16. SISTEMAS DE MEMÓRIA

Os sistemas baseados em microprocessadores 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 principal é constituída por elementos de memória, volátil ou não (RAMs, ROMs, EEPROMs, Flash, etc.) com baixos tempos de acesso, endereçáveis directamente através do bus de endereços, mas com uma ralação custo/dimensão muito elevado. Esta memória 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., e é caracterizada por grande capacidade de armazenamento de informação. A memória secundária tem um tempo de acesso muito mais lento que a memória principal, não é directamente endereçável através do bus de endereços, mas tem uma relação custo/dimensão muito menor que a memória principal.

16.1 Projecto do Sistema de Memória

No projecto do sistema de memória de um micro-sistema, há numa primeira fase que considerar a existência de várias arquitecturas de micro-sistemas e consequentemente diferentes sistemas de memória. Entre as várias arquitecturas, há a considerar essencialmente, dois tipos: Os sistemas de utilização genérica (Computador Pessoal PC) que dão suporte à execução de aplicações genéricas e ao desenvolvimento de aplicações em várias áreas; e os sistemas embebidos (Telemóveis, POS, etc.), que são projectados com um objectivo específico, ou seja, não pressupõem expansão futura.

No caso do PC, o sistema de memória principal é desenhado por forma a apresentar grande flexibilidade para a inserção de novos elementos de memória, com o objectivo de aumentar a capacidade de memória instalada, ou suportar a inserção de placas gráficas, placas de comunicação, etc.

Nos sistemas embebidos, o sistema de memória é normalmente estável e não inclui memória secundária. É sobre o sistema de memória deste tipo de micro-sistemas que nos iremos debruçar neste documento.

O projecto do sistema de memória principal consiste em identificar as dimensões e o tipo dos módulos de memórias necessários, os endereços a atribuir a cada um deles e os dispositivos de memória a usar na sua implementação.

Seguidamente analisaremos como se implementam blocos de memória de determinada dimensão à custa de dispositivos de memória disponíveis no mercado. Teremos igualmente que analisar como se atribuem os endereços disponíveis no espaço de endereçamento do CPU a cada um dos blocos de memória ligados ao bus do sistema.

16.1.1 Implementação dos Blocos de Memória

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

Como foi visto em capítulos anteriores, um módulo de memória tem como entradas um conjunto de linhas de endereço, de dados e de controlo. As linhas de controlo incluem um sinal de selecção denominado *CS* (*Chip Select*) cujo objectivo, é poder activar e desactivar o dispositivo, no sentido de este poder ser integrado com outros elementos sem interferência entre eles. 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.

A partir de dispositivos de memória com uma determinada dimensão 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 um módulo de memória, como é mostrado na Figura 16-1, que disponibiliza o dobro das palavras do dispositivo de memória que o constitui.

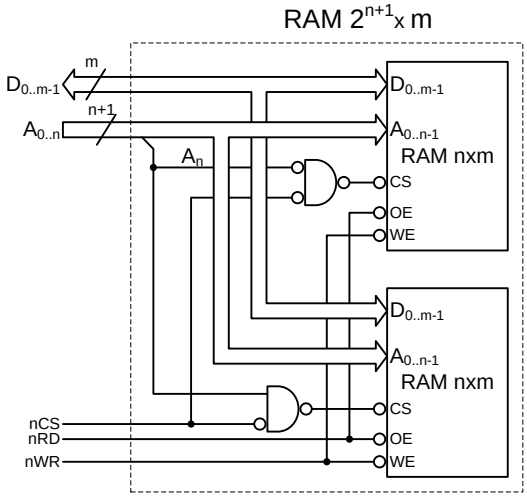


Figura 16-1- Módulo RAM com o dobro das palavras

Neste caso, cada dispositivo fica com metade das palavras do módulo 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 dispositivos.
A estrutura é facilmente escalável para implementar RAM maiores. Para gerar um módulo de memória com dimensão 2^x maior que a dimensão dos dispositivos que o constituem, 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 dispositivos.
No caso de se pretender gerar um bloco de memória com o mesmo número de palavras, mas com o dobro do tamanho da palavra do dispositivo, são necessários dois dispositivos, ficando cada um dos dispositivos de memória com metade da palavra. Os dois dispositivos são concatenados como se mostra na Figura 16-2.

