

5

Memória Cache

5.1 INTRODUÇÃO

No item 4.2.2 foi apresentada uma breve explicação sobre o conceito de memória cache, bem como dados de seu desempenho e características de modo a situá-la adequadamente na estrutura piramidal de uma hierarquia de memória.

Neste capítulo serão apresentados, com mais detalhes, conceitos e técnicas que permitiram o desenvolvimento, o projeto, o funcionamento e o uso de memórias cache em sistemas de computação atuais.

Inicialmente será apresentado o motivo impulsionador da existência das memórias cache: o *gap* de velocidade entre memória principal e processador; devido à necessidade de redução desse *gap*, os pesquisadores concluíram, de vários estudos de comportamento dos programas, pela existência de um princípio originado no modo pelo qual os programas em geral são executados, denominado princípio da localidade, e que esse princípio é importante para a inserção da memória cache na hierarquia de memória dos sistemas de computação. Em seguida, será mostrada a organização básica de uma memória cache e, finalmente, serão apresentados e analisados diversos itens que impactam o projeto e construção de memórias cache.

5.2 CONCEITUAÇÃO

Para que seja possível entender perfeitamente o sentido da criação e do desenvolvimento das memórias cache e de sua crescente e permanente utilização nos sistemas de computação, deve-se, primeiramente, analisar dois aspectos relacionados com o funcionamento e a utilidade dessas memórias: um deles refere-se à diferença de velocidade processador/MP, e o outro, ao conceito de localidade, sendo o primeiro o motivador para se ter chegado ao último.

5.2.1 Diferença de Velocidade Processador/MP

Nesse caso, trata-se de uma constatação inevitável. Parte do problema de limitação de desempenho dos processadores, que qualquer projetista de sistemas de computação enfrenta, refere-se à diferença de velocidade entre o ciclo de tempo do processador e o ciclo de tempo da memória principal. Ou seja, a MP transfere bits para o processador em velocidades sempre inferiores às que o processador pode receber e operar os dados, o que acarreta, muitas vezes, a necessidade de acrescentar-se um tempo de espera para o processador (*wait state* - estado de espera). Mesmo atualmente existindo memórias DRAM síncronas, que eliminam o estado de espera, a diferença de velocidade processador/memória principal permanece grande. A Fig. 5.1 mostra um exemplo da diferença de velocidade processador/MP.

Se todos os circuitos do processador e da MP fossem fabricados com elementos de mesma tecnologia, este problema deixaria de existir e não estaríamos aqui explicando o que é e para que serve uma memória cache.

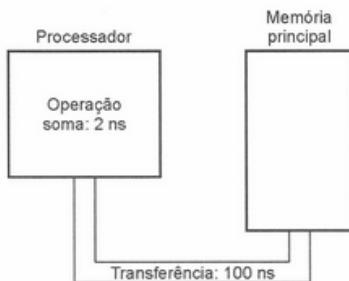


Figura 5.1 Exemplo de diferença de velocidade P/MP. Enquanto o processador gasta 2 ns adicionando dois dados a MP gasta 100 ns transferindo os dados para o processador.

O problema de diferença de velocidade se torna difícil de solucionar apenas com melhorias no desempenho das MP, devido a fatores de custo e tecnologia. Já foi anteriormente mencionado que, enquanto o desempenho dos microprocessadores, por exemplo, vem dobrando a cada 18 a 24 meses, o mesmo não acontece com a velocidade de transferência (tempo de acesso) das memórias DRAM (RAM dinâmicas), largamente utilizadas como MP, que vem aumentando relativamente pouco de ano para ano (cerca de 10%).

Apesar de a tecnologia para aumentar a velocidade das MP ser bem conhecida, fazer isso neste momento penalizaria consideravelmente o sistema em custo, pois memórias rápidas são muito caras, como é o caso das memórias SRAM.

Embora correndo o risco de redundância, é bom enfatizar, então, que, apesar de todos os avanços na tecnologia de construção das memórias DRAM, passando das obsoletas FPM, EDO etc. para as atuais DDR, DDR2 e Rambus, elas continuam a ser constituídas de um capacitor/transistor por bit e, portanto, requerem recarregamento. Naturalmente, são consideravelmente mais rápidas que as antecessoras, mas ainda mais lentas que as memórias SRAM e mais lentas ainda que os tempos de transferência interna dos processadores (p.ex., entre registradores).

Esse problema permanente, existente desde o surgimento dos computadores, foi-se agravando com a possibilidade de uso concorrente de vários programas (multiprogramação), quando a manutenção da ocupação do processador é essencial para o aumento de desempenho do sistema como um todo e a velocidade de transferência da MP tem papel relevante.

Na década de 1960, então, com o propósito de encontrar uma solução para este problema (eles poderiam ter vários objetivos em mente, mas o problema do *gap* de velocidade processador/MP era um deles) diversos pesquisadores, notadamente na IBM, analisaram de forma extensiva o comportamento dos processos (programas em execução), e de suas conclusões a respeito surgiu um princípio de funcionamento dos programas, denominado genericamente princípio da localidade (*locality of reference* ou *principle of locality*).

5.2.2 Conceito de Localidade

Se observarmos um programa de computador de forma ampla e genérica, podemos observar que:

- Considerando que o programa já esteja em sua forma executável (já sofreu o processo de compilação e está transformado em instruções de máquina binárias), ele tem suas instruções ordenadas seqüencialmente, de acordo com o algoritmo desenvolvido pelo programador. Nesse instante, o programa nada mais é do que um conjunto de linhas de instruções que são armazenadas seqüencialmente na memória (em um endereço após o outro).
- Quando o programa é colocado em execução (se transforma em um ou mais processos), as instruções vão sendo buscadas pelo processador na MP para interpretação e execução (ver Cap. 6) e, naturalmente, os endereços vão se sucedendo no CI (ver descrição e operação do contador de instrução no Cap. 6) seqüencialmente, exceto quando ocorre algum loop ou comando de desvio, em que a seqüência de execução é abruptamente alterada.

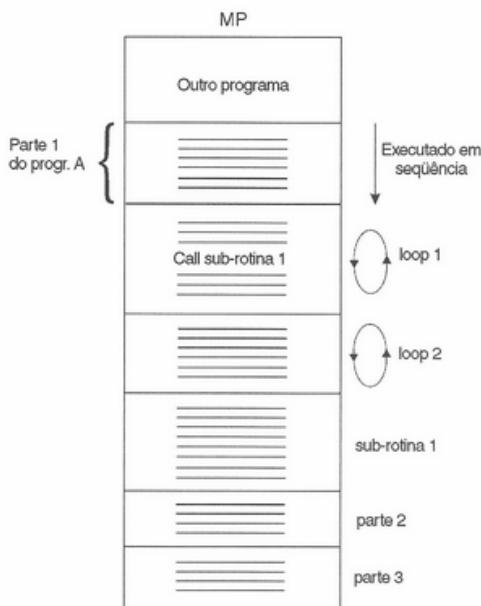


Figura 5.2 Um programa em execução com várias partes (exemplo do princípio de localidade especial).

Ao analisar a estrutura e a execução de diversos programas (comerciais, científicos, programas de exercícios etc.), os pesquisadores verificaram que os programas são, em média, executados de forma semelhante, isto é, em blocos de instruções seqüenciais, sendo algumas delas muitas vezes executadas mais de uma vez em curtos intervalos de tempo.

Há vários exemplos em serviços e práticas cotidianas, seja em computação ou não, que seguem essa tendência de realizar uma pequena quantidade do total de tarefas (ou qualquer coisa semelhante) mais freqüentemente e, por isso, essas tarefas ou ações são separadas para mais rápido acesso. Assim, quando estamos navegando na Internet temos a tendência de acessar uma página de um site mais de uma vez em curto espaço de tempo, ou seja, acessamos aquela página, passamos para outra e outra e, pouco depois, voltamos àquela página inicial. Os sistemas tendem a guardar essas páginas acessadas em nossa memória para evitar um longo acesso várias vezes à mesma página.

Outro exemplo pode ser de uma recepcionista de um escritório qualquer, que precisa realizar (ou receber) contatos de muitos clientes em um período de tempo de trabalho. As fichas de todos os clientes podem estar em ordem alfabética em um arquivo de aço, afastado de sua mesa e, assim, ela precisa buscar uma a uma seqüencialmente, embora muitos dos clientes há muito tempo não procurem o escritório e um grupo menor é freqüentemente contatado. Para facilitar seu trabalho, aumentando a produtividade nos contatos, é preciso que ela perca pouco tempo buscando a ficha de um cliente, o que pode ser obtido se a ficha estiver em uma pequena pasta suspensa, colocada em cima de sua mesa (mas a pasta tem espaço para poucas fichas).

Ao longo do tempo de trabalho (de execução, no caso de um programa, por exemplo), a recepcionista vai colocando na pasta suspensa em cima da mesa as fichas dos clientes de contato mais freqüente, de modo que sempre que ela precisar fazer contato com algum ela procura sua ficha primeiro na pasta da mesa, onde ela deve, na maioria das vezes (mais de 90% a 95%), encontrar a ficha desejada, ganhando um tempo considerável em relação ao processo inicial, em que ela precisaria levantar-se e ir ao arquivo de aço cada vez que desejasse uma ficha. É desnecessário mencionar que seria praticamente impossível a recepcionista trazer o arquivo para cima de sua mesa, pois o tempo de busca de uma ficha entre várias centenas delas continuaria a ser maior do que procurar uma ficha entre algumas poucas dezenas, na pequena pasta suspensa.

Da mesma forma, uma loja armazena nas prateleiras artigos semelhantes em uma mesma prateleira, pois um cliente quando localiza e apanha um artigo é muito provável que queira também examinar um outro parecido

e gastará pouco tempo buscando-o, pois está armazenado ao lado. O mais provável é que o atendente retire da prateleira para cima do balcão todos os produtos semelhantes ao que o cliente deseja, e que estão armazenados juntos. O tempo de acesso no balcão é naturalmente muito menor que a ida à prateleira.

Poderíamos ficar aqui apresentando inúmeros outros exemplos da localidade de busca, ou seja, agrupar pequenos itens de acesso freqüente próximo ao usuário para seu uso freqüente.

A este tipo de comportamento dos programas em execução, concluído pelos pesquisadores, chamou-se de **princípio da localidade**.

Basicamente, e de modo simplista, podemos definir o conceito de localidade como sendo o fenômeno relacionado com o modo pelo qual os programas em média são escritos pelo programador e executados pelo processador. Este princípio, aliás, não é aplicado apenas em memórias cache, fisicamente existentes em um chip, mas deu origem, como já observamos anteriormente, ao desenvolvimento da hierarquia de memória, com a implementação de diversos tipos diferentes de memória em um sistema de computação e mesmo, como mostrado nos exemplos, ao seu amplo uso na nossa vida comum.

Prosseguindo, este princípio pode ser decomposto em duas facetas ou modalidades: localidade *espacial* e *temporal*.

Na realidade, os programas não são executados de modo que a MP seja acessada randomicamente como seu nome sugere (RAM). Se um programa acessa uma palavra da memória, há uma boa probabilidade de que ele em breve acesse a mesma palavra novamente. Este é o princípio da *localidade temporal*. E se ele acessa uma palavra da memória, há uma boa probabilidade de que o acesso seguinte seja uma palavra subsequente ou de endereço adjacente àquela palavra que ele acabou de acessar. Nesse caso, trata-se da modalidade *localidade espacial*.

A modalidade de localidade espacial é aceitavelmente simples de ser entendida, pois se refere ao fato já mencionado e mostrado anteriormente de que os programas são executados em pequenos blocos de instruções, blocos esses constituídos de instruções executadas seqüencialmente. Na realidade, na maior parte do tempo é isso mesmo que acontece, tanto que o hardware do controle de execução das instruções nos processadores é construído com este propósito (no Cap. 6 será mostrado que, após a busca de cada instrução o hardware – registrador que armazena o endereço de acesso a instrução (o contador de instrução CI) – é incrementado para apontar para o endereço da próxima instrução na seqüência, pressupondo, então, que é ela a desejada). Eventualmente, esta ordem seqüencial é quebrada por uma instrução de desvio, como, p.ex, um IF-THEN-ELSE ou um DO-WHILE, ou uma chamada de rotina etc.

A *modalidade temporal* refere-se ao fato de os programas tenderem a usar freqüentemente o mesmo endereço em curtos espaços de tempo, como, por exemplo, em um *loop* (ou no caso da visita ao mesmo site várias vezes, como mencionamos no início deste capítulo). No caso de nossa recepcionista, é comum ela acessar a ficha de um mesmo cliente várias vezes em curtos intervalos de tempo.

A Fig. 5.2 já mostra um exemplo do princípio da localidade espacial. Nesta figura, um certo programa pode ser constituído de um grupo de instruções iniciais, realizadas em seqüência (parte 1), de dois loops (loop 1 e loop 2) de uma sub-rotina chamada dentro do loop 1, o que significa que ela será repetida diversas vezes, e do resto do código (parte 2 e parte 3).

O que acontece é que cada parte do programa (parte 1, loop 1, sub-rotina, loop 2, parte 2 e parte 3) é realizada separadamente, isto é, durante um tempo o processador somente acessa o grupo de instrução da parte 1, depois se dedica ao loop 1 e, neste, diversas vezes salta para a área da sub-rotina e acessa somente seu código, e assim por diante. Ou seja, o programa não salta indiscriminadamente da primeira instrução para uma no meio do programa, depois para outra no final, retornando para o início etc.

Na Fig. 5.3, apresentamos um programa em C, cuja estrutura e execução podem ser analisadas do mesmo modo, e com isso poderemos identificar a existência de ambos os princípios, da localidade espacial e da localidade temporal.

