Organisasi dan Arsitektur Komputer CII2A3

Cache Memory (bagian 2)

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	-	-		-	-	-	-	-	-				
11	Arsitektur SAP-3	-	-	-	-	-	-	-	-	-	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	3	-	-	-	7	2		-	-	-	-	-	-	--	--	30	--
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 cara kerja metode *set-associative mapping* pada *cache memory*?

Set Associative Mapping (1)

- *Cache* dibagi menjadi sejumlah set
- Setiap set terdiri dari sejumlah baris
- Setiap blok memori dipasangkan ke nomor set tertentu, tetapi boleh menempati baris mana saja dalam satu set
- Misal: jika tiap set terdiri dari 2 baris, maka:
  - Model pemetaannya disebut: *2-way set associative mapping*
  - Setiap blok memori boleh menempati satu dari dua baris yang tersedia asalkan masih dalam satu set

Set Associative Mapping (2)

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

Tag (s-v)	Set (v)	Word (w)

- Untuk keperluan akses *cache* → setiap alamat memori dibagi menjadi 2 bagian:
  - word(w) = bit-bit identitas word atau *byte* di dalam blok memori
  - s = bit-bit identitas blok memori
    - set field (v): bit-bit nomor set
    - tag (s-v): bit-bit identitas blok data yang ada di memori

Set Associative Mapping - Baca data (1)

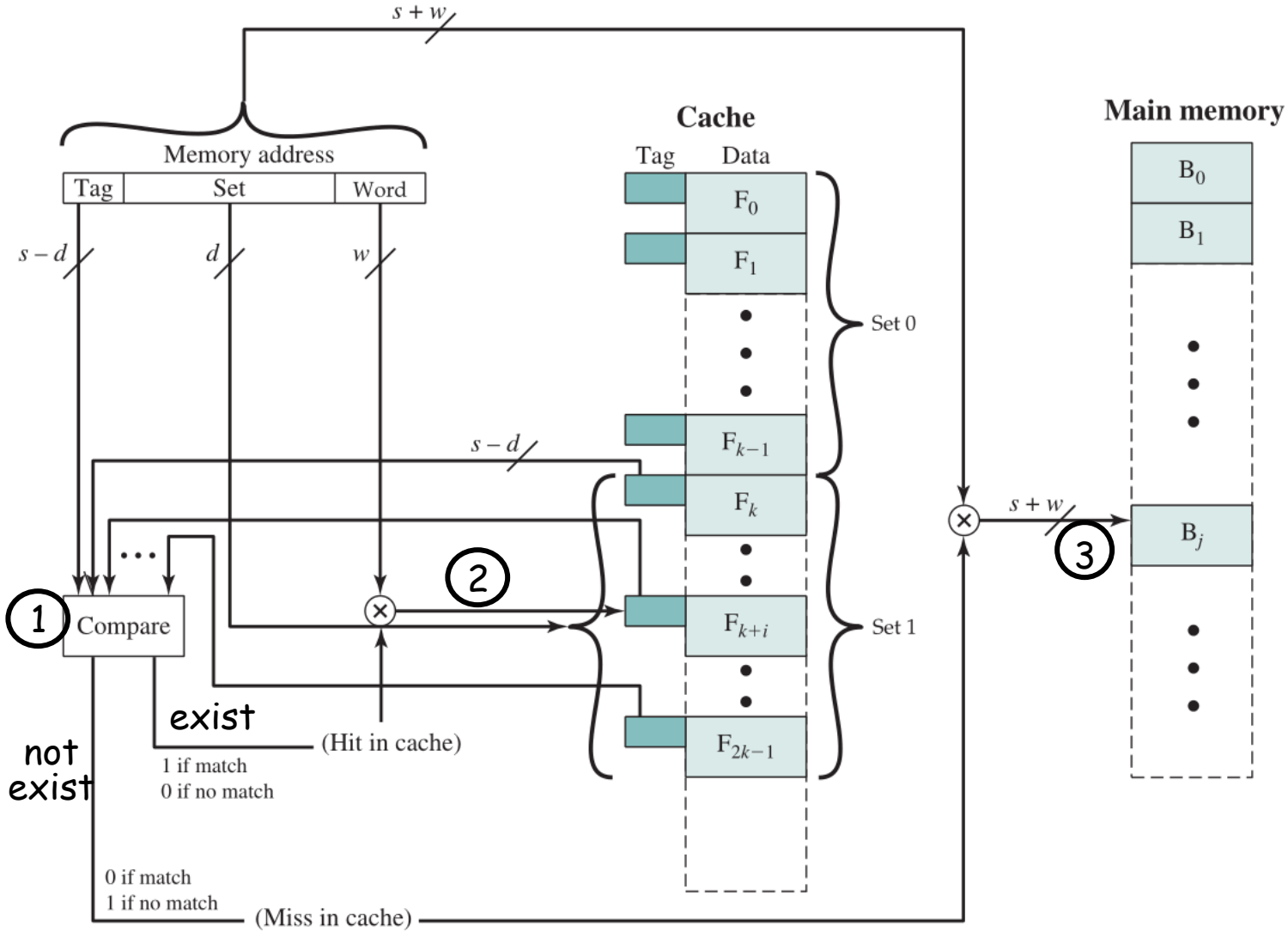


Figure 4.14 *K*-Way Set Associative Cache Organization

Set Associative Mapping - Baca data (2)

① CPU membandingkan nomor tag yang akan dibaca dengan semua nomor tag di *cache*. Tag dalam satu set dibandingkan bersama-sama

② Bila nomor tag tsb ada di *cache* (*cache hit*) → pilih word yang diinginkan yang terletak pada nomor set yang diinginkan

③ Bila nomor tag tsb tidak ada di *cache* (*cache miss*) → ambil (*fetch*) satu blok data di memori sesuai dengan nomor tag dan nomor set yang diinginkan

Set Associative Mapping - Contoh (1)

