**ORGANISASI DAN ARSITEKTUR KOMPUTER**

**CACHE MEMORY**

****

**Program Studi Teknik Informatika**

**Fakultas Ilmu Komputer**

**Universitas Esa Unggul**

**Jakarta**

**2018**

Memori Cache

# **A. Karakteristik sistem memori**

Karakteristik yang jelas dari memori adalah kapasitasnya. Untuk memori internal, ini
biasanya dinyatakan dalam bentuk byte (1 byte = 8 bit) atau kata-kata. Kata umum panjangnya 8, 16, dan 32 bit. Kapasitas memori eksternal biasanya dinyatakan dalam hal byte.

Konsep terkait adalah unit transfer. Untuk memori internal, unit transfer sama dengan jumlah saluran listrik masuk dan keluar dari memori modul. Ini bisa saja sama dengan panjang kata, tetapi lebih sering dalam ukuran besar, seperti 64, 128, atau 256 bit. Untuk memperjelas poin ini, pertimbangkan tiga konsep terkait untuk memori internal:

1. Word (Kata) : Unit "alami" dari dalam organisasi memori. Ukuran kata biasanya sama dengan jumlah bit yang digunakan untuk merepresentasikan integer dan instruksi panjangnya.

2. Addresableunits (Unit yang dialamatkan) : Di beberapa sistem, unit beralamat adalah kata. Namun, banyak sistem yang mengalamatkannya dalam bentuk byte. Jadi, hubungannya antara panjang dalam bit A dari suatu alamat dan jumlah N dari unit beralamat adalah 2A = N.

3. Unit transfer: Untuk memori utama, ini adalah jumlah bit yang terbaca atau ditulis ke dalam memori pada suatu waktu. Unit transfer tidak sama dengan kata atau unit beralamat. Untuk memori eksternal, data sering banyak mentransfer unit yang lebih besar dari sebuah kata, dan ini disebut sebagai blok.

 Perbedaan lain di antar jenis memori adalah metode mengakses unit data, yaitu:

1. Akses Berurutan : Memori diatur ke dalam unit data, yang disebut daftar riwayat. Akses harus dibuat dalam urutan linier tertentu. Menyimpan alamat informasi digunakan untuk memisahkan catatan dan membantu dalam proses pengambilan.Mekanisme baca-tulis digunakan dan ini harus dipindahkan dari lokasinya saatke lokasi yang diinginkan, melewati dan menolak setiap antar catatan. Dengan demikian, waktu untuk mengakses catatan yang sewenang-wenang sangat bervariasi.

2. Akses langsung : Seperti akses berurutan, akses langsung melibatkan mekanisme baca-tulis bersama. Namun, pencekalan atau catatan individu memiliki alamat unik berdasarkan lokasi fisik. Akses dilakukan dengan akses langsung untuk mencapai suatu tempat umum ditambah pencarian berurutan, menghitung, atau menunggu untuk mencapai lokasi akhir. Sekali lagi, waktu akses kembali bervariasi

3. Akses acak : Setiap lokasi beralamat di memori memiliki unik, secara fisikmekanisme pengalamatan kabel. Waktu untuk mengakses lokasi tertentu bersifat independendari urutan akses sebelumnya dan konstan. Jadi, lokasi apa saja
dapat dipilih secara acak dan langsung ditangani dan diakses. Memori utama
dan beberapa sistem cache adalah akses acak.

4. Asosiatif : Ini adalah jenis akses acak dari memori yang memungkinkan seseorang untuk membuat perbandingan lokasi bit yang diinginkan dalam kata untuk pertandingan yang ditentukan, dan untuk melakukan ini untuk semua kata secara bersamaan. Jadi, sebuah kata diambil berdasarkan sebagian isinya daripada alamatnya. Seperti halnya memori akses acak biasa, setiap lokasi memiliki mekanisme pengalamatannya sendiri, dan waktu pengambilan selalu independen dari lokasi atau pola akses sebelumnya.

## 1. Hirarki Memori

Untuk mencapai kinerja terbaik, memori harus mampu mengikuti prosesor. Yaitu, karena prosesor sedang menjalankan instruksi, kami tidak ingin menunggu menunggu instruksi atau operand. Untuk sistem praktis, biaya memori harus masuk akal dalam hubungan dengan komponen lain.