Pode-se observar, na figura, um exemplo da existência de localidade espacial na execução de programas, através da apresentação de um programa que calcula a média de duas turmas; com ele podemos vislumbrar a propriedade do emprego da memória cache em sua execução.

```

Cálculo da média de 2 turmas, A e B

void main ()
{
    printf ("Número de alunos da turma A: ");
    scanf ("%d", &quant_A);           início de execução em seqüência
    maior_nota_A = -1;
    soma_nota_A = 0;
    for (i=0;i < quant_A; i++)
    {
        printf ("Informe a matrícula do aluno: ");
        scanf ("%d",&matr[0][i]);
        printf ("Informe a nota: ");
        scanf ("%f",&nota[0][i]);
        if (nota[0][i] > maior_nota_A)
            maior_nota_A = nota[0][i];
        soma_nota_A = soma_nota_A + nota[0][i];
    }
    media_nota_A = soma_nota_A / quant_A;

    clrscr ();                      sub-rotina (limpa tela)

    printf ("Número de alunos da turma B: ");
    scanf ("%d", &quant_B);          início de exec. em seq.
    total = 0;
    soma_nota_B = 0;
    for (i=0;i < quant_B; i++)
    {
        printf ("Informe a matrícula do aluno: ");
        scanf ("%d",&matr[1][i]);
        printf ("Informe a nota: ");
        scanf ("%f",&nota[1][i]);
        if (nota[1][i] > maior_nota_A)
            total++;
        soma_nota_B = soma_nota_B + nota[1][i];
    }
    media_nota_B = soma_nota_B / quant_B;    término - loop       cálculo média turma B
    printf ("A média dos alunos da turma A foi: %4.2f", media_nota_A);
    printf ("A média dos alunos da turma B foi: %4.2f", media_nota_B);
    printf ("A nota mais alta da turma A foi: %4.2f", maior_nota_A);
    printf ("%d alunos da turma B obtiveram nota superior à maior nota da turma A", total);
}

```

Figura 5.3 Exemplo de programa para demonstração de localidades na sua execução.

Pode-se observar no programa os trechos de execução em seqüência, bem como dois loops, que mostram também a localidade temporal.

Assim, seria uma boa idéia tirar vantagem daqueles princípios de localidade se se colocasse a parte repetitiva de um pedaço do programa em uma memória bem rápida, mantendo o restante do programa, que não está sendo utilizado no momento, na memória mais lenta e de maior capacidade, porém mais barata.

5.2.3 Organização e Funcionamento da Memória Cache

Como aproveitar, então, os dois princípios da localidade? Conforme mencionado no parágrafo anterior, o projetista do sistema cria um elemento de memória intermediário entre o processador e a MP, como mostrado na Fig. 5.4. Este elemento de memória, denominado *memória cache*, deve possuir elevada velocidade de transferência e um tamanho capaz de armazenar partes de um programa, suficientemente grandes para obter o máximo rendimento do princípio da localidade espacial e suficientemente pequenas para não elevar em excesso o custo do sistema de computação.

Praticamente em todo este capítulo estamos nos referindo à memória cache fisicamente real, que é localizada internamente no invólucro do processador ou que é um chip inserido na placa-mãe, seja na sua organização e funcionamento, seja também referente aos seus elementos de projeto. Este foi o objetivo dos pesquisadores e projetistas de sistema nos primórdios da computação; naquela época e durante algum tempo sua

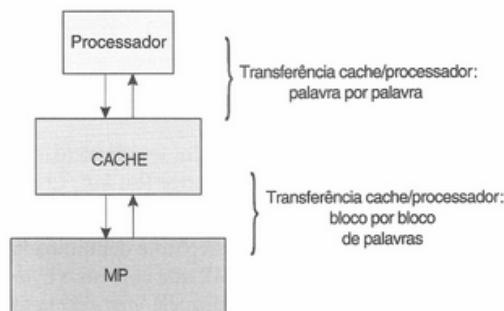


Figura 5.4 Organização para transferência de informações entre Processador/Cache/MP.

aplicação se resumia nesse uso. No entanto, atualmente se emprega o termo *cache* para expressar e organizar diferentes tipos de aplicação de armazenamento em sistemas de computação (ver item seguinte, 5.3), como cache de disco, cache no navegador (browser) e qualquer outro sistema de armazenamento que se valha das características do princípio da localidade.

Com a inclusão da memória cache entre o processador e a memória principal, podemos descrever os seguintes elementos e um procedimento de funcionamento entre os três componentes, conforme mostrado na Fig. 5.5.

Na figura observa-se que a conexão entre os dispositivos é comum ao processador e às duas memórias, de modo que, colocado um endereço no BE, este é “visto” tanto pela memória cache quanto pela MP; o mesmo ocorre com o BD, embora sob outro aspecto. Isto, como veremos a seguir, tem por propósito acelerar o processo de transferência ao facilitar a comunicação entre os três componentes.

Desde o princípio, e pelo menos até os dias atuais, o processador é projetado para, quando desejado, solicitar um dado que esteja armazenado na MP e, por isso, coloca no BE um valor binário correspondente ao endereço de uma célula (1 byte) da MP, independentemente do fato de existir ou não memória cache. Sem nos determos nesse instante sobre a organização da cache, vamos apenas descrever o processo operacional de uma transferência de dados entre os três componentes e, em seguida, descreveremos a organização do sistema todo.

Funcionamento Genérico de Acesso

Este procedimento operacional pode ser apresentado (sempre de forma simplificada, ver Fig. 5.5) referindo-se a uma operação de leitura de 1 byte de dados (1 célula da MP), conforme enumerado na página seguinte.

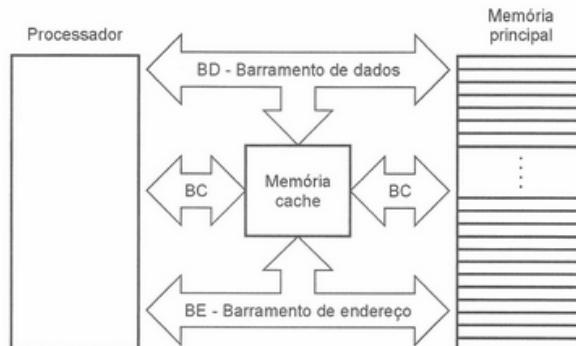


Figura 5.5 Exemplo de conexão e funcionamento do sistema processador, cache e memória principal.

1. o processador inicia a operação de leitura e coloca o endereço desejado da MP no BE (ver item 4.3.3);
2. o sistema de controle da cache intercepta o endereço, interpreta seu conteúdo (dependendo do método de mapeamento de endereço cache/MP a interpretação dos bits do endereço é diferente, conforme veremos no item 5.4.3);
3. da interpretação do endereço o controle da cache conclui se o byte (dado) solicitado está ou não armazenado na cache. Se estiver, este fato é denominado *acerto* (ou *hit*). O dado (cópia) é transferido, pelo BD, da cache para o processador na velocidade desses elementos, muito maior que a MP/processador.
4. se o byte desejado não estiver armazenado na cache, este fato é denominado *falta* (ou *miss*). Nesse caso, o controle da MP é acionado para localizar o bloco da MP que contém o byte desejado e este bloco é transferido para a cache, sendo armazenado em uma linha daquela memória (a seguir será definido o que é um bloco de MP e uma linha da memória cache). Em seguida, o byte desejado é transferido para o processador. Naturalmente, a operação decorrente de uma *falta* é muito mais demorada que a de um acerto, e será mais ainda se o conteúdo da linha (algum byte dela) tiver sido alterado, pois esta terá que ser transferida de volta para a MP para garantir a integridade dos dados alterados após o término da execução do programa;
5. considerando o que é estabelecido no princípio da localidade espacial, de que, realizado um acesso a um determinado endereço, o acesso seguinte deve (é muito provável) ser realizado no endereço contíguo de memória e tendo em vista aproveitar ao máximo a maior velocidade (pequeno tempo de acesso) das memórias cache, quando o sistema de controle tiver que buscar um dado na MP ele busca esse dado e mais alguns que se supõe o processador desejará em seguida. Daí o conceito de divisão da MP em blocos de X bytes e da cache em linhas com X bytes de largura.

O que se deseja, então, é um máximo de *acertos* (*hits*) e um mínimo de *faltas* (*misses*), para que o sistema tenha um bom desempenho. Podemos definir um valor de eficiência da cache pela relação entre acertos e o total de acessos:

$$E_c = \frac{\text{Acertos (Hit)}}{\text{Total acessos}} * 100 \quad \text{sendo } E_c = \text{eficiência da cache.}$$

Exemplo 5.1

Um determinado sistema de computação possui uma memória cache, MP e processador. Em operações normais, obtém-se 96 acertos para cada 100 acessos do processador às memórias. Qual deve ser a eficiência do sistema cache/MP?

Solução

Se em 100 acessos ocorrem 96 acertos, teremos quanto faltas e a eficiência do sistema será:

$$E_c = \frac{96}{100} = 0,96 * 100 = 96\%$$

Organização Genérica de Memórias Cache

Para permitir, com aproveitamento, o funcionamento adequado dos sistemas de armazenamento, as memórias cache são organizadas de modo diferente da memória principal (RAM) e essas, por sua vez, passam a ter, para o sistema de acesso processador/cache/MP, uma organização lógica diferente da tradicional organização física (conjunto de N células sequencialmente organizadas por endereços subseqüentes de 1 a N – 1). Na realidade, uma outra memória, chamada virtual, também se encaixaria nesse mesmo tipo de organização a ser apresentado (na memória secundária), mas não faz parte do escopo deste livro).

A Fig. 5.6 mostra um exemplo de organização básica de memória cache, e na Fig. 5.7 é mostrada essa mesma organização de memória cache, mas foi acrescentada a memória principal, com sua organização física-padrão (em células) e a organização lógica apropriada, em blocos, para funcionamento com a memória cache.

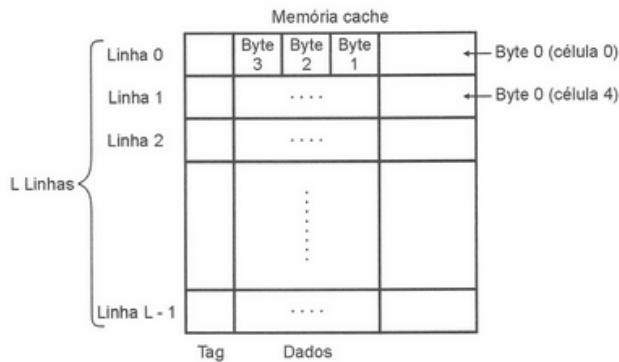


Figura 5.6 Organização básica de uma memória cache.

Desse modo, a MP permanece fisicamente como uma seqüência ordenada e contínua de células ou bytes, visto que na prática, conforme já mencionado no Cap. 4, atualmente todas as memórias principais (RAM) são organizadas com células de largura igual a 8 bits, ou 1 byte.

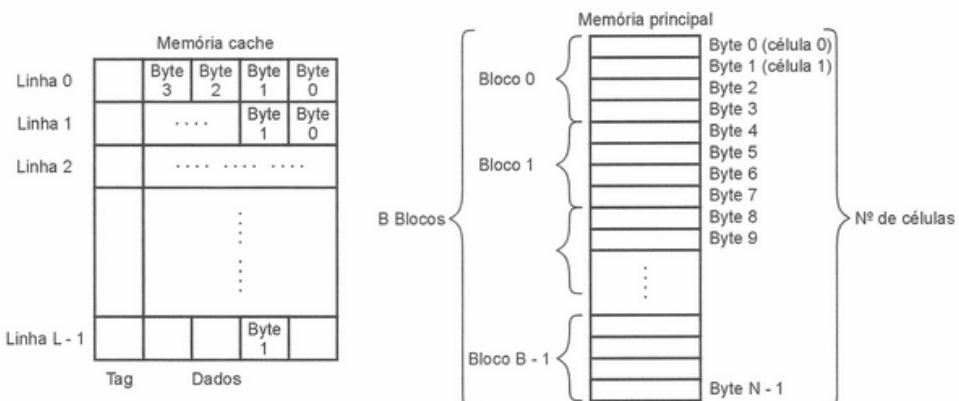


Figura 5.7 Organização memória cache/memória principal.

No entanto, para funcionar com a cache inserida no sistema o controle da cache (também localizado no *chipset* – ver Caps. 4 e 6) considera a MP organizada em blocos de X células ou X bytes cada, de modo que, quando há uma solicitação de transferência de dados pelo processador e o sistema de controle não encontra o dado desejado na cache, ocorre a sua transferência da MP para a memória cache, porém não só este byte é transferido, mas também os subsequentes, que fazem parte do conjunto (bloco). Repetindo o que foi mencionado anteriormente, isso se baseia no princípio da *localidade espacial*. Ou seja, transfere-se o dado desejado e mais alguns outros que se pressupõe o processador irá precisar logo em seguida.

No exemplo das Figs. 5.6 e 5.7, cada bloco possui 4 bytes (quatro células) de largura, que deve ser a mesma largura de uma linha da cache, isto é, o $X = 4$ (largura de bloco/Linha).

Em resumo (Fig. 5.7):

- A MP é fisicamente organizada em uma seqüência contínua de N células cada uma com 1 byte de largura; então, é constituída fisicamente de N bytes de dados, do endereço 0 (byte 0) até o endereço $N - 1$ (byte $N - 1$).

- Para funcionar integrada ao sistema processador/cache/MP, esta última é organizada em B blocos, de X bytes (X células cada um), cada um deles com endereço B_i , sendo $i = 0$ a $B - 1$.

$$\text{Quantidade de blocos} = \frac{\text{Quantidade de células (bytes)}}{\text{Largura de 1 bloco (X bytes)}}$$

- A memória cache é organizada em um conjunto de L linhas, sendo cada linha constituída de X bytes, mesma largura de 1 bloco da MP. As linhas têm endereço de 0 até $L - 1$. Além disso, cada linha possui um campo indicador do endereço do bloco que está naquele instante armazenado nela. Este campo é chamado Tag ou rótulo (usaremos no decorrer deste capítulo o termo Tag, mais conhecido na literatura).
- Então, cada bloco ou linha, possuindo X bytes de largura, cada byte é endereçado igualmente por endereços 0, 1, 2... X - 1 e, portanto, sua localização é composta, no mínimo, pelo endereço do bloco e do byte em seu interior.
- No exemplo da Fig. 5.7, a largura de 1 bloco/Linha = X = 4 bytes, de modo que teremos sempre os bytes 0, 1, 2 e 3 (que é X - 1). O bloco de endereço 0 (bloco 0) é constituído dos bytes 0, 1, 2 e 3, correspondentes, respectivamente, às células de endereços 0, 1, 2 e 3; o bloco 1 é constituído dos bytes 0, 1, 2 e 3 dele, correspondentes, respectivamente, às células de endereços 4, 5, 6 e 7; o bloco 2, constituído dos bytes 0, 1, 2 e 3 dele, correspondentes, respectivamente, às células de endereços 8, 9, 10 e 11, e assim por diante, até o bloco $B - 1$ (último bloco da MP).