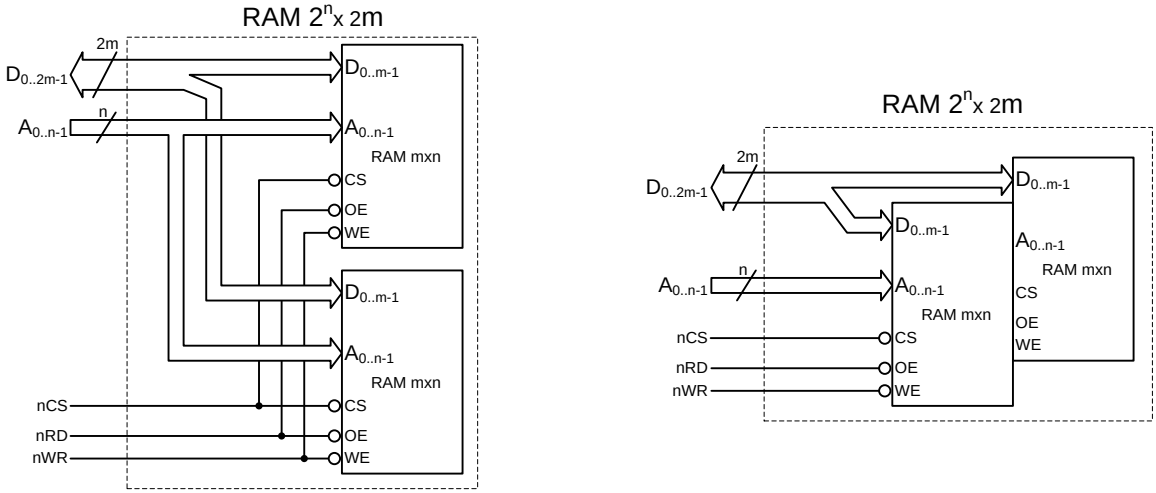


Figura 16-2 - Módulo RAM com o dobro da largura da palavra

Para gerar esta configuração, cada dispositivo fica com metade da palavra de memória. As saídas de dados de ambos os dispositivos são depois concatenados para produzir a palavra de dados completa do módulo.
Também neste caso, a estrutura é facilmente escalável para implementar RAM com palavras maiores. Para gerar módulos de memórias com palavras de dimensão x vezes maior que a dimensão das palavras dos dispositivos utilizados, basta concatenar as saídas de dados de x dispositivos de memória. As duas configurações podem ser usadas em conjunto para formar um bloco de memória com a dimensão desejada.

16.1.2 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 número de bits do seu bus de endereço, constituindo assim o seu espaço de endereçamento. Consideremos como exemplo o PDS16, que dispondo de bus de endereço com 16 bits tem um espaço de endereçamento de 2^{16}, ou seja, consegue gerar 2^{16} endereços diferentes. No caso de se pretenderem ligar vários módulos de memória ao processador através do seu bus, é necessário atribuir o espaço de endereços disponíveis aos vários dispositivos de memória, o mesmo é dizer que se tem de mapear as memórias no espaço de endereçamento do processador. A atribuição de endereços aos módulos de memória podem representar-se graficamente através do que comummente se designa por mapa de memória como mostra a Figura 16-3.

Figura 16-3 - Mapa de memória com um espaço de endereçamento de 2^{16} bytes.

No exemplo da Figura 16-3, o espaço de endereçamento vai do endereço 0 até ao FFFFH. Sobre este espaço de endereçamento, vamos agora mapear os blocos de memória necessários. Após o mapeamento dos vários blocos de memória, seleccionam-se os dispositivos de memória disponíveis no mercado que sejam mais adequados (custo/dimensão/tempo de acesso) para realizar os vários módulos de memória. Será então necessário, para garantir este funcionamento, controlar a entrada CS dos dispositivos de memória através de um circuito denominado lógica de descodificação. Este circuito activa o CS se o valor presente no bus de endereços estiver compreendido dentro do espaço do mapa que lhe foi atribuído, caso contrário, fica inactivo.

16.1.3 Lógica de descodificação de endereços

A lógica de descodificação de endereços e o mapeamento estão muito condicionados aos seguintes factores: tecnologia utilizada na implementação da descodificação (MSI, PLD, etc); características do micro-sistema; processo de produção do equipamento final. Um dos elementos que mais determina a escolha do tipo descodificação, é a característica do micro-sistema, e que coloca ao projectista as seguintes questões:

É necessária a utilização optimizada do mapa de memória?

A aplicação exige contiguidade dos blocos de memória?

O equipamento é para ser fabricado em grande escala?

Comecemos por um exemplo muito simples. Considere que se pretende construir um mapa de memória com um bloco de memória de 16Kb. 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 como mostra a Figura 16-4.