Seperti yang diharapkan, ada trade-off antara tiga karakteristik kunci dari memori: kapasitas, waktu akses, dan biaya. Berbagai teknologi digunakan untuk mengimplementasikan sistem memori, dan di seluruh spektrum teknologi ini, hubungan berikut ini berlaku:
1. Waktu akses yang lebih cepat, biaya per bit lebih besar;
2. Kapasitas yang lebih besar, biaya per bit lebih kecil;
3. Kapasitas lebih besar, waktu akses lebih lambat.



# **B. Prinsip Memori Cache**

Memori cache dirancang untuk menggabungkan waktu akses memori yang mahal, kecepatan memori yang tinggi dikombinasikan dengan ukuran memori yang lebih besar serta lebih murah, dan kecepatan memori yang rendah.

Cache berisi salinan bagian-bagian memori utama. Saat prosesor mencoba untuk membaca kata di memori, ia akan melakukan pemeriksaan untuk menentukan apakah kata tersebut ada di dalam cache. Jika iya, kata tersebut dikirimkan ke prosesor. Jika tidak, satu blok memori utama, yang terdiri dari sejumlah kata yang tetap, dibaca ke dalam cache dan kemudian kata itu dikirimkan ke prosesor.

Jalur data dan alamat juga melampirkan data dan alamat penyangga, yang menempel pada bus sistem dari mana memori utama tercapai. Ketika cache hit terjadi, buffer data dan alamat dinonaktifkan dan komunikasi hanya antara prosesor dan cache, tanpa lalu lintas sistem bus. Ketika cachemiss terjadi, alamat yang diinginkan dimuat ke bus sistem dan data dikembalikan melalui buffer data ke cache dan prosesor.

Di organisasi lain, cache secara fisik dialihkan antara prosesor dan memori utama untuk semua data, alamat, dan garis kontrol. Dalam kasus terakhir ini, untuk cachemiss, kata yang diinginkan pertama kali dibaca ke cache dan kemudian ditransfer dari cache ke prosesor.

# **C. Element di Desain Cache**



## 1. Alamat Cache

Virtual memory adalah fasilitas yang memungkinkan program untuk mengatasi memori dari sudut pandang logis, tanpa memperhatikan jumlah memori utama yang tersedia secara fisik. Ketika memori virtual digunakan, bidang alamat instruksi mesin berisi alamat virtual.

Salah satu keunggulan nyata dari cache logis adalah kecepatan akses cache lebih cepat daripada cache fisik, karena cache dapat merespon sebelum MMU (MemoryManagement Unit/Manajemen Unit Memori) bekerja terjemahan alamat. Kerugiannya mengatakan bahwa kebanyakan sistem memori virtual menyediakan setiap aplikasi dengan alamat virtual memori yang beruangan sama. Artinya, setiap aplikasi melihat memori virtual yang dimulai pada alamat 0.

## 2.Ukuran Cache

Ada beberapa penjelasan lainnya untuk meminimalkan ukuran cache. Semakin besar cache, semakin besar angkanya gerbang yang terlibat dalam mengalamatkan cache. Hasilnya adalah bahwa cache yang besar cenderung menjadi sedikit lebih lambat daripada yang kecil. Bahkan,
ketika dibangun dengan teknologi sirkuit terpadu yang sama dan diletakkan di tempat yang sama pada chip dan papan sirkuit.

## 3. MappingFunction

Mapping function diperlukan untuk menentukan blok memori utama yang saat ini menempati jalur cache. Pilihan fungsi pemetaan menentukan bagaimana cache diatur. Tiga teknik dapat digunakan: langsung, asosiatif, dan set-asosiatif.

a) Direct Mapping/Pemetaan Langsung

Teknik paling sederhana, yang dikenal sebagai pemetaan langsung, memetakan setiap blok memori utama menjadi hanya satu baris cache.

Teknik pemetaan langsung berbentuk sederhana dan murah untuk diterapkan. Kerugian utama adalah bahwa ada lokasi cache yang tetap di setiap blok yang diberikan. Jadi,
jika suatu program terjadi kata referensi berulang kali dari dua blok berbeda memetakan ke baris yang sama, maka blok akan terus bertukar di cache, dan rasio akan menjadi rendah (fenomena yang dikenal sebagai labrakan).

b) AssociativeMapping/Pemetaan Asosiatif

Pemetaan asosiatif mengatasi kerugian pemetaan dengan mengizinkan setiap blok memori utama untuk dimuat ke setiap baris cache. Dengan pemetaan asosiatif, ada fleksibilitas dimana blok akan digantikan oleh blok baru yang dibaca ke cache. Kerugian utama dari asosiatif pemetaan adalah sirkuit kompleks diperlukan untuk memeriksa semua tagdari cache garis secara paralel

c) Set AssociativeMapping

Set - pemetaan asosiatif adalah kompromi yang menunjukkan kekuatan pendekatan secara langsung dan asosiatif sekaligus mengurangi kerugian mereka.
Untuk pemetaan set-asosiatif, logika kontrol cache menafsirkan alamat memori sebagai tiga bidang: Tag, Set, dan Word.

