**ORGANISASI DAN ARSITEKTUR KOMPUTER**

****

**Kelompok 3**

Satria Alief Putra Hidayat – 140810210051

Prames Ray Lapian – 140810210059

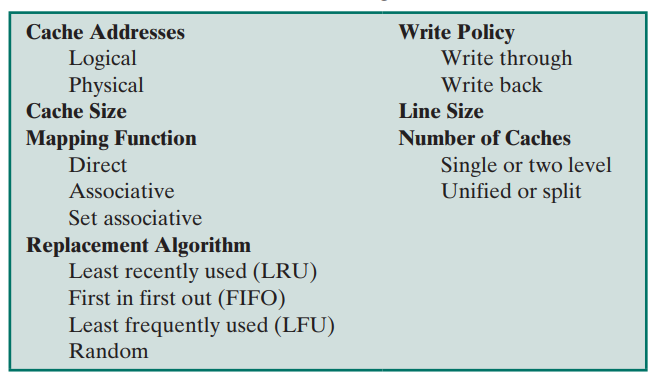
UNIVERSITAS PADJADJARAN

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

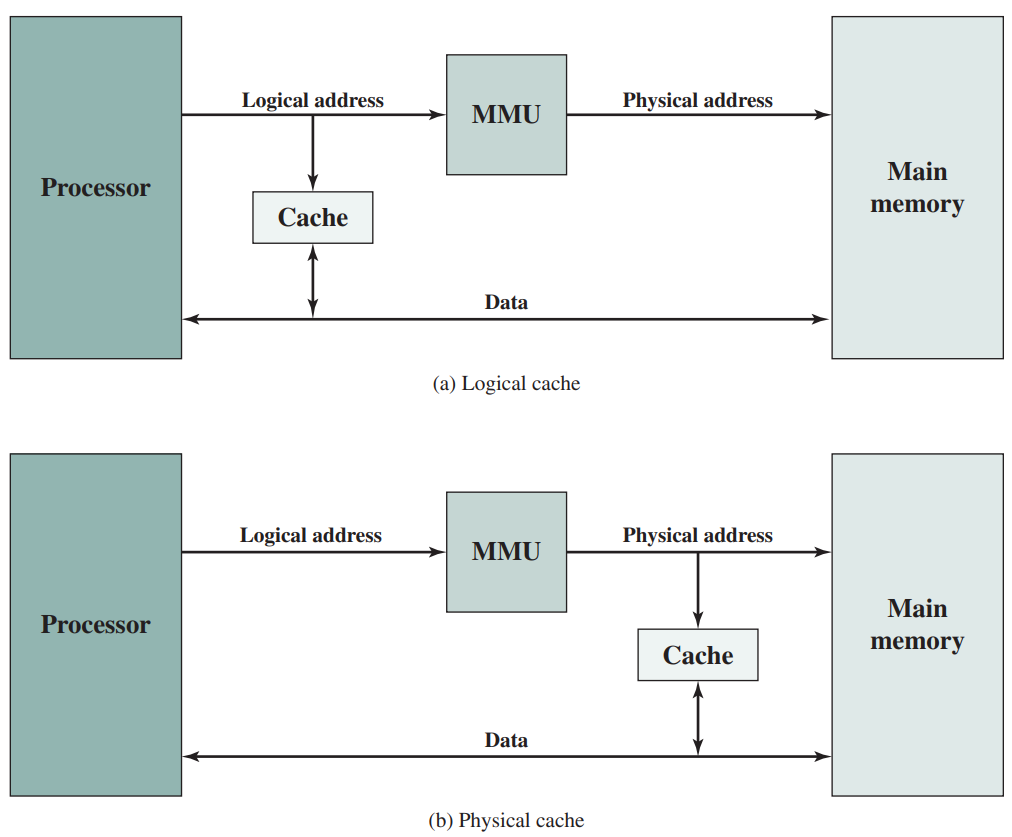
Program Studi INFORMATIKA

2022

**ELEMENTS OF CACHE DESIGN**

****

**Cache Addresses**



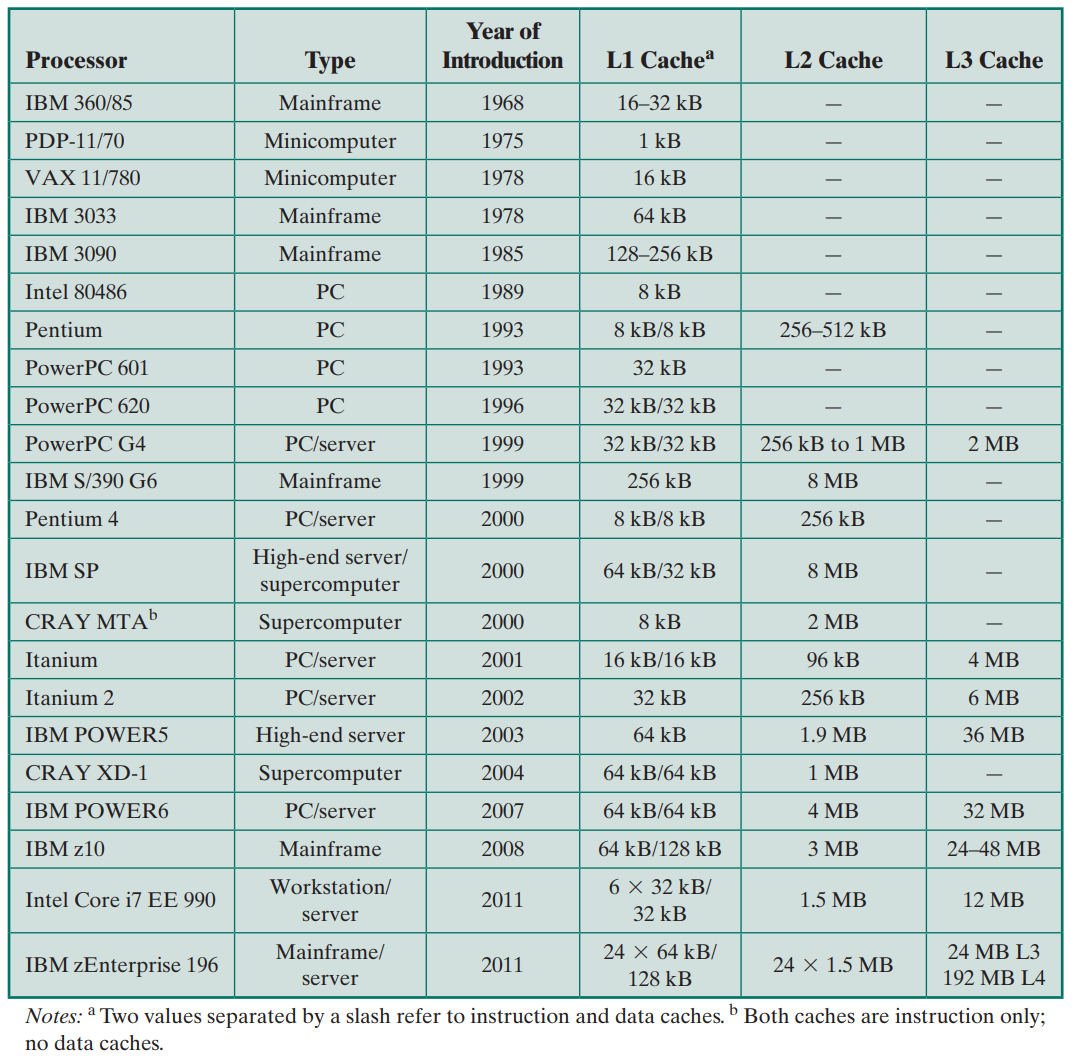
Hampir semua nonembedded processor, dan banyak embedded processor, mendukung memori virtual. Pada dasarnya, memori virtual adalah fasilitas yang memungkinkan program untuk menangani memori dari sudut pandang logis, tanpa berkaitan dengan jumlah memori utama yang tersedia secara fisik. Ketika memori virtual sedang digunakan, bidang alamat instruksi mesin berisi alamat virtual. Untuk perintah “read” dan “write” dari memori utama. Memory Management Unit (MMU) menerjemahkan setiap alamat virtual menjadi alamat fisik di memori utama.