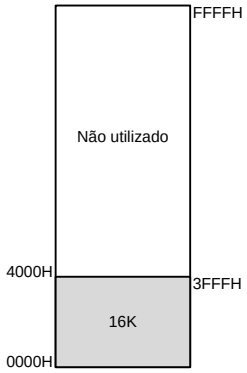


Figura 16-4 – Mapeamento do bloco de memória de 16Kb no espaço de 64Kb

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.
Como se trata de um módulo de memória com 16Kb, são necessários 14 bits (A_0 a A_{13}) para endereçar cada um dos registos do módulo. Este facto implica que a lógica de descodificação só tome como entradas, os bits restantes A_{14} e A_{15}. Se olharmos com atenção para as configurações binárias dos endereços limites, concluímos facilmente que, neste caso, só deve ser gerado CS quando os dois bits de endereço de maior peso estiverem ambos a 0.

```
0000 0000 0000 0000

0011 1111 1111 1111
```

Desta forma podemos pensar que o espaço de endereçamento está dividido em quatro espaços de 16K e que bastam os dois bits de maior peso do endereço para identificar cada um deles, como se ilustra na Figura 16-5. Isto só acontece porque o mapeamento foi feito no início de uma página de 16 K. Assim sendo, $CS = \overline{A}_{15}.\overline{A}_{14}$.

Para percebermos a influência que tem a escolha do endereço inicial do bloco de memória sobre a malha de descodificação, consideremos o mapeamento ilustrado na Figura 16-5.

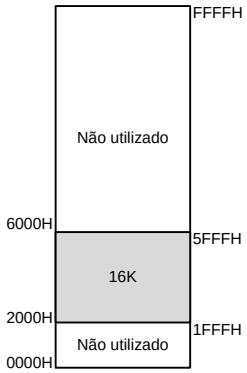


Figura 16-5 – Mapa de memória

Neste exemplo, o bloco de memória 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.

A_{15} A_0

001- ---- ---- ----

001- ---- ---- ----

010- ---- ---- ----

010- ---- ---- ----

Neste caso, o valor de CS é dado por: $!A_{15}$ & $!A_{14}$ & A_{13} # $!A_{15}$ & A_{14} & $!A_{13}$.

Como se pode ver na Figura 16-5, uma vez que os endereços A0-13 estão ligados ao bloco de memória, o registo de endereço 0x0000 do bloco de memória só é acedido quando o CPU endereçar ao endereço 4000H. Se considerarmos que o módulo de memória é RAM, e logo todos os valores válidos nela contidos, foram escritos pelo CPU, tal facto não representa nenhum problema, o que já não é verdade se o módulo de memória for ROM, uma vez que os valores nela contidos estão comprometidos com os endereços. Para conseguir a descodificação mais simplificada, devemos mapear os blocos de memória a começarem num endereço múltiplo do seu tamanho. Por exemplo, um bloco de 16Kb deve começar nos endereços 0000, 4000H, 8000H ou C000H.
Outro método de gerar a lógica de descodificação, e talvez o modo mais natural de pensar a lógica de descodificação de endereços, consiste na utilização de circuitos descodificadores, como mostra a Figura 16-6. O método consiste em seccionar o mapa de memória em módulos simétricos a partir dos bits de maior peso, com a dimensão do maior bloco de memória. Como exemplo, na Figura 16-6 está representada este método para um sistema de memória principal do PDS16, com 16Kb ROM de boot (programa de arranque) 4Kb de flash e 2Kb de RAM, que utiliza dispositivos ROMs de 8Kx16, flash de 2Kx8 e RAMs de 1Kx8.

