang sedang disimpan di *cache memory*
- Mrpk bagian dari alamat *main memory*

Cache Memory – Operasi Baca data (1)

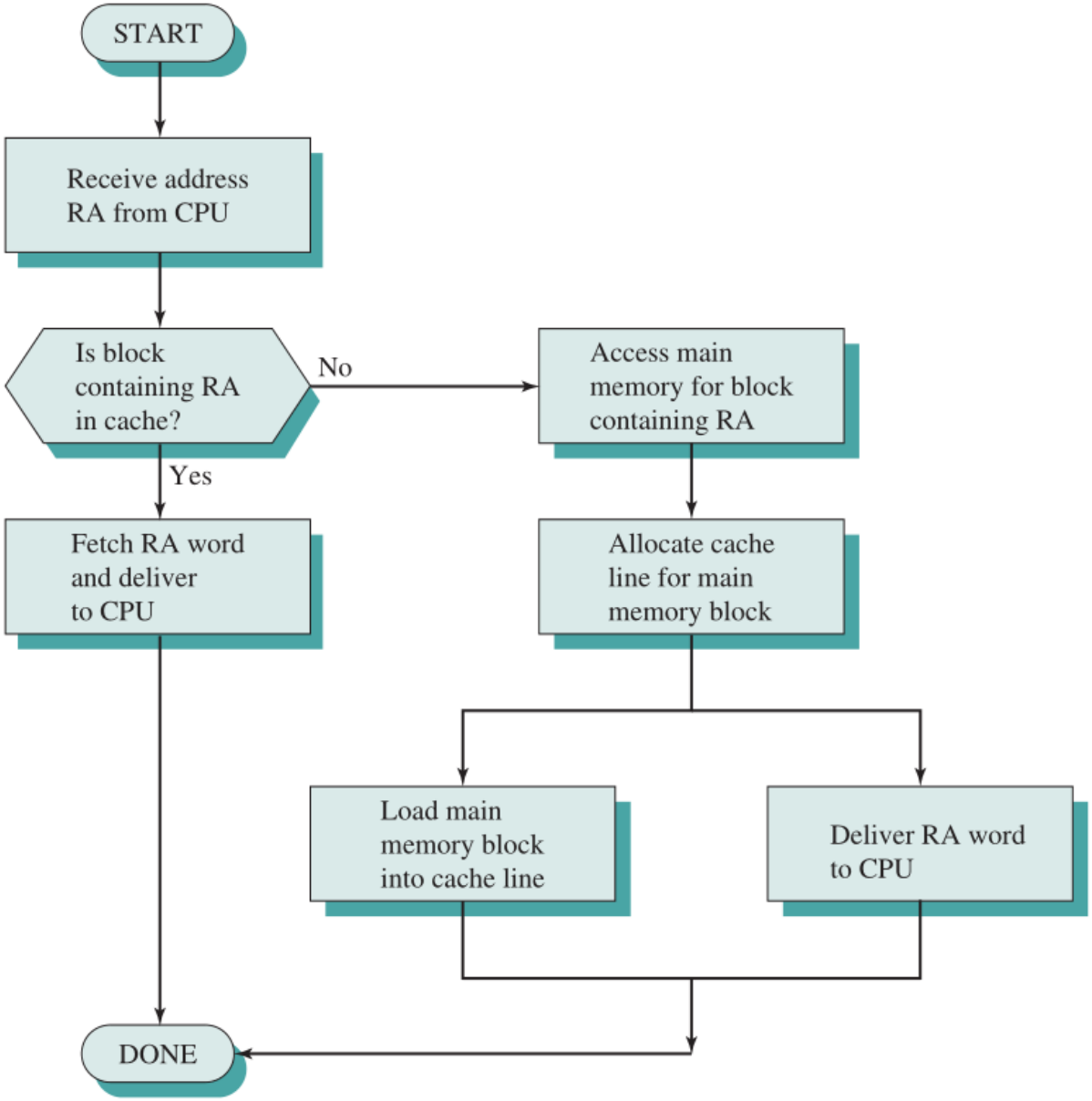


Figure 4.5 Cache Read Operation

Cache Memory – Operasi Baca data (2)

- CPU mengeluarkan alamat data *RA (Relative Address)* yang akan dibaca
- Periksa data di *cache memory*
- Jika data ada di *cache memory* → diambil (*cache hit*) → cepat
- Jika data tidak ada (*cache miss*) → cari data di *main memory*
- *Copy* satu blok data ke *cache memory*
- Berikan data yang diperlukan ke CPU

Cache Memory – Operasi Baca data (3)

