URI - Universidade Regional Integrada do Alto Uruguai e das Missões

Memórias

Luciano L. Caimi
caimi@urisan.tche.br

Santo Ângelo, junho de 2004

Sumário

Lista de Figuras

Capítulo 1

Subsistema de Memória

A memória é o componente do sistema de computação cuja função é armazenar as informações que são ou serão manipuladas pelo sistema.

Conceitualmente, a memória é um componente muito simples: é um "depósito" onde são guardados os elementos (as informações) para serem usados quando desejado (recuperação da informação armazenada).

Para que a informação possa ser armazenada na memória (operação de escrita) é necessário que seja definido um local disponível, identificado de alguma forma precisa e única (um número, por exemplo). O número ou código de acesso ao local é o *endereço* e irá permitir que a informação possa ser localizada. Na recuperação da informação (operação de leitura), o endereço onde a informação desejada está armazenada deve ser fornecido para que seja fornecido o dado contido naquela posição.

Devido a grande variedade de características desejadas e dos diferentes tipos de memória não é possível implementar um sistema de computação com uma única memória. Na realidade, há muitas memórias no computador, as quais se interligam de forma bem estruturada, constituindo um sistema em si, parte do sistema global, podendo ser denominada subsistema de memória.

1.1 Hierarquia de memória

Para o correto e eficaz funcionamento da manipulação das informações (instruções e dados de um programa) de e para a memória de um computador, verifica-se a necessidade de diferentes tipos de memória. Para certas atividades, é fundamental que a transferência de informações seja a mais rápida possível (por exemplo, execução das instruções no processador), em outros casos, a rapidez não é o fator fundamental mas sim a capacidade de armazenamento (armazenar dados

referente as precipitações pluviométricas nas diferentes regiões do estado por 1 ano).

No sentido de atender as diferentes características necessárias a um sistema de armazenamento, bem como, manter uma relação custo/benefício atraente, as diferentes tecnologias de fabricação e armazenamento de dados são utilizadas e organizadas de forma hierárquica.

A figura 1.1 apresenta uma pirâmide que resume com bastante fidelidade as características do subsistema de memória.

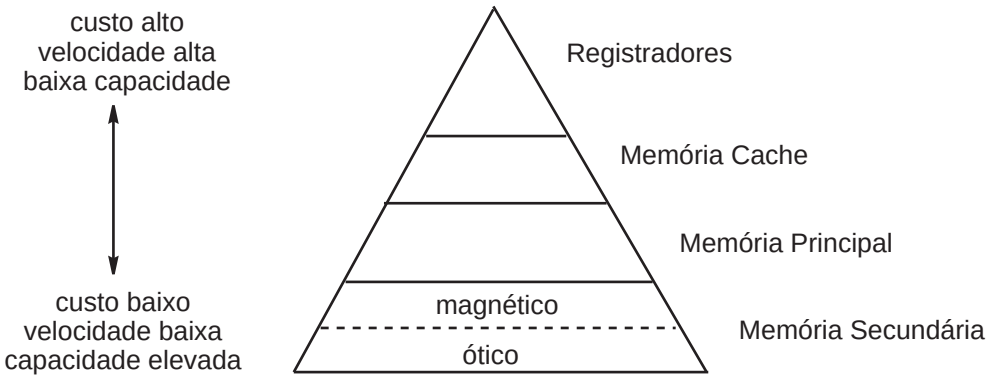

Figura 1.1: Hierarquia de Memória

É comum apresentar a hierarquia de memória de um computador através de uma pirâmide. A variação crescente de certas características pode ser mostrado adequadamente por esta figura. Os principais parâmetros de análise das características das memórias serão apresentadas a seguir.

- Tempo de acesso - indica quanto tempo a memória gasta para colocar uma informação no barramento de dados após uma determinada posição ter sido endereçada. Isto é, é o perríodo de tempo decorrido desde o instante em que foi iniciada a operação, até que a informação seja disponibilizada.

  O tempo de acesso de uma memória é dependente do modo como o sistema de memória é construído e da velocidade de seus circuitos. Deve-se lembrar que o tempo de acesso das medidas eletrônias (RAM, ROM, etc) é independente da distância física entre o local de um acesso e o do próximo acesso, ao passo que no caso de dispositivos eletromecânicos (discos, fitas, etc) o tempo varia conforme a distância física entre dois acessos consecutivos.

- Tempo de ciclo de memória - é o período de tempo decorrido entre duas operações sucessivas de acesso a memória, sejam de leitura ou escrita. Este tempo depende de outros fatores relacionados ao funcionamento da memória. Certos tipos de memória impedem por um intervalo de tempo que um novo acesso seja processado logo após a conclusão do

anterior. Nas memórias onde isto não ocorre o tempo de acesso é igual ao tempo do ciclo de memória.

- Capacidade - é a quantidade de informação que pode ser armazenada na memória; a unidade de medida mais comum é o byte, embora também possam ser usadas outras unidades como células, setores, etc. Dependendo do tamanho da memória, isto é, da sua capacidade, indica-se o valor numérico total de elementos de forma simplificada, através da inclusão de K (kilo), M (mega), ou T (tera).

- Volatilidade - memórias podem ser do tipo volátil ou não volátil. Uma memória não volátil é a que retém a informação armazenada quando a energia elétrica é desligada. Memória volátil é aquela que perde a informação armazenada quando a energia elétrica é desligada.

  Registradores e memória RAM são memórias voláteis. mamérias magnéticas, óticas e também as do tipo ROM, EPROM são memórias do tipo não volátil.

- Tecnologia de fabricação - ao longo do tempo diversas tecnologias vem sendo desenvolvidas para a fabricação de memórias. Entre as mais conhecidas temos as memórias de semicondutores, as memórias de meio magnético e as memórias de meio ótico.

- Temporariedade - diz respeito ao tempo de permanência da informação em um dado tipo de memória.

- Custo - o custo de fabricação de uma memória é bastante variado em função de diversos fatores, entre os quais pode-se mencionar principalmente a tecnologia de fabricação, que resulta em um maior ou menor tempo de acesso, ciclo de memória, etc. Uma boa unidade de medida de custo é o preço por byte armazenado, em vez do custo total da memória em si. Isto porque, devido às diferentes capacidades de armazenamento, seria irreal considerar, para comparação, o custo pelo preço da memória em si.

De uma maneira geral, cada um dos itens acima se faz presente ao longo da hierarquia de memória e cada um deles influi decisivamente nos tipos de memória existenstes nos sistemas computacionais atuais.

O termo RAM (*Random Acess Memory*) designa uma memória cujos endereços podem ser acessados em quaquer ordem. Ela surgiu em uma época em que existiam memórias com acessos obrigatoriamente seqüenciais. A sigla RAM adicionam-se as letras D ou S para indicar memória dinâmica (DRAM) ou memória estática (SRAM). A memória estática realiza apenas duas operações básicas: esrita e leitura. O armazenamento é feita em células (flip-flops) que guardam a informação enquanto o circuiito permanece energizado.

Deve-se lembrar ainda a existência de memórias não voláteis. Tais memórias recebem o nome genérico de ROM (*Read Only Memory*) e que respondem apenas aos ciclos de leitura. Existem vários tipos de ROM:

PROM: ROM programável pelo usuário;

EPROM: ROM programável eletricamente e apagável por exposição a luz ultravioleta;

EEPROM: ROM programável pelo usuário e apagável eletricamente (as operações de escrita são bem mais lentas;

Flash-ROM: Este é um tipo especial de ROM que também recebe o nome de *flash memory*, e é muito utilizada nas *flash BIOS* sendo um meio termo entre EPROM e EEPROM. Assim como a EPROM ela pode ser totalmente apagada, mas isso é feito eletricamente em um tempo bem mais curto. Porém ela não permite o apagemento a nível de byte, como faz a EEPROM;

Flash-BIOS: É um tipo especial de BIOS que permite a sua atualização; nos micros antigos a BIOS era gravada em uma memória EPROM e não soria alterações. Nos computadores modernos, a BIOS esta colocada em uma flash-BIOS que pode ser alterada para receber uma nova versão.

1.2 Memória Principal

Uma das primeiras características definidas no projeto de arquitetura do distema Von Neumann consistia de ser uma máquiina "de programa armazenado". O fato de as instruções, uma após a outra,poderem ser imediatamente acessadas pela CPU é que garante o automatismo do sistema e aumenta a velocidade de execução dos programas.

Desde o princípio a memória utilizada para armazenar o programa (e seus dados) a ser executado é a memória que chamamos memória principal, para distingui-la da memória de discos, fitas, CD-ROM, denominadas memória secundária.

A memória principal é, então, a memória básica de um sistema de computação desde seus primórdios. É o dispositivo onde o programa (e sus dados) que vai ser executado é armazenado para que a CPU vá "buscando" instrução por instrução.

