

Capítulo 4

- [Vista Geral do Sistema de Memória de um Computador](#)
 - [Características dos Sistemas de Memória](#)
 - [A Hierarquia de Memória](#)
- [Memória Principal de Semi-condutores](#)
 - [Tipos de Memória de Semi-condutores de Acesso-Aleatório](#)
 - [Organização](#)
 - [Lógica do Integrado](#)
 - [Empacotamento de Integrados](#)
 - [Organização dos Módulos](#)
 - [Correcção de Erros](#)
- [Memória Oculta](#)
 - [Princípios](#)
 - [Elementos do projecto de *cache*](#)
 - [Tamanho da *cache*](#)
 - [Função de Correspondência](#)
 - [Correspondência Directa](#)
 - [Correspondência associativa](#)
 - [Correspondência em Jogos Associativos](#)
 - [Algoritmo de Substituição](#)
 - [Política de Escrita](#)
 - [Tamanho do bloco](#)
 - [Número de *caches*](#)
 - [*Caches* Simples versus Dois-Níveis](#)
 - [Unificada versus Repartida](#)
 - [Organização da *Cache* do Pentium](#)
 - [Consistência da *cache* de Dados](#)
 - [Controlo da *cache*](#)
 - [Organização da *cache* do PowerPC](#)
- [Organização DRAM avançada](#)
 - [DRAM melhorada](#)
 - [*Cache* DRAM](#)
 - [DRAM Síncrona](#)
 - [DRAM Rambus](#)
 - [RamLink](#)

Memória Interna

Apesar de parecer simples como conceito, a memória de um computador exhibe, talvez, a mais vasta gama de: tipos, tecnologia, organização, rendimento e custos, de entre todas as especificidades de um sistema de computação. Nenhuma tecnologia pode ser considerada como óptima em termos da satisfação dos requisitos de memória de um sistema de computação. Como consequência, o sistema de computação típico está equipada com uma hierarquia de sub-sistemas de memória, alguns internos ao sistema (directamente acessíveis ao processador) e outros externos (acessíveis ao processador através de um módulo de E/S).

Este capítulo põe a ênfase nos elementos da memória interna, enquanto que o capítulo 5 é dedicado à memória externa. Para começar, a primeira secção deste capítulo examina as características principais das memórias dos computadores. A seguir, olhamos para os sub-sistemas de semi-condutores da memória principal, o que inclui as memórias ROM, DRAM e SRAM. Na continuação, ficamos em posição de poder regressar à memória DRAM e olhar para arquitecturas de memória DRAM mais avançadas.

Vista Geral do Sistema de Memória de um Computador

Características dos Sistemas de Memória

O complexo tema da memória de um computador é mais facilmente manuseável se classificarmos os sistemas de memória segundo as suas características principais. As mais importantes das quais são listadas na tabela 4.1.

Começamos pelo mais visível dos aspectos da memória: a sua *localização*. Tal como o título deste capítulo e do próximo sugerem, há memória interna e memória externa a um computador. A memória interna é muitas vezes equiparada à memória principal. Mas há outras formas de memória interna. O processador necessita da sua própria memória local, sob a forma de registos (ver Figura 2.3). Além de que, como veremos, a unidade de controlo do processador que faz parte do processador pode também necessitar da sua própria memória interna. Adiaremos a discussão destes dois últimos tipos de memória interna para capítulos mais à frente. A memória externa consiste de dispositivos de armazenamento de memória, tais como discos e leitores de banda magnética, que estão acessíveis ao processador através de controladores de E/S.

Tabela 4.1: Características básicas dos sistemas de memória de computadores.

| | |
|-------------------------|---------------------------------|
| Location | Performance |
| Processor | Access time |
| Internal (main) | Cycle time |
| External (secondary) | Transfer rate |
| Capacity | Physical Type |
| Word size | Semiconductor |
| Number of words | Magnetic |
| Unit of Transfer | Optical |
| Word | Magneto-Optical |
| Block | Physical Characteristics |
| Access Method | Volatile/nonvolatile |
| Sequential | Erasable/nonerasable |
| Direct | Organization |
| Random | |
| Associative | |

Uma característica óbvia da memória é a sua *capacidade*. Esta, para a memória interna, é tipicamente expressa em termos de octectos (1 octeto = 8 bits) ou palavras. Tamanhos

de palavra habituais são 8, 16 e 32 bits. A capacidade de memória externa é tipicamente expressa em termos de octetos.

Um conceito relacionado é a *unidade de transferência*. Para a memória interna a unidade de transferência iguala o número de linhas de dados que entram e saem dos módulos de memória. Isto é muitas vezes igual ao tamanho da palavra, mas pode não ser assim. Para clarificar este ponto, consideremos três conceitos relacionados com a memória interna:

- *Palavra*: A unidade natural de organização da memória. O tamanho da palavra é tipicamente igual ao número de bits usados para representar um número ou ao tamanho da instrução. Infelizmente, há muitas excepções. Por exemplo, o CRAY-1 tem um tamanho de palavra de 64-bits mas usa uma representação de 24-bits para o inteiro. O VAX tem uma formidável variedade de tamanhos de instrução, expressos como múltiplos de um octeto e um tamanho de palavra de 32 bits.
- *Unidades de Endereçamento*: Em muitos sistemas, a unidade de endereçamento é a palavra. Contudo, alguns sistemas permitem endereçamento ao nível do octeto. Em qualquer dos casos, a relação entre o tamanho A de um endereço e o número de unidades de endereçamento é $2^A = N$
- *Unidade de Transferência*: Para a memória principal, este é o número de bits lidos ou escritos na memória num determinado instante. A unidade de transferência não necessita de ser igual à palavra ou à unidade de endereçamento. Para a memória externa, os dados são muitas vezes transferidos em unidades muito maiores do que a palavra e estas são referidas como blocos.

Uma das mais finas distinções entre tipos de memória é o *método de acesso* das unidades de dados. Podem identificar-se quatro tipos:

- *Acesso Sequencial*: A memória é organizada em unidades de dados, chamados registos. Os acessos devem ser feitos numa sequência linear específica. Informação armazenada de endereçamento é usada para separar os registos e assistir no processo de extracção. É usado um mecanismo de leitura/escrita partilhado que deve ser movido da posição corrente para a posição desejada, passando e rejeitando cada registo intermédio. Assim, o tempo de acesso a um registo arbitrário é altamente variável. As unidades de banda magnética, discutidas no capítulo 5, são de acesso sequencial.
- *Acesso Directo*: Tal como com o acesso sequencial, o acesso directo envolve um mecanismo de leitura/escrita partilhado. Contudo, os blocos individuais ou os registos têm um endereço único baseado na posição física. O acesso é conseguido através de um acesso directo para chegar a uma certa vizinhança, mais uma pesquisa sequencial, contagem ou espera, até chegar à posição final. De novo, o tempo de acesso é variável. As unidades de disco, discutidas no capítulo 5, são de acesso directo.
- *Acesso Aleatório*: Cada posição endereçável na memória tem um mecanismo de endereçamento directo por via física. O tempo de acesso a uma determinada posição é constante e independente da sequência de acessos anteriores. Assim, qualquer posição pode ser escolhida ao acaso e o endereçamento e acesso feitos directamente. Os sistemas de memória principal tem acesso aleatório.

- *Acesso Associativo:* Esta é um tipo de memória de acesso aleatório que dá a hipótese de comparar a posição dos bits pretendido, dentro de uma palavra, para um padrão específico e fazer isto para todas as palavras simultaneamente. Assim, uma palavra é extraída com base numa porção do seu conteúdo em vez do seu endereço. Tal como com o acesso aleatório convencional, cada posição tem o seu próprio mecanismo de endereçamento e o tempo de extracção é constante, independente da posição ou da sequência de acessos anteriores. As memórias *cache*, discutidas na secção 4.3, podem empregar acessos associativos.

Sob o ponto de visto do utilizador, as duas mais importantes características da memória são a capacidade e o *rendimento*. Três parâmetros de rendimento são usados:

- *Tempo de Acesso:* Para a memória de acesso aleatório, é o tempo que demora a concluir uma operação de leitura ou de escrita, isto é, o tempo que decorre desde o instante em que um endereço é apresentado à memória, até ao instante em que os dados são armazenados ou tornados disponíveis para uso. Para as memórias de acesso não aleatório, o tempo de acesso é o tempo que demora a levar o mecanismo de leitura/escrita até à posição desejada.
- *Tempo do Ciclo de Memória:* Este conceito é primariamente aplicável à memória de acesso aleatório e consiste no tempo de acesso mais um tempo adicional, necessário antes de poder começar um segundo acesso. Este tempo adicional pode ser necessários para que se extingam as transições nos sinais das linhas, ou para regenerar os dados se a leitura for destrutiva.
- *Taxa de Transferência:* Esta é a velocidade a que os dados podem ser transferidos para a ou da unidade de memória. Para a memória de acesso-aleatório, esta taxa é igual a $1/(\text{Tempo de Ciclo})$. Para a memória de acesso-não-aleatório pode aplicar-se a seguinte relação: $T_n = TA + N/R$, onde T_n = Tempo médio de leitura ou escrita de N bits, TA = Tempo de acesso médio, N = Número de bits e R = Taxa de transferência em bits por segundo (bps)

Uma variedade de *tipos físicos* de memória tem vindo a ser empregado. Hoje, os dois mais comuns são as memórias de semi-condutores que usam tecnologias LSI ou VLSI e as memórias de superfície magnética, usadas em discos e bandas.

Várias *características físicas* de armazenamento são importantes. Na memória volátil, a informação extingue-se naturalmente ou perde-se quando a fonte de alimentação é desligada. Na memória não volátil, a informação uma vez registada permanece sem deterioração até ser deliberadamente modificada; não é necessária fonte de alimentação para reter a informação. As memórias de superfície magnética são não voláteis. As memória de semi-condutores podem ser tanto voláteis como não voláteis. As memória não apagáveis não podem ser alteradas, excepto através da destruição da unidade de armazenamento. As memória de semi-condutores deste tipo são conhecidas por *leitura apenas* (ROM). Em termos práticos, é desejável que as memórias não apagáveis sejam, também, não voláteis.

Para as memórias de acesso-aleatório, a organização é um ponto chave no projecto. Por *organização* queremos significar o arranjo físico dos bits para formar palavras. O arranjo óbvio não é sempre usado, como será explicado aqui.

A Hierarquia de Memória

As restrições no projecto de memória de um computador podem ser sumariadas em três questões: Quanta? Quão rápida? A que preço?

A questão da quanta está, de alguma forma, sempre em aberto. Se a capacidade existir, com toda a certeza haverá aplicações desenvolvidas para usá-la. A questão de quão rápida é, em certo sentido, mais fácil de responder. Para atingir um elevado rendimento, a memória deve estar ao nível do processador. Isto é, à medida que o processador executa instruções, gostaríamos de não ter de esperar por instruções ou por operandos. A questão final deve, também, ser apreciada. Para sistemas de interesse prático, o custo da memória deve ser razoável em relação aos outros componentes.

Tal como poderia ser esperado, há um compromisso entre as três características da memória, nomeadamente, custo, capacidade e tempo de acesso. Em cada instante, uma variedade de tecnologias são usadas para implementar os sistemas de memória. Neste espectro de tecnologias, podem considerar-se as seguintes relações:

- Tempos de acesso mais curtos, maior o custo por bit.
- Maior capacidade, mais pequeno o custo por bit.
- Maior capacidade, maior o tempo de acesso.

O dilema que se apresenta ao projectista é claro. O projectista gostaria de usar as tecnologias de memória que disponibilizam uma elevada capacidade de memória, tanto porque a capacidade é necessária como pelo baixo custo por bit. Contudo, para atingir os requisitos de rendimento, o projectista necessita de usar memórias dispendiosas, de relativamente baixa capacidade e rápidos tempos de acesso.

A solução deste dilema não está dependente de um simples componente de memória ou da tecnologia, mas no emprego de uma *hierarquia de memória*. Uma hierarquia típica é ilustrada na fig 4.1a. À medida que progredimos no sentido descendente na hierarquia, assistimos ao seguinte:

- a) Redução do custo/bit
- b) Aumento de capacidade
- c) Aumento do tempo de acesso
- d) Redução da frequência dos acessos à memória pelo processador

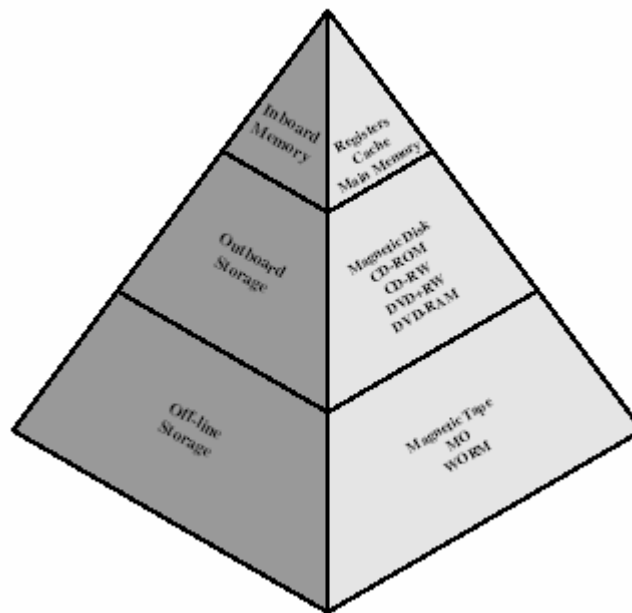


Figura 4.1: Hierarquia de memória.

Assim, memórias, mais pequenas, mais dispendiosas e mais rápidas são complementadas por memórias, maiores, mais baratas e mais lentas. A chave para o sucesso desta organização está no último item, redução da frequência dos acessos. Examinaremos este conceito em detalhe quando discutirmos a *cache*, mais à frente neste capítulo, e a memória virtual, no capítulo 7, mas avançamos, aqui, com uma breve explicação .

Se a memória poder ser organizada de acordo com os itens (a) a (c) acima e se os dados e as instruções poderem ser distribuídos por toda a memória em consonância com (d), então parece ser claro, intuitivamente, que este esquema levará à redução do custo total ao mesmo tempo que se mantém um determinado nível de rendimento. Apresentamos um exemplo simples para ilustrar este ponto.

Suponha que o processador tem acesso a dois níveis de memória. O nível 1 contém 1000 palavras e tem tempo de acesso de $R = s$. O nível 2 contém 100 000 palavras e tem um tempo de acesso de $1\mu s$.

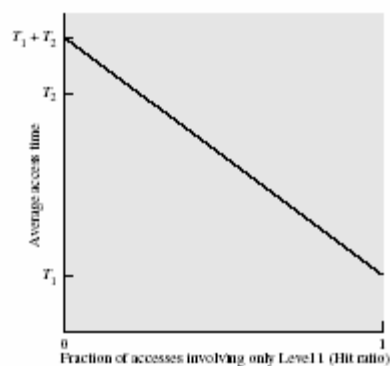


Figura 4.2: Rendimento de uma memória simples de dois níveis.

Assuma que, se uma palavra a que se pretende ter acesso está no nível 1, então o processador tem-lhe acesso directo. Se está no nível 2, então a palavra é primeiramente transferida para o nível 1, podendo o processador ter-lhe acesso por aí. Por simplicidade, vamos ignorar o tempo necessário para o processador determinar se a palavra está no nível 1 ou no nível 2. A figura 4.2 mostra o tempo de acesso médio total como uma função da percentagem do tempo que a desejada palavra está já no nível 1. Como pode ser visto, para percentagens elevadas de acessos no nível 1, o tempo de acesso médio total está muito mais perto do nível 1 do que do nível 2.

