4

MEMÓRIA CACHE

OBJETIVOS DE APRENDIZAGEM

Após ler este capítulo, você será capaz de:

- Apresentar uma visão geral das principais características dos sistemas de memória do computador e do uso da hierarquia da memória.
- Descrever os conceitos básicos e o objetivo da memória cache.
- Discutir os elementos-chave do projeto da cache.
- Fazer distinção entre mapeamento direto, mapeamento associativo e mapeamento associativo por conjunto.
- Explicar as razões para usar diversos níveis de cache.
- Compreender as implicações do desempenho dos diversos níveis de memória.

Embora aparentemente simples em conceito, a memória do computador apresenta talvez a mais variada gama de tipos, tecnologia, organização, desempenho e custo comparado com qualquer outro recurso de um sistema de computação. Nenhuma tecnologia em si é ideal para os requisitos de memória de um computador. Como consequência, o sistema de computação típico é equipado com uma hierarquia de subsistemas de memória, algumas internas ao sistema (acessíveis diretamente pelo processador) e algumas externas (acessíveis pelo processador por meio de um módulo de E/S).

Este capítulo e o seguinte destacam os elementos da memória interna, enquanto o Capítulo 6 é dedicado à memória externa. Para começar, a primeira seção examina as principais características das memórias de computador. O restante do capítulo examina um elemento essencial de todos os sistemas de computação modernos: a memória cache.

4.1 VISÃO GERAL DO SISTEMA DE MEMÓRIA DO COMPUTADOR

Características dos sistemas de memória

O complexo assunto da memória de computador pode ser mais bem compreendido se classificarmos os sistemas de memória de acordo com suas principais características. As mais importantes estão listadas na Tabela 4.1.

O termo **localização** na Tabela 4.1 indica se a memória é interna ou externa ao computador. A memória interna em geral significa a memória principal, mas existem outras formas de memória interna. O processador necessita de uma memória local própria, na forma de registradores (por exemplo, veja a Figura 2.3). Além do mais, como será visto, a parte da unidade de controle do processador também pode exigir sua própria memória interna. Vamos deixar a discussão desses dois últimos tipos de memória interna para capítulos posteriores. A cache é outra forma de memória interna. A memória externa consiste em dispositivos de armazenamento periféricos, como discos e fitas, que são acessíveis ao processador por meio de controladores de E/S.

Uma característica óbvia da memória é a sua **capacidade**. Para a memória interna, isso costuma ser expresso em termos de bytes (1 byte = 8 bits) ou palavras. Os tamanhos comuns de palavra são 8, 16 e 32 bits. A capacidade da memória externa normalmente é expressa em termos de bytes.

Tabela 4.1

Principais características dos sistemas de memória do computador.

Localização	Desempenho
Interna (por exemplo, registradores do processador, memória principal, cache)	Tempo de acesso
Externa (por exemplo, discos ópticos, discos magnéticos, fitas)	Tempo de ciclo
	Taxa de transferência
Método de acesso	**Tipo físico**
Sequencial	Semicondutor
Direto	Magnético
Aleatório	Óptico
Associativo	Magneto-óptico
Unidade de transferência	**Características físicas**
Palavra	Volátil/não volátil
Bloco	Apagável/não apagável
Capacidade	**Organização**
Número de palavras	Módulos de memória
Número de bytes	

Um conceito relacionado é a **unidade de transferência**. Para a memória interna, a unidade de transferência é igual ao número de linhas elétricas que chegam e que saem do módulo de memória. Isso pode ser igual ao tamanho da palavra, mas em geral é maior, como 64, 128 ou 256 bytes. Para esclarecer esse ponto, considere três conceitos relacionados à memória interna:

- **Palavra**: a unidade "natural" de organização da memória. O tamanho da palavra costuma ser igual ao número de bits usados para representar um inteiro e ao tamanho da instrução. Infelizmente, existem muitas exceções. Por exemplo, o CRAY C90 (um modelo de supercomputador CRAY mais antigo) tem um tamanho de palavra de 64 bits, mas usa a representação de inteiros com 46 bits. A arquitetura Intel x86 tem uma grande variedade de tamanhos de instrução, expressos como múltiplos de bytes e uma palavra com tamanho de 32 bits.
- **Unidades endereçáveis**: em alguns sistemas, a unidade endereçável é a palavra. Porém, muitos sistemas permitem o endereçamento no nível de byte. De qualquer forma, o relacionamento entre o tamanho em bits A de um endereço e o número N de unidades endereçáveis é $2^A = N$.
- **Unidade de transferência**: para a memória principal, este é o número de bits lidos ou escritos na memória de uma só vez. A unidade de transferência não precisa ser igual a uma palavra ou uma unidade endereçável. Para a memória externa, os dados em geral são transferidos em unidades muito maiores que uma palavra, e estas são chamadas de blocos.

Outra distinção entre os tipos de memória é o **método de acesso** das unidades de dados. Ele inclui:

- **Acesso sequencial**: a memória é organizada em unidades de dados chamadas registros. O acesso é feito em uma sequência linear específica. Uma informação de endereçamento armazenada é usada para separar registros e auxiliar no processo de recuperação. Usa-se um mecanismo compartilhado de leitura-escrita, o qual precisa ser movido de seu local atual para o local desejado, passando e rejeitando cada registro intermediário. Assim, o tempo para acessar um registro qualquer é altamente variável. As unidades de fita, discutidas no Capítulo 6, são de acesso sequencial.
- **Acesso direto**: assim como o acesso sequencial, o acesso direto envolve um mecanismo compartilhado de leitura-escrita. Porém, os blocos ou registros individuais têm um endereço exclusivo, baseado no local físico. O acesso é realizado pelo acesso direto, para alcançar uma vizinhança geral, mais uma busca sequencial, contagem ou espera, até chegar ao local final. Novamente, o tempo de acesso é variável. As unidades de disco, discutidas no Capítulo 6, são de acesso direto.
- **Acesso aleatório**: cada local endereçável na memória tem um mecanismo de endereçamento exclusivo, fisicamente interligado. O tempo para acessar determinado local é independente da sequência de acessos anteriores e é constante. Assim, qualquer local pode ser selecionado aleatoriamente, bem como endereçado e acessado diretamente. A memória principal e alguns sistemas de cache são de acesso aleatório.
- **Associativo**: este é o tipo de memória de acesso aleatório que permite fazer uma comparação de um certo número de bits, dentro de uma palavra, com uma combinação específica, fazendo isso com todas as palavras simultaneamente. Assim, uma palavra é recuperada com base em uma parte de seu conteúdo, em vez de seu endereço. Assim como a memória de acesso aleatório comum, cada local tem seu próprio mecanismo de endereçamento, e o tempo de recuperação é constante, independentemente do local ou padrões de acesso anteriores. As memórias cache podem empregar o acesso associativo.

Do ponto de vista do usuário, as duas características mais importantes da memória são capacidade e **desempenho**. Três parâmetros de desempenho são usados:

- **Tempo de acesso (latência)**: para a memória de acesso aleatório, esse é o tempo gasto para realizar uma operação de leitura ou escrita, ou seja, o tempo desde o instante em que um endereço é apresentado à memória até o instante em que os dados foram armazenados ou se tornaram disponíveis para uso. Para a memória de acesso não aleatório, o tempo de acesso é o tempo gasto para posicionar o mecanismo de leitura-escrita no local desejado.
- **Tempo de ciclo de memória**: esse conceito é aplicado principalmente à memória de acesso aleatório, e consiste no tempo de acesso mais qualquer tempo adicional antes que um segundo acesso possa ter início. Esse tempo adicional pode ser exigido para a extinção de transientes nas linhas de sinal ou para a regeneração de dados, se eles forem lidos destrutivamente. Observe que o tempo de ciclo de memória se refere ao barramento do sistema, e não do processador.
- **Taxa de transferência**: é a taxa em que os dados podem ser transferidos para dentro ou fora de uma unidade de memória. Para a memória de acesso aleatório, ela é igual a 1/(tempo de ciclo). Para a memória de acesso não aleatório, existe a seguinte relação:

$$T_n = T_A + \frac{n}{R} \quad \textbf{(4.1)}$$

em que

T_n = tempo médio para ler ou escrever n bits
T_A = tempo de acesso médio
n = número de bits
R = taxa de transferência em bits por segundo (bps)

Uma variedade de **tipos físicos** da memória tem sido empregada. As mais comuns hoje em dia são memória semicondutora, memória de superfície magnética, usada para disco e fita, e óptica e magneto-óptica.

Várias **características físicas** de armazenamento de dados são importantes. Em uma memória volátil, a informação se deteriora naturalmente ou se perde quando a energia elétrica é desligada. Em uma memória não volátil, a informação, uma vez gravada, permanece sem deterioração até que seja deliberadamente mudada; nenhuma energia elétrica é necessária para reter a informação. As memórias com superfície magnética são não voláteis. A memória semicondutora (memória em circuitos integrados) pode ser volátil ou não. A memória não apagável não pode ser alterada, exceto destruindo-se a unidade de armazenamento. A memória semicondutora desse tipo é conhecida como *memória somente de leitura* (ROM — do inglês, *Read-Only Memory*). Inevitavelmente, na prática uma memória não apagável também precisa ser não volátil.

Para a memória de acesso aleatório, a **organização** é um aspecto-chave do projeto. Nesse contexto, *organização* refere-se à disposição física de bits para formar palavras. A disposição óbvia nem sempre é usada, conforme explicado no Capítulo 5.

A hierarquia de memória

As restrições de projeto sobre a memória de um computador podem ser resumidas por três questões: Quanto? Com que velocidade? A que custo?

A questão da quantidade é, de certa forma, livre. Se houver capacidade, as aplicações provavelmente serão desenvolvidas para utilizá-la. A questão da velocidade, de certa forma, é mais fácil de responder. Para conseguir maior desempenho, a memória deve ser capaz de acompanhar a velocidade do processador. Ou seja, enquanto o processador está executando instruções, não gostaríamos que ele tivesse que parar, aguardando por instruções ou operandos. A questão final também precisa ser considerada. Para um sistema prático, o custo da memória deve ser razoável em relação a outros componentes.

Como se pode esperar, existe uma relação entre as três principais características da memória, a saber: capacidade, tempo de acesso e custo. Diversas tecnologias são usadas para implementar sistemas de memória e, por meio desse espectro de tecnologias, existem as seguintes relações:

- Tempo de acesso mais rápido, maior custo por bit.
- Maior capacidade, menor custo por bit.
- Maior capacidade, tempo de acesso mais lento.

O dilema que o projetista enfrenta é claro. O projetista gostaria de usar tecnologias de memória que oferecessem grande capacidade de memória, tanto porque a capacidade é necessária quanto porque o custo por bit é baixo. Porém, para atender aos requisitos de desempenho, ele precisa usar memórias caras, relativamente com menor capacidade e com menores tempos de acesso.

Para sair desse dilema, é preciso não contar com um único componente ou tecnologia de memória, mas empregar uma **hierarquia de memória**. Uma hierarquia típica é ilustrada na Figura 4.1. Conforme se desce na hierarquia, ocorre o seguinte:

a. Diminuição do custo por bit.
b. Aumento da capacidade.
c. Aumento do tempo de acesso.
d. Diminuição da frequência de acesso à memória pelo processador.

Assim, memórias menores, mais caras e mais rápidas são complementadas por memórias maiores, mais baratas e mais lentas. A chave para o sucesso dessa organização é o item (d): diminuição na frequência de acesso. Veremos esse conceito com mais detalhes quando discutirmos sobre a memória cache, mais adiante neste capítulo, e a memória virtual, no Capítulo 8. Neste ponto, oferecemos uma rápida explicação.

Figura 4.1
A hierarquia de memória.

O uso de dois níveis de memória para reduzir o tempo médio de acesso funciona em princípio, mas somente se as condições (a) a (d) se aplicarem. Empregando diferentes tecnologias, existe um espectro de sistemas de memória que satisfaz às condições (a) a (c). Felizmente, a condição (d) também costuma ser válida.

A base para a validade da condição (d) é um princípio conhecido como **localidade de referência** (DENNING, 1968). Durante a execução de um programa, as referências de memória pelo processador, para instruções e para dados, tendem a se agrupar. Os programas em geral contêm uma série de loops iterativos e sub-rotinas. Quando um loop ou sub-rotina inicia sua execução, existem referências repetidas a um pequeno conjunto de instruções. De modo semelhante, operações sobre tabelas e arrays envolvem o acesso a um conjunto de palavras de dados agrupadas. Após um longo período os conjuntos mudam, mas para um pequeno período de tempo o processador trabalha com conjuntos fixos de referências à memória.

EXEMPLO 4.1

Suponha que o processador tenha acesso a dois níveis de memória. O nível 1 contém 1.000 palavras e tem tempo de acesso de 0,01 μs; o nível 2 contém 100.000 palavras e tem tempo de acesso de 0,1 μs. Suponha que, se uma palavra a ser acessada estiver no nível 1, então o processador a acessa diretamente. Se estiver no nível 2, então a palavra primeiro é transferida para o nível 1 e depois é acessada pelo processador. Para simplificar, ignoramos o tempo necessário para o processador determinar se a palavra está no nível 1 ou no nível 2. A Figura 4.2 mostra o formato geral da curva que representa essa situação. A figura mostra o tempo médio de acesso para uma memória de dois níveis como uma função da razão de acerto H, em que H é definido como a fração de todos os acessos à memória que são encontrados na memória mais rápida (por exemplo, a cache), T_1 é o tempo de acesso ao nível 1, e T_2 é o tempo de acesso ao nível 2[1]. Como podemos ver, para altas porcentagens de acesso ao nível 1, o tempo médio de acesso total é muito mais próximo daquele do nível 1 do que do nível 2.

1 Se a palavra acessada for encontrada na memória mais rápida, tem-se um **acerto** (*hit*). Uma **falha** (*miss*) ocorre se a palavra acessada não for encontrada na memória mais rápida.

Em nosso exemplo, suponha que 95% dos acessos à memória sejam encontrados no nível 1. Então, o tempo médio para acessar uma palavra pode ser expresso como

$$(0{,}95)(0{,}01\ \mu s) + (0{,}05)(0{,}01\ \mu s + 0{,}1\ \mu s) = 0{,}0095 + 0{,}0055 = 0{,}015\ \mu s$$

O tempo de acesso médio é muito mais próximo de 0,01 μs do que de 0,1 μs, como desejado.

Desta forma, é possível organizar dados pela hierarquia em que a porcentagem de acessos em cada nível imediatamente inferior é muito menor que para o nível acima. Considere o exemplo de dois níveis, já apresentado. Considere que a memória de nível 2 contenha todas as instruções e dados do programa. Os conjuntos atuais podem ser temporariamente colocados no nível 1. De vez em quando, um dos conjuntos no nível 1 terá de ser passado para o nível 2, para dar espaço para um novo conjunto chegando ao nível 1. Porém, na média, a maioria das referências será para instruções e dados contidos no nível 1.

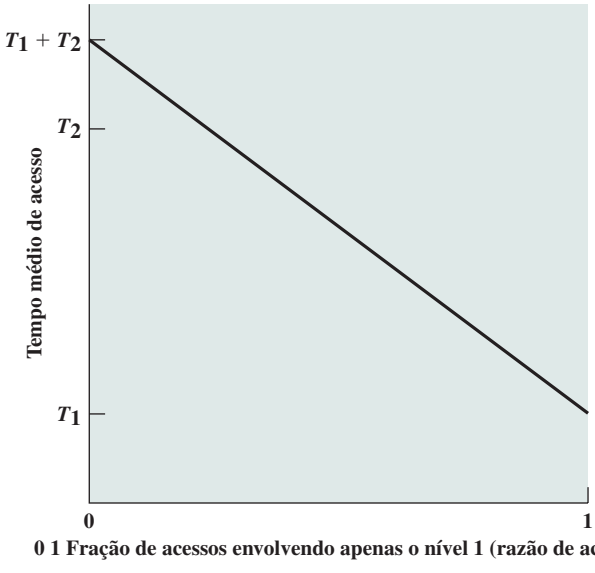
Esse princípio pode ser aplicado para mais de dois níveis de memória, conforme sugerido pela hierarquia mostrada na Figura 4.1. O tipo de memória mais rápido, menor e mais caro consiste nos registradores internos ao processador. Em geral, um processador terá poucas dúzias desses registradores, embora algumas máquinas contenham centenas de registradores. A memória principal é o principal sistema de memória interna do computador. Cada localização na memória principal tem um endereço exclusivo. A memória principal costuma ser estendida com uma memória cache menor, de maior velocidade. A cache normalmente não é visível ao programador ou, na verdade, ao processador. Esse é um dispositivo para organizar a movimentação de dados entre a memória principal e os registradores do processador, para melhorar o desempenho.

As três formas de memória que descrevemos em geral são voláteis e empregam a tecnologia semicondutora. O uso de três níveis explora o fato de que existem diversos tipos de memória semicondutora, que diferem em velocidade e custo. Os dados são armazenados de forma mais permanente em dispositivos externos, de armazenamento em massa, sendo os mais comuns o disco rígido e a mídia removível, como disco magnético removível, fita e armazenamento óptico. A memória externa, não volátil, também é chamada de **memória secundária** ou **memória auxiliar**. Estas são usadas para armazenar arquivos de programa e dados e, normalmente, são visíveis ao programador apenas em termos de arquivos e registros, ao contrário de bytes ou palavras individuais. O disco também é usado para oferecer uma extensão à memória principal, conhecida como memória virtual, que será discutida no Capítulo 8.

Outras formas de memória podem ser incluídas na hierarquia. Por exemplo, grandes mainframes IBM incluem uma forma de memória interna conhecida como armazenamento expandido. Este usa uma tecnologia semicondutora que é mais lenta e menos dispendiosa do que a da memória principal. Estritamente falando, essa

Figura 4.2

Desempenho dos acessos envolvendo apenas o nível 1 (razão de acerto).



memória não se encaixa na hierarquia, mas é um apêndice: os dados podem ser movidos entre a memória principal e o armazenamento expandido, mas não entre o armazenamento expandido e a memória externa. Outras formas de memória secundária incluem os discos ópticos e magneto-ópticos. Finalmente, outros níveis podem ser efetivamente introduzidos à hierarquia por meio do uso de software. Uma parte da memória principal pode ser usada como um buffer para manter temporariamente os dados que devem ser levados ao disco. Essa técnica, às vezes chamada de cache de disco,[2] melhora o desempenho de duas maneiras:

- As gravações em disco são agrupadas. Em vez de muitas transferências de dados pequenas, temos algumas transferências de dados grandes. Isso melhora o desempenho do disco e minimiza o envolvimento do processador.
- Alguns dados destinados para escrita podem ser referenciados por um programa antes da próxima cópia no disco. Nesse caso, os dados são recuperados rapidamente da cache de disco, ao invés de lentamente do disco.

O Apêndice 4A, no final deste capítulo, examina as implicações de desempenho das estruturas de memória multinível.