No que diz respeito a tecnologia de fabricação a memória principal atualmenmte é fabricada com tecnologia DRAM. A memória DRAM armazena a informação como carga elétrica em um pequeno capacitor (construído com semicondutor). Este capacitor é carregado quando se armazena o bit 1 e é descarregado quando se armazena o bit 0. Só ha um problema, ele é muito pequeno e, devido a correntes de fuga, perde rapidamente sua carga. Assim de tempos em tempos é necessário restaurar sua carga. Isto é feito através de uma operação chamada de repontencialização, refresco ou "*refresh*". Então, as DRAMs realizam três operações básicas: a escrita, a leitura e o refresco. Pode parecer que a DRAM só traz complicações mas é evidente que ela também oferece vantagens como o baixo consumo de energia e o tamanho reduzido da sua implementação.

É claro que seria interessante que o computador usasse apenas SRAM de alta velocidade mas a máquina ficaria muito cara, grande e apresentaria um consumo de energia elevado. A solução encontrada foi a de construir a memória principal com uma tecnologia mais lenta, porém barata, e uma memória cache com tecnogia rápida, porém cara. Em resumo usamos a DRAM para compor a memória principal e a SRAM para a memória cache.

1.2.1 As Modernas DRAM

Para evitar que o chip de memória tenha um número excessivo de pinos, as linhas de endereços são multiplexadas. Um chip de 4M x 1 precisaria de 22 pinos só para os endereços. Acomodar esses 22 pinosjunto com os de controle, dados e alimentação, levaria a um número excessivo de pinos. Devido a essa multiplexação, um chip de 4 Mbits usa apenas 20 pinos.

Os endereços são divididos em dois grupos chamados de "*row address*"ou endereço de linha e "*column address*", ou endereço de coluna. O endereço de linha é formado pelos bits mais significaticos do endereço, enquanto que os endereços de coluna são formados dos bits menos significativos desse endereço.. É necessário informar à memória a porção do endereço que está sendo enviada. Para isso usam-se: "*Row Address Strobe*"(*RAS), ou strobe do endereço da linha , e "*Column Address Strobe* (*CAS), ou strobe do endereço da Coluna. A figura 1.2 mostra um diagrama simplificado de uma memória DRAM.

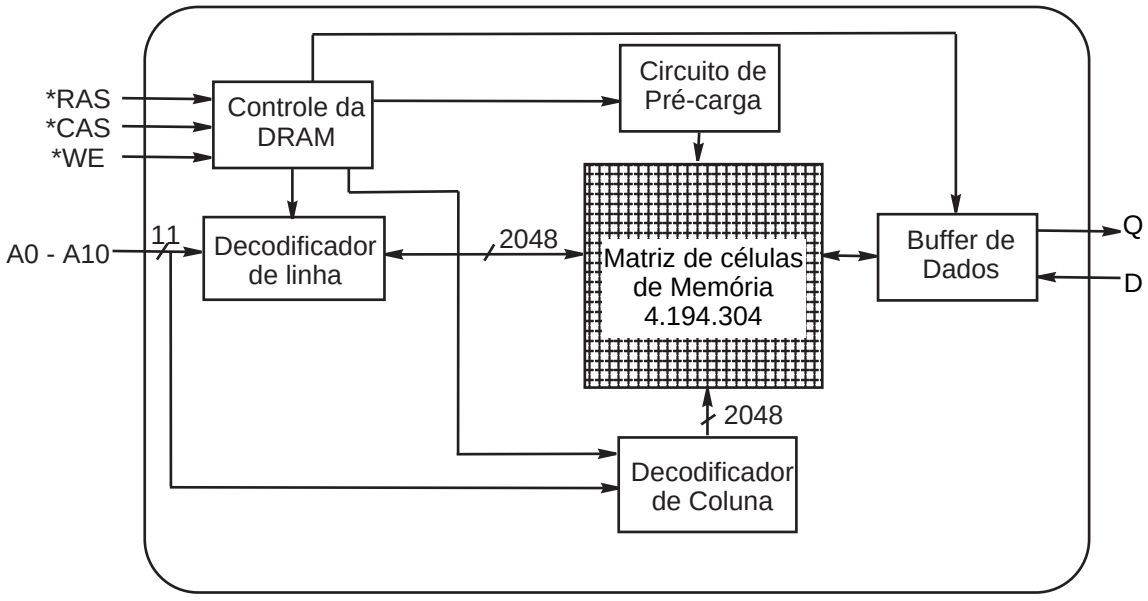

Figura 1.2: Diagrama Simplificado de uma memória DRAM

Na figura a entrada *WE é usada para solicitar uma operação de escrita; As entradas A0-A10 são utilizadas informar o endereço da célula acessada na operação de leitura ou escrita; D

é a entrada do bit a ser escrito enquanto Q é a saída do dado lido.

A seguir são abordadas as memórias FPM RAM, EDO RAM, BEDO RAM, SDRAM, DDR RAM, e RDRAM.

FPM RAM - Fast Page Mode RAM: a tradução do nome seria RAM com modo de página veloz. Esta memória era usada nos antigos 486 com os pentes de 30 pinos. A expressão *"fast page mode"* refere-se ao fato de que a memória espera que o próximo acesso aconteça na mesma linha e que portanto, será necessário enviar apenas o novo endereço da coluna, permitindo que se ganhe tempo nos acessos subsequentes.

As memórias FPM foram muito úteis para as CPUs que trabalhavam com acessos no modo rajada ("*burst*"). Para um acesso em modo rajada, transferindo 4 dados em seqüência, essa memória gasta 5-3-3-3 períodos de relógio, quando o barramento é de 66MHz. A notação 5-3-3-3 significa que gastam-se 5 ciclos de relógio para transferir a primeira linha e a primeira coluna e depois disso 3 períodos para cada uma das colunas subseqüentes. Para um barramento de dados de 64 bits isto significa uma largura de banda de 151 MB/s.

EDO RAM - Extended Data Output RAM: o nome desta memória poderia ser traduzido como RAM com saída de dados estendida. Essas memórias surgiram em 1994 e vieram para atender aos Pentoium que estavam limitados à tecnologias da FPM. A EDO RAM também trabalha com acessos sequenciais, com a diferença que eles acontecem de forma mais rápida. Isto foi possível com a adição de um latch na saída de dados, como mostrado na figura 1.3. Este latch retém o dado na saída, liberando o *CAS para ser usado na transferência do endereço de uma nova locuna. Deve-se lembrar que até então o *CAS controlava o buffer de saída que só estava habilitado enquanto esta linha fosse mantida em nível baixo.

Essas memórias estiveram disponíveis em barramentos de 66 MHz e atenderam aos Pentium acima de 100 MHz. Nesta situação, essas memórias respondiam com ciclos 5-2-2-2. Deve-se notar que o primeiro acesso continua de 5 períodos de relógio, porém os acessos subseqüentes gastam apenas 2 ciclos. Tomando ainda por base um barramento de dados de 64 bits, operando a 66 MHz essa memória permite uma banda passante igual a 192 MB/s, ou seja, um ganho de 27% em relação a FPM.

BEDO RAM - Burst Extended Data Output RAM: a figura 1.4 apresenta um diagrama de blocos de uma BEDO RAM, cujo nome seria traduzido como RAM com rajada de dados estendida. Ela é muito parecida com a EDO RAM, onde foi inspirada. Nas EDO RAM os 4 endereços são consecutivos, assim apenas os dois últimos bits do endereço da coluna sofrem alterações e os demais permanecem estáticos. Pensando nisso, os fabricantes de memória construiram um contador de 2 bits e dispensaram a fase de transferência dos endereços consecutivos. Ou seja, numa BEDO RAM, transfere-seapenas o primeiro endereço da coluna e os 3 seguintes são gerados internamente pelo contador.