Este exemplo ilustra que, em princípio, a estratégia funciona. Funciona, na prática, se as condições de (a) a (b) forem aplicáveis. As figuras 4.3 e 4.4. mostram características típicas de sistemas de memória actuais alternativos. A figura 4.3 mostra que empregando uma variedade de tecnologias, existe um espectro de sistemas de memória que satisfazem (b) e (c) e a figura 4.4 confirma que a condição (a) é satisfeita. Felizmente, a condição (d) é, também, válida em geral.

A base para a validade da condição (d) é um princípio conhecido como *localidade das referências*[#!denn68!#]. Durante o curso da execução de um programa, as referências à memória com origem no processador, tanto para as instruções como para os dados, tendem a aglomerar-se. Os programas contém, tipicamente, um certo número de ciclos iterativos e sub-rotinas. Uma vez entrado num ciclo ou sub-rotina há repetidas referências a um pequeno conjunto de instruções. De forma similar, as operações em tabelas e listas envolvem acessos a agrupamentos de conjunto de palavras de dados. Em períodos de tempo longos os agrupamentos em uso mudam, mas em períodos de tempo curtos o processador está predominantemente a trabalhar com agrupamentos fixos de referências à memória.

Figura 4.3: Comparação do armazenamento.

Em concordância, é possível organizar os dados ao longo da hierarquia de tal forma que a percentagem de acessos sucessivos a cada nível mais abaixo é substancialmente inferior aos níveis mais acima. Considere o exemplo de dois níveis já apresentado. Deixe a memória de nível 2 conter todo o programa e dados. Os agrupamentos correntes podem ser colocados temporariamente no nível 1.

Figura 4.4: Previsão do custo das tecnologias de armazenamento secundário [weiz9].

De tempos a tempos, um dos agrupamentos no nível 1 deverá ter de ser posto, de novo, na memória de nível 2 para deixar lugar para um novo agrupamento vindo do nível 1. Em termo médios, contudo, a maior parte das referências serão para instruções e dados contidos no nível 1.

Este princípio pode ser aplicado ao longo de mais do que dois níveis de memória. Considere a hierarquia mostrada na figura 4.1a. O tipo de memória, mais rápida, mais pequena e mais cara consiste nos registos internos do processador. Tipicamente, um processador contém uma escassa dezena desses registos, embora algumas máquinas contenham centenas de registos. Saltando por cima de dois níveis, a memória principal, também referenciada como memória real, é o sistema de memória principal do computador. Cada posição na memória principal tem um único endereço e a maior parte das instruções da máquina referem-se a um ou a mais endereços na memória principal. A memória principal é habitualmente estendida com uma pequena *cache* de velocidade superior. A *cache* não é habitualmente visível ao programador nem ao processador. É um dispositivo que serve de plataforma intermédia para o movimento de dados entre a memória principal e os registos do processador para melhorar o rendimento.

As três formas de memória que acabamos de descrever são, tipicamente, voláteis e empregam tecnologia de semi-condutores. O uso de três níveis explora a variedade de tipos de memória de semi-condutores que diferem em velocidade e custo. Os dados são armazenados mais permanentemente em dispositivos de memória externa de grande capacidade de armazenamento, dos quais os mais comuns são os discos magnéticos e bandas magnéticas. A memória externa, não volátil, é também referida como secundária ou auxiliar. Esta é usada para guardar programas e ficheiros de dados e são habitualmente visíveis ao programador apenas como ficheiros e registos, por oposição aos octetos e às palavras individuais. Os discos são também usados para fornecer uma extensão à memória principal conhecida como armazenagem virtual ou memória virtual que é discutida no capítulo 7.

Outras formas de memória podem ser incluídas na hierarquia. Por exemplo, computadores IBM de grande porte incluem uma forma de memória interna conhecida como Armazenamento Expandido. Esta usa uma tecnologia de semi-condutores que é mais lenta e menos cara do que a da memória principal. Falando em termos estritos, esta memória não se enquadra na hierarquia mas é um braço lateral: os dados podem ser movidos entre a memória principal e o armazenamento expandido mas não entre o armazenamento expandido e a memória externa. Outras formas de memória secundária incluem discos ópticos e dispositivos de memória em bolha. Finalmente, níveis adicionais podem ser adicionados, efectivamente, à hierarquia por software. Uma porção de memória principal pode ser usada como tampão para conter, temporariamente, dados que deverão ser lidos para disco. Esta técnica alguma vezes referida como *cache* de discos^{4.1}, melhora o rendimento de duas formas;

- As escrita para disco são agrupadas. Em vez de muitas pequenas transferências de dados, temos apenas algumas grandes transferências. Isto melhora o rendimento do disco e minimiza o envolvimento do processador.

- Alguns dos dados destinados a escrita podem ser referenciados por um programa antes do próximo despejo para disco. Neste caso, os dados são extraídos rapidamente da *cache* em software em vez de lentamente do discos.

A figura 4.1b mostra uma hierarquia de memória actual que inclui uma *cache* de disco, um disco óptico e um tipo adicional de memória secundária.

O apêndice 4A examina as implicações no rendimento de uma estrutura de memória multi-nível.

Memória Principal de Semi-condutores

Nos primeiros computadores, a forma mais habitual de armazenagem de acesso aleatória da memória principal dos computadores empregava um vector de anéis ferromagnéticos com a forma de rosca (doughnut) referida como núcleos, *cores*. Assim, a memória principal era muita vezes designada por *core*, um termo que ainda hoje persiste. O advento, e as vantagens, da micro-electrónica há muito que superaram os núcleos de memória magnética. Hoje, o uso de integrados semi-condutores na memória principal é quase universal. Os aspectos chave desta tecnologia são examinados nesta secção.

Tipos de Memória de Semi-condutores de Acesso-Aleatório

Todos os tipos de memória que iremos examinar nesta secção são de acesso-aleatório. Isto é, o acesso a palavras individualizadas da memória é feito directamente através de lógica de endereçamento por via física.

A tabela 4.2 lista os tipos mais importantes de memórias de semi-condutores. O mais habitual é referido por *memória de acesso-aleatório* (RAM). Esta é, obviamente, uma má utilização do termo uma vez que todos os tipos listados na tabela são de acesso aleatório. Uma característica distintiva da RAM é que é possível, tanto ler dados da memória como fácil e rapidamente escrever novos dados na memória. Tanto a leitura como a escrita são realizáveis com base em sinais eléctricos.

A outra característica distintiva da RAM é ser volátil. A RAM tem de ser fornecida com uma alimentação constante. Se a alimentação for interrompida, então os dados perdem-se. Assim, a RAM só pode ser usada para armazenamento temporário.

A tecnologia RAM está dividida em duas tecnologias: estática e dinâmica. A RAM *dinâmica* é feita com células que armazenam os dados como carga em capacidades. A presença ou ausência de carga numa capacidade é interpretada como os binários 1 ou 0. Como as capacidades tem a tendência natural para se descarregarem, as RAM dinâmicas requerem um refrescamento periódico da carga para manter a armazenagem dos dados. Numa RAM *estática* valores binários são armazenados usando configurações tradicionais de portas-lógicas de flip-flops (ver apêndice A para uma descrição dos flip-flops). A RAM estática mantém os seus dados enquanto estiver ligada à alimentação.

Tabela 4.2: Tipos de Memória de Semi-condutores.

| Memory Type | Category | Erasure | Write Mechanism | Volatility |
|-------------------------------------|--------------------|---------------------------|-----------------|-------------|
| Random-access memory (RAM) | Read-write memory | Electrically, byte-level | Electrically | Volatile |
| Read-only memory (ROM) | Read-only memory | Not possible | Masks | Nonvolatile |
| Programmable ROM (PROM) | | | Electrically | |
| Erasable PROM (EPROM) | Read-mostly memory | UV light, chip-level | | |
| Electrically Erasable PROM (EEPROM) | | Electrically, byte-level | | |
| Flash memory | | Electrically, block-level | | |

As RAM tanto estáticas como dinâmicas são voláteis. Uma célula de memória dinâmica é mais simples e por isso mais pequena do que uma célula de memória estática. Assim, uma memória RAM dinâmica é mais densa (células mais pequenas = mais células por unidade de superfície) e menos dispendiosas do que a memória RAM estática correspondente.

Por outro lado, uma RAM dinâmica requer o suporte de circuito de refrescamento. Para grandes quantidades de memória, o custo fixo do circuito de refrescamento é mais do que compensador face ao custo variável mais pequeno das células RAM dinâmicas. Assim, as memórias RAM dinâmicas tendem a ser favorecidas quando há requisitos de grandes quantidades de memória. Um ponto final é que a RAM estática é em geral significativamente mais rápida que as RAM dinâmicas.

Em estreito contraste com a RAM está a *memória só de leitura*. Tal como o nome sugere, a ROM contém um padrão permanente de dados que não pode ser alterado. Enquanto é possível ler uma ROM, não é possível escrever lá novos dados. Um aplicação importante das ROM é a micro-programação, discutida na parte IV. Outras aplicações potenciais incluem:

- Bibliotecas de sub-rotinas para funções evocadas frequentemente
- Programas de sistema
- Tabelas de funções

Para requisitos de tamanho modestos, a vantagem da ROM é que o dado ou programa está permanentemente em memória e não necessita nunca de ser carregado de um dispositivo de memória secundária.

Uma ROM é feita como qualquer outro circuito integrado, com os dados rigidamente fixados no integrado como resultado do processo de fabrico. Isto apresenta dois problemas:

- O passo de inserção dos dados inclui um custo fixo relativamente elevado, quer sejam fabricados uma ou milhares de cópias de uma ROM particular.

- Não há lugar para erros. Se um bit estiver errado, o lote completo de ROMs tem de ser descartado.

Quando apenas é necessário um pequeno número de ROMs com um conteúdo de memória particular, uma alternativa menos dispendiosa é a ROM *programável* (PROM). Tal com a ROM, a PROM não é volátil e só pode ser escrita uma vez. Para a PROM, o processo de gravação é feito electricamente e pode ser realizado pelo fabricante ou pelo consumidor numa altura posterior ao processo original de fabricação. É necessário equipamento específico para a gravação ou processo de "programação". As PROMs oferecem flexibilidade e conveniência. A ROM permanece atractiva para a produção de grandes quantidades.

Uma outra variante das memórias só de leitura são as memórias para leitura a maior parte das vezes, úteis em aplicações em que as operações de leitura são substancialmente mais frequentes do que as operações de escrita mas em que é necessário armazenamento não volátil. Há três formas habituais de memória de leitura a maior parte das vezes: EPROM, EEPROM e memória relâmpago (flash).

A *memória apagável programável só de leitura* (EPROM) é de leitura e escrita eléctrica, tal como as PROM. Contudo, antes de uma operação de escrita, toda as células de armazenamento devem ser conduzidas para o mesmo estado inicial por exposição do integrado a radiações ultravioleta. Este processo de apagamento pode ser realizado repetidamente; cada apagamento pode levar tanto como 20 minutos para ser concluído. Assim, a EPROM pode ser alterada múltiplas vezes e, tal com a ROM e a PROM, mantém os seus dados por tempo virtualmente infinito. Para quantidades comparáveis de armazenamento, a EPROM é mais cara do que a PROM, mas tem a vantagem da capacidade para múltiplas actualizações.

Uma forma mais atractiva de memória de leitura a maior parte das vezes é a *memória electricamente apagável só de leitura programável* (EEPROM). Esta é uma memória de leitura a maior parte das vezes que pode ser gravada em qualquer momento sem apagar o conteúdo anterior; só o octeto ou os octetos endereçados são actualizados. A operação de escrita leva um tempo consideravelmente maior do que a operação de leitura, na ordem das várias centenas de micro-segundos por octeto. A EEPROM combina as vantagens da não volatilidade com a flexibilidade da actualização no lugar, usando o barramento de controlo convencional, endereços e linhas de dados. A EEPROM é mais cara do que a EPROM e também menos densa, comportando menos bits por integrado.

A mais recente forma de memória de semi-condutores é a memória relâmpago (*flash*) (assim chamada por causa da velocidade com que pode ser reprogramada). Introduzida em primeiro mão nos meados dos anos 80, a memória relâmpago está a meio caminho entre a EPROM e a EEPROM tanto em custo como em funcionalidade. Tal como a EPROM, a memória relâmpago usa uma tecnologia de apagamento eléctrico. Uma memória relâmpago completa pode ser apagada em um ou poucos segundos, o que é muito mais rápido do que a EPROM. Cumulativamente, a memória relâmpago não oferece apagamento ao nível do octeto. Tal como a EPROM, a memória relâmpago usa apenas um transistor por bit e por isso atinge a elevada densidade (quando comparada com a EEPROM) da EPROM.

Organização

O elemento básico de uma memória de semi-condutores é a célula de memória. Embora seja usada uma variedade de tecnologias electrónicas, todas as células de memória de semi-condutores partilham certas propriedades:

- Exibem dois estados estáveis (ou semi-estáveis), os quais podem ser usados para representar os valores binários 1 e 0.
- São capazes de ser escritas (pelo menos uma vez) para ajustar o estado.
- São passíveis de ser lidas para dar a conhecer o seu estado.

A figura 4.5 ilustra a operação de uma célula de memória. Muito habitualmente, a célula tem três terminais funcionais capazes de transportar sinais eléctricos.

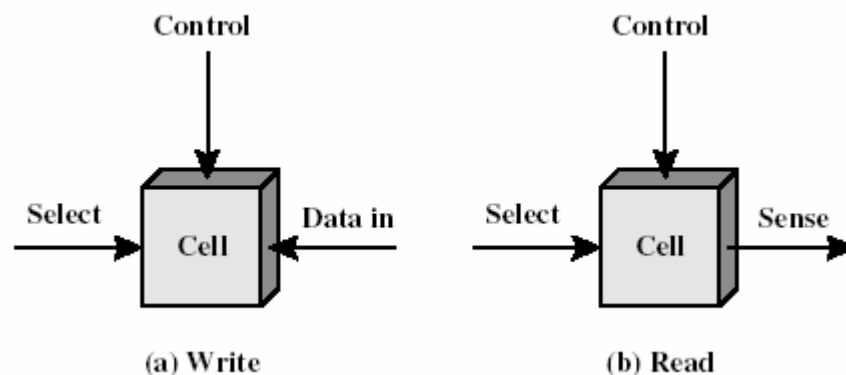


Figura 4.5: Operação de uma célula de memória.

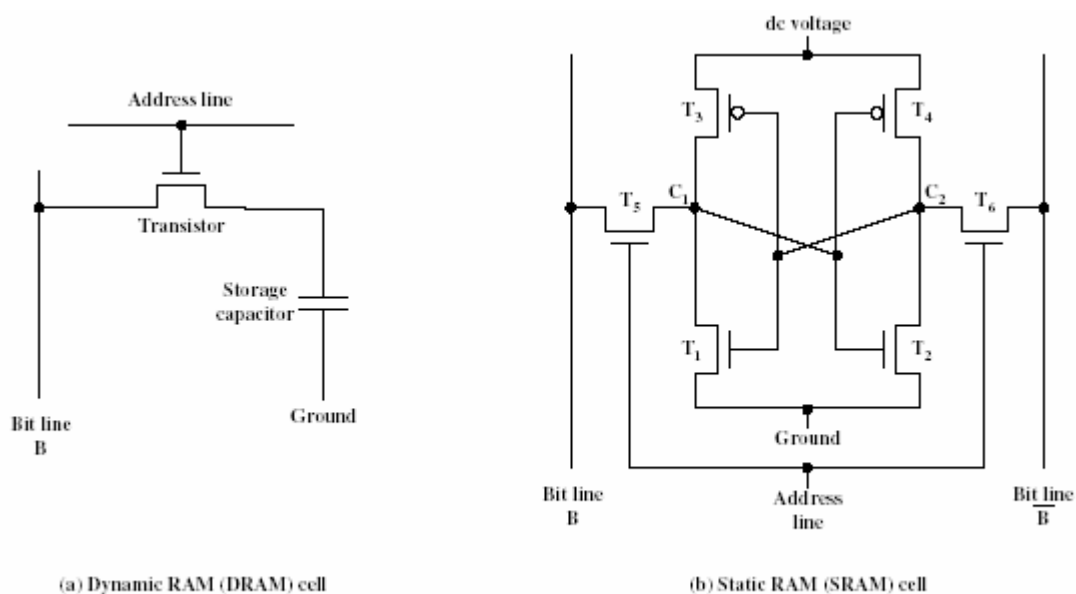


Figura 4.5 c) : Estrutura típica de célula de memória.