Ketika alamat virtual digunakan, perancang sistem dapat memilih untuk menempatkan cache antara prosesor dan MMU atau antara MMU dan memori utama.

Logical Cache juga dikenal sebagai cache virtual, menyimpan data dan menggunakan alamat virtual. Prosesor mengakses cache secara langsung, tanpa melalui MMU. Cache fisik menyimpan data menggunakan alamat fisik memori utama.

Salah satu keuntungan nyata dari logical cache adalah kecepatan akses cache lebih cepat daripada cache fisik, karena cache dapat merespons sebelum MMU melakukan penerjemahan alamat. Kerugiannya adalah berkaitan dengan fakta bahwa sebagian besar sistem memori virtual memasok setiap aplikasi dengan ruang alamat memori virtual yang sama. Artinya setiap aplikasi melihat memori virtual yang dimulai pada alamat 0. Jadi, alamat virtual yang sama dalam dua aplikasi yang berbeda mengacu pada dua alamat fisik yang berbeda. Oleh karena itu, memori cache harus sepenuhnya dikosongkan dengan masing-masing saklar konteks aplikasi, atau bit tambahan harus ditambahkan ke setiap baris cache untuk mengidentifikasi ruang alamat virtual mana yang dirujuk oleh alamat ini.

**Cache Size**

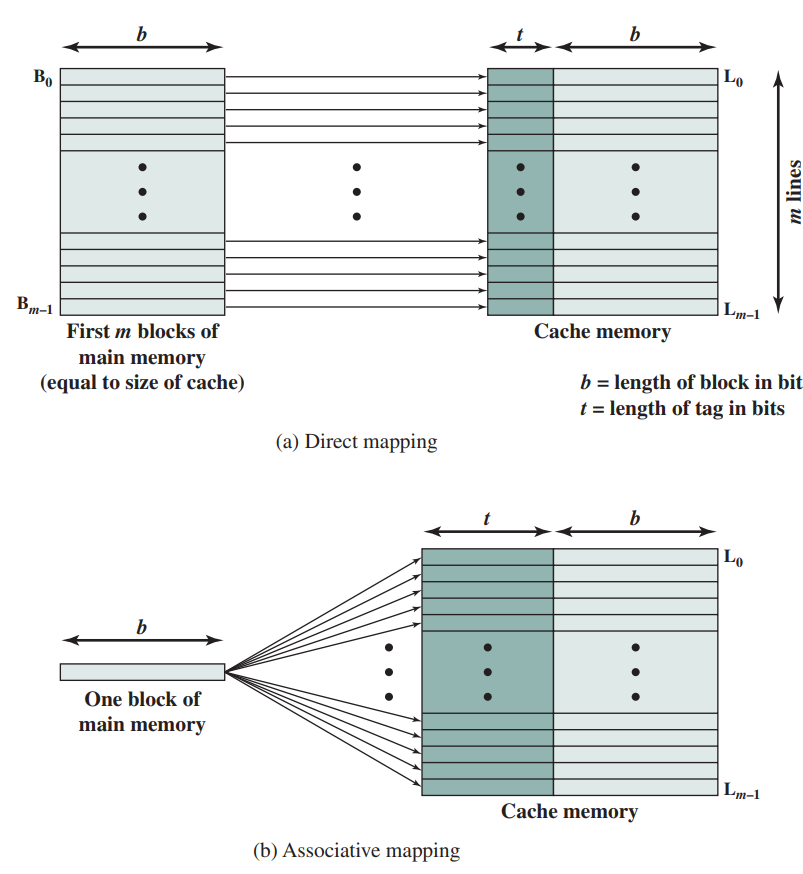


Komputer membutuhkan ukuran cache yang cukup kecil sehingga rata-rata keseluruhan biaya per bit dekat dengan memori utama saja dan cukup besar sehingga secara keseluruhan waktu akses rata-rata dekat dengan cache saja. Selain itu, ada beberapa lainnya motivasi untuk meminimalkan ukuran cache. Semakin besar cache, semakin besar jumlah gerbang yang terlibat dalam menangani cache. Hasilnya adalah cache yang besar cenderung menjadi sedikit lebih lambat daripada yang kecil, bahkan ketika dibangun dengan integrasi yang sama, teknologi sirkuit dan diletakkan di tempat yang sama pada chip dan papan sirkuit. Area chip dan board yang tersedia juga membatasi ukuran cache. Karena kinerja cache sangat sensitif terhadap sifat beban kerja, tidak mungkin untuk mencapai satu ukuran cache "optimal". Tabel 4.3 mencantumkan ukuran cache dari beberapa arus dan prosesor masa lalu.

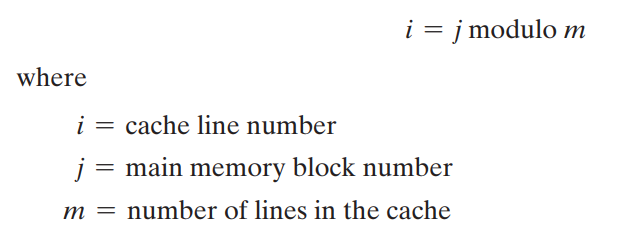
**Mapping Function**

Karena ada lebih sedikit saluran cache daripada blok memori utama, sebuah algoritma diperlukan untuk memetakan main memory ke jalur cache. Pilihan fungsi pemetaan menentukan bagaimana cache diatur. Tiga teknik dapat digunakan yaitu direct, associative, dan set-associative.

1. **Direct mapping**



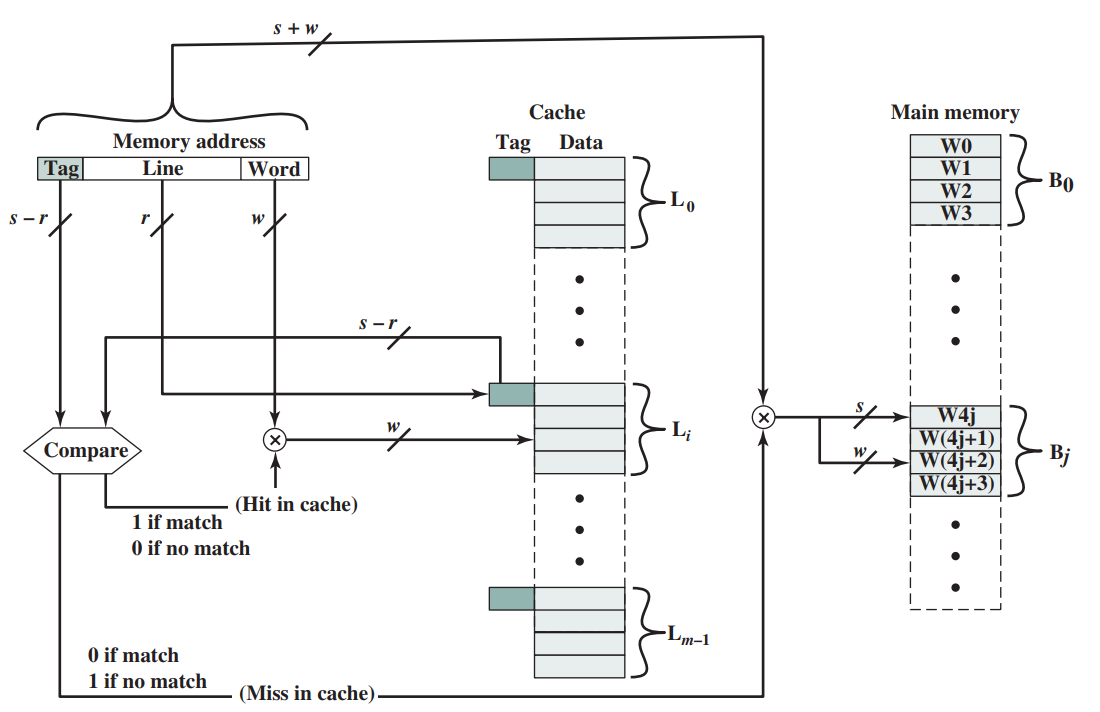
Teknik paling sederhana yang dikenal sebagai pemetaan langsung. Memetakan setiap blok memori utama hanya ke dalam satu jalur cache yang mungkin. Pemetaan dinyatakan sebagai



Gambar diatas menunjukkan pemetaan untuk m blok pertama dari memori utama. Setiap blok peta memori utama ke dalam satu baris cache yang unik. m blok berikutnya dari peta memori utama ke dalam cache dengan cara yang sama; yaitu, blok Bm utama peta memori ke baris L0 cache, blok peta Bm+1 ke baris L1, dan seterusnya.