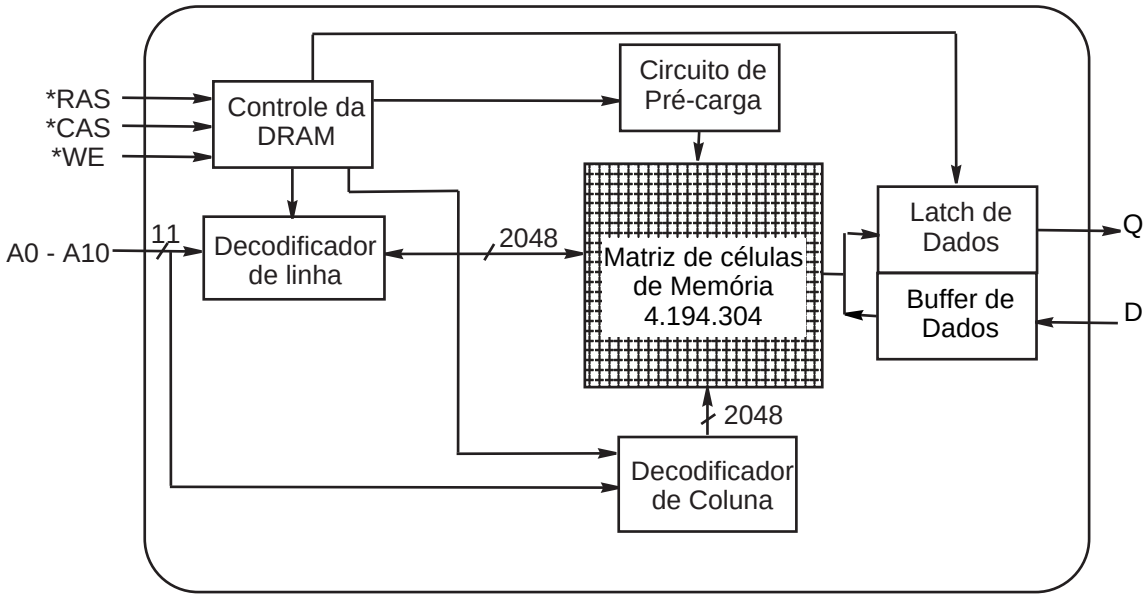

Figura 1.3: Diagrama Simplificado de uma memória EDO DRAM

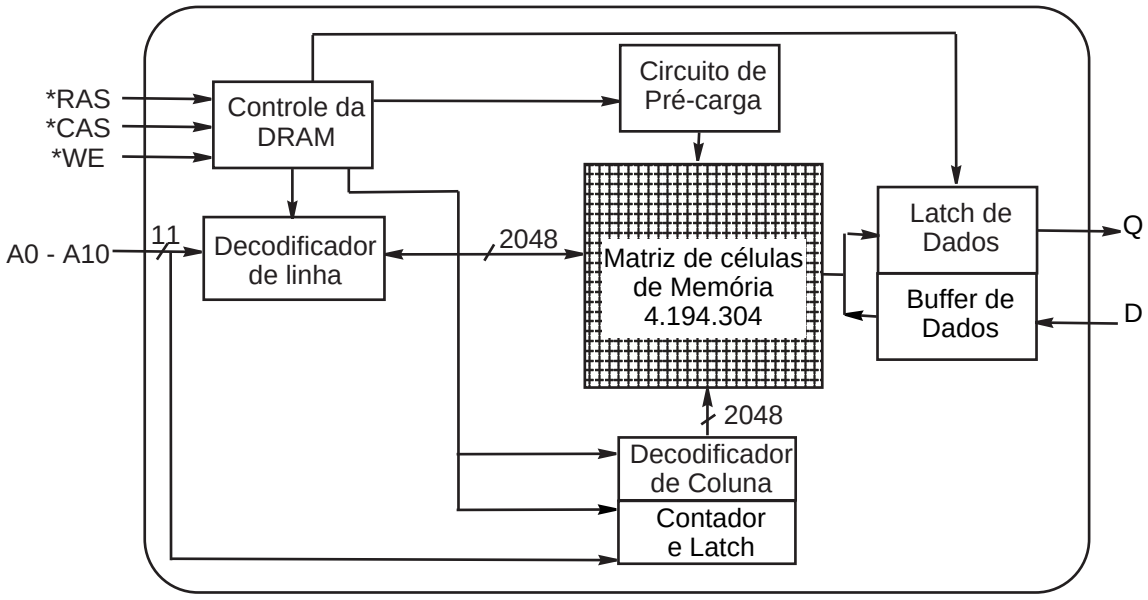

Figura 1.4: Diagrama Simplificado de uma memória BEDO DRAM

As memórias BEDO, quando trabalhando com barramentos de 66 MHz ofereciam ciclos de modo rajada do tipo 5-1-1-1. Deve-se notar que 5 períodos iniciais não puderam ser ser eliminados. Trabalhando com 64 bits ela oferece uma banda passante de 264 MB/s ou

seja um ganho de 38% em relação a EDO RAM.

Esta memória, por dois motivos, nunca teve grande aceitação: primeiro, porque sua tecnologia não permitia trabalhar com barramentos de maior velocidade e, segundo, porque junto com ela surgiu a SDRAM que era mais eficiente e oferecia o potencial de trabalhar com barramentos de 100 MHz.

SDRAM - Syncronous Dynamic RAM: o nome poderia ser traduzido como memória dinâmica síncrona, o que parece estranho, mas de fato, esta memória trabalha sincronizada com o relógio da CPU.

Numa memória síncrona (figura 1.5), o endereço e os sinais de controle são armazenados pela memória. Após um número específico de períodos de relógio a memória começa a disponibilizar os dados em seqüência, um a cada período de clock, sem que seja necessário algum sinal de controle. Os endereços agora são gerados a partir de um contador interno, que fornece os novos endereços de coluna.

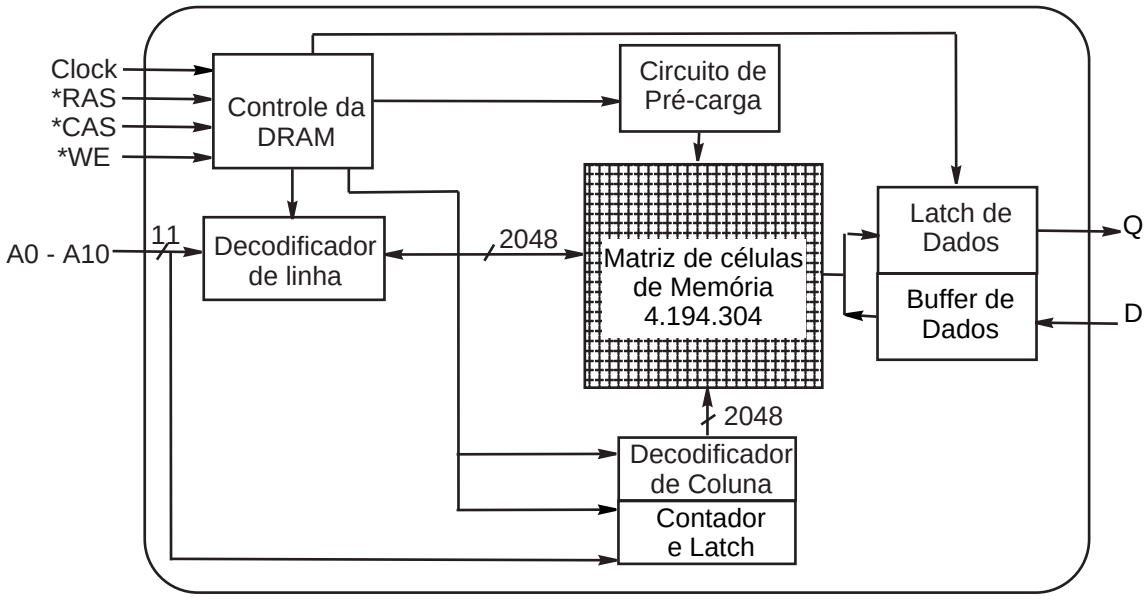

Figura 1.5: Diagrama Simplificado de uma memória SDRAM

Por dispensar sinais de controle durante os acessos seguintes essa memória consegue trabalhar facilmente com barramentos acima de 66 MHz. A SDRAM tem condições de transferir 4 endereços consecutivos a taxa 5-1-1-1. Em um barramento de 64 bits trabalhando a 66 MHz essa memória oferece uma largura de banda de 264 MB/s, ou seja, igual a banda BEDO RAM. Sua grande vantagem está no fato de de operar com barramento de 100 MHz quando então oferece uma banda de 400 MB/s e a 133 MHz, oferecendo uma banda de 532 MB/s. As memórias SDRAM que operam a 100 MHz e 133 MHz são denominadas comercialmente de PC100 e PC133.

DDR SDRAM - Doubled Data Rate SDRAM: essa memória, cujo nome é traduzido como SDRAM com taxa de dados dobrada, não traz grandes novidades. Ela trabalha com duas memórias SDRAM em paralelo permitindo assim dobrar a taxa de transferência de dados. Ela é capaz de entragar dois dados por cada período do relógio, um sincronizado com o flanco de subida e outro com o flanco de descida. A memória gasta 1 período de relógio para receber o comando e 1/2 período para cada transferência.

DR DRAM - Direct Rambus DRAM: esta memória é construída com base na tecnologia de propriedade de uma companhia denominada Rambus Inc.. Nos dias de hoje, com o interesse em barramentos rápidos, a Intel fez parceria com essa companhia para comercializar memórias mais rápidas. Na verdade a especificação Rambus está na sua terceira versão e vem sendo usada em diversos equipamentos, como, por exemplo, o console de jogos da Nintendo.

A rambus é constituída por uma arquitetura totalmente nova, baseada em diversos componentes:

- RCM: Rambus Channel Master, que trabalha como mestre do barramento;
- RC: Rambus Channel, que é o canal para o tráfego de dados;
- RCL: Rambus Channel Slave, que são os dispositivos de memória e;
- RSL: Rambus Signalling Logic, que é a tecnologia de sinalização.

O RCM que atua como controlador da memória está localizado em uma estremidade e as memórias (RCS) são espalhadas ao longo de barramento (RC), onde cada barramento possui uma terminação, como mostrado na figura 1.6.