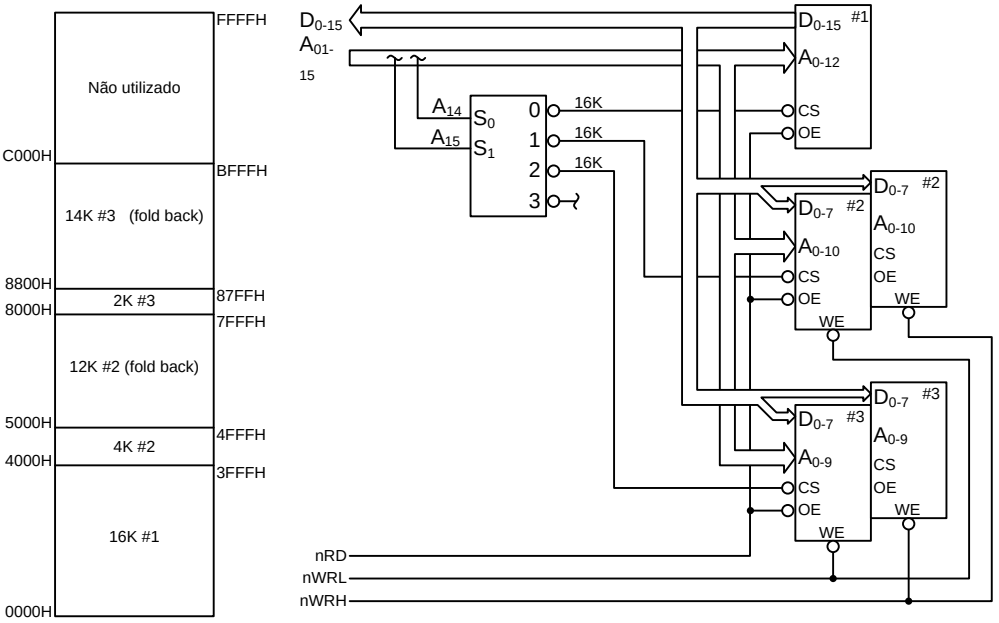


Figura 16-6 – Mapa de memória e diagrama de blocos

Esta descodificação embora simples, apresenta como inconveniente um espaço de memória descontínuo, e desaproveitamento do mapa de memória por geração de sombras (*fold back*), ou seja, um qualquer registo é endereçável em diferentes localizações. Este comportamento poderá constituir um inconveniente para a aplicação. Se pretendermos utilizar esta técnica, garantindo continuidade e utilização optimizada do mapa de memória sem geração de situações de *fold back*, poderemos aplicar o método sucessivamente de forma arborescente, até atingirmos páginas com a dimensão do menor dispositivo de memória, ordenando dos de dimensão maior para menor, como mostra a Figura 16-7.

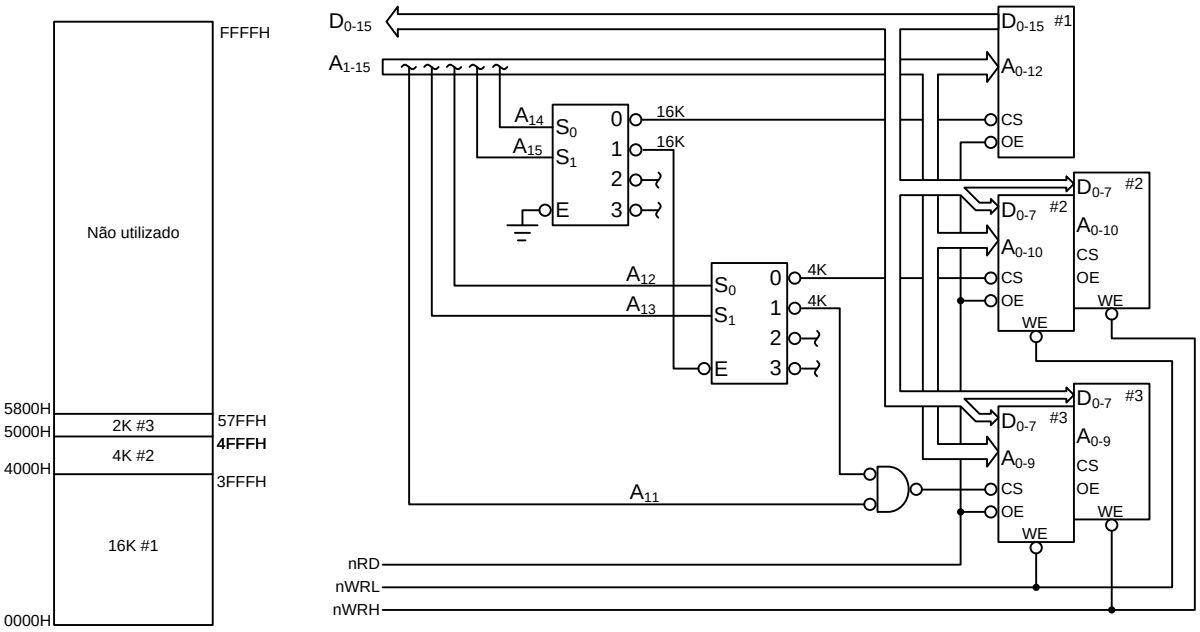


Figura 16-7 - Mapa de memória e diagrama de blocos

Esta forma de descodificação baseada na concatenação de circuitos descodificadores, apresenta como inconveniente o tempo de descodificação provocado pela propagação através dos sucessivos descodificadores, aumentando os tempos de leitura e escrita, razão pela qual só deverá ser utilizada se não forem utilizados muitos níveis de descodificação e o tempo de acesso não for crítico.

