Parte IV – Sistema de Memória

Os sistemas de computação utilizam vários tipos de dispositivos para armazenamento de dados e de instruções. Os dispositivos de armazenamento consistem em memória principal e memória secundária. A memória primária é volátil com tempos de acesso menores, mas mais cara e de menor dimensão. É usada para guardar dados e instruções durante a execução de aplicações. A memória secundária é não volátil e consiste em discos, CDROMs, DVDs, etc. É mais lenta que a memória principal, mas tem um custo menor e uma capacidade maior. É usada como memória virtual e para armazenamento de ficheiros.

1 Projecto do Sistema de Memória

O projecto do sistema de memória principal consiste em identificar as memórias necessárias, os endereços a atribuir a cada uma delas e os blocos de memória física a usar na sua implementação. Nas secções seguintes analisamos como se implementam blocos de memória de determinada dimensão à custa de blocos físicos com determinado tamanho e como se atribuem os endereços disponíveis a cada um dos blocos de memória ligados ao bus do sistema.

1.1 Implementação dos Blocos de Memória

Os módulos de memória do sistema são implementados com memórias comerciais que se apresentam em dimensões pré-determinadas. Como tal, é normalmente necessário implementar um determinado módulo de memória à custa de vários blocos de memória comerciais.

Como foi visto no capítulo 1, um módulo de memória tem como entradas um conjunto de linhas de endereço, de dados e de controlo (ver figura 1). As linhas de controlo consistem num sinal de controlo do tipo de acesso (leitura/escrita), num sinal de selecção da memória (*CS – Chip Select*) e, pode ter ainda um sinal de activação das saídas de dados (*OE – Output Enable*). O sinal CS permite desactivar o acesso à memória colocando as saídas de dados em alta impedância Desta forma, é possível ligar vários módulos a um bus único, tendo apenas de garantir que apenas um dos módulos está activo de cada vez.

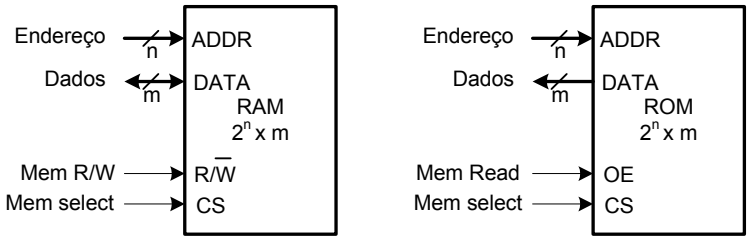


Figura 1 – Entradas/saídas dos módulos de memória RAM e ROM

A partir de módulos de memória com um determinado tamanho é possível implementar blocos de memória com uma dimensão de palavra e de número de palavras múltiplos do módulo original.

Consideremos, em primeiro lugar, a implementação de módulos de memória com o dobro das palavras (ver figura 2).

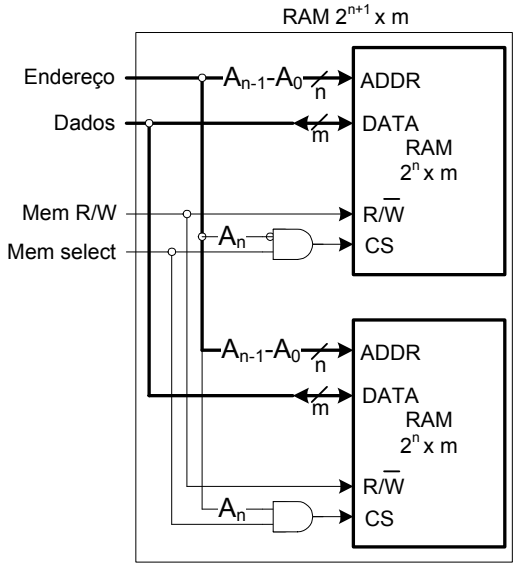


Figura 2 – Implementação de um módulo RAM com o dobro das palavras

Neste caso, cada módulo original fica com metade das palavras de memória. Uma vez que temos o dobro das palavras, é necessária mais uma linha de endereço, A_n. Esta linha de endereço é usada para seleccionar um dos módulos.

A estrutura é facilmente escalável para implementar RAM maiores. Para gerar memórias com dimensão 2^x maior que a dimensão original basta incluir mais x linhas de endereços e depois usar um descodificador $x \times 2^x$ para gerar os sinais de CS de cada um dos módulos.