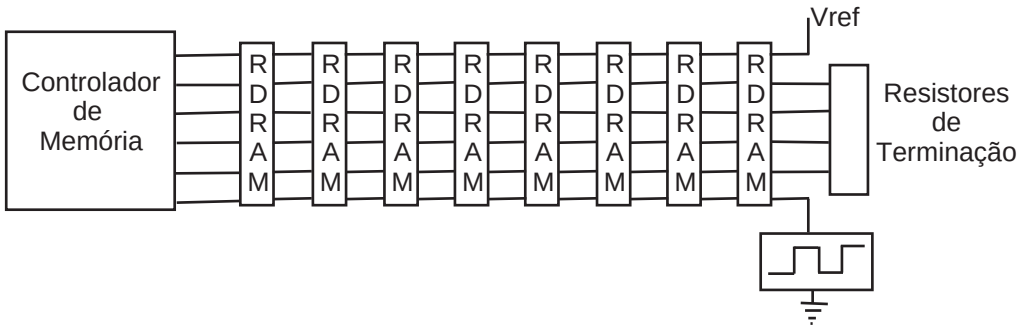

Figura 1.6: Diagrama em blocos de um canal RAMBUS

A largura do canal costuma ser de 6 bits. As placas de memória terão barramento de dados com uma largura de 128 bits ou então 144 bits com 1 bit de ECC para cada byte. É empregado um pequeno conjunto de sinais de alta velocidade para controlar o fluxo de informações numa frequência de até 800 MHz. Isto leva a uma banda passante de 1.600 MB/s ou 1.6 GB/s.

1.2.2 SIMM e DIMM

A sigla SIMM significa "*Single In line Memory Module*"(Módulo de Memória em Linha Única), constrastando com a sigla DIMM "*Dual In line Memory Module*"(Módulo de Memória em Linha Dupla). Os termos especificam apenas como é feito o empacotamento das memórias RAM e não o tipo de memória. É possível usar estes dois empacotamentos para qualquer tipo de memória. Tem havido a tendência de se usar DIMM para empacotar as memórias SDRAM. O benefício é que o DIMM trabalha com barramento de 64 bits formando um banco que pode ser usado individualmente com os processadores Pentium.

30, 72, 168 e 184 Pinos

É usual encontrar módulos SIMM (vulgarmente chamados "pentes") de 30 ou 72 pinos, enquanto o empacotamento DIMM usa 168 pinos. Um módulo SIMM possui um barramento de dados de 8 bits, assim, para comunicar-se com a CPU são necessários 4 módulos SIMM de 30 pinos para formar um barramento de 32 bits (lembre-se dos 486 que exigia colocação que preenchessemos os bancos de memória de 4 em 4. A evolução foi o SIMM de 72 pinos que tem um barramento de 32 bits. Apenas um desses módulos atende o 486, porém não o Pentium que possui um barramento de 64 bits necessitando portando de pares de módulos SIMM 72 pinos.

Motivado pelo mercado de máquinas Pentium surgiram os DIMM com 168 pinos, oferecendo um barramento de 64 bits. Hoje há uma tendência de se oferecerem memória SDRAM em módulos DIMM de 168 pinos. As memórias RDRAM, por sua vez, são entregues em um pente de 184 pinos e oferecerão um barramento de dados de 128 bits, ainda com a possibilidade de usar um bit de ECC para cada byte.

1.3 Memória Cache

Cache foi o nome escolhido para designar o nível da hierarquia de memória entre o processador e a memória principal. Assim, a memória cache é um dispositivo de memória colocado entre a CPU e a memória principal, com o intuito de aumentar o desempenho do sistema. Possui tecnologia de fabricação semelhante a CPU (VLSI) garantindo redução da espera por parte desta.

As caches apareceram primeiro nas máquinas desenvolvidas para pesquisa, no início da década de 60, sendo implementadas em máquinas comerciais mais tarde, ainda na mesma década; virtualmente todas as máquinas comerciais desenvolvidas hoje, desde as mais rápidas até as mais lentas, incluem cache.

Parâmetros:

- tempo de acesso: atualmente na ordem de 10 a 25 ns, razão pela qual é colocada na pirâmide logo abaixo dos registradores;

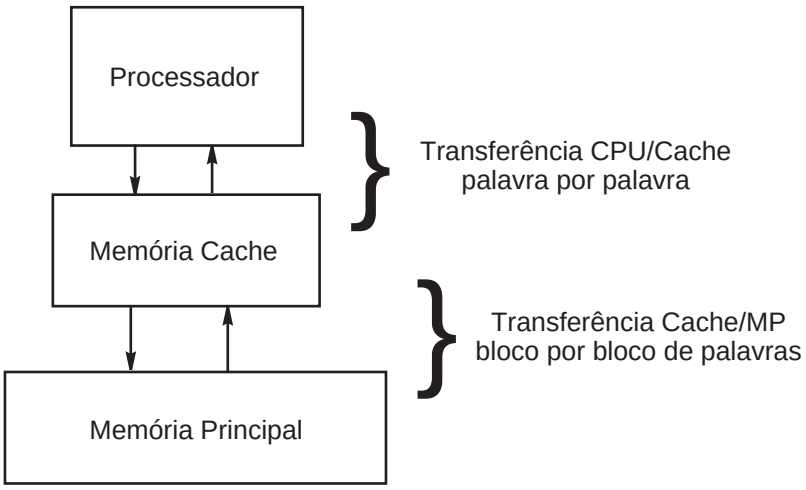

Figura 1.7: Organização para transferência de informações entre CPU/Cache/MP

- capacidade: atualmente oscilam entre 256Kb e 1Mb entre cache primária e secundária;
- volatilidade: construída com circuitos eletrônicos, requer energia elétrica para seu funcionamento. Portanto são voláteis;
- custo: mais baratas que os registradores e mais caras que a MP. Ainda assim seu custo é alto;
- temporariedade: tempo de permanência de dado ou instrução é relativamente pequeno. Mesmo assim é dependente de uma política de substituição de informação.

Nos sistemas computacionais atuais existe uma grande diferença de velocidade entre CPU e Memória principal. Comparativamente o ciclo de instrução é dezenas/centenas de vezes mais rápido que o ciclo de memória. Esta diferença faz com que a CPU introduza tempos de espera no acesso aos dados da MP. O número de ciclos de espera na transferência de dados inseridos pela CPU depende do tipo de memória principal (EDO, FPM, SDRAM, RDRAM, etc) e da diferença de velocidade entre a CPU e a MP;

Com o tempo o problema se agrava pois o desempenho das CPU dobra a cada 18 meses aproximadamente, enquanto as velocidades de transferência da memória permanecem basicamente iguais.

1.3.1 Conceitos

Suponha que você necessite pesquisar sobre a evolução da arquitetura de computadores na biblioteca da universidade. Imagine que você esteja sentado em uma mesa com um conjunto de

livros sobre o assunto que você selecionou previamente, retirou das estantes e colocou sobre a mesa. Ao examinar os livros em seu poder, você observa que eles contêm a descrição de várias máquinas e tecnologias, com exceção da arquitetura proposta por Von Neumann. Por causa disto você retorna às estantes em busca de um livro adicional que trate sobre o assunto. Caso tenha selecionado bem os livros que se encontram sobre sua mesa existe uma grande probabilidade de você encontrar neles a maioria dos tópicos que você precisa para o seu trabalho. O fato de você possuir vários livros a sua frente faz com que o tempo necessário para escrita do trabalho seja bem menor, quando comparado ao tempo gasto na escrita do mesmo trabalho caso você nao pudesse ter mais de um livro em seu poder e precisasse constantemente levantar-se da sua mesa para levar um livro até a estante e trocá-lo por outro.

O mesmo princípio permite criar a ilusão de uma memória muito grande que possa ser acessada de modo tão rápido quanto é uma memória pequena. Assim como você não precisa acessar todos os livros de uma biblioteca de uma vez, com igual prioridade, um programa não acessa todo o seu código e todos os seus dados de uma vez com igual probabilidade. Se assim fosse seria impossível fazer com que os acessos a essa memória fossem rápidos. Pela mesma linha de raciocínio seria impossível para você colocar todos os livros de uma biblioteca em cima da sua mesa, mantendo a chance de encontrar rapidamente a informação que você precisa.

O chamado *princípio da localidade* está por trás tanto da maneira como você escreve o trabalho quanto o modo como os programas operam. Este princípio estabelece que os programas acessam uma parte relativamente pequena do seu espaço de endereçamento em um instante qualquer, assim como você acessa uma parcela extremamente pequena dos recursos de uma biblioteca, em um dado instante. Existem dois diferentes tipos de localidade:

1. localidade temporal: se um item é referenciado, ele tende a ser referenciado novamente dentro de um curto espaço de tempo. Se você tiver trazido um livro para sua mesa para examiná-lo é provável que em breve você vá precisar dele novamente.

2. localidade espacial: se um ítem é referenciado, ítens cujos endereços sejam próximos a eles tendem a ser logo referenciados. Quando você lê/pesquisa sobre determinado tópico na sequência você já está lendo o tópico consecutivo, adjascente ao primeiro.