O terminal de selecção, tal como o nome sugere, selecciona uma célula de memória para uma operação de leitura ou de escrita. O terminal de controlo indica a leitura ou a escrita. Para escrever, o outro terminal fornece um sinal eléctrico que ajusta o estado da célula a 1 ou a 0. Para ler, o terminal é usado para saída do estado da célula. Os detalhes da organização interna, funcionamento e temporização da célula de memória dependem da tecnologia do circuito integrado específico usado e está para além dos objectivos deste livro. Para os nossos propósitos, tomaremos isso como assumindo que as células individuais podem ser seleccionadas para operações de leitura e de escrita.

Lógica do Integrado

Tal como com outros produtos de circuitos-integrados, as memórias de semi-condutores estão disponíveis em circuitos acondicionados (Figura 2.7). Cada integrado contém uma matriz de células de memória. Com a tecnologia actual, integrados de 4-Mbits são comuns, os integrados de 16-Mbits começam a ser usados.

Na hierarquia de memória como um todo, vimos que há compromissos entre velocidade, capacidade e custo. Estes compromissos também existem quando consideramos a organização das células de memória e a lógica funcional de um integrado. Para as memórias de semi-condutores, um dos pontos chave do projecto é o número de bits de dados que podem ser lidos/escritos num dado instante. Num dos extremos está uma organização em que o arranjo físico das células na matriz é igual ao arranjo lógico (tal como é visto do processador) das palavras na memória. A matriz é organizada em W palavras de B bits cada. Por exemplo, um integrado de 16-Mbits pode ser organizado em 1M palavras de 16-Mbits. No outro extremo está a organização chamada de 1-bit por integrado, na qual os dados são lidos/escritos um bit de cada vez. Ilustraremos a organização dos integrados de memória com uma DRAM; A organização da ROM é similar, embora mais simples.

A figura 4.6 mostra a organização típica de uma DRAM de 16-Mbits. Neste caso, quatro bits são lidos ou escritos em cada momento. Logicamente, a matriz de memória é organizada como quatro matrizes quadradas de 2048 por 2048 elementos. Vários arranjos físicos são possíveis. Em qualquer dos casos, os elementos da matriz são ligados tanto por linhas horizontais (fiadas) como por linhas verticais (colunas). Cada linha horizontal liga-se ao terminal Select de cada célula na respectiva fiada; cada linha vertical liga-se ao terminal de Data-In/Sense de cada célula na respectiva coluna.

As linhas de endereços disponibilizam o endereço da palavra a ser seleccionada. Um total de ¹⁰ linhas são necessárias. No nosso exemplo, são necessárias 11 linhas para seleccionar uma das 2048 fiadas.

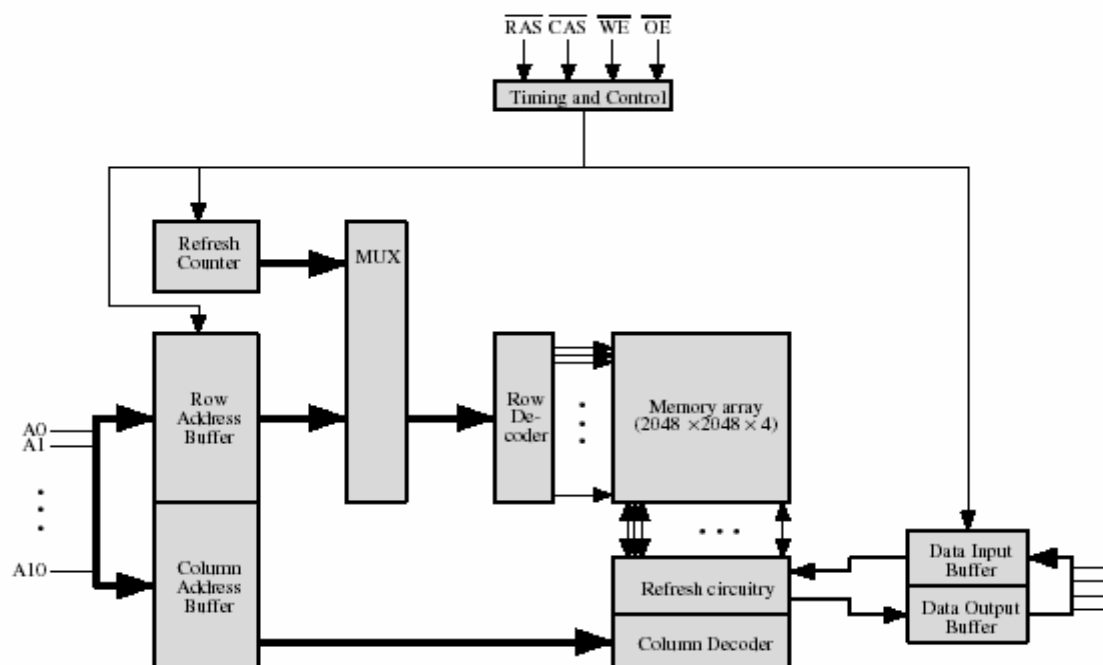


Figura 4.6: DRAM de 16 mega-bits típica (4Mx4).

Estas 11 linhas vão alimentar um descodificador de fiadas que possui 11 linhas de entrada e 2048 linhas para saída. A lógica do descodificador activa simplesmente uma das 2048 saídas dependendo do padrão de bits nas $(\log_2 W)$ linhas de entrada.

11 linhas de endereços adicionais seleccionam uma das 2048 colunas de quatro bits por coluna. Quatro linhas de dados são usadas para a entrada e saída de quatro bits de e para o tampão de dados. Na entrada (escrita) o condutor do bit em cada linha é levado a 1 ou 0 de acordo com o valor da linha de dados correspondente. Na saída (leitura) o valor de cada bit é passado através de um sensor amplificador e apresentado as linhas de dados.

A linha da fiada selecciona a fiada de células usada na leitura ou escrita.

Uma vez que só são lidos/escritos quatro bits nesta DRAM, tem de haver múltiplas DRAMs ligadas ao controlador da memória de forma a ler/escrever uma palavra ou um dado no barramento.

Notar que há apenas 11 linhas de endereço (A0-A10), metade do número que seria esperado para uma matriz de 2048 x 2048. Isto faz-se para poupar no número de pinos. As 22 linhas de endereços necessárias são passadas através de lógica escolhida externa ao integrado e multiplexada em 11 linhas de endereços. Primeiro, 11 linhas são passadas ao integrado para definir os endereços das fiadas da matriz e então os outros 11 sinais de endereços são apresentados como endereços de colunas. Estes sinais são acompanhados pela escolha dos sinais de endereços das fiadas (RAS) e endereços das colunas (CAS) para fornecer temporização ao integrado.

Como efeito lateral, o endereçamento multiplexado mais o uso de matrizes quadradas resulta na quadriplicação do tamanho da memória com cada nova geração de integrados de memória. Um pino mais dedicado ao endereçamento duplica o número de fiadas e de colunas e consequentemente o tamanho da memória no integrado cresce por um factor de 4. Assim, temos vindo a seguir as seguintes gerações, a uma taxa, grosso modo, de um cada três anos: 1K, 4K, 16K, 64K, 256K, 1M, 4M, 16M.

A figura 4.6 também mostra a inclusão de um circuito de refrescamento. Todas as DRAM requerem uma operação de refrescamento. Uma técnica simples de refrescamento inibe, para todos os efeitos, o integrado DRAM enquanto todas as células de dados são refrescadas. O contador de refrescamento prossegue através de todos os valores das fiadas. Por cada fiada, as linhas de saída do contador de refrescamento vão alimentar o descodificador de fiadas e activar a linha de RAS. A consequência, disto, é cada célula na fiada ser refrescada.

Empacotamento de Integrados

Tal como foi mencionado no Capítulo 2, um circuito integrado é montado num invólucro que contém pinos para conexão ao mundo exterior.

A figura 4.7a mostra o exemplo de um invólucro de uma EPROM, que é um circuito integrado de 8M-bits organizado em 1M x 8. Neste caso, a organização faz-se numa palavra por integrado. O invólucro inclui 32 pinos, que é um dos tamanhos estandardizados de invólucros de integrados. Os pinos suportam as seguintes linhas de sinal:

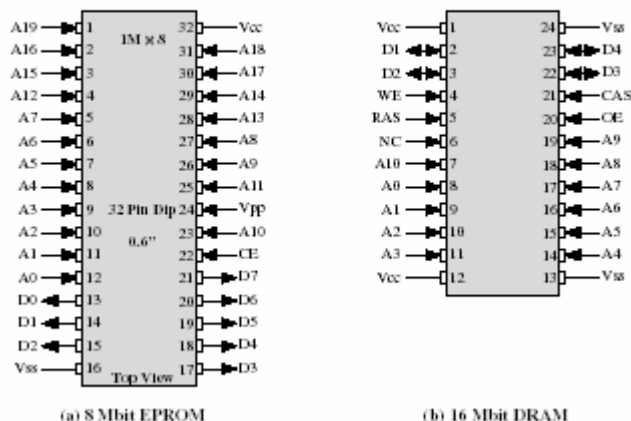


Figura 4.7: Pinos e sinais da memória de um invólucro típico.

- O endereço da palavra para o acesso corrente. Para 1M palavras, é necessário um total de 20 pinos ($2^{20} = 1\,048\,576$).
- Os dados corrente, consistindo de 8 linhas (D0-D7).
- A alimentação do integrado (V_{cc}).
- Um pino para massa (V_{ss}).
- A habilitação do integrado. Uma vez que pode haver mais do que um integrado de memória, cada um dos quais está ligado ao mesmo barramento de endereços, o pino CE é usado para indicar a existência de um endereço válido para este integrado. O pino CE é activado pela lógica associada aos bits de ordem superior do barramento de endereços (i.e. os bits de endereço acima de A19) O uso deste sinal é ilustrado em seguida.
- uma tensão programável (V_{pp}) é suprida durante a programação (operações de escrita)

Uma configuração de pinos típica de uma DRAM é mostrada na figura 4.7b para um integrado de 16M-bits organizado em 4M x 4. Apresenta variadas diferenças para com um integrado ROM. Porque a RAM pode ser alterada os pinos (OE) indicam se a operação é de escrita ou de leitura. Porque o acesso a DRAM é feito por linha e coluna e o endereço é multiplexado, só 11 pinos de endereços são necessários para especificar 4M combinações de linhas/colunas ($2^{20} = 1\,048\,576$). A função dos pinos do selector de linhas (RAS) e do selector de colunas (CAS) foi, já, discutidos atrás.

Organização dos Módulos

Quando um integrado contém apenas um bit por palavra, então na verdade necessitamos de um número de integrados pelo menos igual ao número de bits na palavra. Como exemplo, a figura 4.8 mostra de que forma um módulo de memória consistindo de 256K palavras de 8-bits pode ser organizado. Para 256K palavras, é necessário um endereço de 18-bits suprido ao módulo através de uma fonte externa (e.g., as linhas de endereço do barramento aos quais o módulo está fixado). O endereço é apresentado a 8 integrados de 256K x 1-bit, cada um dos quais oferece 1 bit para entrada/saída.

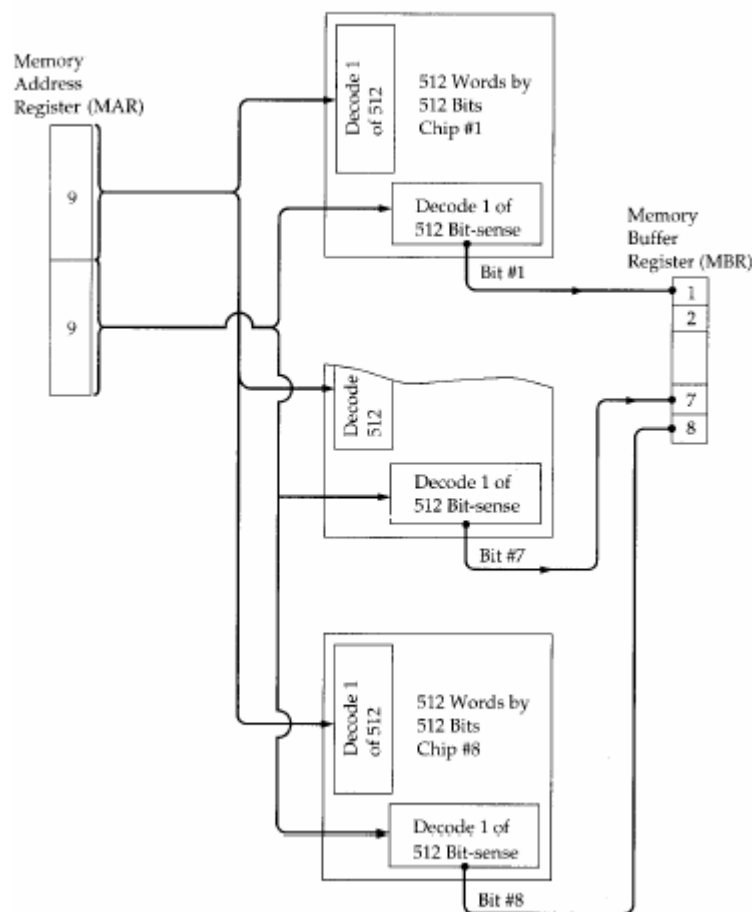


Figura 4.8: Organização de memória de 256k octetos.

Esta organização é apropriada enquanto o tamanho da memória igualar o número de bits por integrado. No caso de ser requerido uma quantidade de memória superior, é necessário uma configuração de integrados. A figura 4.9 mostra a, possível, organização de uma memória consistindo de 1M palavras, com 8 bits por palavra. Neste caso temos quatro colunas de integrados, cada coluna contém 256K palavras arranjadas como na figura 4.8. Para 1M palavras, são necessárias 20 linhas de endereços. Os 18 bits menos significativos são dirigidos para todos os 32 módulos. Os 2 bits de ordem superior entram num módulo lógico de selecção de grupos que emite o sinal de habilitação de integrado para uma das quatro colunas de módulos.

