

Capítulo 4

O Sub-Sistema de Memória

Os capítulos anteriores discutiram conceitos no nível de arquitetura de processador. Com este conhecimento, podemos agora abordar o nível superior, o de arquitetura de computadores.

A organização básica de um computador inclui o processador, a memória principal e as *interfaces* de entrada/saída, interconectados por um barramento. Na realidade, a memória principal é apenas um dos componentes dentro do **sub-sistema de memória** de um computador.

Este capítulo introduz os componentes normalmente encontrados no sub-sistema de memória de um computador pessoal ou de uma estação de trabalho. Inicialmente, é analisado em detalhe o mecanismo de acesso do processador à memória principal. Em seguida, são apresentados os principais tipos e características de dispositivos de memória. Finalmente, são apresentados os conceitos de memória *cache* e memória virtual.

4.1 Introdução

Todos os programas que são executados nos computadores apresentam um princípio denominado **localidade de referência**. Este princípio estabelece que os programas acessam uma parte relativamente pequena do seu espaço de endereçamento em um dado instante de tempo. Existem dois tipos diferentes de localidade: **temporal** e **espacial**.

A **localidade temporal** estabelece que se um item é referenciado, ele tende a ser referenciado novamente dentro de um espaço de tempo pequeno. Já a **localidade espacial** estabelece que se um item é referenciado, outros itens cujos endereços estejam próximos dele tendem a ser referenciados rapidamente.

Estas propriedades aparecem nos programas pela sua própria natureza. Como exemplo, podemos citar os *loops* onde as instruções e os dados tendem a ser acessados de modo repetitivo (localidade temporal). Além disso, se as instruções são acessadas sequencialmente os programas apresentam localidade espacial.

O princípio da localidade é utilizado para implementar o sub-sistema de memória de um computador. Esta organização é denominada **hierarquia de memória**.

A hierarquia de memória prevê a existência de vários níveis de memória. Cada um destes níveis possui capacidades de armazenamento e tempos de acesso diferentes. As memórias mais rápidas são também as mais caras, considerando o custo por *bit* armazenado.

Já as memórias mais lentas são menos custosas e possuem maior capacidade de armazenamento. A Figura 4.1 mostra uma hierarquia de memória genérica com n níveis.

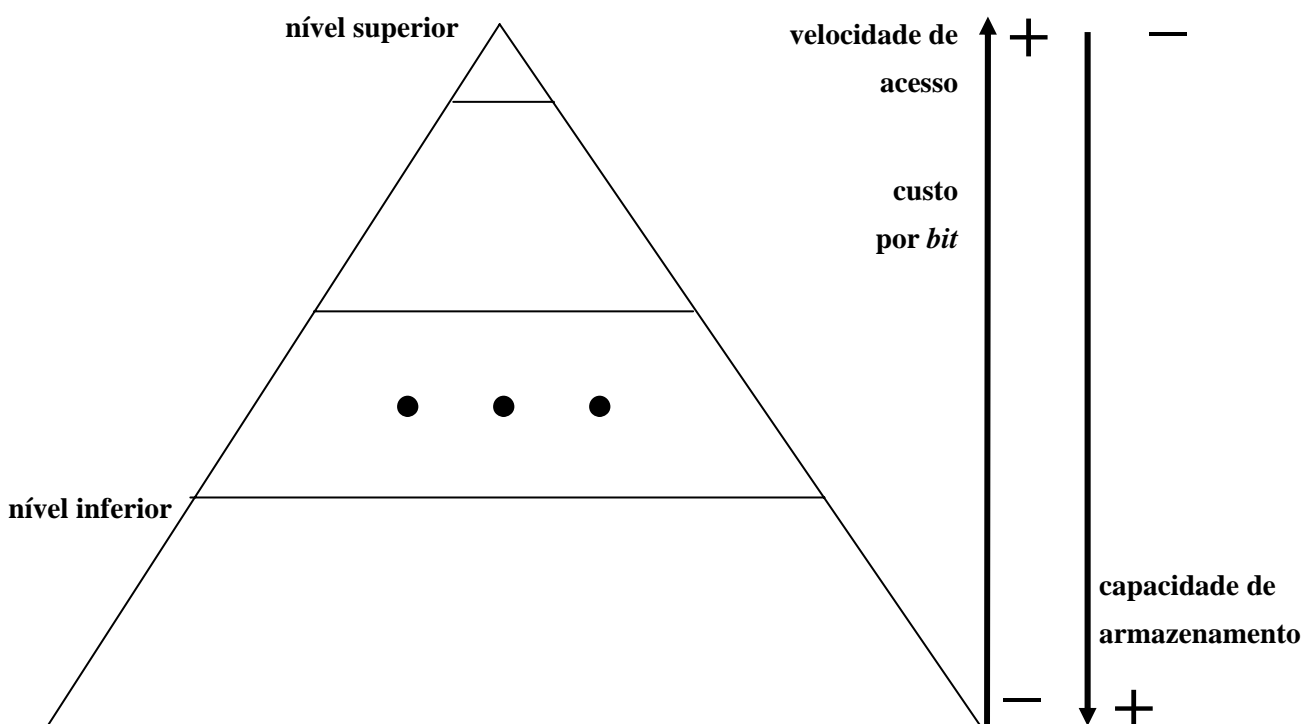


Figura 4.1. Hierarquia de memória genérica.

4.2. Hierarquia de Memória

Uma hierarquia de memória pode ser formada por diversos níveis. Para qualquer quantidade de níveis os dados são copiados entre dois níveis adjacentes. Os níveis superiores são os mais próximos ao processador e os níveis inferiores são mais distantes do processador.

A unidade mínima de transferência de dados entre dois níveis adjacentes é chamada de **bloco**. Se a informação solicitada estiver contida num nível ocorre um **acerto**. Caso contrário, estando a informação ausente, dizemos que ocorreu uma **falha**. Neste caso, o nível inferior da hierarquia é acessado para que seja possível recuperar o bloco com a informação solicitada.

A medida de desempenho da hierarquia de memória é dada pela **taxa de acertos**, que corresponde aos dados solicitados que estão presentes naquele nível da memória. A **taxa de falhas** corresponde aos dados não encontrados naquele nível da memória. A **penalidade** por falha corresponde ao somatório dos tempos necessários para identificar que o dado não está presente no nível procurado, acessar o nível inferior e trazer o bloco para o nível superior da hierarquia e enviar o dado ao processador.

Num sistema de computador tanto a CPU quanto o sub-sistema de E/S interagem com a memória. Dado que cada conteúdo (palavra ou *byte*) armazenado na memória possui seu próprio endereço, a interação é feita através de uma seqüência de leituras e escritas a endereços de memória específicos.

Num sistema de computador existem tipos de memória com diferentes características que formam uma hierarquia, como a heirarquia hipotética ilustrada na Figura 4.2.

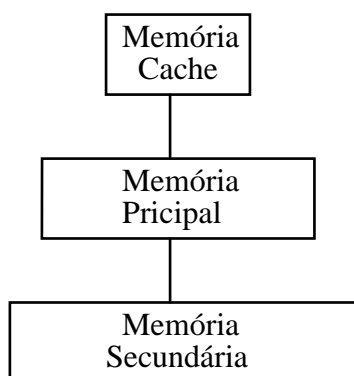


Figura 4.2 Hierarquia de memória.

A **hierarquia de memória** pode ser analisada segundo suas capacidades de armazenamento, custo por *bit* e tempo de acesso. Partindo do nível mais inferior da Figura 4.2, as memórias secundárias são capazes de armazenar uma grande quantidade de informação, seu custo por *bit* é menor e o seu tempo de acesso é relativamente maior do que as memórias dos níveis superiores.

No segundo nível, está a memória principal. Ela é capaz de armazenar uma quantidade menor de informação, seu custo por *bit* é maior e seu tempo de acesso é menor do que as memórias secundárias.

No nível mais superior estão as memórias *cache*. Estas memórias são as mais rápidas e com o maior custo por *bit*. Por serem muito caras as memórias *cache* são pequenas. Isto é, são as que têm menor capacidade de armazenamento em relação as demais. Qualquer que seja a quantidade de níveis utilizados na hierarquia de memória os dados são sempre copiados entre os níveis adjacentes.

Existem diversas políticas de gerenciamento para a memória primária ou memória principal. O gerenciamento de memória usa regras ou políticas de gerenciamento para:

- **busca.** Determina quando um bloco de informação deve ser transferido da memória secundária para a memória principal;
- **armazenamento.** Determina onde o bloco de informação deve ser colocado na memória principal; e
- **substituição.** Determina qual bloco de informação deve ser substituído por um novo bloco.