Assim como os acessos aos livros da biblioteca que estão sobre a mesa exibem localidade, tal propriedade aparece nos programas de computador, pura e simplesmente em função de sua própria estrutura. Por exemplo, a imensa maioria dos programas contém loops (for, while, repeat, do, etc), de maneira que as instruções e os dados tendem a ser acessados de maneira repetitiva, exibindo um alto grau de localidade temporal. Considerando que as instruções são quase sempre acessadas seqüencialmente, os programas também exibem um alto grau de localidade espacial.

No exemplo da figura 1.8 um certo programa é constituído de um grupo de instruções iniciais realizadas em seqüência (Seqüência 1) dois loops e o resto do código (Seqüência 2). Assim

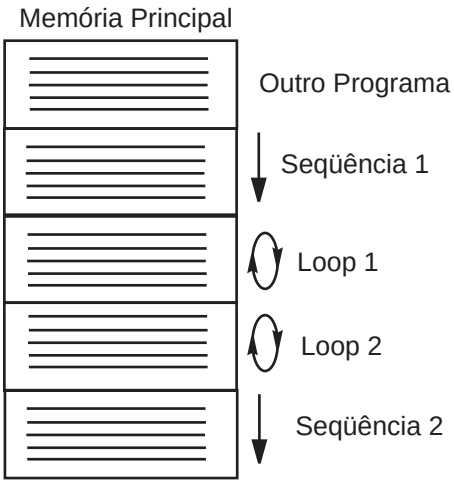

Figura 1.8: Exemplo dos princípios de localidade espacial e temporal

sendo, quando o programa é iniciado as intruções da seqüência 1 são consecutivamente executadas uma após a outra até qua a região de código constituída pelo loop 1 seja executada. A localidade espacial aparece explicitamente na execução de cada uma das seqüências. A localidade temporal, por sua vez, pode ser percebida quando da execução de qualquer um dos loops, cuja sequência de instruções nele contidas será executada de tempos em tempos (a cada repetição do laço).

1.3.2 Uso da Memória Cache

Pode-se tratar o sistema como tendo dois níveis de memória o superior (neste caso cache) e o inferior(neste caso MP) e a unidade nínima de informação que pode ou não estar presente nesta hierarquia de dois níveis é o bloco (na analogia com a biblioteca um bloco corresponde a um livro).

Se a informação solicitada pelo processador estiver presente no nível superior (cache), ocorre um *acerto*. Se a informação não puder ser encontrada no nível superior, a tentativa de encontrá-la gera uma *falta*. Quando ocorre uma falta o nível mais baixo da hierarquia (MP) é acessado para que seja possível recuperar o bloco com a informação solicitada pelo processador (eqüivalente a levantarmos da mesa e irmos até a estante buscar um livro que contenha a informação que desejamos e que não existia nos que já estavam na mesa). A *taxa de acertos*, ou *razão de acertos*, corresponde a fração dos acessos à memória encontrados no nível superior, e com freqüência é usada como medida de performance da hierarquia de memória. A *taxa de faltas* (1 - taxa de acertos) é a fração de acessos à memória não encontrados no nível superior (na cache).

Considerando que a melhora da performance é o principal fato motivador da adoção do conceito de hierarquia de memória, a taxa de acertos é um parâmetro extremamente importante para o sucesso da implementação do conceito.

Define-se como *tempo de acerto* o tempo necessário para acessar o nível superior da hierarquia (cache), que inclui o tempo necessário para determinar se o acesso a informação vai gerar um acerto ou uma falta (ou seja, o tempo necessário para o exame dos livros sobre a mesa, em busca da informação necessária).

O *penalidade por falta* é o tempo necessário para substituir um dos blocos do nível superior, pelo bloco do nível inferior contendo a informação desejada, mais o tempo para enviar a informação ao processador. Em razão do nível superior (cache) ser menor e de ser constituído de memórias mais rápidas, o tempo de acerto é muito menor que o tempo necessário ao nível mais baixo da hierarquia (MP). Justamente este tempo de acesso é o maior tempo dentro da penalidade por falta (o tempo para examinar os livros sobre a mesa é muito menor do que o tempo de ir até as extantes buscar informações contidas em outro livro).

Para que haja aumento de desempenho, com a inclusão da memória cache, é necessário que exista mais acertos do que faltas. Isto implica no dimensionamento adequado da memória cache como um todo e principalmente do tamanho de cada bloco.

1.3.3 Elementos de Projeto de uma Cache

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

- Definição do tamanho da Cache;
- Função de mapeamento de dados entre Memória Principal e Memória Cache;
- Algoritmo de substituição de dados na Cache;
- Política de escrita na Cache.

Tamanho da Memória Cache

Anteriormente foi colocado que uma memória cache só é produtiva se, durante a execução de um programa ocorrerem mais acertos (dado requerido se encontra na cache) do que faltas, no acesso a cache.

O objetivo principal no dimensionamento da memória cache está na maximixação da taxa de acertos e a redução da penalidade por falta do sistema.

A definição da faixa de tamanho adequada para uma cache depende de uma série de fatores, específicos de cada sistema, tais como: tamanho da memória principal; relação de acertos/faltas pretendida; tempo de acesso da MP; custo médio por bit da Mp e da M. Cache; tempo de acesso da cache; natureza do programa em execução (localidade).

O dimensionamento do tamanho da memória cache é influenciado diretamente pela política de mapeamento utilizada, tendo em vista que cada uma delas oferece maior ou menor flexibilidade na colocação dos dados na mesm, bem como, maior ou menor quantidade de circuitos de controle.

Políticas de Mapeamento entre MP/Cache

Até agora foram abordados os conceitos e benefícios do uso da memória cache, entretanto aspectos de implementação da mesma não foram abordados. Os aspectos significativos da implementação dizem respeito as seguintes questões: Quando solicitado um determinado endereço de memória, em qual localização da memória cache o mesmo deve ser colocado? Uma vez dispostos os dados na memória cache, como se faz para localizá-los dentro da mesma?

Estas questões dizem respeito a política de mapeamento utilizada na implementação da memória cache. No entanto, antes de abordar cada política de mapeamento veremos alguns aspectos que são inerentes a implementação da memória cache, independente da política de mapeamento utilizada.

A memória principal é formada por um conjunto sequencial **N** = 2^E células de memória, cada uma possuindo um endereço único com **E** bits. Esta memória é dividida em blocos (**B** - numerados de 0 a B-1), cada um constituído de K células de memória.

Assim a quantidade de blocos de memória é dado por: $B = \frac{N}{K}$ ou $B = \frac{2^E}{K}$. Em outras palavras a memõria principal é constituída de **B** blocos cada um com **K** células de memória.

A memória cache, por sua vez, é constituída de Q linhas, denominadas quadros, cada uma podendo armazenar K células. Aqui reside outro aspecto importante, obrigatoriamente o tamanho do bloco (**B**) da memória principal deve ter o mesmo tamanho do quadro (**Q**) da memória cache. Assim a quantidade de células de memória (**K**) existente em um bloco da MP é a mesma existente em um quadro da memória cache. A figura 1.9 ilustra esta situação.

Sabendo-se o tamanho do bloco (ou quadro) é possível saber a qual bloco da MP determinado endereço pertence, pois:

$$Número\ do\ Bloco\ =\ \frac{Endereco}{K}$$

onde somente a parte interia deve ser considerada. O deslocamento dentro do bloco, por sua vez, é dado por:

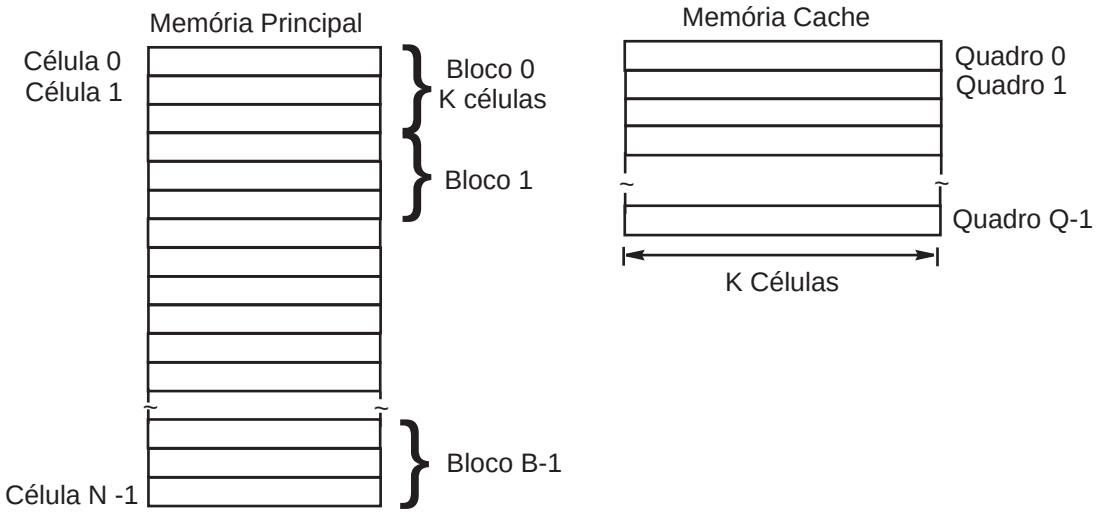

Figura 1.9: Estrutura entre MP e Cache para transferência de dados

$$Deslocamento\,no\,Bloco\ =\ Endereco\ MOD\ K$$

onde MOD é a operação matemática resto da divisão.