4.2 PRINCÍPIOS DA MEMÓRIA CACHE

A memória cache é desenvolvida para combinar o tempo de acesso de memórias de alto custo e alta velocidade com as memórias de menor velocidade, maior tamanho e mais baixo custo. O conceito é ilustrado na Figura 4.3a. Existe uma memória principal relativamente grande e lenta junto com a memória cache, menor e mais rápida. A cache contém uma cópia de partes da memória principal. Quando o processador tenta ler uma palavra da memória, é feita uma verificação para determinar se a palavra está na cache. Se estiver, ela é entregue ao processador. Se não, um bloco da memória principal, consistindo em algum número fixo de palavras, é transferido para a cache, e depois a palavra é fornecida ao processador. Em virtude do fenômeno da localidade de referência, quando um bloco de dados é levado para a cache para satisfazer uma única referência de memória, é provável que haja referências futuras a esse mesmo local da memória ou a outras palavras no mesmo bloco.

A Figura 4.3b representa o uso de múltiplos níveis de cache. A cache L2 é mais lenta e em geral maior que a cache L1, e a cache L3 é mais lenta e normalmente maior que a cache L2.

A Figura 4.4 representa a estrutura de um sistema de cache/memória principal. A memória principal consiste em até 2^n palavras endereçáveis, com cada palavra tendo um endereço distinto de n-bits. Para fins de mapeamento, essa memória é considerada como sendo uma série de blocos de tamanho fixo com K palavras cada. Ou seja, existem $M = 2^n/K$ blocos na memória principal. A cache consiste em m blocos, chamados de **linhas**.[3] Cada uma contém K palavras, mais um tag de alguns bits. Cada linha também inclui bits de controle (não mostrados), como um bit para indicar se a linha foi modificada desde que foi carregada na cache. A extensão de uma linha, sem incluir tag e bits de controle, é o **tamanho da linha**. O tamanho da linha pode ter apenas 32 bits, com cada "palavra" sendo um único byte; nesse caso, o tamanho da linha é de 4 bytes. O número de linhas é consideravelmente menor que o número de blocos da memória principal ($m << M$). A qualquer momento, algum subconjunto dos blocos de memória reside nas linhas na cache. Se uma palavra em um bloco de memória for lida, esse bloco é transferido para uma das linhas da cache. Como existem mais blocos do que linhas, uma linha individual não pode ser dedicada exclusiva e permanentemente a determinado bloco. Assim, cada linha inclui um **tag** que identifica qual bloco em particular está atualmente sendo armazenado. O tag em geral é uma parte do endereço da memória principal, conforme descrito posteriormente nesta seção.

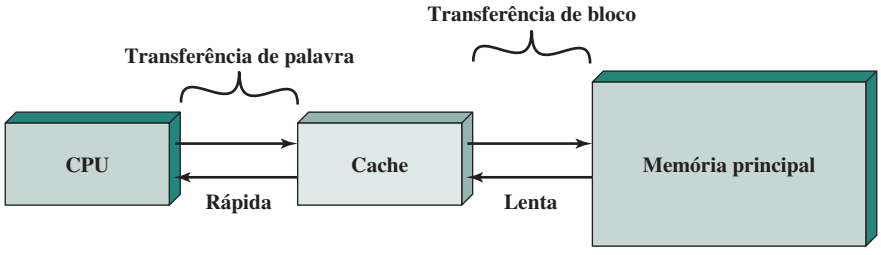
A Figura 4.5 ilustra a operação de leitura. O processador gera o endereço de leitura (RA — do inglês, *Read Address*) de uma palavra a ser lida. Se a palavra estiver na cache, ela é entregue ao processador. Caso contrário, o bloco contendo essa palavra é carregado na cache e então a palavra é entregue ao processador. A Figura 4.5 mostra essas duas operações ocorrendo em paralelo e reflete a organização mostrada na Figura 4.6, que é típica das organizações modernas de cache. Nessa organização, a cache conecta-se ao processador por meio de linhas de dados, controle e endereço. As linhas de dados e endereços também se conectam a buffers de dados e endereços, que se conectam a um barramento do sistema, do qual a memória principal é acessada. Quando ocorre um

2 Em geral, a cache de disco é uma técnica puramente de software, e não é examinada neste livro. Isso é discutido em outro livro deste autor (STALLINGS, 2015).

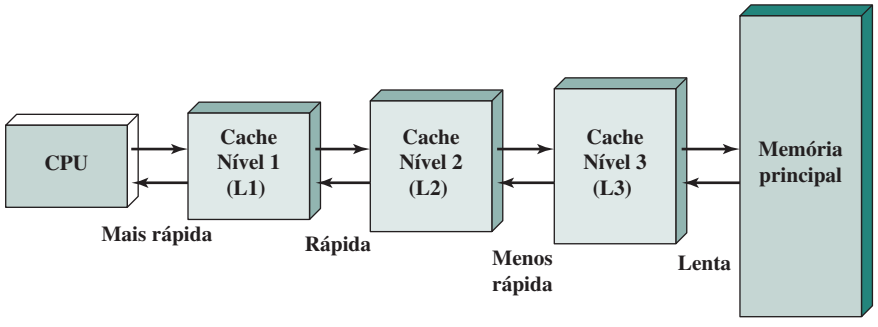
3 Referindo-se à unidade básica da cache, o termo *linha*, em vez de bloco, é usado por dois motivos: (1) para evitar confusão com um bloco da memória principal, que contém o mesmo número de palavras de dados que uma linha de cache; e (2) porque uma linha de cache inclui não apenas K palavras de dados, e um bloco da memória principal, mas também inclui tag e bits de controle.

Figura 4.3

Cache e memória principal.



(a) Cache única



(b) Organização de cache de três níveis

Figura 4.4

Estrutura de cache/memória principal.

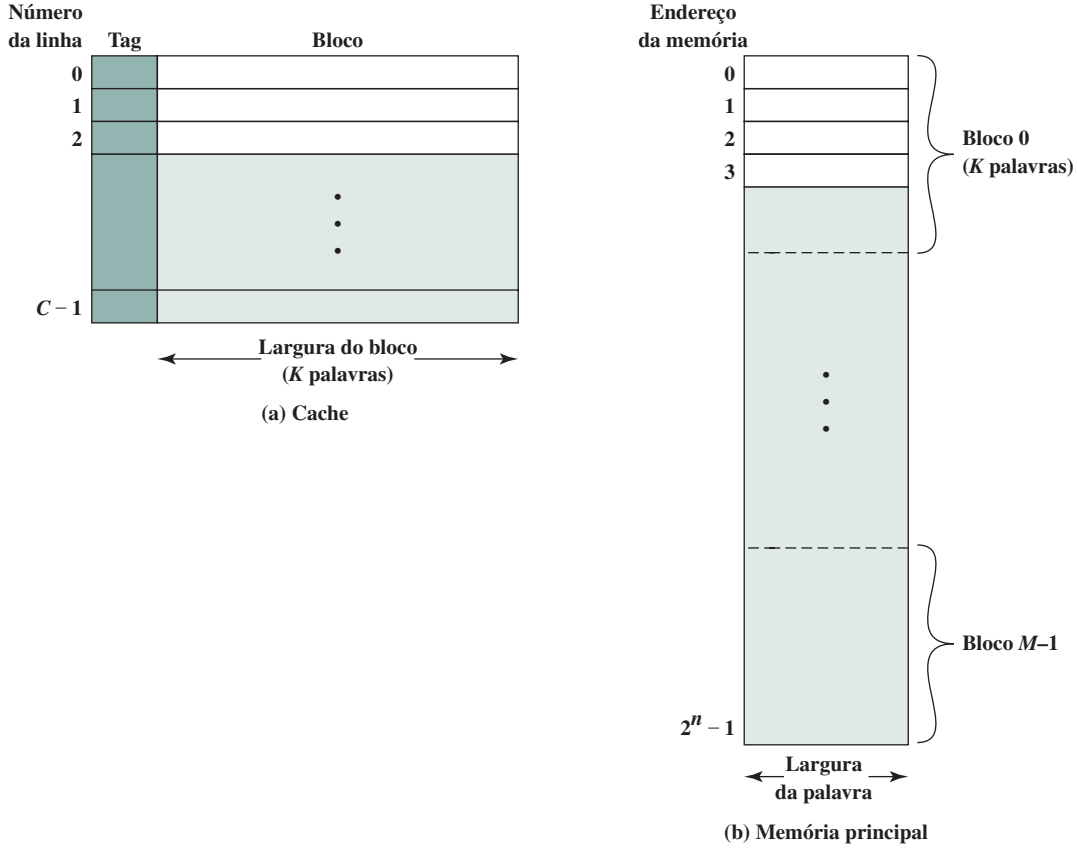


Figura 4.5

Operação de leitura de cache.

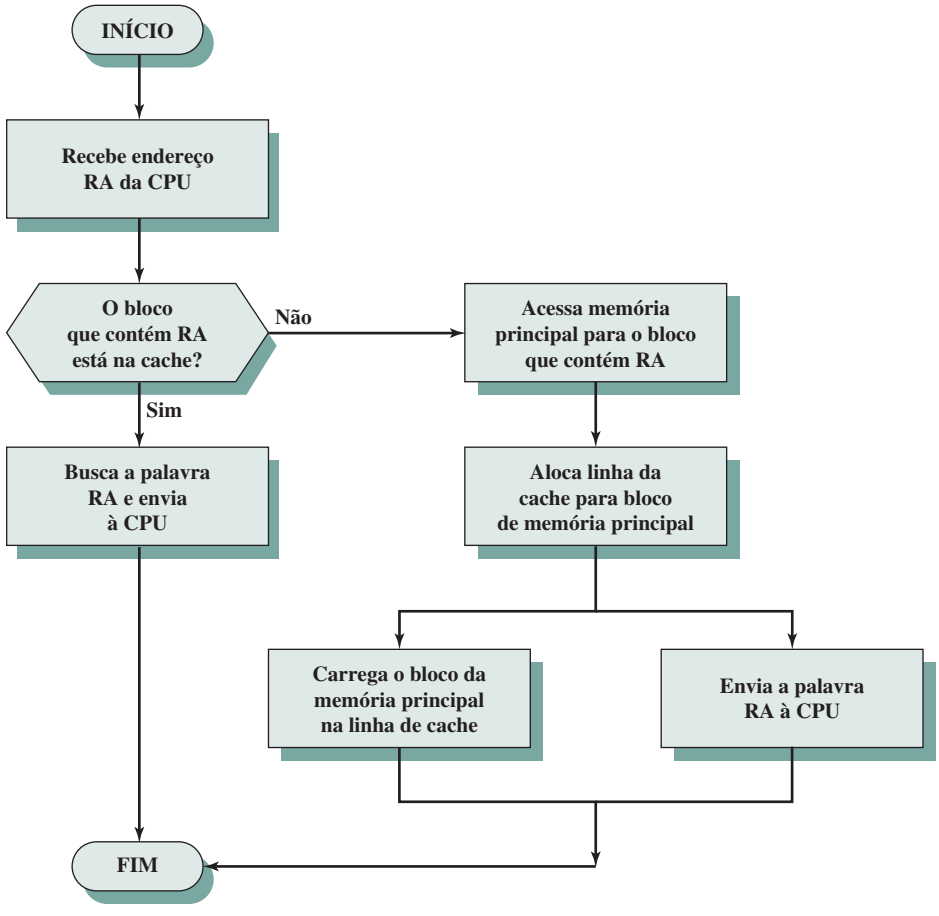
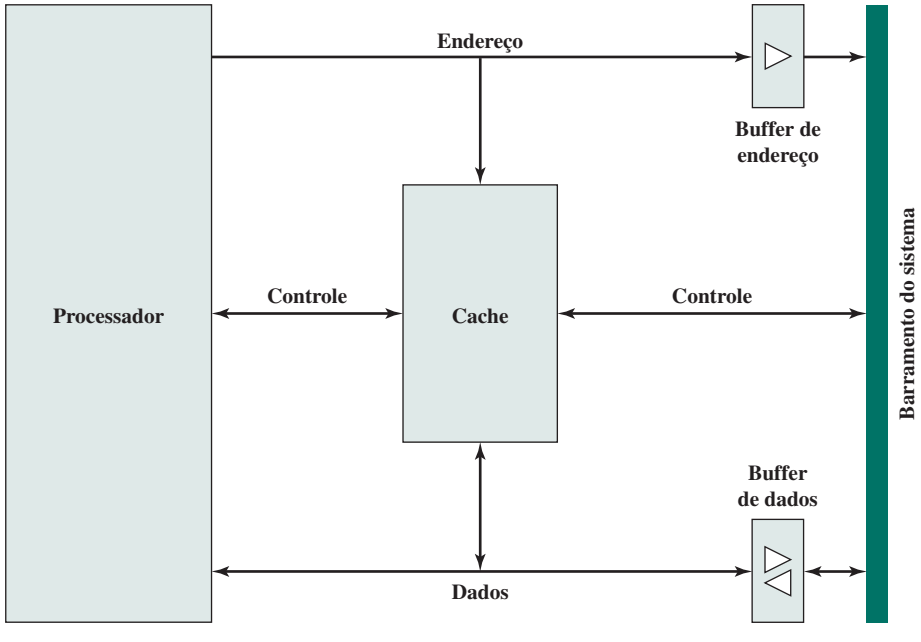


Figura 4.6

Organização típica da memória cache.



acerto de cache, os buffers de dados e endereço são desativados e a comunicação é apenas entre o processador e a memória cache, sem tráfego no barramento do sistema. Quando ocorre uma falha de cache, o endereço desejado é carregado no barramento do sistema e os dados são transferidos através do buffer de dados para a cache e para o processador. Em outras organizações, a cache é fisicamente interposta entre o processador e a memória principal para todas as linhas de dados, endereço e controle. Nesse último caso, para uma falha de cache, a palavra desejada primeiro é transferida para a cache e depois transferida da cache para o processador.

Uma discussão sobre os parâmetros de desempenho relacionados ao uso da cache pode ser vista no Apêndice 4A.

4.3 ELEMENTOS DE PROJETO DA CACHE

Esta seção oferece uma visão geral dos parâmetros de projeto de memória cache e informa alguns resultados típicos. Ocasionalmente, nos referimos ao uso de caches na **computação de alto desempenho (HPC — do inglês, *High Performance Computing*)**. A HPC lida com supercomputadores e seus softwares, especialmente para aplicações científicas, que envolvem grandes quantidades de dados, cálculos de vetores e matrizes e uso de algoritmos paralelos. O projeto de memória cache para HPC é muito diferente daquele para outras plataformas de hardware e aplicações. Na verdade, muitos pesquisadores descobriram que aplicações HPC não funcionam bem em arquiteturas de computador que empregam memórias caches (BAILEY, 1993). Outros pesquisadores, desde então, têm demonstrado que uma hierarquia de memória cache pode ser útil para melhorar o desempenho se o software de aplicação for ajustado para explorar a cache (WANG; TAFTI, 1999, PRESSEL, 2001).[4]

Embora haja um grande número de implementações de memória cache, existem alguns elementos básicos de projeto que servem para classificar e diferenciar as arquiteturas de memórias cache. A Tabela 4.2 lista os principais elementos.

Endereços da cache

Quase todos os processadores não embarcados, e muitos processadores embarcados, suportam memória virtual, um conceito discutido no Capítulo 8. Basicamente, a memória virtual é uma facilidade que permite que os programas enderecem a memória a partir de um ponto de vista lógico, sem considerar a quantidade de memória principal disponível fisicamente. Quando a memória virtual é usada, os campos de endereço das instruções de máquina contêm endereços virtuais. Para leituras e escritas da memória principal, uma unidade de gerenciamento da memória (MMU — do inglês, *Memory Management Unit*) traduz cada endereço virtual para um endereço físico na memória principal.

Tabela 4.2

Elementos do projeto de cache.

Endereços da cache	Política de escrita
Lógico	*Write through*
Físico	*Write back*
Tamanho da memória cache Função de mapeamento	**Tamanho da linha Número de caches**
Direto	Um ou dois níveis
Associativo	Unificada ou separada
Associativo em conjunto	
Algoritmo de substituição	
Usado menos recentemente (LRU — do inglês, *Least Recently Used*)	
Primeiro a entrar, primeiro a sair (FIFO — do inglês, *First In, First Out*)	
Usado menos frequentemente (LFU — do inglês, *Least Frequently Used*)	
Aleatória	

4 Para ver uma discussão geral sobre HPC, consulte a obra de Dowd e Severance (1998).

Quando são usados endereços virtuais, o projetista do sistema pode escolher colocar a cache entre o processador e a MMU ou entre a MMU e a memória principal (Figura 4.7). Uma **cache lógica**, também conhecida como **cache virtual**, armazena dados usando **endereços virtuais**. O processador acessa a cache diretamente, sem passar pela MMU. Uma **cache física** armazena dados usando **endereços físicos** da memória principal.

Uma vantagem óbvia da cache lógica é que a velocidade de acesso a ela é maior do que para uma cache física, pois a cache pode responder antes que a MMU realize uma tradução de endereço. A desvantagem é que a maioria dos sistemas de memória virtual fornece, a cada aplicação, o mesmo espaço de endereços de memória virtual. Ou seja, cada aplicação vê uma memória virtual que começa no endereço 0. Assim, o mesmo endereço virtual em duas aplicações diferentes refere-se a dois endereços físicos diferentes. A memória cache, portanto, precisa ser completamente esvaziada a cada troca de contexto da aplicação, ou então bits extras precisam ser adicionados a cada linha da cache para identificar a que espaço de endereço virtual esse endereço se refere.

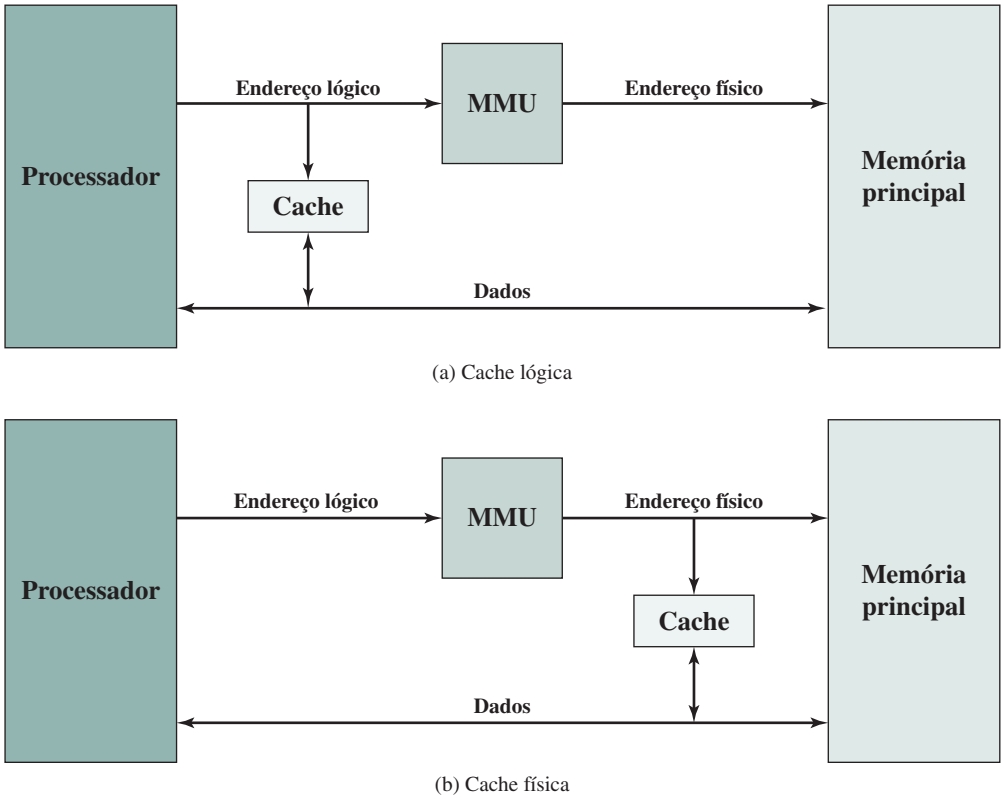
O assunto de cache lógica *versus* física é complexo, e está fora do escopo deste livro. Para obter uma discussão mais profunda, consulte Cekleov (1997) e Jacob (2008).