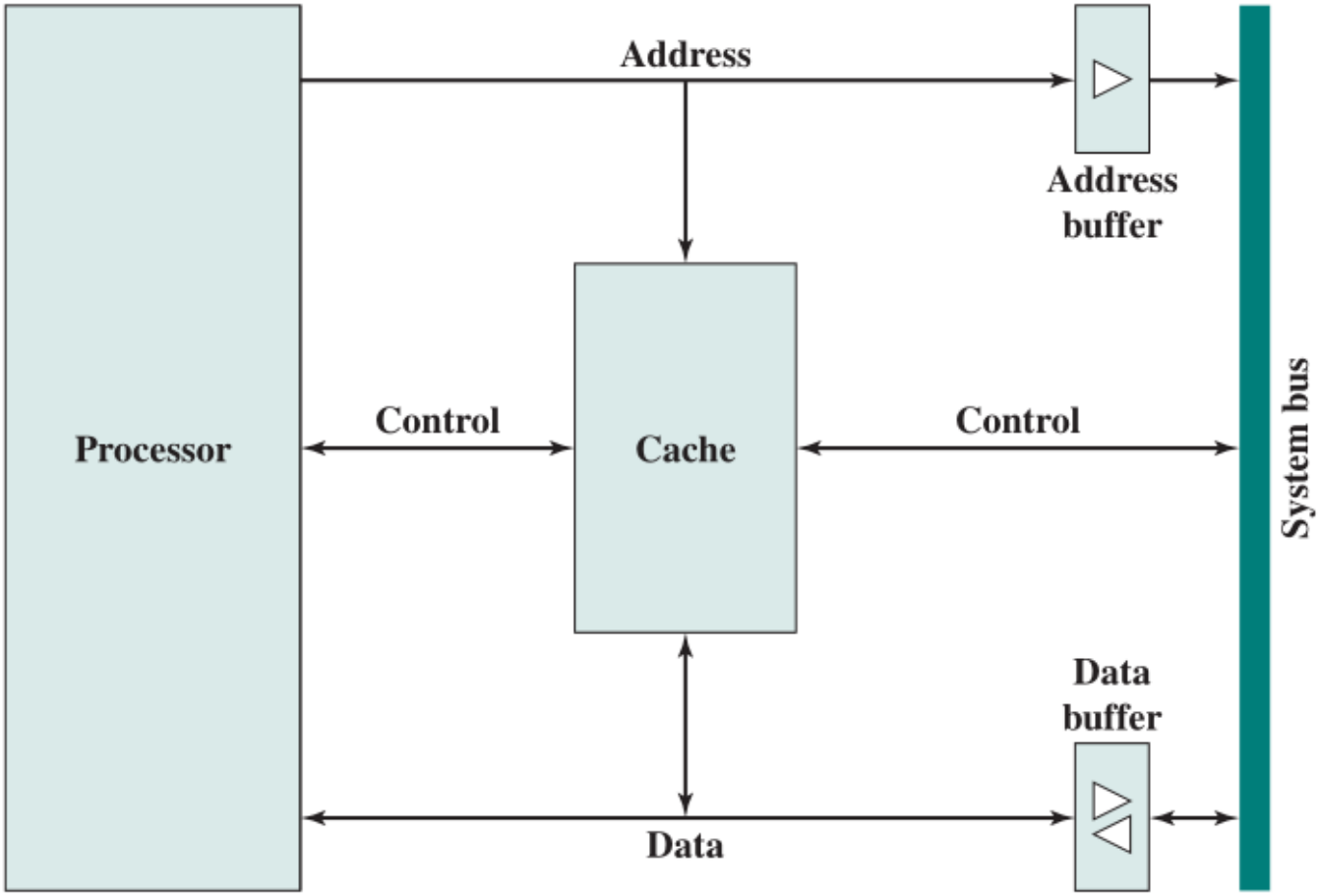


Figure 4.6 Typical Cache Organization

--*Cache hit* → *buffer* di-*disable*, tdk perlu akses ke *system bus*

Elemen Perancangan Cache

- Ukuran (Size) *cache*
- *Mapping Cache-Main memory*
  - *Direct*
  - *Associative*
  - *Set associative*
- Algoritma Penggantian (*Replacement*)
  - *Least Recently Used (LRU)*
  - *First In First Out (FIFO)*
  - *Least Frequency Used (LFU)*
  - *Random*
- Cara penulisan (*Write Policy*)
  - *Write through*
  - *Write back*
- Ukuran blok (*Block Size*)
- Jenis cache
  - *Off-chip, On-chip*
  - *Single level, Multilevel*
  - *Unified, Split*

Ukuran (*size*) *cache*

Fakultas Informatika
School of Computing
Telkom University

- Mengapa ukuran *cache* kecil?
  - Biaya: makin besar → makin mahal
  - Makin besar → makin banyak jumlah gerbang yang digunakan untuk pengalamatan → butuh waktu akses lebih lama → performansi menurun
  - Area *chip/board* yang tersedia terbatas
- Mengapa ukuran *cache* berbeda-beda?
  - Untuk menyesuaikan dengan beban kerja komputer
  - Performansi *cache* sangat terpengaruh oleh beban kerja sistem (misal: beban PC berbeda dengan beban *mainframe*)

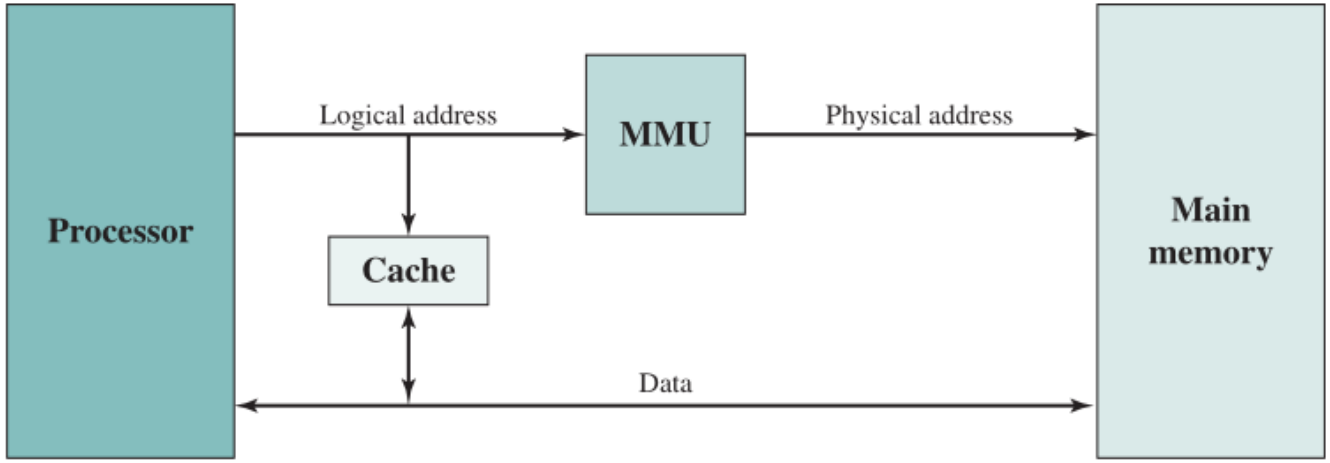
Elemen Perancangan Cache

- Ukuran (*Size*) *cache*
- Mapping *Cache-Main memory*
  - *Direct*
  - *Associative*
  - *Set associative*
- Algoritma Penggantian (*Replacement*)
  - *Least Recently Used (LRU)*
  - *First In First Out (FIFO)*
  - *Least Frequency Used (LFU)*
  - *Random*
- Cara penulisan (*Write Policy*)
  - *Write through*
  - *Write back*
- Ukuran blok (*Block Size*)
- Jenis *cache*
  - *Off-chip, On-chip*
  - *Single level, Multilevel*
  - *Unified, Split*