Da mesma forma, sabendo-se a quantidade de blocos (B) e o tamanho de cada bloco (K) é possível determinar a quantidade total de células de memória (N):

$$N = B * K$$

por conseguinte é possível determinar a quantidade de bits usada no endereçamento da memória já que N = 2^E, onde E é quantidade de bits.

Tendo o endereço expresso em bits e sabendo-se o número de blocos existentes na memória principal, pode visualizar a que bloco um determinado endereço de memória pertence extraindo os bits mais significativos deste endereço. Por exemplo em uma MP com 256 células de memória (endereço de 8 bits) que possui K = 8 teremos B = 32. Assim, os 5 bits mais significativos do endereço representam o número do bloco. O endereço 01001011 do exemplo acima pertence ao bloco 9 (em binário 01001). Já a parte menos significativa do endereço nos informa qual o deslocamento da célula dentro do bloco. No exemplo acima o deslocamento é 3 (em binário 011)

Exemplo: Suponha uma memória principal com 256 blocos, cada um com tamanho de 8 células de memória. Pergunta-se: a) Qual a quantidade total de memória? b) Qual o tamanho do endereço? c) A qual bloco pertence a célula de memória cujo endereço é 713?

Respostas: De acordo com o enunciado temos B = 256 e K = 8.

a) A quantidade de memória é dada por $N = B * K$, assim temos: N = 256 * 8 = 2048 células de memória.

b) A quantidade de bits do endereço é dada por N = 2^E, como: 2048 = 2^E, temos E = 11, ou seja os endereços são de 11 bits.

c) O bloco a qual pertence o endereço 713 é dado por: $Número\ do\ Bloco = \frac{Endereco}{K}$, assim: $Número\ do\ Bloco = \frac{713}{8} = 89$, ou seja o endereço 713 pertence ao bloco 89.

A cada instante a memória cache possui um grupo de blocos da MP armazenados em seus quadros. Porém, como Q $<$ B (há mais blocos que quadros) não é possível armazenar todo o conteúdo da memória cache simultâneamente. Tampouco um quadro da cache pode estar dedicado a armazenar unicamente um bloco específico da MP.

Assim, um quadro é utilizado por mais de um bloco da MP e, por conseguinte é necessário identificar, em cada instante, qual o bloco específico que está utilizando determinado quadro da memória cache. Para isso, cada quadro contém um campo denominado rótulo, que contém a identificação do bloco e que, como será visto adiante, faz parte dos bits do endereço completo da MP.

1.3.4 Políticas de Mapeamento MP/Memória Cache

Para efetuar a transferência de um bloco da MP para um quadro da memória cache temos três procedimentos de mapeamento. São as chamadas políticas de mapeamento, quais sejam:

- mapeamento direto;
- mapeamento associativo;
- mapeamento associativo por conjunto;

Mapeamento direto

Por esta técnica, cada bloco da MP tem um quadro da cache previamente definido onde poderá ser armazenado.

Como há mais blocos do que quadros da cache, muitos blocos serão destinados a um mesmo quadro, sendo necessário definir a regra a ser seguida para a escolha do quadro que cada bloco será destinado.

Para definir quais blocos da MP serão alocados a um quadro específico , efetua-se um cálculo usando-se aritmética de módulo:

$$QD = NB \,\%\, Q$$

onde:

QD = número do quadro de destino na cache

NB = número do bloco da MP

Q = número de quadros da cache

Para entendermos este mapeamento vamos apresentar um exemplo simples, com uma memória principal com apenas 8 células e uma memória cache com 2 quadros. O tamanho de cada bloco da MP (e por conseqüência de cada quadro da cache) é 2 células.

De acordo com o exposto temos: N = 8, B = 4, Q = 2 e K = 2. Os 4 blocos da memória principal são numerados de 0 a 3 e, para sabermos o quadro da cache ao qual cada bloco está destinado basta aplicarmos a equação acima. Assim:

QD	=	NB	%	Q
0	=	0	%	2
1	=	1	%	2
0	=	2	%	2
1	=	3	%	2

Dado a “geometria”da questão, pode-se perceber que os blocos 0 e 2 são destinados ao quadro 0 da cache e os blocos 1 e 3 são destinados ao quadro 1 da cache, conforme podemos visualizar na figura 1.10.

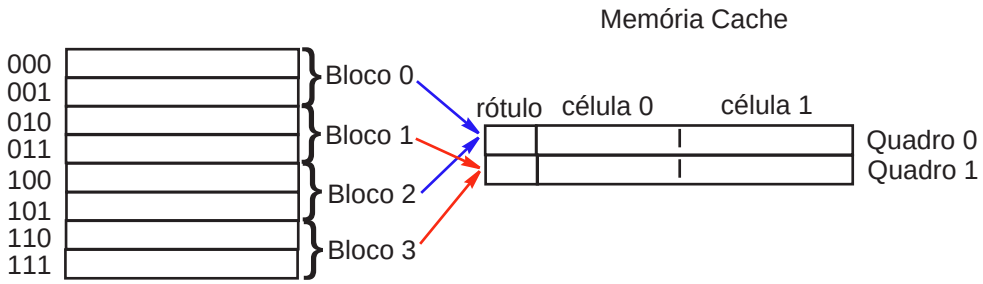

Figura 1.10: Mapeamento Direto

No exemplo apresentado cada quadro da memória cache é “disputado”por dois blocos da memória principal, assim sendo o campo rótulo (presente na cache) é colocado para definir se em um dado momento encontra-se o bloco 0 ou o bloco 2 no quadro 0 (da mesma forma acontece para o quadro 1).

Perceba que, neste exemplo, para definir qual dos dois blocos está presente, faz-se necessário apenas 1 bit no campo rótulo. Quando estiver presente o bloco 0 o bit do rótulo recebe 0 e, quando estiver presente o bloco 2 o bit do rótulo recebe 1.

Vamos supor agora uma memória principal com 64 células de memória dividida em blocos de 4 células e uma memória cache de 4 quadros. Para este sistema teremos então: N = 64; B = 16; K = 4; Q = 4, como pode ser visto na figura 1.11.

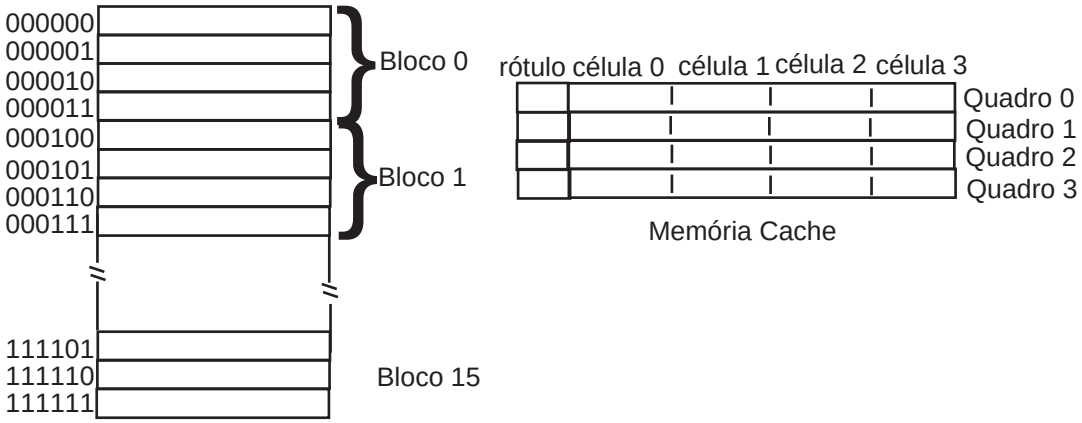

Figura 1.11: Mapeamento Direto

No exemplo acima os blocos 0, 4, 8 e 12 disputam o quadro 0 da cache, os blocos 1, 5, 9, 13 disputam o quadro 1 e assim sucessivamente. Desta forma temos 4 blocos disputando cada quadro da cache o que torna necessário um rótulo de 2 bits para informar qual dos 4 rótulos possíveis se encontra no quadro ($2^2 = 4$).

No mapeamento direto o endereço de memória é dividido em três regiões para se fazer o mapeamento absoluto da célula da memória principal na memória cache:

- bits menos significativos → deslocamento dentro do quadro;
- bits intermediários → número do quadro;
- bits mais significativos → rótulo da cache.

Assim, temos

Perceba que para o exemplo anterior cuja memória é de 64 células e portanto os endereços de 6 bits a seguiinte divisão do endereço é obtida. Como temos 4 células dentro de cada bloco precisamos de 2 bits para expressar qual o deslocamento; como temos 4 quadros na cache precisamos de 2 bits para expressar qual o número do quadro de destino na cache, e; como temos 4 blocos disputando um único quadro na cache temos 2 bits para o rótulo.