- Diketahui:
  - *Main memory* berukuran 16 MByte
  - *Cache* berukuran 64 kByte
  - 1 alamat = 1 byte
  - 1x transfer data = 1 blok memori = 4 byte = 4 alamat
  - Model mapping = 2 way set associative

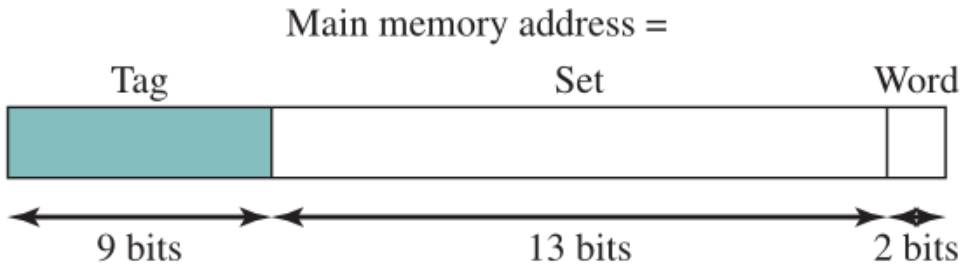
  Sebutkan jumlah bit untuk tag (s-v), set (v), dan word (w) !

  Gambarkan mappingnya !

  Berikan contohnya !
- Maka:
  - Memory 16 Mbyte = $2^4 . 2^{20} = 2^{24}$ → Jumlah bit alamat yang diperlukan = 24 bit
  - Jumlah bit identitas word (w) = 2 bit (1 blok = 4 byte = 2^2)
  - Jumlah ***line*** *cache* = 64 kbyte/4 byte = 16 k line
  - 1 set = 2 line → jumlah set = 16k/2 = 8 k set
  - 8 k = $2^3 . 2^{10} = 2^{13}$ → Jumlah bit set = 13 bit (0000 - 1FFF)
  - Jumlah bit tag = 24 – 13 – 2 = 9 bit (range: 000 – 1FF)
  - Jumlah tag = 2^9 = 512 tag

Set Associative Mapping - Contoh (2)

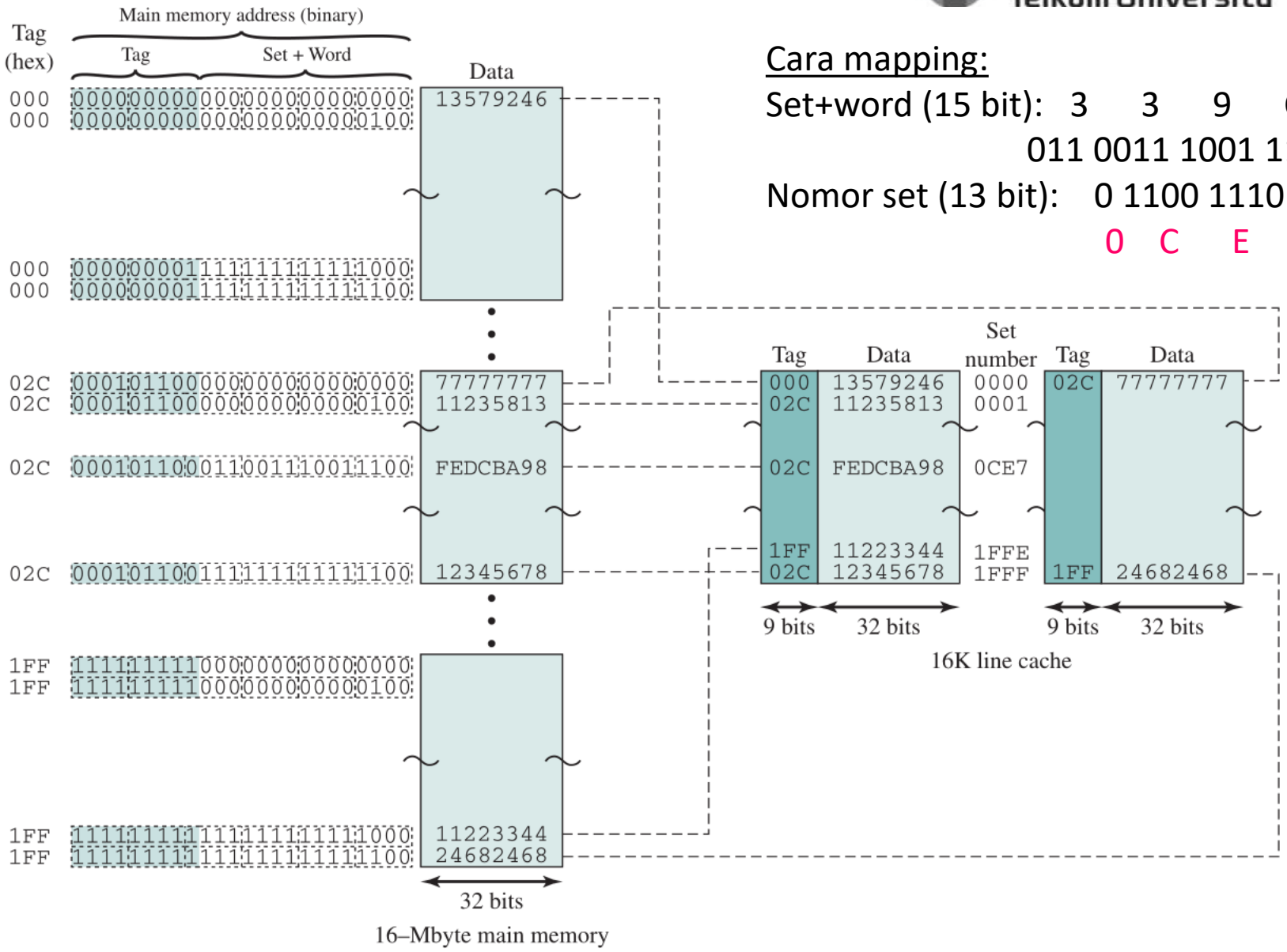
- Format alamat memori: (dari sisi *cache*)