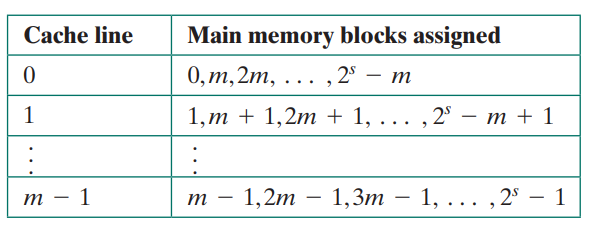
Fungsi pemetaan mudah diimplementasikan menggunakan alamat memori utama. Gambar 4.9 mengilustrasikan mekanisme umum. Untuk tujuan akses cache, masing-masing alamat memori utama dapat dilihat sebagai terdiri dari tiga bidang. W bit paling tidak signifikan mengidentifikasi kata atau byte unik di dalam blok memori utama; di sebagian besar mesin kontemporer, alamatnya berada pada level byte. Bit s yang tersisa menentukan salah satu dari 2s dari blok memori utama. Logika cache menafsirkan bit-bit ini sebagai tag s - r bit (porsi paling signifikan) dan bidang garis r bit. Bidang terakhir ini mengidentifikasi salah satu dari m = 2r baris cache. Untuk meringkas:

* Address length = (s + w) bits
* Number of addressable units = 2^(s+w) words or bytes
* Block size = line size = 2^(w) words or bytes
* Number of blocks in main memory = 2^(s+w) / 2^(w) = 2^(s)
* Number of lines in cache = m = 2^(r)
* Size of cache = 2^(r+w) words or bytes
* Size of tag = (s - r) bits



|  |  |
| --- | --- |
|  |  |

Efek dari pemetaan ini adalah bahwa blok memori utama ditugaskan ke jalur dari cache sebagai berikut:

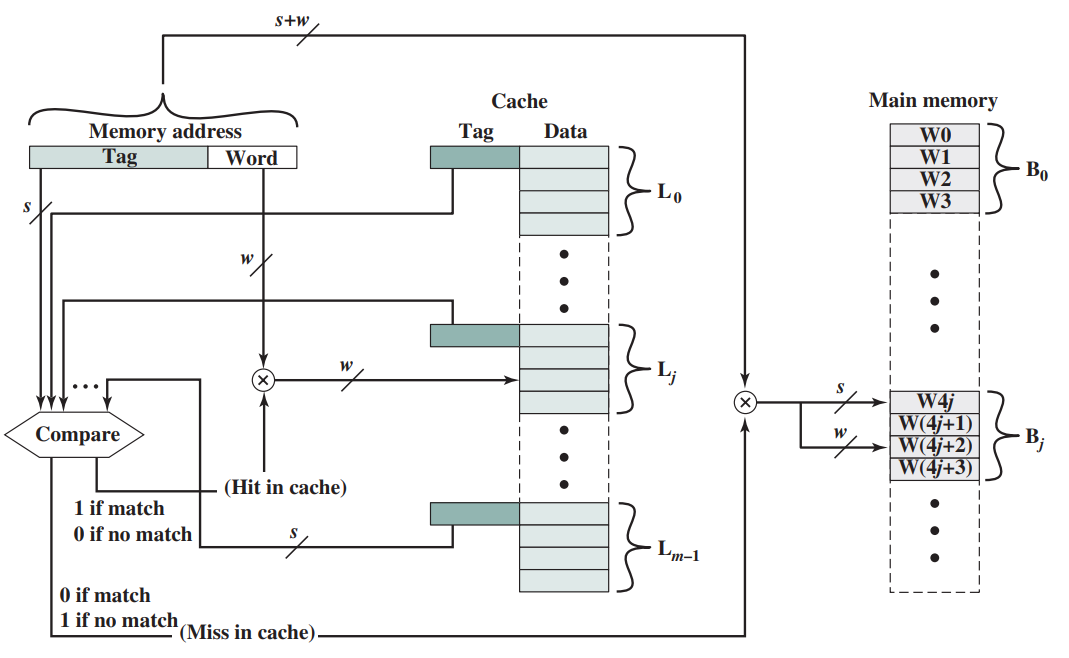


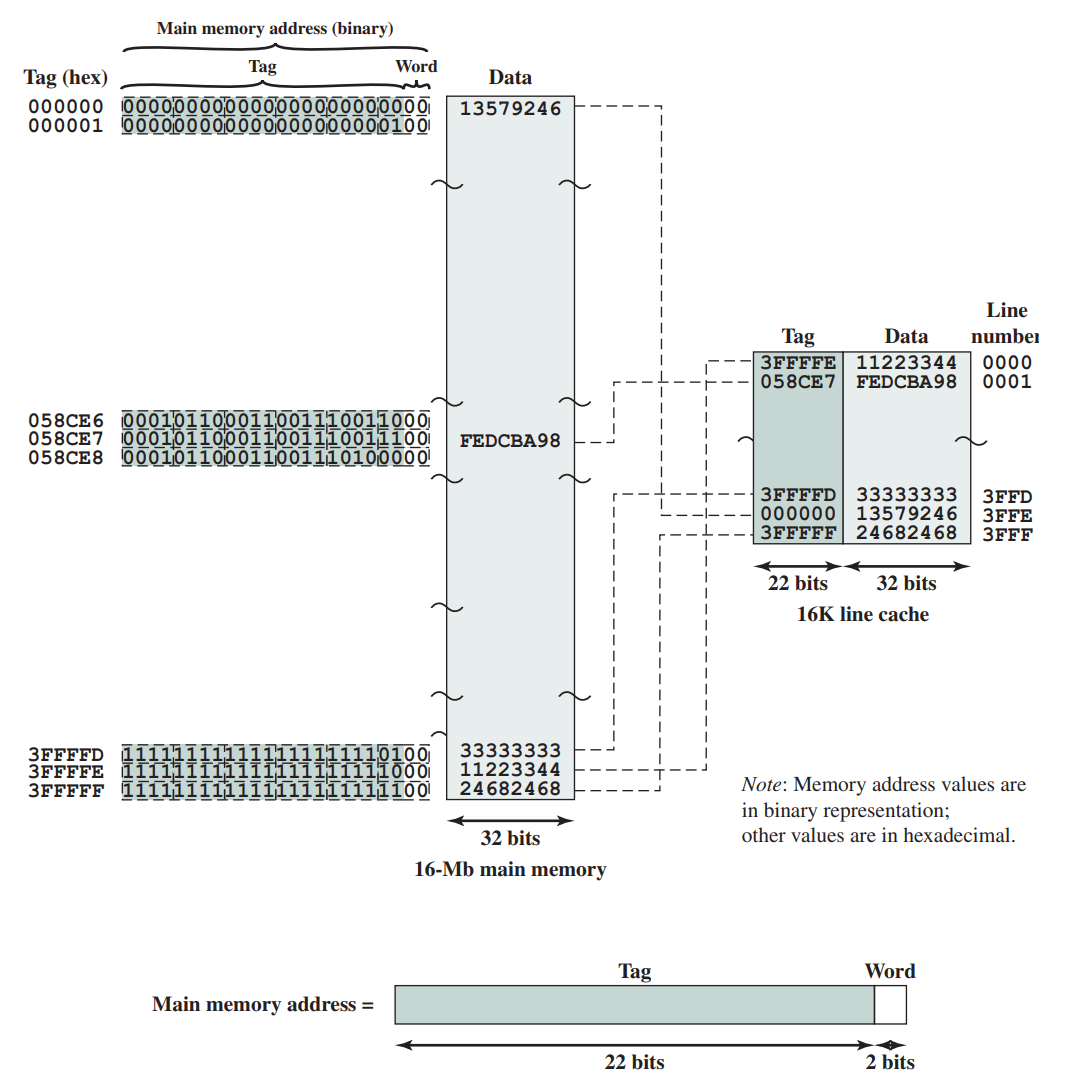
Jadi, penggunaan sebagian alamat sebagai nomor baris menyediakan pemetaan unik dari setiap blok memori utama ke dalam cache. Ketika sebuah blok benar-benar dibaca ke dalam baris yang ditetapkan, perlu untuk menandai data untuk membedakannya dari blok lain yang dapat masuk ke dalam baris itu. Bit s - r lah yang mengatasi tujuan tersebut.