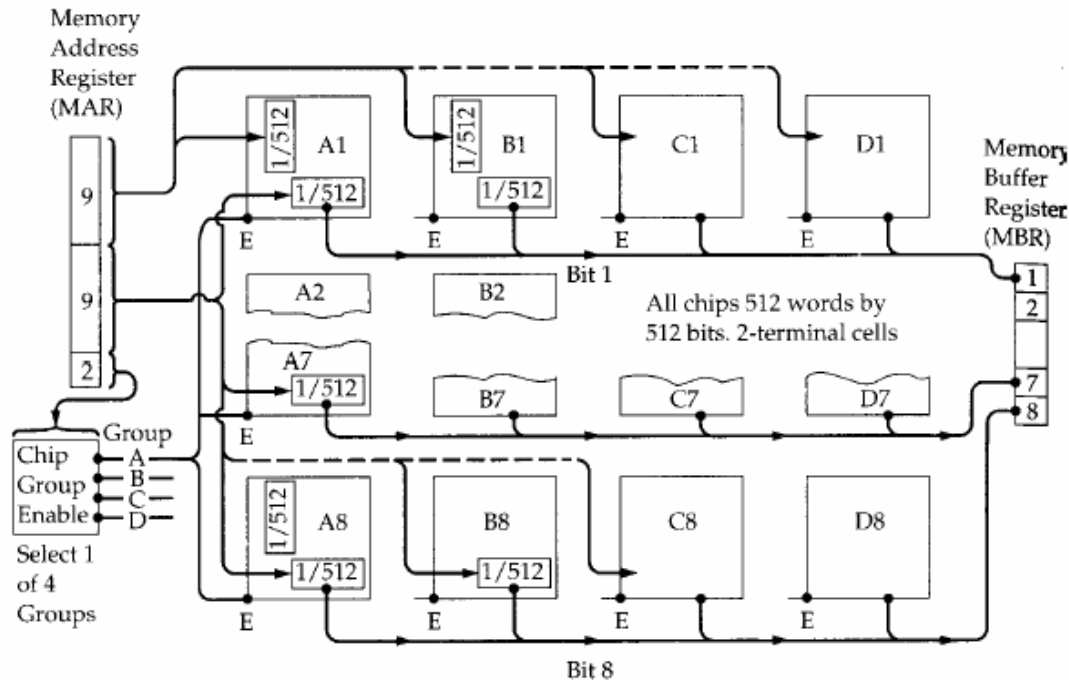


Figura 4.9: Organização de memória de 1M-byte.

Correcção de Erros

Um sistema de memória de semi-condutores está sujeito a erros. Estes podem ser categorizados como falhas físicas e erros lógicos. Uma falha física é um defeito físico permanente, tal que a célula de memória ou células afectadas não pode com segurança armazenar dados, mas fica colado a zero ou a 1, ou erráticamente alterna entre 0 e 1. Os erros físicos podem ser provocados por abusos severos do ambiente, defeitos de fabrico e de uso. Um erro lógico é um evento aleatório não destructivo que altera o conteúdo de uma ou mais células de memória, sem a danificar. Os erros lógicos podem ser provocados por problemas de alimentação de potência ou partículas alfa. Estes partículas resultam das perdas radioactivas e são lamentavelmente comuns porque a radioactividade nuclear encontra-se em pequenas quantidades em praticamente todos os materiais. Tanto os erros físicos como lógicos são claramente indesejáveis e a maior parte dos sistemas de memória principal modernos incluem lógica tanto para detectar como para corrigir erros.

A figura 4.10 ilustra em termos gerais a forma como o processo decorre. Quando os dados são lidos para a memória, um cálculo, descrito como uma função 2^{11} , é efectuado nos dados para gerar um código. Tanto o código como os dados são salvaguardados. Assim, se um palavra de M-bits for guardada e o código for de k bits, então o tamanho efectivo das palavras guardadas é k bits.

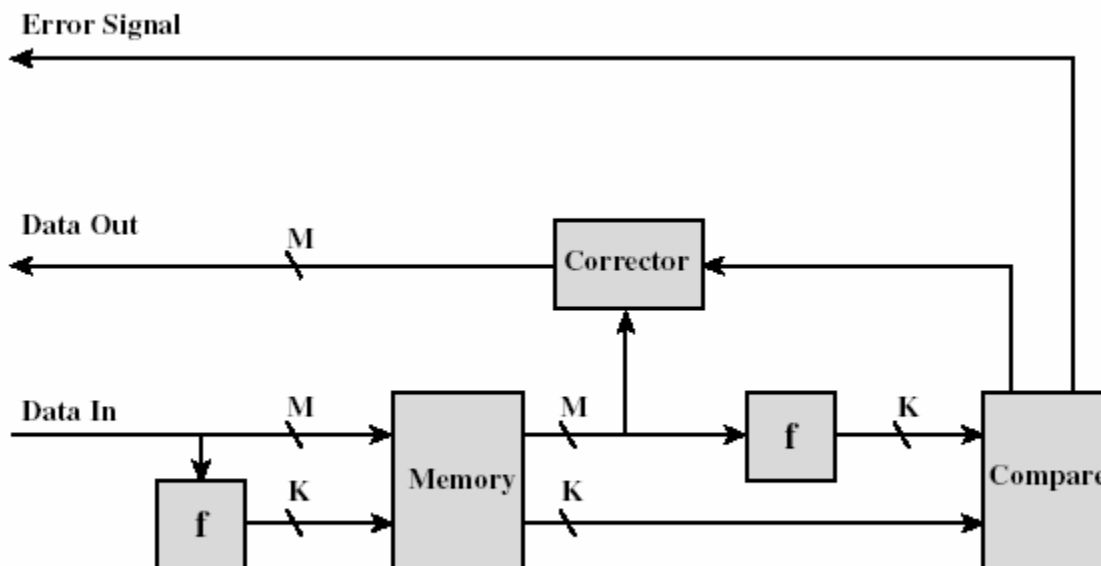


Figura 4.10: Função de código de correcção de erros.

Quando a palavra previamente guardada é extraída, o código é usado para detectar e se possível corrigir erros. Um novo conjunto de K bits de código é gerada a partir dos M bits e comparado com os bits extraídos. A comparação produz um de três resultados:

- Não foram detectados erros. Os dados extraídos seguem para fora.
- É detectado um erro e é possível corrigi-lo. Os bits de dados mais os bits de correcção vão alimentar um corrector, o qual gera um conjunto corrigido de bits que seguem para fora.
- É detectado um erro, mas não é possível corrigi-lo. É assinalada esta condição.

Os códigos que operam desta maneira são designados por *códigos de correcção de erros*. Um código é caracterizado pelo número de bits com erro numa palavra que pode detectar e corrigir.

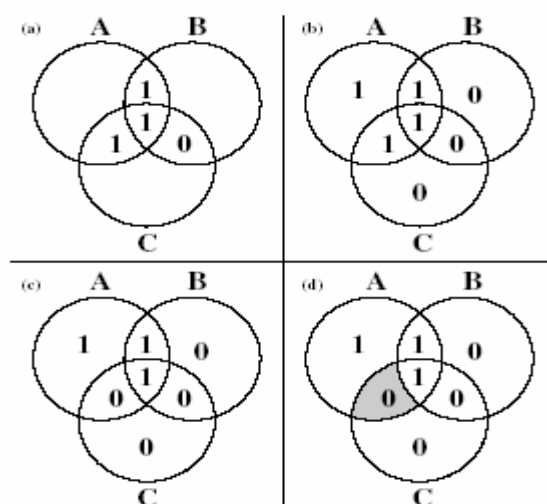


Figura 4.11: Código de correcção de erros de Hamming.

O mais simples dos códigos de correcção de erros é o código de Hamming inventado por Richard Hamming dos Laboratórios Bell. A figura 4.11 usa diagrams de Venn para ilustrar o uso deste código em palavras de 4-bits ($M=4$). A intersecção de três círculos define sete compartimentos. Os 4 bits de dados são atribuídos aos compartimentos interiores (Figura 4.11a). Os compartimentos restantes são preenchidos com o que se chama *bit de paridade*. cada bit de paridade é escolhido de forma a que o número total de 1s em cada círculo é par (Figura 4.11b). Assim, uma vez que o círculo A inclui três dados a 1, o bit de paridade nesse círculo é ajustado a 1. Agora, se um erro modificar um dos bits de dados (Figura 4.11c) é facilmente detectado. Testando os bits de paridade, são encontradas discrepâncias no círculo A e no círculo C, mas não no círculo B. Só um dos setes compartimentos está em A e em C, mas não está em B. O erro pode por isso ser corrigido através da alteração daquele bit.

Tabela 4.3: Aumento do tamanho da palavra com a correcção de erros.

| Word size | Check bits | Total size | Percent overhead |
|-----------|------------|------------|------------------|
| 8 | 4 | 12 | 50 |
| 16 | 5 | 21 | 31 |
| 32 | 6 | 38 | 19 |
| 64 | 7 | 71 | 11 |
| 128 | 8 | 136 | 6 |
| 256 | 9 | 265 | 4 |
| 512 | 10 | 522 | 2 |

| Bit Position | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
|-----------------|------|------|------|------|------|------|------|------|------|------|------|------|
| Position Number | 1100 | 1011 | 1010 | 1001 | 1000 | 0111 | 0110 | 0101 | 0100 | 0011 | 0010 | 0001 |
| Data Bit | D8 | D7 | D6 | D5 | | D4 | D3 | D2 | | D1 | | |
| Check Bit | | | | | C8 | | | | C4 | | C2 | C1 |

Figura 4.12: Disposição dos bits de dados e de teste.

| Bit position | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
|-----------------|------|------|------|------|------|------|------|------|------|------|------|------|
| Position number | 1100 | 1011 | 1010 | 1001 | 1000 | 0111 | 0110 | 0101 | 0100 | 0011 | 0010 | 0001 |
| Data bit | D8 | D7 | D6 | D5 | | D4 | D3 | D2 | | D1 | | |
| Check bit | | | | | C8 | | | | C4 | | C2 | C1 |
| Word stored as | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| Word fetched as | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| Position Number | 1100 | 1011 | 1010 | 1001 | 1000 | 0111 | 0110 | 0101 | 0100 | 0011 | 0010 | 0001 |
| Check Bit | | | | | 0 | | | | 0 | | 0 | 1 |

Figura 4.13: Geração de bits de teste.

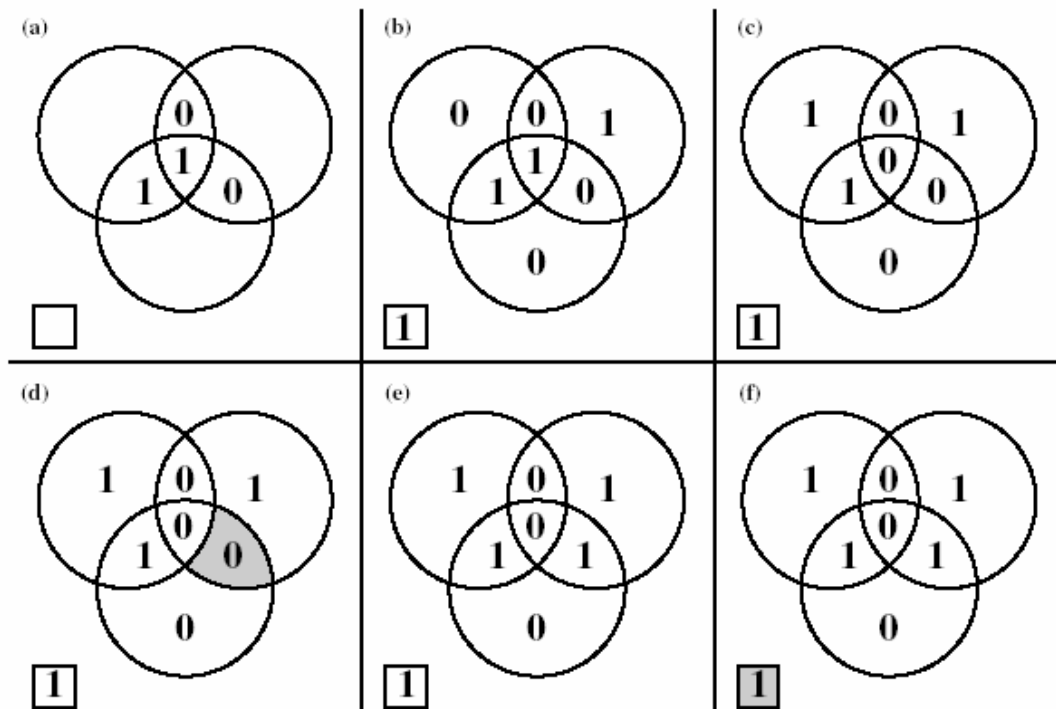


Figura 4.14: Código de Hamming SEC-DED.

Memória Oculta

Princípios

A memória oculta pretende aproximar a velocidade da memória à mais rápida das velocidades das memórias disponíveis e, ao mesmo tempo, disponibilizar grandes quantidades de memória aos preços dos tipos de memórias de semi-condutores menos dispendiosas. O conceito é ilustrado na figura 4.15. Pode ver-se uma memória relativamente grande e lenta em conjunto com uma pequena e rápida memória oculta (*cache*). A *cache* contém uma cópia de porções de memória principal. Quando o processador tenta ler uma palavra da memória, é feito um teste para determinar se a palavra está na *cache*. Em caso afirmativo, a palavra é entregue ao processador. Caso contrário, um bloco de memória consistindo de um número fixo de palavras, é lido para a *cache* e em seguida entregue ao processador. Por causa do fenómeno da localidade das referências, quando um bloco de dados é posto na *cache* para satisfazer uma simples referência à memória é de esperar que as referências futuras possam ser para outras palavras no bloco.

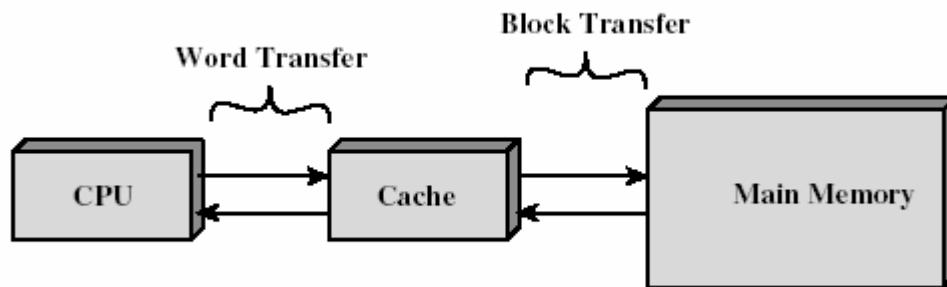


Figura 4.15: Memória principal e *cache*.

A figura 4.17 ilustra a operação de leitura. O processador gera o endereço, RA, da palavra a ler. Se a palavra está contida na *cache*, é enviada para o processador, tal como é descrito mais tarde nesta secção. Doutra forma, o bloco contendo a palavra é carregado para a *cache* e a palavra entregue ao processador.

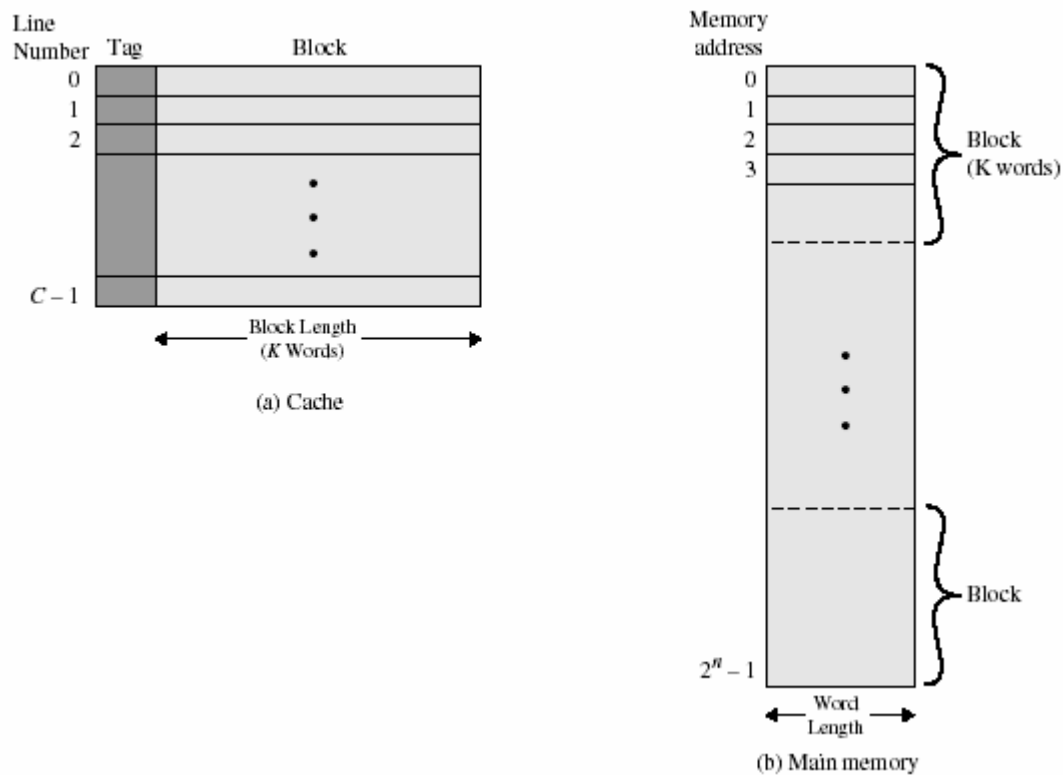


Figura 4.16: Estrutura do sistema *cache*/memória.