- Jumlah alamat memori tiap tag = $2^{13+2} = 2^5 . 2^{10}$ = 32 k alamat
- Range alamat memori tiap tag (dalam format 24 bit) = 000000 – 007FFF, maka alamat memori : 000000, 008000, 010000, …, FF8000 selalu terletak pada set nomor yang sama (0000)
  - 007FFF = **0000 0000 0**111 1111 1111 1111 + 1 → **0000 0000 1**000 0000 0000 0000 = 008000
  - 008000 + 007FFF = 00FFFF = **0000 0000 1**111 1111 1111 1111 + 1 → **0000 0001 0**000 0000 0000 0000 = 010000
- Setiap alamat memori di atas bebas menempati salah satu line pada nomor set tersebut
- Setiap satu nomor blok memori hanya dapat menempati satu nomor set
- Satu nomor set dapat dapat ditempati oleh:
  - 512 nomor tag berbeda tetapi nomor blok-nya sama

Set Associative Mapping - Contoh (3)

Fakultas Informatika
School of Computing
Telkom Universitu



Cara mapping:

Set+word (15 bit): 3 3 9 C

011 0011 1001 1100

Nomor set (13 bit): 0 1100 1110 0111

0 C E 7

Set Associative Mapping - Kelebihan

(+) Setiap blok memori dapat menempati lebih dari satu kemungkinan nomor *line* (dapat menggunakan line yang kosong), sehingga *thrashing* dapat diperkecil

(+) Jumlah tag lebih sedikit (dibanding model *associative*), sehingga jalur untuk melakukan perbandingan tag lebih sederhana

Set Associative Mapping – Hit Ratio

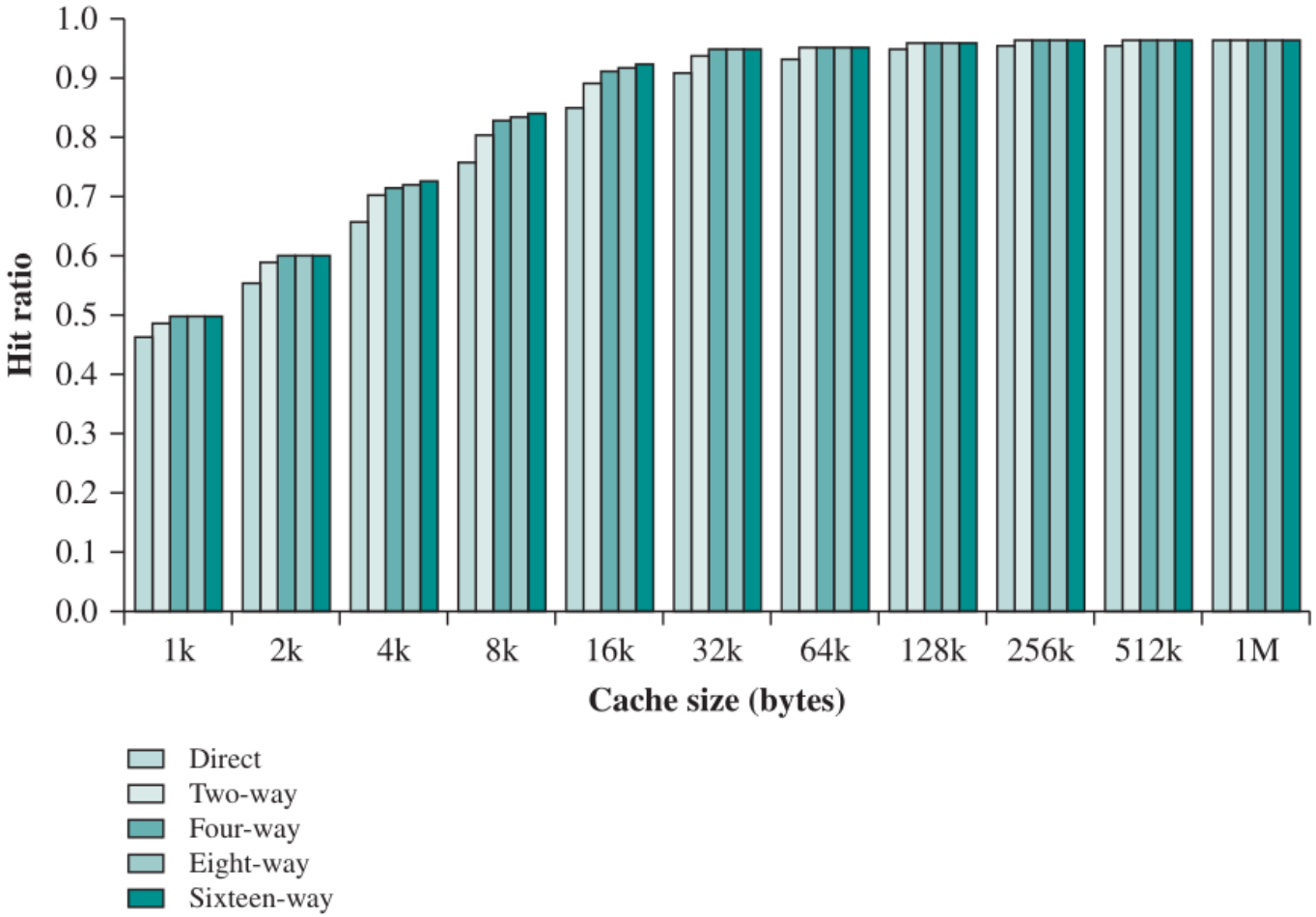


Figure 4.16 Varying Associativity over Cache Size

Part 2: Bagaimana cara kerja metode *replacement algorithm (LRU, FIFO, LFU, random)* pada *cache memory*?