Kerugian utama direct mapping adalah adanya lokasi cache yang tetap untuk setiap blok yang diberikan. Jadi, jika sebuah program kebetulan mereferensikan kata-kata berulang kali dari dua blok berbeda yang dipetakan ke dalam baris yang sama, maka blok-blok tersebut akan terus-menerus ditukar dalam cache dan rasio hit akan rendah (fenomena yang dikenal sebagai thrashing/labrakan).

Salah satu pendekatan untuk menurunkan miss penalty adalah dengan mengingat apa yang dibuang jika diperlukan lagi. Karena data yang dibuang telah diambil, data tersebut dapat digunakan kembali dengan sedikit cost. Daur ulang semacam itu dimungkinkan menggunakan cache “victim”. Cache “victim” awalnya diusulkan sebagai pendekatan untuk mengurangi miss konflik dari cache yang dipetakan langsung tanpa mempengaruhi waktu aksesnya yang cepat. Cache “victim” adalah cache asosiatif penuh yang ukurannya biasanya 4 hingga 16 baris cache, serta berada di antara cache L1 yang dipetakan langsung dan tingkat memori berikutnya.

1. **Associative mapping**

****

****

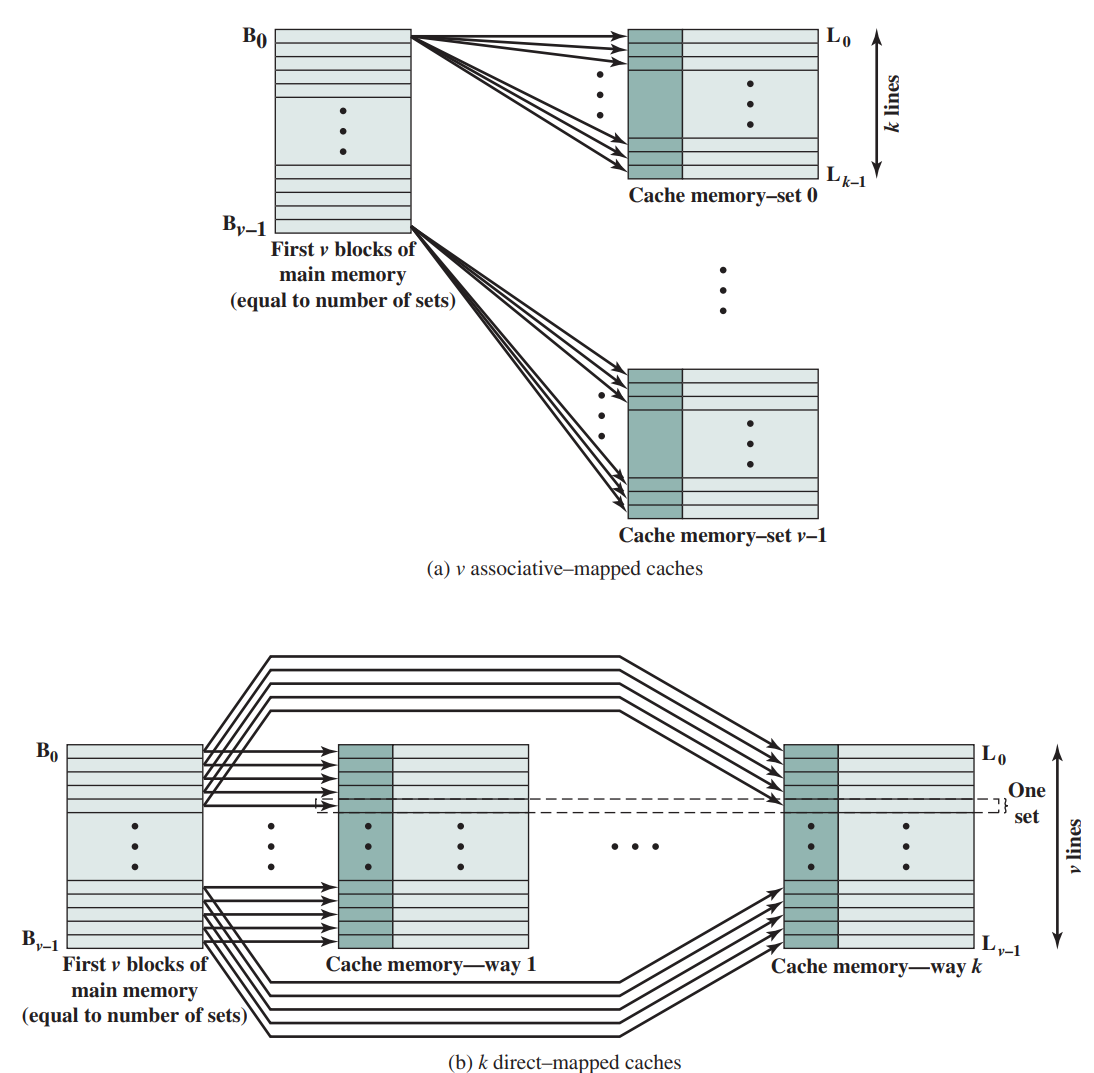
Pemetaan asosiatif mengatasi kelemahan direct pemetaan dengan mengizinkan setiap blok memori utama untuk dimuat ke baris mana pun dari cache. Dalam hal ini, logika kontrol cache menginterpretasikan alamat memori hanya sebagai Tag dan Word field. Tag field secara unik mengidentifikasi blok memori utama. Untuk menentukan apakah sebuah blok ada dalam cache, logika kontrol cache harus secara bersamaan memeriksa tag setiap baris untuk kecocokan. Gambar 4.11 mengilustrasikan logikanya.