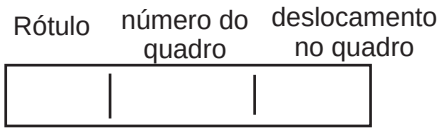

Figura 1.12: Divisão do endereço no Mapeamento Direto

Funcionamento de uma operação de leitura

Uma cache implementada com mapeamento direto terá o seguinte comportamento durante uma operação de leitura na memória

1. CPU apresenta endereço de 6 bits ao circuito de controle da cache; este identifica os campos para definir se a palavra esta na cache;

   endereço solicitado = 100110_2

2. Os 2 bits centrais são examinados para determinar o quadro: assim temos quadro $XX01XX_2$ = Quadro 1.

   Resta verificar se o bloco solicitado encontra-se no quadro 1

3. O controlador de cache examina por comparação o valor do rótulo do quadro 1 da cache com os 2 bits mais significativos do endereço solicitado ($10XXXX_2$). Caso sejam iguais realiza o passo 4, senão o passo 5;

4. Caso sejam iguais os rótulos a célula 2 (conforme solicitada nos 2 bits menos significativos do endereço - $XXXX10_2$) do quadro 1 tem seu conteúdo transferido para a CPU;

5. Se no passo 3 a comparação resultasse negativa (ou seja o bloco desejado não se encontrava no quadro 1), o mesmo deveria ser transferido da MP para a cache, substituindo o bloco atual. Para esta última tarefa seriam utilizados os 4 bits mais significativos do endereço solicitado ($1001XX_2$)já que estes informam o número do bloco a ser transferido da MP para memória cache;

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 (no exemplo 4 blocos estão destinados ao mesmo quadro), o que significa que somente um pode estar lá de cada vez).

Se, por exemplo, durante a execução de um programa, um dado código fizer repetidos acessos a endereços situados em blocos alocados ao mesmo quadro, então haverá a necessidade de sucessivas idas a MP para a substituição de blocos e a relação *acerto/falha* será baixa, com a conseqüente redução de desempenho do sistema.

Mapeamento Associativo

Este tipo de mapeamento tem uma organização oposta ao anterior, pois agora, cada bloco da memória principal pode ocupar qualquer quadro da cache, ou seja, os blocos não possuem um quadro fixado previamente para o seu armazenamento, podendo ser colocados em qualquer quadro da cache, conforme pode ser visualizado na figura 1.13

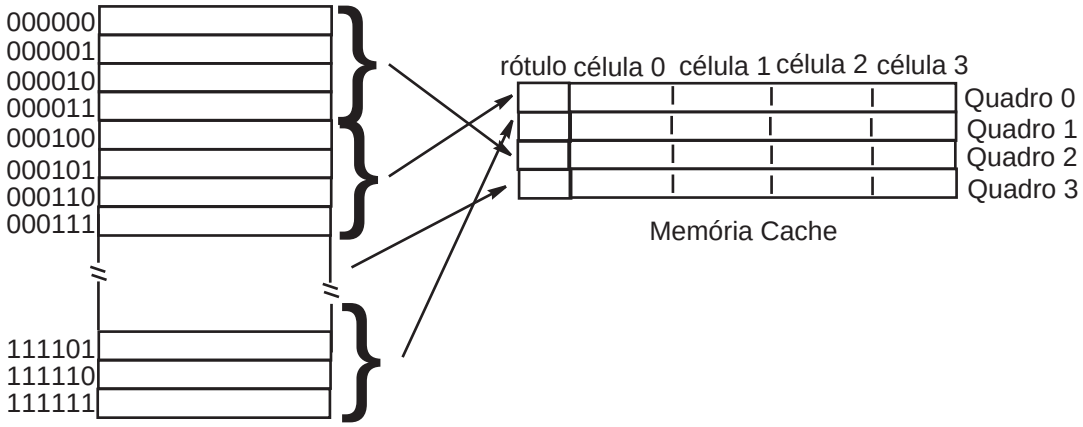

Figura 1.13: Memória Cache com Mapeamento Associativo

Se for verificado que o bloco não está armazenado em nenhum quadro da cache, então o bloco será transferido para a cache, substituindo um bloco já armazenado nela. Que quadro será substituído é um problema a ser resolvido de acordo com o algoritmo de substituição adotado.

O endereço de memória, no mapeamento associativo, é dividido em duas partes:

- os bits mais significativos como rótulo;
- os bits menos significativos como deslocamento dentro do bloco.

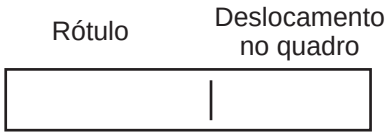

Figura 1.14: Divisão do endereço no Mapeamento Associativo

A diferença é que agora como não a predefinição quanto ao quadro da cache onde o bloco deve ser armazenado o campo rótulo tem o tamanho do número do bloco, ou seja, o rótulo do quadro da cache armazena o número do bloco quem ali se encontra.

Utilizando o exemplo anterior (MP com 64 células), sempre que a CPU realizar um acesso o controlador da memória cache deve comparar os 4 bits mais significativos relativos ao número do bloco ($1001XX_2$) com o rótulo de cada um dos quadros da cache para verificar se o bloco desejado está ou não na cache.

Caso afirmativo, o conteúdo da célula desejada é transferido, caso contrário, o número do bloco solicitado é usado para ir até à MP e trazer o bloco, substituindo um bloco existente conforme o algoritmo de substituição.

Embora esta técnica evite a fixação dos blocos nos quadros, por outro lado acarreta a necessidade de uma lógica complexa para, rapidamente, examinar o campo rótulo de todos os quadros da cache a fim de localizar o dado solicitado.

Outro aspecto nesta política de mapeamento é que o campo rótulo presente na memória cache necessariamente é maior que no caso do mapeamento direto tendo em vista que agora temos que especificar qualquer dos blocos da MP e no direto somente os blocos que estavam destinados para o quadro em questão.

Mapeamento Associativo por Conjuntos

Essa técnica tenta resolver o problema de conflito de blocos em um mesmo quadro (da técnica de mapeamento direto) e o problema da técnica de mapeamento associativo de exaustiva busca e comparação do campo rótulo de toda a memória cache. É, assim, um compromisso entre as duas técnicas anteriores.

A técnica consiste em organizar os quadros da cache em grupos denominados conjuntos. Desta forma, dentro de um conjunto teremos vários quadros.

Genericamente a cache é dividida em C conjuntos com capacidade para D quadros. A figura 1.15 ilustra uma memória cache com 4 conjuntos, cada um deles com 2 quadros (D = 2) sendo cada quadro com tamanho K = 4.

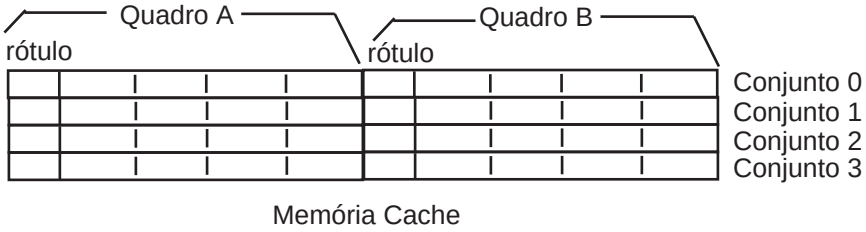

Figura 1.15: Cache com 4 Conjuntos

Para definir quais blocos da MP serão alocados a um determinado conjunto, efetua-se um cálculo usando-se aritmética de módulo:

$$CD = NB \,\%\, C$$

onde:

CD = número do conjunto de destino na cache

NB = número do bloco da MP

C = número de conjuntos da cache

Tal operação é análoga ao mapeamento direto sendo pré-definido o conjunto de destino de cada bloco. Entretanto, uma vez que vários blocos podem ser destinados ao mesmo conjunto e pelo fato de cada conjunto ser capaz de armazenar mais de um quadro, é possível (no exemplo anterior) haver dois blocos simultâneamente em um determinado conjunto.

Para determinar qual dos blocos destinados a um determinado conjunto estão ali armazenados em um determinado momento deve-se comparar o rótulo de cada um dos quadrops presentes naquele conjunto (exatamente como no mapeamento associativo). Caso encontre-se um rótulo igual então basta buscar dentro do quadro o deslocamento relativo ao bloco solicitado. Caso não haja deve-se escolher através de uma política de substituição qual dos blocos presentes neste coonjunto será substituído pelo bloco solicitado.

No mapeamento associativo por conjuntos a divisão do endereço é feito da seguinte forma:

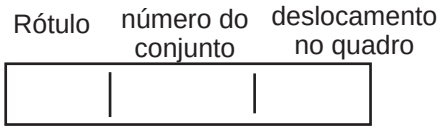

Figura 1.16: Divisão do endereço no mapeamento associativo por conjunto

Desta forma, como pode-se observar, os bits menos significativos indicam o deslocamento dentro do quadro, os bits intermediários representam o conjunto de destino do bloco referenciado e, os bits mais significativos representam o rótulo.