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*)
- Jumlah *cache*
  - *Off-chip, On-chip*
  - *Single level, Multilevel*
  - *Unified, Split*

Replacement Algorithms (1)

- Adalah algoritma yang digunakan untuk memilih blok data mana yang ada di *cache* yang ***dapat diganti*** dengan blok data baru
- *Direct mapping*
  - Tidak perlu algoritma
  - *Mapping* pasti (tidak ada alternatif lain)
- *Associative & Set Associative*
  - Algoritma diimplementasi dengan H/W (supaya cepat)
  - Jenis algoritma:
    - *Least Recently Used (LRU)*
    - *First In First Out (FIFO)*
    - *Least Frequently Used (LFU)*
    - *Random*

Replacement Algorithms (2)

- *Least Recently Used (LRU)*
  - Blok yang diganti adalah blok yang ***paling lama*** di *cache* dan ***tidak digunakan***
  - Kelebihan:
    - Paling efektif
    - Mempunyai hit ratio tinggi → data yang sering digunakan saja yang ditaruh di *cache*
    - Paling mudah diimplementasikan pada *two-way set associative mapping* (digunakan sebuah bit tambahan = USE bit, line yang direfer → USE bit = 1))
  - Contoh:
    - Kapasitas *cache* hanya 4 baris sedangkan jumlah blok data jauh lebih banyak. Jika urutan pengaksesan data adalah:

      a b c d c b a b c kemudian datang data e maka data yang diganti adalah ???

      Jawaban: d

      Kalau data yang diakses sebelum data e adalah d, maka data yang diganti adalah ???

      Jawaban: a (a lebih lama tidak diakses dibanding d)

Replacement Algorithms (3)

- *First In First Out (FIFO)*
  - Blok yang diganti adalah blok yang ***paling awal*** berada di *cache*
  - Contoh:
    - Kapasitas *cache* hanya 4 baris sedangkan jumlah blok data jauh lebih banyak. Jika urutan pengaksesan data adalah:

      a b c d c b a b c kemudian datang data e maka data yang diganti adalah …….

      Jawaban: a (a paling lama/awal berada di *cache*)

      Kalau data yang diakses sebelum data e adalah d, maka data yang diganti adalah …….

      Jawaban: a (a paling lama/awal berada di *cache*)

Replacement Algorithms (4)

- *Least Frequently Used (LFU)*
  - Blok yang ***paling jarang*** digunakan yang diganti
  - Setiap baris mempunyai *counter*
  - Contoh:
    - Kapasitas *cache* hanya 4 baris sedangkan jumlah blok data jauh lebih banyak. Jika urutan pengaksesan data adalah:

      a b c d c b a b c a d kemudian datang data e maka data yang diganti adalah …….

      Jawaban: d (d paling jarang diakses)

      Kalau urutan data yang diakses sebelum data e adalah a b c d c b a b c a d d, maka data yang diganti adalah ……..

      Jawaban: a (nilai *counter* a sama dengan yang lain, tetapi karena a datang paling awal maka a berada pada baris paling awal) → FIFO

      b (paling lama tidak diakses) → LRU
- *Random*
  - Penggantian blok dilakukan secara ***acak***

Part 3: Bagaimana cara menjaga konsistensi data di memori dan *cache memory*?

Elemen Perancangan *Cache*

Fakultas Informatika
School of Computing
Telkom University

- 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*

Write Policy (1)

- Bertujuan untuk memastikan apakah blok di *cache* yang akan diganti telah di-*copy* di memori
- Problem 1: bagaimana jika *main memory* dapat diakses oleh lebih dari satu *device* tanpa melalui *cache* (misal dengan DMA)
- Solusi:

  (a) Write through

  (b) Write back

Write Policy (2)

Fakultas Informatika
School of Computing
Telkom University

(a) Write through

- Setiap ada perubahan data di *cache* selalu di-*copy* di memori, demikian pula sebaliknya
- Kelebihan/kekurangan:

  (+) Teknik paling sederhana

  (–) Menyebabkan *memory traffic* tinggi (*cache – main memory*)

  (–) Dapat terjadi *bottleneck* (bus data penuh)

(b) *Write back*

- *Update* data hanya dilakukan di *cache*
- Jika baris di *cache* akan ditempati data lain → data lama di-*copy* ke memori **hanya jika** data tsb mengalami perubahan
- Kelebihan/kekurangan:

  (+) *Memory traffic* berkurang

  (–) Data di memori tidak valid → akses I/O ke memori harus melalui *cache*

  (–) *Circuitry* lebih kompleks

  (–) Dapat terjadi *bottleneck*

Write Policy (3)

- Problem 2: bagaimana jika terdapat lebih dari satu prosesor dan masing-masing mempunyai *cache* tersendiri?
- Solusi:
  - *Cache* saling berhubungan (*cache coherency*)
    - Macam *cache coherency*:
      - *(a) Bus watching with write through*
      - *(b) H/W transparency*
      - *(c) Non cacheable memory*

Write Policy (4)

Fakultas Informatika
School of Computing
Telkom University

(a) *Bus watching with write through*

- Setiap *cache controller* memonitor baris alamat untuk mengetahui apakah ada *device* lain yang menulis ke memori

(b) *H/W transparency*

- Digunakan H/W tambahan untuk menjamin perubahan data di memori selalu melalui *cache* dan di-*copy*-kan ke *cache-cache* yang lain

(c) *Non cacheable memory*

- Sebagian dari *main memory* di-*sharing* oleh beberapa prosesor
- Data pada *shared memory* tidak di-*copy*-kan ke *cache* → selalu terjadi *cache miss*

Part 4: Mengapa *cache memory* dibuat menjadi beberapa tingkatan/level?