Perhatikan bahwa tidak ada bidang di alamat yang sesuai dengan nomor baris, sehingga nomornya baris dalam cache tidak ditentukan oleh format alamat. Untuk meringkas,

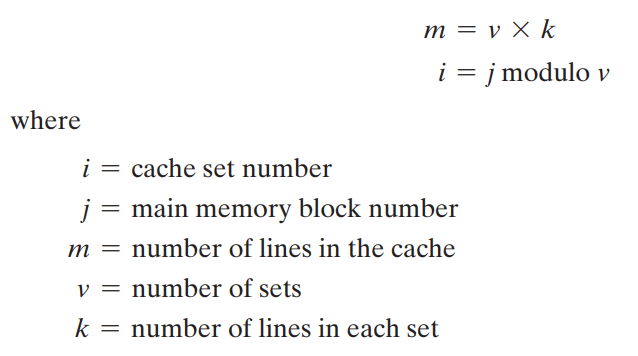
* Address length = (s + w) bits
* Number of addressable units = 2^(s+w) words or bytes
* Block size = line size = 2^(w) words or bytes
* Number of blocks in main memory = 2^(s+w) / 2^(w) = 2^(s)
* Number of lines in cache = undetermined
* Size of tag = s bits

Dengan associative mapping, ada fleksibilitas mengenai blok mana yang harus diganti Ketika blok baru dibaca ke dalam cache. Algoritma penggantian, yang dibahas nanti di bagian ini, dirancang untuk memaksimalkan rasio hit. Kerugian utama dari associative mapping adalah sirkuit kompleks yang diperlukan untuk memeriksa tag dari semua saluran cache secara paralel.

1. **Set associative mapping**



Set- associative mapping adalah pendekatan direct dan associative dengan mengurangi kerugian. Dalam hal ini, cache terdiri dari kumpulan angka yang masing-masing terdiri dari sejumlah baris. Hubungannya:



Disebut juga sebagai k-way set-associative mapping. Dengan set-associative mapping, blok Bj dapat dipetakan ke salah satu garis himpunan j. Gambar 4.13a mengilustrasikan pemetaan ini untuk blok v pertama dari memori utama. Seperti associative mapping, setiap kata dipetakan menjadi beberapa baris cache. Untuk set-associative mapping, setiap kata dipetakan ke semua baris cache dalam set tertentu. Sehingga blok memori utama B0 memetakan ke set 0, dan seterusnya. Dengan demikian, cache set-associative dapat diimplementasikan secara fisik sebagai v cache associative. cache set-associative dapat juga diimplementasikan sebagai k cache direct massage (gambar 4.13b). Setiap cache yang dipetakan langsung disebut sebagai “cara” yang terdiri dari v baris. Baris v pertama dari memori utama dipetakan langsung ke dalam baris v di setiap jalan dan begitu juga untuk v baris berikutnya dan seterusnya. Implementasi yang dipetakan langsung biasanya digunakan untuk derajat asosiatif yang kecil (nilai k yang kecil) sedangkan implementasi yang dipetakan secara asosiatif biasanya digunakan untuk derajat asosiatif yang lebih tinggi.

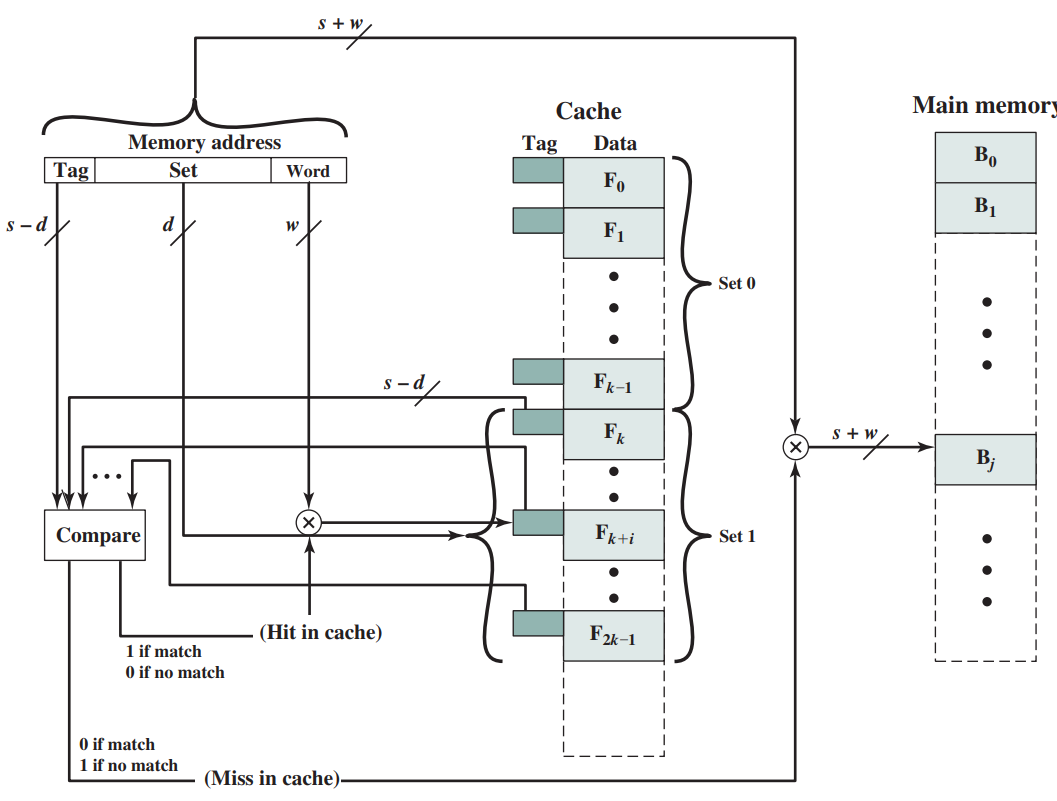
Logika kontrol cache set-associative mapping menginterpretasikan alamat memori sebagai tiga bidang: Tag, Set, dan Word. Bit set d menentukan salah satu set v = 2d. Bit s dari bidang Tag dan Set menentukan salah satu dari 2 blok memori utama. Gambar 4.14 mengilustrasikan logika kontrol cache. Dengan pemetaan asosiatif penuh, tag di alamat memori cukup besar dan harus dibandingkan dengan tag setiap baris dalam cache. Dengan k-way set-associative mapping, tag di alamat memori jauh lebih kecil dan hanya dibandingkan dengan k tag dalam satu set. Ringkasnya,

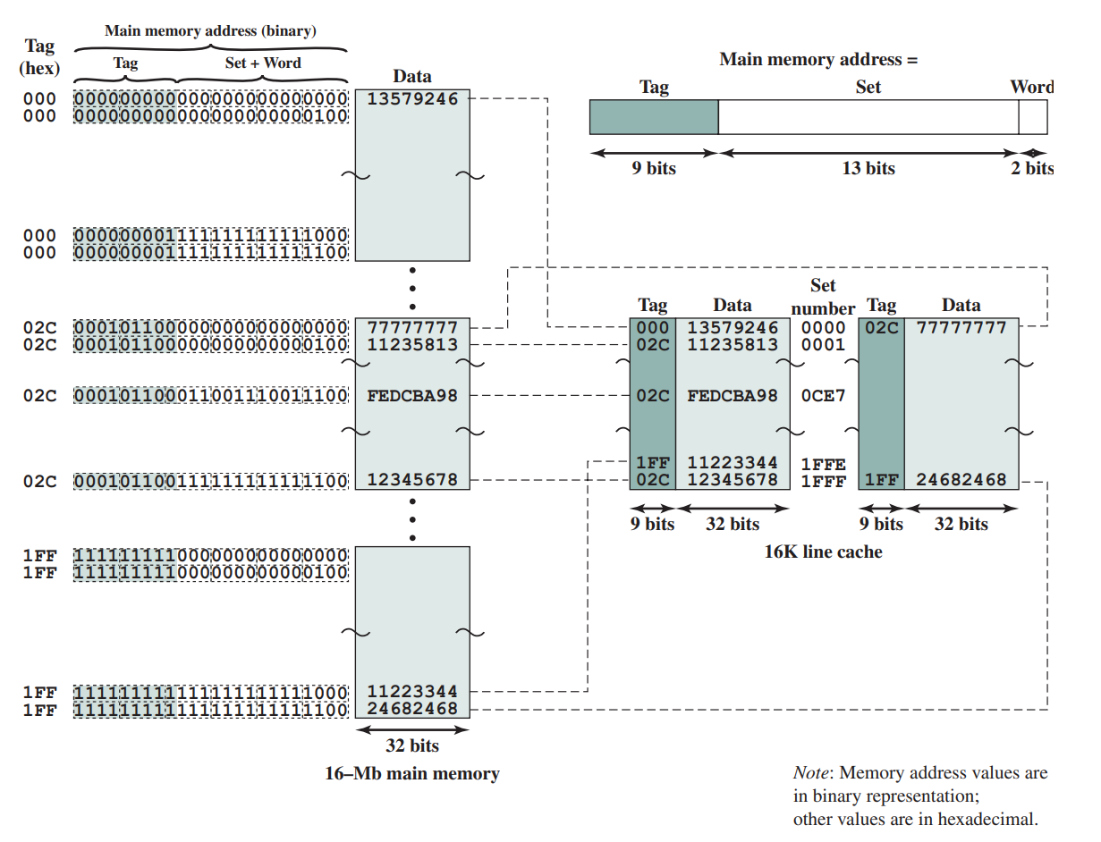
* Address length = (s + w) bits
* Number of addressable units = 2^(s+w) words or bytes
* Block size = line size = 2w words or bytes
* Number of blocks in main memory = 2^(s+w) / 2^(w) = 2^(s)
* Number of lines in set = k
* Number of sets = v = 2^(d)
* Number of lines in cache = m = kv = k \* 2^(d)
* Size of cache = k \* 2^(d+w) words or bytes
* Size of tag = (s - d) bits