Dengan pemetaan asosiatif penuh, tag di alamat memori cukup besar dan harus dibandingkan dengan tag setiap baris dalam cache.

## 4. Penempatan ulang algoritma

Untuk mencapai kecepatan tinggi, Algoritma harus diimplementasikan dalam perangkat keras.

Sejumlah algoritma harus dicoba. Mekanisme cache tersebut mempertahankan daftar indeks yang terpisah ke semua baris dalam cache.

Ketika suatu garis direferensikan, ia bergerak ke depan daftar. Untuk penggantian, garis
di belakang daftar digunakan. Karena kesederhanaan pelaksanaannya, LRU (*LeastRecentlyUsed*/Baru saja dipakai) adalah algoritma penggantian paling populer.

Kemungkinan lain adalah first-in-first-out (FIFO)/masuk pertama keluar pertama : Mengganti blok di set yang telah berada di cache paling lama. FIFO mudah diimplementasikan sebagai teknik round-robin atau circularbuffer. Masih ada kemungkinan lain yang paling tidak sering digunakan, yaitu (LFU): Ganti blok itu di set yang telah mengalami referensi paling sedikit. LFU bisa dilaksanakan
dengan mengaitkan penghitung dengan setiap baris.

## 5. WritePolicy/Kebijakan Menulis

Berbagai kebijakan menulis, dengan kinerja dan ekonomi, mungkin terjadi. Ada dua masalah yang harus dihadapi. Pertama, lebih dari satu perangkat mungkin memiliki akses ke memori utama. Masalah yang lebih rumit terjadi ketika beberapa prosesor dilampirkan pada bus yang sama dan setiap prosesor memiliki cache lokalnya sendiri. Kemudian, jika kata diubah dalam satu cache, itu bisa membatalkan kata di cache lain.

Modul prosesor-cache lainnya dapat memonitor lalu lintas ke memori utama untuk menjaga konsistensi dalam cache-nya sendiri. Kerugian utama dari teknik ini adalah lalu lintas memori yang cukup besar dan dapat membuat kemacetan.

Kemungkinan pendekatan untuk koherensi cache mencakup hal-hal berikut:

1. Bus mengawasi dengan menulis melalui: Setiap pengontrol cache memonitor alamatgaris untuk mendeteksi operasi tulis ke memori oleh master bus lainnya. Jika master lain menulis kelokasi dalam memori bersama yang juga berada dalam memori cache, pengontrol cachemembatalkan entri cache tersebut. Strategi ini tergantung pada penggunaan kebijakan write-through oleh semua pengontrol cache.

2. Transparansi perangkat keras: Perangkat keras tambahan digunakan untuk memastikan semuanyapembaruan ke memori utama melalui cache tercermin di semua cache. Jadi, jika satu prosesor mengubah kata dalam cache, pembaruan ini ditulis ke memori utama. DiSelain itu, kata-kata yang cocok dalam cache lain juga diperbarui.

3. Memori non-cache: Hanya sebagian memori utama dibagi oleh lebih dari satu prosesor, dan ini ditetapkan sebagai non-cache. Dalam sistem seperti itu, semua akses ke memori bersama adalah cachemisses, karena memori bersamatidak pernah disalin ke cache. Memori yang tidak bisa di-cache dapat diidentifikasimenggunakan pemilihan cip logika atau bit alamat tinggi.

## 6. Line Size (Ukuran Blok)

Karena ukuran blok meningkat dari ukuran yang sangat kecil ke ukuran yang lebih besar, rasio klik pada awalnya akan meningkat karena prinsip lokalitas, yang menyatakan bahwa data dalam
sekitar kata yang direferensikan kemungkinan akan direferensikan dalam waktu dekat. Sebagai
ukuran blok meningkat, lebih baiknya data dibawa ke cache.
Dua efek spesifik ikut bermain:

* Blok yang lebih besar dapat mengurangi jumlah blok yang masuk ke dalam cache. Karena masing-masing blockfetch menimpa isi cache lama, sejumlah kecil hasil blok dalam data akan ditimpa setelah mereka diambil.
* Sebagai blok besar, setiap kata tambahan menjauh dari yang diminta kata dan karena itu kurang dibutuhkan dalam waktu dekat.