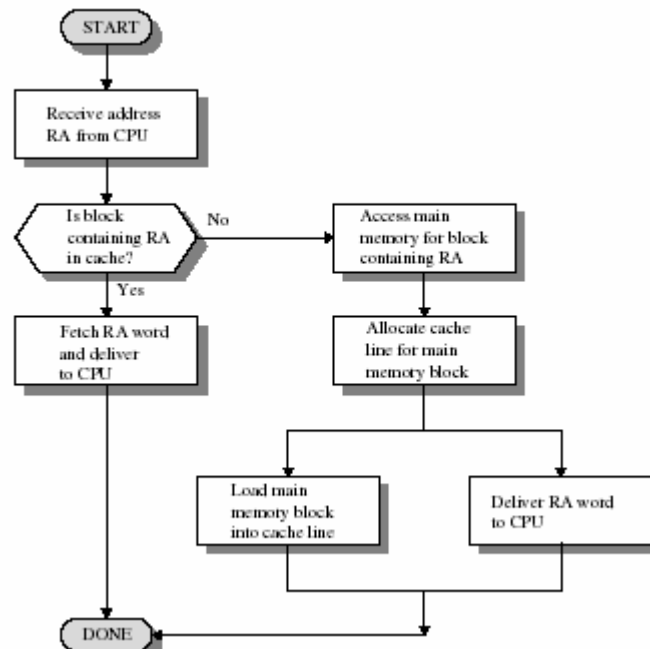


Figura 4.17: Operação de leitura da *cache*.

Elementos do projecto de *cache*

Embora haja um grande número de implementações de *cache* há apenas um número reduzido de elementos que podem ser usados para classificar e diferenciar diferentes arquitecturas de *cache*. A tabela 4.4 lista os elementos chave da lista.

Tamanho da *cache*

O primeiro elemento, tamanho da *cache*, já foi discutido. Nós gostaríamos que o tamanho da *cache* pudesse ser suficientemente pequeno para que o custo médio total por bit rondasse o da memória principal tomada isoladamente e suficientemente grande porque o tempo de acesso médio total se aproximasse do da *cache* isoladamente. Há muitas outras motivações para a minimização do tamanho da *cache*. Quanto maior a *cache*, maior o número de portas lógicas envolvidas no endereçamento da *cache*. O resultado é que as *caches* grandes tendem a ser ligeiramente mais lentas que as pequenas - mesmo quando construídas com as mesma tecnologia de integrados e postas na mesma localização tanto no integrado como na placa mãe. O tamanho da *cache* está também limitado pela superfície disponível na placa e no integrado. No Apêndice 4A chamamos a atenção para um certo número de estudos que sugerem que tamanhos da *cache* entre 1K e 512K palavras poderiam ser considerados óptimos. Porque o rendimento da *cache* é muito sensível à natureza da carga, é impossível chegar a um tamanho 'óptimo' para o tamanho da *cache*.

Tabela 4.4: Elementos do projecto de *cache*.

- Design issues
 - Size
 - Mapping Function
 - direct, associative, set associative
 - Replacement Algorithm
 - LRU, FIFO, LFU, Random
 - Write Policy
 - Write through, write back
 - Line Size
 - Number of Caches
 - single or two level
 - unified or split

Função de Correspondência

Uma vez que há menos linhas de *cache* do que blocos de memória é necessário um algoritmo para estabelecer a correspondência dos blocos da memória para linhas da *cache*. Além disso é necessário encontrar a forma de determinar quais os blocos de memória correntes que ocupam linha de *cache*. A escolha da função de correspondência dita a forma como a *cache* é organizada. Podem ser usadas três técnicas: directa, associativa e jogos associativos. Examinamos à vez cada uma delas. Em cada caso, olhamos para a estrutura geral e seguidamente para um exemplo específico. Para os três casos, os exemplos incluem os seguintes elementos:

- A *cache* possui 64KBytes
- Os dados são transferidos entre a memória principal e a memória em blocos de 4 octetos cada. Isto significa que a *cache* é organizada com 16 linhas de 4 octetos cada
- A memória principal consiste de 16MBytes, sendo cada octeto directamente endereçável por um endereço de 24-bits ($16K = 2^{14}$). Assim para efeitos de correspondência podemos considerar que a memória principal consiste de 4M blocos cada um com 4 octetos.

Correspondência Directa

A técnica mais simples, conhecida por correspondência directa, faz uma correspondência entre cada bloco da memória e uma única linha de *cache*. A figura 4.18 ilustra o mecanismo geral. A correspondência pode ser expressa como $i = j \text{ módulo } m$, onde i = número da linha de *cache*, j = número do bloco da memória principal e, m = número de linhas na *cache*

A função de correspondência é facilmente realizável usando o endereço. Para efeitos de acesso à *cache*, cada endereço da memória principal pode ser visto como se consistindo em três campos.

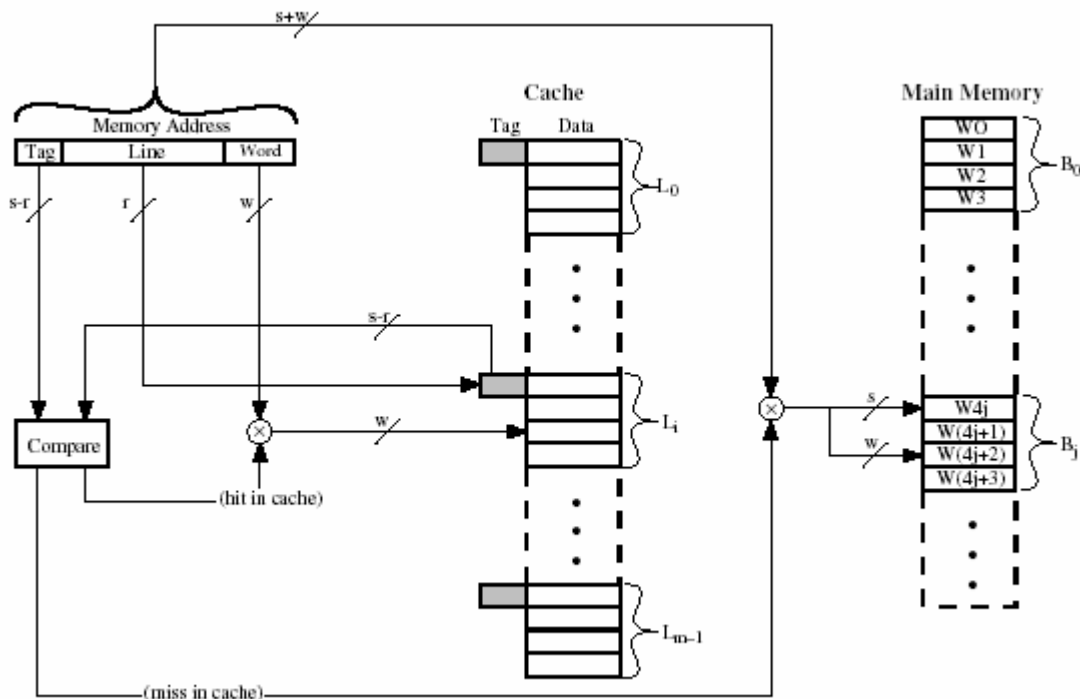


Figura 4.18: Correspondência directa [hwan93].

Os 2^r bits identificam uma única palavra ou octeto dentro de um bloco da memória principal; nas máquinas mais recentes, o endereço está ao nível do octeto. Os restantes w bits especificam um dos m blocos de memória principal. A lógica da *cache* interpreta estes w bits como um etiqueta de 2^r bits (parte mais significativa) e um campo para linha de r bits. Isto mais tarde identifica um dos m linhas de *cache*. O efeito desta correspondência é que aos blocos de memória são atribuídas as linhas de *cache* da forma como se segue:

| Linha de <i>cache</i> | Blocos de memória atribuídos |
|-----------------------|--------------------------------|
| 0 | $m = 2^r$ |
| 1 | $0, m, \dots, 2^r - m$ |
| . | |
| . | |
| . | |
| m - 1 | $1, m + 1, \dots, 2^r - m + 1$ |

Assim, o uso de uma porção do endereço como um número de linha oferece uma correspondência única de cada bloco de memória principal na *cache*. Quando um bloco é lido para a sua linha atribuída, é necessário etiquetar os dados para distingui-los dos

outros blocos que podem encaixar naquela linha. Os 2^m bits mais significativos servem para isso.

A figura 4.19 mostra o nosso exemplo usando um sistema de correspondência directa^{4.2}. No exemplo, $m = 1, 2^m = 2, 2^m - 1 = 1$, $2^m \cdot i = j$ módulo $2^m = 1$.

| Linha de <i>cache</i> | Blocos de memória atribuídos |
|-----------------------|------------------------------|
| 0 | 2^{14} |
| 1 | 00000,01000,...,FF000 |
| . | . |
| . | . |
| . | . |
| 3FFF | 00001,01001,...,FF001 |

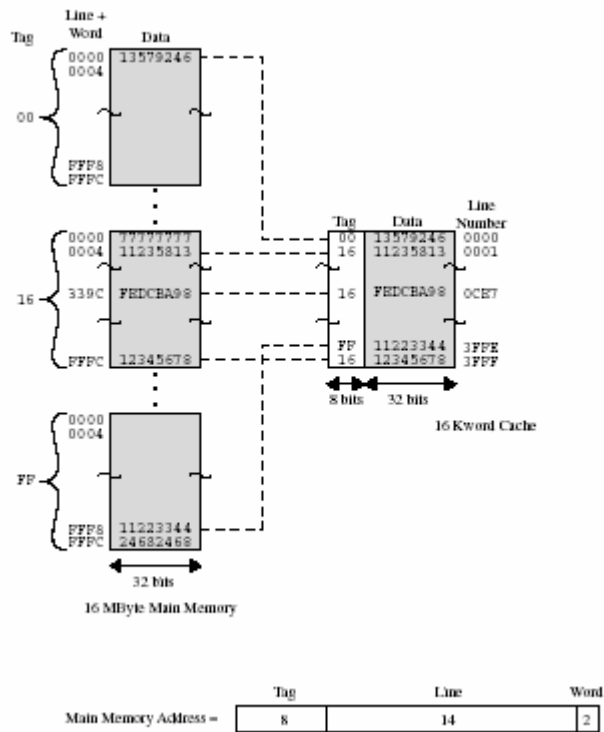


Figura 4.19: Exemplo de correspondência directa.

É de notar que dois blocos com uma correspondência no mesmo número de linha nunca têm o mesmo número de etiqueta. Assim, os blocos 2^{14} têm números de etiqueta $00FFFC, 01FF$ respectivamente.

Voltando à figura 4.18, uma operação de leitura funciona da seguinte maneira. Ao sistema de *cache* é apresentado um endereço de 24-bits. Os 14-bits do número da linha são usados como um índice na *cache* para ter acesso a uma linha particular. Se a etiqueta de 8-bits igualar o número de etiqueta presentemente guardado naquela linha, então o número de 2-bits da palavra é usado para escolher um dos quatro octetos naquela linha. Caso contrário, os 22-bits da etiqueta mais o campo da linha são usados para extrair um bloco da memória principal. O endereço efectivo que é usado para a extracção são os 22-bits da etiqueta mais a linha que são concatenados com dois bits a 0, de forma a que os quatro octetos sejam extraídos nos limites de um bloco.

A técnica de correspondência directa é simples e de realização pouco onerosa. A desvantagem principal é que há uma localização fixa na *cache* para um determinado bloco. Assim, se num programa acontecer uma referência repetida a palavras pertencentes a dois diferentes blocos que correspondem à mesma linha, então os blocos serão continuamente trocados na *cache* e o taxa de sucesso será baixa.

Correspondência associativa

A correspondência associativa supera as desvantagens da correspondência directa permitindo que cada bloco da memória principal possa ser colocado em qualquer uma das linhas da *cache*. Nesta caso, a lógica de controlo interpreta um endereço de memória simplesmente como um campo de etiqueta e de palavra. O campo etiqueta identifica univocamente um bloco da memória principal. Para determinar se um bloco está na *cache*, a lógica de controlo deve examinar simultaneamente todas as etiquetas para encontrar uma que faça a correspondência. A figura 4.20 ilustra a técnica.

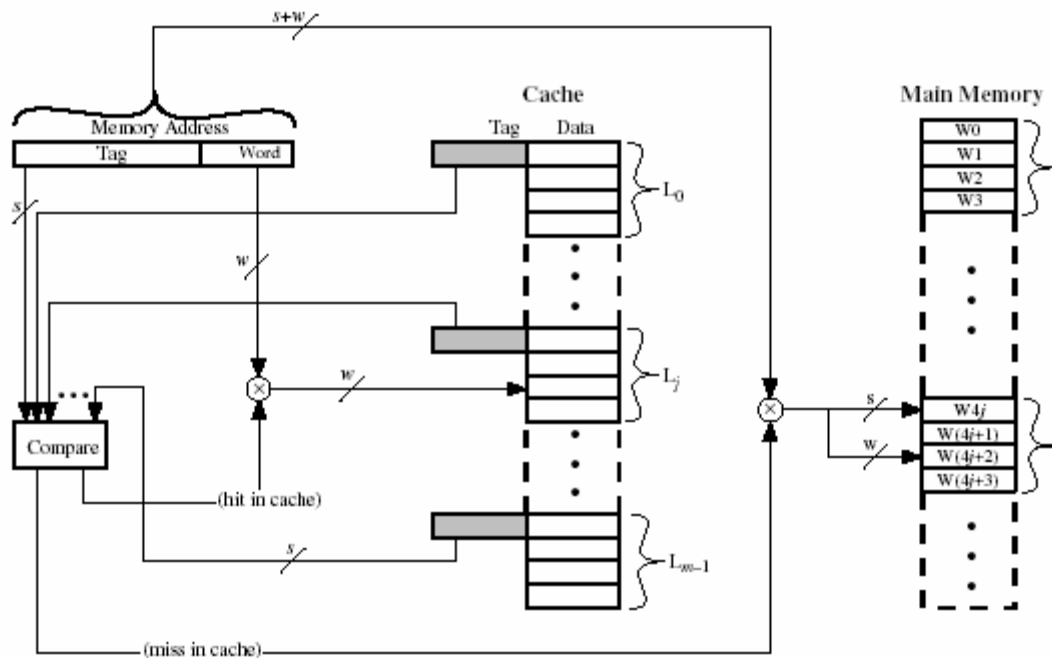


Figura 4.20: Organização da *cache* em jogos associativos.

A figura 4.21 mostra o nosso exemplo usando correspondência associativa. Um endereço na memória principal consiste de uma etiqueta de 22-bits e um número de octeto em 2-bits.