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 Blok Memori

- Ukuran blok memori:
  - Makin besar → *cache hit* makin besar (efek *locality*: bila suatu word ada di suatu blok, maka ada kemungkinan word-word lain yang terdapat di dalam blok tersebut juga akan digunakan
  - Terlalu besar → *cache hit* menurun (jumlah blok yang dapat di-*copy* ke *cache* makin sedikit)
  - Ukuran optimum: 8 – 32 byte

Elemen Perancangan *Cache*

Fakultas Informatika
School of Computing
Telkom University

- 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*

Jenis *cache* (1)

- Jenis *cache memory*:
  - Berdasarkan letaknya:
    - *off-chip cache (eksternal)*
    - *on-chip cache (internal)*
  - Berdasarkan levelnya:
    - *one level cache*
    - *multilevel cache (L1, L2, L3)*
  - Berdasarkan jenis data yang disimpan:
    - *unified cache*
    - *split cache*

(a) *Off-chip cache* (eksternal)

- Terpisah dari chip prosesor
- Komunikasi melalui bus eksternal atau bus khusus

Jenis cache (2)

(b) *On-chip cache* (internal)

– Menjadi satu dengan chip prosesor

(+) Waktu eksekusi lebih cepat → performansi sistem meningkat

(+) Aktifitas bus eksternal berkurang → dapat digunakan untuk keperluan lain

(c) *One-level cache*

– Hanya ada satu level *cache* (sudah ditinggalkan)

(d) *Multilevel cache*

– Terdiri dari 2 level *cache* atau lebih

Jenis cache (3)

- Mengapa perlu 2 level atau lebih???
  - *Cache miss* → CPU akses ke memori, kecepatan memori <<< kecepatan CPU → performansi turun
  - Dengan L2 (SRAM) → mempercepat tersedianya data yang dibutuhkan CPU
- Design L2 *cache*:
  - *Cache* eksternal (dengan bus khusus)
  - *Cache* internal (satu chip dengan prosesor)
- Kelebihan/kekurangan *multilevel cache*:

  (+) Memperbaiki performansi

  (-) Perancangan *cache* lebih rumit (ukuran, algoritma, *write policy*, dll)

Jenis cache (3)

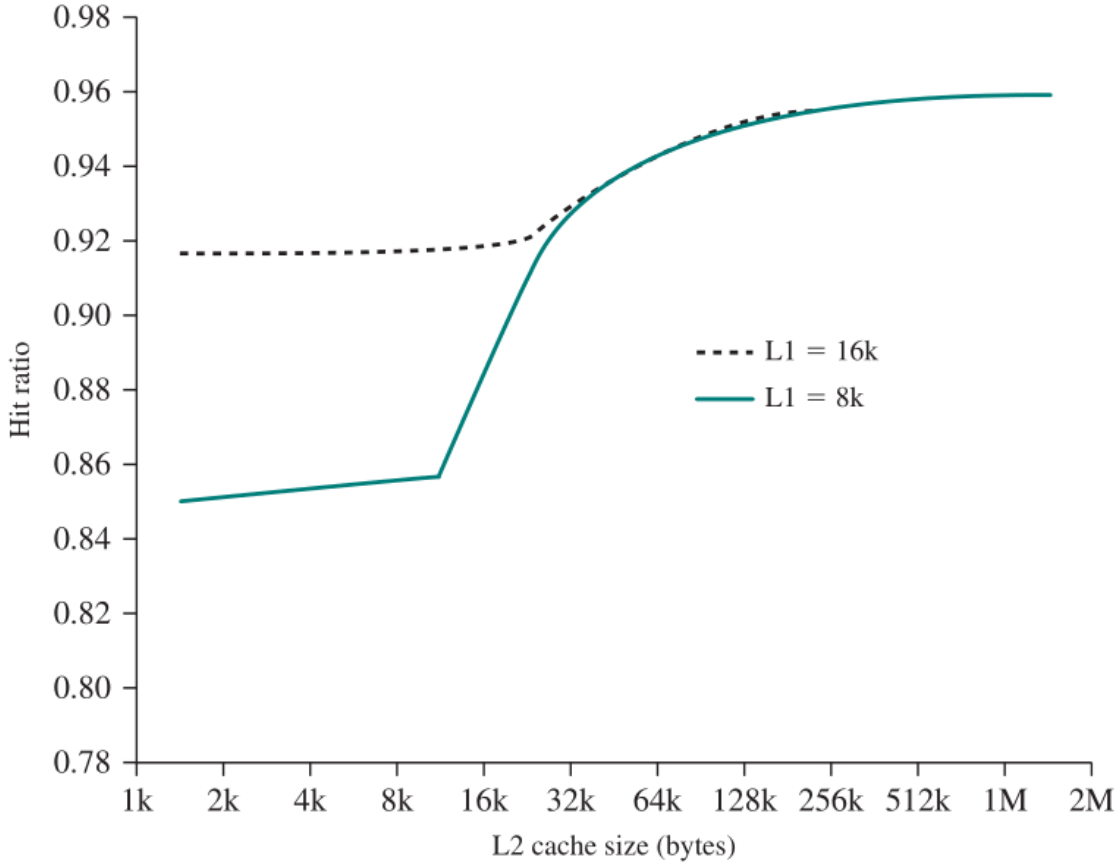


Figure 4.17 Total Hit Ratio (L1 and L2) for 8-Kbyte and 16-Kbyte L1

Jenis cache (4)

(e) *Unified cache*

- Data dan instruksi disimpan pada tempat yang sama (dalam satu *cache*)
- Kelebihan/kekurangan:
  - (+) *Hit rate* lebih tinggi daripada *split cache*
  - (+) Perancangan dan implementasi terfokus pada satu *cache*
  - (-) Tidak mendukung *pipeline*
  - (-) Dapat terjadi *contention* (rebutan) *cache*

Jenis cache (5)

(f) *Split cache* (Pentium, Power PC)

- *Cache* dibagi 2 sehingga data dan instruksi disimpan pada tempat terpisah
- Kelebihan:
  - (+) Mendukung eksekusi instruksi secara paralel
  - (+) Mencegah *contention cache* antara *fetch/decode* instruksi dan eksekusi instruksi → performansi lebih baik