É claro, a quantidade de blocos B é sempre muito maior que a quantidade de Linhas, L, requerendo que se tenha que definir métodos para mapear os blocos da MP nas poucas linhas da cache; no item 5.4.1 serão descritos três métodos desenvolvidos: *direto*, *associativo* e *associativo por conjunto*.

É importante ressaltar a importância do princípio da localidade (importância da construção de programas de forma estruturada), pois, mesmo se tendo uma enorme relação entre quantidade de blocos e quantidade de linhas (tamanho da MP muito maior que o da cache, como, p.ex., em sistemas atuais, com 128MB ou 256MB de MP e caches L2 de 1MB a 4MB e dessas para as L1 com algumas dezenas de KB), ainda assim, a eficiência das memórias cache, E_c , é da ordem de 95% a 98%.

A Fig. 5.8 mostra um processador com barramento único (ônibus) e memória cache incluída.



Figura 5.8 Exemplo de um sistema de computação (microcomputador) com utilização de memória cache em um barramento único.

5.3 TIPOS DE USO DE MEMÓRIA CACHE

A importância das memórias cache nos sistemas de computação é inquestionável, e atualmente elas se tornam cada vez mais imprescindíveis para um correto e eficaz desempenho dos sistemas. Esta importância tem-se traduzido, entre outros elementos, no desenvolvimento e na criação de diferentes tipos de cache.

Em princípio, podem-se definir dois tipos básicos de emprego de cache nos sistemas de computação contemporâneos:

- na relação UCP/MP (cache de RAM ou “RAM Cache”) e
- na relação MP/Discos (cache de disco ou “Disk Cache”).

O primeiro tipo, cache de RAM ou cache para a MP, refere-se ao conceito exposto nos itens anteriores, em que a memória cache é utilizada para substituir o uso da memória principal (ou RAM) pelo processador, acelerando o processo de transferência de dados desejados pelo processador.

No segundo tipo, cache de disco, o sistema funciona segundo os mesmos princípios da cache de memória RAM, porém, em vez de utilizar a memória de alta velocidade SRAM para servir de cache o sistema usa uma parte da memória principal, DRAM, como se fosse um espaço em disco (vale-se de uma parte da RAM como buffer). Deste modo, quando um programa requer um dado que esteja armazenado em disco, o sistema verifica em primeiro lugar se o dado está no espaço reservado na memória RAM e que simula o espaço em disco; da mesma forma que na cache de RAM, se o dado for encontrado no buffer da memória (que simula o espaço em disco) haverá um acerto (hit), e se não for encontrado lá então ocorre uma falta (miss), e nesse caso o sistema interrompe o seu processamento para acessar o disco efetivamente, localizar e transferir o dado desejado e mais um bloco de dados subsequentes, de modo semelhante ao da cache de RAM.

Assim como no caso de cache de RAM, a cache de disco pode aumentar excepcionalmente o desempenho do sistema, visto que o acesso à memória RAM (faixa de nanosegundos) é milhares de vezes mais rápido que o acesso ao disco (faixa de milissegundos).

5.4 ELEMENTOS DE PROJETO DE UMA MEMÓRIA CACHE

Para se efetivar o projeto e a implementação de uma memória cache deve-se decidir entre várias alternativas tecnológicas, atualmente disponíveis, as quais podem ser agrupadas por função:

- Função de mapeamento de dados MP/cache
- Algoritmos de substituição de dados na cache
- Política de escrita pela cache
- Níveis de cache
- Definição do tamanho das memórias cache, L1 e L2
- Escolha de largura de linha de cache

5.4.1 Mapeamento de Dados MP/Cache

Para entender melhor o sentido desta função como elemento de projeto de memória cache, vamos repetir algumas considerações sobre a organização da MP e memória cache, explicadas no item 5.2.3. A memória RAM (MP) consiste em um conjunto sequencial de $N = 2^E$ palavras endereçáveis (células), cada uma possuindo um único e unívoco endereço com E bits de largura; as células são dispostas a partir do endereço 0 até a célula de endereço ($N - 1$) e, na prática, todas possuem largura de 8 bits ou 1 byte para armazenamento de dados (ver exemplo da Fig. 5.7).

Para efeito de funcionamento da memória principal, MP (memória RAM), com a memória cache, consideremos:

- que a MP está organizada como um conjunto de B blocos (numerados de 0 a $B - 1$);
- que cada bloco da MP é constituído de X células (X bytes, já que nos sistemas atuais uma célula armazena 8 bits ou 1 byte de dados);
- que a quantidade B de blocos da MP é $B = N / X$ ou $B = 2^E / X$ (ver Fig. 5.7);
- que a memória cache é organizada como um conjunto de L grupos de bytes, sendo cada um deles denominado linha. Cada uma das L linhas da cache possui X bytes, isto é, a mesma quantidade de bytes de um bloco da MP;
- o tamanho da cache ($L * X$ bytes) é sempre muito menor que o tamanho da MP ($B * X$ bytes).

Observemos, então, a extraordinária tecnologia de uso da memória cache: esta memória possui uma capacidade menor que 1% da capacidade da memória RAM, mas permite obter 90% a 95% de taxa de acertos (hits). É por esta razão que atualmente todos os sistemas utilizam o conceito de cache com mais de um tipo, L1 e L2, e muitos com L3 também. Naturalmente, esta enorme eficácia é possível graças ao conhecimento do princípio da localidade, exposto anteriormente. Para recapitularmos este princípio utilizemos um simples exemplo, através de um pequeno loop, dos inúmeros que qualquer programa tem.

Suponhamos um trecho de um programa do tipo:

```

Índice = 1000
Enquanto Índice diferente de zero
Iniciar
    X(Índice) = A (índice) + B (índice)
    Índice = Índice - 1
Terminar

```

Neste exemplo, consideremos que a memória cache é capaz de armazenar este simples trecho do programa. Isto significa que em 999 vezes das 1000 o sistema obterá um acerto, ou seja, uma taxa de acerto da ordem de 99,9%, mesmo considerando-se que sua capacidade é muito menor que a da memória RAM.

A cada instante, a memória cache possui um conjunto de blocos de MP armazenados em suas *linhas*, com os dados que o processador deve precisar (para haver acertos e não faltas). Porém, como $L \ll B$ (há muito mais blocos que linhas) não é possível uma *linha* da cache estar dedicada a armazenar um específico bloco da MP, ou seja, uma linha é usada por mais de um bloco e, por conseguinte, é preciso identificar, em cada instante, qual o específico bloco que está armazenado na específica linha da cache. Para isso, conforme mencionado anteriormente, cada linha tem um campo (além daquele para armazenar as X palavras), denominado etiqueta (*tag*), que contém a identificação do bloco e que, como veremos a seguir, faz parte dos E bits do endereço completo da MP.

Considerando, então, a grande diferença de tamanho entre as duas memórias, ou seja, há uma grande quantidade de blocos da MP que, durante o funcionamento do sistema, precisarão ser armazenados na pequena quantidade de linhas da memória cache (por demanda do processador ao longo da execução dos programas).

Em outras palavras, há necessidade de se determinar um meio eficaz de determinar qual a linha em que um bloco específico será armazenado, quando solicitado pelo sistema. Ora, como $B \gg L$, então não se pode ter uma relação 1:1 entre eles, daí a necessidade de estabelecer um método de mapear os endereços dos blocos com os endereços das linhas. Há três alternativas:

- Mapeamento direto
- Mapeamento associativo
- Mapeamento associativo por conjuntos

Na realidade, poderíamos classificar em dois: direto e associativo, tendo estas duas modalidades: completo e por conjuntos.

Mapeamento Direto

Por esta técnica, cada bloco da MP tem uma linha da cache previamente definida onde será armazenado. Como há mais blocos do que linhas da cache, isso significa que muitos blocos irão ser destinados a uma mesma linha, naturalmente, um bloco de cada vez; é, portanto, preciso definir a regra a ser seguida para a escolha da linha específica de cada bloco.

Desta forma, os X primeiros bytes (X primeiras células) da MP constituirão o bloco 0 (zero), o qual estará previamente destinado (quando tiver que ser armazenado na cache) à linha de endereço 0 (zero) da cache.

Os X bytes seguintes constituirão o bloco 1 (um), o qual estará previamente destinado à linha de endereço 1 (um) da cache, e assim por diante, até o bloco de endereço correspondente ao valor $L - 1$, o qual estará previamente destinado à linha de endereço $L - 1$ (última linha da cache).

Os X bytes seguintes constituirão o bloco de endereço correspondente ao valor L, o qual também estará destinado ao endereço 0 (zero), assim como o bloco $L + 1$ mapeado à linha 1 (um) da cache, o bloco $L + 2$ à linha 2, e assim por diante.

Assim, a linha 0 (zero) da cache é destino de armazenamento dos blocos 0 (zero), L , $2L$, $3L$ etc. (naturalmente, um em cada instante de tempo); a linha 1 (um) é destino dos blocos 1 (um), $L + 1$; $2L + 1$ etc.

Na Fig. 5.9 apresenta-se um exemplo simples para mostrar, com números, os elementos aqui mencionados acima.

No exemplo, a MP possui 64 células de 1 byte cada, ou um total de 64 bytes (64B), e a memória cache tem uma capacidade de 16 bytes (16B). A MP é organizada em blocos de 4 bytes (células) cada um, de byte 0 a byte 3, mesma largura das linhas da cache.

Como a cache possui um total de 16B e cada linha tem 4B de largura, ela acomoda apenas quatro linhas, endereçadas de linha 0 a linha 3 (ver a Fig. 5.9).

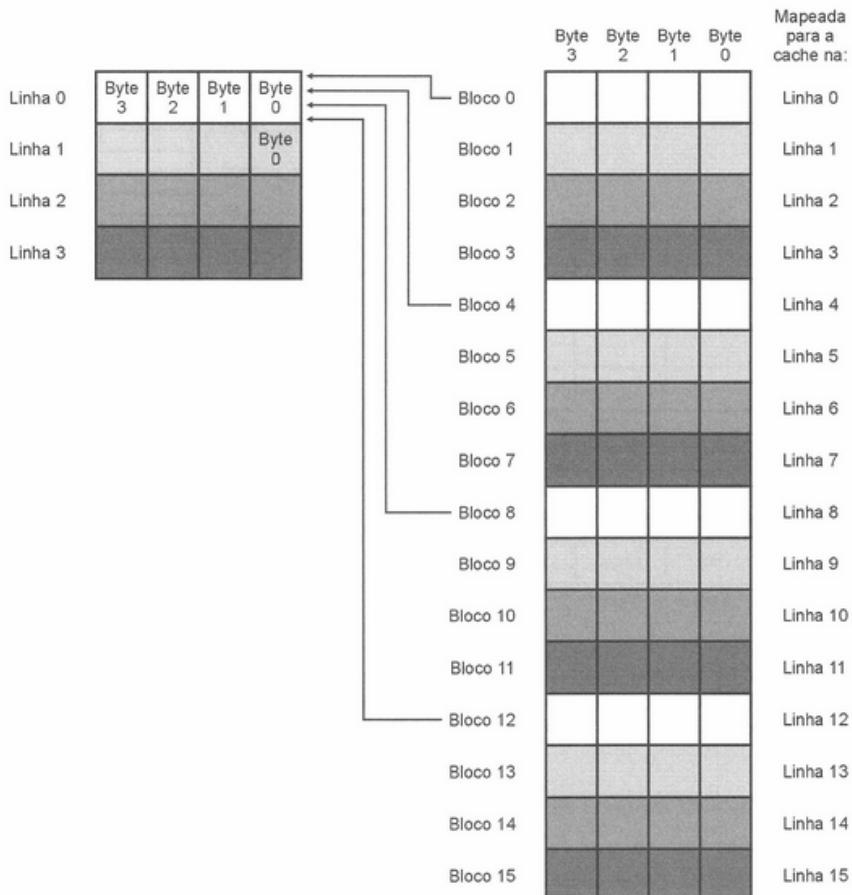


Figura 5.9 Exemplo de mapeamento direto. A memória possui 64 células (64 bytes) e a memória cache possui 16 bytes (quatro linhas com quatro bytes cada). Cada linha pode armazenar quatro blocos, um de cada vez. Exemplo: a linha 0 pode armazenar os blocos 0, 4, 8 e 12.

Em resumo:

- Total de capacidade da MP = 64B ou 2^6 , ou seja, cada endereço E da MP possui seis bits de largura.
- Total de blocos = $64B / 4B = 16$ ou 2^4 .
- Capacidade total da cache = 16B, e cada linha = 4 bytes.
- Total de linhas = $16 / 4 =$ quatro linhas ou 2^2 , ou seja, cada endereço de linha possui 2 bits de largura.
- Como há 16 blocos na MP e quatro linhas na cache, cada linha pode acomodar quatro blocos, naturalmente um de cada vez. Sabemos que $4 = 2^2$, ou seja, 2 bits para tag.

Para detalhar um pouco mais o método, apresenta-se um outro exemplo (exemplos nunca são demais para melhor compreensão do leitor), ainda com valores pequenos, de modo que os desenhos possam ser completos em termos dos elementos das memórias, e que está mostrado na Fig. 5.10.