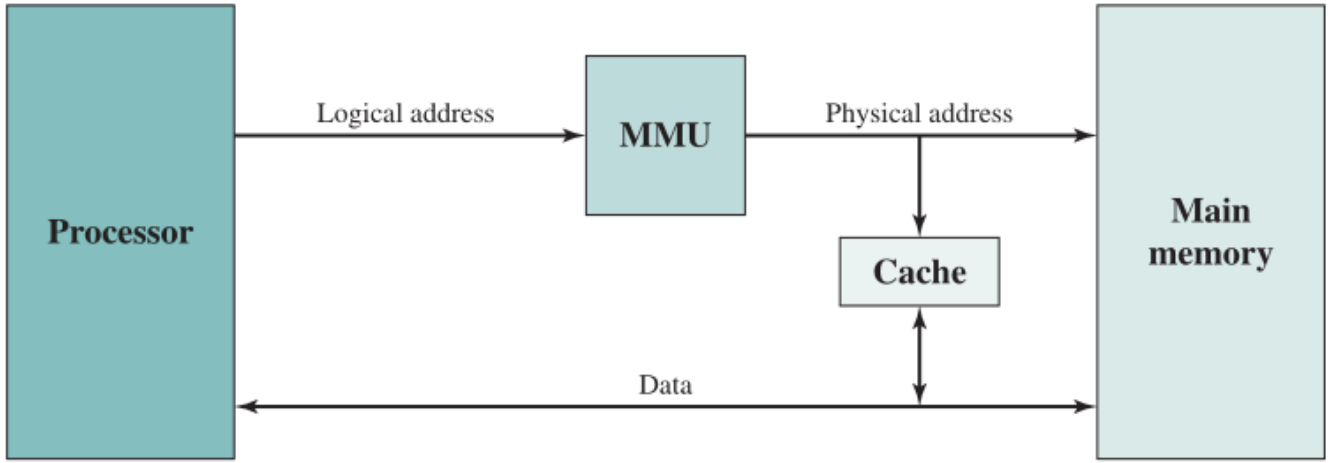
Table 4.2 Elements of Cache Design

Cache Addresses	**Write Policy**
Logical	Write through
Physical	Write back
Cache Size	**Line Size**
Mapping Function	**Number of Caches**
Direct	Single or two level
Associative	Unified or split
Set associative	
Replacement Algorithm	
Least recently used (LRU)	
First in first out (FIFO)	
Least frequently used (LFU)	
Random	

Elemen Perancangan *Cache*



(a) Logical cache



(b) Physical cache

Part 2: Bagaimana cara kerja metode *direct mapping* pada *cache memory*?

Mapping *Cache-Main memory*

Fakultas Informatika
School of Computing
Telkom University

- Mengapa perlu *di-mapping*?
  - Kapasitas *cache* jauh lebih kecil daripada kapasitas *main memory*

(a) Direct Mapping

- Memetakan setiap blok memori ke dalam satu line/baris *cache* secara tetap (sesuai dengan nomor line)