# **D. Number of Caches**

## 1. MultilevelCaches (Cache Bertingkat)

 Ketika kepadatan logika telah meningkat, kemungkinan besar untuk memiliki cache pada chip yang sama dengan prosesor: cacheon-chip. Dibandingkan dengan cache yang dapat dijangkau melalui bus eksternal, cacheon-chip mengurangi aktivitas bus eksternal prosesor. Jadi, hal itu dapat mempercepat waktu eksekusi dan meningkatkan keseluruhan kinerja sistem.

 Ketika instruksi atau data yang diminta ditemukan dalam cacheon-chip, akses bus dihilangkan. Karena jalur data internal ke prosesor singkat, dibandingkan dengan panjang bus, akses cacheon-chip akan menyelesaikan lebih cepat daripada siklus siklus bus zero-wait (Awal, tunggu). Selanjutnya, selama periode ini bus bebas untuk mendukung transfer lainnya.

## 2. Unified Vs SplitCaches (Cache yang Disatukan vs Cache yang dipisah)

 Ketika cacheon-chip pertama kali muncul, banyak desain terdiri dari cache tunggal yang digunakan untuk menyimpan referensi ke data dan instruksi. Baru-baru ini, secara umumnya, cache dibagi menjadi dua: satu untuk instruksi dan satunya lagi untuk data.

 Ada dua keuntungan potensial dari cache yang disatukan:

1) Untuk ukuran cache yang diberikan, cache yang bersatu memiliki tingkat klik lebih tinggi daripada cacheyang dipisah, karena menyeimbangkan beban antara instruksi dan penjemputan data secara otomatis. Yaitu, jika pola eksekusi melibatkan lebih banyak instruksi yang diambil daripada penjemputan data, maka cache akan cenderung untuk mengisi dengan instruksi, dan jika eksekusi. dan jika pola eksekusi melibatkan lebih banyak data yang diambil, hal yang sebaliknya akan terjadi.

2) Hanya satu cache yang perlu dirancang dan diimplementasikan.

 Keuntungan utama dari desain cache yang dipisah adalah bahwa ia menghilangkan pertikaian untuk cache antara unit pengambilan / dekode instruksi dan unit eksekusi. Ini penting dalam setiap desain yang bergantung pada instruksi jalur pipa. Biasanya, prosesor akan mengambil instruksi sebelumnya dan mengisi buffer, atau jalur pipa, dengan instruksi yang akan dieksekusi.

# **E. Lokalitas**

Dasar untuk keunggulan kinerja dari memori dua tingkat adalah prinsip yang dikenal
sebagai lokalitas referensi [DENN68]. Prinsip ini menyatakan bahwa referensi memori
cenderung mengelompok. Selama jangka waktu yang panjang, kelompok-kelompok yang digunakan berubah, tetapi selama periode waktu yang singkat, prosesor terutama bekerja dengan kelompok referensi memori tetap.

Secara intuitif, prinsip lokalitas masuk akal. Pertimbangkan baris berikutpemikiran:

1. Pengecualian untuk instruksi cabang dan panggilan, yang hanya merupakan sebagian kecil dari semua instruksi program, eksekusi program bersifat sekuensial. Oleh karena itu, dalam banyak kasus, instruksi berikutnya yang akan diambil segera mengikuti instruksi terakhir yang diambil.

2. Sangat jarang memiliki urutan panggilan prosedur yang panjang tanpa gangguan diikuti
oleh urutan pengembalian yang sesuai. Sebaliknya, program tetap terbataske jendela prosedur yang agak sempit tingkat kedalaman panggilannya. Dengan demikian, selama periode waktu yang singkat referensi ke instruksi cenderung dilokalisasi ke beberapa prosedur.

3. Kebanyakan konstruksi iteratif terdiri dari sejumlah instruksi yang relatif kecildiulang berkali-kali. Selama iterasi, perhitungan dibatasi pada bagian kecil yang berdekatan dari suatu program.