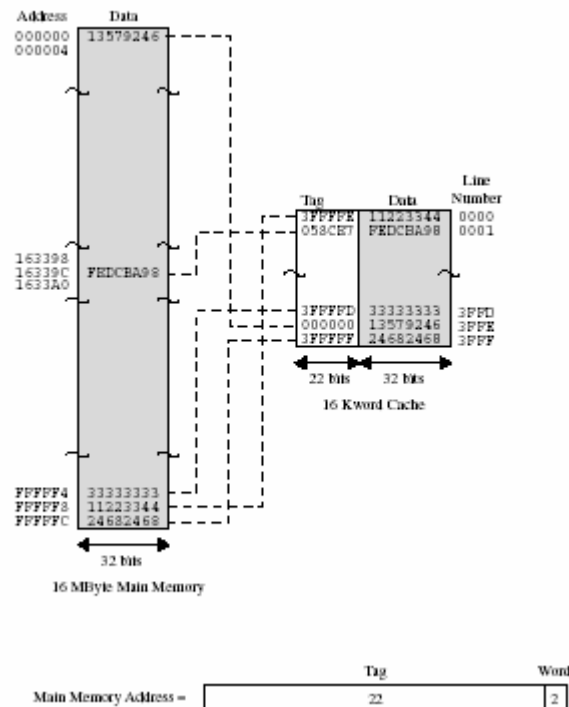


Figura 4.21: Exemplo de *cache* de correspondência associativa.

A etiqueta de 22-bits deve ser guardada conjuntamente com o bloco de 32-bits de dados para cada linha na *cache*.

Com a correspondência associativa é flexibilizada a substituição de blocos quando um novo bloco é trazido para a *cache*. Os algoritmos de substituição, discutidos mais tarde nesta secção, são desenhados para maximizar a taxa de sucesso. A principal desvantagem da correspondência associativa está na complexidade dos circuitos lógicos necessários para examinar as etiquetas de todas as linhas da *cache* em paralelo.

Correspondência em Jogos Associativos

A correspondência em jogos associativos é um compromisso que beneficia das vantagens das abordagens directa e associativa sem as respectivas desvantagens. Neste caso a *cache* é dividida em v jogos, cada um dos quais consiste de m linhas. As relações são: $m = v * k$, $i = j \text{ módulo } v$, onde i = número do jogo na *cache*, j = número do bloco da memória principal, m = número de linhas na *cache*.

Com a correspondência em jogos associativos, o bloco j pode ter uma correspondência com qualquer uma das linhas do jogo B_i . Neste caso, a lógica de controlo,

simplesmente, interpreta um endereço da memória como constituído por três campos: etiqueta, jogo e palavra. Os s -bits do jogo especificam um dos d -jogos. Os w -bits dos campos da etiqueta e do jogo especificam um dos m -blocos de memória principal. A figura 4.22 ilustra a lógica de controlo da *cache*.

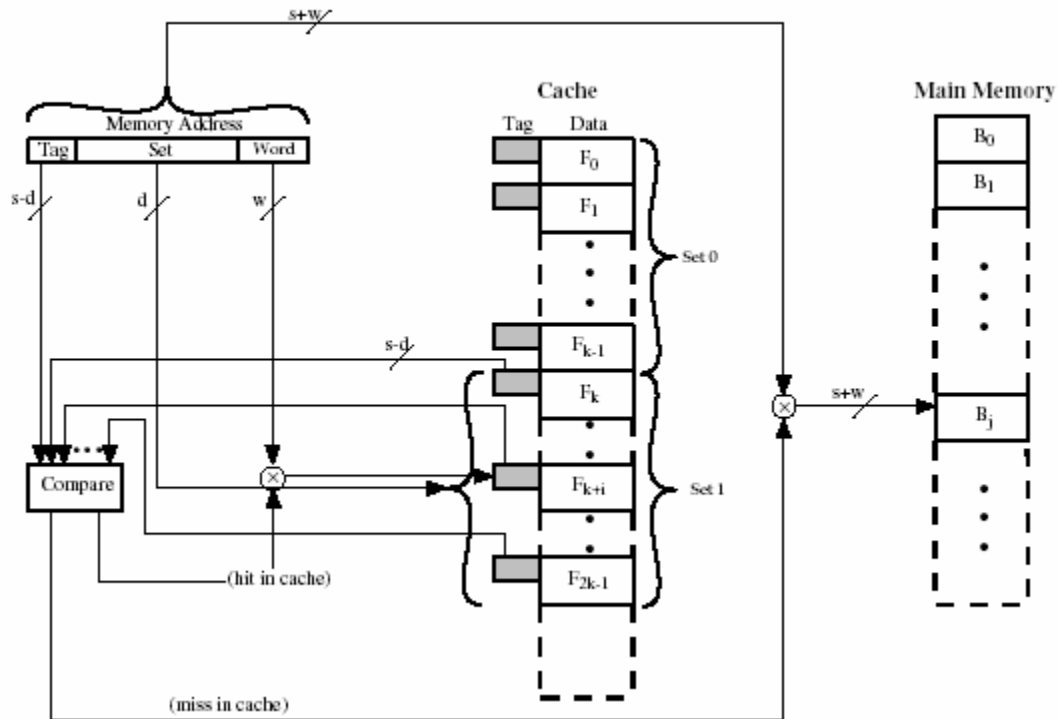


Figura 4.22: Organização de *cache* por jogos associativos de duas vias [hwan93].

A figura 4.23 mostra o nosso exemplo usando correspondência em jogos associativos com duas linhas por jogo, designada por jogo associativos de duas vias. Os 13-bits do número do jogo identificam um conjunto único de duas linhas dentro da *cache*. Também refere o número do bloco na memória principal, módulo $v=2^i$. Este determina a correspondência dos blocos em linhas. Assim, os blocos 2^{i+j} da memória principal estabelecem a correspondência com o jogo j da *cache*. Qualquer um daqueles blocos pode ser carregado para uma das duas entrada do jogo. É de notar que dois blocos que residem no mesmo jogo nunca poderão ter o mesmo valor na etiqueta. Para uma operação de leitura, os 13-bits do número do jogo são usados para determinar qual dos jogos de duas linhas deverá ser examinado. Ambas as linhas no mesmo jogo são examinadas para saber se há alguma com o mesmo número da etiqueta do endereço a que se pretende ter acesso.

No caso extremo de $v=m, k=1$, a técnica de jogos associativos reduz-se à correspondência directa e para $k=1$ reduz-se à correspondência associativa. O uso de duas linhas por jogo $v=1, k=m$ é a mais comum das organizações por jogos associativos. Melhora significativamente a taxa de sucesso em relação a taxa de correspondência directa. Jogos associativos de quatro vias $(v=m/2, k=2)$ apenas introduzem um melhoramento adicional modesto para um relativamente modesto custo adicional [May84, Hill89]. Aumentos superiores do número de linhas por jogo têm efeitos desprezáveis.

oferece, apenas, um rendimento ligeiramente inferior ao dos algoritmos baseados no uso [#!smit82!#].

Política de Escrita

Antes que um bloco residente na *cache* possa ser substituído, é necessário tomar em atenção se este foi alterado na *cache* mas não na memória principal. Se não tiver sido, então o bloco antigo na *cache* pode ser rescrito. Se foi escrito, isso significa que foi feita pelo menos uma operação de escrita naquela entrada da *cache* e que a memória principal deverá ser actualizada em concordância. É possível uma variedade de políticas de escrita, com compromissos económicos e de rendimento. Há que lidar com dois problemas. Em primeiro lugar, mais do que um dispositivo pode ter acesso à memória principal. Por exemplo um módulo de E/S pode ser capaz de ler/escrever directamente na memória. Se uma palavra tiver sido alterada apenas na *cache*, então a palavra na *cache* fica inválida. Um problema mais complexo ocorre quando múltiplos processadores estão agarrados ao mesmo barramento e cada processador tem a sua própria *cache* local. Então, se uma palavra for alterada numa das *caches*, isso iria presumivelmente invalidar uma palavra noutras *caches*.

A técnica mais simples de todas é chamada **escrita-imediata** (*write through*). Usando esta técnica, todas as operações de escrita são feitas na memória assim como na *cache*, assegurando que a memória principal está sempre válida. Qualquer outro módulo *cache*-processador pode monitorizar o tráfego para a memória principal para manter consistência com a sua própria *cache*. A desvantagem principal desta técnica é que é gerado tráfego substancial para a memória o que pode criar um engarrafamento.

Uma técnica alternativa, conhecida como **escrita adiada** (*write back*), minimiza as escritas na memória. Com a escrita adiada, as actualizações são feitas apenas na *cache*. Quando ocorre uma actualização, faz-se uma marca num bit de ACTUALIZAÇÃO (UPDATE) associado com a entrada. Então, quando um bloco é substituído é escrito posteriormente na memória principal se, e só se, o bit de ACTUALIZAÇÃO estiver marcado. O problema com a escrita adiada é que existem porções de memória invalidadas e, por isso, os acessos através dos módulos de E/S só podem ser autorizados com o consentimento da *cache*. Isto obriga a circuitos complexos e a um potencial congestionamento. A experiência tem mostrado que a percentagem das referências à memória que são escritas é da ordem dos 15 por cento [#!smit82!#].

Na organização de um barramento em que mais do que um dispositivo (tipicamente um processador) tem *cache* e a memória é partilhada é introduzido um novo problema. Se os dados numa das *caches* for alterada, isto invalida não apenas a palavra correspondente na memória principal mas, também, a mesma palavra noutras *caches* (se por acaso alguma das outras *caches* contiver a mesma palavra). Mesmo quando se usa uma política de escrita-imediata, as outras podem conter dados inválidos. Um sistema que evita este problema é dito manter coerência de *cache*. Possíveis abordagens a coerência da *cache* incluem

- *Escrita imediata com vigilância do barramento*: cada controlador de *cache* vigia as linhas de endereço para detectar operações de escrita na memória provenientes de outros donos do barramento. Se outro dono escreve numa localização da memória partilhada que também está residente na memória *cache*,

o controlador de *cache* invalida a entrada na *cache*. Esta estratégia depende da utilização de políticas escrita imediata por todos os controladores de *cache*.

- *Transparência do hardware*: hardware adicional é usado para assegurar que todas as actualizações da memória principal através da *cache* se reflectem em todas as *caches*. Assim, se um processador modifica uma palavra na sua própria *cache*, a actualização é escrita na memória principal. Adicionalmente, as mesmas palavras em qualquer das outras *caches* são actualizadas de forma similar.
- *Memória não-ocultável*: Apenas uma porção da memória principal é partilhada por mais do que um processador e isto é designado por não-ocultada. Num sistema como este, todos os acesso à memória partilhada são falhas de *cache*, porque a memória partilhada nunca é copiada para a *cache*. A memória não ocultável pode ser identificada usando lógica de selecção de integrados ou os bits mais significativos do endereço.

Tamanho do bloco

Um outro elemento de projecto é o tamanho de bloco ou de linha. Quando um bloco de dados é extraído e colocado na *cache*, não é apenas a palavra pretendida mas um certo número de palavras adjacentes que são extraídas. Como o tamanho do bloco cresce desde tamanhos pequenos até grandes, a taxa de sucesso começa, primeiro, por aumentar porque causa do princípio da localidade: a elevada probabilidade de que os dados na vizinhança das palavras referenciadas sejam muito provavelmente referenciadas num futuro próximo. À medida que o tamanho do bloco cresce, mais dados úteis são trazidos para a *cache*. Contudo, a taxa de sucesso começa a baixar, à medida que o bloco se torna cada vez maior e a probabilidade de usar a nova informação extraída se torna inferior à probabilidade de reutilizar a informação que tem de ser substituída. Dois efeitos específicos entram em jogo:

1. Blocos maiores reduzem o número de blocos que cabem na *cache*. Porque cada extracção de um bloco rescreve dados mais antigos na *cache*, um número pequeno de blocos resulta em que os dados são rapidamente rescritos depois de entrados na *cache*.
2. À medida que um bloco se torna maior, cada palavra adicional está mais distante da palavra pretendida, por isso é menos provável que venha a ser necessária num futuro próximo.

A relação entre o tamanho do bloco e a taxa de sucesso é complexa, estando dependente nas características de localidade de um programa particular e, em definitivo, nenhum valor óptimo foi encontrado. Um tamanho de 4 a 8 unidades endereçáveis (palavras ou octetos) parece estar razoavelmente próximo do óptimo [smit82,przy88,przy90].

Número de *caches*

Quando, originalmente, se introduziram as *caches*, os sistemas típicos tinham apenas uma *cache*. Mais recentemente, o uso de múltiplas *caches* transformou-se na norma. Dois aspectos do projecto deste tópico dizem respeito ao número de níveis de *cache* e ao uso de *caches* unificadas versus *caches* repartidas.

Caches Simples versus Dois-Níveis

À medida que a densidade da lógica foi crescendo, tornou-se possível ter a *cache* no integrado do processador: *on-chip* (*cache* interna). Quando comparada com a *cache* acessível através do barramento externo, a *cache* interna reduz a actividade externa e dessa forma apressa os tempos de execução e aumento o rendimento global do sistema. Quando as instruções ou dados pretendidos são encontrados na *cache* interna, os acessos ao barramento são eliminados. Por causa dos muito curtos caminhos de dados internos ao processador, quando comparados com o comprimento dos barramentos, os acessos à *cache* interna podem completar-se apreciavelmente mais depressa do que, mesmo, com ciclos de barramento de zero estado de espera. Além de que, durante aquele período, o barramento fica livre para suportar outras transferências.

A inclusão da *cache* interna deixa em aberto a questão de saber se continua a ser desejável uma *cache* fora do integrado do processador, ou externa. Tipicamente, a resposta é sim e a maior parte dos projectos actuais incluem ambas as *caches*, interna e externa. A organização resultante é conhecida como uma *cache* a dois níveis, com a *cache* interna designada por nível 1 (L1) e a *cache* externa designada por nível 2 (L2). A razão para a inclusão de uma *cache* L2 é a seguinte. Se não houver *cache* L2 e o processador fizer um pedido de acesso a uma posição de memória que não está na *cache* de nível 1, então o processador tem de fazer um acesso à memória DRAM ou ROM através do barramento. Devido à baixa velocidade do barramento e aos baixos tempos de acesso à memória, isto resulta num rendimento pobre. Por outro lado, se for usada uma *cache* L2 em SRAM, então, a informação que falha com frequência pode ser rapidamente extraída. Se a SRAM for suficientemente rápida para poder igualar a velocidade do barramento, então pode fazer-se um acesso aos dados usando transacções em zero-estados de espera, o mais rápido dos tipos de transferência pelo barramento.

O potencial de poupança devida ao uso de *cache* L2 depende da taxa de sucesso tanto das *caches* L1 como de L2. Vários estudos têm mostrado que, em geral, o uso de uma *cache* de segundo nível melhora, efectivamente, o rendimento (e.g., ver [#!azim92!#,#!novi93!#]).

Unificada versus Repartida

Quando a *cache* interna fez a sua primeira aparição, muitos dos desenhos consistiam de uma *cache* simples usada para guardar as referências partanto para as instruções como para os dados. Mais recentemente, tornou-se comum dividir a *cache* em duas: uma dedicada às instruções e outra dedicada aos dados.