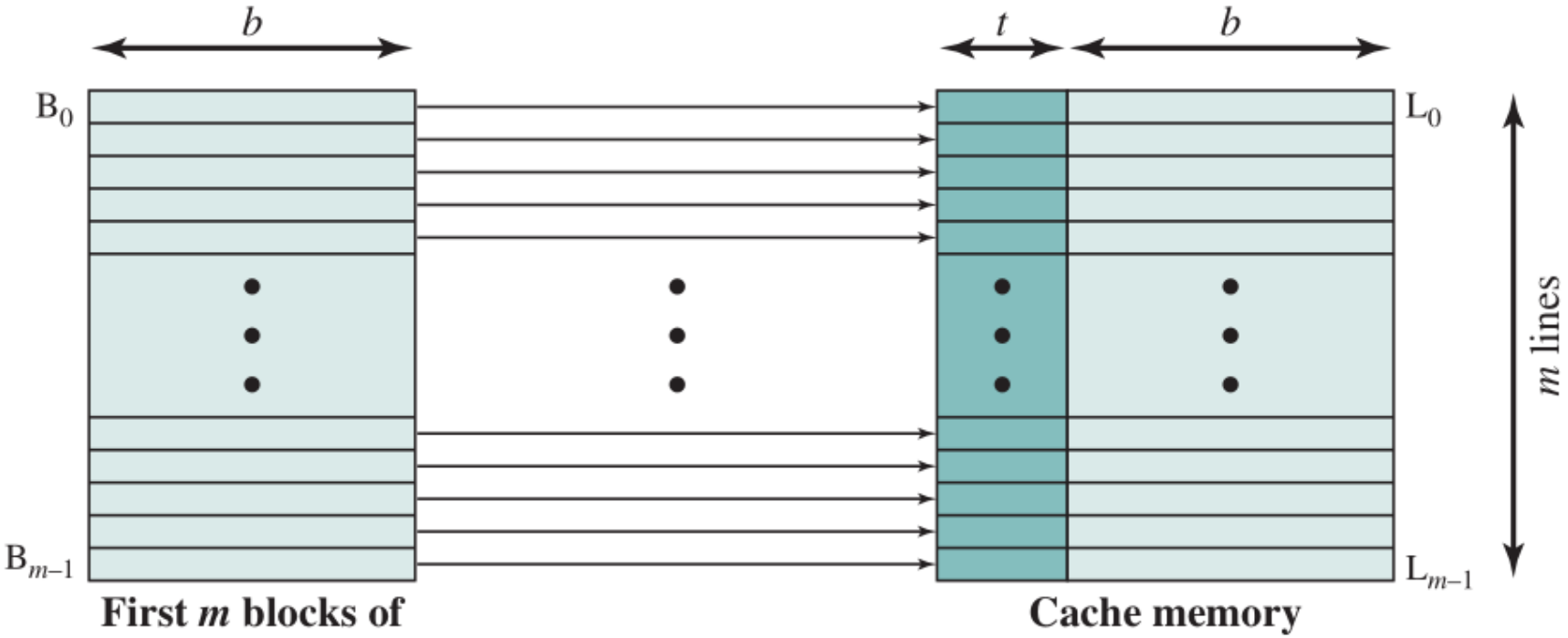
(b) *Associative Mapping*

- Memetakan setiap blok memori ke sembarang baris *cache* (tidak terikat pada nomor line)

(c) *Set Associative Mapping*

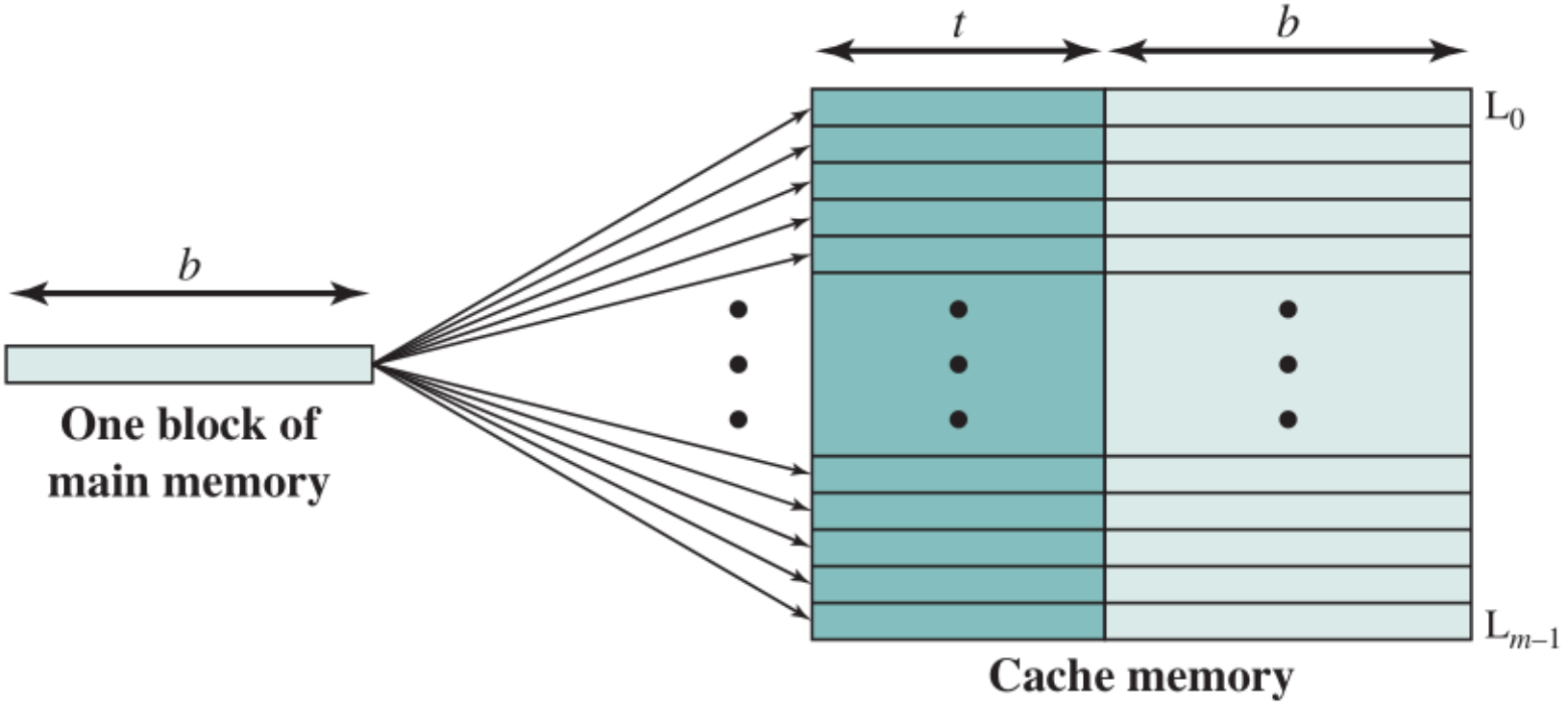
- Memetakan setiap blok memori ke dalam satu set **tertentu** yang di dalamnya terdiri dari beberapa *line* yang dapat digunakan secara bebas

Mapping *Cache-Main memory*



(a) Direct mapping

Mapping *Cache-Main memory*



(b) Associative mapping

Direct Mapping (1)

- Rumus: i = j modulo m

  i = nomor baris *cache*

  j = nomor blok *main memory*

  m = jumlah total baris di dalam *cache*

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

Direct Mapping (2)

- Untuk keperluan akses *cache* → setiap alamat memori dibagi menjadi 2 bagian:
  - w = bit-bit identitas word atau *byte* di dalam blok memori
  - s = bit-bit identitas blok memori
    - s terdiri dari dua bagian:
      - *line field* (r): bit-bit nomor baris *cache*
      - *tag* (s-r): bit-bit identitas blok data yang ada di memori
  - s + w = alamat memori

- Format alamat memori: (dari sisi *cache*)

Tag (s-r)	Line or Slot (r)	Word (w)

Direct Mapping - Baca data (1)