Existem muitos esquemas diferentes de gerenciamento de memória. A seleção de um esquema, em particular, depende de muitos fatores, mas em especial depende do suporte de *hardware* do sistema.

4.3 A Interação entre Processador e Memória Principal

O componente básico da memória principal é chamado **célula de *bit***. A célula de *bit* é um circuito eletrônico que armazena um *bit* de informação. Em uma referência à memória, o processador não acessa o conteúdo de apenas uma célula de *bit*, mas sim a informação armazenada em um grupo de células de *bits*. O menor conjunto de células de *bits* que é acessado pelo processador é chamado **locação de memória**. Na maioria dos computadores, uma locação de memória é formada por 8 células de *bits*. Como 8 *bits* formam o chamado *byte*, uma locação de memória também é conhecida informalmente como **byte de memória**.

Em um acesso, o processador deve fornecer à memória principal uma identificação da locação onde será feito o acesso. Esta identificação é simplesmente um número chamado **endereço de memória**. Ao receber um endereço de memória, a memória principal seleciona a locação correspondente e fornece ao processador a informação contida naquela locação, caso seja um **acesso de leitura**. No caso de um **acesso de escrita**, a memória principal armazena na locação indicada pelo endereço a informação fornecida pelo processador. O processador e a memória principal estão interligados através de três barramentos distintos, como mostra a Figura 4.3.

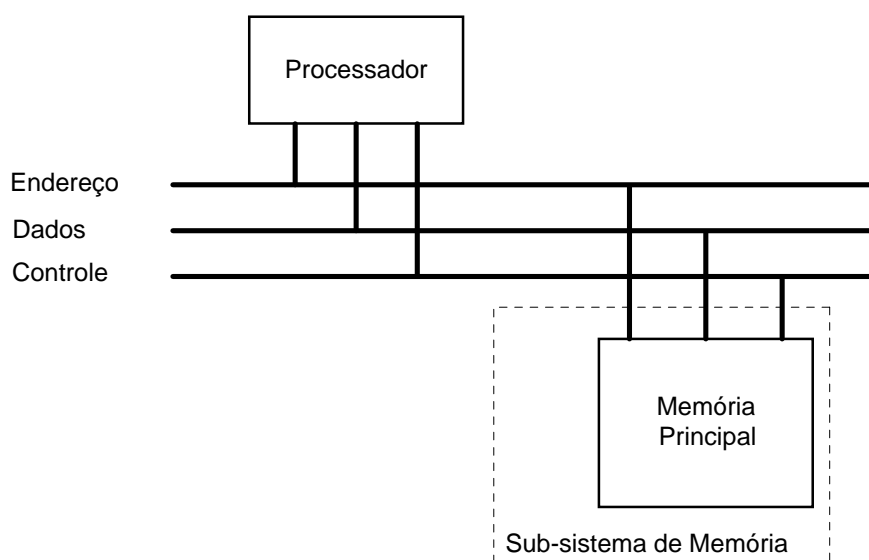


Figura 4.3. Interligação entre processador e memória principal.

O **barramento de endereço** é a via através da qual o processador transmite para a memória principal o endereço da locação onde será feito o acesso. O **barramento de dados** é a via através da qual o conteúdo da locação é transferido entre o processador e a memória principal. Finalmente, o **barramento de controle** é formado por diversos sinais através dos quais o processador controla o acesso à memória, indicando, por exemplo, se o acesso é de leitura ou de escrita.

A largura, em número de *bits*, dos barramentos de endereço e de dados varia entre computadores, sendo determinada basicamente pelo processador usado no sistema.

Um sistema com um barramento de endereço com largura de n *bits* pode endereçar até 2^n locações de memória distintas. No entanto, isto não significa necessariamente que o sistema possui esta capacidade de memória principal instalada. Alguns sistemas como, por exemplo, as estações de trabalho e os servidores, permitem até 4 *Gbytes* de memória principal. A capacidade máxima de memória principal que pode ser instalada em um sistema é determinada pelo compromisso entre custo e desempenho característico do sistema.

Ciclo de Barramento

Como discutido anteriormente, um acesso à memória envolve a seleção de uma locação de memória, a ativação de sinais de controle e a troca de informações entre o processador e a memória principal. A sequência de eventos que acontecem durante um acesso do processador à memória principal é chamado de **ciclo de barramento** (*bus cycle*). Este ciclo é descrito a partir deste ponto em maiores detalhes.

Para descrever um ciclo de barramento, é conveniente usar uma representação gráfica denominada **diagrama de tempo**. O diagrama de tempo indica as alterações no estado dos barramentos durante um acesso à memória. A Figura 4.4 mostra o diagrama de tempo do ciclo de barramento em um acesso de leitura na memória principal.

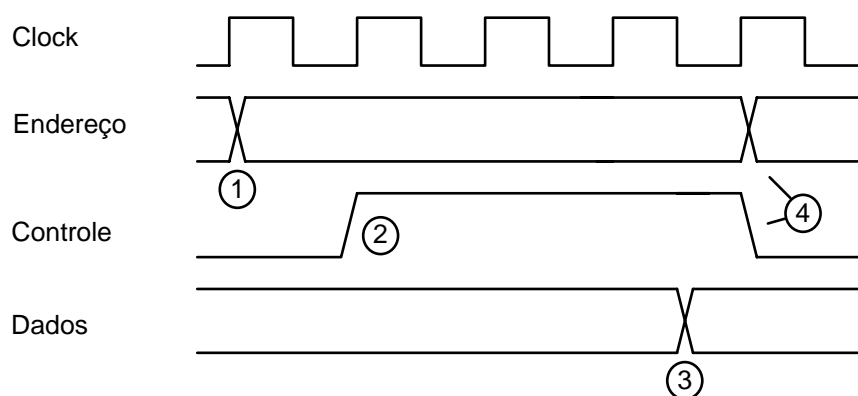

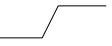
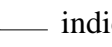


Figura 4.4. Ciclo de barramento em um acesso de leitura.

No diagrama acima, a representação  indica que em um certo momento o barramento mudou de estado, com alguns *bits* mudando do valor 0 para 1 ou vice-versa. A representação  indica que o sinal mudou do estado lógico 0 para o estado lógico 1, enquanto  indica uma transição do sinal do estado lógico 1 para o estado lógico 0. Como indica o diagrama de tempo, as mudanças nos estados dos barramentos estão sincronizados com o sinal de *clock*.

O processador inicia o ciclo de barramento colocando o endereço da localização de memória a ser acessada no barramento de endereço (1). No início do próximo ciclo de *clock* o processador ativa um sinal de controle indicando uma operação de leitura (2). O endereço e o sinal de controle chegam à memória, que após algum tempo coloca o conteúdo da localização de memória endereçada no barramento de dados (3). No início do quarto ciclo de *clock*, o processador captura a informação presente no barramento de dados e finaliza o ciclo de barramento, desativando o sinal de controle e retirando o endereço (4).

A Figura 4.5 mostra o diagrama de tempo do ciclo de barramento em um acesso de escrita na memória. Novamente, o ciclo de barramento inicia-se com o processador colocando o endereço da localização de memória no barramento de endereço (1). Além disso, o processador também coloca a informação a ser armazenada no barramento de dados (2). No próximo ciclo de *clock*, o processador ativa um sinal de controle indicando uma operação de escrita na memória (3). Ao receber o sinal de escrita, a memória armazena o dado na localização endereçada. O processador finaliza o ciclo de barramento retirando o endereço e o dado dos respectivos barramentos, e desativando o sinal de controle (4).

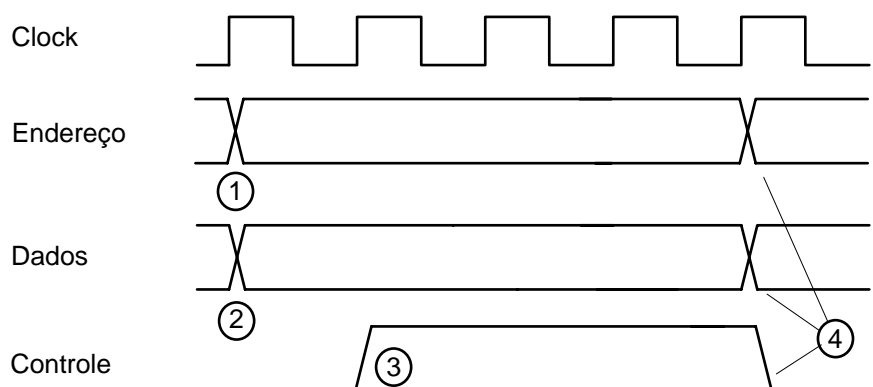


Figura 4.5. *Ciclo de barramento em um acesso de escrita.*

Nos diagramas mostrados, o número de ciclos de *clock* que separam os eventos ao longo de um ciclo de barramento são hipotéticos. O número de ciclos de *clock* consumidos no ciclo de barramento varia entre os processadores. No entanto, a seqüência de eventos mostrada nos diagramas de fato ocorrem para a maioria dos processadores.