Tamanho da memória cache

O primeiro item na Tabela 4.2, o tamanho da memória cache, já foi discutido. Gostaríamos que o tamanho da cache fosse pequeno o suficiente para que o custo médio geral por bit fosse próximo do custo médio da memória principal isolada e grande o suficiente para que o tempo de acesso médio geral fosse próximo do tempo de acesso médio da cache isolada. Existem várias outras motivações para minimizar o tamanho da cache. Quanto maior a cache, maior o número de portas envolvidos no endereçamento da cache. O resultado é que caches grandes tendem a ser ligeiramente mais lentas que as pequenas — mesmo quando construídas com a mesma tecnologia de circuito integrado e colocadas no mesmo lugar no chip e na placa de circuito. A área disponível do chip e da placa também limita o tamanho da cache. Como o desempenho da cache é muito sensível à natureza da carga de trabalho, é impossível chegar a um único tamanho ideal de cache. A Tabela 4.3 lista os tamanhos de cache de alguns processadores atuais e antigos.

Figura 4.7

Caches lógicas e físicas.



(a) Cache lógica

(b) Cache física

Função de mapeamento

Como existem menos linhas de cache do que blocos da memória principal, é necessário haver um algoritmo para mapear os blocos da memória principal às linhas de cache. Além do mais, é preciso haver um meio para determinar qual bloco da memória principal atualmente ocupa uma linha da cache. A escolha da função de mapeamento dita como a cache é organizada. Três técnicas podem ser utilizadas: direta, associativa e associativa por conjunto. Vamos examinar uma por vez. Em cada caso, examinamos a estrutura geral e depois um exemplo específico.

EXEMPLO 4.2

Para todos os três casos, o exemplo inclui os seguintes elementos:

- A cache pode manter 64 kB.
- Os dados são transferidos entre a memória principal e a cache em blocos de 4 bytes cada. Isso significa que a cache é organizada como 16 K = 2^{14} linhas de 4 bytes cada.
- A memória principal consiste em 16 MB, com cada byte endereçável diretamente por um endereço de 24 bits (2^{24} = 16 M). Assim, para fins de mapeamento, podemos considerar que a memória principal consiste em 4 M blocos de 4 bytes cada.

Tabela 4.3

Tamanhos de cache de alguns processadores.

Processador	Tipo	Ano de introdução	Cache L1[a]	Cache L2	Cache L3
IBM 360/85	Mainframe	1968	16–32 kB	—	—
PDP-11/70	Minicomputador	1975	1 kB	—	—
VAX 11/780	Minicomputador	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/servidor	1999	32 kB/32 kB	256 kB a 1 MB	2 MB
IBM S/390 G6	Mainframe	1999	256 kB	8 MB	—
Pentium 4	PC/servidor	2000	8 kB/8 kB	256 kB	—
IBM SP	Servidor avançado/ supercomputador	2000	64 kB/32 kB	8 MB	—
CRAY MTA[b]	Supercomputador	2000	8 kB	2 MB	—
Itanium	PC/servidor	2001	16 kB/16 kB	96 kB	4 MB
Itanium 2	PC/servidor	2002	32 kB	256 kB	6 MB
IBM POWER5	Servidor avançado	2003	64 kB	1.9 MB	36 MB
CRAY XD-1	Supercomputador	2004	64 kB/64 kB	1 MB	—
IBM POWER6	PC/servidor	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	Estação de trabalho/ servidor	2011	6 × 32 kB/ 32kB	1,5 MB	12 MB
IBM zEnterprise 196	Mainframe/ servidor	2011	24 × 64 kB/ 128 kB	24 × 1,5 MB	24 MB L3 192 MB L4

[a] Dois valores separados por uma barra referem-se a caches de instrução e dados. [b] As duas caches são apenas de instrução; não há caches de dados.

MAPEAMENTO DIRETO A técnica mais simples, conhecida como mapeamento direto, mapeia cada bloco da memória principal a apenas uma linha de cache possível. O mapeamento é expresso como:

$$i = j \text{ módulo } m$$

em que

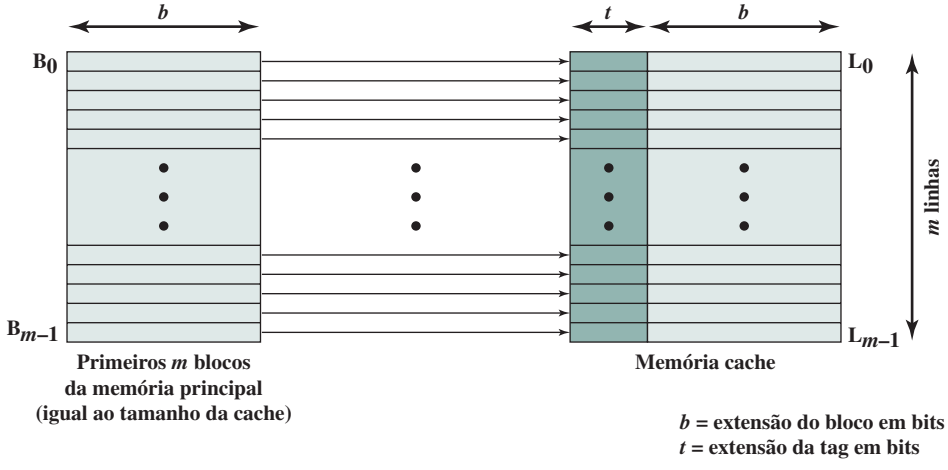
i = número da linha da cache
j = número do bloco da memória principal
m = número de linhas da cache

A Figura 4.8a mostra o mapeamento para os primeiros m blocos de memória principal. Cada bloco da memória principal mapeia uma linha exclusiva da cache. Os próximos m blocos da memória principal mapeiam a cache da mesma forma; ou seja, o bloco B_m da memória principal mapeia a linha L_0 da cache, o bloco B_{m+1} mapeia a linha L_1, e assim por diante.

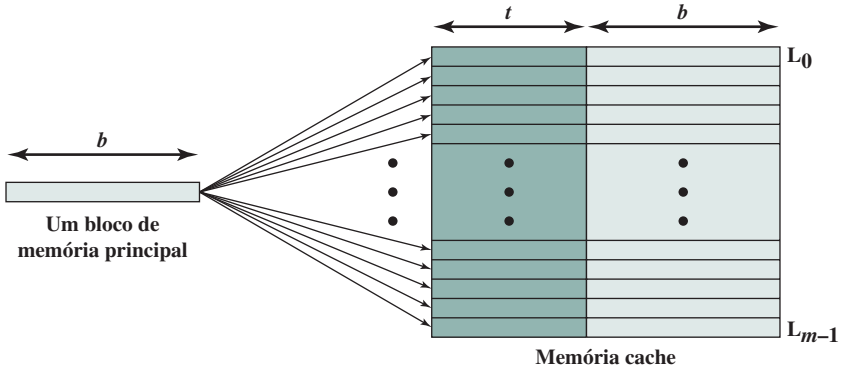
A função de mapeamento é facilmente implementada por meio do endereço da memória principal. A Figura 4.9 ilustra o mecanismo geral. Para fins de acesso à cache, cada endereço da memória principal pode ser visto como consistindo em três campos. Os w bits menos significativos identificam uma palavra ou um byte dentro de um bloco da memória principal; na maioria das máquinas modernas, o endereço está no nível de byte. Os s bits restantes especificam um dos 2^s blocos da memória principal. A lógica de cache interpreta esses s bits como uma tag de $s - r$ bits (parte mais significativa) e um campo de linha de r bits. O segundo campo identifica uma das $m = 2^r$ linhas da cache. Resumindo:

Figura 4.8

Mapeamento da memória principal para a cache: direto e associativo.



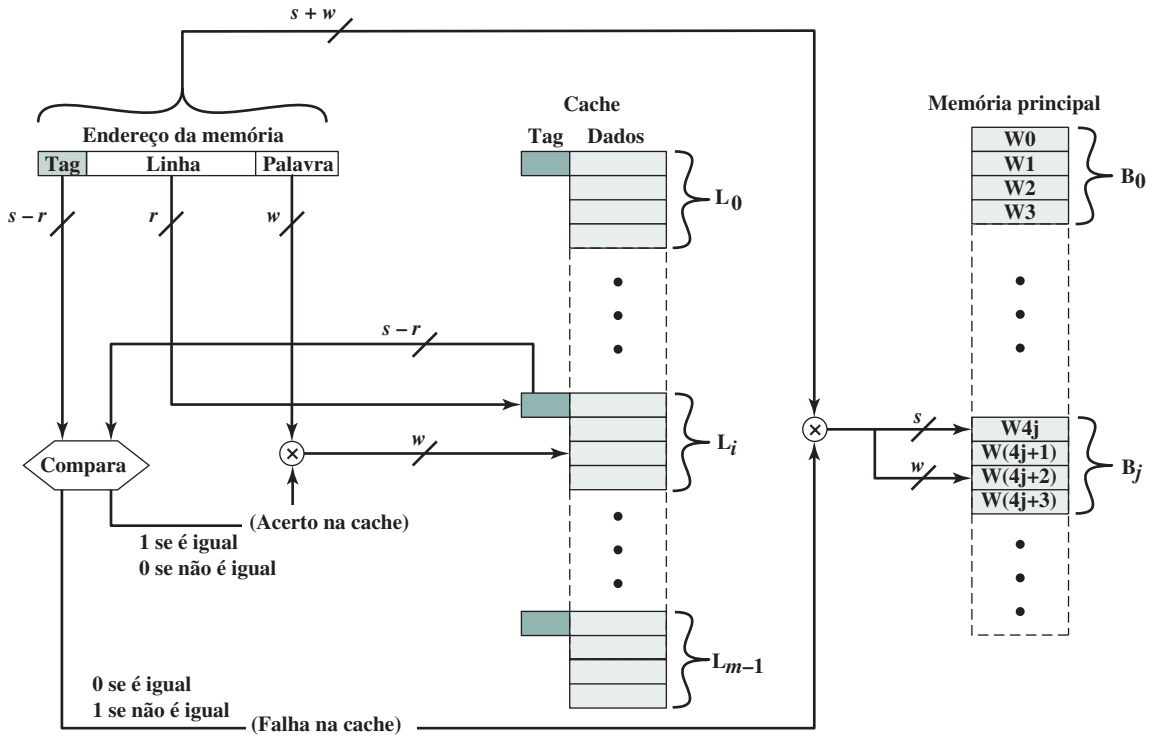
(a) Mapeamento direto



(b) Mapeamento associativo

Figura 4.9

Organização de cache com mapeamento direto.



- Tamanho do endereço = $(s + w)$ bits.
- Número de unidades endereçáveis = 2^{s+w} palavras ou bytes.
- Tamanho do bloco = tamanho da linha = 2^w palavras ou bytes.
- Número de blocos na memória principal = $\frac{2^{s+w}}{2^w} = 2^s$.
- Número de linhas na cache = $m = 2^r$.
- Tamanho da cache = 2^{r+w} palavras ou bytes.
- Tamanho da tag = $(s - r)$ bits.

EXEMPLO 4.2a

A Figura 4.10 mostra nosso sistema exemplo usando o mapeamento direto[5]. No exemplo, $m = 16K = 2^{14}$ e $i = j$ módulo 2^{14}. O mapeamento torna-se:

Linha de cache	Endereço de memória inicial do bloco
0	000000, 010000, ..., FF0000
1	000004, 010004, ..., FF0004
⋮	⋮
$2^{14} - 1$	00FFFC, 01FFFC, ..., FFFFFC

Observe que não existem dois blocos mapeados para o mesmo número de linha que tenham o mesmo número de tag. Assim, os blocos com endereços iniciais 000000, 010000, ..., FF0000 possuem números de tag 00, 01, ..., FF, respectivamente.

5 Nesta e nas figuras seguintes, os valores de memória são representados em notação hexadecimal. Veja, no Capítulo 9, um manual básico sobre sistemas numéricos (decimal, binário, hexadecimal).

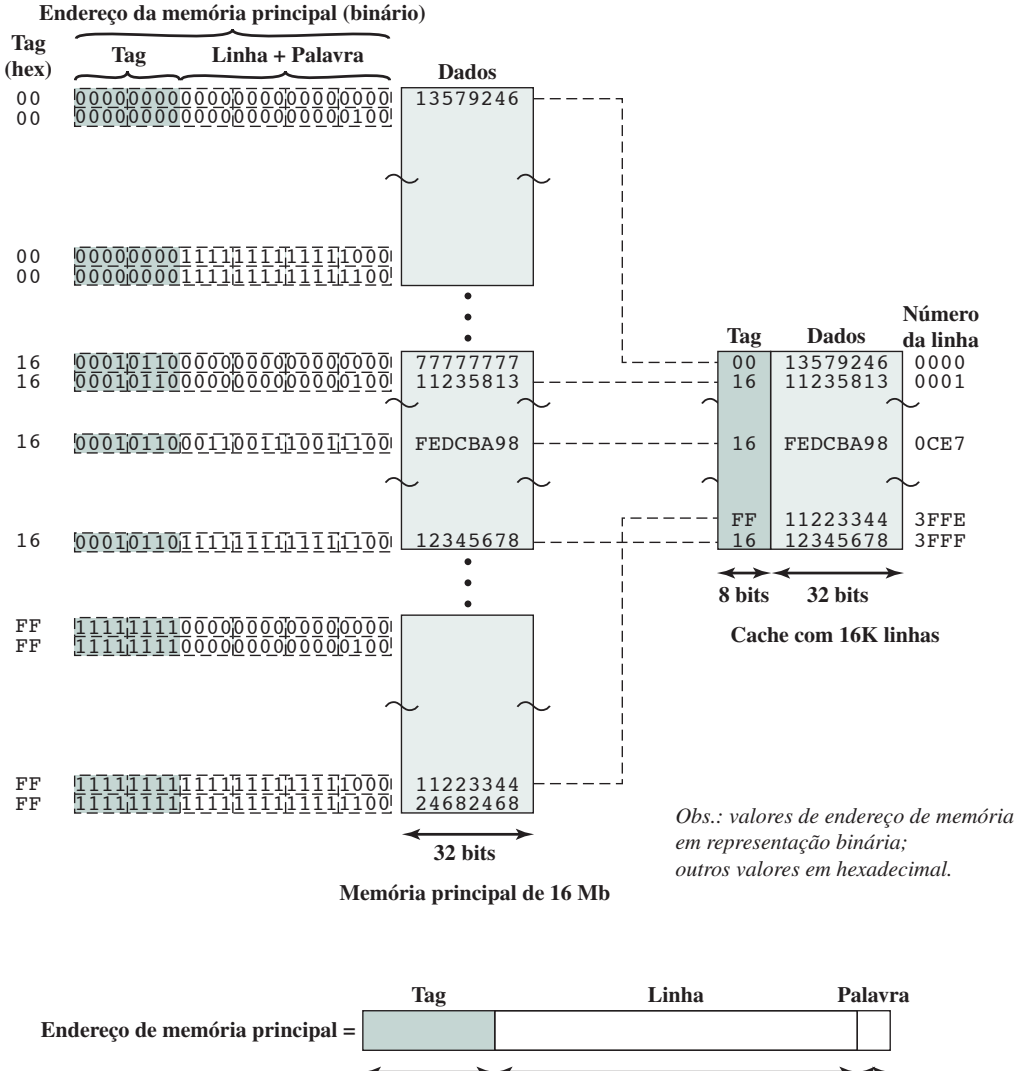
Retornando à Figura 4.5, uma operação de leitura funciona da seguinte forma. O sistema da memória cache recebe um endereço de 24 bits. O número de linha com 14 bits é usado como um índice para a cache acessar uma linha em particular. Se o número de tag com 8 bits for igual ao número de tag atualmente armazenado nessa linha, então o número da palavra com 2 bits é usado para selecionar um dos 4 bytes nessa linha. Caso contrário, o campo de tag-mais-linha com 22 bits é usado para buscar um bloco da memória principal. O endereço real que é usado para a busca é o campo de tag-mais-linha com 22 bits concatenado com dois bits 0, de modo que 4 bytes sejam apanhados a partir do início do bloco.

O efeito desse mapeamento é que os blocos da memória principal são alocados nas linhas da cache, como mostrado a seguir:

Linha de cache	Blocos de memória principal mapeados
0	$0, m, 2m, \ldots, 2^s - m$
1	$1, m + 1, 2m + 1, \ldots, 2^s - m + 1$
⋮	⋮
$m - 1$	$m - 1, 2m - 1, 3m - 1, \ldots, 2^s - 1$

Figura 4.10

Exemplo de mapeamento direto.



Assim, o uso de uma parte do endereço como o número da linha oferece um mapeamento exclusivo de cada bloco da memória principal à cache. Quando um bloco é armazenado em sua respectiva linha, é necessário marcar os dados para distingui-los de outros blocos que podem ser alocados nessa linha. Os $s - r$ bits mais significativos têm esse propósito.