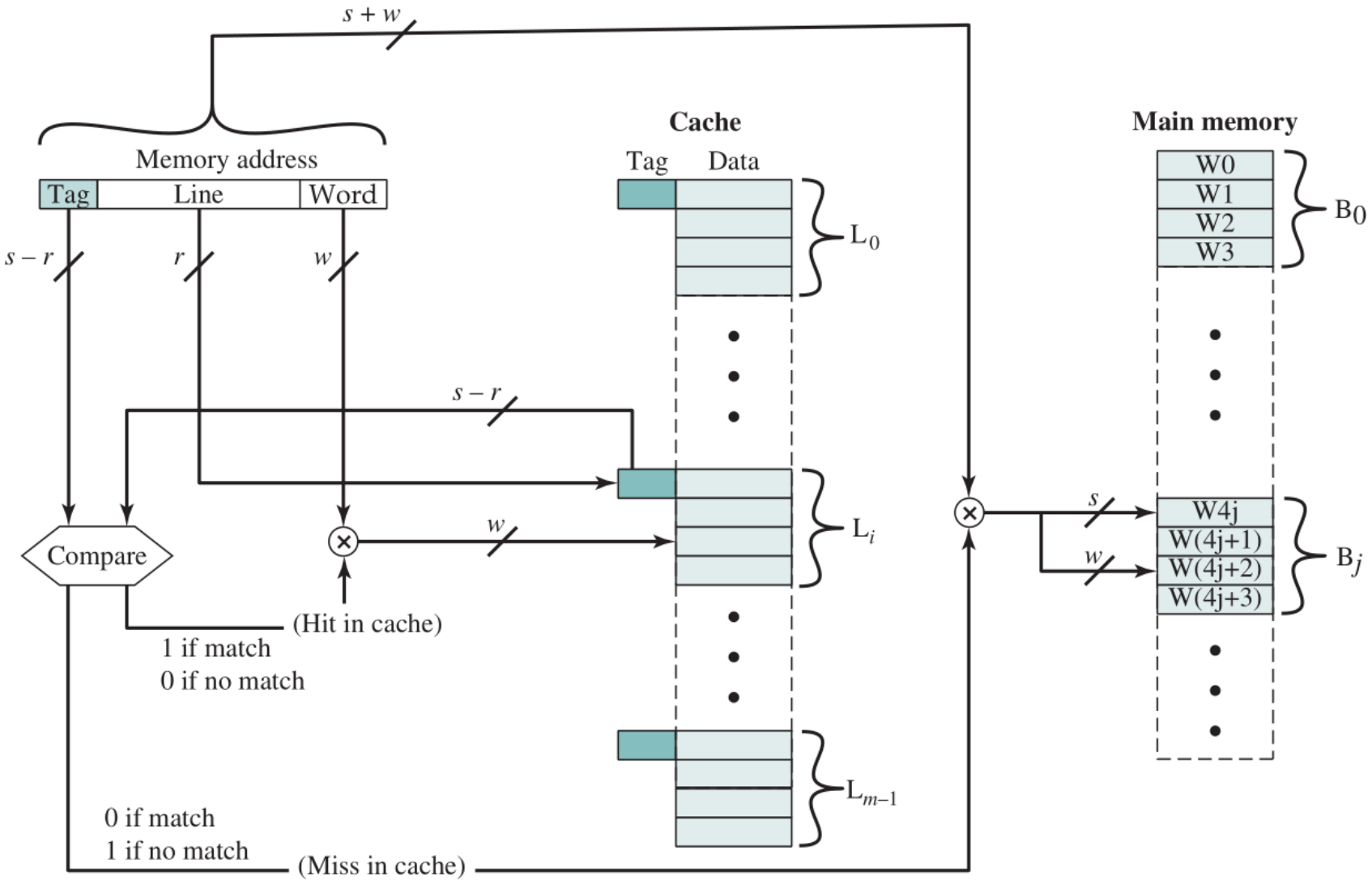


Figure 4.9 Direct-Mapping Cache Organization

Direct Mapping - Baca data (2)

1. Cek apakah baris di *cache valid* (ada isinya)
2. Jika *valid* →CPU membandingkan nomor tag yang akan dibaca dengan nomor tag yang ada di *cache*
3. Bila nomor tag tsb sama (*cache hit*) → pilih *word* yang diinginkan yang terletak pada nomor baris (*line*) yang diinginkan
4. Bila nomor tag tsb berbeda (*cache mis*) → ambil (*fetch*) satu blok data sesuai dengan nomor tag dan line yang diinginkan

Direct Mapping - Contoh (1)

- Diketahui:
  - *Main memory* berukuran 16 MByte
  - *Cache* berukuran 64 kByte
  - 1 byte = 1 alamat
  - 1x transfer data = 1 blok memori = 1 *line cache* = 4 byte = 4 alamat

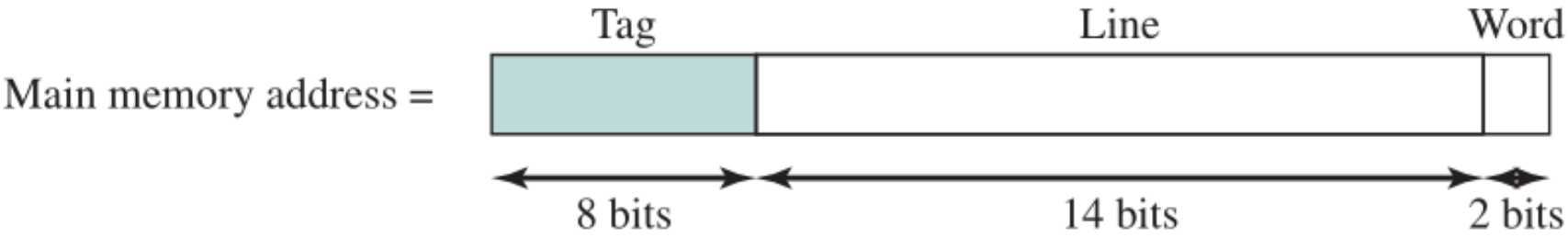
  Sebutkan jumlah bit untuk tag (s-r), line (r), dan word (w)!