4. Dalam banyak program, banyak perhitungan melibatkan pemrosesan struktur data, seperti susunan atau urutan rekaman. Dalam banyak kasus, referensi berturut-turut untuk struktur data ini akan menjadi data yang terletak dekat



Letak Cache Memory di komputer

• Terdapat di dalam Processor (on chip ),Cache internal diletakkan dalam prosesor sehingga tidak memerlukan bus eksternal, maka waktu aksesnya akan sangat cepat sekali. • Terdapat diluar Processor(of

• L1 cache terintegrasi dengan chip prosesor, artinya letak L1 cache sudah menyatu dengan chip prosesor (berada di dalam keping prosesor).

• L2 cache, ada yang menyatu dengan chip prosesor, ada pula yang terletak di luar chip prosesor, yaitu di motherboard dekat dengan posisi dudukan prosesor. Pada era prosesor intel 80486 atau sebelumnya, letak L2 cache kebanyakan berada di luar chip prosesor. Chip cache terpisah dari prosesor, berdiri mandiri dekat chip prosesor. Sejak era prosesor Intel Pentium, letak L2 cache ini sudah terintegrasi dengan chip prosesor (menyatu dengan keping prosesor). Posisi L2 cache selalu terletak antara L1 cache dengan memori utama (RAM).

• L3 cache belum diimplementasikan secara umum pada semua jenis prosesor. Hanya prosesor-prosesor tertentu yang memiliki L3 cache. Kecepatan cache memory

• Transfer data dari L1 cache ke prosesor terjadi paling cepat dibandingkan L2 cache maupun L3 cache (bila ada).

• Kecepatannya mendekati kecepatan register.

\*Register adalah memori berukuran sangat kecil dengan kecepatan akses sangat tinggi.

• Secara fisik L1 cache tidak bisa dilihat dengan mata telanjang.

• L1 cache adalah lokasi pertama yang diakses oleh prosesor ketika mencari pasokan data. Kapasitas simpan datanya paling kecil, antara puluhan hingga ribuan byte tergantung jenis prosesor. Pada beberapa jenis prosesor pentium kapasitasnya 16 KB yang terbagi menjadi dua bagian, yaitu 8 KB untuk menyimpan instruksi, dan 8 KB untuk menyimpan data.

• Transfer data tercepat kedua setelah L1 cache adalah L2 cache. Prosesor dapat mengambil data dari cache L2 yang terintegrasi (on-chip) lebih cepat dari pada cache L2 yang tidak terintegrasi.

• Kapasitas simpan datanya lebih besar dibandingkan L1 cache, antara ratusan ribu byte hingga jutaan byte, ada yang 256 KB, 512 KB, 1 MB, 2 MB, bahkan 8 MB, tergantung jenis prosesornya.

• Kapasitas simpan data untuk L3 cache lebih besar lagi, bisa ratusan juta byte (ratusan mega byte). Perbedaan L1, L2, dan L3 L1 L2 L3 Cache L1 adalah memori yang utama. Cache L2 adalah memori yang kedua (sekunder) Cache L3 memiliki kapasitas lebih besar dari Cache L2 Kecepatannya sama dengan kecepatan processor Kecepatannya dibawah kecepatan Cache L1 Lebih lambat dari Cache L2 tetapi lebih cepat dari memori utama

### Rasio (Kena) dan Waktu Akses

Kemungkinan bahwa suatu kata (word) berupa data/instruksi ditemukan dalam cache (disebut kena atau hit) sehingga prosesor tidak perlu mencarinya dalam memori utama, akan tergantung pada program, ukuran dan organisasi cache. Bila kata yang diperlukan tidak ada dalam cache (berarti luput atau miss), maka prosesor harus merujuknya ke memori utama. Rasio kena (h) didefinisikan sebagai perbandingan antara jumlah perujukan yang berhasil memperoleh kata dari cache dengan banyaknya perujukan yang dilakukan.

|  |
| --- |
| **h = (jumlah perujukan yang berhasil) / ( jumlah perujukan)** |

Dalam studi tentang cache, pengukuran umumnya justru terhadap rasio luput (miss) yang besarnya adalah:

|  |
| --- |
| **m = (1 – h)** |

Waktu akses rata-rata, dengan asumsi bahwa perujukan selalu dilakukan ke cache lebih dahulu sebelum ke memori utama, dapat dihitung sebagai berikut:

|  |
| --- |
| **t a = t c + (1-h) t m** |

Keterangan :

**ta         =**adalah waktu akses rata-rata,

**tc**         =          adalah waktu akses cache dan tm adalah waktu akses ke memori utama.

Setiap kali prosesor terpaksa mengakses memori utama, diperlukan tambahan waktu akses sebesar tm(1-h). Misalnya, bila rasio kena adalah 0,85, waktu akses ke memori utama adalah 200 ns dan waktu akses ke cache adalah 25 ns, maka waktu akses rata-rata adalah 55 ns.