Para exemplificar vamos utilizar uma memória principal composta de 64 células de memória contendo 16 blocos. A memória cache, por sua vez, é formada de dois conjuntos possuindo 2 quadros por conjunto, conforme pode ser visualizado na figura 1.17.

Dada as informações acima extraímos os seguintes dados: N = 64, B = 16, K = 4, C = 2; D = 2. Assim, temos pré-definidos os conjuntos de destino de cada um dos blocos, dado por:

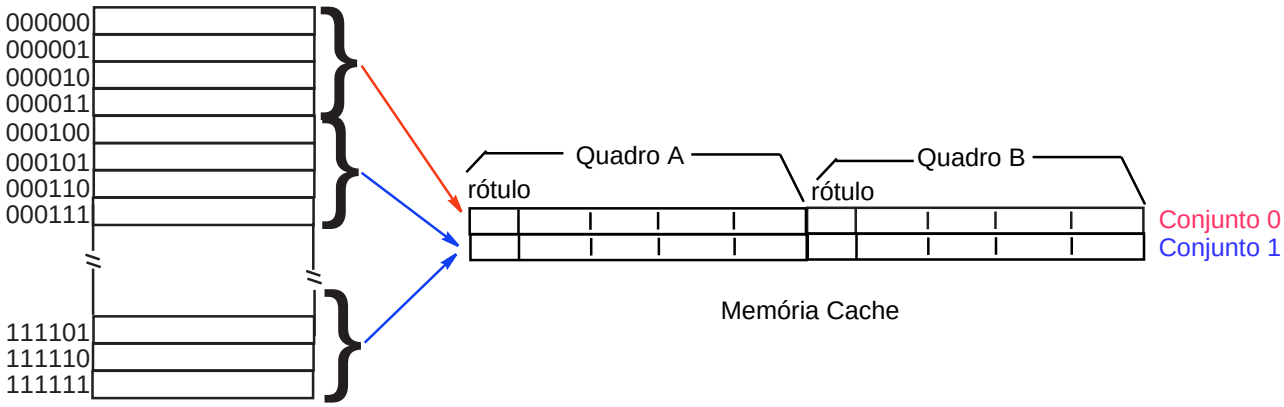

Figura 1.17: Exemplo utilizando mapeamento associativo por conjunto

CD	=	NB	%	C
0	=	0	%	2
1	=	1	%	2
0	=	2	%	2
1	=	3	%	2
⋮		⋮		⋮
1	=	15	%	2

Para este sistema temos o endereço dividido em 2 bits para o deslocamento (tendo em vista que dentro de um quadro existem 4 células de memória) 1 bit para conjunto de destino (a memória cache é formada de dois conjuntos) e 3 bits de rótulo (já que 8 rótulos disputam o mesmo conjunto da cache).

Caso a CPU solicite o endereço de memória 110110_2 o controlador da memória cache deve extrair o bit do número do conjunto para verificar qual é o conjunto de destino, neste caso, $XXX1XX_2$, portanto conjunto 1. Após deve comparar o rótulo do endereço solicitado $110XXX_2$ com o rótulo de cada um dos quadros presentes mo conjunto 1. Caso exista um rótulo idêntico então o dado encontra-se na cache e basta obter o deslocamento dentro do quadro ($XXXX10_2$ - deslocamento 2) para extrair a informação e repassá-la para CPU. Caso não exista nenhum rótulo na cache idêntico ao solicitado, então o controlador deve providenciar a substituição de um dos quadros da cache (conforme a política de substituição adotada) pelo bloco solicitado, neste caso, $1101XX_2$ (bloco 13).

1.3.5 Políticas de Substituição de Dados na Cache

Uma questão que se faz presente diz respeito a qual dos blocos armazenados na cache deve ser retirado para dar lugar a um novo bloco que está sendo transferido.

Dependendo de qual técnica de mapeamento se utiliza, pode-se ter algumas opções de algoritmos. Por exemplo, se o método de mapeamento adotado é o direto, então não há o que definir, pois neste caso, há somente um único quadro possível para um dado bloco (de acordo com a descrição do método).

No entanto, para os outros dois métodos de mapeamento (associativo e associativo por conjunto) pode-se optar por um dos seguintes algoritmos de substituição:

- *O que não é usado a mais tempo* (LRU - Least Recently Used): o sistema de controle da cache escolhe para ser substituído o que está a mais tempo sem ser utilizado.
- *Fila*: ou seja o primeiro a chegar é o primeiro a ser atendido (FIFO - First In First Out). O sistema de controle da cache escolhe para ser substituído o que esta a mais tempo na cache, independente de estar sendo o mais usado ou não pela CPU.
- *O que tem menos referências* (LFU - Least Frequently Used): o sistema de controle da cache escolhe o bloco que tem o menor número de acesso por parte da CPU (menos referenciado).
- *Escolha Aleatória*: trata-se de escolher aleatoriamente u bloco a ser substituído, independente de sua situação em relação aos demais.

1.3.6 Políticas de Escrita de Dados na Cache

Suponha a execução de uma instução *store* (armazenar), com o dado sendo esrcito somente na memória cachem (sem atualizar a memória principal). Após a escrita na cache, a memória principal terá, então, um valor diferente daquele que estiver na cache. Nesse caso, diz-se que a cache e a memória principal são *inconsistentes*. A meneira mais simples de manter a consistência da cache e da memória principal é escrever o dadotanto na memória cache quanto na memória principal. Este esquema é chamado *write-through* (escrita em ambas).

Um outro aspecto a ser considerado no processo de escrita é o que ocorre durante uma falta de escrita, ou seja quando a informação a ser escrita não está na cache. A menira mais simples de tratar é fazer a atualização da memória principal e carregar o bloco acessado na memória cache, já devidamente atualizado.

Apesar de tratar as escritas de maneira bastante simples o procedimento descrito não favorece a performance. Neste esquema, qualquer escrita faz com que seja escrita também na

memória principal . Tais escritas gastam uma quantidade de tempo considerável, e podem retardar muito o processamento de um programa.

Uma solução para este problema consiste no uso do *buffer de escrita*. Tal buffer armazena o dado enquanto este aguarda para ser escrito na memória. Apóis escrever o dado na cache e no buffer de escrita, o processador pode continuar a execução das instruções. Quando a escrita na memória principal terminar, a entrada correspondente no buffer de escrita é liberada. Se o buffer de escrita estiver cheio quando o processador chegar a uma instrução de escrita, é necessário que ele pare, até que haja uma posição disponível no buffer. Fica claro que, se a velocidade à qual a memória pode completar as escritas for menor que a taxa à qual o processador está gerando as escritas, nenhum buffer, por maior que seja, vai resolver o problema, pois as escritas estão sendo geradas mais rapidamente do que o sistema de memória de aceitá-las.

A alternativa ao sistema write-through *write-back*. Neste esquema, quando ocorre uma escrita, o novo valor é escrito apenas no bloco da cache. Tal bloco só será escrito na memória principal quando ele precisar ser substituído na cache. O esquema write-back pode melhorar muito a performance, entretanto ele é muito mais difícil de implementar.

Numa cache write-back precisamos escrever o bloco de volta na memória principal se algum dos blocos na cache tiver sido modificado - um bit de modificação deve ser implementado em cada quadro da cache para sinalizar esta ocorrência.

Tanto a técnica write-back quanto a write-through têm suas vantagens. As principais vantagens da write-back são as seguintes:

- os dados podem ser escritos individualmente pelo processador na velocidade da cache, em vez de fazê-lo na velocidade da memória principal;
- escritas múltiplas dentro de um bloco podem ser feitas em uma única operação de escrita quando todo o bloco é atualizado na memória principal.

A técnica de write-through, por sua vez, oferece as seguines vantagens:

- as faltas são mais simples e baratas de tratar;
- é mas simples de implementar do que a técnica de write-back, apesar de precisar de um buffer de escrita em sistemas de alta performance.

Como a diferença de velocidade entre a memória principal (com tecnologia SDRAM) e a memória cache aumenta cada vez mais, a conseqüência é a opção atual na implementação da política de escrita que recaí sobre a escrita write-back tendo em vista a minimização dos acessos a memória principal.

Referências Bibliográficas

[1] MONTEIRO, M. A. *Introdução a Organização de Computadores*, 3 ed. Ed: LTC, Rio de Janeiro, 1996.

[2] PATTERSON, D. A., AND HENNESSY, J. L. *Organização e projeto de computadores: a interface hardware/software*, 2 ed. Ed: LTC, Rio de Janeiro, 2000.

[3] STALLINGS, W. *Computer organization and architecture : designing for performance*, 5 ed. Ed: Prentice Hall, New Jersey, 1999.

[4] TANNENBAUM, A. S. *Organização Estruturada de Computadores*, 3 ed. Ed: Prentice Hall, Rio de Janeiro, 1992.

[5] TORRES, G. *Hardware Curso Completo*, 2 ed. Ed: Axcel, Rio de Janeiro, 1998.