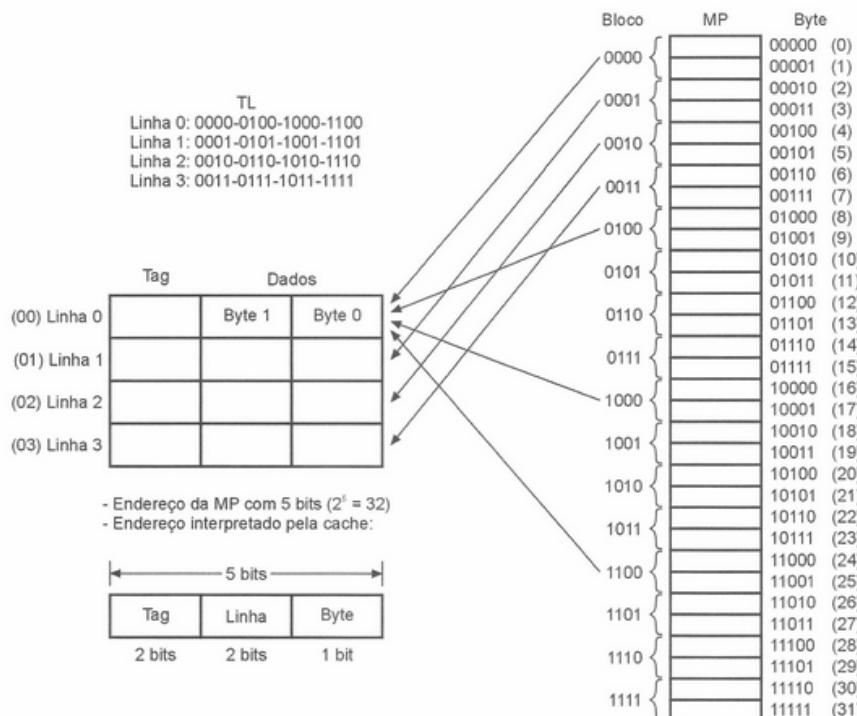


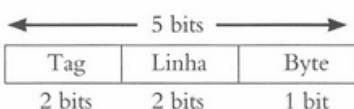
Figura 5.10 Exemplo de organização com mapeamento direto em uma MP com 32 células (bytes) e uma cache com quatro linhas de 2 bytes cada.

Na figura, observam-se os seguintes elementos e valores:

- Memória principal, MP, com 32 células de 8 bits cada uma, ou 32B (bytes), com endereços desde 0 (zero) até 31. Como $2^5 = 32$, cada endereço binário possui 5 bits de largura e vai de 00000 até 11111, como mostrado à direita de cada célula.
- A MP é também organizada em blocos de 2 bytes, totalizando $B = 32 / 2 = 16$ blocos e, como $2^4 = 16$, então cada bloco tem um endereço de 4 bits de largura, indo de 0000 até 1111, como mostrado à esquerda de cada bloco da MP na figura. Observa-se que os endereços dos blocos correspondem aos quatro algarismos mais significativos (mais à esquerda) do endereço de uma célula.

- A memória cache possui 8 bytes de tamanho para dados, sendo organizada em quatro linhas de 2 bytes cada uma, pois os blocos da MP também têm largura de 2 bytes. O endereço de um dos dois blocos precisa apenas de 1 bit (bloco 0 ou bloco 1), sendo, portanto, o último algarismo à direita do endereço da célula. Em outras palavras: os 5 bits de endereço de célula dividem-se em quatro para os endereços de cada um dos 16 blocos e um para o endereço do específico byte.
- Como a cache possui quatro linhas, cada endereço de linha é um número de 2 bits de largura ($2^2 = 4$).
- Tendo 16 blocos para serem armazenados (quando solicitado pelo processador em um determinado acesso) em uma das quatro linhas, isto significa que cada linha terá atribuição de receber quatro blocos ($16 / 4 = 4$), estando previamente determinado como isto ocorrerá. A figura confirma o que já foi mencionado anteriormente:
 - A linha de endereço 00_2 (decimal 0) recebe os blocos $0000 - 0100 - 1000$ e 1100 (observa-se, nesse caso, que os dois bits à direita de cada bloco são 00 nos quatro blocos – endereço da linha).
 - As linhas 01, 10 e 11 receberão os blocos indicados na figura. Verifica-se que os dois últimos bits são sempre o endereço da linha.
- Como temos quatro blocos para cada linha, é necessário um meio de identificar qual deles está armazenado em um dado instante para que o sistema identifique se ocorreu uma falta (miss) ou acerto (hit) em cada acesso e tome as providências de acordo. Para isso, as memórias cache possuem um campo adicional ao dos dados, denominado etiqueta (ou, em inglês, *tag*), o qual permite identificar qual bloco, daqueles destinados àquela linha, está armazenado. No exemplo mostrado na Fig. 5.10, o campo tag possui 2 bits de largura (correspondendo aos 2 bits mais significativos do endereço da célula, mais à esquerda).

Neste ponto, podemos mostrar o formato de endereço de célula como é interpretado pelo sistema de controle da cache para determinar se o byte requerido está na cache (hit) ou se não está (miss).



O endereço é subdividido em três campos distintos, que serão interpretados de forma diferente pelo sistema de controle da cache:

Tag – 2 bits ($2^2 = 4$ blocos por linha) – serve para se saber que bloco está armazenado no momento na linha especificada;

Linha – 2 bits ($2^2 =$ quatro linhas) – indica o endereço da linha onde pode estar o dado;

Byte – 1 bit ($2^1 = 2$ bytes por bloco) – caso haja um acerto (hit), indica qual byte está sendo requerido pelo processador.

Vamos considerar, em seguida, um exemplo com valores mais reais.

Seja uma memória principal (MP) com espaço de endereçamento de 4GB (células), tendo cada uma um endereço com 32 bits ($2^{32} = 4G$), como acontece nos processadores INTEL Pentium ou AMD K7 e outros.

Para exemplificar, vamos assumir que a cache associada a esta memória RAM (MP) possui um tamanho correspondente a 64 Kbytes, divididos em 1024 (2^{10}) ou 1K linhas, com 64 bytes de dados cada uma (largura de 1 bloco/1 linha = 64 bytes ou células de dados = 64B). Cada bloco ou Linha terá, então, os bytes de 0 a 63 (de modo semelhante ao que mostramos no exemplo da Fig. 5.7).

A memória principal será, então, dividida em 64M blocos de 64 bytes cada (64B é a mesma quantidade de bytes do bloco de dados da memória principal e da linha da cache). A Fig. 5.11 mostra, com dados, o exemplo que estamos descrevendo.

Relembrando as informações do item anterior, vemos que a MP possui N palavras (células), divididas em B blocos e que a cache possui L linhas de 64B cada. Sendo $B = N/L$, então: $B = 4G / 64K = 64M$ blocos, numerados de bloco 0 até bloco $64M - 1$.

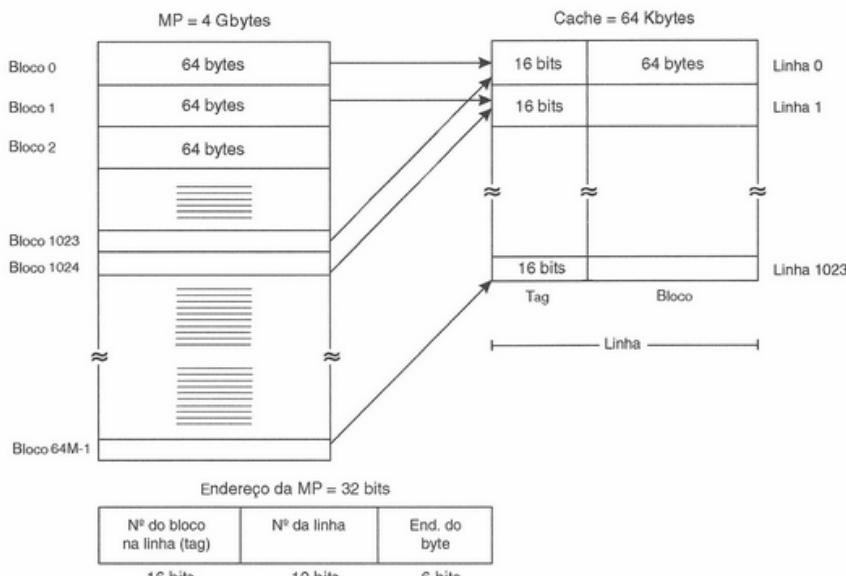


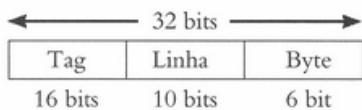
Figura 5.11 Memória cache com mapeamento direto.

Como há 64M blocos na MP e 1024 linhas na cache, então cada linha deverá acomodar (um de cada vez, é claro) 65.536 blocos ($64K = 2^{16}$).

Em resumo:

- Total de capacidade da MP = 4GB ou 2^{32} , ou seja, cada endereço E da MP possui 32 bits de largura.
- Total de capacidade da cache: 64KB. Cada linha possui 64B.
- Total de blocos da MP = $N / L = 4GB / 64B = 2^{32} / 2^6 = 2^{26}$, portanto cada endereço de bloco tem 26 bits de largura (campo Linha + Tag do formato de endereços).
- Total de linhas = $64 KB / 64 B = 1024$ linhas ou 1K, ou 2^{10} , ou seja, cada endereço de linha possui 10 bits de largura.
- Como há 64M blocos na MP e 1024 linhas na cache, cada linha pode acomodar $64M / 1K = 2^{26} / 2^{10} = 2^{16}$ ou 64K blocos, naturalmente um de cada vez. Assim, o campo Tag possuirá 16 bits.

O formato de endereço de célula como é interpretado pelo sistema de controle da cache para determinar se o byte requerido está na cache (hit) ou se não está (miss).



Tag – 16 bits – há 65.536 blocos atribuídos a cada linha ($64K = 2^{16}$);

Linha – 10 bits – há 1024 ou 1K linhas;

Byte – 6 bits – há 64 ($2^6 = 64$) bytes por bloco/linha.

Para definir quais blocos da MP serão alocados a uma linha específica, pode-se efetuar um simples cálculo usando-se aritmética de módulo (resto da divisão). Assim, para definir a linha de cada bloco, seja:

EL = endereço da linha da cache (desde linha 0 até linha 1024 – 1)

E = endereço da RAM (MP), que vai de endereço 0 até end. $4G - 1$

L = quantidade de linhas da cache, sendo no caso $Q = 1024$

e, então: $EL = E$ módulo L .

No exemplo da Fig. 5.11, $L = 1024$ e, portanto, $EL = E$, módulo 1024, e teríamos o seguinte mapeamento previamente definido:

- para a linha 0 estarão destinados os blocos 0, 1024, 2048, 3072 e assim por diante, num total de 64K blocos;
- para a linha 1 estarão destinados os blocos 1, 1025, 2049, 3073 e assim por diante;
- e assim sucessivamente até a linha 1023, que receberá o bloco 1023, 2047, até o bloco $64M - 1$.

Deste modo, cada bloco da MP estará **diretamente mapeado** a uma linha específica da cache, daí o nome do método de **mapeamento direto**.

Uma outra observação interessante refere-se ao tamanho da cache em bits, valor que determina, entre outras coisas, seu custo, devido à quantidade de componentes físicos (transistores etc.) que deverão ser requeridos.

A quantidade de bits de uma memória cache compreende os bits necessários para armazenamento dos dados (no exemplo da Fig. 5.10 é de 64 bits (quatro linhas * 2 bytes cada), e no exemplo da Fig. 5.11 é de 524.288 bits = $64KB * 8$ bits) mais os bits necessários para os L campos *tag* (etiqueta) da memória (no exemplo da Fig. 5.10 é de 8 bits = quatro linhas * 2 bits, e no exemplo da Fig. 5.11 é de 1.048.576 bits = 65.536 linhas * 16 bits).

Em seguida, vamos apresentar o procedimento básico de uma operação de leitura efetuada pelo processador e que é interpretada primeiramente pelo sistema de controle da cache. Para isso, utilizaremos os dois exemplos mostrados nas Figs. 5.10 e 5.11.

Em primeiro lugar, considera-se o exemplo da Fig. 5.10, onde temos:

- endereço de byte (célula) = 5 bits
- endereço de linha = 2 bits
- tag = 2 bits

A Fig. 5.12 auxilia na compreensão do procedimento, que é descrito a seguir:

- 1) O endereço de 5 bits (endereço de célula/byte) que se apresenta para cache e MP é interpretado pelo sistema de controle da cache conforme os campos específicos:
tag: 2 bits; linha: 2 bits; byte: 1 bit.
- 2) Primeiramente, o sistema decodifica a parte do endereço referente ao endereço de linha (dois bits centrais); daí, aponta para a linha selecionada.
- 3) Em seguida, o sistema irá checar se o bloco que contém o byte desejado está armazenado naquela linha (**acerto ou hit**) ou não (**falta ou miss**). Para isso, é realizada uma comparação (usualmente por meio de portas **xor**) entre o valor do campo *tag* da linha e o valor no campo *tag* do endereço (2 bits).
- 4) Se forem valores iguais, significa que o bloco procurado se encontra armazenado na cache (um acerto ou *hit*). Nesse caso, o endereço do byte desejado é passado para um decodificador de endereços que localiza e transfere o byte para o processador pelo barramento de dados (procedimento semelhante ao mostrado no item D.1).
- 5) Se os valores forem diferentes, significa que o bloco não se encontra na cache (falta ou *miss*). O sistema interrompe o processamento e vai iniciar a localização e a busca do bloco requerido (o endereço corresponde aos 4 bits mais significativos) para transferir uma cópia da MP para a linha específica.
- 6) Nesse ponto, o sistema precisa verificar se algum byte do bloco foi alterado durante o período em que esteve armazenado (se o processador realizou alguma operação de escrita). Se houve alteração, então o bloco a ser substituído retorna para a MP, caso contrário ele simplesmente é destruído pelo armazena-

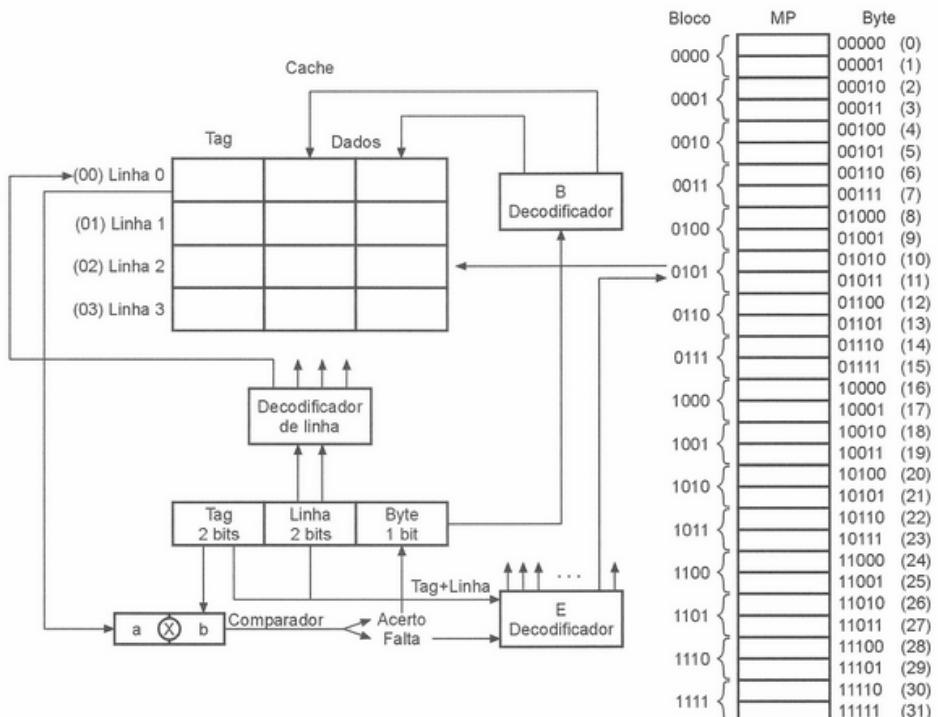


Figura 5.12 Exemplo de acesso à memória cache por meio de mapeamento direto.

mento em cima do novo bloco. Para saber se houve ou não alteração no bloco, os sistemas usam um bit adicional em cada linha, que será setado (bit 1) caso tenha ocorrido alguma operação de escrita ou se mantém em 0 (zero) caso não.