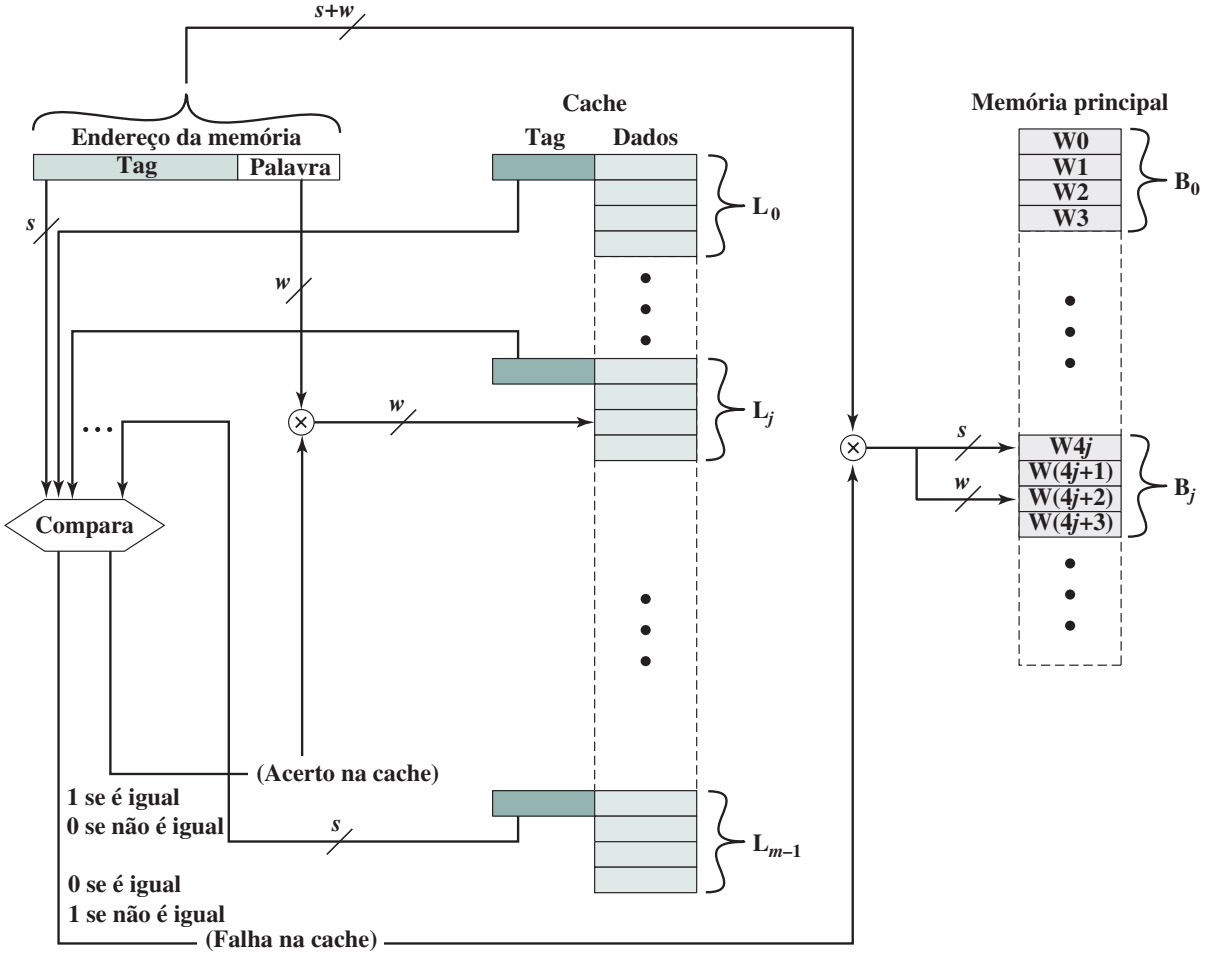
A técnica de mapeamento direto é simples e pouco dispendiosa para se implementar. Sua principal desvantagem é que existe um local de cache fixo para cada bloco. Assim, se um programa referenciar palavras repetidamente de dois blocos diferentes, mapeados para a mesma linha, então os blocos serão continuamente trocados na cache, e a razão de acerto será baixa (um fenômeno conhecido como *thrashing*).

Uma técnica para diminuir a penalidade de **falha** é guardar o que foi descartado caso seja necessário novamente. Como os dados descartados já foram lidos, podem ser usados novamente a um custo pequeno. Essa reciclagem é possível usando uma *victim cache*. A *victim cache* foi proposta primeiro como um método de reduzir as perdas de conflito das caches mapeadas diretamente sem afetar seu tempo de acesso. Trata-se de uma cache totalmente associativa, cujo tamanho em geral é de 4 a 16 linhas de cache, residindo entre uma cache L1 mapeada diretamente e o próximo nível de memória. Esse conceito é explorado no Apêndice F (disponível em inglês na Sala Virtual).

MAPEAMENTO ASSOCIATIVO O mapeamento associativo compensa a desvantagem do mapeamento direto, permitindo que cada bloco da memória principal seja carregado em qualquer linha da cache (Figura 4.8b). Nesse caso, a lógica de controle da cache interpreta um endereço de memória simplesmente como um campo Tag e um campo Palavra. O campo Tag identifica o bloco da memória principal. Para determinar se um bloco está na cache, a lógica de controle da cache precisa comparar simultaneamente o tag de cada linha. A Figura 4.11 ilustra a lógica.

Figura 4.11

Organização da memória cache totalmente associativa.



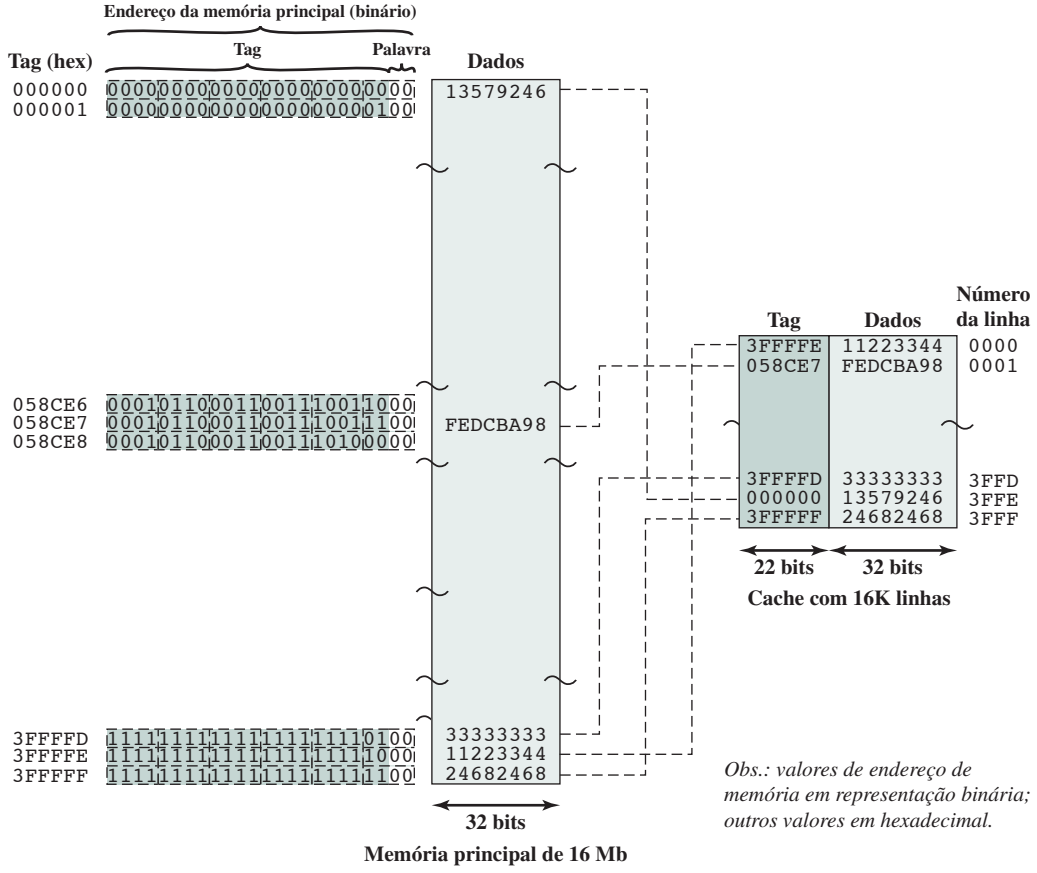
EXEMPLO 4.2b

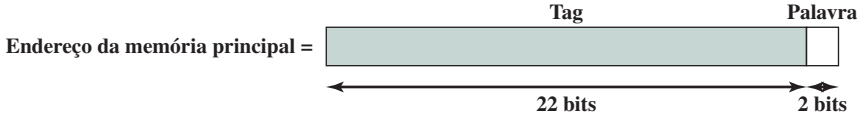
A Figura 4.12 mostra nosso exemplo usando o mapeamento associativo. Um endereço da memória principal consiste em um tag de 22 bits e um número do byte de 2 bits. O tag de 22 bits precisa ser armazenado com o bloco de dados de 32 bits para cada linha na cache. Observe que são os 22 bits mais à esquerda (mais significativos) do endereço que formam o tag. Assim, o endereço hexadecimal de 24 bits 16339C tem o tag de 22 bits 058CE7. Isso pode ser visto facilmente na notação binária:

Endereço da memória	0001	0110	0011	0011	1001	1100	(binário)
	1	6	3	3	9	C	(hexa)
Tag (22 bits mais à esquerda)	00	0101	1000	1100	1110	0111	(binário)
	0	5	8	C	E	7	(hexa)

Figura 4.12

Exemplo de mapeamento associativo.





Observe que nenhum campo no endereço corresponde ao número de linha, de modo que o número de linhas na cache não é determinado pelo formato do endereço. Resumindo,

- Tamanho do endereço = $(s + w)$ bits.
- Número de unidades endereçáveis = 2^{s+w} palavras ou bytes.
- Tamanho do bloco = tamanho da linha = 2^w palavras ou bytes.
- Número de blocos na memória principal = $\frac{2^{s+w}}{2^w} = 2^s$.
- Número de linhas na cache = indeterminado.
- Tamanho da tag = s bits.

Com o mapeamento associativo, existe flexibilidade em relação a qual bloco substituir quando um novo bloco for lido para a cache. Os algoritmos de substituição, discutidos mais adiante nesta seção, são projetados para maximizar a razão de acerto. A principal desvantagem do mapeamento associativo é a complexidade do circuito necessário para examinar as tags de todas as linhas da cache em paralelo.

MAPEAMENTO ASSOCIATIVO POR CONJUNTO O mapeamento associativo por conjunto é um meio-termo que realça os pontos fortes das técnicas direta e associativa, enquanto reduz suas desvantagens.

Neste caso, a cache é uma série de conjuntos, cada um consistindo em uma série de linhas. As relações são:

$$m = v \times k$$

$$i = j \text{ módulo } v$$

em que

i = número do conjunto de cache
j = número de bloco da memória principal
m = número de linhas na cache
v = número de conjuntos
k = número de linhas em cada conjunto

Isso é conhecido como mapeamento associativo em conjunto com k-linhas. Com o mapeamento associativo em conjunto, o bloco B_j pode ser mapeado para qualquer uma das linhas do conjunto j. A Figura 4.13a ilustra esse mapeamento para os primeiros v blocos da memória principal. Assim como no mapeamento associativo, cada palavra é mapeada para múltiplas linhas de cache. Para o mapeamento associativo em conjunto, cada palavra é mapeada para todas as linhas de cache em um conjunto específico, de modo que o bloco B_0 da memória principal é mapeado no conjunto 0, e assim por diante. Assim, a cache associativa em conjunto pode ser implementada fisicamente como v caches associativas. Também é possível implementar a cache associativa em conjunto como k caches de mapeamento direto, como mostrado na Figura 4.13b. Cada cache mapeada diretamente é conhecida como uma *via*, consistindo em v linhas. As primeiras v linhas da memória principal são mapeadas diretamente nas v linhas de cada *via*; o próximo grupo de v linhas da memória principal é mapeado de modo similar, e assim por diante. A implementação mapeada diretamente em geral é usada para pequenos graus de associatividade (valores pequenos de k), enquanto a implementação com mapeamento associativo costuma ser usada para graus de associatividade mais altos (JACOB; WANG, 2008).

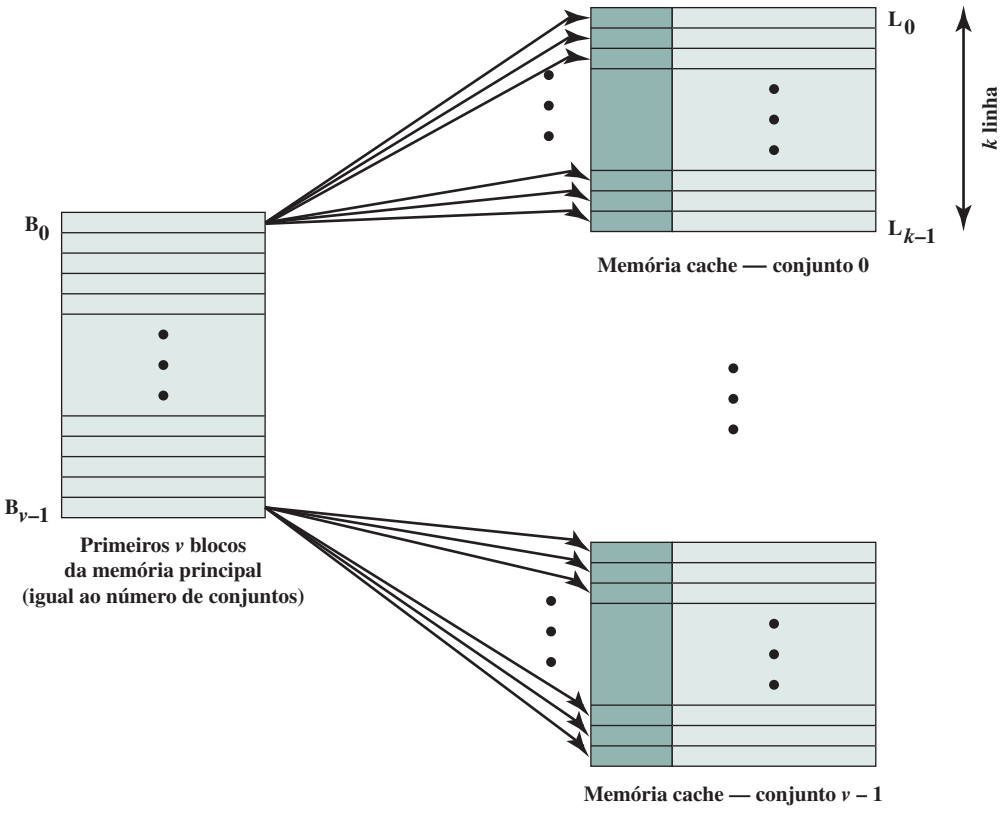
Para o mapeamento associativo em conjunto, a lógica de controle de cache interpreta um endereço de memória como três campos: Tag, Conjunto e Palavra. Os d bits especificam um dos $v = 2^d$ conjuntos. Os s bits dos campos Tag e Conjunto especificam um dos 2^s blocos da memória principal. Figura 4.14 ilustra a lógica de controle de cache. Com o mapeamento totalmente associativo, a tag em um endereço de memória é muito grande e precisa ser comparada à tag de cada linha na cache. Com o mapeamento associativo em conjunto com k linhas (k-*way*), a tag em um endereço de memória é muito menor e só é comparada com as k tags dentro de um único conjunto. Resumindo:

- Tamanho do endereço = $(s + w)$ bits.
- Número de unidades endereçáveis = 2^{s+w} palavras ou bytes.
- Tamanho do bloco = tamanho da linha = 2^w palavras ou bytes.
- Número de blocos na memória principal = $\frac{2^{s+w}}{2^w} = 2^s$.

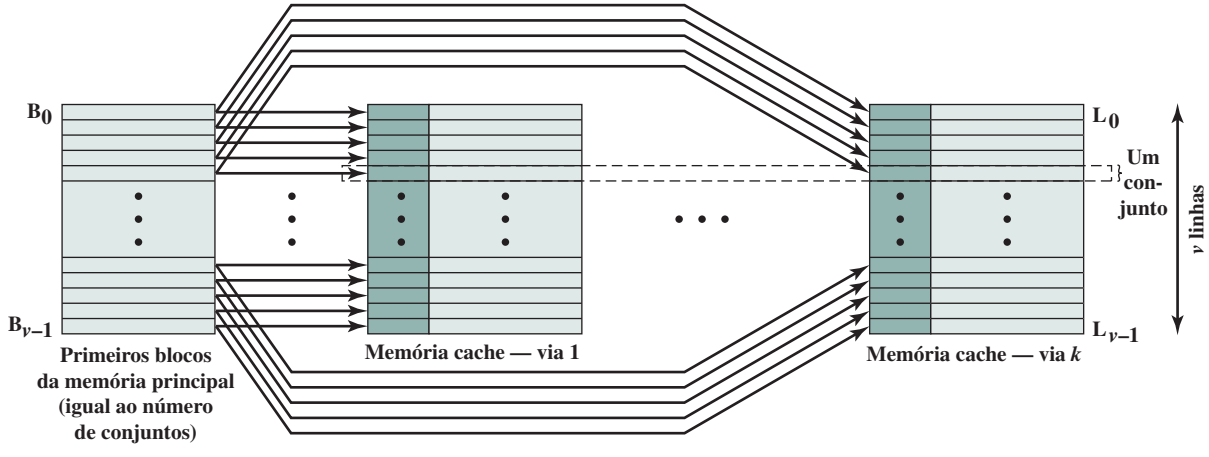
- Número de linhas no conjunto = k.
- Número de conjuntos = $v = 2^d$.
- Número de linhas na cache = $m = kv = k \times 2^d$.
- Tamanho da cache = $k \times 2^{d+w}$ palavras ou bytes.
- Tamanho da tag = $(s - d)$ bits.

Figura 4.13

Mapeamento da memória principal na cache: associativa em conjunto com k linhas (*k-way*).



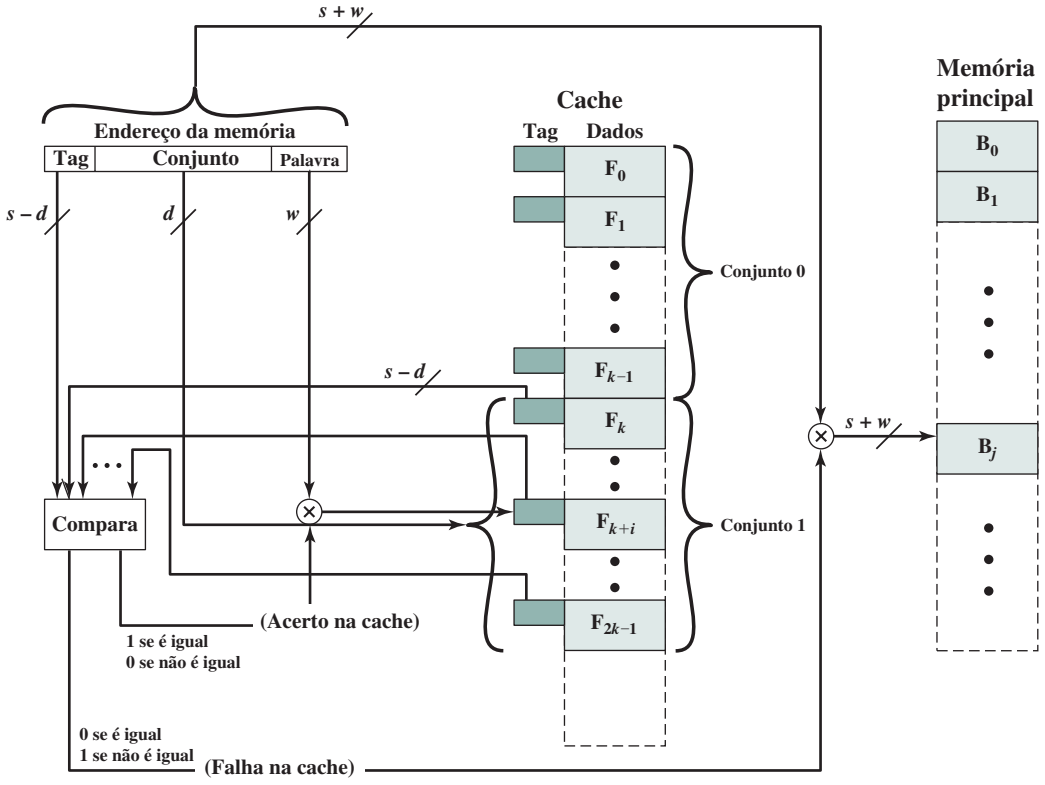
(a) Caches mapeadas associativamente com v conjuntos



(b) Caches mapeadas diretamente com k linhas

Figura 4.14

Organização da memória cache associativa em conjunto com *k* linhas.