Estados de Espera

A “velocidade” da memória principal é medida pelo **tempo de acesso** dos dispositivos de memória (circuitos integrados) usados em sua implementação. O tempo de acesso é o intervalo de tempo entre o instante em que a memória recebe o endereço até o instante em que a memória fornece a informação endereçada. Na Figura 4.5 é representado o caso onde o tempo de acesso da memória é equivalente à cerca de 3,5 ciclos de *clock*.

Um ponto importante a observar quanto ao ciclo de barramento é que o seu tamanho, em termos do número de ciclos de *clock*, é a princípio determinado pelo processador. Caso não haja uma solicitação externa quanto ao tamanho apropriado do ciclo de barramento, o processador sempre executará um **ciclo de barramento padrão** com um certo número de ciclos. Nos exemplos hipotéticos das Figuras 4.4 e 4.5, o ciclo de barramento padrão possui um tamanho de 4 ciclos de *clock*.

A questão que se coloca é: o que acontece se o processador possui um ciclo de barramento padrão cujo tamanho é menor que o tempo de acesso da memória principal? O diagrama de tempo na Figura 4.6(a) representa esta situação.

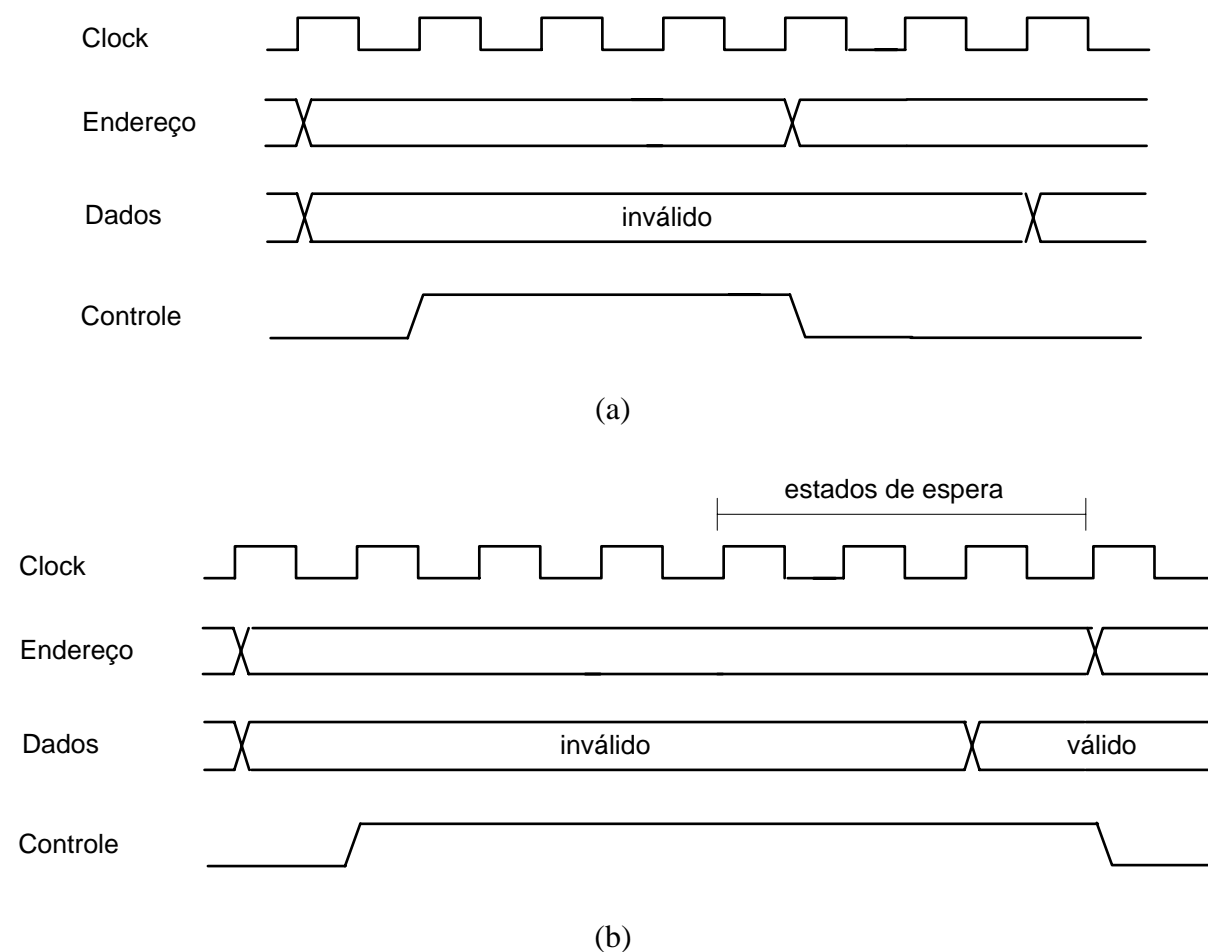


Figura 4.6. Os estados de espera no ciclo de barramento.

Na Figura 4.6(a) está representado o caso onde o ciclo de barramento padrão continua com 4 ciclos de *clock*. Porém, agora o processador se encontra em um sistema cuja memória principal possui um tempo de acesso de 6 ciclos de *clock*. Ao final do ciclo de barramento, o processador captura uma informação no barramento de dados que naquele momento é inválida. Isto porque a memória forneceria a informação endereçada apenas dois ciclos de *clock* depois. Nesta situação, o sistema opera incorretamente.

Em geral, os processadores oferecem um mecanismo que permite compatibilizar o tamanho do ciclo de barramento ao tempo de acesso da memória. O processador possui um sinal de entrada que, ao ser ativado, faz com que o tamanho do ciclo de barramento seja estendido pela introdução de ciclos de *clock* extras. Estes ciclos de *clock* adicionais são denominados **estados de espera**.

A Figura 4.6(b) mostra um ciclo de barramento com estados de espera. Quando a memória detecta o início de um ciclo de barramento, a memória principal envia um sinal solicitando ao processador que o ciclo de barramento seja estendido. O processador não finaliza o ciclo de barramento enquanto esta solicitação for mantida. Ao colocar a informação endereçada no barramento de dados, a memória principal retira a solicitação de espera. O processador captura a informação, agora válida, presente no barramento de dados, e finaliza o ciclo de barramento. No exemplo mostrado foram introduzidos três ciclos de espera.

Os computadores do tipo IBM PC oferecem este recurso dos estados de espera. O número de estados de espera é um dos parâmetros de configuração do sistema, e pode ser escolhido através da *interface* de configuração do sistema. Note que o uso deste recurso afeta o desempenho do sistema, já que o tempo consumido nos acessos à memória torna-se maior. No entanto, as memórias *cache*, que são discutidas mais adiante neste capítulo, atenuam em grande parte o efeito do aumento do ciclo de barramento sobre o desempenho.

4.4 Tipos de Dispositivos de Memória

No sub-sistema de memória de um computador é possível encontrar três diferentes tipos de dispositivos de memória. O primeiro tipo é a **memória programável somente de leitura**, mais conhecida como PROM (*Programmable Read Only Memory*). Este tipo de dispositivo permite apenas acessos de leitura. As informações armazenadas são atribuídas pelo fabricante do sistema, e não podem ser modificadas após o seu armazenamento.

O segundo tipo é a **memória programável e apagável somente de leitura**, ou simplesmente EPROM (*Erasable Programmable Read Only Memory*). Assim como no caso da PROM, o processador realiza apenas acessos de leitura a uma EPROM. No entanto, ao contrário da PROM, o conteúdo de uma memória EPROM pode ser apagado, e a memória pode ser novamente usada para armazenar um novo conjunto de informações.

As memórias PROM ou EPROM são usadas para armazenar o *firmware* do computador. Em geral, o *firmware* é formado por sub-rotinas usadas pelo sistema operacional, e que interagem diretamente com o *hardware* do computador. O BIOS (*Basic Input/Output System*) em sistemas computadores do tipo IBM PC é um exemplo de *firmware*. As rotinas que formam o BIOS são armazenadas em memórias do tipo EPROM.