Bila persamaan ta disusun ulang, dapat ditulis menjadi:

|  |
| --- |
| **ta = t c {1/k + (1-h)}** |

dengan **K** adalah rasio antara waktu akses memori utama dengan waktu akses cache (tm/tc).

Dari persamaan di atas dapat dilihat bahwa waktu akses rata-rata didominasi oleh rasio waktu akses memori utama dengan cache bila k kecil. Pada kasus di atas, dengan waktu akses memori utama 200 ns dan waktu akses cache 25 ns, maka k = 8. Rasio luput 1 prosen menyebabkan waktu akses rata-rata menjadi 27 ns, tidak jauh beda dengan waktu akses cache. Pada umumnya k berkisar antara 3-10.

## MAPPING CACHE MEMORY

**Mapping (Pemetaan)**

Saluran cache lebih sedikit dibandingkan dengan blok memori utama sehingga diperlukan algoritma untuk pemetaan blok-blok memori utama ke dalam saluran cache. Selain itu, diperlukan juga alat untuk menentukan blok memori utama mana yang sedang memakai saluran cache. Pemilihan fungsi pemetaan akan menentukan bentuk organisasi cache. Terdapat tiga metode yang digunakan yaitu :

#### 1.      Pemetaan Langsung (Direct Mapping)

Pemetaan langsung adalah teknik yang paling sederhana, yaitu teknik ini memetakan blok memori utama hanya ke sebuah saluran cache saja. Jika suatu block ada di cache, maka tempatnya sudah tertentu. Keuntungan dari direct mapping adalah sederhana dan murah. Sedangkan kerugian dari direct mapping adalah suatu blok memiliki lokasi yang tetap (jika program mengakses 2 blok yang di map ke line yang sama secara berulang-ulang, maka cache-miss sangat tinggi).

Berikut penjelasan lebih detail :

§  Setiap blok pada main memory dipetakan dengan line tertentu pada *cache*. ***i = j modulo C***di mana i adalah nomor line pada cache yang digunakan untuk meletakkan blok main memory ke-j.

§  Jika M = 64 dan C = 4, maka pemetaan antara line dengan blok menjadi

seperti berikut :

Line 0 can hold blocks 0, 4, 8, 12, ...

            Line 1 can hold blocks 1, 5, 9, 13, ...

            Line 2 can hold blocks 2, 6, 10, 14, ...

            Line 3 can hold blocks 3, 7, 11, 15, ...

§  Pada cara ini, *address*pada main memory dibagi 3 *field*atau bagian, yaitu:

o    Tag identifier.

o    Line number identifier

o    Word identifier (offset)

§  *Word identifier*berisi informasi tentang lokasi word atau unit *addressable*lainnya dalam line tertentu pada cache.

§  *Line identifier*berisi informasi tentang nomor fisik (bukan logika) line pada chace

§  *Tag identifier*disimpan pada cache bersama dengan blok pada *line*.

o   Untuk setiap alamat memory yang dibuat oleh CPU, line tertentu yang menyimpan copy alamat tsb ditentukan, jika blok tempat lokasi data tersebut sudah dikopi dari main memory ke *cache*.

o   *Tag*yang ada pada line akan dicek untuk melihat apakah benar blok yang dimaksud ada line tsb.



Gambar 2.1 : Gambar Organisasi Direct Mapping.

Keuntungan Menggunakan Direct Mapping antara lain :

§  Mudah dan Murah diimplementasikan

§  Mudah untuk menentukan letak salinan data main memory pada chace.

Kerugian menggunakan Direct Mapping antara lain :

§  Setiap blok *main memory*hanya dipetakan pada 1 line saja.

§  Terkait dengan sifat lokal pada *main memory*, sangat mungkin mengakses blok yang dipetakan pada *line*yang sama pada *cache*. Blok seperti ini akan menyebabkan seringnya sapu masuk dan keluar data ke/dari *cache*, sehingga *hit ratio*mengecil. *Hit ratio*adalah perbandingan antara jumlah ditemukannya data pada cache dengan jumlah usaha mengakses *cache*.