No caso de se pretender gerar um bloco de memória com o mesmo número de palavras, mas com o dobro do tamanho, são necessários dois módulos de memória sendo que cada um fica com metade da palavra (ver figura 3).

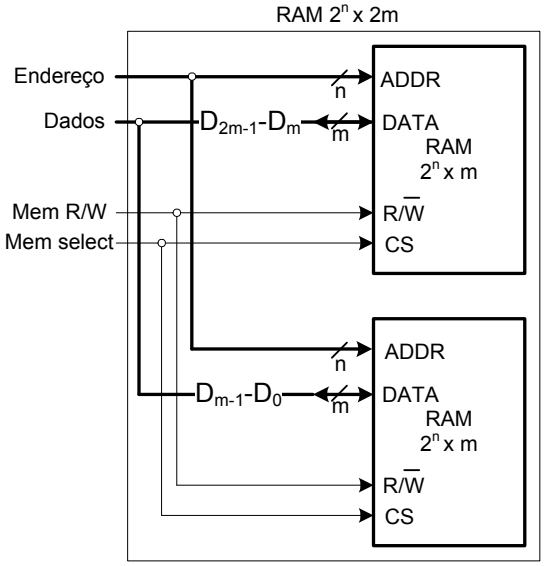


Figura 3 – Implementação de um módulo RAM com palavras com o dobro do tamanho original

Para gerar esta configuração, cada módulo original fica com metade da palavra de memória. As saídas de dados de ambos os módulos são depois concatenados para produzir a palavra de dados completa.

Também neste caso, a estrutura é facilmente escalável para implementar RAM com palavras maiores. Para gerar memórias com palavras de dimensão *x* maior que a dimensão original basta incluir concatenar as saídas de dados de *x* módulos de memória.

As duas configurações podem ser usadas em conjunto para formar um bloco de memória com a dimensão desejada.

1.2 Mapa de Memória

A interligação dos blocos de memória ao processador faz-se, geralmente, através de um conjunto único de buses de endereços, de dados e de controlo. O processador tem uma determinada capacidade de geração de endereços, de acordo com o seu número de bits de endereço, ou seja, o seu espaço de endereçamento. Por exemplo, um processador com 16 bits de endereço tem um espaço de endereçamento de 2^{16}, ou seja, consegue gerar 2^{16} endereços diferentes. No caso de se pretender ligar vários módulos de memória ao processador através do seu bus, é necessário atribuir os endereços disponíveis aos dispositivos de memória, o mesmo é dizer que se tem de mapear as memórias no espaço de endereçamento do processador.

Para nos ajudar a realizar o mapeamento das memórias no espaço de endereçamento, utiliza-se um mapa de memória (ver figura 4).

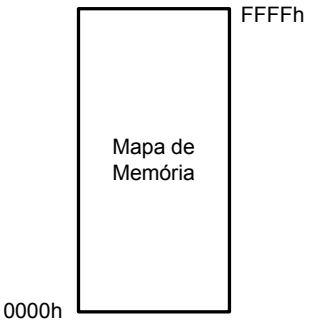


Figura 4 – Mapa de memória de um processador com um espaço de endereçamento de 2^{16} palavras

No exemplo da figura, o espaço de endereçamento vai do endereço 0 até ao FFFFh. Sobre este espaço de endereçamento, vamos agora mapear as memórias necessárias. Por exemplo, considere-se que se pretende incluir uma memória de 16Kb no sistema. O mapeamento pode ser feito em qualquer posição do espaço de endereçamento. No entanto, como veremos mais à frente, determinados mapeamentos são melhores pois simplificam a lógica de descodificação. Neste caso, vamos colocar a memória a partir do endereço 0 (ver figura 5a).