O terceiro tipo de dispositivo de memória encontrado em um computador é a memória de leitura e escrita, ou RAM (*Random Access Memory*). Como a própria denominação indica, o processador pode efetuar acessos de leitura e escrita a um dispositivo RAM. As memórias deste tipo são usadas para armazenar as instruções e dados de um programa em execução. Existem dois tipos de memória RAM: as memórias RAM estáticas, ou SRAM (*Static RAM*) e as memórias RAM dinâmicas, ou DRAM (*Dynamic RAM*). Estes dois tipos de memórias RAM diferem quanto à capacidade de armazenamento e ao tempo de acesso.

Para memórias fabricadas com a mesma tecnologia, a capacidade de um dispositivo DRAM é até 16 vezes maior que a de um dispositivo SRAM. Esta diferença na capacidade de armazenamento deve-se ao modo como uma célula de *bit* é implementada. Nas memórias DRAM, a célula de *bit* é implementada por um circuito eletrônico que ocupa uma área de integração menor que a ocupada pelo circuito usado nas memórias SRAM. Como a área por célula de *bit* é menor, para a mesma área total de integração o número total de células de *bit* em uma DRAM é maior.

No entanto, as memórias SRAM apresentam uma vantagem sobre as memórias DRAM no que se refere ao tempo de acesso. Para a mesma tecnologia, o tempo de acesso de uma SRAM é até 4 vezes menor que o tempo de acesso de uma DRAM. Melhorias no tempo de acesso dos dispositivos DRAM não acontecem na mesma taxa que o aumento observado na sua capacidade de armazenamento.

Os dispositivos DRAM são utilizados em sistemas de baixo e médio custo, tais como computadores pessoais e estações de trabalho. Com este tipo de dispositivo é possível dotar um sistema com uma capacidade de memória principal elevada, sem aumentar significativamente o seu custo. No entanto, o uso de dispositivos DRAM favorece a capacidade de armazenamento da memória principal em detrimento do seu tempo de acesso, comprometendo o desempenho do sistema.

As classes de aplicações como processamento gráfico e multimídia necessitam que o desempenho seja um fator cada vez mais importante na faixa de computadores pessoais e estações de trabalho. A solução para obter o compromisso desejado entre capacidade de armazenamento, desempenho e custo encontra-se no uso apropriado de tecnologias de memória com diferentes relações entre estes três fatores. Esta é a idéia por detrás das memórias *cache*, cujo conceito e operação são examinados a seguir.

4.5 Memórias Cache

A **memória cache** é uma pequena memória inserida entre o processador e a memória principal, conforme mostra a Figura 4.7.

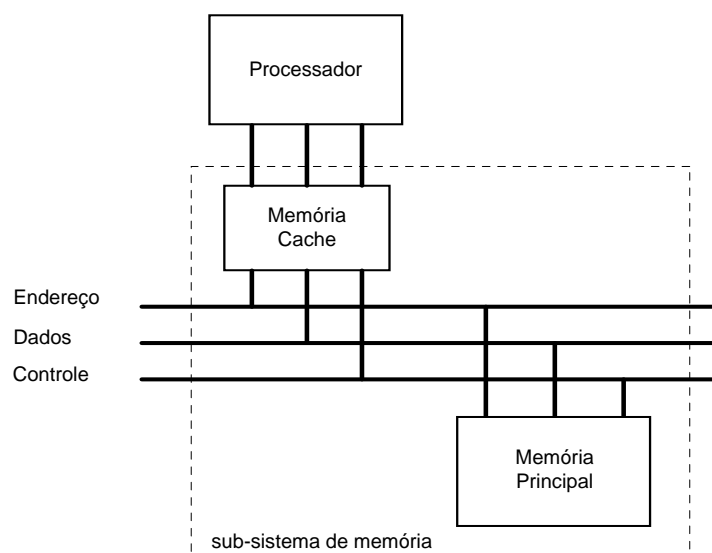


Figura 4.7. Localização da memória cache.

A memória *cache* é implementada com o mesmo tipo de circuito eletrônico usado nas células de *bit* de uma memória RAM estática. Assim, a capacidade de armazenamento de uma memória *cache* situa-se normalmente entre 256 Kbytes e 512 Kbytes. No entanto, a memória *cache* apresenta um baixo tempo de acesso, possibilitando que o processador acesse dados armazenados em apenas um único ciclo de *clock*.

Nesta nova organização, o processador direciona os acessos inicialmente para a memória *cache*. Se o dado referenciado encontra-se na *cache* diz-se que ocorreu um **cache hit**, e o acesso é completado em apenas um ciclo de *clock*. Se o dado não se encontra na *cache*, diz-se que ocorreu um **cache miss**. Quando acontece um *cache miss*, um bloco contendo o dado referenciado e os dados armazenados em sua vizinhança são copiados da memória principal para a memória *cache*. Após a transferência deste bloco de dados para a memória *cache*, o processador reinicia a execução da instrução, gerando uma nova busca na *cache*, onde o dado agora será encontrado. Quando ocorre um *cache miss*, o acesso consome vários ciclos de *clock* para ser completado.

Na prática, observa-se que entre 90% e 98% dos acessos resultam em um *cache hit*. Isto acontece devido à propriedade exibida por vários tipos de programas, a **propriedade da localidade de referência**. Com a inclusão de uma memória *cache*, apenas uma pequena parcela dos acessos realizados sofrem com o alto tempo de acesso da memória principal. Assim, as memórias *cache* são extremamente importantes no sentido de obter-se um melhor desempenho. Atualmente, a maioria das arquiteturas de computador incluem memórias *cache*

em seu sub-sistema de memória. Alguns processadores possuem memórias *cache* separadas, uma para instruções e outra para dados.

Mapeamentos de Memória *Cache*

As memórias *cache* podem ser estruturadas de formas diferentes. A memória *cache* é formada por uma memória RAM e pelo seu **diretório**. A memória RAM é organizada em **linhas**, sendo que em cada linha são armazenados blocos de dados proveniente da memória principal. Elas podem ser **mapeadas diretamente** ou por **associatividade**.

A organização mais simples é a da *cache* **mapeada diretamente**. Neste tipo de *cache* o bloco tem comprimento igual a uma palavra, e tanto os acertos quanto às falhas são simples de serem verificados e tratados.

Na *cache* mapeada diretamente uma determinada palavra só pode ocupar uma única posição na *cache*, havendo ainda um rótulo separado para cada palavra. A maioria das *caches* mapeadas diretamente usa o seguinte processo de mapeamento:

(endereço do bloco) módulo (número de blocos da *cache*).

Para saber se um dado está na *cache* basta comparar o rótulo. O rótulo contém informações sobre um endereço de memória, que permite identificar se a informação desejada está ou não na *cache*. Ele só precisa conter a parte superior do endereço, correspondente os *bits* que não estão sendo usados como índice da *cache*. Os índices são usados para selecionar a única entrada da *cache* que corresponde ao endereço fornecido. Além disso, existe um *bit* de validade para indicar se uma entrada contém ou não um endereço válido. A Figura 4.8 mostra uma *cache* mapeada diretamente com oito blocos.

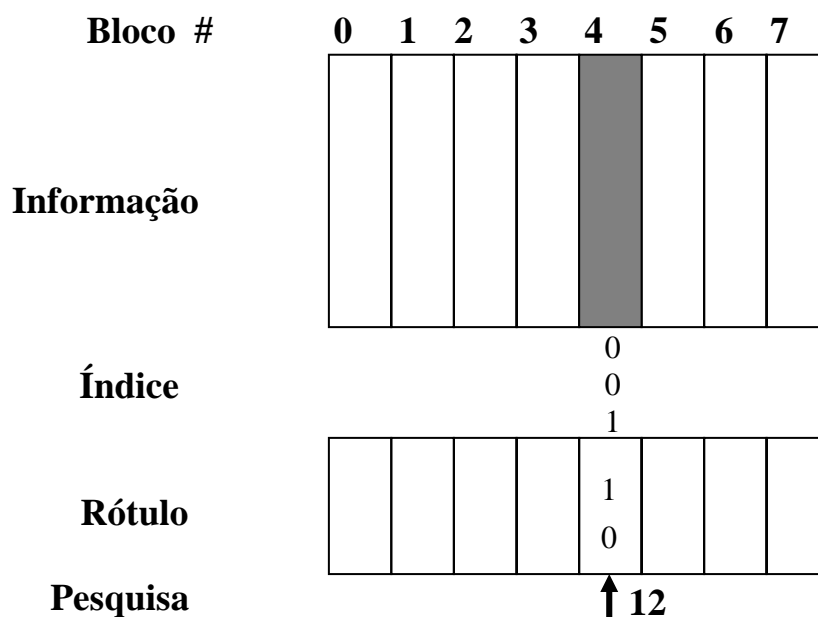


Figura 4.8. Memória *cache* mapeada diretamente.