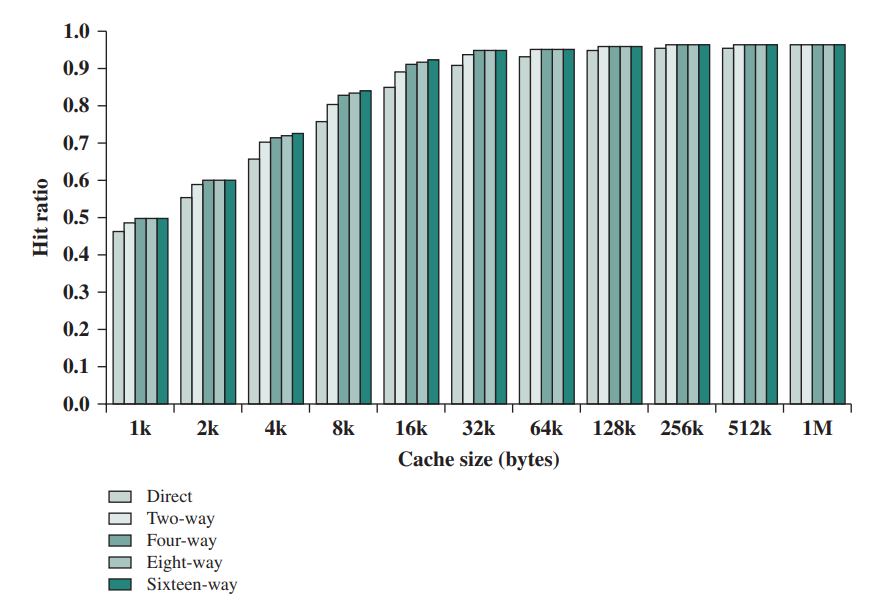
Dalam kasus ekstrim v = m, k = 1, teknik set-associative direduksi menjadi pemetaan langsung. Dan untuk v = 1, k = m, teknik ini direduksi menjadi associative mapping. Penggunaan dua garis perhimpunan (v = m/2, k = 2) adalah organisasi set association yang paling umum. Ini secara signifikan meningkatkan rasio hit dibandingkan pemetaan langsung. set associative empat arah (v = m/4, k = 4) membuat peningkatan tambahan sederhana dengan biaya tambahan yang relatif kecil [MAYB84, HILL89]. Peningkatan lebih lanjut dalam jumlah baris per set memiliki sedikit efek.

Gambar menunjukkan hasil dari satu studi simulasi kinerja cache set-asosiatif sebagai fungsi ukuran cache [GENU04]. Perbedaan kinerja antara asosiatif set langsung dan dua arah signifikan hingga setidaknya ukuran cache 64 kB. Perhatikan juga bahwa perbedaan antara dua arah dan empat arah pada 4 kB jauh lebih kecil daripada perbedaan ukuran cache dari 4 kB hingga 8 kB. Kompleksitas cache meningkat secara proporsional dengan asosiatif, dan dalam hal ini tidak dapat dibenarkan untuk meningkatkan ukuran cache hingga 8 atau bahkan 16 kB. Poin terakhir yang perlu diperhatikan adalah bahwa di luar sekitar 32 kB, peningkatan ukuran cache tidak membawa peningkatan kinerja yang signifikan.

Hasil Gambar 4.16 didasarkan pada simulasi eksekusi compiler GCC. Aplikasi yang berbeda dapat menghasilkan hasil yang berbeda. Misalnya, [CANT01] melaporkan hasil kinerja cache menggunakan banyak benchmark CPU2000 SPEC. Hasil [CANT01] dalam membandingkan rasio hit dengan ukuran cache mengikuti pola yang sama seperti Gambar 4.16, tetapi nilai spesifiknya agak berbeda.







**Replacement Algorithm**

Bila Cache sudah terisi, saat ada blok baru yang dibawa ke cache, salah satu blok pada cache harus diganti dengan cache yang baru. Untuk direct mapping, hanya ada satu cara untuk seluruh blok, dan tidak ada pilihan yang memungkinkan. Untuk associative dan set-associative, algoritma penggantian dibutuhkan. Untuk mencapai kecepatan yang tinggi, algoritma harus diimplementasikan di dalam hardware. Banyak algoritma yang sudah dibuat. Berikut merupakan 4 cara yang paling umum.

Least Recently Used (LRU) mungkin adalah cara yang paling efektif. Caranya adalah mengganti blok dalam set yang sudah berada dalam cache terlama tanpa direferensikan. Untuk two-way set associative, algoritma ini mudah untuk diimplementasikan. Setiap baris memiliki bit USE. saat sebuah baris direferensikan, USE bit baris tersebut di set menjadi 1 dan USE bit baris lain dari set yang sama di set menjadi 0. Saat akan melakukan penggantian, baris dengan USE bit 0 adalah yang dipakai. karena kita berasumsi baris yang paling terkini dipakai lebih mungkin untuk di referensi, LRU seharusnya memberikan hit ratio yang terbaik. LRU juga relatif mudah diimplementasikan untuk fully associative cache.

Mekanisme cachenya memiliki list terpisah dari semua baris di dalam cache. Saat sebuah baris di referensi, list memajukan index baris tersebut ke depan list. Untuk penggantian, baris dengan index terbelakanglah yang dpakai. Karena implementasinya yang simple, LRU adalah algoritma penggantian yang paling populer.

First-in First-Out (FIFO) merupakan salah satu cara lain. Caranya adalah mengganti blok yang sudah ada dalam cache terlama. FIFO mudah diimplementasikan sebagai round-robin atau circular buffer technique.

Least Frequenty Used (LFU) merupakan salah satu cara lain. Blok yang paling sedikit direferensi adalah blok yang akan diganti. LFU dapat diimplementasikan dengan mengasosiasi sebuah counter dalam setiap baris.

Teknik lain yang tidak berbasis penggunaan (seperti LRU, FIFO, LFU, atau varian lainnya) adalah dengan memilih sebuah baris secara acak dari beberapa baris kandidat. Sebuah simulasi telah menunjukan bahwa performa penggantian acak hanya sedikit lebih rendah daripada algoritma yang berbasis penggunaan.