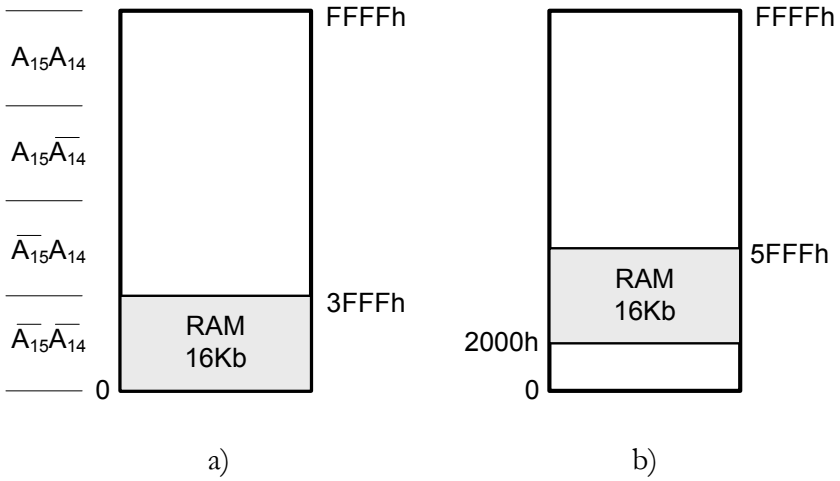


Figura 5 – Mapeamento do bloco de memória de 16Kb no espaço de 64Kb

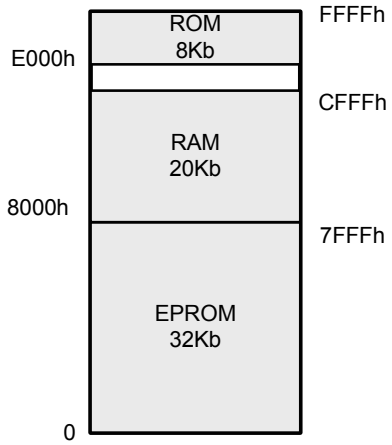
A memória ocupa os endereços que vão do 0 até ao 3FFFh. Sempre que o processador gerar um endereço nesta gama a memória terá de estar activa para um acesso de leitura ou de escrita. Em qualquer um dos outros endereços, a memória terá de estar inactiva. Para garantir este funcionamento controlando a entrada CS da memória. Se for um endereço entre 0 e 3FFFh, activa o CS. Caso contrário, fica inactivo. Basta agora determinar a função lógica do CS respondendo à questão: "Como se distingue os primeiros 16K endereços dos restantes?". Se olharmos com atenção para o binário dos endereços concluímos facilmente que, neste caso, basta que os dois bits de endereço de maior peso estejam ambos a 0. Podemos pensar que o espaço de endereçamento está dividido em quatro e que bastam os dois bits de maior peso do endereço para identificar cada um deles, como se ilustra na figura 5a. Assim, CS = A_{15}'A_{14}' (leia-se X' como sendo a negação de X).

Para percebermos que influência tem a colocação do bloco de memória sobre o processo de descodificação, consideremos o mapeamento ilustrado na figura 5b. Neste exemplo, a RAM começa no endereço 2000h. Para identificar a gama de endereços que vai do 2000h até ao 5FFFh não bastam apenas dois bits do endereço, como no caso anterior. Neste caso, o valor de CS é dado por A_{15}'A_{14}'A_{13} + A_{15}'$A_{14}A_{13}$'.

Para conseguir a descodificação mais simplificada devemos mapear os blocos a começar num endereço múltiplo do seu tamanho. Por exemplo, um bloco de 16Kb deve começar nos endereços 0, 16k, 32k ou 48k.

Exercício – Projecte o mapeamento de uma RAM de 20Kb, uma ROM de 8Kb e uma EPROM de 32Kb num sistema com um espaço de endereçamento único de 64k. Para a implementação do sistema de memória dispões dos seguintes blocos de memória: ROM de 16Kb, RAM de 32Kb e EPROM de 32Kb.

O primeiro passo é mapear os blocos de memória no espaço de endereçamento do processador (ver figura).



Uma vez que os blocos maiores são mais difíceis de colocar, tendo em conta a regra dos múltiplos), será uma boa opção começar pelos blocos maiores, como se fez neste exemplo. No mapeamento proposto todos os blocos começam em múltiplos do seu tamanho por forma a simplificar a descodificação.