Cache Intel

- 80386 – tanpa *cache*
- 80486:
  - *Single on-chip*
  - Ukuran 8 Kbyte
  - Ukuran line 16 byte
  - *Four way set associative*
- Pentium (semua versi) – 2 buah *on chip cache L1*
  - Untuk data dan instruksi
- Pentium 4
  - L1 *caches*:
    - Ukuran 8 Kbyte
    - Ukuran line 64 byte
    - *Four way set associative*
  - L2 *cache*
    - *Feeding both L1 caches*
    - Ukuran 256 Kbyte
    - Ukuran line 128 byte
    - *8 way set associative*

Cache Intel

Table 4.4 Intel Cache Evolution

Problem	Solution	Processor on Which Feature First Appears
External memory slower than the system bus.	Add external cache using faster memory technology.	386
Increased processor speed results in external bus becoming a bottleneck for cache access.	Move external cache on-chip, operating at the same speed as the processor.	486
Internal cache is rather small, due to limited space on chip.	Add external L2 cache using faster technology than main memory.	486
Contention occurs when both the Instruction Prefetcher and the Execution Unit simultaneously require access to the cache. In that case, the Prefetcher is stalled while the Execution Unit's data access takes place.	Create separate data and instruction caches.	Pentium
Increased processor speed results in external bus becoming a bottleneck for L2 cache access.	Create separate back-side bus that runs at higher speed than the main (front-side) external bus. The BSB is dedicated to the L2 cache.	Pentium Pro
	Move L2 cache on to the processor chip.	Pentium II
Some applications deal with massive databases and must have rapid access to large amounts of data. The on-chip caches are too small.	Add external L3 cache.	Pentium III
	Move L3 cache on-chip.	Pentium 4

Blok Diagram Pentium 4 (*Simplified*)

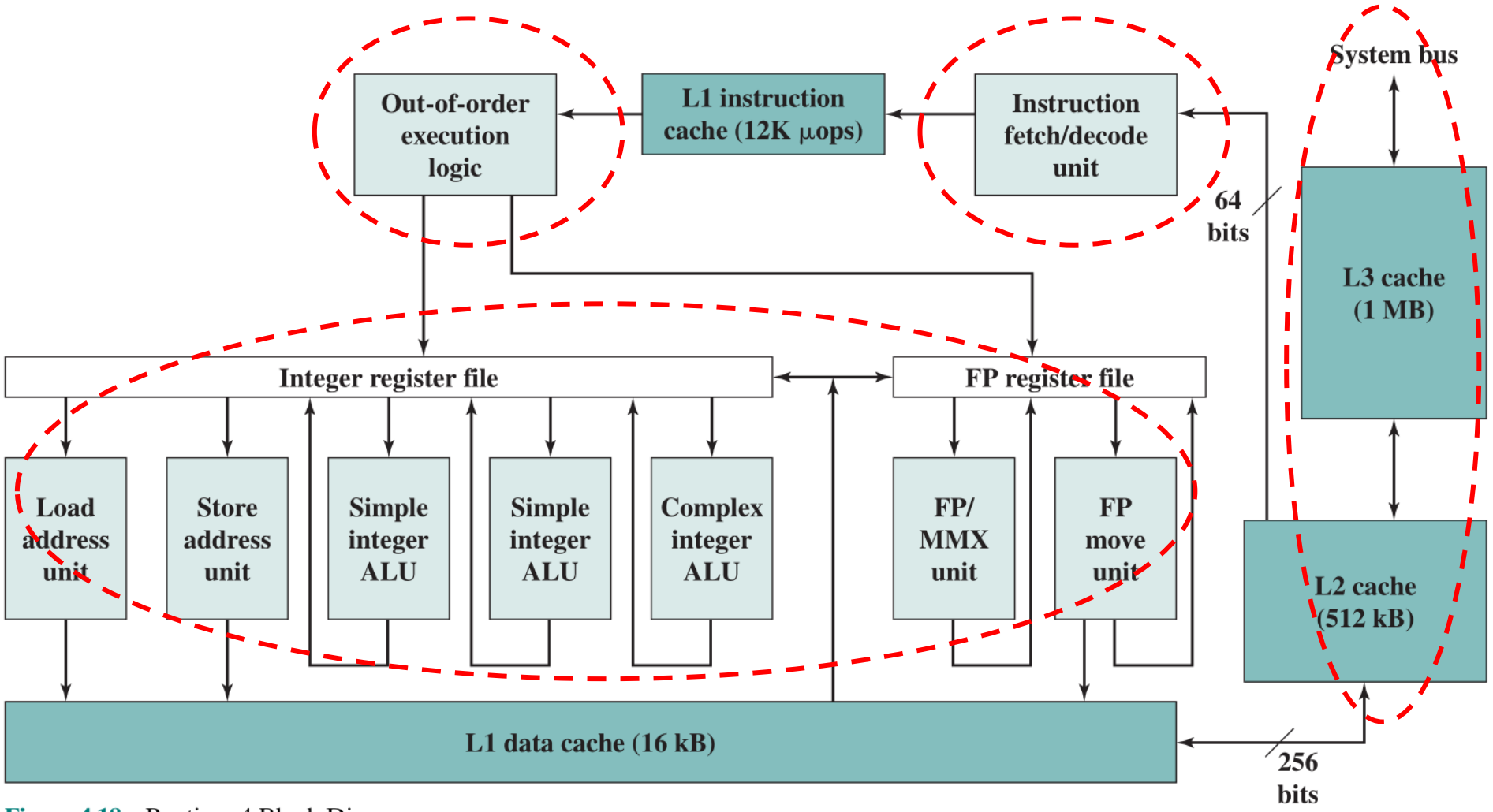


Figure 4.18 Pentium 4 Block Diagram

Bagian Utama Prosesor Pentium 4 (1)