16.1.4 *Linear Select*

Outro modo descodificação é designado por selecção linear (*Linear Select*). O modo *linear select* utiliza um bit de endereço por cada periférico, diminuindo assim a lógica de descodificação, mas leva às seguintes circunstâncias:

- Ocupa um espaço no mapa de memória que é superior à dimensão do dispositivo (*fold-back*), o que pode tornar a solução inviável;
- Permite que o CPU seleccione vários dispositivos de memória em simultâneo. Razão pela qual não deve ser utilizado durante o desenvolvimento da aplicação, pois um erro de endereçamento poderá destruir os dispositivos de memória, se forem lidos em simultâneo.

Tomemos como exemplo a descodificação dos seguintes módulos de memória: 8K ROM de boot (programa de arranque) 4K de flash e 2K de RAM utilizando *Linear Select*, como mostra a Figura 16-8.

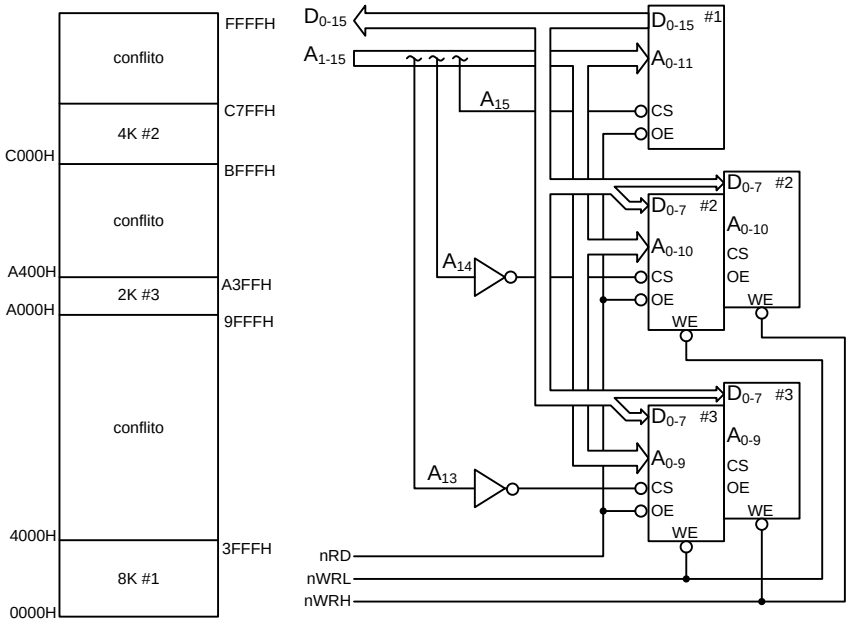


Figura 16-8 - Mapa de memória e diagrama de blocos

Quando se utiliza tecnologia PLD (ROM ou PLA) na síntese da lógica de descodificação, a geração dos vários CS reduz-se a uma união de produtos. Tomemos como exemplo o projecto do mapeamento do sistema de memória principal do PDS16, com os seguintes módulos de memória: RAM de 20Kb e RAM não volátil de 8Kb para dados, Flash de 32Kb e EPROM de 4Kb para código. Para a implementação do sistema de memória dispõe dos seguintes dispositivos de memória: RAMs não volátil de 4Kx8, RAMs de 16Kx8, Flash 16Kx16 e EPROMs de 8Kx8. Dispondo destes dispositivos, podemos concluir que são necessárias 2 EPROMs, 1 Flash, 4 NVRAMs e 2 RAMs. A EPROM contém o programa de arranque (boot), pelo que tem que

ocupar obrigatoriamente os primeiros endereços de memória. O mapeamento deve garantir continuidade dos módulos destinados a código e a dados.
O primeiro passo é mapear os blocos de memória no espaço de endereçamento do processador como mostra a Figura 16-9.