Há um série de potenciais vantagens numa *cache* unificada:

1. Para um determinado tamanho da *cache*, a *cache* unificada tem um taxa de sucesso superior à da *cache* repartida porque aquela faz um balanceamento automatico de carga entre extracção de instruções e de dados. Isto é, se um padrão de execução envolve muito mais extracções de instruções do que extracções de dados, então a *cache* tende a ficar cheia com instruções e se um padrão de execução envolve relativamente mais extracções de dados, acontece o oposto.
2. É apenas necessário projectar e implementar uma *cache*

Apesar destas vantagens, a tendência é para *caches* repartidas, particularmente em máquinas super-escalares tais como o Pentium e o Power PC, que põem a ênfase na execução paralela de instruções e a predição e pré-extracção de instruções futuras. A chave para a vantagem do projecto de *cache* repartida é que esta elimina a contenção pela *cache* entre o processador de instruções e a unidade de execução. Isto é importante em qualquer desenho que assenta o encadeamento de instruções. Tipicamente, o processador extrai as instruções num tempo em avanço e enche um tampão, ou linha de encadeamento, com instruções para serem executadas. Suponha agora que temos uma *cache* unificada de instruções e de dados. Quando a unidade de execução faz um acesso à memória para carregar e guardar dados, o pedido é submetido à *cache* unificada. Se, ao mesmo tempo, a unidade de pré-extracção de instruções emitir um pedido de leitura para a *cache* por causa de uma instrução, aquele pedido ficará temporariamente bloqueado até que a *cache* possa servir primeiro a unidade de execução, de forma a permitir-lhe completar a instrução correntemente em execução. Esta competição pela *cache* pode degradar o rendimento interferindo com o uso eficiente da linha de encadeamento de instruções. A estrutura de um *cache* repartida supera esta dificuldade.

Organização da *Cache* do Pentium

A evolução da organização da *cache* vê-se claramente na evolução dos microprocessadores da Intel. O 80386 não inclui *cache* interna. O 80846 inclui uma *cache* interna unificada de 8Kbytes, usando um tamanho de linha de 16 octetos e uma organização em jogos associativos de quatro vias. O Pentium inclui duas *caches* internas, uma para dados e outra para instruções. Cada *cache* é de 8Kbytes, usando um tamanho de linha de 32 octetos e uma organização em jogos associativos de duas vias.

A figura 4.24 mostra uma versão simplificada da organização do Pentium que destaca a colocação das duas *caches*. A parte mais importante das unidades de execução são duas unidades de lógica e de aritmética inteira que podem executar em paralelo e uma unidade de vírgula flutuante com os seus próprios registos e componentes próprios de multiplicação, soma e divisão. A *cache* de dados alimenta tanto as operações inteiras como em vírgula flutuante. A *cache* de dados é de porta-dupla. As duas portas de 32-bits podem ser usadas para ligar, separadamente, às duas unidades inteiras da ALU e podem ser combinadas para uma ligação de 64-bits à unidade de vírgula flutuante. A *cache* de instruções, apenas de leitura, alimenta um tampão para pré-extracção; a operação de encadeamento de instruções é discutida no capítulo 13.

A figura 4.25 representa os elementos chave da *cache* interna de dados. Os dados na *cache* consistem em 128 jogos de 2 linhas cada. Esta é logicamente organizada em duas vias de 4KBytes. Associada com cada entrada está uma etiqueta e dois bits de estado; estes estão logicamente organizados em dois directórios, de forma a que há uma entrada no directório para cada linha da *cache*. A etiqueta corresponde aos 20 bits mais significativos do endereço da memória dos dados guardados na linha correspondente. O controlador de *cache* usa um algoritmo de substituição LRU e um simples bit (LRU) está associado a cada jogo de duas entradas.

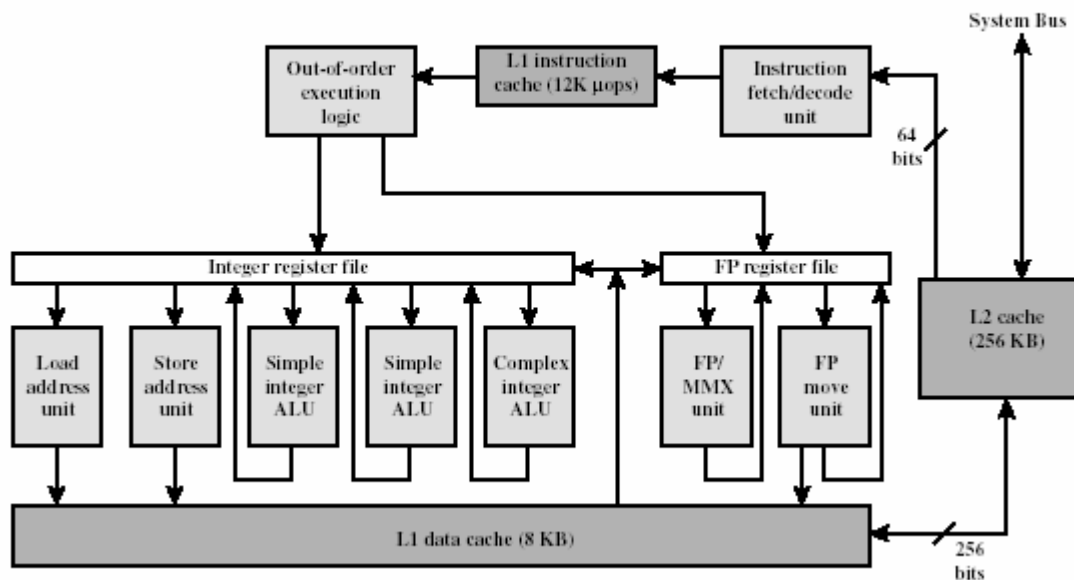


Figura 4.24: Diagrama de blocos do processador Pentium.4

A *cache* de dados emprega uma política de escrita adiada: os dados são escritos na memória principal quando são descartados da *cache*, apenas, se antes tiverem sido actualizados. O processador Pentium pode ser dinamicamente configurado para suportar *cache* de escrita imediata.

O processador Pentium suporta o uso de uma *cache* externa de nível 2. Esta *cache* pode ser de 256 ou 512 KBytes, usando 32, 64, ou 128 octetos por linha. A *cache* externa é em jogos associativos de duas vias.

Consistência da *cache* de Dados

Para garantir a consistência, a *cache* de dados suporta um protocolo conhecido como MESI (modificada/exclusiva/partilhada/inválida). O MESI foi projectado para suportar os requisitos de consistência de um sistema de multi-processador mas é, também, útil numa organização baseada num único processador Pentium.

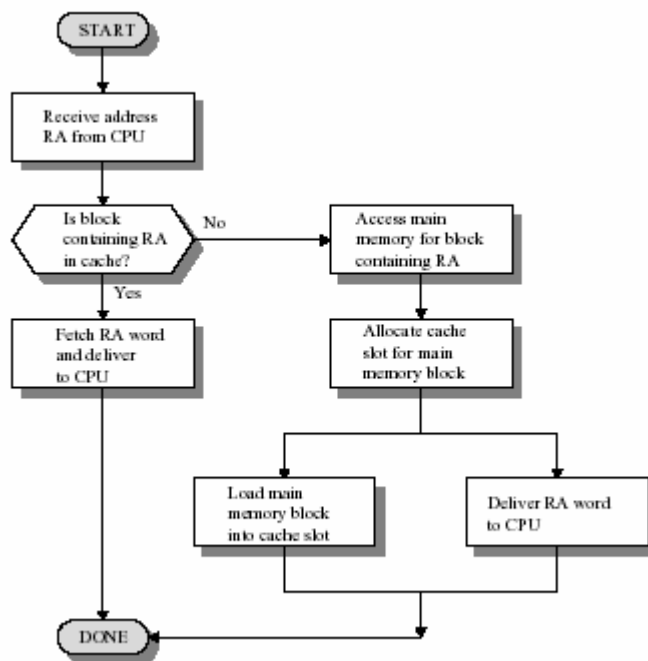


Figura 4.25: Estrutura da *cache* interna de dados do Pentium II [ande93].

A *cache* de dados inclui dois bits de estado por etiqueta, de forma a que cada linha pode estar num de quatro estados:

- *Modificada*: A linha de *cache* foi modificada (diferente da memória principal) e está disponível apenas nesta *cache*.
- *Exclusiva*: A linha de *cache* está igual à da memória principal e não existe em nenhuma outra *cache*.
- *Partilhada*: A linha na *cache* é igual à da memória principal e pode co-existir em outra *cache*.
- *Inválida*: A linha na *cache* não contém dados válidos.

A tabela 4.5 resume o significado dos quatro estados. Começamos por considerar a acção no caso de um único processador^{4.3}.

Tabela 4.5: MESI-Estados das linhas na *cache*.

Aqui, a principal preocupação está na interacção entre as *caches* de nível 1 e nível 2. Uma linha de *cache* parte de um estado inicial de inválida (I), após o arranque. Quando novos dados são lidos para a linha invalidada, esses dados são extraídos da memória principal e guardados em primeiro lugar na *cache* L2 e seguidamente na *cache* de dados L1. O estado da linha na *cache* de dados L1 passa a partilhado (S). As leituras subsequentes não afectam o estado da *cache*.

Agora suponhamos que uma das unidades de execução escreve nesta linha. A linha na *cache* L1 deverá ser actualizada. Neste ponto, a linha na *cache* L2 fica obsoleta. Para evitar a inconsistência da *cache*, a primeira vez que a linha é actualizada na *cache* L1, a operação de escrita propaga-se à *cache* L2 e o estado da linha em L1 passa para o (E) exclusiva. Depois disso, uma actualização posterior da linha em L1 altera o estado para (M) modificada. Nas actualizações posteriores, a linha mantém-se no estado M e para todas aquelas actualizações subsequentes, não há nenhuma transferência para L2. Assim, isto é conhecido como uma política de escreve uma vez (*write-once*). Finalmente, quando é necessário substituir uma linha na *cache* L1, esta, se está no estado S ou E, não é necessário propagar a escrita para o exterior. Se a linha está no estado M, o seu valor reverte para a *cache* L2 e seguidamente descartada da *cache* L1. Quando novos dados são lidos na linha L1, a linha é marcada com o estado (S), como antes.

A operação do ponto de vista da *cache* L2 é mais complexa. Quando uma linha é inicialmente lida para as *caches* L1 e L2, a linha da *cache* L2 é marcado com o estado (E), o que indica que os dados são exclusivos da *cache* L2 e do seu processador e *cache* L1 associados. Quando ocorre uma escrita uma vez, a *cache* L2 actualiza a linha e passa-a ao estado (M). A *cache* L2 não será avisada das subsequentes actualizações desta linha pela *cache* L1. Em consonância, se um outro senhor do barramento (*bus master*) tentar ler os dados guardados numa linha L2 que está no estado M, a *cache* L2 faz o senhor de barramento retroceder (*back-off*) e passa o endereço pretendido para o processador. O processador Pentium provoca um ciclo de escrita-adiada para actualizar a memória principal. A *cache* L2 permite, então, que o senhor do barramento efectue a respectiva operação de leitura.

Se um outro dono de barramento tenta escrever dados que estão numa linha L2 no estado M, então de novo, a lógica da *cache* L2 bloqueia, temporariamente, a acção. A *cache* L2 tem de assegurar que as operações são executadas na sequência apropriada. A *cache* L2 não pode descartar, simplesmente, a sua linha e deixar que a operação prossiga, porque o outro senhor do barramento pode alterar um porção da linha diferente da que tinha sido modificada na linha da *cache* L2. Por isso, a linha L2 deve ser escrita para memória antes de a outra escrita se efective. Mas, antes que isso possa acontecer, a *cache* L2 tem de determinar se a linha correspondente na linha da *cache* L1 tinha sido actualizada desde a escrita uma vez. Assim, a sequência correcta de passos é a seguinte:

1. A *cache* L2 detecta e bloqueia a operação de escrita
2. A *cache* L2 assinala a *cache* L1 com o endereço da operação de escrita. Se a *cache* L1 tiver sido actualizada desde a escrita uma vez, esta executa uma escrita imediata para a memória principal. Em qualquer caso, a linha afectada é declarada inválida (estado I).
3. Se a *cache* L1 não tiver efectuado uma escrita imediata, a *cache* L2 actualiza a memória principal. Em qualquer caso, esta declara a linha afectada como inválida (estado I).
4. A *cache* L2 liberta o dono do barramento, permitindo-lhe completar a operação de escrita.

Regressaremos ao protocolo MESI no capítulo 16 e examinaremos a sua operação num configuração multi-processador.

Controlo da *Cache*

A *cache* interna é controlada por dois bits em um dos registos de controlo, chamados bits inibe *cache*, CD (*cache disable*), e escrita não-imediata, NW (*not writethrough*) (Tabela 4.6). Há também duas instruções Pentium que podem ser usadas para controlar a *cache*: INVD descarta a memória *cache* e assinala a *cache* externa (se existir) para esta, também, descartar. WBINVD executa a mesma função e também assinala uma *cache* externa de escrita-adiada para reverter o valor dos blocos modificados antes de descartar.

Tabela 4.6: Modos de operação da *cache* do Pentium.

Organização da *Cache* do PowerPC

A organização da *cache* no PowerPC tem vindo a sofrer alteração com cada novo modelo da família PowerPc, reflectindo o não afrouxar da procura de rendimento que é a força condutora de todos os projectistas de microprocessadores.

A tabela 4.7 mostra esta evolução. O modelo original, o 601, inclui uma *cache* simples código/dados de 32KOctetos em jogos associativos de 8-vias. O 603 emprega um desenho RISC mais sofisticado mas tem uma *cache* mais pequena: 16KOctetos em *caches* separadas para instruções e para dados, ambas usando uma organização em jogos associativos de 2-vias. O resultado é que o 603 atinge aproximadamente o mesmo rendimento que o 601 a mais baixo custo. O 604 e 620 duplicaram, cada um, o tamanho das *caches* dos modelos precedentes.

Tabela 4.7: *Caches* internas do PowerPC

| Model | Size | Bytes/Line | Organization |
|-------------|------------|------------|-----------------------|
| PowerPC 601 | 1 32-KByte | 32 | 8-way set associative |
| PowerPC 603 | 2 8-KByte | 32 | 2-way set associative |
| PowerPC 604 | 2 16-KByte | 32 | 4-way set associative |
| PowerPC 620 | 2 32-KByte | 64 | 8-way set associative |
| PowerPC G3 | 2 32-Kbyte | 64 | 8-way set associative |
| PowerPC G4 | 2 32-Kbyte | 32 | 8-way set associative |

A figura 4.26 fornece uma vista simplificada da organização do PowerPc 620, realçando a colocação das duas *caches*; a organização dos outros membros da família é semelhante. As unidades de execução nucleares são três unidades de aritmética e lógica, que podem trabalhar em paralelo, e uma unidade de vírgula flutuante com os seus próprios registos e os seus próprios componentes para multiplicação, adição e divisão. A *cache* de dados alimenta tanto as operações inteiras como em vírgula-flutuante, através da unidade de carga/armazena. A *cache* de instruções que é apenas de leitura, alimenta a unidade de instruções, cuja operação é discutida no capítulo 12.