Em seguida, utilizaremos os elementos do sistema exemplificado na Fig. 5.11, onde temos:

- endereço de byte (célula) = 32 bits
- endereço de linha = 10 bits
- tag = 16 bits

O procedimento, a seguir apresentado, pode ser acompanhado por meio da Fig. 5.13.

- 1) O processador apresenta o endereço de 32 bits no BE, comum à cache e MP e sinal de controle correspondente à operação desejada no BC, que é primeiramente interceptado pelo circuito de controle da cache (ver Fig. 4.5). Este circuito inicia a identificação de seus campos para definir primeiramente se a palavra desejada está na cache ou não. Para exemplificar, vamos considerar o endereço binário 00000000000001000000110001001000. Para efeitos de processamento pelo sistema de controle da cache, este endereço será dividido em três partes, conforme já mostrado na Fig. 5.11, da esquerda para a direita, a saber:

0000000000000100	0000110001	001000
Tag (16 bits)	Linha (10 bits)	Byte (6 bits)

- 2) Os 10 bits centrais são examinados (ver Fig. 5.13), e seu valor indica que se trata da linha 000011001₂ = 25₁₀. Resta verificar se o bloco solicitado é o que se encontra armazenado no quadro 25 ou se lá está um outro bloco.

- 3) O controlador da cache examina por comparação se o valor do campo tag do endereço – no caso, o valor é $0000000000000000100_2 = 4_{10}$ – é igual ao do campo tag da linha. No exemplo dado eles são iguais. Isso significa um acerto (Hit), que o dado desejado encontra-se também na cache.
- 4) Em seguida, é acessado o dado do endereço $= 001000_2 = 8_{10}$ (últimos 6 bits do endereço) e transferido para o processador.
- 5) Se os valores dos campos *tag*, do endereço e da linha da cache não fossem iguais, isto significaria que o bloco desejado não se encontrava armazenado na cache (uma falta ou miss) e, portanto, deveria ser transferido da MP para o quadro 25, substituindo o atual bloco, para, em seguida, o dado (o byte de dados) requerido – byte 8 – ser transferido para o processador pelo barramento de dados, BD. Na realidade, dois acessos: uma cópia do bloco da MP para a cache e o dado da cache para o processador.

Para tanto, os 26 bits mais significativos do endereço (16 bits do campo tag mais 10 bits do campo quadro) seriam utilizados como endereço do bloco desejado, pois: $2^{26} = 64M$, ou seja, cada um dos 64M blocos tem um endereço com 26 bits.

Deve ser observado que os procedimentos mostrados são básicos para qualquer tipo de cache, porém há algumas pequenas diferenças, específicas em cada caso, como, por exemplo, para operações de leitura ou de escrita; ou, ainda, se se trata de uma cache L1 de dados (em que a atualização do bloco que retorna é importante) ou de instruções (nunca há retorno de instrução para MP) e estas são buscadas pelo endereço contido no CI (*program counter*, ou *PC*).

A técnica de mapeamento direto é, sem dúvida, simples e de baixo custo de implementação, além de não acarretar sensíveis atrasos de processamento dos endereços. O seu problema consiste justamente na fixação da localização para os blocos (em um dos exemplos dados (Fig. 5.11), 65.536 blocos estão destinados a uma linha, o que indica que somente um de cada vez pode estar lá armazenado).

Se, por exemplo, durante a execução de um programa um dado código fizer repetidas referências (acessos) a palavras situadas em blocos alocados na mesma linha, então haverá necessidade de sucessivos acessos à MP para substituição de blocos (muitas faltas) e a relação acerto/faltas será baixa, com a consequente redução de desempenho do sistema. No exemplo da Fig. 5.11 poderíamos ter referências seguidas a uma célula do bloco 0 e a outra célula do bloco 1024, como também, por coincidência, outro acesso subsequente a uma célula do bloco 2048, todos destinados à mesma linha 0 da cache.

Para concluir, apresentam-se alguns exemplos que auxiliam a entender melhor o conceito e a sistemática do mapeamento direto.

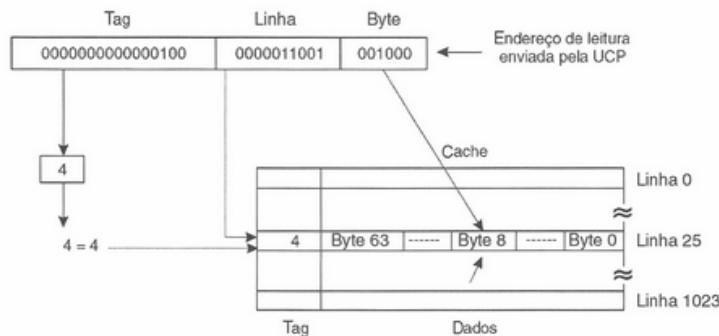


Figura 5.13 Exemplo de operação de leitura em memória cache com mapeamento direto.

Exemplo 5.2

Cálculo da quantidade de bits necessários para uma determinada memória cache.

Considere um sistema de computação com uma memória cache de 32KB de capacidade, constituída de linhas com 8 bytes de largura. A MP possui uma capacidade de 16MB.

Solução

O total de bits a serem usados na cache é a soma dos bits de dados (produto da quantidade de linhas pela largura em bits de cada uma) mais os bits usados na coluna *tag*, cujo valor existe para cada linha.

Assim, temos:

$$T_{\text{total de bits da cache}} = T_d (\text{total de bits para a parte de dados}) + T_t (\text{total de bits dos tags})$$

$$T_d = 32\text{KB} \times 8 \text{ bits} = 32 * 1024 * 8 = 262.144 \text{ bits}$$

$$T_t = \text{quantidade de linhas} * \text{largura do campo tag}$$

$$\text{Quantidade de linhas} = 32\text{KB} / 8 \text{ bytes} = 4\text{KB} = 2^{12}$$

$$\text{Largura do campo tag} = \text{quantidade de blocos} / \text{quantidade de linhas}$$

$$\text{Quantidade de blocos} = 16\text{MB} / 8\text{B} = 2\text{MB} = 2^{21}$$

$$\text{Quantidade de blocos por linha} = 2^{21} / 2^{12} = 2^9 = 512, \text{ e o campo tag possui 9 bits de largura.}$$

$$T_t = 4 * 1024 * 9 = 36.864$$

$$T = 262.144 + 36.864 = 299.008 = 292\text{KB}$$

Exemplo 5.3

Cálculo de formato de endereço para memórias cache com mapeamento direto.

Considere uma MP com 64MB de capacidade associada a uma memória cache que possui 2K linhas, cada uma com largura de 16 bytes. Determine o formato do endereço para ser interpretado pelo sistema de controle da cache.

Solução

$$\text{MP} = 64\text{MB} = 2^{26} \quad \text{Largura de bloco} = \text{linha} = 16\text{B} = 2^4$$

$$L = 2\text{K} = 2^{11}$$

O formato do endereço possui três campos: tag – linha – byte.

Tag – endereço do bloco desejado, atribuído a uma específica linha. Seu valor em bits é diretamente dependente da quantidade de blocos atribuídos a cada linha, o que pode ser obtido da relação entre o total de blocos, T_b , e o total de linhas, L.

Linha – endereço da linha desejada. Sua largura depende diretamente da quantidade de linhas.

Byte – endereço do byte desejado. Sua largura depende diretamente da quantidade de bytes em cada bloco/linha.

$$T_b = 64\text{MB} / 16\text{B} = 4\text{MB} = 2^{22}.$$

Byte = 4 bits, pois há 16 bytes por bloco.

Linha = 11 bits, pois $L = 2^{11}$.

Tag = 11 bits, pois: $T_b / L = 4 \text{ MB} / 2\text{K} = 2^{22} / 2^{11} = 2^{11} = 2\text{K}$ blocos por linha.

26 bits		
Tag	Linha	Byte
11 bits	11 linhas	4 bits

Exemplo 5.4

Seja uma MP constituída de blocos com largura de 32 bytes, associada a uma cache com 128KB. Em dado instante o processador realiza um acesso, colocando o seguinte endereço (expresso em algarismos hexadecimais): 3FC92B6. Determine qual deverá ser o valor binário da linha que será localizada pelo sistema de controle da cache.

Solução

O endereço completo da MP possui sete algarismos hexadecimais, ou $7 \times 4 \text{ bits} = 28 \text{ bits}$.

Campo byte = 5 bits, pois $2^5 = 32$ e há 32 bytes por bloco/linha.

Quantidade de linhas, L, da cache é igual a $128\text{KB} (\text{total da cache}) / 32\text{B} (\text{larga da linha} = 4\text{K linhas}) = 2^{12}$.

O campo “linha” do endereço tem largura de 12 bits, pois $L = 2^{12}$.

O campo tag terá 11 bits, pois: $28 - 12 - 5 = 11$.

O endereço 3FC92B6 em bits será:

0011 1111 1100 1001 0010 1011 0110

Dividindo pelos campos do endereço, teremos:

0011111110	010010010101	10110
Tag	Linha	Byte

Deste modo, o endereço da linha desejada é: 010010010101.

Mapeamento Associativo

Conforme foi exposto na descrição do mapeamento direto, este método é simples de implementar e funcionar; no entanto, acarreta uma razoável inflexibilidade no uso do mapeamento, pois os blocos são fixamente determinados para uma linha específica. Com isso, pode ocorrer um aumento de faltas (misses) devido justamente a essa inflexibilidade (dois acessos próximos em tempo podem fazer referência a blocos alocados para uma mesma linha, resultando, por exemplo, na retirada de um bloco que acabou de ser trazido da MP e sua utilização em seguida – nova falta).

Para enfrentar este problema e tornar a distribuição dos blocos mais flexível, foi desenvolvido um outro método, exatamente o oposto do mapeamento direto (ver Fig. 5.14). Trata-se do método chamado **mapeamento associativo**.

Por este método não há local fixo na memória cache para alocação de um bloco da MP; quando é requisitado pelo sistema de controle da cache em um acesso o específico bloco pode ser armazenado em qualquer linha, substituindo a que lá estiver armazenada. Neste caso, no entanto, há necessidade de se escolher cuidadosamente qual deverá ser o bloco a ser substituído (pelo menos este problema o mapeamento direto não possui). Há diferentes estratégias para se definir esta escolha, caracterizando diferentes políticas de substituição de linhas, as quais serão tratadas no item 5.4.2.

Além disso, para o sistema determinar se o bloco acessado está armazenado na cache (acerto – hit) ou não (falta – miss), há necessidade de se efetuar a verificação em cada linha, comparando o endereço do bloco com o que está armazenado no campo *tag* da linha. Para que esta verificação seja rápida e, por isso, eficaz, há necessidade de que a comparação seja simultânea a todas as linhas (no exemplo da Fig. 5.14 seriam quatro comparações, mas em memórias bem maiores há um grande consumo de hardware para isso).

A Fig. 5.14 mostra o mesmo exemplo anterior de MP (com 32 células) e cache (com quatro linhas), adotando agora o método associativo, em que cada bloco pode ir para qualquer linha.

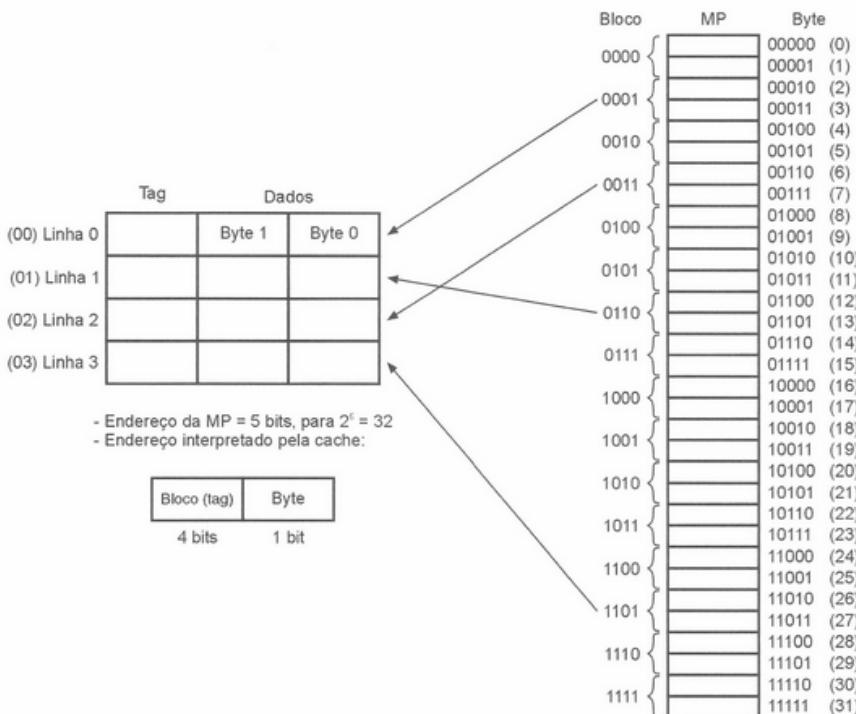


Figura 5.14 Exemplo de organização com mapeamento associativo completo, com MP de 32 células (bytes) e uma cache com quatro linhas de 2 bytes cada.