Gambar 2.2 : Gambar Contoh Pengalamatan Direct Mapping.

Ringkasan *direct mapping*nampak pada tabel berikut:

|  |  |
| --- | --- |
| **Item** | **Keterangan** |
| Panjang alamat | (s+w) bits |
| Jumlah unit yang dapat dialamati | 2s+w words or bytes |
| Ukuran Bloks sama dengan ukuran Line | 2w words or bytes |
| Jumlah blok memori utama | 2s+ w/2w = 2s |
| Jumlah line di chace | M = 2r |
| Besarnya tag | (s - r) bits |

#### 2.      Pemetaan Asosiatif (Associative Mapping)

Pemetaan asosiatif mengatasi kekurangan pemetaan langsung dengan cara mengizinkan setiap blok memori utama untuk dimuatkan ke sembarang saluran cache. Dengan pemetaan assosiatif, terdapat fleksibilitas penggantian blok ketika blok baru dibaca ke dalam cache. Kekurangan pemetaan asosiatif yang utama adalah kompleksitas rangkaian yang diperlukan untuk menguji tag seluruh saluran cache secara parallel, sehingga pencarian data di cache menjadi lama.

§  Memungkinkan blok diletakkan di sebarang line yang sedang tidak terpakai.

§  Diharapkan akan mengatasi kelemahan utama *Direct Mapping*.

§  Harus menguji setiap *cache*untuk menemukan blok yang diinginkan.

o   Mengecek setiap tag pada line

o   Sangat lambat untuk *cache*berukuran besar.

§  Nomor line menjadi tidak berarti. *Address main memory*dibagi menjadi 2 field saja, yaitu tag dan *word offset*.



Gambar 2.3 : Gambar Organisasi *Associative Mapping*.

§  Melakukan pencarian ke semua tag untuk menemukan blok.

§  Cache dibagi menjadi 2 bagian :

o   lines dalam SRAM

o   tag dalam associative memory



Gambar 2.4 : Gambar Contoh Pengalamatan Associative Mapping

Keuntungan *Associative Mapping*: Cepat dan fleksibel.

Kerugian *Associative Mapping*: Biaya Implementasi, misalnya untuk cache ukuran 8 kbyte dibutuhkan 1024 x 17 bit *associative memory*untuk menyimpan *tag identifier*.

Ringkasan *Associative Mapping*nampak pada tabel berikut:

|  |  |
| --- | --- |
| **Item** | **Keterangan** |
| Panjang alamat | (s+w) bits |
| Jumlah unit yang dapat dialamati | 2s+w words or bytes |
| Ukuran Bloks sama dengan ukuran Line | 2w words or bytes |
| Jumlah blok memori utama | 2s+ w/2w = 2s |
| Jumlah line di chace | Undetermined |
| Besarnya tag | s bits |

#### 3.      Pemetaan Asosiatif Set (Set Associative Mapping)

Pada pemetaan ini, cache dibagi dalam sejumlah sets. Setiap set berisi sejumlah line. Pemetaan asosiatif set memanfaatkan kelebihan-kelebihan pendekatan pemetaan langsung dan pemetaan asosiatif.

§  Merupakan kompromi antara *Direct*dengan *Full Associative Mapping*.

§  Membagi cache menjadi sejumlah set (v) yang masing-masing memiliki sejumlah line (k)

§  Setiap blok dapat diletakkan di sebarang line dengan nomor set: ***nomor set = j modulo v***



Gambar 2.5 : Gambar Organisasi K-Way Set Associative Mapping.

§  Jika sebuah set dapat menampung X line, maka cache disebut memiliki X *way set associative cache*.

§  Hampir semua *cache*yang digunakan saat ini menggunakan organisasi 2 atau 4-*way set associative mapping*.



Gambar 2.6 : Gambar Contoh Pengalamatan 2-Way Associative Mapping.

Keuntungan menggunakan *Set Associative Mapping*antara lain:

 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.