**Write Policy**

Ketika sebuah blok yang ada di cache akan diganti, ada dua kasus yang perlu dipertimbangkan. Jika blok lama dalam cache belum diubah, maka blok tersebut dapat ditimpa dengan blok baru tanpa terlebih dahulu menulis blok lama. Jika setidaknya satu operasi tulis telah dilakukan pada sebuah kata di baris cache itu, maka memori utama harus diperbarui dengan menulis baris cache ke blok memori sebelum memasukkan blok baru. Berbagai kebijakan penulisan, dengan pertukaran kinerja dan ekonomi, dimungkinkan.

Ada dua masalah yang harus dihadapi. Pertama, lebih dari satu perangkat mungkin memiliki akses ke memori utama. Misalnya, modul I/O mungkin dapat membaca-tulis langsung ke memori. Jika sebuah kata telah diubah hanya dalam cache, maka kata memori yang sesuai tidak valid. Selanjutnya, jika perangkat I/O telah mengubah memori utama, maka kata cache tidak valid. Masalah yang lebih kompleks terjadi ketika beberapa prosesor terpasang ke bus yang sama dan setiap prosesor memiliki cache lokalnya sendiri. Kemudian, jika sebuah kata diubah dalam satu cache, itu bisa dibayangkan membatalkan kata di cache lain

Teknik paling sederhana disebut write through. Dengan menggunakan teknik ini, semua operasi tulis dibuat ke memori utama dan juga ke cache, memastikan bahwa memori utama selalu valid. Modul prosesor-cache lainnya dapat memantau lalu lintas ke memori utama untuk menjaga konsistensi dalam cache-nya sendiri. Kerugian utama teknik ini adalah menghasilkan lalu lintas memori yang substansial dan dapat membuat kemacetan.

Teknik alternatif, yang dikenal sebagai write back, meminimalkan penulisan memori. Dengan write back, pembaruan hanya dilakukan di cache. Ketika pembaruan terjadi, bit kotor, atau bit penggunaan, yang terkait dengan saluran diatur. Kemudian, ketika sebuah blok diganti, itu ditulis kembali ke memori utama jika dan hanya jika bit kotor diset. Masalah dengan write back adalah bahwa bagian dari memori utama tidak valid, dan karenanya akses oleh modul I/O hanya dapat diizinkan melalui cache. Hal ini membuat sirkuit yang kompleks dan potensi kemacetan. Pengalaman menunjukkan bahwa persentase referensi memori yang ditulis berada di urutan 15% [SMIT82]. Namun, untuk aplikasi HPC, angka ini bisa mendekati 33% (perkalian vektor-vektor) dan bisa mencapai 50% (transposisi matriks).

**Contoh 4.3**

Pertimbangkan cache dengan ukuran baris 32 byte dan memori utama yang membutuhkan 30 ns untuk mentransfer kata 4-byte. Untuk setiap baris yang ditulis setidaknya sekali sebelum dikeluarkan dari cache, berapa kali rata-rata baris harus ditulis sebelum ditukar dengan cache tulis-balik agar lebih efisien daripada cache write through?

Untuk kasus write-back, setiap baris kotor ditulis kembali satu kali, pada waktu swap-out, mengambil 8 \* 30 = 240 ns. Untuk kasus write-through, setiap pembaruan baris mengharuskan satu kata ditulis ke memori utama, membutuhkan waktu 30 ns. Oleh karena itu, jika rata-rata baris yang ditulis setidaknya satu kali ditulis lebih dari 8 kali sebelum ditukar, maka penulisan kembali lebih efisien.

Dalam organisasi bus di mana lebih dari satu perangkat (biasanya prosesor) memiliki cache dan memori utama digunakan bersama, masalah baru muncul. Jika data dalam satu cache diubah, ini tidak hanya membatalkan kata yang sesuai di memori utama, tetapi juga kata yang sama di cache lain (jika ada cache lain yang kebetulan memiliki kata yang sama). Bahkan jika kebijakan write-through digunakan, cache lain mungkin berisi: data tidak valid. Sistem yang mencegah masalah ini dikatakan memelihara koherensi cache. Kemungkinan pendekatan untuk koherensi cache termasuk yang berikut:

* Bus watching with write through: Setiap pengontrol cache memonitor jalur alamat untuk mendeteksi operasi tulis ke memori oleh master bus lainnya. Jika master lain menulis ke lokasi di memori bersama yang juga berada di memori cache, pengontrol cache akan membatalkan entri cache tersebut. Strategi ini bergantung pada penggunaan kebijakan write-through oleh semua pengontrol cache.
* Hardware transparency: Perangkat keras tambahan digunakan untuk memastikan bahwa semua pembaruan ke memori utama melalui cache tercermin di semua cache. Jadi, jika satu prosesor memodifikasi kata dalam cache-nya, pembaruan ini ditulis ke memori utama. Selain itu, setiap kata yang cocok di cache lain juga diperbarui.
* Non cacheable memory: Hanya sebagian dari memori utama yang digunakan bersama oleh lebih dari satu prosesor, dan ini ditetapkan sebagai non cacheable. Dalam sistem seperti itu, semua akses ke memori bersama adalah cache miss, karena memori bersama tidak pernah disalin ke cache. Memori non cacheable dapat diidentifikasi menggunakan logika chip-select atau bit alamat tinggi.Koherensi cache adalah bidang penelitian yang aktif. Topik ini dieksplorasi lebih lanjut di Bagian Lima.

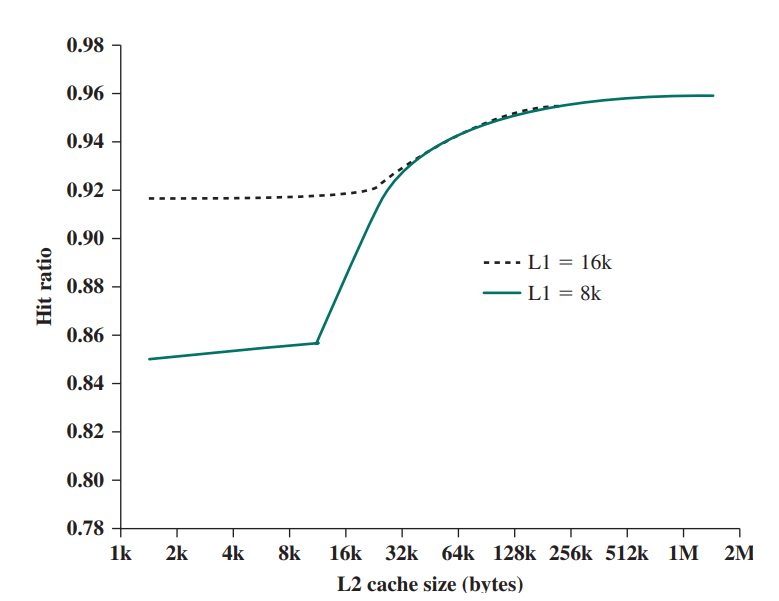
**Line Size**

Elemen desain lainnya adalah ukuran garis. Ketika sebuah blok data diambil dan ditempatkan di cache, tidak hanya kata yang diinginkan tetapi juga sejumlah kata yang berdekatan akan diambil. Ketika ukuran blok meningkat dari ukuran yang sangat kecil ke ukuran yang lebih besar, rasio hit pada awalnya akan meningkat karena prinsip lokalitas, yang menyatakan bahwa data di sekitar kata yang direferensikan kemungkinan akan direferensikan dalam waktu dekat. Saat ukuran blok meningkat, lebih banyak data yang berguna dibawa ke dalam cache. Rasio hit akan mulai menurun, bagaimanapun, karena blok menjadi lebih besar dan kemungkinan menggunakan informasi yang baru diambil menjadi lebih kecil daripada kemungkinan menggunakan kembali informasi yang harus diganti.

* Blok yang lebih besar mengurangi jumlah blok yang masuk ke dalam cache. Karena setiap pengambilan blok menimpa konten cache yang lebih lama, sejumlah kecil blok mengakibatkan data ditimpa segera setelah diambil.
* Sebagai blok menjadi lebih besar, setiap kata tambahan lebih jauh dari kata yang diminta dan karena itu kurang mungkin dibutuhkan dalam waktu dekat.