EXEMPLO 4.2c

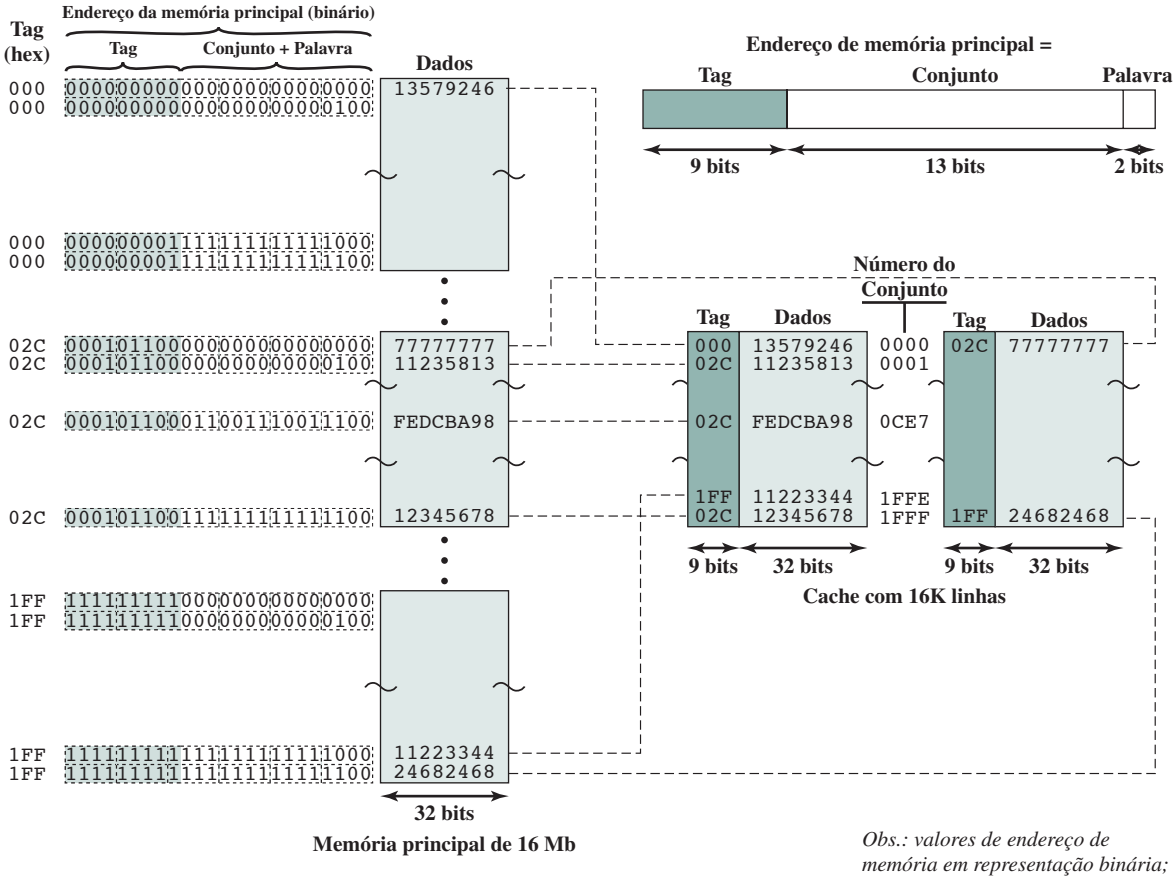
A Figura 4.15 mostra nosso exemplo usando o mapeamento associativo em conjunto com duas linhas em cada conjunto, denominado associativo em conjunto 2-*way*. O número de 13 bits identifica um conjunto exclusivo de duas linhas dentro da cache. Ele também oferece o número do bloco na memória principal, módulo 2^{13}. Isso determina o mapeamento dos blocos nas linhas. Assim, os blocos 000000, 008000, ..., FF8000 da memória principal são mapeados no conjunto 0 da cache. Qualquer um desses blocos pode ser carregado em qualquer uma das duas linhas no conjunto. Observe que nenhum dos dois blocos mapeados no mesmo conjunto de cache possui o mesmo número de tag. Para uma operação de leitura, o número de 13 bits é usado para determinar qual conjunto de duas linhas deve ser examinado. As duas linhas no conjunto são examinadas comparando-as com o número do tag do endereço a ser acessado.

No caso extremo de $v = m, k = 1$, a técnica associativa em conjunto se reduz ao mapeamento direto, e para $v = 1, k = m$, ela se reduz ao mapeamento associativo. O uso de duas linhas por conjunto ($v = m/2, k = 2$) é a organização associativa em conjunto mais comum. Ela melhora significativamente a razão de acerto em relação ao mapeamento direto. A associação em conjunto com quatro linhas por conjunto ($v = m/4, k = 4$) cria uma melhoria adicional modesta por um custo adicional relativamente pequeno (MAYBERRY; EFLAND, 1984, HILL, 1989). Outros aumentos no número de linhas por conjunto têm pouco efeito.

A Figura 4.16 mostra os resultados de um estudo de simulação do desempenho da cache associativa em conjunto em função do tamanho da cache (GENU, 2004). A diferença no desempenho entre mapeamento direto e associativo em conjunto com duas linhas por conjunto é significativa até pelo menos um tamanho de cache de 64 kB. Observe também que a diferença entre duas linhas por conjunto e quatro linhas a 4 kB é muito menor do que a diferença ao passar de 4 kB para 8 kB no tamanho da cache. A complexidade da cache aumenta em proporção com a associatividade e, nesse caso, não seria justificável contra o aumento no tamanho da cache para 8 ou mesmo 16 kB. Um ponto final a ser observado é que, além de cerca de 32 kB, o aumento no tamanho da cache não ocasiona aumento significativo no desempenho.

Figura 4.15

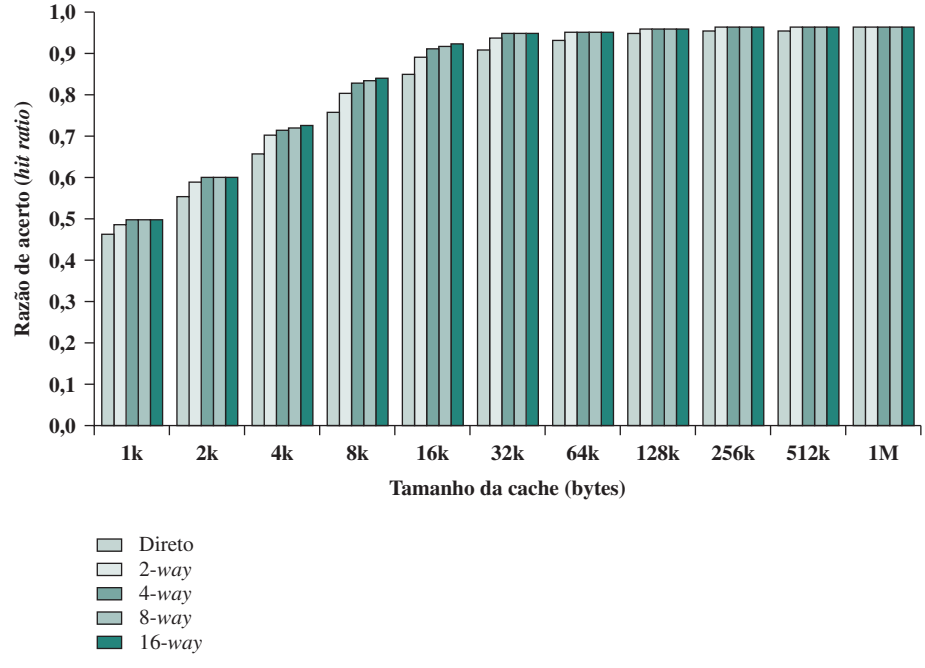
Exemplo de mapeamento associativo em conjunto com duas linhas.



Obs.: valores de endereço de memória em representação binária; outros valores em hexadecimal.

Figura 4.16

Associatividade variável pelo tamanho da cache.



Os resultados da Figura 4.16 são baseados na simulação da execução de um compilador GCC. Diferentes aplicações podem gerar resultados diferentes. Por exemplo, Cantin e Hill (2001) relatam os resultados para o desempenho da cache usando muitos dos benchmarks SPEC CPU2000. Os resultados de Cantin e Hill na comparação da razão de acerto com o tamanho da cache seguem o mesmo padrão da Figura 4.16, mas os valores específicos são ligeiramente diferentes.

Algoritmos de substituição

Uma vez que a cache esteja cheia, e um novo bloco seja trazido para a cache, um dos blocos existentes precisa ser substituído. Para o mapeamento direto, existe apenas uma linha possível para qualquer bloco em particular, e nenhuma escolha é possível. Para as técnicas associativa e associativa em conjunto, um algoritmo de substituição é necessário. Para alcançar alta velocidade, tal algoritmo precisa ser implementado em hardware. Diversos algoritmos foram experimentados. Mencionamos quatro dos mais comuns. Provavelmente, o mais eficaz é o **usado menos recentemente (LRU — do inglês, *Least Recently Used*)**: substitua aquele bloco no conjunto que permaneceu na cache por mais tempo sem qualquer referência a ele. Para a associatividade em conjunto com duas linhas por conjunto, isso é facilmente implementado. Cada linha inclui um bit USE. Quando uma linha é referenciada, seu bit USE é definido como 1, e o bit USE da outra linha nesse conjunto é definido como 0. Quando um bloco lido for direcionado para um conjunto, a linha cujo bit USE for 0 é utilizada para este bloco. Como estamos supondo que os locais de memória usados mais recentemente são mais prováveis de serem referenciados, o LRU deverá dar a melhor razão de acerto. O LRU também é relativamente fácil de implementar para uma cache totalmente associativa. O mecanismo de cache mantém uma lista separada de índices para todas as linhas na cache. Quando uma linha é referenciada, ela passa para a frente da lista. Para substituição, a linha no final da lista é usada. Por conta de sua simplicidade de implementação, o LRU é o algoritmo de substituição mais popular.

Outra possibilidade é **"primeiro a entrar, primeiro a sair" (FIFO — do inglês, *First In, First Out*)**: substitua o bloco no conjunto que esteve na cache por mais tempo. O algoritmo FIFO é facilmente implementado como uma técnica *round-robin* ou de buffer circular. Outra possibilidade de algoritmo, ainda, é o **usado menos frequentemente (LFU — do inglês, *Least Frequently Used*)**: substitua aquele bloco no conjunto que teve menos referências. O algoritmo LFU poderia ser implementado associando um contador a cada linha. Uma técnica não baseada no uso (ou seja, não LRU, LFU, FIFO ou alguma variante) é escolher uma linha aleatória entre as linhas candidatas. Estudos de simulação têm mostrado que a substituição aleatória oferece um desempenho apenas ligeiramente inferior a um algoritmo baseado no uso (SMITH, 1982).

Política de escrita

Quando um bloco que está residente na cache estiver para ser substituído, existem dois casos a serem considerados. Se o bloco antigo na cache não tiver sido alterado, então ele pode ser substituído por um novo bloco sem primeiro atualizar o bloco antigo. Se pelo menos uma operação de escrita tiver sido realizada em uma palavra nessa linha da cache, então a memória principal precisa ser atualizada escrevendo a linha de cache no bloco de memória antes de trazer o novo bloco. Diversas políticas de escrita são possíveis, com escolhas econômicas e de desempenho. Existem dois problemas a serem considerados. Primeiro, mais de um dispositivo pode ter acesso à memória principal. Por exemplo, um módulo de E/S pode ser capaz de ler-escrever diretamente na memória. Se uma palavra tiver sido alterada apenas na cache, então a palavra correspondente da memória é inválida. Além do mais, se o dispositivo de E/S tiver alterado a memória principal, então a palavra da cache é inválida. Um problema mais complexo ocorre quando múltiplos processadores são conectados ao mesmo barramento e cada processador tem sua própria cache local. Então, se uma palavra for alterada em uma cache, ela possivelmente poderia invalidar esta palavra em outras caches.

A técnica mais simples é denominada ***write through***. Usando esta técnica, todas as operações de escrita são feitas na memória principal e também na cache, garantindo que a memória principal sempre seja válida. Qualquer outro módulo processador-cache pode monitorar o tráfego para a memória principal para manter a consistência dentro de sua própria cache. A principal desvantagem dessa técnica é que ela gera um tráfego de memória considerável e pode criar um gargalo. Uma técnica alternativa, conhecida como ***write back***, minimiza as escritas na memória. Com *write back*, as atualizações são feitas apenas na cache. Quando ocorre uma atualização, um **bit de modificação**, ou **bit de uso**, associado à linha, é marcado. Depois, quando um bloco é substituído, ele é escrito de volta na memória principal se, e somente se, o bit de modificação estiver marcado.

O problema com *write back* é que partes da memória principal podem ficar inválidas, e daí os acessos pelos módulos de E/S só podem ser permitidos pela cache. Isso exige circuitos complexos e gera um gargalo em potencial. A experiência tem mostrado que a porcentagem de referências à memória que são escritas está na ordem de 15% (SMITH, 1982). Porém, para aplicações de HPC, esse número pode se aproximar a 33% (multiplicação de vetores), e pode chegar a 50% (transposição de matrizes).

EXEMPLO 4.3

Considere uma cache com um tamanho de linha de 32 bytes e uma memória principal que requer 30 ns para transferir uma palavra de 4 bytes. Para qualquer linha que seja escrita pelo menos uma vez antes de ser retirada da cache, qual é o número médio de vezes que a linha precisa ser escrita antes de ser retirada para que uma cache *write back* seja mais eficiente do que uma cache *write through*?

Para o caso *write back*, cada linha modificada é escrita de volta uma vez, no momento da troca, usando $8 \times 30 = 240$ ns. Para o caso *write through*, cada atualização da linha requer que uma palavra seja escrita na memória principal, usando 30 ns. Portanto, se a linha que é escrita pelo menos uma vez for escrita mais de 8 vezes antes de ser trocada, então *write back* é mais eficiente.

Em uma organização de barramento em que mais de um dispositivo (em geral, um processador) tem uma cache e a memória principal é compartilhada, um novo problema é introduzido. Se os dados em uma cache forem alterados, isso invalida não apenas a palavra correspondente na memória principal, mas também essa mesma palavra em outras caches (se qualquer outra cache tiver essa mesma palavra). Mesmo que uma política *write through* seja usada, as outras caches podem conter dados inválidos. Diz-se que um sistema que impede esse problema mantém coerência de cache. Algumas das técnicas possíveis para a coerência de cache são:

- **Observação do barramento com *write through***: cada controlador de cache monitora as linhas de endereço para detectar as operações de escrita para a memória por outros mestres de barramento. Se outro mestre escrever em um local na memória compartilhada que também reside na memória cache, o controlador de cache invalida essa entrada da cache. Essa estratégia depende do uso de uma política *write through* por todos os controladores de cache.
- **Transparência do hardware**: um hardware adicional é usado para garantir que todas as atualizações na memória principal por meio da cache sejam refletidas em todas as caches. Assim, se um processador modificar uma palavra em sua cache, essa atualização é escrita na memória principal. Além disso, quaisquer palavras correspondentes em outras caches são atualizadas de maneira semelhante.
- **Memória não cacheável**: somente uma parte da memória principal é compartilhada por mais de um processador, e esta é designada como não cacheável. Neste tipo de sistema, todos os acessos à memória compartilhada são falhas de cache, pois a memória compartilhada nunca é copiada para a cache. A memória não mantida em cache pode ser identificada usando lógica de seleção de chip ou os bits mais significativos de endereço.

A coerência de cache é um campo de pesquisa atual. Esse assunto é explorado com mais detalhes na Parte V.

Tamanho da linha

Outro elemento de projeto é o tamanho da linha. Quando um bloco de dados é recuperado e colocado na cache, não apenas a palavra desejada, mas também algumas palavras adjacentes são armazenadas. À medida que o tamanho do bloco aumenta, de tamanhos muito pequenos para maiores, a razão de acerto a princípio aumentará por causa do princípio da localidade, que diz que os dados nas vizinhanças de uma palavra referenciada provavelmente serão referenciados no futuro próximo. À medida que o tamanho do bloco aumenta, dados mais úteis são trazidos para a cache. Contudo, a razão de acerto começará a diminuir enquanto o bloco se torna ainda maior e a probabilidade de uso da informação recém-trazida se torna menor que a probabilidade de reutilizar as informações que foram substituídas. Dois efeitos específicos entram em cena:

- Blocos maiores reduzem o número de blocos que cabem em uma cache. Como cada busca de bloco escreve sobre o conteúdo antigo da cache, um número pequeno de blocos resulta em dados sendo modificados pouco depois de serem buscados.
- À medida que o bloco se torna maior, cada palavra adicional fica mais distante da palavra solicitada e, portanto, tem menos probabilidade de ser necessária no futuro próximo.

O relacionamento entre o tamanho do bloco e a razão de acerto é complexo, dependendo das características de localidade de um programa em particular, e nenhum valor ideal definitivo foi encontrado. Um tamanho de 8 a 64 bytes parece ser razoavelmente próximo do ideal (SMITH, 1987, PRZYBYLSKI; HOROWITZ, HENNESSY, 1988, PRZYBYLSKI, 1990, HANDY, 1998). Para sistemas HPC, tamanhos de linha de cache com 64 e 128 bytes são usados com mais frequência.

Número de caches

Quando as memórias caches foram originalmente introduzidas, o sistema de memória típico tinha uma única cache. Mais recentemente, o uso de múltiplas caches tem se tornado comum. Dois aspectos de projeto dizem respeito ao número de níveis de memórias caches e ao uso de caches unificadas ou separadas.