Para entender a Figura 4.8 considere que uma memória principal que possui 32 palavras. Seus endereços em números binários variam entre 00000 e 11111. Neste caso, são necessários três *bits* para indexar a memória *cache* e os dois *bits* restantes são utilizados como rótulos. No mapeamento direto cada palavra só pode ocupar uma única posição na memória *cache*. Suponha que desejamos saber se o bloco com endereço de memória principal 12 está contido na memória *cache*. Assim, para o endereçamento temos $12 \text{ módulo } 8 = 4$. O bloco 12 só pode estar armazenado no bloco 4 da memória *cache*. Basta então comparar os dois *bits* mais significativos do bloco 12 (01) com o rótulo do bloco 4 da memória *cache* para saber se o conteúdo está presente. No caso do exemplo da Figura 5.8, no bloco 4 (**100**) da memória *cache* só podem ficar armazenadas as palavras da memória principal com índice igual a 4 nos seus três *bits* menos significativos. Isto é, 4 (00**100**), 12 (01**100**), 20 (10**100**) e 28 (11**100**).

Para manter a **coerência entre a *cache* e a memória principal**, podemos utilizar o esquema *write-through* ou o esquema *write-back*. No esquema *write-through* a memória principal é atualizada a cada modificação da memória *cache*. No esquema *write-back* a memória principal só é atualizada quando um bloco da *cache* precisa ser substituído. Para o esquema *write-back* é necessário um *bit* de controle que indica se o bloco armazenado foi ou não modificado desde o seu carregamento na memória *cache*. Se ele foi modificado é necessário que seu conteúdo seja salvo na memória principal, caso contrário basta simplesmente descartá-lo.

Para aproveitar melhor as vantagens da localidade espacial uma *cache* deve ter um bloco com comprimento maior do que uma palavra. O uso de blocos maiores diminui a taxa de falhas e melhora a eficiência da *cache*. Isso porque reduz o número de rótulos que são necessários armazenar em relação a quantidade de informações da *cache*. Embora, um bloco maior possa aumentar a penalidade nas falhas, pois ela cresce exponencialmente com o tamanho do bloco transferido.

As *caches* **mapeadas por associatividade** possuem um bloco maior do que uma palavra. Neste esquema, ao ocorrer uma falha, várias palavras adjacentes são trazidas para a *cache*. Este fato aumenta a probabilidade de ocorrer um *cache hit* nos próximos acessos. As *caches* mapeadas por associatividade usam o seguinte processo de mapeamento, o conjunto que contém o bloco de memória é dado por:

(endereço do bloco) módulo (número de conjuntos da *cache*).

As *caches* mapeadas por associatividade por ser classificadas em **associativas por conjuntos** ou **completamente associativas**. O número total de rótulos e *bits* de validade nas *caches* mapeadas por associatividade é menor do que nas *caches* mapeadas diretamente porque são usados para todas as palavras que estão contidas num mesmo bloco e que formam um conjunto. Estas memórias são chamadas **memórias *cache* associativas por conjunto** (*set*

associative cache memory). Este compartilhamento contribui para um uso mais eficiente do espaço disponível para armazenamento na *cache*.

Uma memória *cache* associativa por conjunto com n linhas em cada conjunto é chamada **cache n -associativa por conjunto** (*n-way set associative cache*). As memórias *cache* associativas por conjunto são as encontradas na maioria dos processadores. É importante ressaltar que o tamanho de linha, o número de linhas por conjunto e o número de conjuntos podem variar, resultando em configurações diferentes.

A memória *cache* associativa por conjunto, ao receber o endereço de uma locação de memória, interpreta o endereço da seguinte forma. O endereço é logicamente dividido em três campos: o campo **byte**, formado pelos *bits* menos significativos; o campo **conjunto**, formado pelos n *bits* seguintes; e o campo **rótulo**, composto pelos m *bits* mais significativos do endereço. A memória *cache* usa os *bits* do campo *byte* para selecionar um *byte* específico dentro de uma linha. O campo conjunto é usado para selecionar um dos conjuntos, enquanto o campo rótulo é usado para verificar se o dado referenciado se encontra em alguma das linhas do conjunto selecionado.

Num acesso a memória *cache* seleciona primeiro o conjunto, e em seguida compara o rótulo do endereço recebido com os *rótulos* armazenados nas entradas de diretório do conjunto selecionado. O bloco pode ser colocado em qualquer elemento deste conjunto. Se o rótulo no endereço coincide com algum dos rótulos no diretório, isto significa que o bloco com o *byte* referenciado encontra-se na linha associada à entrada do diretório que contém o rótulo coincidente. Esta linha é então selecionada e o *byte* dentro desta linha é finalmente acessado. A Figura 4.9 mostra uma *cache* mapeada por associatividade com quatro conjuntos.

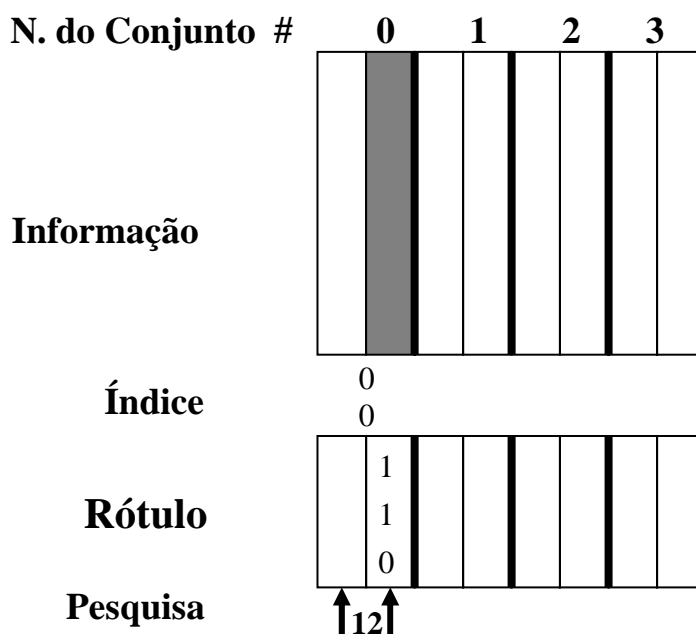


Figura 4.9. Memória cache mapeada por associatividade.

Suponha o mesmo exemplo da Figura 4.8 com uma memória principal de 32 palavras, e que desejamos saber se o bloco com endereço de memória principal 12 está contido na memória *cache*. A memória *cache* é formada por quatro conjuntos de duas linhas. Assim, para o endereçamento temos $12 \bmod 4 = 0$. O bloco 12 só pode estar armazenado no conjunto 0 da memória *cache*. Basta então comparar o rótulo do bloco 12 (011) com os rótulos do conjunto 0 da memória *cache* para saber se o conteúdo está presente.

Uma memória **cache totalmente associativa** pode ser vista como uma *cache* onde existe **um único conjunto**. O bloco da memória principal pode ser colocado em qualquer um dos elementos deste conjunto. A Figura 4.10 mostra uma *cache* totalmente associativa.

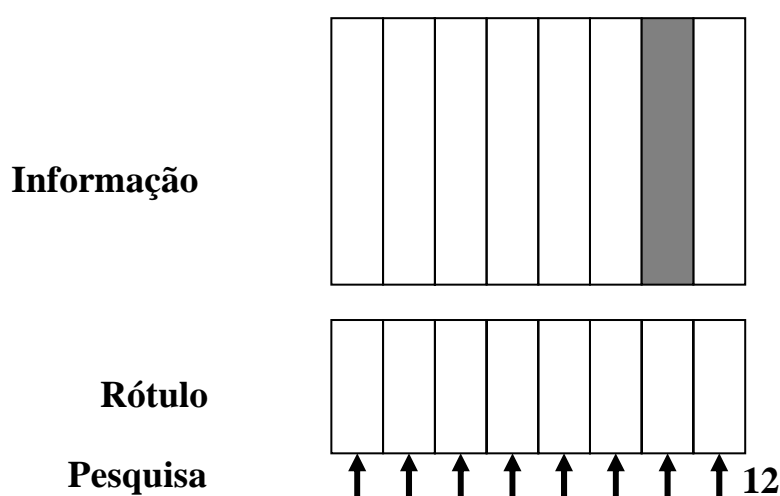


Figura 4.10. Memória cache com mapeamento totalmente associatividade.

Suponha ainda o mesmo exemplo de uma memória principal com 32 palavras e que desejamos saber se o bloco com endereço de memória igual a 12 está contido na memória *cache*. Neste caso, onde o bloco pode ser colocado em qualquer elemento da *cache*, é necessário pesquisar todos os blocos.