Hubungan antara ukuran blok dan rasio hit adalah kompleks, tergantung pada karakteristik lokalitas dari program tertentu, dan tidak ada nilai optimal yang pasti telah ditemukan. Ukuran dari 8 hingga 64 byte tampaknya cukup mendekati optimal [SMIT87, PRZY88, PRZY90, HAND98]. Untuk sistem HPC, ukuran baris cache 64- dan 128-byte paling sering digunakan.

**Number of Cache**



Ketika cache awalnya diperkenalkan, sistem tipikal memiliki satu cache. Baru-baru ini, penggunaan beberapa cache telah menjadi norma. Dua aspek dari masalah desain ini berkaitan dengan jumlah level cache dan penggunaan cache terpadu versus terpisah.

Multi Level cache Karena kepadatan logika telah meningkat, menjadi mungkin untuk memiliki cache pada chip yang sama dengan prosesor: cache on-chip. Dibandingkan dengan cache yang dapat dijangkau melalui bus eksternal, cache on-chip mengurangi aktivitas bus eksternal prosesor dan oleh karena itu mempercepat waktu eksekusi dan meningkatkan kinerja sistem secara keseluruhan. Ketika instruksi atau data yang diminta ditemukan dalam cache onchip, akses bus dihilangkan. Karena jalur data internal yang pendek ke prosesor, dibandingkan dengan panjang bus, akses cache on-chip akan selesai jauh lebih cepat daripada siklus bus keadaan nol-tunggu. Selanjutnya, selama periode ini bus gratis untuk mendukung transfer lainnya.

Dimasukkannya cache on-chip membuka pertanyaan apakah cache off-chip, atau eksternal, masih diinginkan. Biasanya, jawabannya adalah ya, dan sebagian besar desain kontemporer menyertakan cache on-chip dan eksternal. Yang paling sederhana seperti organisasi dikenal sebagai cache dua tingkat, dengan level internal 1 (L1) dan cache eksternal ditetapkan sebagai level 2 (L2). Alasan untuk menyertakan cache L2 adalah sebagai berikut: Jika tidak ada cache L2 dan prosesor membuat permintaan akses untuk lokasi memori yang tidak ada di cache L1, maka prosesor harus mengakses DRAM atau memori ROM di seluruh bus. Karena kecepatan bus yang biasanya lambat dan waktu akses memori yang lambat, ini menghasilkan kinerja yang buruk.

Di sisi lain, jika cache L2 SRAM (RAM statis) digunakan, maka seringkali informasi yang hilang dapat diambil dengan cepat. Jika SRAM cukup cepat untuk menyamai kecepatan bus, maka data dapat diakses menggunakan transaksi keadaan nol-tunggu, jenis transfer bus tercepat.

Dua fitur desain cache kontemporer untuk cache bertingkat patut diperhatikan. Pertama, untuk cache L2 off-chip, banyak desain tidak menggunakan bus sistem sebagai jalur untuk transfer antara cache L2 dan prosesor, tetapi menggunakan jalur data terpisah, untuk mengurangi beban pada bus sistem. Kedua, dengan terus menyusutnya komponen prosesor, sejumlah prosesor kini menggabungkan cache L2 pada chip prosesor, sehingga meningkatkan kinerja.

Potensi penghematan karena penggunaan cache L2 tergantung pada hit rate di cache L1 dan L2. Beberapa penelitian telah menunjukkan bahwa, secara umum, penggunaan cache tingkat kedua meningkatkan kinerja (misalnya, lihat [AZIM92], [NOVI93], [HAND98]). Namun, penggunaan cache bertingkat memang memperumit semua masalah desain yang terkait dengan cache, termasuk ukuran, algoritma penggantian, dan kebijakan penulisan; lihat [HAND98] dan [PEIR99] untuk diskusi.

Gambar 4.17 menunjukkan hasil dari satu studi simulasi kinerja cache dua tingkat sebagai fungsi ukuran cache [GENU04]. Gambar tersebut mengasumsikan bahwa kedua cache memiliki ukuran baris yang sama dan menunjukkan rasio hit total. Artinya, hit dihitung jika data yang diinginkan muncul di cache L1 atau L2. Gambar tersebut menunjukkan dampak L2 pada total hit sehubungan dengan ukuran L1. L2 memiliki pengaruh yang kecil pada jumlah total cache hits sampai setidaknya dua kali lipat ukuran cache L1. Perhatikan bahwa bagian paling curam dari kemiringan untuk cache L1 sebesar 8 kB adalah untuk cache L2 sebesar 16 kB. Sekali lagi untuk cache L1 16 kB, bagian paling curam dari kurva adalah untuk ukuran cache L2 32 kB. Sebelum titik itu, cache L2 memiliki sedikit, jika ada, berdampak pada kinerja cache total. Kebutuhan cache L2 lebih besar dari cache L1 untuk mempengaruhi kinerja masuk akal. Jika cache L2 memiliki ukuran dan kapasitas baris yang sama dengan cache L1, isinya kurang lebih akan mencerminkan isi cache L1.

Dengan meningkatnya ketersediaan area on-chip yang tersedia untuk cache, sebagian besar mikroprosesor kontemporer telah memindahkan cache L2 ke chip prosesor dan menambahkan cache L3. Awalnya, cache L3 dapat diakses melalui bus eksternal. Baru-baru ini, sebagian besar mikroprosesor telah memasukkan cache L3 on-chip. Dalam kedua kasus, tampaknya ada keuntungan kinerja untuk menambahkan tingkat ketiga (misalnya, lihat [GHAI98]). Lebih lanjut, sistem besar, seperti sistem zEnterprise mainframe IBM, sekarang menggabungkan 3 level cache on-chip dan level keempat cache yang dibagi di beberapa chip [CURR11].

Unified versus split caches Ketika cache on-chip pertama kali muncul, banyak desain terdiri dari cache tunggal yang digunakan untuk menyimpan referensi ke data dan instruksi. Baru-baru ini, telah menjadi umum untuk membagi cache menjadi dua: satu didedikasikan untuk instruksi dan satu didedikasikan untuk data. Kedua cache ini ada pada level yang sama, biasanya sebagai dua cache L1. Ketika prosesor mencoba untuk mengambil sebuah instruksi dari memori utama, pertama-tama ia berkonsultasi dengan cache instruksi L1, dan ketika prosesor mencoba untuk mengambil data dari memori utama, pertama-tama ia berkonsultasi dengan cache data L1.

Ada dua keuntungan potensial dari cache terpadu, yaitu:

* Untuk ukuran cache tertentu, cache terpadu memiliki hit rate yang lebih tinggi daripada cache split karena menyeimbangkan beban antara instruksi dan pengambilan data secara otomatis. Artinya, jika pola eksekusi melibatkan lebih banyak pengambilan instruksi daripada pengambilan data, maka cache akan cenderung terisi dengan instruksi, dan jika eksekusi pola melibatkan pengambilan data yang relatif lebih banyak, sebaliknya akan terjadi
* Hanya satu cache yang perlu dirancang dan diimplementasikan.

Kecenderungannya adalah menuju split cache pada L1 dan unified caches untuk level yang lebih tinggi, khususnya untuk mesin superscalar, yang menekankan eksekusi instruksi paralel dan prefetching dari prediksi instruksi yang akan datang. Keuntungan utama dari desain cache split adalah menghilangkan pertentangan untuk cache antara instruksi unit fetch/decode dan unit eksekusi. Ini penting dalam desain apa pun yang bergantung pada pipelining instruksi. Biasanya, prosesor akan mengambil instruksi sebelumnya dan mengisi buffer, atau pipa, dengan instruksi yang akan dieksekusi.

**PUSTAKA**

Stalings, W. (2016). Computer Organization and Architecture Designing For Performa 10th Edition. PEARSON Education.