Ringkasan *Set* *Associative Mapping*nampak pada tabel berikut:

|  |  |
| --- | --- |
| **Item** | **Keterangan** |
| Panjang alamat | (s+w) bits |
| Jumlah unit yang dapat dialamati | 2s+w words or bytes |
| Ukuran Bloks sama dengan ukuran Line | 2w words or bytes |
| Jumlah blok memori utama | 2d |
| Jumlah line dalam set | k |
| Jumlah set | V=2d |
| Jumlah line di chace | Kv = k\*2d |
| Besarnya tag | ( s – d )bits |

### 2.2       Algoritma Penggantian

Algoritma penggantian adalah suatu mekanisme pergantian blok-blok dalam memori cache yang lama dengan data baru. Dalam pemetaan langsung tidak diperlukan algoritma ini, namun dalam pemetaan assosiatif dan asosiatif set, algoritma ini mempunyai peranan penting untuk meningkatkan kinerja cache memori.

Banyak algoritma penggantian yang telah dikembangkan, algoritma yang paling efektif adalah Least Recently Used (LRU), yaitu mengganti blok data yang terlama berada dalam cache dan tidak memiliki referensi. Algoritma lainnya adalah First In First Out (FIFO), yaitu mengganti blok data yang awal masuk. Kemudian Least Frequently Used (LFU) adalah mengganti blok data yang mempunyai referensi paling sedikit. Teknik lain adalah algoritma Random, yaitu penggantian tidak berdasarkan pemakaian datanya, melainkan berdasar slot dari beberapa slot kandidat secara acak.

**2.3       Write Policy**

Apabila suatu data telah diletakkan pada cache maka sebelum ada penggantian harus dicek apakah data tersebut telah mengalami perubahan. Apabila telah berubah maka data pada memori utama harus di-update. Masalah penulisan ini sangat kompleks, apalagi memori utama dapat diakses langsung oleh modul I/O, yang memungkinkan data pada memori utama berubah,lalu bagaimana dengan data yang telah dikirim pada cache? Tentunya perbedaan ini menjadikan data tidak valid.

Teknik yang dikenal diantaranya, write through, yaitu operasi penulisan melibatkan data pada memori utama dan sekaligus pada cache memori sehingga data selalu valid. Kekurangan teknik ini adalah menjadikan lalu lintas data ke memori utama dan cache sangat tinggi sehingga mengurangi kinerja system, bahkan bisa terjadi hang.

Teknik lainyya adalah write back, yaitu teknik meminmasi penulisan dengan cara penulisan pada cache saja. Pada saat akan terjadi penggantian blok data cache maka baru diadakan penulisan pada memori utama. Masalah yang timbul adalah manakala dat di memori utama belum di-update telah diakses modul I/O sehingga data di memori utama tidak valid.

Penggunaan multi cache terutama untuk multi prosesor akan menjumpai masalah yang lebih kompleks. Masalah validasi data tidak hanya antara cache dan memori utama saja, namun antar cache juga harus diperhatikan. Pendekatan penyelesaian masalah yang dapat dilakukan adalah dengan :

Bus Watching with Write Through. Yaitu setiap cache controller akan memonitoring bus alamat untuk mendeteksi adanya operasi tulis. Apabila ada operasi tulis di alamat yang datanya digunakan bersama maka cache controller akan menginvalidasi data cache-nya.

Hardware Transparency. Yaitu adanya perangkat keras tambahan yang menjamin semua updating data memori utama melalui cache direfleksikan pada seluruh cache yang ada.

Non Cacheable Memory. Yaitu hanya bagian memori utama tertentu yang digunakan secara bersama. Apabila ada pengaksesan data yang tidak di-share merupakan kegagalan cache.

### 2.4     Miss Cache

Saat miss menulis, anda bisa punya pilihan antara membawa blok ke *cache*(*writeallocate*) atau tidak (*write*-*no*-*allocate*). Saat miss membaca, anda selalu membawa blok ke *cache*(lokalitas spasial atau temporal) - blok mana yang diganti:

    tidak ada pilihan untuk *direct-mapped cache*

    memilih secara acak way yang akan diganti

    mengganti *way*yang paling jarang dipakai (LRU)

    penggantian FIFO (*round-robin*)

Tipe *miss cache*adalah sebagai berikut:

    *Miss*wajib: terjadi saat pertama kali word memori diakses. Merupakan *miss*untuk *cache*yang infinit.

    *Miss*kapasitas: terjadi karena program menyentuh banyak word yang lain sebelum menyentuh ulang *word*yang sama. Merupakan miss untuk *cache fullyassociative*.

    *Miss*konflik: terjadi karena dua *work*dipetakan ke lokasi yg sama di *cache*. Merupakan *miss*yang terjadi ketika berganti dari *cache fully*-associ*a*tive ke *direct*-*mapped*.