A não coincidência do rótulo do endereço com os rótulos armazenados no conjunto indica que o *byte* referenciado não se encontra na *cache* (ou seja, ocorreu um *cache miss*). Neste caso, a lógica de controle da memória *cache* se encarrega de copiar o bloco apropriado a partir da memória principal. Este bloco será armazenado em uma das linhas do conjunto indicado pelo endereço. Uma vez que o bloco tenha sido carregado na memória *cache*, o acesso prossegue como descrito anteriormente.

Da mesma forma como mencionado para as *caches* mapeadas diretamente, na coerência entre a *cache* e a memória principal podemos utilizar os esquemas *write-through* ou *write-back*. Além disso, existe um *bit* de validade e um *bit* de controle, caso seja utilizado o esquema *write-back*.

A questão central na substituição é a escolha do bloco a ser descartado. A maneira mais simples seria escolher aleatoriamente qualquer um dos blocos do conjunto selecionado. A escolha aleatória é atrativa devido à sua simplicidade de implementação. No entanto, esta política aumenta a chance de ser escolhido um bloco que será referenciado em breve. Em outras palavras, com a escolha aleatória existe o risco de um aumento excessivo no número de *cache misses*.

Em geral, é adotada uma política de escolha mais segura, denominada **LRU** (*Least Recently Used*). A política LRU diz que o bloco a ser substituído é aquele que não é referenciado há mais tempo. Este critério de escolha se baseia no seguinte raciocínio. Pelo princípio da localidade, quando um bloco é referenciado, existe uma grande chance de que ele seja novamente referenciado em breve. Se um bloco não é referenciado há algum tempo, este fato **pode** indicar que as referências concentradas naquele bloco já aconteceram e que o bloco não mais será referenciado, pelo menos no futuro próximo. Assim, este bloco pode ser substituído sem o risco de gerar *cache misses* imediatos. *Bits* de controle no diretório indicam se um bloco foi ou não referenciado, sendo consultados quando um bloco deve ser substituído. Na prática, observa-se que a política LRU de fato não introduz um aumento significativo no número de *cache misses*.

Memórias *Cache* Primária e Secundária

Normalmente, a frequência de *cache hits* aumenta à medida que o tamanho da memória *cache* aumenta. No entanto, observa-se que a partir de um certo ponto o número de *cache hits* não aumenta significativamente com o aumento da capacidade da memória *cache*. A partir deste ponto não compensa aumentar o tamanho da memória *cache*, pois isto acarretaria um aumento no custo do processador que não traria um benefício proporcional.

Para minimizar o efeito dos *cache misses* residuais, alguns sistemas incorporam duas memórias *cache*, uma denominada **primária** e a outra **secundária**, em uma estrutura conhecida como **memória *cache* em dois níveis** (*two-level cache*). A memória *cache* primária é integrada com o processador em um mesmo dispositivo, possui um tamanho pequeno, e apresenta um tempo de acesso que permite acessos em um único ciclo de *clock*. Por sua vez, a *cache* secundária é externa ao processador, localizando-se na placa-mãe do sistema, possui um tamanho bem maior, e apresenta um tempo de acesso maior que a memória *cache* primária. Um acesso a *cache* secundária normalmente consome dois ou três ciclos de *clock*.

Em um sistema com esta organização, a memória *cache* primária captura a maioria dos acessos realizados pelo processador. Os acessos que resultam em falhas são em sua maioria capturados pela memória *cache* secundária. Apesar de ser um pouco mais lenta que a memória *cache* primária, a memória *cache* secundária ainda é significativamente mais rápida

que a memória principal. Assim, as falhas da *cache* primária apresentam uma penalidade menor quando capturadas pela *cache* secundária do que se fossem direcionadas diretamente para a memória principal.

4.6 Memória Virtual

Como mencionado anteriormente, a memória principal disponível em um computador é, em geral, bem menor do que o tamanho máximo de memória permitido pelo processador. O esquema de memória virtual foi originalmente criado para permitir a execução de programas cujas exigências quanto ao tamanho da memória sejam maiores do que a capacidade de memória instalada no sistema.

O Conceito de Memória Virtual

Em um sistema sem memória virtual, o endereço gerado pelo programa em execução é o próprio endereço usado para acessar a memória principal. O mesmo não acontece em um sistema com memória virtual. O endereço gerado pelo programa, ou **endereço virtual**, é diferente do **endereço real** usado para acessar a memória principal. Os possíveis endereços virtuais que podem ser gerados pelo programa formam o **espaço de endereçamento virtual**. Enquanto os endereços na memória principal constituem o **espaço de endereçamento real**.

Sob o ponto de vista de um programa a memória disponível é aquela representada pelo espaço de endereçamento virtual. O espaço de endereçamento virtual visto e utilizado pelo programa pode ser bem maior do que o espaço de endereçamento real. Efetivamente retirando do programa as limitações impostas pela capacidade da memória física de fato existente no sistema. É importante perceber que o espaço de endereçamento virtual é uma abstração. Embora sob o ponto de vista do programa as instruções e dados estejam armazenados dentro do espaço de endereçamento virtual na realidade eles continuam armazenados na memória principal, representada pelo espaço de endereçamento real.

Esta distinção entre endereços e espaços de endereçamento exige um mecanismo que faça a correspondência entre o endereço virtual gerado pelo programa e o endereço real que é usado para acessar a memória principal. Além disso, a técnica de memória virtual permite que as instruções e os dados do programa que se encontram no espaço virtual, não estejam presentes na memória principal no momento em que são referenciados. Assim, além do mapeamento anteriormente mencionado, é necessário um mecanismo para o carregamento automático na memória principal das instruções e dados que são referenciados pelo programa dentro da sua memória virtual e que não se encontram presentes na memória física. Estes dois aspectos são discutidos em seguida.

O Mecanismo de Memória Virtual

O mapeamento entre endereços virtuais e reais é feito por um componente do sub-sistema de memória denominado **tradutor dinâmico de endereços**, ou DAT (*Dynamic Address Translator*). A Figura 4.11 mostra como o DAT se insere dentro do sub-sistema de memória.

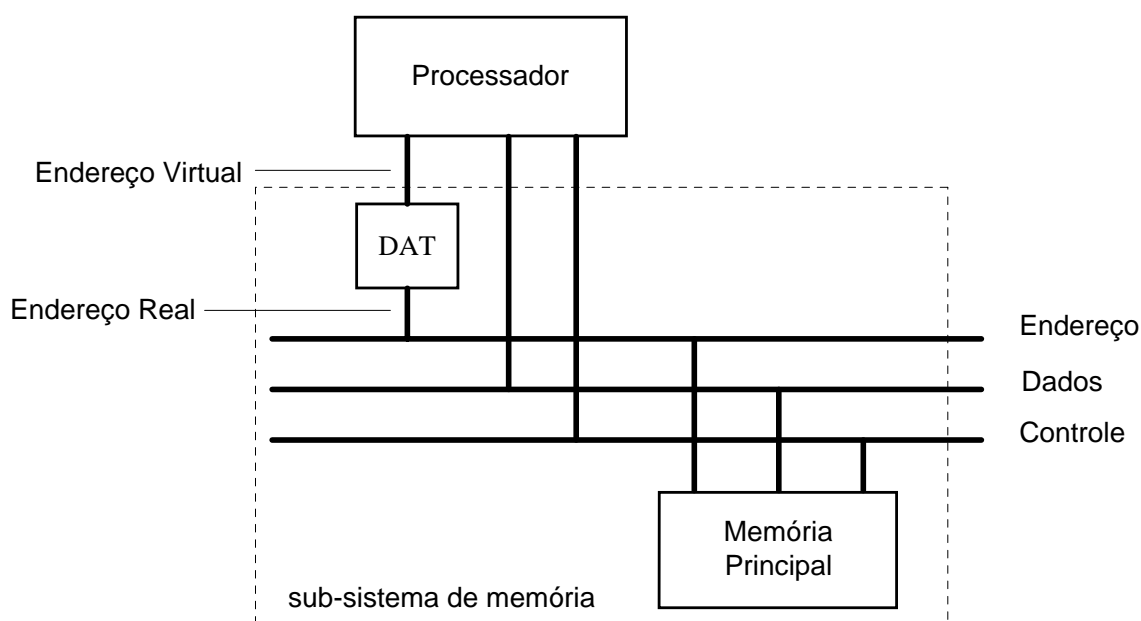


Figura 4.11. O tradutor dinâmico de endereços em um sistema com memória virtual.

O processador fornece ao DAT o endereço virtual gerado pelo programa em execução. O endereço virtual é mapeado em um endereço real pelo DAT e enviado para a memória principal através do barramento de endereço. Note que o mapeamento é feito em cada acesso à memória, no início do acesso.