  Gambarkan *mapping*nya!

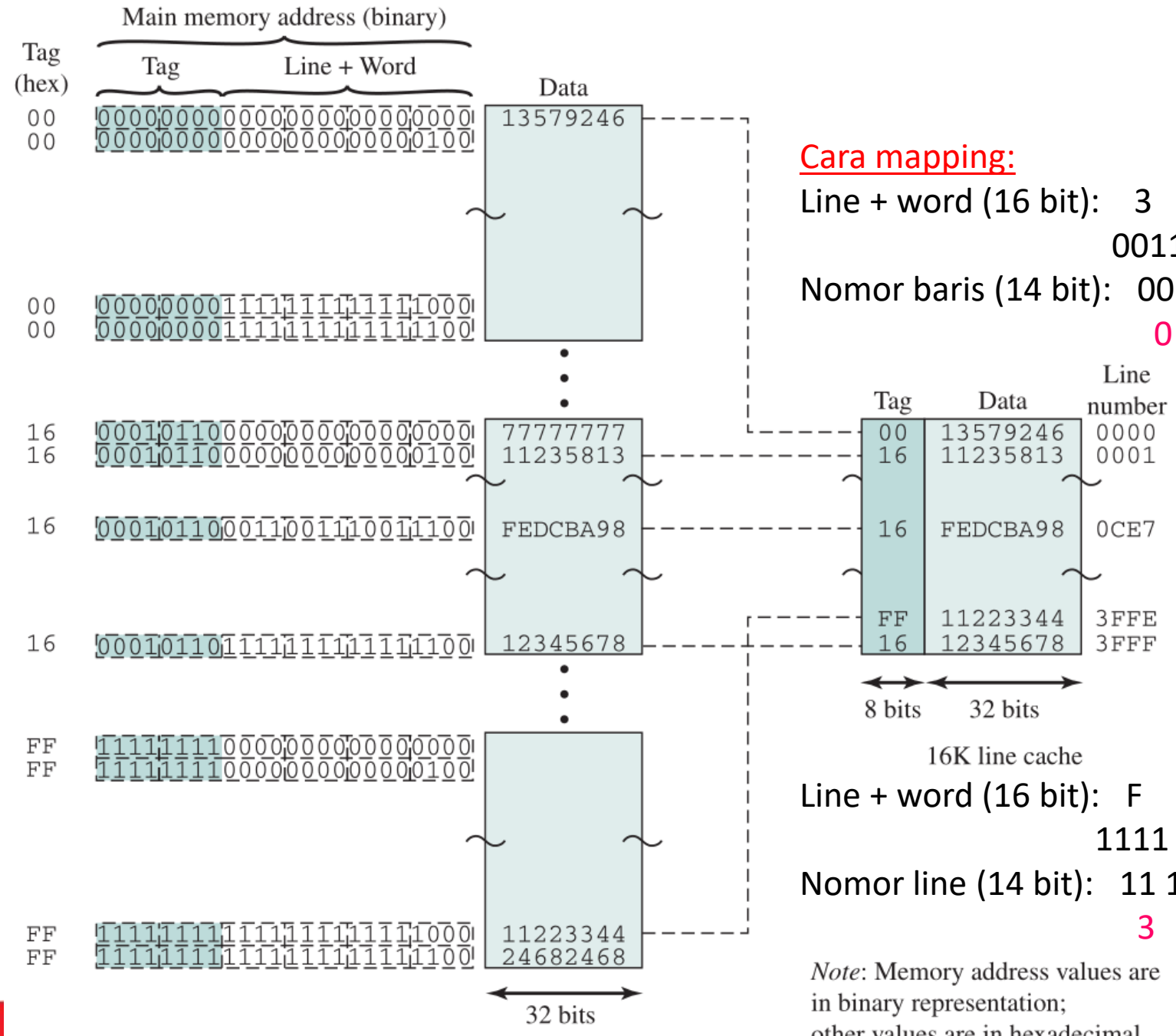
  Berikan contohnya!
- Jawab:
  - Jumlah alamat total = 16 MB/1 byte = 16 M alamat
  - *Memory* 16 M alamat = $2^4 . 2^{20} = 2^{24}$ → Jumlah bit alamat yang diperlukan = 24 bit (lebar alamat)
  - 1 blok = 4 alamat = 2^2 , maka
    - Jumlah bit identitas word (w) = 2 bit
  - Jumlah ***line*** *cache* = 64 kbyte/4 byte = 16 k *line*
  - 16 k = $2^4 . 2^{10} = 2^{14}$ → Jumlah *bit line* = 14 bit
  - Jumlah bit tag (s-r) = 24 – 14 – 2 = 8 bit

Direct Mapping - Contoh (2)

- Format alamat memori: (dari sisi *cache*)



- Jumlah blok memori = 16 Mbyte/4 byte = 4 M blok
- 1 *line cache* ≈ 1 blok memori, maka:
  - line:blok = 16 K: 4 M = 1:(2^2 . 2^{10})/ 2^4 = 1:256
  - 1 line *cache* mempunyai kemungkinan ditempati oleh 256 data yang berbeda (nomor tag berbeda)
- Jumlah tag = 2^8 = 256 tag
- 1 tag = 4 M blok / 256 = 2^2 x 2^{20} / 2^8 = 2^{14} = 16 kblok
- Setiap nomor blok memori hanya akan menempati satu nomor line *cache* tertentu saja (tidak berpindah-pindah)



Cara mapping:

Line + word (16 bit): 3 3 9 C

0011 0011 1001 1100

Nomor baris (14 bit): 00 1100 1110 0111

0 C E 7

Line + word (16 bit): F F F 8

1111 1111 1111 1000

Nomor line (14 bit): 11 1111 1111 1110

3 F F E

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

Direct Mapping - Kelebihan/kekurangan

(+) Sederhana

(+) Mudah diimplementasikan

(+) Tidak mahal

(-) Lokasi mapping setiap blok sudah tertentu (tidak fleksibel)

(-) Dapat terjadi *thrashing* bila program mengakses 2 blok yang terletak pada *line cache* yang sama secara berulang-ulang → terjadi proses *swap* memori berkali-kali → *hit ratio* menjadi rendah

Part 3: Bagaimana cara kerja metode *associative mapping* pada *cache memory*?