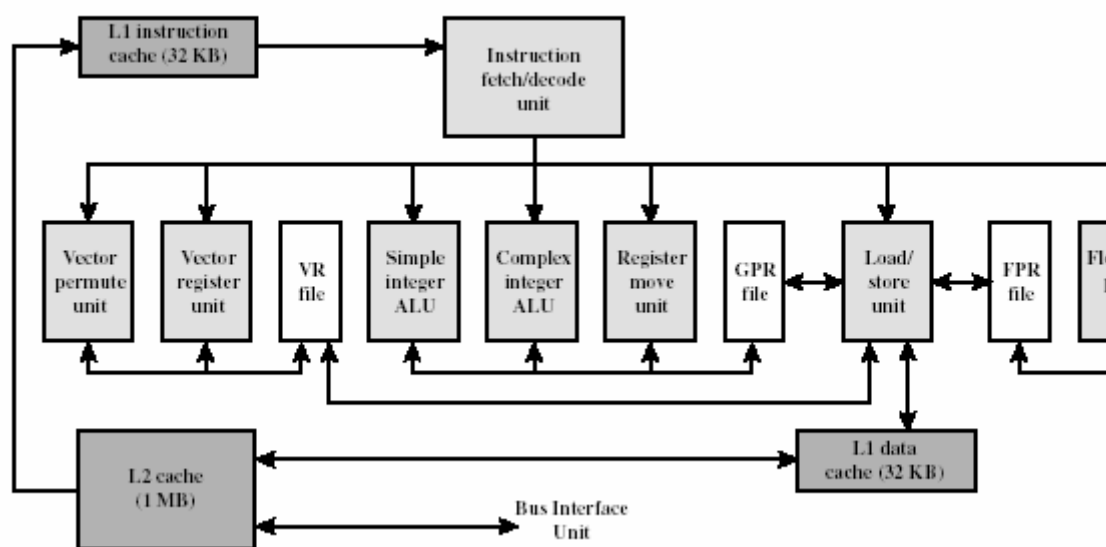


Figura 4.26: Diagrama de blocos do PowerPC 620.

As *caches* internas são em jogos associativos de oito-vias e usam o protocolo de coerência MESI. O protocolo foi estendido para incluir um novo estado chamado Atribuído (A) (*Allocated*). Este estado é usado quando um bloco de dados numa linha é enviado para o exterior e substituído. O estado é A até que os dados antigos sejam salvaguardados e os novos dados trazidos para a *cache*. Nesse momento, o estado passa para S ou E, conforma as circunstâncias (Figura 4.27).

Organização DRAM avançada

Tal como foi discutido no capítulo 2, um dos mais críticos gargalos de um sistema quando se usam processadores de elevado rendimento é o interface com a memória interna. Este interface é o caminho mais importante do sistema de computação, na sua totalidade. O bloco básico de construção da memória principal reside nos integrados de DRAM, tal foi por mais de 20 anos e, até recentemente, não tem havido mudanças significativas na arquitectura das DRAM desde o início dos anos 70. Os integrados DRAM tradicionais estão limitados tanto pelo sua arquitectura interna como pelo seu interface com o barramento de memória do processado.

Vimos que uma forma de atacar o problema do rendimento das DRAM da memória principal foi inserir um ou mais níveis de *cache* SRAM de alta-velocidade entre a DRAM da memória principal e o processador. Mas as SRAM são muito mais dispendiosas que a DRAM e a expansão do tamanho da *cache* a partir de um certo ponto produz uma diminuição proporcional ao crescimento.

Nos últimos anos, têm vindo a ser explorados um certo número de melhoramentos à arquitectura básica das DRAM e, alguns deles, estão agora no mercado. Não é claro neste ponto se algum dos melhoramentos irá emergir como a única norma DRAM ou se sobreviverão várias. Esta secção oferece uma visão geral destas novas tecnologias.

DRAM melhorada

Talvez a mais simples das novas arquitecturas de DRAM seja a DRAM melhorada (EDRAM), desenvolvida por Ramtron [#!bond94!#]. A EDRAM integra uma pequena SRAM *cache* num integrado DRAM genérico.

A figura 4.28 ilustra uma versão de 4-Mbit da EDRAM. A *cache* SRAM guarda o conteúdo total da última fiada lida que consiste em 2048 bits, ou 512 porções de 4-bits. Um comparador guarda o valor de 11-bits do mais recente endereço de fiada seleccionado. Se o próximo acesso for à mesma fiada, então o acesso apenas necessita de ser feito à *cache* SRAM

A EDRAM inclui outras possibilidades que melhoram o rendimento. As operações de refrescamento podem ser conduzidas em paralelo com as operações de leitura da *cache*, minimização do tempo em que o integrado está indisponível devido ao refrescamento. É também de notar que o caminho de leitura, desde a fiada na *cache* até à porta de saída, é independente do caminho de escrita do módulo de E/S até aos sensores amplificadores. Isto torna possível que um acesso subsequente de leitura à *cache* possa ser satisfeito em paralelo com a conclusão da operação de escrita.

Estudos feitos pela Ramtron indicam que a EDRAM funciona tão bem, ou melhor, que uma DRAM convencional com uma maior SRAM exterior (à DRAM).

Cache DRAM

A *cache* DRAM (CDRAM) desenvolvida pela Mitsubishi [#!hida90!#] é semelhante à EDRAM. A CDRAM inclui uma *cache* SRAM maior do que a EDRAM (16 vs 2kb).

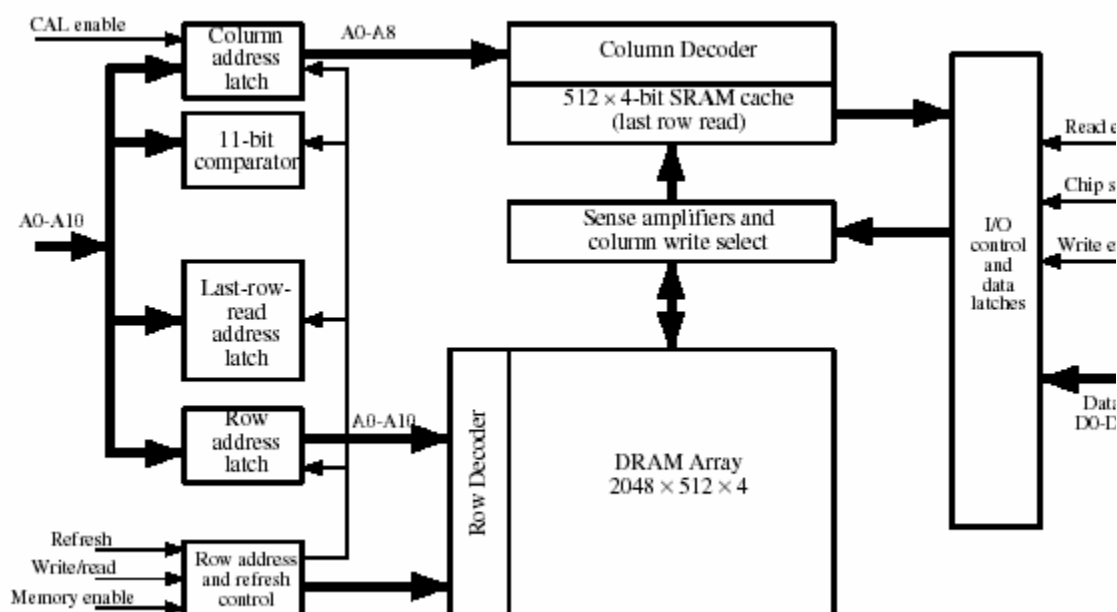


Figura 4.28: DRAM dinâmica melhorada (EDRAM).

A SRAM da CDRAM pode ser usada de duas formas. Na primeira, esta pode ser usada como uma *cache* verdadeira, consistindo de um certo número de linhas de 64-bits. Isto contrasta com EDRAM, na qual a *cache* SRAM apenas contém um bloco, nomeadamente, o da linha a que mais recentemente foi feito o acesso. O modo *cache* dos CDRAM é efectivo para os acessos aleatórios habituais à memória .

A SRAM da CDRAM pode também ser usada como um tampão para suportar o acesso em série a um bloco de dados. Por exemplo para refrescar um ecrã bit-mapped, a CDRAM pode pré-extrair os dados da DRAM para o tampão da SRAM. Os acessos subsequentes ao integrado tem como resultado o acesso exclusivo a SRAM.

DRAM Síncrona

Uma abordagem assaz diferente, para melhorar o rendimento da DRAM, é a DRAM síncrona (SDRAM) que tem vindo a ser desenvolvido por um consórcio de várias companhias [194].

Ao contrário das DRAM típicas que são assíncronas, a SDRAM troca dados com o processador sincronizada por um sinal de relógio externo que corre à velocidade do barramento memória/processador sem impôr estados de espera.

Numa DRAM típica, o processador apresenta endereços e níveis de controlo à memória, indicando que um determinado conjunto de dados de uma localização particular na memória deveriam ser lidos ou escritos na DRAM. Depois do retardamento provocado pelo tempo de acesso, a DRAM lê ou escreve os dados. Durante a demora do tempo de acesso, a DRAM executa várias funções internas, tais como a activação das linhas e linhas de colunas, de elevada capacidade eléctrica, apreendendo os dados e encaminhando-os para fora através dos tampões de saída. O processador tem, simplesmente, de esperar durante aquele intervalo, o que reduz o rendimento do sistema .

Com acessos síncronos, a DRAM move os dados para dentro e para fora sob o controlo do relógio do sistema. O processador ou outro dono do barramento envia instruções e informação de endereçamento que é agarrada pela DRAM. O integrado DRAM responde a seguir a um certo número de ciclos de relógio. Entretanto o dono pode realizar outras tarefas, em segurança, enquanto a SDRAM está a processar o pedido.

A figura 4.29 mostra a lógica interna da SDRAM. A SDRAM emprega um modo rajada para eliminar o tempo de estabelecimento do endereço e os tempos de pré-carga que se seguem ao primeiro acesso. No modo rajada, uma série de bits de dados podem ser postos rapidamente cá fora, ao ritmo do relógio, a seguir ao acesso do primeiro bit. Este modo é útil quando todos os bits a que se pretende ter acesso fazem parte de uma sequência e na mesma linha da matriz do acesso inicial.

Adicionalmente, a SDRAM tem uma arquitectura interna de duplo-banco que aumenta as oportunidades para o paralelismo no integrado.

O registo de modo e a lógica de controlo associada é uma outra característica que distingue as SDRAMs das DRAMs convencionais. O registo de modo especifica o comprimento da rajada que é o número de unidades separadas de dados que, sincronamente, alimentam o barramento. O registo, também, permite ao programador ajustar a o tempo que decorre entre a recepção do pedido de leitura e o início da transferência dos dados.

A SDRAM funciona melhor quando transfere grandes quantidades de dados em série, tal como é o caso de aplicações de processamento de texto, folhas de cálculo e multi-média.

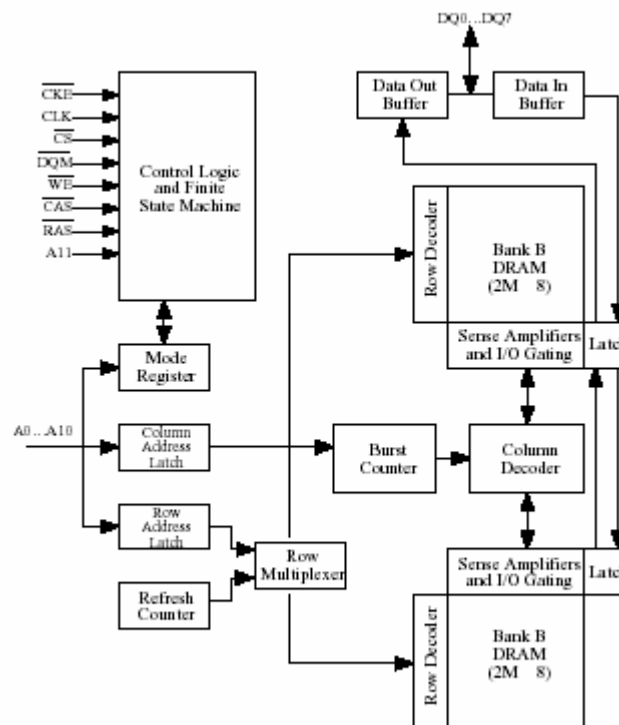


Figura 4.29: DRAM dinâmica Síncrona (SDRAM)[PRZY94].

DRAM Rambus

A RDRAM, desenvolvida por Rambus[#!garr94!#], é uma abordagem mais revolucionária ao problema da largura de banda. Os integrados RDRAM são invólucros verticais, com todos os pinos do mesmo lado. O integrado troca dados com o processador através de 28 fios com não mais de 12 centímetros de extensão. O barramento pode endereçar até 320 RDRAM integrados a uma taxa de 500 Mbps. Comparável com os 33 Mbps das DRAM assíncronas.

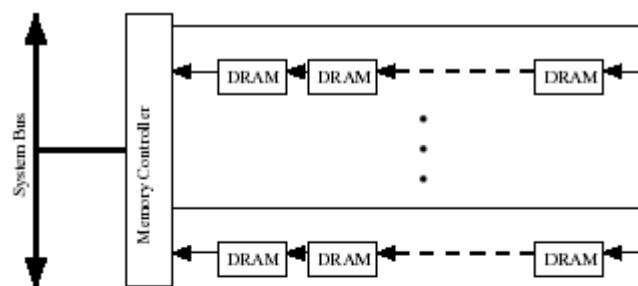
O barramento especial da RDRAM despacha endereços e informação de controlo usando um protocolo assíncrono orientado ao bloco. Depois de de 480 nano-segundos iniciais de tempo de acesso, esta é capaz de uma taxa de transferência de dados de 500

Mbps. O que torna esta possível velocidade é o próprio barramento que define impedâncias, impulsos de relógio e sinais de forma muito precisa. Em vez de ser controlada pelos sinais explícitos de RAS,CAS, R/W e CE, usados na DRAM convencionais, uma RDRAM obtém um pedido de memória através do barramento de alta-velocidade. Este pedido contém o endereço pretendido, o tipo de operação e o número de octetos a operar.

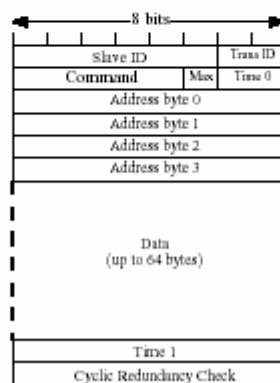
RamLink

A mais radical das alterações feitas às DRAM tradicionais encontra-se no produto RamLink [gjes92], desenvolvido como uma parte do esforço de um grupo de trabalho da IEEE chamado Scalable Coherent Interface (SCI). A RamLink concentra-se na interface memória/processador mais do que na arquitectura interna dos integrados DRAM.

A RamLink é um interface de memória com ligações ponto a ponto organizadas em anel (figura 4.30a) O tráfego no anel é gerido por um controlador de memória que envia mensagens para os integrado DRAM que actuam como nodos do anel. Os dados são trocados sob a forma de pacotes (figura 4.30b).



(a) RamLink Architecture



(b) Packet format

Figura 4.30: RAMLink.

Os pacotes pedidos iniciam transacções de memória. São enviados pelo controlador e contêm um cabeçalho com o comando, o endereço, o somatório para teste de erros e no caso de um comando de escrita, os dados a escrever. O cabeçalho consiste no tipo, no tamanho e na informação de controlo e contém, ou um tempo específico para resposta,

ou o tempo máximo permitido para o servo responder. A informação de controlo inclui um bit que indica se os pedidos subsequentes são para endereços sequenciais. Podem ser activadas simultaneamente até quatro transacções por dispositivo; assim, todos os pacotes têm dois bits ID de identificação da transacção para estabelecer a correspondência exacta entre pacotes de pedidos e pacotes de resposta.

Quando a leitura se faz com sucesso, a DRAM serva envia um pacote de resposta que inclui os dados lidos. Quando a leitura é mal sucedida a DRAM serva emite um pacote de retransmissão que indica quanto tempo adicional é necessário para completar a transacção.

A pujança da RamLink resulta de uma abordagem asente numa arquitectura escalável que suporta um pequeno número ou grande número de DRAM que não dita a estrutura interna da DRAM. O arranjo em anel da RamLink é desenhado para coordenar a actividade de muitas DRAM e fornecer um interface eficiente para o controlador de memória.

A. Pina/2000-05-10

Departamento de Informática/Universidade do Minho

This document was created with Win2PDF available at <http://www.daneprairie.com>.
The unregistered version of Win2PDF is for evaluation or non-commercial use only.