Para realizar o mapeamento, o DAT utiliza uma **tabela de mapeamento** localizada na memória principal. A tabela de mapeamento permanece na memória principal durante a execução do programa. Ao receber um endereço virtual, o DAT usa este endereço para indexar a tabela de mapeamento. A entrada indexada contém o endereço real correspondente ao endereço virtual. Na realidade, o mapeamento não é feito no nível de cada locação de memória, pois isto exigiria uma tabela de mapeamento com um número de entradas igual ao

tamanho do espaço de endereçamento virtual. Para manter um tamanho de tabela aceitável o mapeamento é feito no nível de blocos

A Figura 4.12(a) mostra o mecanismo de mapeamento básico.

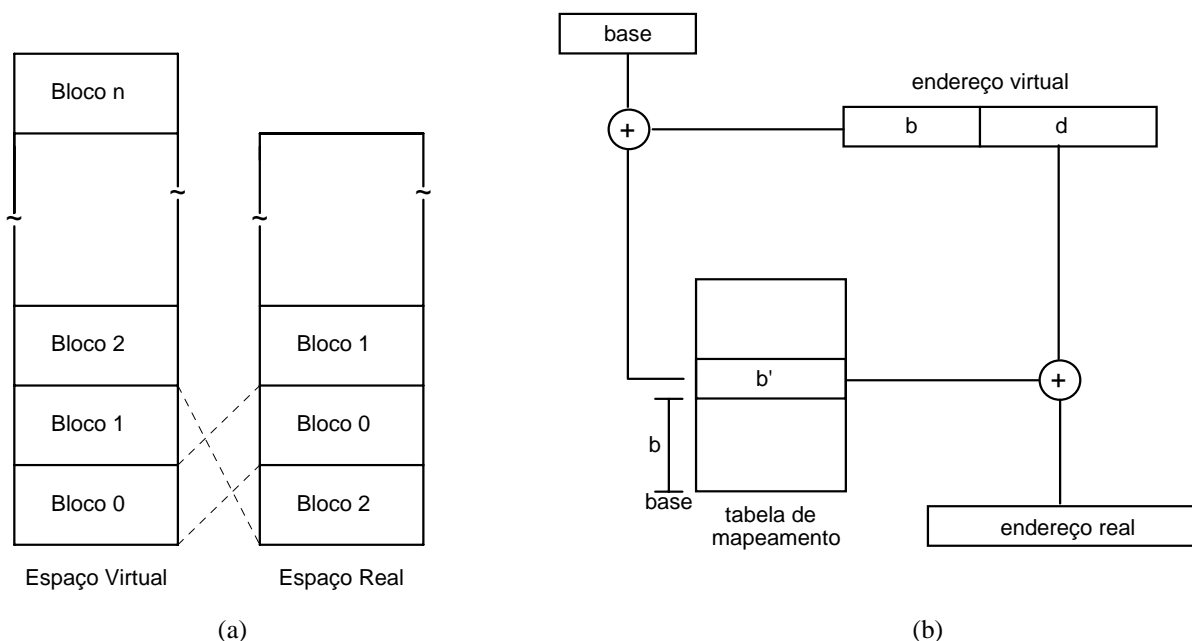


Figura 4.12. Mecanismo básico de mapeamento.

O espaço de endereçamento virtual é logicamente dividido em blocos, que são mapeados para o espaço de endereçamento real pelo DAT. Cada entrada na tabela de mapeamento contém o endereço-base de um bloco, ou seja, o endereço real a partir do qual o bloco está armazenado na memória principal. Em uma **memória virtual segmentada**, os blocos são chamados **segmentos**, e podem ter **tamanhos variáveis**. Em uma **memória virtual paginada**, os blocos são chamados **páginas** e possuem **tamanho fixo**.

A Figura 4.12(b) mostra como é feito o mapeamento dos blocos. O DAT considera que o endereço virtual recebido do processador é constituído por dois campos. O **campo *b***, formado pelos *bits* mais significativos do endereço virtual, indica o número do bloco onde se encontra a localização de memória referenciada. O **campo *d***, formado pelos *bits* menos significativos, indica o deslocamento da localização de memória em relação ao início do bloco.

O DAT usa o valor do campo *b* para realizar o acesso a uma entrada da tabela de mapeamento. Para tanto, o DAT soma o valor do campo *b* ao endereço-base da tabela de mapeamento. O endereço-base da tabela é mantido internamente no próprio DAT, em um registrador chamado TBR (*Table Base Register*). A soma resultante fornece o endereço de uma entrada da tabela e, com este endereço, o DAT acessa a tabela de mapeamento e obtém o endereço-base do bloco (indicado por *b'*, na Figura 4.12). Para obter finalmente o endereço real da localização de memória referenciada, o DAT soma o valor do campo *d*, que indica a posição da localização dentro do bloco, ao endereço-base do bloco.

Note que no mecanismo de mapeamento mostrado na Figura 4.12, para cada referência à memória realizada pelo programa é necessário um acesso adicional para consultar a tabela de mapeamento. Neste esquema o número de acessos à memória principal durante a execução de um programa seria duplicado, comprometendo seriamente o desempenho. Para solucionar este problema, o DAT possui internamente uma pequena memória, denominada **TLB** (*Translation Lookaside Buffer*). A TLB age como uma memória *cache*, armazenando os pares (b, b') que foram usados nos acessos mais recentes.

Uma operação de mapeamento com a TLB ocorre, então, da seguinte forma. O DAT usa o número do bloco no campo b para endereçar a TLB. Se acontece um *TLB hit*, o DAT obtém o endereço-base do bloco (b') a partir da TLB. Neste caso, o mapeamento não acrescenta nenhum retardo significativo ao acesso. Se acontece um *TLB miss*, o DAT consulta a tabela de mapeamento na memória para realizar o mapeamento. Além disso, o DAT armazena na TLB o par (b, b') usado neste mapeamento. Na prática, a maioria dos mapeamentos é satisfeita pela TLB. Isto acontece devido ao princípio da localidade, discutido anteriormente, e cada mapeamento poderá ser completado rapidamente.

A Memória Secundária

Devido à diferença de tamanho entre os espaços de endereçamento virtual e real, pode acontecer que um bloco referenciado pelo programa não esteja presente na memória principal no momento em que a referência acontece. Os blocos de instruções e de dados de um programa em execução ficam armazenados na chamada **memória secundária**, que na realidade é uma unidade de disco do sistema.

A ausência de um bloco referenciado pelo programa é detectada pelo DAT no momento do mapeamento, e o bloco é então carregado da memória secundária para a memória principal. A Figura 4.13 mostra em detalhes como este processo ocorre.

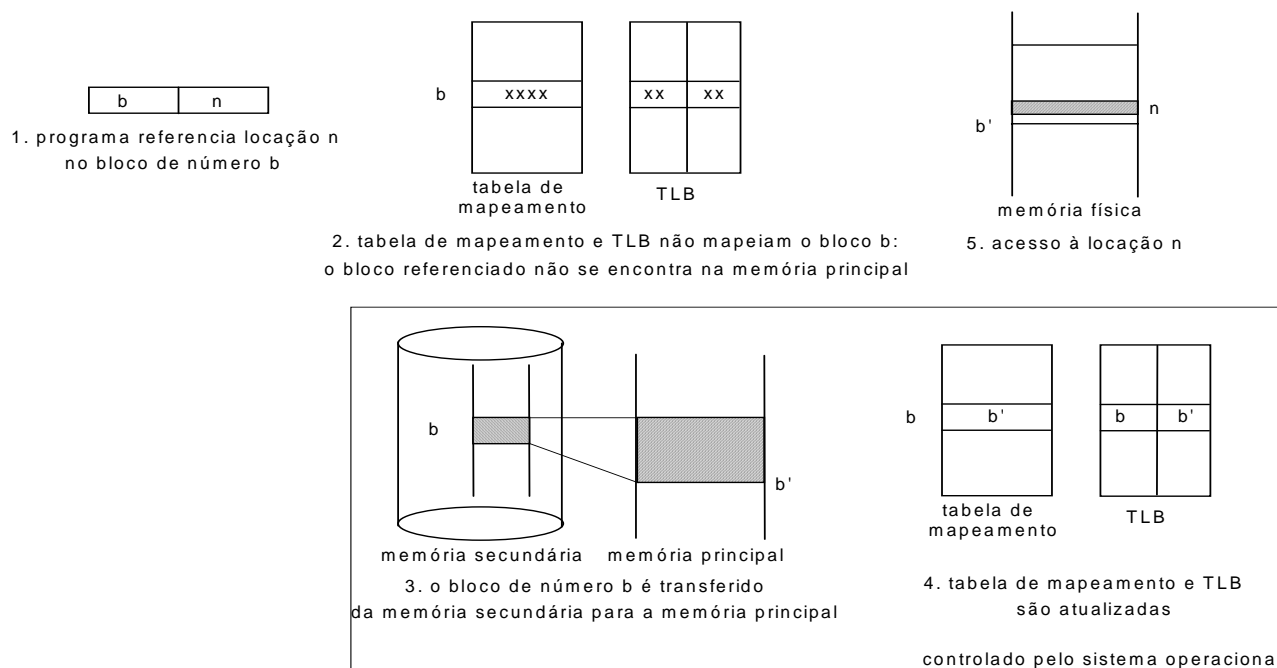


Figura 4.13. Carregamento de blocos na memória principal.