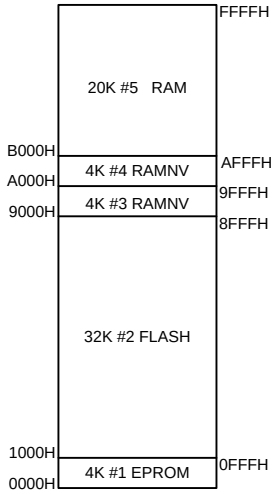


Figura 16-9 – Mapa de memória

Podemos agora passar ao passo seguinte que consiste em determinar as expressões lógicas do CS das memórias. Neste ponto, convém verificar quantos dispositivos de memória são necessários para cada caso, tendo em conta as especificações do problema. Temos assim as expressões do CS de cada memória:

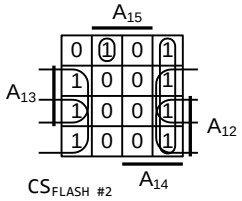
$CS_{EPROM\#1} = \ !A15 \ \& \ !A14 \ \& \ !A13 \ \& \ !A12$

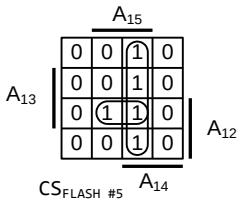
$CS_{FLASH\ \#2} = \ !A15 \ \& \ A14 \ \# \ !A15 \ \& \ A12 \ \# \ !A15 \ \& \ A13 \ \# \ A15 \ \& \ !A14 \ \& \ !A13 \ \& \ !A12$

$CS_{RAMNV\ \#3} = \ A15 \ \& \ !A14 \ \& \ !A13 \ \& \ A12$

$CS_{RAMNV\ \#4} = \ A15 \ \& \ !A14 \ \& \ A13 \ \& \ !A12$

$CS_{RAM\ \#5} = \ A15 \ \& \ A13 \ \& \ A12 \ \# \ A15 \ \& \ A14$





Como se pode observar pelas expressões obtidas, teríamos os 4 bits de maior peso A15, A14, A13 e A12 como entradas do circuito gerador dos vários CS. Isto reflecte a seguinte lógica. O total do mapa que é de 64Kb, foi dividido em 16 blocos de 4Kb, por corresponder à dimensão do dispositivo de menor dimensão, e todos os restantes dispositivos terem dimensão múltipla de 4, dividindo os 64K totais por 4K obtemos 16 módulos, para gerar 16 diferentes configurações são necessários 4 bits (2^4=16). A utilização desta tecnologia (ROM ou PLD), tem impacto no fabrico dos equipamentos, uma vez que implica o manuseamento dos PLD para gravação, ou quando em grande escala a aquisição já gravada, que condiciona a correcção de possíveis erros de concepção. Actualmente, alguns dispositivos PLD, suportam gravação *inboard*, o que minimiza em parte este problema.

A implementação deste mesmo mapeamento utilizando circuitos descodificadores MSI, seria bastante complexo. No entanto uma solução, poderia seguir a mesma lógica anteriormente descrita, ou seja, dividir o espaço total com um descodificador de 4x16, e realizar uniões das saídas do descodificador, para gerar cada um dos CS. As uniões seriam muito simplificadas se as saídas do descodificador fossem OPEN DRAIN, e nesse caso poderíamos realizar com lógica *wired* OR, como mostra a Figura 16-10.

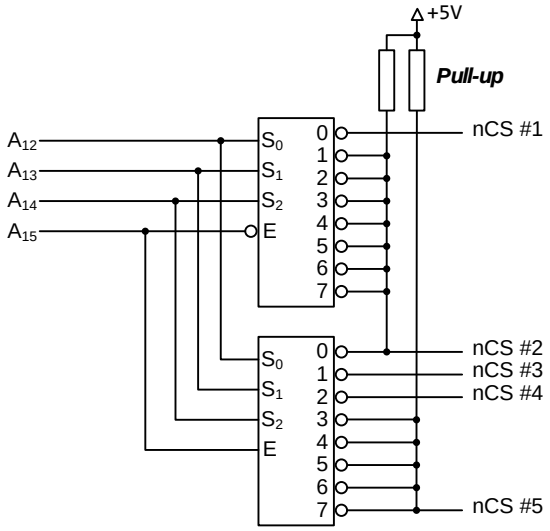


Figura 16-10 – Diagrama de blocos – descodificação de memória

16.2 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 directamente 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 como mostra a Figura 16-11.

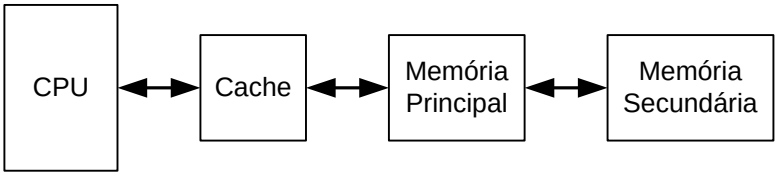


Figura 16-11 – Hierarquia de memória

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.