Nesse caso, o endereço passado pelo processador no BE (5 bits) é interpretado pelo sistema de controle da cache apenas com dois campos: o de identificação do bloco (com 4 bits) e o de endereço do específico byte (ou palavra), com 1 bit.

Então, uma das diferenças em relação ao método anterior (mapeamento direto) reside no tamanho do campo *tag* incluído em cada linha: neste método o campo *tag* possui uma largura igual à do endereço do bloco (no exemplo, são 4 bits), valor sempre maior do que o usado no mapeamento direto (consome-se mais bits neste tipo de mapeamento).

A Fig. 5.15 mostra a organização lógica da cache para o funcionamento do sistema com o método de mapeamento associativo.

Na figura, observa-se a mesma MP e cache, os mesmos decodificadores de endereço de bloco e de byte e os comparadores, sendo no caso quatro, a saída dos quais está conectada a uma porta indicadora de acerto ou falta.

O procedimento básico para o método de mapeamento associativo completo é:

- 1) O processador inicia o acesso pela colocação do endereço no BE, o qual é interceptado pelo sistema de controle da cache.
- 2) O valor do campo bloco do endereço (4 bits mais significativos) é replicado em todos os elementos de comparação, os quais possuem, no outro lado, o valor armazenado em cada campo tag de linha (endereço de bloco).
- 3) Usualmente, esta comparação pode ser realizada por uma porta lógica xor (ver Apêndice B); a saída de todas as comparações (realizadas em um único instante de tempo, qualquer que seja a quantidade de

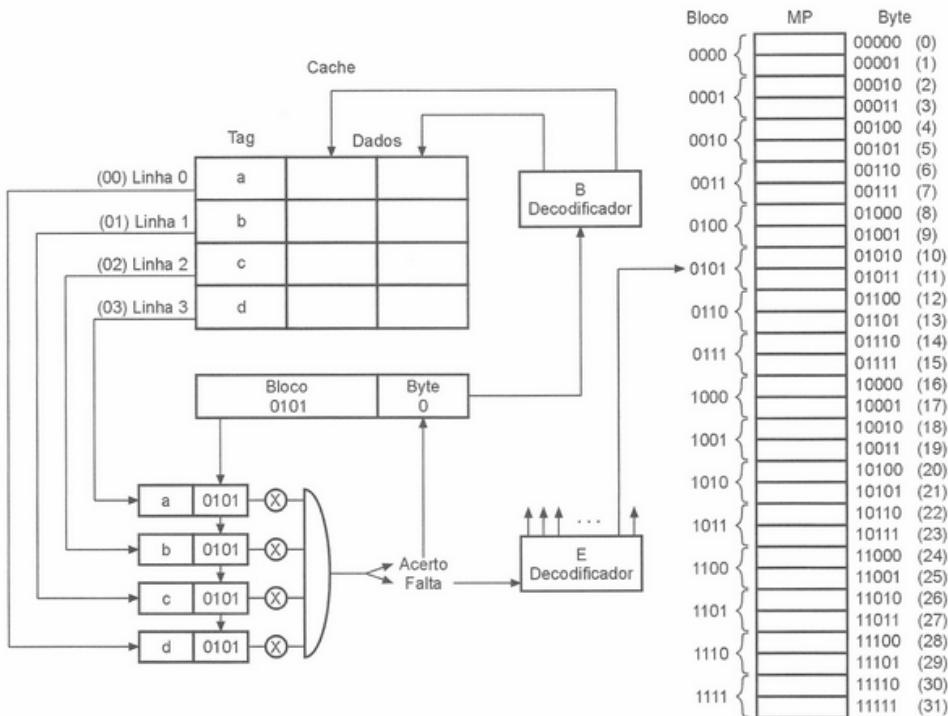


Figura 5.15 Exemplo de acesso à memória cache por meio de mapeamento associativo completo.

comparadores) é entrada de uma porta lógica and, cuja saída acusa se o bloco está (acerto – hit) ou não (falta – miss) na cache.

- 4) Caso haja acerto (hit), o endereço do byte (bit menos significativo) é acionado para se efetuar a transferência para o processador do respectivo valor pelo barramento de dados; caso haja falta, o elemento de acesso à MP é acionado para se efetuar a localização do bloco e, segundo a política estabelecida de substituição de linha (item 5.4.2), transferi-lo para a linha escolhida.

Naturalmente, este método possui a grande vantagem de garantir uma maior flexibilidade na alocação e armazenamento dos blocos, evitando maior quantidade de faltas (hits), porém acarreta um aumento considerável de hardware, com o consequente aumento de custo e complexidade do sistema. Há maior quantidade de bits na cache (conforme podemos verificar pelo Exemplo 5.4), bem como de componentes necessários para cada um dos L comparadores (no exemplo da Fig. 5.14 são quatro comparadores, pois há apenas quatro linhas na cache, mas um sistema real possui caches com milhares de linhas, o que redundaria em milhares de comparadores, respectiva fiação de conexão, portas lógicas e outros elementos, encarecendo bastante o sistema).

Em função desses óbices do mapeamento associativo, bem como considerando-se os inconvenientes do mapeamento direto, mas também as vantagens de ambos os métodos, desenvolveu-se um método alternativo, que procurou agregar as vantagens de ambos os métodos anteriores e eliminar ou minimizar suas desvantagens. Chamou-se a este método de associativo por conjunto; na realidade, é conhecido como associativo por conjunto de N .

Antes de apresentar os detalhes desse método, vamos mostrar alguns exemplos do emprego e funcionamento do método associativo completo.

Exemplo 5.5

Cálculo da quantidade de bits necessários para uma determinada memória cache.

Considere um sistema de computação com uma memória cache de 32KB de capacidade, constituída de linhas com 8 bytes de largura. A MP possui uma capacidade de 16MB.

Solução

O total de bits a serem usados na cache é a soma dos bits de dados (produto da quantidade de linhas pela largura em bits de cada uma) mais os bits usados no campo **bloco** (coluna bloco) existente em cada linha.

Assim, temos:

$$T_c \text{ (total de bits da cache)} = T_d \text{ (total de bits para a parte de dados)} + T_b \text{ (total de bits do campo bloco)}$$

$$T_d = 32\text{KB} \times 8 \text{ bits} = 32 * 1024 * 8 = 262.144 \text{ bits.}$$

$T_b = \text{quantidade de linhas} * \text{largura do campo bloco.}$

$$\text{Quantidade de linhas} = 32\text{KB} / 8 \text{ bytes} = 4\text{KB} = 2^{12}.$$

Largura do campo bloco = larg_b .

Quantidade de blocos = $2^{\text{larg bloco}}$.

$$\text{Quantidade de blocos} = 16 \text{ MB} / 8 \text{ B} = 2 \text{ MB} = 2^{21}.$$

Largura do campo bloco = 21 bits.

$$T_b = 4\text{KB} * 21 \text{ bits} = 84 \text{ Kbits} = 84 * 1024 = .$$

$$T = 262.144 + 86.016 = 348.160 = 340 \text{ Kbits.}$$

Exemplo 5.6

Cálculo de formato de endereço para memórias cache com mapeamento associativo completo.

Considere uma MP com 64MB de capacidade associada a uma memória cache que possui 2K linhas, cada uma com largura de 16 bytes. Determine o formato do endereço para ser interpretado pelo sistema de controle da cache.

Solução

$$\text{MP} = 64\text{MB} = 2^{26}. \quad \text{Largura de bloco} = \text{linha} = 16\text{B} = 2^4.$$

$$L = 2\text{K} = 2^{11}.$$

$$\text{Total de blocos} = 4\text{M blocos} (2^{26} / 2^4 = 2^{22} \text{ ou } 4\text{M})$$

O formato do endereço possui dois campos: bloco – byte

O campo byte é calculado de forma idêntica ao do mapeamento direto, possuindo, pois, 4 bits de largura.

A largura do campo bloco é obtida calculando-se o total de blocos, que é de 4MB = 2^{22} e será, portanto, de 22 bits.

Assim, teremos:

Bloco	Byte
22 bits	4 bits

Exemplo 5.7

Seja uma MP constituída de blocos com largura de 32 bytes, associada a uma cache com 64KB. Em dado instante o processador realiza um acesso, colocando o seguinte endereço (expresso em algarismos hexadecimais): 3FC92B6. Determine qual deverá ser o valor binário do campo bloco que será localizado pelo sistema de controle da cache.

Solução

O endereço completo da MP possui sete algarismos hexadecimais, ou $7 \times 4 \text{ bits} = 28 \text{ bits}$.

Campo byte = 5 bits, pois $2^5 = 32$ e há 32 bytes por bloco/linha.

Quantidade de blocos, B, igual a $2^{28} / 2^5 = 2^{23} = 8\text{M}$.

O campo bloco possui, então, 23 bits de largura.

O endereço 3FC92B6 em bits será:

0011 1111 1100 1001 0010 1011 0110

Dividindo pelos campos do endereço, teremos:

0011111110	010010010101	10110
Bloco	Byte	

Deste modo, o endereço do bloco desejado é: 0011 1111 1100 1001 0010 101.

Mapeamento Associativo por Conjuntos

Esta técnica tenta resolver o problema de conflito de blocos em uma mesma linha (da técnica de *mapeamento direto*) e o problema da técnica de *mapeamento associativo*, relativo à custosa busca e comparação simultâneas do campo *tag* de toda a memória cache.

É, pois, um compromisso entre ambas as técnicas anteriores.

A Fig. 5.16 mostra o mesmo exemplo de MP (32 células) e de memória cache (quatro linhas de 2 bytes cada) adequado ao método associativo por conjunto.

Pode-se observar na figura que o sistema dividiu o espaço de dados da cache em dois conjuntos de duas linhas cada. Cada conjunto é tratado pelo sistema como no método direto, ou seja, cada bloco da cache é previamente dedicado a um conjunto, mas dentro do conjunto o bloco pode ser armazenado em qualquer das linhas do conjunto (método associativo).

Como são 16 blocos, atribuídos fixamente a dois conjuntos, teremos oito blocos por conjunto; conforme se observa na Fig. 5.15, temos oito blocos atribuídos ao conjunto 0 (zero) e oito blocos atribuídos ao conjunto 1 (um) e, por isso, o campo *tag* do endereço possui 3 bits ($2^3 = 8$).

O endereço interpretado pelo sistema de controle da cache possui três campos, identificados da esquerda para a direita:

Tag (3 bits) — Conjunto (1 bit) — Byte (1 bit)

A Fig. 5.17 auxilia na descrição do procedimento de acesso do processador a um byte (palavra) específico, identificando se ocorreu um acerto (e o byte especificado é imediatamente transferido da cache para o processador) ou uma falta (há necessidade de se transferir o bloco desejado para a cache e daí o byte para o processador).

O procedimento básico para acessos em sistemas que usam o método de mapeamento associativo por conjunto é:

- O processador inicia o acesso pela colocação do endereço no BE, o qual é interceptado pelo sistema de controle da cache.

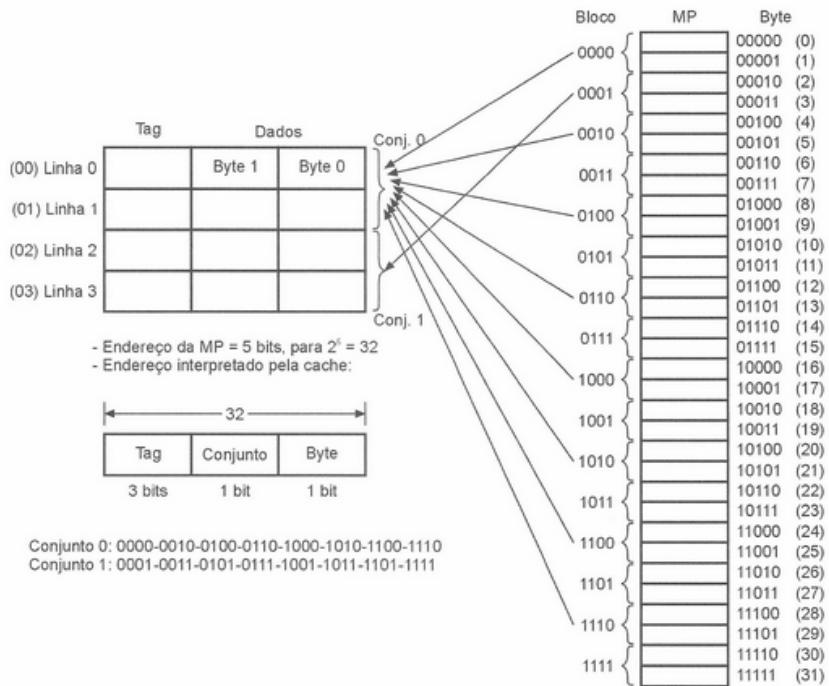


Figura 5.16 Exemplo de organização com mapeamento associativo por conjunto em MP com 32 células (bytes) e uma cache com quatro linhas de 2 conjuntos de duas linhas.