- *Fetch/Decode Unit*
  - Mengambil instruksi dari *cache* L2
  - Men-*decode* instruksi menjadi mikro operasi
  - Menyimpan hasilnya (mikro operasi) di *cache* instruksi L1
- *Out of order execution logic*
  - Membuat jadual eksekusi mikro operasi sesuai dengan data dan *resource* yang tersedia
  - Membuat jadual eksekusi mikro yang mungkin akan digunakan (spekulasi)
- *Execution units*
  - Mengeksekusi mikro operasi
  - Mengambil data yang diperlukan dari *cache* L1
  - Menyimpan hasil pemrosesan ke *register*

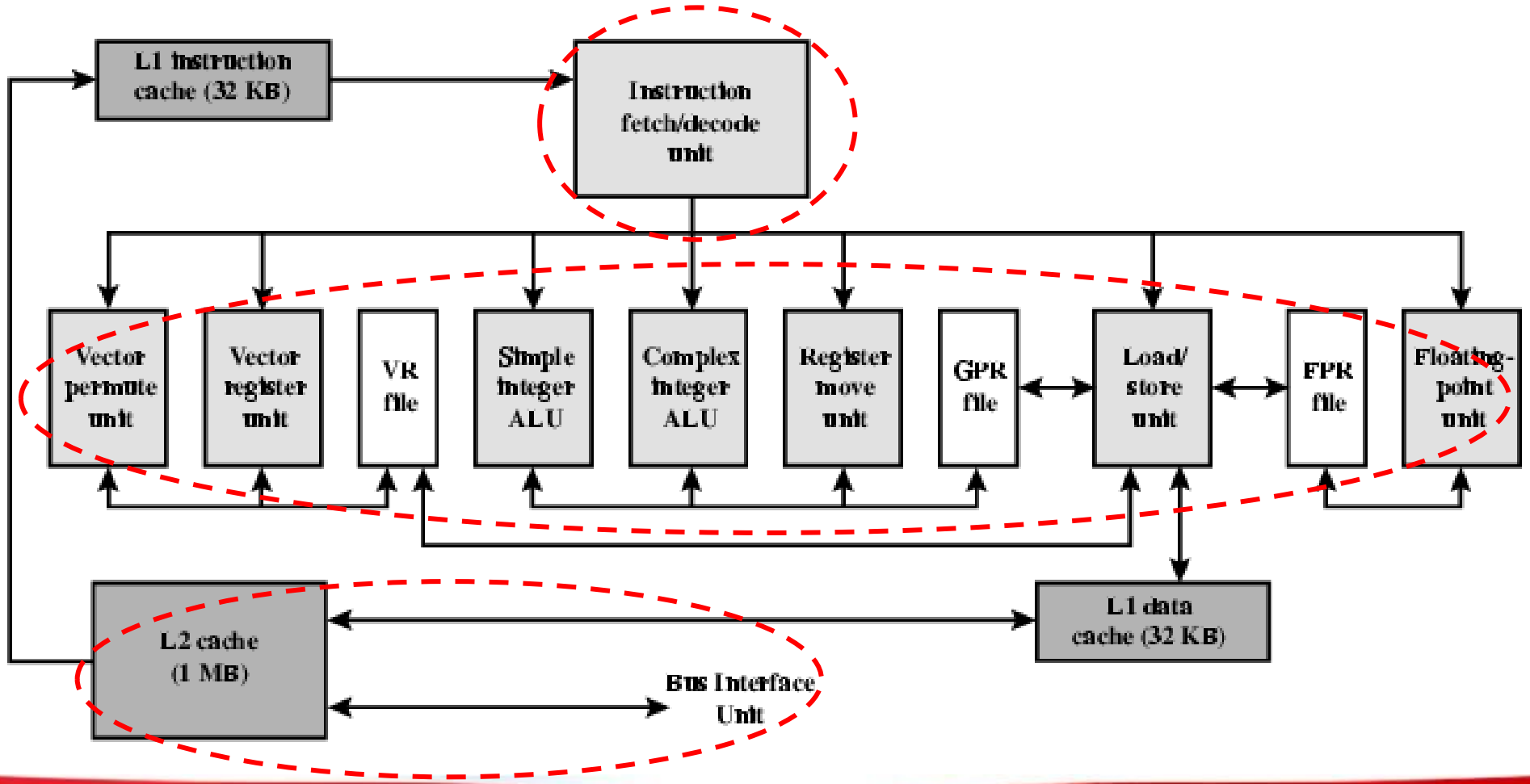
Bagian Utama Prosesor Pentium 4 (2)

- *Memory subsystem*
  - Terdiri dari *cache* L2 dan sistem bus
  - Untuk mengakses main memory jika terjadi *cache miss*
  - Untuk mengakses *resource* I/O

Organisasi *Cache* Power PC

- 601 – *single* 32 Kbyte, *8 way set associative*
- 603 – 16 Kb (2 x 8 Kb) *two way set associative*
- 604 – 32 Kbyte
- 610 – 64 Kbyte
- G3 & G4
  - L1 *cache:*
    - 64 Kbyte
    - *8 way set associative*
  - L2 *cache*
    - 256 Kbyte, 512k atau 1M
    - *two way set associative*

Blok Diagram Power PC G4



ARM Cache Features

Table 4.6 ARM Cache Features

Core	Cache Type	Cache Size (kB)	Cache Line Size (words)	Associativity	Location	Write Buffer Size (words)
ARM720T	Unified	8	4	4-way	Logical	8
ARM920T	Split	16/16 D/I	8	64-way	Logical	16
ARM926EJ-S	Split	4-128/4-128 D/I	8	4-way	Logical	16
ARM1022E	Split	16/16 D/I	8	64-way	Logical	16
ARM1026EJ-S	Split	4-128/4-128 D/I	8	4-way	Logical	8
Intel StrongARM	Split	16/16 D/I	4	32-way	Logical	32
Intel Xscale	Split	32/32 D/I	8	32-way	Logical	32
ARM1136-JF-S	Split	4-64/4-64 D/I	8	4-way	Physical	32