16.3 Exercícios

[1] Considere um sistema baseada no microprocessador PDS16 com uma memória cuja descodificação é ilustrada na Figura 16-12. Pretende-se adicionar ao sistema 12K∗16 de memória RAM de boot, dois portos de saída de 8 bits e um porto de entrada de 16bit.

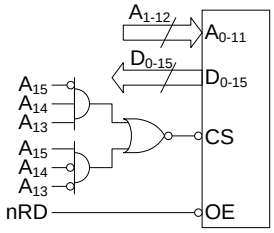


Figura 16-12

a) Desenhe o mapa de memória que fica presente no sistema.

b) Utilizando RAM de 16K*8, desenhe a descodificação dos endereços da memória de acordo com o mapa proposto.

c) Desenhe a descodificação dos portos de acordo com o mapa proposto (Nota: optimize a lógica de descodificação).

[2] Considere um sistema baseada no microprocessador PDS16 com uma memória cuja descodificação é ilustrada na Figura 16-13. Pretende-se adicionar ao sistema 10K∗16 de memória RAM, dois portos de entrada de 8 *bits* com acesso exclusivo ao byte e um porto de saída de 16 *bits* com acesso só a *word*.

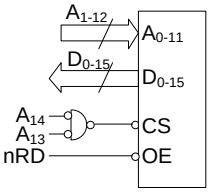


Figura 16-13

a) Desenhe o mapa de memória que fica presente no sistema.

b) Utilizando RAM de 16K*8, desenhe a descodificação dos endereços da memória de acordo com o mapa proposto.

c) Desenhe a descodificação dos portos de acordo com o mapa proposto (Nota: optimize a lógica de descodificação). Dispõe de buffers tristate de 8 bits e registadores de 16 bits.

[3] Considere um sistema baseado no microprocessador PDS16 com as ligações indicadas na Figura 16-14. Pretende-se adicionar ao sistema 32Kbyte de RAM constituída por dois blocos de memória, suportando acesso a 8 e 16 bits. O bloco #1 ocupa os endereços de 0x2000 a 0x7FFF e o bloco #2 de 0xA000 a 0xBFFFh. Para a implementação dispõe de circuitos de memória de 8K*8.

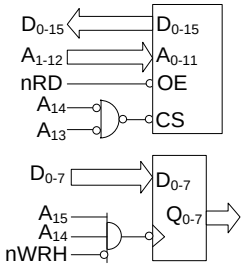


Figura 16-14

[4] Considere um sistema baseado no microprocessador PDS16 ao qual se pretende adicionar 14K*16 RAM com endereço base em 0x8000. Para a implementação dispõe de dispositivos RAM de 8K*8.

[5] Considere que pretende construir um sistema de computação utilizando um processador PDS16. O sistema deverá ter a seguinte estrutura de memória: 4K*16 de ROM, 1K*16 de I/O e o espaço restante preenchido com memória RAM. Todos os dispositivos deverão suportando acesso ao byte e à *word*. Após reset, deverá apresentar no endereço 0x0000 a memória ROM, para conter o código de arranque (*Boot*) e o BIOS (*Basic Input Output System*). Como o espaço para variáveis de acesso directo são os primeiros 128 endereços de memória, após o arranque do sistema, os endereços iniciais deverão ser preenchidos com memória RAM sendo a ROM deslocada para os últimos endereços mantendo-se assim o BIOS disponível às aplicações. Os periféricos de I/O serão alocados entre a RAM e a ROM. Os dispositivos de memória disponíveis para a implementação do sistema são: ROMs de 4K*8 e SRAMs de 32K*8.

16.3.1 Soluções

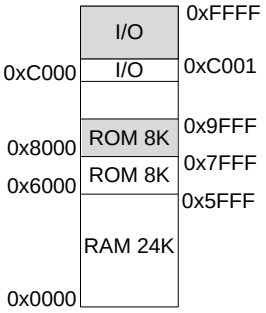
[1] a)

ROM 4K∗16 de 0x6000 a 0x7FFF e 0x8000 a 0x9FFF (4K∗16 *fold back*)

RAM 12K∗16 de 0x0000 a 0x5FFF

Portos de saída 0xC000 a 0xFFFF (16K∗8 *fold back*)

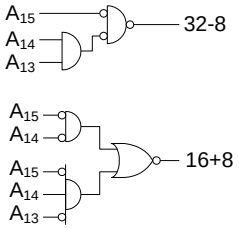
Porto de entrada 0xC000 a 0x_FFFF (8K∗16 *fold back*)