CACHES MULTINÍVEL À medida que a densidade lógica aumenta, torna-se possível ter uma cache no mesmo chip que o processador: a cache no chip. Em comparação com uma cache conectada por meio de um barramento externo, a cache no chip reduz a atividade do barramento externo do processador e, portanto, agiliza o tempo de execução e aumenta o desempenho geral do sistema. Quando a instrução ou dados necessários são encontrados na cache no chip, não existe o acesso ao barramento. Como os caminhos de dados internos ao processador são curtos, em comparação com o tamanho do barramento, os acessos à cache no chip serão feitos mais rapidamente que os ciclos de barramento com estado *zero-wait* (tempo de espera nulo). Além do mais, durante esse período, o barramento estará livre para aceitar outras transferências.

A inclusão de uma cache no chip deixa aberta a questão de se uma cache fora do chip, ou externa, ainda é desejável. Em geral, a resposta é sim, e a maior parte dos projetos atuais inclui caches dentro e fora do chip. A organização mais simples desse tipo é conhecida como uma cache de dois níveis, com a cache interna designada como nível 1 (L1 – *level* 1) e a cache externa designada como nível 2 (L2). O motivo para incluir uma cache L2 é o seguinte: se não houver cache L2 e o processador fizer uma solicitação de acesso para um local de memória que não esteja na cache L1, então o processador precisa acessar a memória DRAM ou ROM pelo barramento. Em razão da baixa velocidade do barramento e do alto tempo de acesso à memória, tem-se um desempenho fraco. Por outro lado, se uma cache L2 SRAM (RAM estática) for usada, então normalmente a informação que falta pode ser recuperada rapidamente. Se a SRAM for rápida o suficiente para igualar sua velocidade à do barramento, então os dados podem ser acessados usando uma transação no estado *zero-wait*, o tipo mais rápido de transferência de barramento.

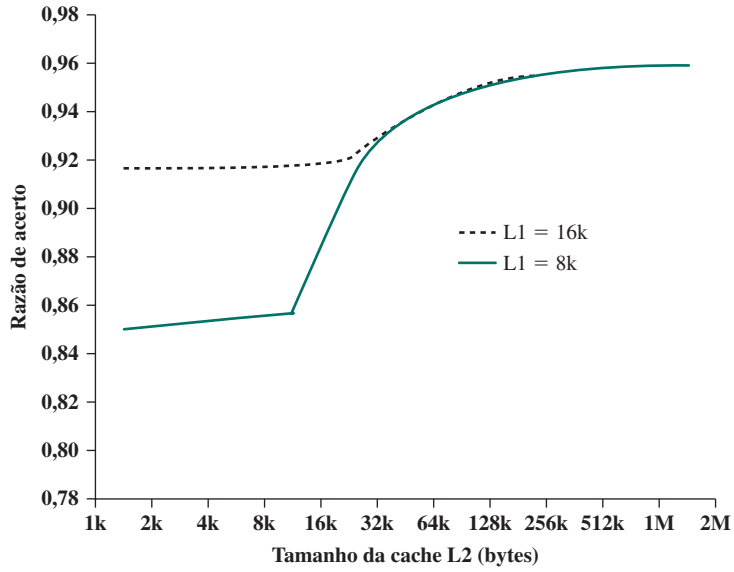
Dois recursos de projeto moderno de cache para caches multinível são dignos de nota. Primeiro, para uma cache L2 fora do chip, muitos projetos não usam o barramento do sistema como caminho para transferência entre a cache L2 e o processador, mas usam um caminho de dados separado, a fim de reduzir a carga sobre o barramento do sistema. Segundo, com o encolhimento contínuo dos componentes do processador, diversos processadores agora incorporam a cache L2 no chip do processador, melhorando o desempenho.

A economia em potencial em razão do uso de uma cache L2 depende das taxas de acerto nas caches L1 e L2. Vários estudos têm mostrado que, em geral, o uso de uma cache de segundo nível melhora o desempenho (por exemplo, ver AZIMI; PRASAD; BHAT, 1992, NOVITSKY; AZIMI; GHAZNAVI, 1993, HANDY, 1998). Porém, o uso de caches multinível complica todas as questões de projeto relacionadas a caches, incluindo tamanho, algoritmo de substituição e política de escrita; veja algumas discussões relacionadas a isso nas obras de Handy (1998) e Peir, Hsu e Smith (1999).

A Figura 4.17 mostra os resultados de um estudo de simulação de desempenho da cache de dois níveis em função do tamanho da cache (GENU, 2004). A figura pressupõe que as duas caches têm o mesmo tamanho de linha e mostra a razão de acerto total. Ou seja, um acerto é contado se os dados desejados aparecerem na cache L1 ou L2. A figura mostra o impacto da L2 sobre os acertos totais com relação ao tamanho da L1. L2 tem pouco efeito sobre o número total de acertos de cache até que seja pelo menos o dobro do tamanho da cache L1. Observe que a parte mais íngreme da inclinação para uma cache L1 de 8 kB é para uma cache

Figura 4.17

Razão de acerto total (L1 e L2) para L1 de 8 kB e 16 kB.



L2 de 16 kB. Novamente para uma cache L1 de 16 kB, a parte mais íngreme da curva é para um tamanho de cache L2 de 32 kB. Antes desse ponto, a cache L2 tem pouco ou nenhum impacto sobre o desempenho da cache total. A necessidade de a cache L2 ser maior que a cache L1 para afetar o desempenho faz sentido. Se a cache L2 tiver o mesmo tamanho de linha e capacidade da cache L1, seu conteúdo mais ou menos espelhará o da cache L1.

Com a disponibilidade cada vez maior de área no chip, a maior parte dos microprocessadores modernos passou a cache L2 para dentro do chip processador e acrescentou uma cache L3. Originalmente, a cache L3 era acessível pelo barramento externo. Mais recentemente, a maioria dos microprocessadores incorporou uma cache L3 no chip. De qualquer forma, parece haver uma vantagem no desempenho em acrescentar um terceiro nível (por exemplo, ver GHAI; JOYNER; JOHN, 1998). Além disso, sistemas maiores, como os sistemas mainframe da IBM zEnterprise, atualmente incorporam 3 níveis de cache no chip e um quarto nível de cache compartilhada por outros diversos chips (CURRAN, 2011).

CACHES UNIFICADAS *VERSUS* SEPARADAS Quando a cache no chip apareceu, inicialmente, muitos dos projetos consistiam em uma única cache usada para armazenar referências a dados e instruções. Mais recentemente, tornou-se comum dividir a cache em duas: uma dedicada a instruções e uma dedicada a dados. Essas duas caches existem no mesmo nível, normalmente como duas caches L1. Quando o processador tenta buscar uma instrução da memória principal, ele primeiro consulta a cache L1 de instrução, e quando o processador tenta buscar dados da memória principal, ele primeiro consulta a cache L1 de dados.

Existem duas vantagens em potencial de uma cache unificada:

- Para determinado tamanho de cache, uma cache unificada tem uma taxa de acerto mais alta que as caches divididas, pois ela equilibra automaticamente a carga entre buscas de instruções e dados. Ou seja, se um padrão de execução envolve muito mais buscas de instruções do que buscas de dados, então a cache tenderá a ser preenchida com instruções, e se um padrão de execução envolve relativamente mais buscas de dados, acontecerá o oposto.
- Somente uma cache precisa ser projetada e implementada.

A tendência é em direção a caches separadas no L1 e caches unificadas para altos níveis, particularmente para máquinas superescalares, que enfatizam a execução de instrução paralela e a pré-busca de instruções futuras previstas. A principal vantagem do projeto de cache separada é que isso elimina a disputa pela cache entre a unidade de busca/decodificação de instrução e a unidade de execução. Isso é importante em qualquer projeto que conta com o pipeline de instruções. Em geral, o processador buscará instruções antes da hora e preencherá um buffer, ou pipeline, com instruções a serem executadas. Suponha, agora, que tenhamos uma cache de instrução/dados unificada. Quando a unidade de execução realiza um acesso à memória para carregar e armazenar

dados, a solicitação é submetida à cache unificada. Se, ao mesmo tempo, o mecanismo de pré-busca de instrução emitir uma solicitação de leitura à cache para uma instrução, essa solicitação será temporariamente bloqueada para que a cache possa atender à unidade de execução primeiro, permitindo que ela complete a instrução atualmente em execução. Essa disputa pela cache pode diminuir o desempenho, interferindo com o uso eficiente da pipeline de instruções. A estrutura de cache separada contorna essa dificuldade.

4.4 ORGANIZAÇÃO DA CACHE DO PENTIUM 4

A evolução da organização da memória cache é vista claramente na evolução dos microprocessadores Intel (Tabela 4.4). O 80386 não inclui uma cache no chip. O 80486 inclui uma única cache de 8 kB no chip, usando um tamanho de linha de 16 bytes e uma organização associativa em conjunto com quatro linhas (4-*way*). Todos os processadores Pentium incluem duas caches L1 no chip, uma para dados e uma para instruções. Para o Pentium 4, a cache de dados L1 tem 16 kB, usando um tamanho de linha de 64 bytes e uma organização associativa em conjunto com quatro linhas. A cache de instruções do Pentium 4 é descrita mais adiante. O Pentium II também inclui uma cache L2 que alimenta ambas as caches L1. A cache L2 é associativa em conjunto com oito linhas por conjuntos, com um tamanho de 512 kB e um tamanho de linha de 128 bytes. Uma cache L3 foi acrescentada para o Pentium III, e passou a residir no chip com as versões avançadas do Pentium 4.

A Figura 4.18 contém uma visão simplificada da organização do Pentium 4, destacando o posicionamento das três caches. O *core* do processador consiste em quatro componentes principais:

Unidade de busca/decodificação: busca instruções do programa em ordem a partir da cache L2, decodifica-as para uma série de micro-operações e armazena os resultados na cache de instruções L1.

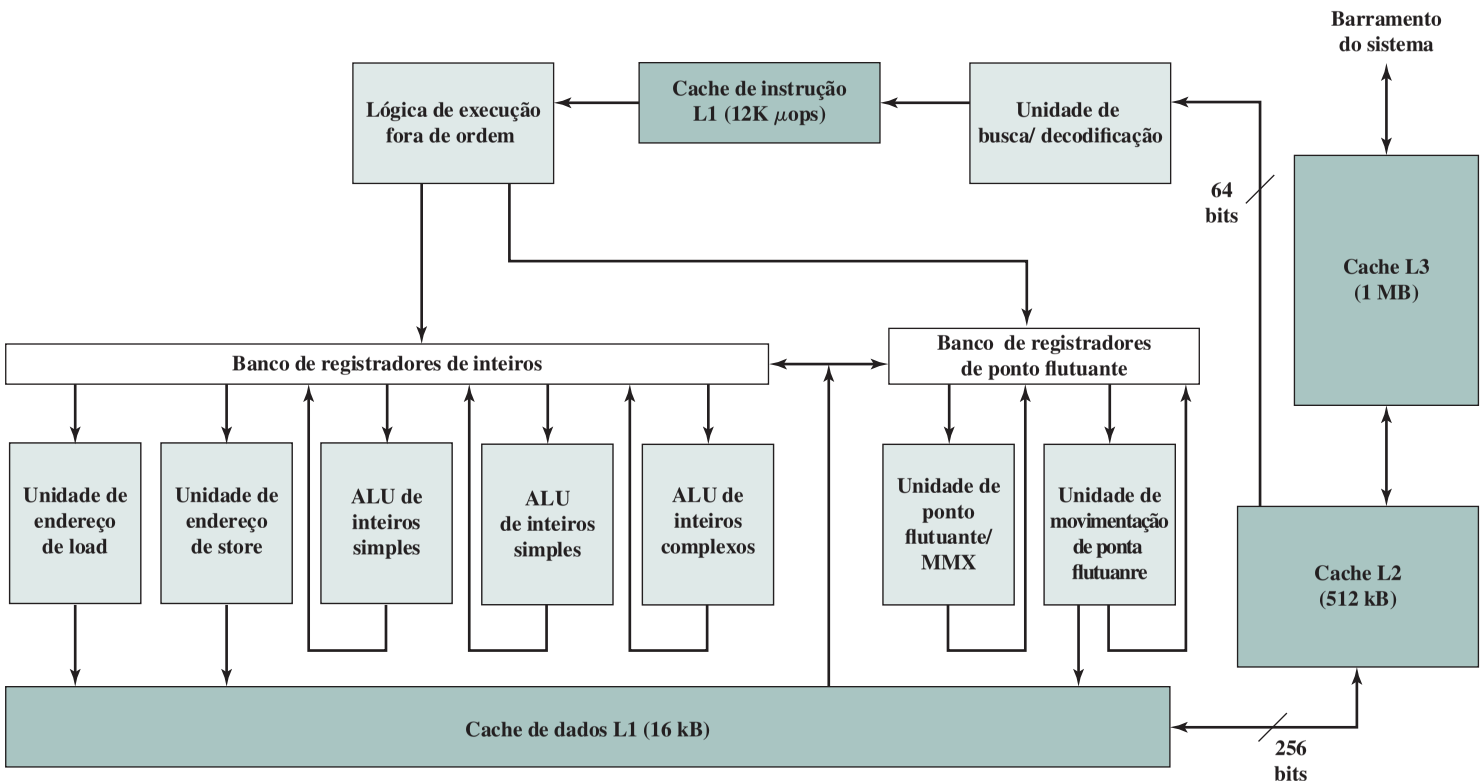
Tabela 4.4

Evolução de cache da Intel.

Problema	Solução	Processador em que o recurso apareceu inicialmente
Memória externa mais lenta que o barramento do sistema	Acrescentar cache externa usando tecnologia de memória mais rápida	386
Maior velocidade do processador torna o barramento externo um gargalo para o acesso à cache L2	Mover a cache externa para o chip, trabalhando na mesma velocidade do processador	486
Cache interna um tanto pequena, por conta do espaço limitado no chip	Acrescentar cache L2 externa usando tecnologia mais rápida que a memória principal	486
Quando ocorre uma disputa entre o mecanismo de pré-busca de instruções e a unidade de execução no acesso simultâneo à memória cache. Nesse caso, a busca antecipada é adiada até o término do acesso da unidade de execução aos dados	Criar caches separadas para dados e instruções	Pentium
Maior velocidade do processador torna o barramento externo um gargalo para o acesso à cache L2	Criar barramento *back-side* separado, que trabalha com velocidade mais alta que o barramento externo principal (*front-side*). O barramento *back-side* é dedicado à cache L2	Pentium Pro
	Mover cache L2 para o chip do processador	Pentium II
Algumas aplicações lidam com bancos de dados enormes, e precisam ter acesso rápido a grandes quantidades de dados. As caches no chip são muito pequenas	Acrescentar cache L3 externa	Pentium III
	Mover cache L3 para o chip	Pentium 4

Figura 4.18

Diagrama em bloco do Pentium 4.



Lógica de execução fora da ordem: escalona a execução das micro-operações, sujeita a dependências de dados e disponibilidade de recursos; assim, as micro-operações podem ser escalonadas para execução em uma ordem diferente daquela em que foram obtidas da sequência de instruções. Se o tempo permitir, essa unidade escalona a execução especulativa de micro-operações que podem ser solicitadas no futuro.

Unidades de execução: essas unidades executam micro-operações, buscando os dados solicitados da cache de dados L1 e armazenando os resultados temporariamente em registradores.

Subsistema de memória: essa unidade inclui as caches L2 e L3 e o barramento do sistema, que é usado para acessar a memória principal quando as caches L1 e L2 tiverem uma falta de cache e para acessar os recursos de E/S do sistema.

Diferente do que ocorre com a organização usada em todos os modelos Pentium anteriores, e na maioria dos outros processadores, a cache de instruções do Pentium 4 localiza-se entre a lógica de decodificação de instrução e o *core* de execução. O raciocínio por trás dessa decisão de projeto é o seguinte: conforme discutiremos com mais detalhes no Capítulo 16, o processo do Pentium decodifica, ou traduz, instruções de máquina do Pentium para instruções simples tipo RISC, chamadas de micro-operações. O uso de micro-operações simples, de tamanho fixo, permite o uso do pipeline superescalar e técnicas de escalonamento que melhoram o desempenho. Contudo, as instruções de máquina do Pentium são difíceis de decodificar; elas têm um número variável de bytes e muitas opções diferentes. Acontece que o desempenho é melhorado se essa decodificação for feita independentemente da lógica de escalonamento e do pipeline. Retornaremos a esse tópico no Capítulo 16.

A cache de dados emprega uma política *write back*: os dados são escritos na memória principal apenas quando são removidos da cache e quando houver uma atualização. O processador Pentium 4 pode ser configurado dinamicamente para aceitar a política *write through*.

A cache de dados L1 é controlada por dois bits em um dos registradores de controle, rotulados como bits CD (cache disable) e NW (*not write through*) (Tabela 4.5). Há também duas instruções do Pentium 4 que podem ser usadas para controlar a cache de dados: INVD invalida (esvazia) a memória cache interna e sinaliza a cache externa (se houver) para invalidar. WBINVD escreve de volta e invalida a cache interna e depois escreve de volta e invalida a cache externa.

As caches L2 e L3 são associativas em conjunto com oito linhas, com um tamanho de linha de 128 bytes.

Tabela 4.5

Modos de operação da cache do Pentium 4.

Bits de controle		Modo de operação		
CD	NW	Preenchimento da cache	*Write throughs*	Invalidado
0	0	Habilitado	Habilitado	Habilitado
1	0	Desabilitado	Habilitado	Habilitado
1	1	Desabilitado	Desabilitado	Desabilitado

Obs.: CD = 0; NW = 1 é uma combinação inválida.

4.5 TERMOS-CHAVE, QUESTÕES DE REVISÃO E PROBLEMAS

Acerto (*hit*), 103	Cache de instrução, 123	Cache separada, 123
Acerto (*hit*) de cache, 108	Cache física, 109	Cache unificada, 123
Acesso aleatório, 101	Cache L1, 105	Cache virtual, 109
Acesso direto, 101	Cache L2, 105	Computação de alto desempenho (HPC), 108
Acesso sequencial, 101	Cache L3, 105	Conjunto de cache, 116
Algoritmo de substituição, 120	Cache lógica, 109	Endereço físico, 109
Cache de dados, 126	Cache multinível, 122	Endereço virtual, 109

Falha (*miss*), 103	Localidade espacial, 134	Memória secundária, 104
Falha (*miss*) de cache, 108	Localidade temporal, 134	Razão de acerto, 114
Hierarquia de memória, 102	Mapeamento associativo, 114	Tag, 105
Linha, 105	Mapeamento associativo em conjunto, 116	Tempo de acesso, 101
Linha de cache, 105	Mapeamento direto, 111	*Write back,* 120
Localidade, 132	Memória cache, 105	*Write through,* 120

QUESTÕES DE REVISÃO

4.1. Quais são as diferenças entre acesso sequencial, acesso direto e acesso aleatório?

4.2. Qual é a relação entre tempo de acesso, custo de memória e capacidade?

4.3. Como o princípio de localidade se relaciona com o uso de múltiplos níveis de memória?

4.4. Quais são as diferenças entre mapeamento direto, mapeamento associativo e mapeamento associativo em conjunto?

4.5. Para uma cache mapeada diretamente, um endereço de memória principal é visto como consistindo em três campos. Liste e defina os três campos.