Podemos agora passar a passo seguinte que consiste em determinar as expressões lógicas do CS das memórias. Neste ponto, convém verificar quantos módulos de memória física são necessários para cada caso, tendo em conta as especificações do problema. No exemplo, basta um módulo de cada para implementar o sistema de memória pedido. Temos assim as expressões do CS de cada memória:

$CS_{EPROM} = A_{15}'$

$CS_{RAM} = A_{15} (A_{14}' + A_{14} A_{13}' A_{12}') = A_{15} (A_{14}' + A_{13}' A_{12}')$

$CS_{EPROM} = A_{15} A_{14} A_{13}$

A expressão associada ao CS da RAM pode ser mais simplificado se os endereços não ocupados do espaço de endereçamento. Caso admitamos que este espaço pode ser usado na totalidade para realizar o mapeamento actual, então podemos considerá-los como parte do conjunto de endereços do mapeamento da RAM. Neste caso, temos uma RAM de 20Kb a ocupar um espaço total de endereçamento de 24Kb. A vantagem é que a descodificação passa a ser:

$CS_{RAM} = A_{15} (A_{14}' + A_{13}')$

A desvantagem é que deixámos de ter espaço livre para adicionar memória ao sistema.

Até este ponto, apenas se considerou que a arquitectura do computador apenas tinha uma única memória para guardar os dados e as instruções. Em arquitecturas com memória de dados separada da memória de código (arquitectura de Harvard), o espaço de endereçamento será representado utilizando um mapa de memória com duas zonas distintas (ver figura 6).

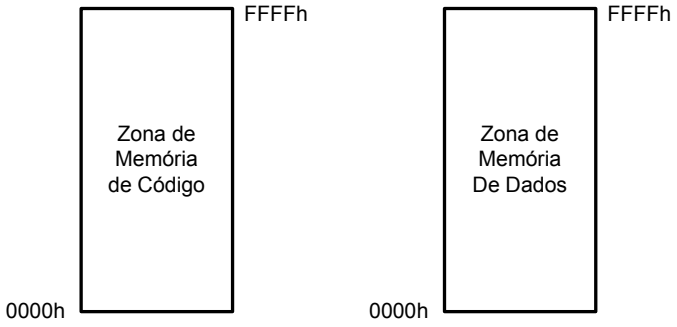


Figura 6 – Mapa de memória com espaços de endereçamento de 2^{16} palavras para dados e código

Neste caso, o mapeamento das memórias no espaço de endereçamento deverá ter em conta o tipo de informação a ser armazenada. No caso de uma memória usada para armazenar unicamente instruções esta deverá ser mapeada na zona de memória de código. No caso de ser usada apenas para dados, deverá ser mapeada na zona de memória de dados. Por fim, caso seja utilizada para armazenar dados e instruções, então deverá ser mapeada em ambas as zonas.

Consideremos, como exemplo, uma arquitectura com capacidade de endereçamento de 64K de memória de dados e de 64K de memória de código. Pretende-se adicionar uma memória RAM de 32K para dados, uma EPROM de 16K para instruções e uma memória RAM de 8K para dados e instruções. Um mapeamento possível é apresentado na figura 7.

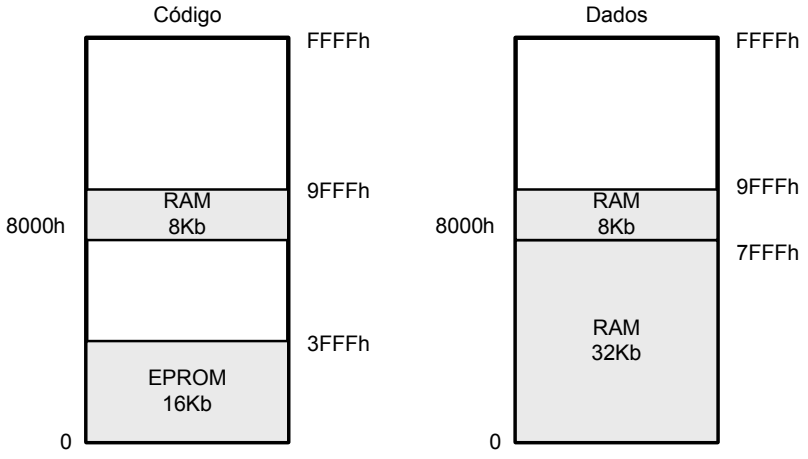


Figura 7 – Mapeamento dos blocos de memória no espaço de endereçamento

O mapeamento da RAM de dados e da EPROM é similar ao realizado no caso de espaço de endereçamento único, mas tendo em atenção o tipo de informação armazenada. A EPROM será mapeada no espaço de código e a RAM no espaço de dados. Quanto à RAM de 8Kb destinada a ambos os tipos de informação, esta ocupa 8Kb em ambos os espaços de endereçamento, uma vez que, nos casos extremos, tanto pode ser usada para armazenar 8Kb de código como 8Kb de dados. Tenha em atenção, que apesar de ocupar um total de 16Kb do espaço de endereçamento, é apenas uma memória de 8Kb. Esta configuração deverá ser tida em conta na ligação dos sinais de endereço e de controlo (deverá ser activada tanto para acesso de dados como para acesso de código).