ARM Cache Features

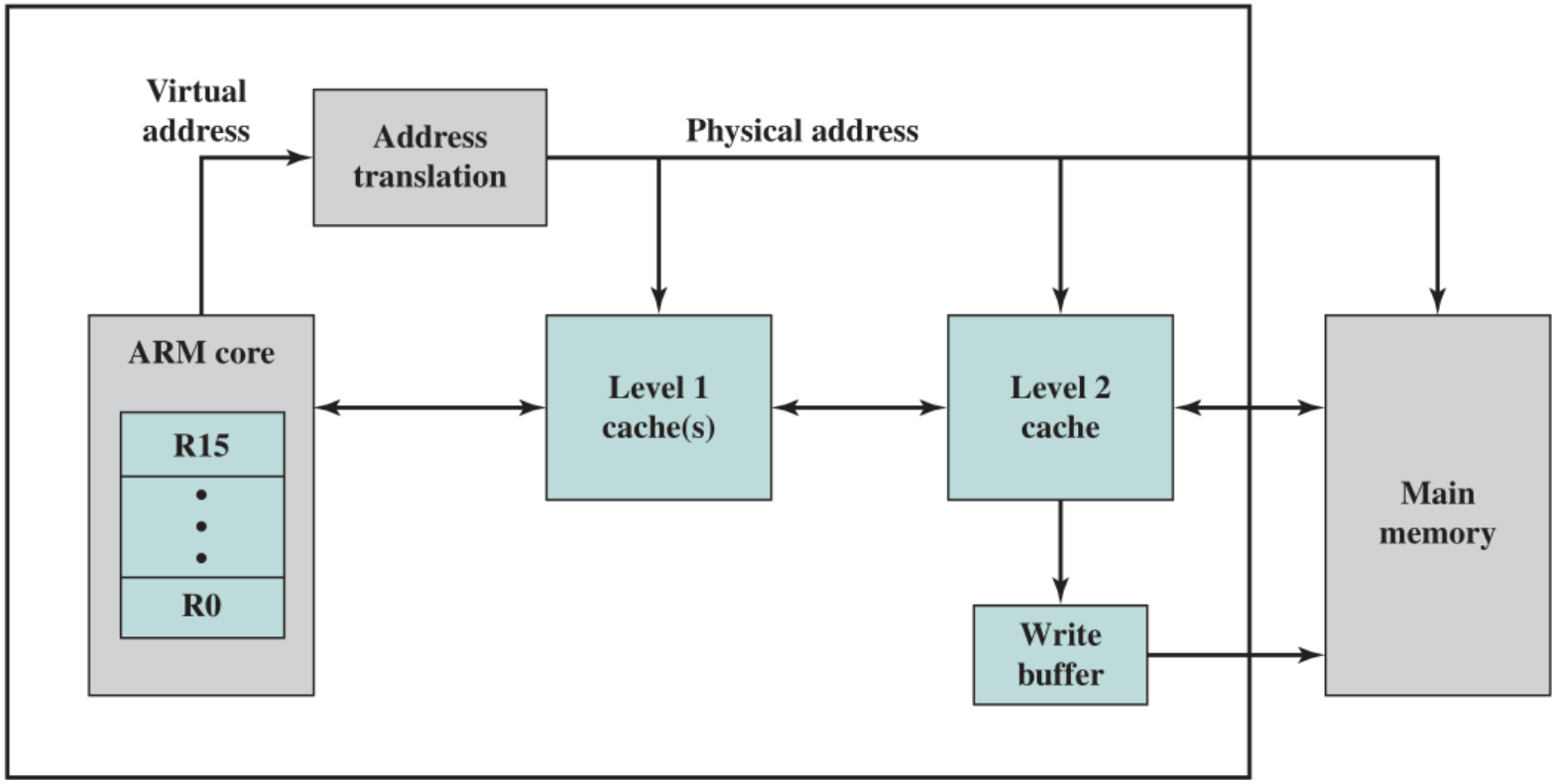


Figure 4.19 ARM Cache and Write Buffer Organization

Perbandingan Ukuran *Cache*

Processor	Type	Year of Introduction	L1 Cache[a]	L2 Cache	L3 Cache
IBM 360/85	Mainframe	1968	16–32 kB	—	—
PDP-11/70	Minicomputer	1975	1 kB	—	—
VAX 11/780	Minicomputer	1978	16 kB	—	—
IBM 3033	Mainframe	1978	64 kB	—	—
IBM 3090	Mainframe	1985	128–256 kB	—	—
Intel 80486	PC	1989	8 kB	—	—
Pentium	PC	1993	8 kB/8 kB	256–512 kB	—
PowerPC 601	PC	1993	32 kB	—	—
PowerPC 620	PC	1996	32 kB/32 kB	—	—
PowerPC G4	PC/server	1999	32 kB/32 kB	256 kB to 1 MB	2 MB
IBM S/390 G6	Mainframe	1999	256 kB	8 MB	—
Pentium 4	PC/server	2000	8 kB/8 kB	256 kB	—
IBM SP	High-end server/ supercomputer	2000	64 kB/32 kB	8 MB	—
CRAY MTA[b]	Supercomputer	2000	8 kB	2 MB	—
Itanium	PC/server	2001	16 kB/16 kB	96 kB	4 MB
Itanium 2	PC/server	2002	32 kB	256 kB	6 MB
IBM POWER5	High-end server	2003	64 kB	1.9 MB	36 MB
CRAY XD-1	Supercomputer	2004	64 kB/64 kB	1 MB	—
IBM POWER6	PC/server	2007	64 kB/64 kB	4 MB	32 MB
IBM z10	Mainframe	2008	64 kB/128 kB	3 MB	24–48 MB
Intel Core i7 EE 990	Workstation/ server	2011	6 × 32 kB/ 32 kB	1.5 MB	12 MB
IBM zEnterprise 196	Mainframe/ server	2011	24 × 64 kB/ 128 kB	24 × 1.5 MB	24 MB L3 192 MB L4

Referensi

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