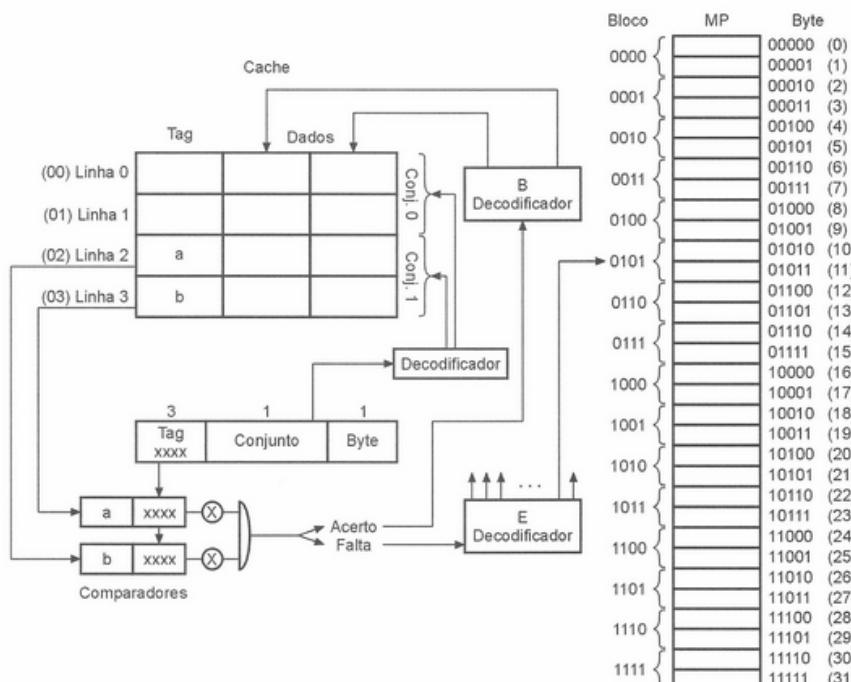


Figura 5.17 Exemplo de acesso à memória cache por meio de mapeamento associativo por conjunto.

- 2) O valor do campo **conjunto** do endereço (no exemplo da Fig. 5.17, é 1 bit — o segundo bit da direita para a esquerda no formato do endereço) é transferido para o decodificador, cuja saída aponta para o conjunto formado por duas linhas da cache (linhas 2 e 3, que correspondem ao conjunto de endereço 1). Trata-se de procedimento exatamente igual ao do mapeamento direto, sendo que, neste caso, cada bloco está associado diretamente a um conjunto e não a uma linha.
- 3) Em seguida, o controle da cache irá localizar qual das linhas se deseja, comparando o valor existente no campo **tag** (3 bits) com o valor existente na coluna **tag** de cada uma das duas linhas do conjunto, exatamente como no método associativo completo).
- 4) Caso haja acerto (hit), o endereço do byte (bit menos significativo) é acionado para se efetuar a transferência para o processador do respectivo valor pelo barramento de dados; caso haja falta, o elemento de acesso à MP é acionado para se efetuar a localização do bloco; primeiramente, é definido o conjunto para onde ele será transferido (previamente definido como no mapeamento direto) e, segundo a política estabelecida de substituição de linha (item 5.4.2), escolhe-se qual linha do conjunto será usada, como no mapeamento associativo completo.

Para finalizar, apresentam-se alguns exemplos do emprego desse método de mapeamento:

Exemplo 5.8

Cálculo da quantidade de bits necessários para uma determinada memória cache, que funciona com mapeamento por conjunto de quatro.

Considere um sistema de computação com uma memória cache de 32KB de capacidade, constituída de linhas com 8 bytes de largura e conjunto de 4. A MP possui uma capacidade de 16MB.

Solução

O total de bits a serem usados na cache é a soma dos bits de dados (produto da quantidade de linhas pela largura em bits de cada uma) mais os bits usados na coluna *tag*, cujo valor existe para cada linha de um conjunto.

Assim, temos:

$$T \text{ (total de bits da cache)} = T_d \text{ (total de bits para a parte de dados)} + T_t \text{ (total de bits dos tags)}.$$

$$T_d = 32\text{KB} \times 8 \text{ bits} = 32 * 1024 * 8 = 262.144 \text{ bits.}$$

$$T_t = \text{quantidade de linhas} * \text{largura do campo tag}.$$

$$\text{Quantidade de linhas} = 32\text{KB} / 8\text{B} = 4\text{K} = 2^{12}.$$

$$\text{Quantidade de conjuntos, } C = 4\text{K} \text{ (quantidade de linhas)} / 4 \text{ (quantidade de linhas por conjunto)} = 1\text{K} = 2^{10}.$$

$$\text{Quantidade de blocos} = 16\text{MB} / 8\text{B} = 2\text{MB} = 2^{21}.$$

$$\text{Campo tag: quantidade de blocos por conjunto} = 2\text{M} / 1\text{K} = 2^{21} / 2^{10} = 2^{11}.$$

A largura do campo tag é, pois, de 11 bits.

$$T_t = 4\text{K} \times 2\text{K} = 2^{12} * 2^{11} = 2^{23} = 8\text{K} = 8 * 1024 = 8192 \text{ bits.}$$

$$T = 262.144 + 8.192 = 270.336 = 264 \text{ Kbits.}$$

Exemplo 5.9

Cálculo de formato de endereço para memórias cache com mapeamento associativo por conjunto.

Considere uma MP com 64MB de capacidade associada a uma memória cache que funciona com mapeamento associativo por conjunto de 4 e que possui 32KB, com linhas de largura de 16 bytes. Determine o formato do endereço para ser interpretado pelo sistema de controle da cache.

Solução

$MP = 64MB = 2^{26}$ Largura de bloco = linha = $16B = 2^4$.

Quantidade de blocos da MP = $64MB / 16B = 4M = 2^{22}$

Quantidade de linhas da cache = $32KB / 16B = 8K = 2^{13}$.

Quantidade de conjuntos = quantidade de linhas / 4 = $8K / 4 = 2K = 2^{11}$.

Quantidade de blocos por conjunto = $4M / 2K = 2^{22} / 2^{11} = 2^{11} = 2K$.

O formato do endereço possui três campos: tag – conjunto – byte.

Campo byte = 4 bits (16 bytes por bloco = 2^4).

Campo conjunto = 11 bits.

Campo tag = 11 bits (devido à quantidade de blocos por conjunto).

Tag	Conjunto	Byte
11 bits	11 bits	4 bits

Exemplo 5.10

Seja uma MP constituída de blocos com largura de 32 bytes, associada a uma cache com 64KB; a cache usa mapeamento por conjunto de 4. Em dado instante o processador realiza um acesso, colocando o seguinte endereço (expresso em algarismos hexadecimais): 3FC92B6. Determine qual deverá ser o valor binário do conjunto que será localizado pelo sistema de controle da cache.

Solução

O endereço completo da MP possui sete algarismos hexadecimais, ou 7×4 bits = 28 bits.

Campo byte = 5 bits, pois $2^5 = 32$ e há 32 bytes por bloco/linha.

Quantidade de linhas, L, da cache é igual a $128KB$ (total da cache) / $32B$ (largura da linha = 4K linhas = 2^{12}).

Quantidade de conjuntos = quantidade de linhas / 4 = $4K / 4 = 1K = 2^{10}$.

O campo “conjunto” do endereço tem largura de 10 bits, pois $C = 1K = 2^{10}$.

O campo tag terá 13 bits, pois: $28 - 10 - 5 = 13$.

O endereço 3FC92B6 em bits será:

0011 1111 1100 1001 0010 1011 0110

Dividindo pelos campos do endereço, teremos:

001111111001	001 0010 101	10110
Tag	Conjunto	Byte

Deste modo, o endereço do conjunto desejado é: 001 0010 101

Atualmente, grande parte dos sistemas emprega mapeamento por conjunto, variando apenas o valor de N, podendo-se encontrar caches com conjuntos de 4, conjuntos de 8 e conjuntos de 16. À medida que a capacidade das caches vai aumentando, tende a aumentar o valor de N, de modo a se manter o equilíbrio das vantagens dos métodos anteriores.

5.4.2 Algoritmos de Substituição de Dados na Cache

O problema se resume em definir qual dos blocos atualmente armazenados na cache deve ser retirado para dar lugar a um novo bloco que está sendo transferido (é bom lembrar que isto é necessário porque todos os quadros da cache estão sempre ocupados, visto que $L << B$). Se o leitor esqueceu o significado das letras, L = quantidade de linhas da cache e B = quantidade de blocos da MP.

A necessidade de decisão sobre substituição de linhas somente ocorre quando se usam os métodos de mapeamento associativo, seja o completo ou por conjunto, pois é neles que se considera a flexibilidade de alocação de blocos em linhas. No método de mapeamento direto esta possibilidade não ocorre, pois previamente são estabelecidas as linhas onde serão armazenados os blocos.

No caso, então, dos métodos associativos, pode-se optar por algum dos seguintes algoritmos de substituição de linhas:

O que não é usado há mais tempo (LRU - *least recently used*), o sistema escolhe para ser substituído o bloco que está mais tempo sem ser utilizado. Ou seja, trata-se de um bloco que o processador não acessa há mais tempo.

O algoritmo LRU pode ser implementado de forma simples em caches associativas por conjuntos de 2, utilizando-se 1 bit adicional em cada linha. Quando uma das duas linhas do conjunto é acessada, o bit é setado (passa para valor 1) e o bit da outra linha passa para valor 0. De modo que na ocasião em que um bloco vai ser armazenado na cache em um determinado conjunto (lembre-se de que a escolha do conjunto não é opcional, pois se faz de modo igual ao método direto), escolhe-se a linha que possui bit igual a 0 (pois é o que não é usado há mais tempo entre os dois do conjunto).

Quando aumenta a associatividade, como, p.ex., para conjunto de 4, o problema também cresce; uma das maneiras para solucionar isso é considerar pares de linhas e, nesse caso, têm-se dois pares, podendo-se estabelecer 1 bit para cada par, funcionando de modo idêntico ao do conjunto de 2. Em seguida, determina-se, no par escolhido, qual a linha que foi usada há mais tempo por meio de outro bit. Com isso, gasta-se mais hardware e mais tempo para processamento dos tais bits.

Com o crescimento da associatividade, o problema se torna bem maior, tornando-se difícil implementar esse processo de forma exata; nesse caso, opta-se por uma estratégia de aproximação na escolha da linha a ser substituída ou por outra modalidade, como a de fila (FIFO), LFU ou escolha aleatória, como descrito a seguir.

Fila, ou seja, o primeiro a chegar é o primeiro a ser atendido (FIFO - *first-in, first-out*). O sistema escolhe o bloco que está há mais tempo na cache (chegou primeiro, *first-in*, é o que sai primeiro, *first-out*), independentemente de estar sendo usado ou não com freqüência pelo processador.

O que tem menos referências (LFU - *least frequently used*). Nesse caso, o sistema de controle escolhe o bloco que tem tido menos acessos por parte do processador (menos referências).

Escolha aleatória – trata-se de escolher aleatoriamente um bloco para ser substituído, independentemente de sua situação no conjunto.

Um estudo realizado sobre memórias cache, baseado em diversas simulações (ver SMIT82), obteve diversas conclusões, entre as quais a de que escolher um bloco aleatoriamente reduz muito pouco o desempenho do sistema em comparação com os demais algoritmos, baseados em algum tipo de uso, e é extremamente simples de implementar. E, mais ainda, quando a associatividade aumenta em associação por conjunto (p. ex., conjuntos de 4 ou de 8), ambos os métodos, LRU e aleatório, quase se equivalem em desempenho, sendo o aleatório muito mais simples e barato em termos de hardware.

5.4.3 Política de Escrita pela Memória Cache

Em sistemas com memória cache, toda vez que o processador realiza uma operação de escrita, esta ocorre imediatamente na cache. Como a cache é apenas uma memória intermediária, não a principal, é necessário que, em algum momento, a MP seja atualizada, para que o sistema mantenha sua correção e integridade.

Antes que um bloco possa ser substituído na cache, é necessário considerar se ele foi ou não alterado na cache e se essas alterações também foram realizadas na MP; caso contrário, isto significa que o bloco da cache

está diferente do da MP, e isto não pode acontecer, pois a MP precisa ser tão corretamente mantida quanto a cache.

Atualmente podem ser encontradas algumas políticas de escrita, cada uma contendo suas vantagens e desvantagens em relação às outras, no que se refere principalmente ao custo e ao desempenho.

O problema é complicado se levarmos em conta algumas ponderações, tais como:

- A memória principal pode ser acessada tanto pela cache quanto por elementos de Entrada e Saída (um dispositivo de acesso direto à memória, DMA, por exemplo). Neste caso, é possível que uma célula da MP tenha sido alterada na cache e ainda não na MP e, assim, esta célula da MP está desatualizada. Ou um elemento de E/S pode ter alterado o conteúdo da célula da MP e, então, a célula da cache é que estará desatualizada.
- A memória principal pode ser acessada por vários processadores, cada um contendo sua memória cache. Neste caso, é possível que o conteúdo de uma célula da MP seja alterado para atender à alteração de uma cache específica de um processador, e as demais caches cujo conteúdo esteja ligado a esta célula estarão desatualizadas.

Entre as técnicas conhecidas, temos:

- *Escrita em ambas (write through)*

Nesta técnica, cada escrita em uma palavra da cache acarreta escrita igual na palavra correspondente da MP, assegurando validade permanente e igual ao conteúdo de ambas as memórias. Caso haja outros módulos UCP/cache, estes alterarão também suas caches correspondentemente.

- *Escrita somente no retorno (write back)*

Esta técnica não realiza atualização simultânea como a anterior, mas sim quando o bloco foi substituído e se houver ocorrido alteração. Em outras palavras, sempre que ocorrer uma alteração do byte na cache o quadro correspondente será marcado através de um bit adicional, que pode ser denominado ATUALIZA, por exemplo. Assim, quando o bloco armazenado no quadro específico foi substituído, o sistema verifica o valor do bit ATUALIZA; caso seja de valor igual a 1, então o bloco é escrito na MP; caso contrário, não.

- *Escrita uma vez (write once)*

É uma técnica apropriada para sistemas de multiprocessadores, cada um com sua cache, que compartilhem um mesmo barramento. Por ela, o controlador da cache escreve atualizando o bloco da MP sempre que o bloco correspondente na cache foi atualizado pela primeira vez. Este fato não só atualiza ao mesmo tempo ambos os blocos (como na técnica *write through*), mas também alerta os demais componentes que compartilham o barramento único. Estes são cientificados de que houve alteração daquela palavra específica e impedem seu uso. Outras alterações (escritas) naquele bloco apenas são realizadas na cache local, pois o bloco somente é atualizado na MP quando foi substituído na cache.