2 Hierarquia de Memória

A arquitectura de computador estudada assenta no acesso do processador a memória para aceder às instruções ou a dados do programa. Idealmente, o processador deveria ter disponível toda a memória necessária à velocidade mais rápida possível. No entanto, se tivermos em conta que o custo da memória é inversamente proporcional à velocidade de acesso, facilmente concluímos que esta solução não é possível.

A solução encontrada para obter um sistema de memória próximo do ideal consiste em ter diversos tipos de memória com velocidades de acesso e custos diferentes. Se as instruções e os dados mais utilizados se encontrarem nas memórias mais rápidas, temos o problema resolvido. Para tal, considera-se uma hierarquia de memória (ver figura 6).

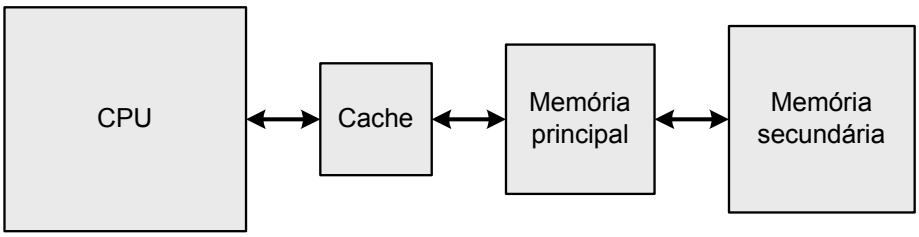


Figura 6 – Hierarquia de memória

A *cache* é o nível de memória mais próximo do CPU. A cache pode ser dividida em mais níveis (nível 1, L1, e nível 2, L2), sendo que um deles é, geralmente, implementado dentro do CPU. Seguem-se a memória principal, na forma DERAM, e a memória secundária que pode ser um disco rígido, um CD-ROM, DVD-ROM, etc.

Nesta configuração, o acesso a uma memória é feito em primeiro lugar à memória cache, o tipo de memória com acesso mais rápido. Se o pretendido não se encontrar nesta memória, é feito um acesso à memória principal. Caso também não se encontre aqui, é feito um acesso à memória secundária.

Alguns estudos indicam que este tipo de hierarquia consegue tempo médios de acesso próximos dos que seriam obtidos caso se usasse apenas memória cache. Tal significa, que a maior parte dos acessos à cache têm sucesso. Tal parece um contra senso uma vez que a cache tem espaço para armazenar uma pequena parte da totalidade da memória principal. De facto, tal não seria possível se não se verificassem os princípios de localidade na maioria dos programas. Este princípio estabelece que os acessos à memória estão bastante correlacionados.

No que se refere ao princípio da localidade, são identificados dois tipos: temporal e espacial. A localidade temporal indica que após o acesso a um determinado endereço de memória, é bastante provável que este mesmo endereço volte a ser acedido num tempo próximo. A localidade espacial indica que após o acesso a um determinado endereço de memória, é bastante provável que se aceda a um endereço próximo deste. Para perceber empiricamente estes princípios basta pensar nos ciclos, em que se identifica facilmente estes dois princípios.

Seguindo os factos indicados por estes dois princípios, para que se consigam bons desempenhos no acesso à memória, basta guardar os dados acedidos em memória cache, bem como os dados vizinhos para que os próximos acessos sejam feitos com grande probabilidade de sucesso à memória cache (verificam-se taxas de sucesso superiores a 95%).

Se por um lado a cache permite um acesso rápido à informação, os dispositivos de memória secundária permitem o armazenamento de uma grande quantidade de informação. Em particular, permitem a implementação de memória virtual que garante o armazenamento em programas que exigem mais memória que a existente em memória principal.

A utilização do conceito de cache e de memória virtual permite que se tenha um sistema com muita memória e com tempos de acesso bastante baixos.