4.6. Para uma cache associativa, um endereço de memória principal é visto como consistindo em dois campos. Liste e defina os dois campos.

4.7. Para uma cache associativa em conjunto, um endereço da memória principal é visto como consistindo em três campos. Liste e defina os três campos.

4.8. Qual é a diferença entre localidade espacial e localidade temporal?

4.9. Em geral, quais são as estratégias para explorar a localidade espacial e a localidade temporal?

PROBLEMAS

4.1. Uma cache associativa em conjunto consiste em 64 linhas, ou slots, divididas em conjuntos de quatro linhas. A memória principal contém 4 K blocos de 128 palavras cada. Mostre o formato dos endereços da memória principal.

4.2. Uma cache associativa em conjunto com duas linhas por conjunto possui linhas de 16 bytes e um tamanho total de 8 kB. A memória principal de 64 MB é endereçável por byte. Mostre o formato dos endereços da memória principal.

4.3. Para os endereços hexadecimais da memória principal 111111, 666666, BBBBBB, mostre a seguinte informação, em formato hexadecimal:

- **c.** Valores de Tag, Linha e Palavra para uma cache de mapeamento direto, usando o formato da Figura 4.10.
- **d.** Valores de Tag e Palavra para uma cache associativa, usando o formato da Figura 4.12.
- **e.** Valores de Tag, Conjunto e Palavra para uma cache associativa em conjunto com duas linhas, usando o formato da Figura 4.15.

4.4. Liste os seguintes valores:

- **a.** Para o exemplo de cache direta da Figura 4.10: tamanho do endereço, número de unidades endereçáveis, tamanho de bloco, número de blocos na memória principal, número de linhas na cache e tamanho da tag.
- **b.** Para o exemplo de cache associativa da Figura 4.12: tamanho do endereço, número de unidades endereçáveis, tamanho de bloco, número de blocos na memória principal, número de linhas na cache e tamanho da tag.

c. Para o exemplo de cache associativa em conjunto com duas linhas por conjunto da Figura 4.15: tamanho do endereço, número de unidades endereçáveis, tamanho de bloco, número de blocos na memória principal, número de linhas no conjunto, número de conjuntos, número de linhas na cache e tamanho da tag.

4.5. Considere um microprocessador de 32 bits que tem uma cache associativa em conjunto de 16 kB no chip com quatro linhas por conjunto. Suponha que a cache tenha um tamanho de linha de quatro palavras de 32 bits. Desenhe um diagrama de blocos dessa cache, mostrando sua organização e como os diferentes campos de endereço são usados para determinar um acerto/falha de cache. Onde, na cache, a palavra no local de memória ABCDE8F8 é mapeada?

4.6. São dadas as seguintes especificações para uma memória cache externa: associativa em conjunto com quatro linhas; tamanho de linha de duas palavras de 16 bits; capaz de acomodar um total de 4 K palavras de 32 bits da memória principal; usada com um processador de 16 bits que emite endereços de 24 bits. Projete a estrutura de cache com todas as informações pertinentes e mostre como ela interpreta os endereços do processador.

4.7. O Intel 80486 tem uma cache unificada no chip. Ela contém 8 kB e tem uma organização associativa em conjunto com quatro linhas por conjunto e uma extensão de bloco de quatro palavras de 32 bits. A cache é organizada em 128 conjuntos. Existe um único "bit de validade de linha" e três bits, B0, B1 e B2 (os bits "LRU"), por linha. Em uma falha de cache, o 80486 lê uma linha de 16 bytes da memória principal em apenas uma leitura de memória pelo barramento. Desenhe um diagrama simplificado da cache e mostre como os diferentes campos do endereço são interpretados.

4.8. Considere uma máquina com uma memória principal endereçável por byte com 2^{16} bytes e um tamanho de bloco de 8 bytes. Suponha que uma cache mapeada diretamente, consistindo em 32 linhas, seja usada com essa máquina.

a. Como um endereço de memória de 16 bits é dividido em tag, número de linha e número de byte?

b. Em que linha seriam armazenados os bytes com cada um dos seguintes endereços?

0001	0001	0001	1011
1100	0011	0011	0100
1101	0000	0001	1101
1010	1010	1010	1010

c. Suponha que o byte com endereço 0001 1010 0001 1010 seja armazenado na cache. Quais são os endereços dos outros bytes armazenados junto com ele?

d. Quantos bytes de memória no total podem ser armazenados na cache?

e. Por que a tag também é armazenada na cache?

4.9. Para sua cache no chip, o Intel 80486 usa um algoritmo de substituição conhecido como ***pseudo-LRU* (*pseudo Least Recently Used*)**. Associados a cada um dos 128 conjuntos de quatro linhas (rotuladas como L0, L1, L2, L3), existem três bits, B0, B1 e B2. O algoritmo de substituição funciona da seguinte maneira: quando uma linha tiver de ser substituída, a cache primeiro determinará se o uso mais recente foi de L0 e L1 ou L2 e L3. Depois, a cache determinará qual do par de blocos foi usado menos recentemente e o marcará para substituição. A Figura 4.19 ilustra a lógica.

a. Especifique como os bits B0, B1 e B2 são definidos e depois descreva, em palavras, como eles são usados no algoritmo de substituição representado na Figura 4.19.

b. Mostre que o algoritmo do 80486 se aproxima de um algoritmo LRU verdadeiro. *Dica*: considere o caso em que a ordem de uso mais recente é L0, L2, L3, L1.

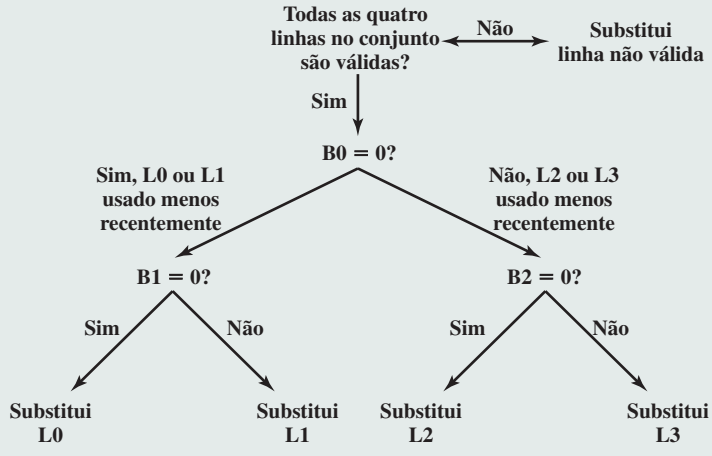
c. Demonstre que um algoritmo LRU verdadeiro exigiria 6 bits por conjunto.

4.10. Uma cache associativa em conjunto tem um tamanho de bloco de quatro palavras de 16 bits e um tamanho de conjunto de 2. A cache pode acomodar um total de 4.096 palavras. O tamanho da memória principal que pode ser mantido em cache é de 64K $\times$ 32 bits. Projete a estrutura da cache e mostre como os endereços do processador são interpretados.

4.11. Considere um sistema de memória que usa um endereço de 32 bits para endereçar em nível de byte, mais uma cache que usa um tamanho de linha de 64 bytes.

Figura 4.19

Estratégia de substituição de cache no chip do Intel 80486.



a. Considere uma cache mapeada diretamente com um campo de tag no endereço de 20 bits. Mostre o formato de endereço e determine os seguintes parâmetros: número de unidades endereçáveis, número de blocos na memória principal, número de linhas na cache e tamanho da tag.

b. Considere uma cache associativa. Mostre o formato de endereço e determine os seguintes parâmetros: número de unidades endereçáveis, número de blocos na memória principal, número de linhas na cache e tamanho da tag.

c. Considere uma cache associativa em conjunto com quatro linhas por conjunto, com um campo de tag no endereço de 9 bits. Mostre o formato de endereço e determine os seguintes parâmetros: número de unidades endereçáveis, número de blocos na memória principal, número de linhas no conjunto, número de conjuntos na cache, número de linhas na cache e tamanho da tag.

4.12. Considere um computador com as seguintes características: total de 1 MB de memória principal; tamanho de palavra de 1 byte; tamanho de bloco de 16 bytes; e tamanho de cache de 64 kB.

a. Para os endereços de memória principal F0010, 01234 e CABBE, indique os valores correspondentes ao tag, ao endereço de linha de cache e ao offset da palavra, para uma cache mapeada diretamente.

b. Indique dois endereços quaisquer da memória principal com diferentes tags que são mapeados para a mesma linha de cache para uma cache mapeada diretamente.

c. Para os endereços da memória principal F0010 e CABBE, indique os valores correspondentes de tag e offset para uma cache totalmente associativa.

d. Para os endereços da memória principal F0010 e CABBE, indique os valores correspondentes ao tag, ao conjunto de cache e ao offset, para uma cache associativa em conjunto com duas linhas.

4.13. Descreva uma técnica simples para implementar um algoritmo de substituição LRU em uma cache associativa em conjunto com quatro linhas por conjunto.

4.14. Considere novamente o Exemplo 4.3. Como a resposta mudaria se a memória principal usasse uma capacidade de transferência em bloco com um tempo de acesso da primeira palavra de 30 ns e um tempo de acesso de 5 ns para cada palavra subsequente?

4.15. Considere o código a seguir:

```
for (i = 0; i < 20; i++)
    for (j = 0; j < 10; j++)
        a[i] = a[i]*j
```

a. Dê um exemplo da localidade espacial no código.

b. Dê um exemplo da localidade temporal no código.

4.16. Generalize as equações 4.2 e 4.3, no Apêndice 4A, para hierarquias de memória de *N* níveis.

4.17. Um sistema de computação contém uma memória principal de 32 K palavras de 16 bits. Ele também tem uma cache de 4 K palavras, dividida em conjuntos de quatro linhas com 64 palavras por linha. Considere que a cache esteja inicialmente vazia.

O processador busca palavras das localizações 0, 1, 2, ... , 4351, nessa ordem. Depois, ele repete essa sequência de busca mais nove vezes. A cache é 10 vezes mais rápida que a memória principal. Estime a melhoria resultante do uso da cache. Considere uma política de LRU para a substituição em bloco.

4.18. Considere uma cache de 4 linhas de 16 bytes cada. A memória principal é dividida em blocos de 16 bytes cada. Ou seja, o bloco 0 tem bytes com endereços de 0 a 15, e assim por diante. Agora, considere um programa que acessa a memória na seguinte sequência de endereços:

Uma vez: 63 até 70.

Loop dez vezes: 15 até 32; 80 até 95.

a. Suponha que a cache seja organizada como mapeada diretamente. Os blocos de memória 0, 4 e assim por diante são atribuídos à linha 1; os blocos 1, 5 e assim por diante à linha 2; e assim sucessivamente. Calcule a razão de acerto.

b. Suponha que a cache seja organizada como associativa em conjunto com duas linhas por conjunto, com dois conjuntos de duas linhas cada. Os blocos de numeração par são atribuídos a um conjunto 0, e os blocos de numeração ímpar são atribuídos ao conjunto 1. Calcule a razão de acerto para a cache associativa em conjunto com duas linhas usando o esquema de substituição LRU ("usado menos recentemente").

4.19. Considere um sistema de memória com os seguintes parâmetros:

$$T_c = 100 \text{ ns} \qquad C_c = 10^{-4} \text{ \$/bit}$$
$$T_m = 1.200 \text{ ns} \qquad C_m = 10^{-5} \text{ \$/bit}$$

a. Qual é o custo de 1 MB de memória principal?

b. Qual é o custo de 1 MB de memória principal usando a tecnologia de memória cache?

c. Se o tempo de acesso efetivo for 10% maior que o tempo de acesso à cache, qual será a razão de acerto H?

4.20. **a.** Considere uma cache L1 com um tempo de acesso de 1 ns e uma razão de acerto de $H = 0{,}95$. Suponha que possamos mudar o projeto da cache (tamanho da cache, organização) de modo que aumentem o H para 0,97, mas aumentando o tempo de acesso para 1,5 ns. Quais condições deverão ser atendidas para que essa mudança resulte em um desempenho melhorado?

b. Explique por que esse resultado faz sentido intuitivamente.

4.21. Considere uma cache de um único nível com um tempo de acesso de 2,5 ns, um tamanho de linha de 64 bytes e uma razão de $H = 0{,}95$. A memória principal tem uma capacidade de transferência em bloco com um tempo de acesso da primeira palavra (4 bytes) de 50 ns e um tempo de acesso de 5 ns para cada palavra subsequente.

a. Qual é o tempo de acesso quando existe uma falha de cache? Suponha que a cache espere até que a linha tenha sido buscada da memória principal e depois reexecute para um acerto.

b. Suponha que aumentar o tamanho da linha para 128 bytes aumente o H para 0,97. Isso reduz o tempo de acesso médio à memória?

4.22. Um computador tem uma cache, uma memória principal e um disco usado para memória virtual. Se uma palavra referenciada estiver na cache, 20 ns serão necessários para acessá-la. Se estiver na memória principal, mas não na cache, 60 ns serão necessários para carregá-la para a cache, e depois a referência é iniciada novamente. Se a palavra não estiver na memória principal, 12 ms serão necessários para buscar a palavra do disco, seguidos por 60 ns para copiá-la para a cache e depois a referência é iniciada novamente. A razão de acerto da cache é 0,9 e a razão de acerto da memória principal é 0,6. Qual é o tempo médio em nanossegundos necessário para acessar uma determinada palavra nesse sistema?

4.23. Considere uma cache com um tamanho de linha de 64 bytes. Suponha que, na média, 30% das linhas na cache estejam modificadas. Uma palavra consiste em 8 bytes.

a. Suponha que haja uma taxa de falha de 3% (razão de acerto de 0,97). Calcule a quantidade de tráfego da memória principal, em termos de bytes por instrução para políticas *write through* e *write back*. A memória é lida na cache uma linha por vez. Porém, para *write back*, uma única palavra pode ser escrita da cache para a memória principal.

b. Repita a parte (a) para uma taxa de 5%.

c. Repita a parte (a) para uma taxa de 7%.

d. Com esses resultados, a que conclusão você pode chegar?

4.24. No microprocessador Motorola 68020, um acesso à cache leva dois ciclos de clock. O acesso a dados da memória principal pelo barramento até o processador leva três ciclos de clock no caso de nenhuma inserção de estado de espera; os dados são entregues ao processador em paralelo com a entrega à cache.

a. Calcule o tamanho efetivo de um ciclo de memória dada uma razão de acerto de 0,9 e uma frequência de clock de 16,67 MHz.

b. Repita os cálculos considerando a inserção de dois estados de espera de um ciclo cada por ciclo de memória. Com esses resultados, a que conclusão você pode chegar?

4.25. Considere um processador que possui um tempo de ciclo de memória de 300 ns e uma taxa de processamento de instrução de 1 MIPS. Na média, cada instrução requer um ciclo de memória do barramento para busca de instrução e um para o operando envolvido.

a. Calcule a utilização do barramento pelo processador.

b. Suponha que o processador esteja equipado com uma cache de instruções e a razão de acerto associada seja 0,5. Determine o impacto sobre a utilização do barramento.

4.26. O desempenho de um sistema de cache de um único nível para uma operação de leitura pode ser caracterizado pela seguinte equação:

$$T_a = T_c + (1 - H)T_m$$

em que T_a é o tempo médio de acesso, T_c é o tempo de acesso à cache, T_m é o tempo de acesso à memória (memória ao registrador do processador) e H é a razão de acerto. Para simplificar, consideramos que a palavra em questão é carregada na cache em paralelo com o carregamento para o registrador do processador. Essa é a mesma forma da Equação 4.2.

a. Defina T_b = tempo para transferir uma linha entre a cache e a memória principal, e W = fração de referências de escrita. Revise a equação anterior para considerar as escritas e também as leituras, usando uma política *write through.*

b. Defina W_b como a probabilidade de que uma linha na cache tenha sido alterada. Ofereça uma equação para T_a para a política *write back*.

4.27. Para um sistema com dois níveis de cache, defina T_{c_1} = tempo de acesso da cache de primeiro nível; T_{c_2} = tempo de acesso da cache de segundo nível; T_m = tempo de acesso à memória; H_1 = razão de acerto da cache de primeiro nível; H_2 = razão de acerto da cache de primeiro e segundo níveis combinadas. Ofereça uma equação para T_a para uma operação de leitura.

4.28. Considere as seguintes características de desempenho em uma falha de leitura de cache: um ciclo de clock para enviar um endereço à memória principal e quatro ciclos de clock para acessar uma palavra de 32 bits da memória principal e transferi-la para o processador e a cache.

a. Se o tamanho da linha de cache for uma palavra, qual é a penalidade de falha (ou seja, o tempo adicional exigido para uma leitura no evento de uma falha de leitura)?

b. Qual é a penalidade de falha se um tamanho de linha de cache tiver quatro palavras e for executada uma transferência múltipla, não repetitivamente (*burst mode*)?

c. Qual é a penalidade de falha se o tamanho da linha de cache for quatro palavras e uma transferência for executada com um ciclo de clock por transferência de palavra?

4.29. Para o projeto de cache do problema anterior, suponha que aumentar o tamanho da linha de uma palavra para quatro palavras resulta em uma diminuição da taxa de falha de leitura de 3,2% para 1,1%. Para os casos de transferência sem repetição e com repetição, qual é a penalidade média de falha, considerando todas as leituras, para os dois tamanhos de linha diferentes?

APÊNDICE 4A CARACTERÍSTICAS DE DESEMPENHO DE MEMÓRIAS DE DOIS NÍVEIS

Neste capítulo, é feita uma referência a uma cache que atua como um buffer entre a memória principal e o processador, criando uma memória interna de dois níveis. Essa arquitetura de dois níveis explora uma propriedade conhecida como localidade, para oferecer melhor desempenho comparada a uma memória de um nível.

O mecanismo de cache da memória principal faz parte da arquitetura do computador, implementada no hardware e normalmente invisível ao sistema operacional. Existem dois outros casos de uma técnica de memória de dois níveis que também exploram a localidade e que são, pelo menos parcialmente, implementadas no sistema operacional: memória virtual e a cache de disco (Tabela 4.6). A memória virtual é explorada no Capítulo 8; a cache de disco está fora do escopo deste livro, mas é examinada em outra obra deste autor (STALLINGS, 2015). Neste apêndice, examinamos algumas das características de desempenho das memórias de dois níveis que são comuns às três técnicas.

Localidade

A base para a vantagem de desempenho de uma memória de dois níveis é um princípio conhecido como **localidade de referência** (DENNING, 1968). Esse princípio afirma que as referências à memória tendem a se agrupar. Por um longo período, os agrupamentos em uso mudam, mas, por um período curto, o processador está trabalhando principalmente com grupos fixos de referências à memória.