Associative Mapping

- Format alamat memori: (dari sisi *cache*)

Tag	Word (w)

- Alamat memori diinterpretasikan sebagai tag dan *word*
- Tag merupakan identitas blok memori
- Setiap satu baris *cache* mempunyai satu tag
- Tag menjadi kata kunci dalam setiap pencarian data

Associative Mapping

- Format alamat memori: (dari sisi *cache*)

Tag	Word (w)

- Alamat memori diinterpretasikan sebagai tag dan *word*
- Tag merupakan identitas blok memori
- Setiap satu baris *cache* mempunyai satu tag
- Tag menjadi kata kunci dalam setiap pencarian data

Associative Mapping - Baca data (1)

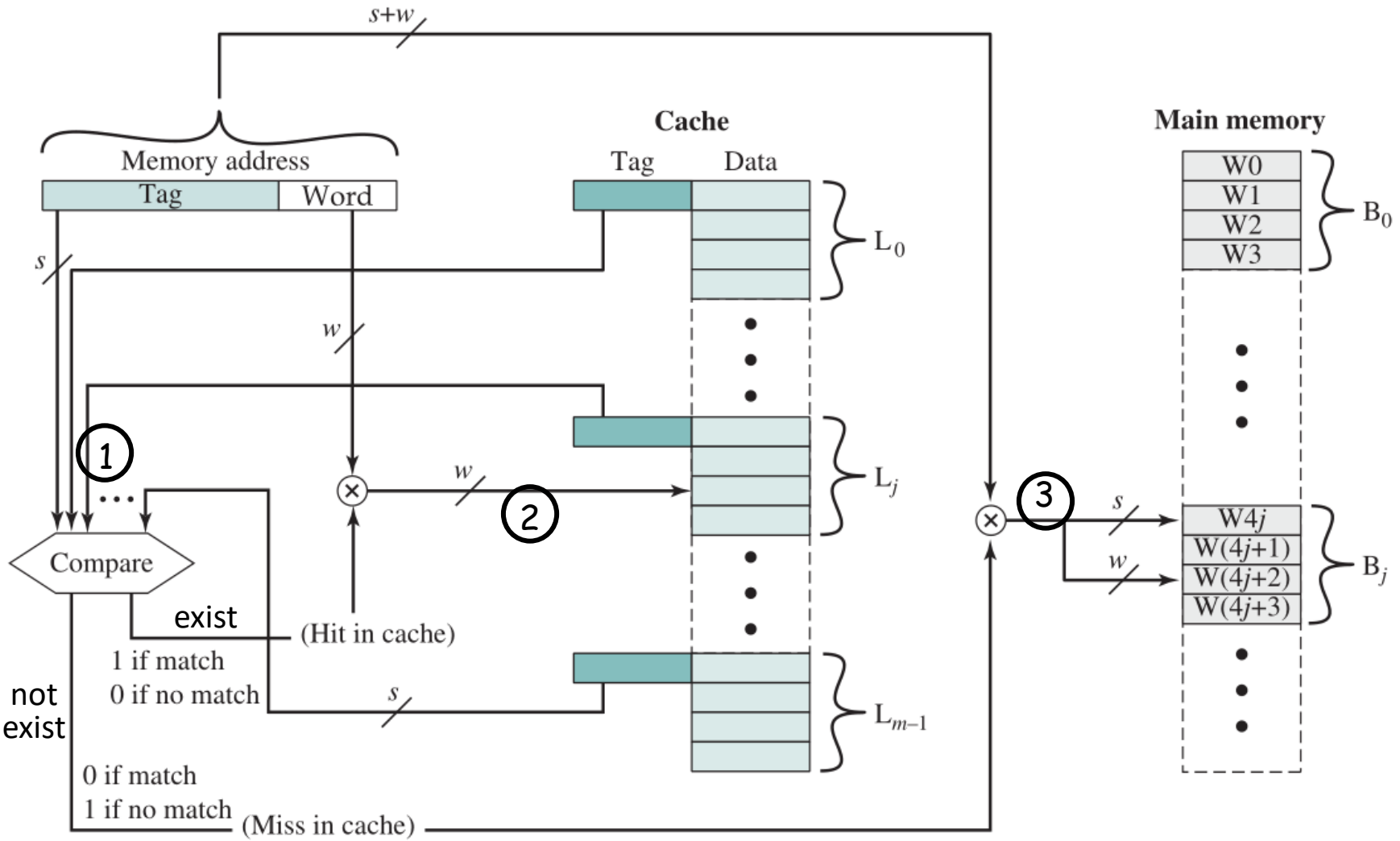


Figure 4.11 Fully Associative Cache Organization

Associative Mapping - Baca data (2)

1. CPU membandingkan nomor tag yang akan dibaca dengan semua nomor tag yang ada di *cache* secara bersamaan
2. Bila nomor tag tsb ada di *cache* (*cache hit*) → pilih *word* yang diinginkan yang terletak pada nomor tag yang diinginkan
3. Bila nomor tag tsb tidak ada di *cache* (*cache miss*) → ambil (*fetch*) satu blok data di memori sesuai dengan nomor tag yang diinginkan

Associative Mapping - Contoh (1)

- Diketahui:
  - *Main memory* berukuran 16 MByte
  - *Cache* berukuran 64 kByte
  - 1 alamat = 1 byte
  - 1x transfer data = 1 blok memori = 1 *line cache* = 4 byte = 4 alamat

  Sebutkan jumlah bit untuk tag dan word (w)!