Comparando-se as três políticas, podem ser estabelecidas algumas conclusões:

- Com a política *write through* pode haver uma grande quantidade de escritas desnecessárias na MP, com a natural redução de desempenho do sistema.
- A política *write back* minimiza aquela desvantagem, porém a MP fica potencialmente desatualizada para utilização por outros dispositivos a ela ligados, como o módulo de E/S, o que os obriga a acessar o dado através da cache, o que é um problema.
- A política *write once* pode ser conveniente, mas apenas para sistemas com múltiplos processadores, não sendo ainda muito usada.

O mesmo estudo sobre caches mencionado anteriormente [SMIT82] mostra que a percentagem de escritas na memória é baixa, da ordem de 15%, o que pode apontar para uma simples política *write through*.

5.4.4 Níveis de Cache de Memória RAM

No que se refere às memórias cache de RAM, o aumento crescente da velocidade dos processadores e o compromisso de não se aumentar demasiado o custo das memórias cache (o que ocorreria inevitavelmente com o aumento de sua capacidade) conduziram os projetistas e cientistas a desenvolver caches com diferentes

características de velocidade e capacidade, formando também um sistema hierárquico em níveis ou camadas, a exemplo do próprio sistema global de memória.

Assim é que, atualmente, os fabricantes e montadores de sistemas estabeleceram dois e até três diferentes níveis de memória cache, todos constituídos de memórias SRAM, porém cada um com características e localização diferentes. Estes níveis/camadas podem ser, dependendo do projeto do sistema de computação:

- Nível 1 (Level 1) ou L1, sempre localizada no interior do processador;
- Nível 2 (Level 2) ou L2, sendo localizada, em geral, na placa-mãe do computador, ou seja, externa ao processador. Porém, têm sido lançados processadores em que a cache L2 está localizada no interior da pastilha do processador, separada deste;
- Nível 3 (Level 3) ou L3 – existente em alguns processadores, quando estes possuem L1 e L2 internamente em seu invólucro; nesse caso, é localizada externamente ao processador, na placa-mãe.

Os computadores que possuem múltiplas memórias cache iniciam o acesso pela cache L1, também denominada cache primária. Esta memória, constituída de elementos com tecnologia SRAM e com velocidade de acesso igual à do processador é, por esta razão, a que possui menor tempo de acesso; sempre que o processador buscar um dado/instrução e obtiver um acerto (hit), a velocidade de transferência será a do processador, com um tempo extremamente rápido.

O funcionamento do sistema é sempre o mesmo de casos anteriores: a memória de nível mais baixo é a que tem maior velocidade (e também maior custo) e menor capacidade. O processador sempre procura o dado/instrução na memória de menor nível; não encontrando na L1, buscará na L2 (se houver uma); desta para a memória DRAM e, finalmente, não encontrando na DRAM irá buscar o dado/instrução no disco.

Apesar das memórias cache L1 serem sempre internas, isto é, construídas com os mesmos elementos do processador, pode haver diferenças de arquitetura entre elas que modificam seu desempenho. É o caso, por exemplo, de um processador com cache de maior tamanho que outro ou se um processador possui ou não a cache dividida, cache L1 de instruções e cache L1 de dados; a maior capacidade e/ou a divisão da cache interna, L1, podem tornar um processador mais rápido que outro, até mesmo se esse outro possuir um processador mais rápido. Por exemplo, os processadores Pentium possuem cache dividida, normalmente de 32KB (16KB para dados e 16KB para instruções, outros possuem 24KB para instruções e 8KB para dados), enquanto alguns AMD Athlon possuem cache L1 unificada e outros dividida.

A Tabela 5.1 apresenta alguns processadores e suas caches primárias, L1. As caches podem ser organizadas de forma unificada (nesse caso, na coluna Tamanho consta um único valor) ou dividida (nesse caso, mostram-se os dois valores, iniciando-se cada um pelas iniciais I, cache de instrução, e D, de cache de dados).

Tabela 5.1 Exemplos de Caches em Processadores

Processadores	Fabricante	Tamanho da Cache
80486	Intel	8 KB
Athlon K7	AMD	I-64KB e D-64KB
Pentium	Intel	I-8KB e D-8KB
Pentium MMX	Intel	I-8KB e D-24KB
Pentium PRO	Intel	I-8KB e D-8KB
PowerPC 601	Motorola/IBM	32 KB
Pentium III	Intel	I-8KB e D-24KB
Pentium 4	Intel	I-8KB e D-8KB
Athlon 64	AMD	I-64KB e D-64KB
Power 5	IBM	64KB
Itanium	Intel	I-16KB e D-16KB

A memória cache L2 ou cache externa, ou ainda denominada cache secundária, é usualmente instalada na placa-mãe do sistema, externamente ao processador, mas ainda constituida de elementos com tecnologia SRAM. Naturalmente, possui menor velocidade de transferência do que a cache L1, visto que opera na velocidade do barramento de dados da placa-mãe, sempre mais lento que a velocidade do processador. No entanto, possui bem maior capacidade de armazenamento que as memórias cache L1, em uma faixa de 64KB a 4MB (é possível que ao ler este texto o leitor já tenha tido informação sobre valores ainda maiores, o que é perfeitamente natural em face da velocidade de avanço da tecnologia desses componentes).

Apesar de a cache L2 ser também conhecida como cache externa, alguns processadores têm sido fabricados com a cache L2 instalada internamente na pastilha do processador, embora em um invólucro separado deste, mas não podendo mais ser chamada de cache externa. É o caso, por exemplo, do Pentium Pro, Pentium II, III e 4.

Alguns modelos de processadores AMD Athlon possuem cache L2 interna, integrada ao processador, e foi desenvolvido para permitir um terceiro nível de memória cache, denominado L3, externo, com controlador na placa-mãe.

5.4.5 Tamanho da Memória Cache

Já foi explicado que uma memória cache só é produtiva, isto é, o desempenho do sistema cresce bastante com seu uso se, durante a execução de um programa, ocorrerem muito mais acertos (o dado requerido pelo processador se encontra na cache) do que faltas (o dado requerido pelo processador não se encontra na cache e deve, então, ser transferido da memória DRAM (MP) para a cache (seja L1 ou L2), para desta ir para o processador), e se isto for obtido através do emprego de uma memória cache de tamanho relativamente pequeno, para que o custo do sistema seja baixo.

A definição da faixa de tamanho adequada para uma cache depende de uma série de fatores específicos de um certo sistema, tais como:

- tamanho da memória principal,
- relação acertos/faltas,
- tempo de acesso da MP,
- custo médio por bit, da MP, e da memória cache L1 ou L2,
- tempo de acesso da cache L1 ou L2,
- natureza do programa em execução (princípio da localidade).

Vários estudos têm sido realizados a respeito, podendo-se considerar uma faixa entre 32K e 256 Kbytes para memórias cache do tipo L1 (cache primária ou interna) e de 64K até 4 Mbytes para memórias cache do tipo L2 (cache secundária). Naturalmente, estes valores não são definitivos nem estáticos, o que significa que, logo adiante, pode-se estar trabalhando com outras faixas de valores.

5.4.6 Largura de Linha da Memória Cache

A escolha da largura ótima de uma cache é tão difícil quanto qualquer outra escolha neste tipo de assunto, devido às diversas variáveis envolvidas. Normalmente se tentam valores estimados em laboratório, que a prática e novos testes aperfeiçoam. É comum se encontrar caches com linhas de 8 a 32 bytes de largura.

O tamanho da linha de uma cache está naturalmente associado ao princípio da localidade espacial; como se sabe que acessando um determinado endereço os acessos seguintes têm grande probabilidade de serem endereços contíguos, então uma linha com vários bytes atende a este princípio. Ao buscar um dado, o sistema busca este e vários outros em sequência, que serão provavelmente acessados em seguida (mas agora em tempo bem mais rápido, pois já estarão na cache).

Nesse caso deveríamos imaginar que crescendo a largura da linha (muitos bytes) o princípio da localidade espacial seria mais bem atendido, o que não é necessariamente verdade. Realmente, aumentando até uma certa (não específica nem calculável) quantidade de bytes o tamanho da linha vamos aumentando a taxa de

acertos. Porém, a manutenção da seqüência de execução de um programa não é permanente, pois vez por outra aparece um desvio, p.ex., em um comando IF-THEN- ELSE, e neste caso alguns dos bytes a mais existentes na linha estarão sendo desnecessários, acarretando aumento de faltas.

Uma outra possível desvantagem de uma largura de linha excessiva é a redução da quantidade de linhas, $L = \text{Tamanho da cache} / \text{largura da linha}$. Mantido L , se aumentarmos largura da linha, reduz-se a quantidade de linhas e isso aumentará a quantidade de blocos para cada linha, p.ex., aumentando a probabilidade de faltas.

EXERCÍCIOS

- 1) Uma memória cache associativa por conjunto consiste em 256 linhas divididas em conjuntos de duas linhas cada. A memória principal contém 4K blocos de 128 bytes cada um. Mostre o formato de um endereço de MP e da cache.
- 2) Em que circunstâncias uma cache que funciona com mapeamento associativo por conjunto pode ser considerada igual à cache que funciona com mapeamento direto?
- 3) Considere um sistema de computação que possui uma memória principal (RAM) com capacidade máxima de endereçamento de 64K células, e que cada célula armazena um byte de informação. Para criar um sistema de controle e funcionamento da sua memória cache, a memória RAM é constituída de blocos de 8 bytes cada. A memória cache do sistema é do tipo mapeamento direto, contendo 32 linhas. Pergunta-se:
 - a) Como seria organizado o endereço da MP (RAM) em termos de tag, número da linha e do byte dentro de uma linha?
 - b) Em que linha estaria contido o byte armazenado no seguinte endereço da MP: 0001 0001 0001 1011?
 - c) Qual é a capacidade da memória cache em bytes?
- 4) Como a memória principal pode ser organizada de modo diferente, tendo uma única estrutura física, e quais são essas diferentes formas de organização daquela memória?
- 5) Explique por que não é conveniente escolher valores muito grandes de tamanho de linha de caches e porque este tamanho é sempre muito maior que o de uma célula de memória principal.
- 6) Ao se verificar a organização de uma memória cache e da MP de um sistema de computação, observa-se que a cache tem uma capacidade de armazenamento muito menor que a MP e, ainda assim, sabe-se que em 100 acessos do processador à cache ele obtém cerca de 95 a 98% de acertos. Por quê?
- 7) Por que é necessário se estabelecer uma política para substituição de linhas para os métodos de mapeamento associativo e não para o método direto?
- 8) Considere uma MP que possui 4K blocos de 128 células de 1 byte cada e uma memória cache do tipo associativa por conjunto que possui 64 linhas divididas em conjuntos de quatro linhas. Qual deverá ser o formato do campo de endereçamento para acesso à MP e à memória cache?
- 9) Considere um sistema de computação que possui uma memória principal organizada em células de 1 byte cada e apenas uma memória cache externa, organizada com mapeamento direto, sendo cada linha de 32 bytes. Em um dado instante, o processador inicia um acesso colocando no BE comum o endereço hexadecimal: 5D7A9F2. Sabendo-se que neste sistema cada linha da cache tem atribuídos 512 blocos da MP, pergunta-se:
 - a) Qual deverá ser a largura do BE do sistema?
 - b) Qual foi a linha acessada pelo processador?
- 10) Qual é a diferença entre caches unificadas e divididas? Que categoria de cache em um sistema multicache usa um ou outro tipo daquela organização?
- 11) Considere uma memória cache organizada com mapeamento associativo por conjunto, sendo cada conjunto de quatro linhas. A MP tem uma capacidade de armazenar 64MB, sendo organizada byte a byte, e o sistema transfere de cada vez (MP \leftrightarrow cache) 32 bytes. Considerando que a capacidade da cache é de 64KB, mostre como deve ser o formato dos campos de endereço para a memória cache.

- 12) Por que a memória cache é um dispositivo de armazenamento volátil?
- 13) Considere um sistema de armazenamento constituído de uma memória principal, que é endereçada por byte e que tem uma capacidade de 256KB, sendo organizada em blocos de 16 bytes de largura. Considerando que se usa neste sistema o método de mapeamento direto para uma cache constituída de 128 linhas, pergunta-se:
- Qual deverá ser o formato do endereço a ser interpretado pelo sistema de controle da cache, indicando a largura de cada campo?
 - Em que linhas deverão ser armazenados os bytes que possuam os seguintes endereços:
 - 1011 1110 0010 1001 1101 0000 1100?
 - 0001 1010 0011 0001 0111 1000 1111?
 - Qual deverá ser o total de bits consumido nessa cache?
 - Qual deverá ser o endereço do bloco que contém um byte com o seguinte endereço:
0010 1110 1001 0001 1110 0011 1110?
- 14) Considere um sistema de armazenamento com MP endereçada por byte, onde cada endereço tem uma largura de 30 bits e uma memória cache constituída de 256KB, possuindo L linhas com largura de 16 bytes. Calcule o total de bits da memória cache para um dos métodos de mapeamento: direto, associativo e associativo por conjunto.
- 15) Considere um sistema de armazenamento onde a MP é endereçada por byte, que utiliza o método de mapeamento direto na sua cache e onde o formato dos endereços interpretados pelo sistema de controle é:

Tag	Linha	Byte
8 bits	12 bits	4 bits

Pergunta-se:

- Qual a capacidade de armazenamento da MP, em bytes?
 - Quantas linhas possui a memória cache?
 - Qual é a largura de cada bloco/linha?
 - Qual é a quantidade de blocos atribuído a cada linha da cache?
- 16) Supondo que esse sistema utilize o método de mapeamento associativo por conjunto de 4 e que o formato do endereço de cache é:

Tag	Conjunto	Byte
8 bits	8 bits	4 bits

pergunta-se:

- Qual a capacidade de armazenamento da MP, em bytes?
- Quantas linhas possui a memória cache?
- Quantos conjuntos possui a memória cache?
- Qual é a largura de cada bloco/linha?
- Qual é a quantidade de blocos atribuído a cada linha da cache?

Mario A. Monteiro

Introdução
à
Organização
de
Computadores

Q U I N T A E D I Ç Ã O

LTC