Intuitivamente, o princípio de localidade faz sentido. Considere a seguinte linha de raciocínio:

1. Com exceção das instruções de desvio e chamada, que constituem apenas uma pequena fração de todas as instruções do programa, a execução do programa é sequencial. Logo, na maior parte dos casos, a próxima instrução a ser buscada vem imediatamente após a última instrução buscada.
2. É raro ter uma longa sequência ininterrupta de chamadas de procedimento seguidas pela sequência de retornos correspondente. Em vez disso, um programa permanece confinado a uma janela de procedimento de profundidade um tanto pequena. Assim, por um curto período, as referências às instruções tendem a estar localizadas em alguns poucos procedimentos.
3. A maioria das construções iterativas consiste em um número relativamente pequeno de instruções repetidas muitas vezes. Pela duração da iteração, o cálculo é, portanto, confinado a uma pequena parte contígua de um programa.
4. Em muitos programas, grande parte do cálculo envolve processamento de estruturas de dados, como arrays ou sequências de registros. Em muitos casos, referências sucessivas a essas estruturas de dados serão localizadas em itens de dados próximos.

Tabela 4.6

Características das memórias de dois níveis.

	Cache de memória principal	Memória virtual (paginação)	Cache de disco
Razões típicas de tempo de acesso	5:1 (memória principal *vs.* cache)	10^6:1 (memória principal *vs.* disco)	10^6:1 (memória principal *vs.* disco)
Sistema de gerenciamento de memória	Implementada por hardware especial	Combinação de hardware e software de sistema	Software de sistema
Tamanhos típicos de bloco ou página	4 a 128 bytes (bloco de cache)	64 a 4096 bytes (página de memória virtual)	64 a 4096 bytes (bloco ou páginas de disco)
Acesso do processador ao segundo nível	Acesso direto	Acesso indireto	Acesso indireto

Essa linha de raciocínio tem sido confirmada em muitos estudos. Com referência ao ponto 1, diversos estudos têm analisado o comportamento dos programas em linguagem de alto nível. A Tabela 4.7 inclui os principais resultados, medindo o surgimento de vários tipos de instruções durante a execução, a partir dos estudos a seguir. O estudo mais antigo do comportamento da linguagem de programação, realizado por Knuth (1971), examinou uma coleção de programas FORTRAN usados como exercícios para alunos. Tanenbaum (1978) publicou medidas coletadas de mais de 300 procedimentos usados em sistemas operacionais e escritos em uma linguagem que aceita programação estruturada (SAL). Patterson e Sequin (1982a) analisaram um conjunto de medições tomadas de compiladores e programas para editoração, projeto auxiliado por computador (CAD), classificação e comparação de arquivos. As linguagens de programação C e Pascal foram estudadas. Huck (1983) analisou quatro programas que representam uma mistura de computação científica de uso geral, incluindo transformação rápida de Fourier e a integração de sistemas de equações diferenciais. Existe um acordo tão bom sobre os resultados dessa mistura de linguagens e aplicações que as instruções de desvio e chamada representam apenas uma fração das instruções executadas durante o tempo de vida de um programa. Assim, esses estudos confirmam a afirmação 1.

Com relação à afirmação 2, os estudos relatados por Patterson (1985a) confirmam. Isso é ilustrado na Figura 4.20, que mostra o comportamento de chamada-retorno. Cada chamada é representada pela linha descendo para a direita, e cada retorno, pela linha subindo para a direita. Na figura, uma *janela* com profundidade igual a 5 é definida. Somente uma sequência de chamadas e retornos com um movimento de profundidade 6 em qualquer direção faz com que a janela se mova. Como podemos ver, o programa em execução pode permanecer dentro de uma janela estacionária por longos períodos. Um estudo feito pelos mesmos analistas de programas C e Pascal mostrou que uma janela de profundidade 8 precisará se deslocar apenas em menos de 1% das chamadas ou retornos (TAMIR; SEQUIN, 1983).

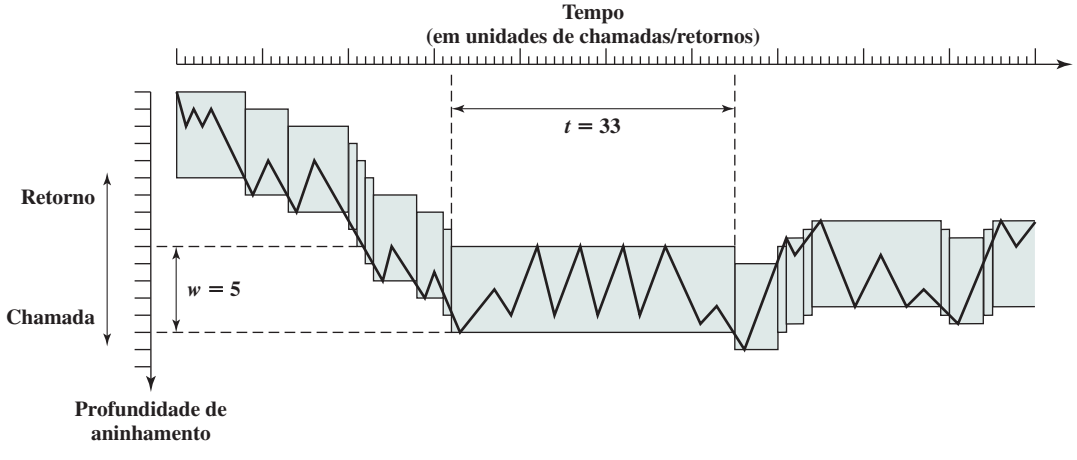
Tabela 4.7

Frequência dinâmica relativa das operações em linguagens de alto nível.

Carga de trabalho da linguagem em estudo	HUCKI, 1983 Pascal Científico	KNUTH, 1971 FORTRAN Estudante	PATTERSON; SEQUIN, 1982a		TANENBAUM, 1978 Sistema SAL
			Sistema Pascal	Sistema C	
Assinalamento	74	67	45	38	42
Loop	4	3	5	3	4
Call	1	3	15	12	12
IF	20	11	29	43	36
GOTO	2	9	—	3	—
Outras	—	7	6	1	6

Figura 4.20

Exemplo de comportamento de chamada-retorno de um programa.



A literatura faz uma distinção entre localidade espacial e localidade temporal. A **localidade espacial** refere-se à tendência da execução de envolver uma série de locais de memória que estão agrupados. Isso reflete a tendência de um processador de acessar as instruções sequencialmente. A localidade espacial também reflete a tendência de um programa de acessar locais de dados sequencialmente, como ao processar uma tabela de dados. A **localização temporal** refere-se à tendência de um processador de acessar locais de memória que foram usados recentemente. Por exemplo, quando um loop é executado, o processador executa o mesmo conjunto de instruções repetidamente.

Tradicionalmente, a localidade temporal é explorada mantendo valores de dados e instruções usados recentemente na memória cache e explorando uma hierarquia de cache. A localidade espacial geralmente é explorada usando blocos de cache maiores e incorporando mecanismos de pré-busca (buscando itens de uso antecipado) na lógica de controle da cache. Recentemente, tem havido uma pesquisa considerável sobre o refinamento dessas técnicas para alcançar maior desempenho, mas as estratégias básicas continuam sendo as mesmas.

Operação da memória de dois níveis

A propriedade de localidade pode ser explorada na formação de uma memória de dois níveis. A memória de nível superior (M1) é menor, mais rápida e mais cara (por bit) do que a memória de nível inferior (M2). M1 é usada como um armazenamento temporário para parte do conteúdo da M2 maior. Quando é feita uma referência à memória, é feita uma tentativa de acessar o item em M1. Se isso tiver sucesso, então é feito um acesso rápido. Se não, então um bloco de locais de memória é copiado de M2 para M1, e o acesso então ocorre através de M1. Em razão da localidade, quando um bloco é trazido para M1, deve haver uma série de acessos a locais nesse bloco, resultando em um atendimento geral mais rápido.

Para expressar o tempo médio para acessar um item, temos de considerar não apenas as velocidades dos dois níveis de memória, mas também a probabilidade de que determinada referência possa ser encontrada em M1. Temos:

$$T_S = H \times T_1 + (1 - H) \times (T_1 + T_2) = T_1 + (1 - H) \times T_2 \quad \textbf{(4.2)}$$

em que

T_s = tempo médio de acesso (sistema)
T_1 = tempo de acesso de M1 (por exemplo, *cache*, *cache de disco*)
T_2 = tempo de acesso de M2 (por exemplo, *memória principal*, *disco*)
H = razão de acerto (*hit ratio* — a fração de tempo da referência encontrada em M1)

A Figura 4.2 mostra o tempo médio de acesso como uma função da razão de acerto. Como podemos ver, para uma alta porcentagem de acertos, o tempo médio de acesso total é muito mais próximo ao do M1 do que do M2.

Desempenho

Vejamos alguns dos parâmetros relevantes a uma avaliação de um mecanismo de memória de dois níveis. Primeiro, considere o custo. Temos:

$$C_s = \frac{C_1 S_1 + C_2 S_2}{S_1 + S_2} \quad \textbf{(4.3)}$$

em que

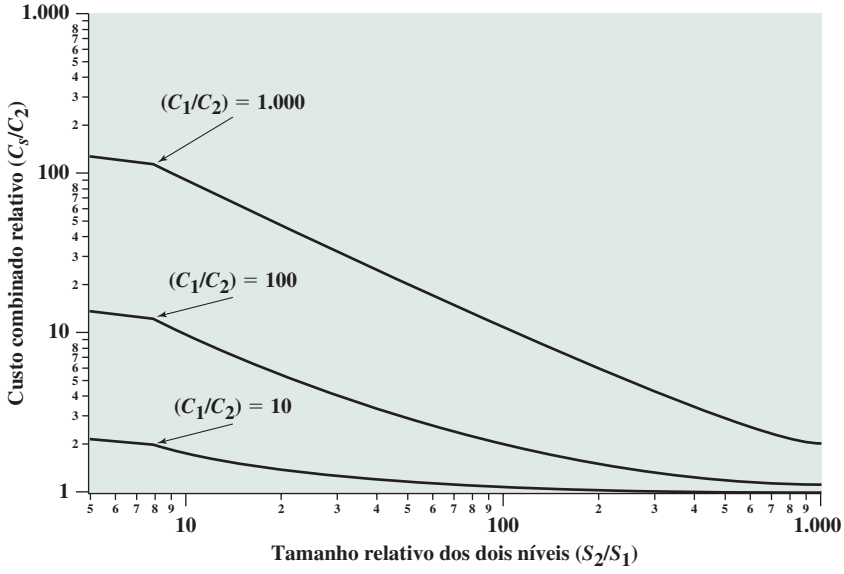
C_s = custo médio por bit para a memória de dois níveis combinados
C_1 = custo médio por bit da memória de nível superior M1
C_2 = custo médio por bit da memória de nível inferior M2
S_1 = tamanho de M1
S_2 = tamanho de M2

Gostaríamos que $C_s \approx C_2$. Dado que $C_1 >> C_2$, isso exige $S_1 < S_2$. A Figura 4.21 mostra essa relação.

Em seguida, considere o tempo de acesso. Para que uma memória de dois níveis ofereça uma melhoria de desempenho significativa, precisamos ter T_s aproximadamente igual a $T_1 (T_s \approx T_1)$. Dado que T_1 é muito menor que $T_2 (T_1 << T_2)$, uma razão de acerto próxima de 1 é necessária.

Figura 4.21

Relação do custo médio de memória com tamanho relativo de memória para uma memória de dois níveis.



Assim, gostaríamos que M1 fosse pequena para reduzir o custo, e grande para melhorar a razão de acerto e, portanto, o desempenho. Existe um tamanho de M1 que satisfaça os dois requisitos de um modo razoável? Podemos responder a essa questão com uma série de subquestões:

- Que valor de razão de acerto é necessário para que $T_s \approx T_1$?
- Que tamanho de M1 garantirá a razão de acerto necessária?
- Esse tamanho satisfaz o requisito de custo?

Para conseguir isso, considere a quantidade T_1/T_s, que é conhecida como *eficiência de acesso*. Essa é uma medida da proximidade entre o tempo de acesso médio (T_s) e o tempo de acesso de M1 (T_1). Pela Equação (4.2),

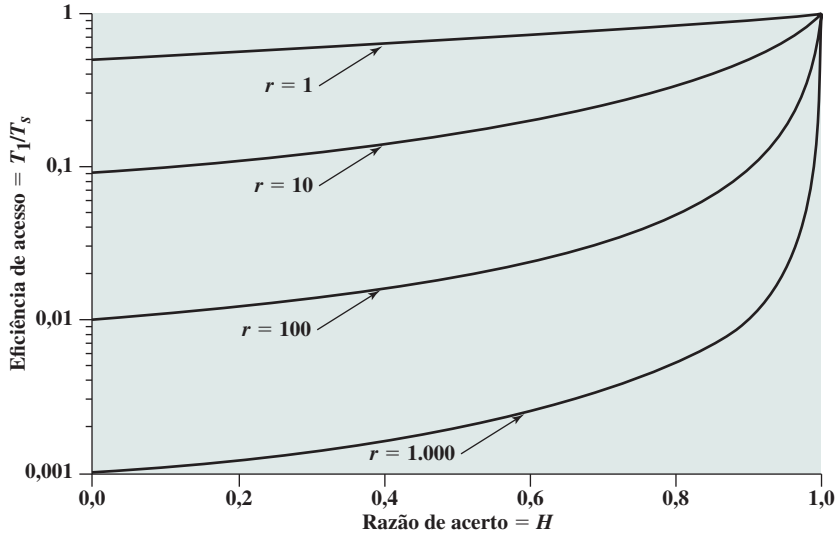
$$\frac{T_1}{T_s} = \frac{1}{1 + (1 - H)\dfrac{T_2}{T_1}} \tag{4.4}$$

A Figura 4.22 representa T_1/T_s como uma função da razão de acerto H, com a quantidade T_2/T_1 como um parâmetro. Em geral, o tempo de acesso à cache no chip é de cerca de 25 a 50 vezes mais rápido que o tempo de acesso à memória principal (ou seja T_2/T_1 é 25 a 50), o tempo de acesso à cache fora do chip é cerca de 5 a 15 vezes mais rápido que o tempo de acesso à memória principal (ou seja, T_2/T_1 é 5 a 15), e o tempo de acesso à memória principal é cerca de 1.000 vezes mais rápido que o tempo de acesso ao disco $T_2/T_1 = 1.000$). Assim, uma razão de acerto na faixa de algo em torno de 0,9 parece ser necessária para satisfazer o requisito de desempenho.

Agora, podemos formular a questão sobre o tamanho relativo da memória com mais exatidão. Uma razão de acerto de, digamos, 0,8 ou melhor, é razoável para $S_1 << S_2$? Isso dependerá de diversos fatores, incluindo a natureza do software sendo executado e dos detalhes do projeto da memória de dois níveis. O principal determinante, logicamente, é o grau de localidade. A Figura 4.24 sugere o efeito que a localidade tem sobre a razão de acerto. Claramente, se M1 tiver o mesmo tamanho que M2, então a razão de acerto será 1,0: todos os itens em M2 sempre são armazenados também em M1. Agora, suponha que não haja localidade, ou seja, as referências são completamente aleatórias. Neste caso, a razão de acerto deverá ser uma função estritamente linear do tamanho relativo da memória. Por exemplo, se M1 tiver a metade do tamanho de M2, então, a qualquer momento, metade dos itens de M2 também estarão em M1, e a razão de acerto será 0,5. Na prática, porém, existe algum grau de localidade nas referências. Os efeitos da localidade moderada e forte são indicados na figura. Observe que a Figura 4.23 não é derivada de qualquer dado ou modelo específico; a figura sugere o tipo de desempenho que é visto com diversos graus de localidade.

Figura 4.22

Eficiência de acesso como uma função da taxa de acerto ($r = T_2/T_1$).



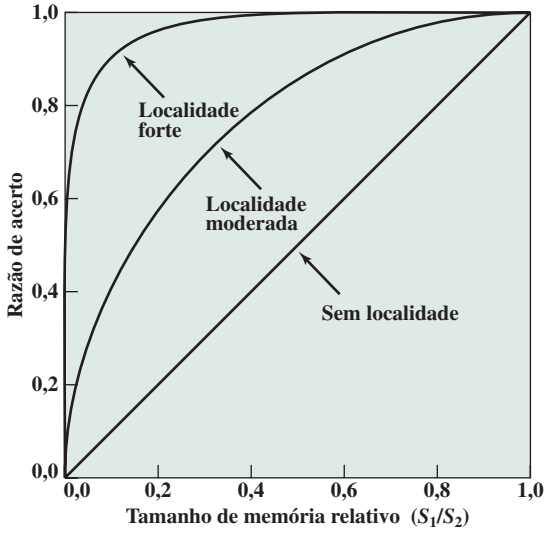
Assim, se houver localidade forte, é possível alcançar altos valores de razão de acerto, mesmo com um tamanho de memória de nível superior relativamente pequeno. Por exemplo, diversos estudos têm mostrado que tamanhos de cache pequenos gerarão uma razão de acerto acima de 0,75, *independentemente do tamanho da memória principal* (por exemplo, AGARWAL, 1989, PRZYBYLSKI; HOROWITZ; HENNESSY, 1988, STRECKER, 1983 e SMITH, 1982). Uma cache na faixa de 1 K a 128 K palavras em geral é adequada, enquanto a memória principal agora normalmente está na faixa dos gigabytes. Quando considerarmos a memória virtual e a cache de disco, citaremos outros estudos que confirmam o mesmo fenômeno, sobretudo que um M1 relativamente pequeno gera um valor alto de razão de acerto, por conta da localidade.

Isso nos leva à última pergunta listada anteriormente: o tamanho relativo das duas memórias satisfaz o requisito de custo? A resposta é clara: sim. Se só precisarmos de uma memória de nível superior relativamente pequena para conseguir um bom desempenho, então o custo médio por bit dos dois níveis de memória se aproximará do menor custo da memória de nível inferior.

Por favor, observe que, com o envolvimento de cache L2, ou, ainda, de caches L2 e L3, a análise é muito mais complexa. Consulte as obras de Peir, Hsu e Smith (1999) e Handy (1998) para mais discussões a esse respeito.

Figura 4.23

Razão de acerto como uma função do tamanho de memória relativo.



5

MEMÓRIA INTERNA

OBJETIVOS DE APRENDIZAGEM

Após ler este capítulo, você será capaz de:

- Apresentar uma visão geral dos tipos fundamentais de memória principal semicondutora.
- Entender a operação de um código básico que pode detectar e corrigir erros de bit único em palavras de 8 bits.
- Resumir as propriedades das organizações modernas DDR-DRAM.
- Entender a diferença entre memória flash NOR e NAND.
- Apresentar uma visão geral das novas tecnologias de memórias de estado sólido não volátil.