Neste exemplo, o programa referencia a localização n no bloco b . Ao receber o endereço virtual, o DAT verifica o conteúdo da TLB. Como a informação de mapeamento não se encontra na TLB, o DAT faz o mapeamento através da tabela de mapeamento. Um *bit* de controle na entrada da tabela indica ao DAT que não existe um mapeamento válido para o bloco. Isto significa que o bloco não se encontra na memória principal. A ausência de um bloco da memória é também denominada **falha de página** (memória virtual paginada) ou **falha de segmento** (memória virtual segmentada). Ao detectar uma falha, o DAT gera uma interrupção, transferindo o controle do processador para o sistema operacional.

O sistema operacional obtém do DAT o número do bloco que foi referenciado. O sistema operacional localiza este bloco na memória secundária, aloca um espaço na memória principal e transfere o bloco da memória secundária para a memória principal. Em seguida, o sistema operacional atualiza a tabela de mapeamento e a TLB e devolve o controle para o programa que estava em execução. O controle é devolvido exatamente para o ponto onde se encontra o acesso que provocou o carregamento do bloco. O acesso é então re-executado pelo programa, sendo agora completado com sucesso porque o bloco referenciado já se encontra na memória principal.

Com a memória virtual, não é necessário que todas as instruções e dados de um programa permaneçam na memória principal durante a execução do programa. Os blocos do programa são transferidos da memória secundária para a memória principal à medida que são referenciados. É importante observar que esta transferência é totalmente transparente para o programa e, mais ainda, para o programador que desenvolveu o programa.

O mecanismo de memória virtual é em parte controlado por *hardware* e em parte por *software*. A consulta a TLB e à tabela de mapeamento é realizada pelo *hardware*,

representado pelo DAT. O *software*, representado pelo sistema operacional, realiza transferências de blocos entre a memória secundária e a memória principal e as atualizações necessárias das estruturas de mapeamento. Além disso, o sistema operacional controla a ocupação da memória principal, localizando áreas livres onde um novo bloco pode ser armazenado, ou ainda determinando blocos que podem ser retirados da memória principal para dar lugar a novos blocos.

4.7 Uma Estrutura Comum para Hierarquias de Memória

As partes componentes do sistema de memória têm muito em comum. Muitas das políticas e das características que determinam o funcionamento da hierarquia são semelhantes em termos qualitativos. A seguir discutimos os aspectos operacionais comuns aos sistemas hierárquicos da memória, e como tais aspectos determinam seu comportamento. Analisamos quatro questões que se aplicam a quaisquer dois níveis da hierarquia de memória. Porém, quando necessário, para simplificar adotamos a terminologia da memória *cache*.

As quatro questões que abordamos está relacionada aos blocos nos diversos níveis da hierarquia de memória. Para tratar um bloco é necessário saber:

- **onde colocar** um bloco;
- **como localizar** um bloco;
- **como substituir** um bloco e
- **como modificar** um bloco.

Onde se pode colocar um bloco

A colocação de um bloco no nível superior da hierarquia pode ser implementada com diversos esquemas. Estes esquemas variam desde o mapeamento direto, associativo por conjunto até o completamente associativo. Todos estes esquemas podem ser expressos com uma variação do número de: conjuntos (NC) e blocos por conjunto (NB). A Tabela 4.1 mostra estes esquemas.

Esquema	NC	NB
Direto	Número de blocos	1
Associativo por conjunto	Número de blocos associatividade	Associatividade
Totalmente associativo	1	Número de blocos

Tabela 4.1. *Esquemas de mapeamento de blocos numa hierarquia de memória.*

A vantagem de aumentar a associatividade é melhorar a localidade espacial, reduzindo assim a taxa de falhas. Porém, suas desvantagens principais são maior custo se houver falhas e a conseqüente redução no tempo de acesso ao dado.

As memórias *cache*, em geral, são implementadas de forma associativa por conjunto, com o número de conjuntos variando entre 2 e 8. Já a memória virtual é implementada de modo totalmente associativo.

Como localizar um bloco

A forma de localizar um bloco depende do esquema implementado para colocação do bloco na hierarquia de memória. A Tabela 4.2 mostra como localizar um bloco na memória.

Esquema	Método de Localização	Número de Comparações
Direto	índice	1
Associativo por conjunto	indexar o conjunto, pesquisar entre os elementos	grau de associatividade
Totalmente associativo	pesquisar todas as entradas	igual ao tamanho
	tabela separada para busca	0

Tabela 4.2. *Resumo dos esquemas para localizar blocos numa hierarquia de memória.*

A escolha do mapeamento depende do custo de uma falha em relação ao custo da implementação da associatividade.

Nas memórias *cache* e nas TLBs mapeadas diretamente são utilizados índices para a localização dos blocos, e aparecem em sistemas recentes. Quando é implementado o esquema de mapeamento por conjunto, nas *caches* e nas TLBs, são utilizados índices para encontrar o conjunto desejado e deve-se pesquisar entre todos os elementos do conjunto. Este é o esquema mais comum encontrados nos sistemas. O esquema totalmente associativo, para estes tipos de memória, necessita que todas as entradas sejam pesquisadas. Assim, ele é proibitivo exceto se as memórias forem muito pequenas.

Na memória virtual é implementado o esquema totalmente associativo. A utilização de uma tabela totalmente associativa é motivada por quatro fatores:

- por causa do alto custo das falhas, a associatividade é benéfica;
- a associatividade total permite que o *software* utilize algoritmos de substituição de páginas sofisticados, que são projetados para reduzir a taxa de falhas;
- o mapeamento completo pode ser facilmente indexado, sem a necessidade de *hardware* extra e
- quanto maior o tamanho da página menor *overhead* relativo ao tamanho da tabela de páginas.

Como substituir um bloco

É necessário substituir um bloco de memória quando não existem mais espaços livres naquele nível da hierarquia memória. O bloco candidato à substituição depende da forma como o mapeamento foi implementado.

No mapeamento direto não há escolha. Um bloco só pode ocupar uma única posição na memória, então só há um candidato. No mapeamento associativo por conjunto, um bloco é escolhido entre os blocos pertencentes a um dos conjuntos. No mapeamento totalmente associativo todos os blocos são possíveis candidatos a serem substituídos.

As estratégias de substituição mais comuns são a **aleatória** e a **LRU**. Na estratégia aleatória os blocos são escolhidos ao acaso. Esta é a implementação mais simples. Porém, é, em geral, usada somente em memórias *cache*. Na estratégia LRU são escolhidos os blocos usados menos recentemente. Ela é uma estratégia mais custosa de ser implementada e pode ser utilizada tanto nas memórias *cache* quanto em memórias virtuais.

Como modificar um bloco

Existem duas opções básicas para tratamento de escritas na memória: ***write-through*** e ***write-back***. Na técnica *write-through* o dado é escrito tanto na memória *cache* quanto na memória principal. As falhas são menos custosas e sua implementação é mais simples. Na técnica *write-back* o dado é escrito somente na memória *cache*. O dado é copiado para a memória principal só no momento da substituição. Nesta técnica as palavras são escritas na velocidade da memória *cache*, e múltiplas escritas num bloco correspondem a uma única escrita no componente inferior da hierarquia de memória. Além disso, aumenta a banda passante na transferência dos blocos. A memória virtual só utiliza esta técnica.

O Modelo dos Três Cs

O modelo dos três Cs é um modelo intuitivo para o entendimento do comportamento da hierarquia de memória. Ele explica quais as fontes das falhas geradas nos sistemas de memória. As falhas podem ser **compulsórias**, devido à **capacidade** e por **conflitos** (ou colisão).

As falhas compulsórias são causadas pelo primeiro acesso a um bloco que nunca está na *cache*. As falhas por capacidade ocorrem porque, por exemplo, a *cache* não pode armazenar todos os blocos, e eventualmente os blocos precisam ser substituídos. As falhas por conflitos ocorrem nas *caches* associativas por conjunto ou mapeadas diretamente quando diversos blocos competem pelo mesmo conjunto.