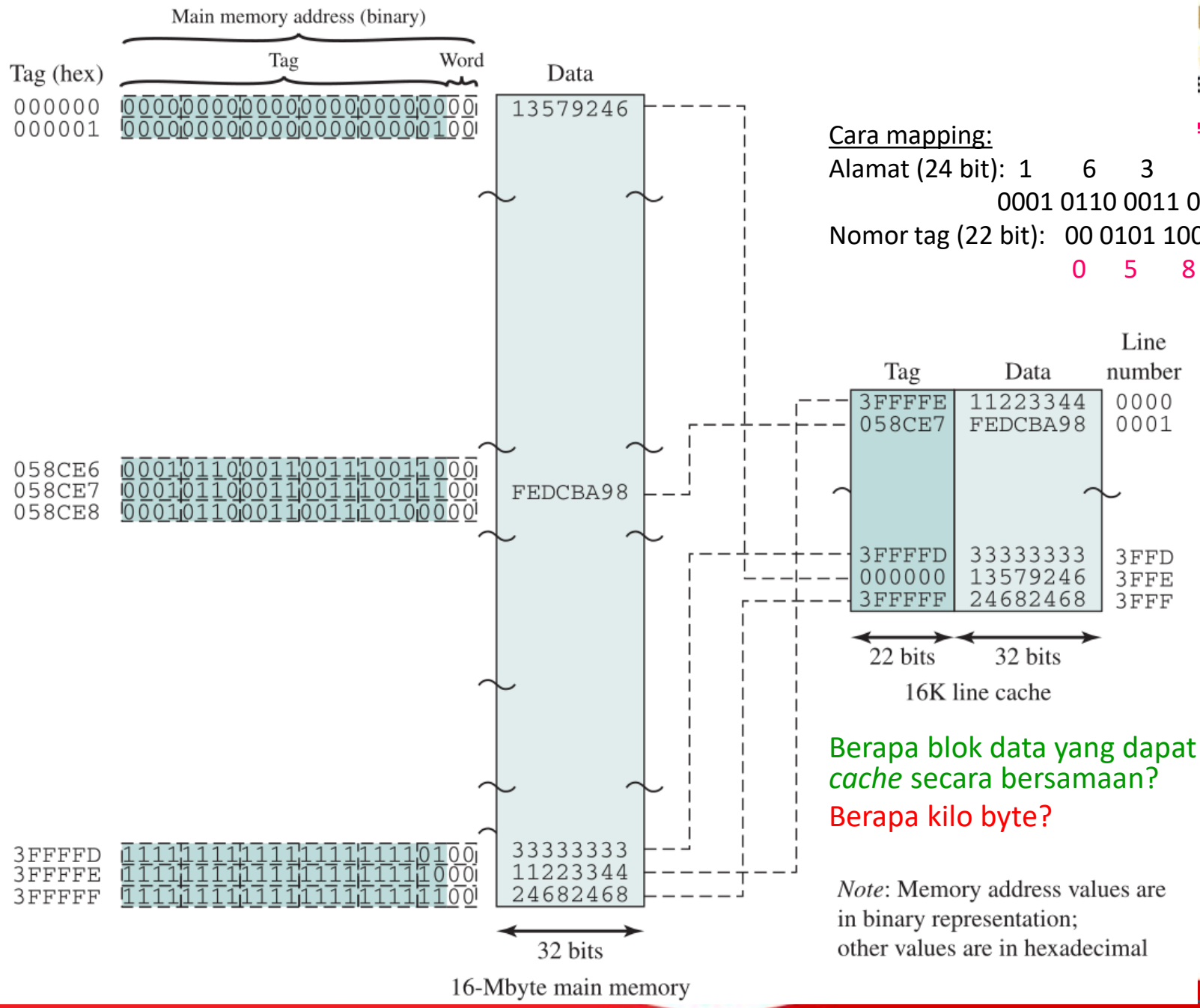
  Gambarkan *mapping*-nya!

  Berikan contohnya!
- Maka:
  - Memori 16 Mbyte = 16 M alamat = $2^4 . 2^{20} = 2^{24}$ → Jumlah bit alamat yang diperlukan = 24 bit
  - Jumlah bit identitas word (w) = 2 bit (1 blok = 4 alamat = 2^2)
  - Jumlah bit tag = 24 – 2 = 22 bit

Associative Mapping - Contoh (2)

- Format alamat memori: (dari sisi *cache*)

- Jumlah **line** cache = 64 kbyte/4 byte = 16 k line
- 16 k = $2^4 . 2^{10} = 2^{14}$ → Jumlah bit line = 14 bit
- Jumlah blok memori = 16 Mbyte/4 byte = 4 M blok = jumlah tag (2^{22})
- 1 *line cache* ≈ 1 blok memori (tag), maka:
  - 1 line cache mempunyai kemungkinan ditempati oleh **4 M blok** yang berbeda
- Setiap tag dapat menempati nomor line yang mana saja (tidak berhubungan dengan nomor baris)
- Setiap blok memori mempunyai satu pasangan nomor tag



Cara mapping:

Alamat (24 bit): 1 6 3 3 9 C

0001 0110 0011 0011 1001 1100

Nomor tag (22 bit): 00 0101 1000 1100 1110 0111

0 5 8 C E 7

Berapa blok data yang dapat ditampung di *cache* secara bersamaan?

Berapa kilo byte?

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

Associative Mapping -
Kelebihan/kekurangan

(+) Mapping setiap blok dapat dilakukan secara fleksibel (tidak terikat pada nomor line tertentu)

(+) Dapat mengatasi masalah *thrashing*

(-) Diperlukan rangkaian yang lebih rumit untuk membandingkan semua tag secara paralel, karena jumlah tag sangat banyak

Referensi

[STA19] Stalling, William. 2019. "*Computer Organization and Architecture: Designing for Performance*". 11th edition. Prentice Hall.