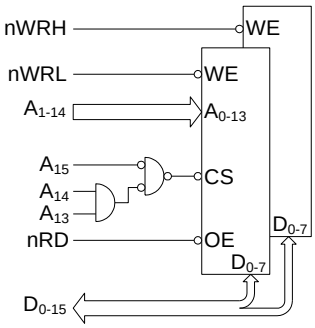


[1] b)

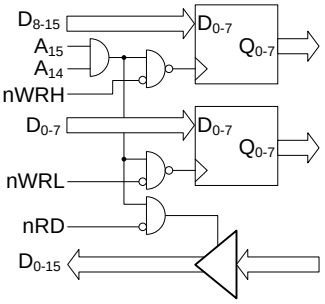
RAM 12K∗16=24K∗8 = 32K∗8 - 8K∗8

RAM 12K∗16=24K∗8 = 16K∗8 + 8K∗8





[1] c)



[2] a)

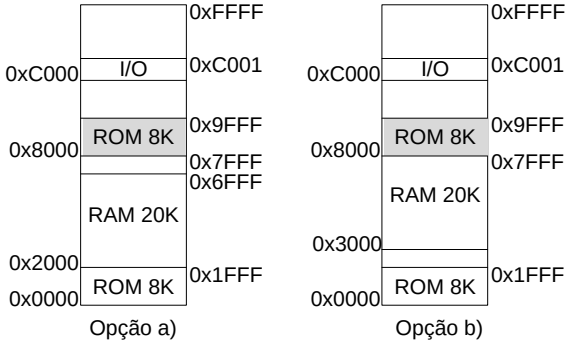
ROM 4K∗16 de 0x0000 a 0x1FFF e 0x08000 a 0x9FFF (4K∗16 *fold back*)

Opção a) RAM 10K∗16 de 0x2000 a 0x5FFF

Opção b) RAM 10K∗16 de 0x3000 a 0x7FFF

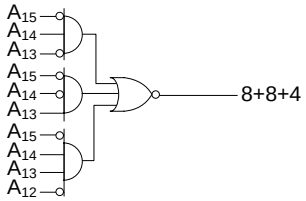
Portos de saída 0xC000 (16K∗8 *fold back*)

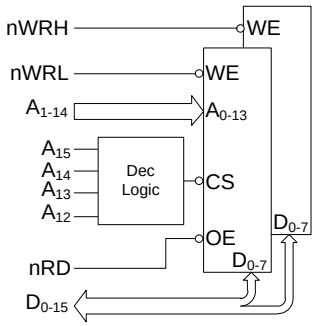
Porto de entrada 0xC000 (16K∗8 *fold back*)



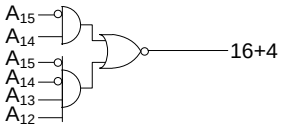
[2] b)

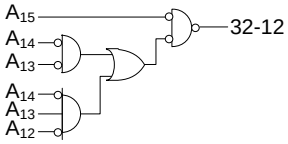
Opção a) RAM=16K+4K=20K



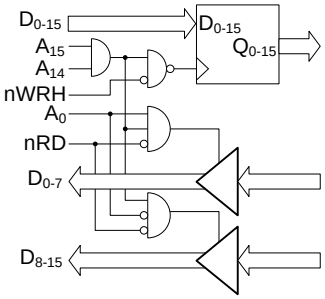


Opção b) RAM=16K+4K=20K RAM=32K-(8K+4K)=20K





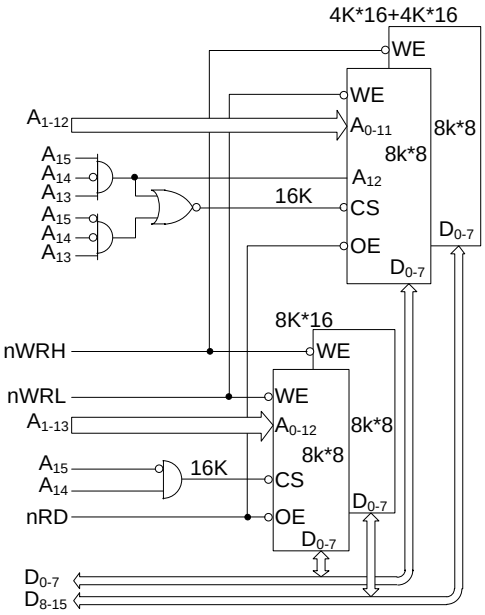
[2] c)

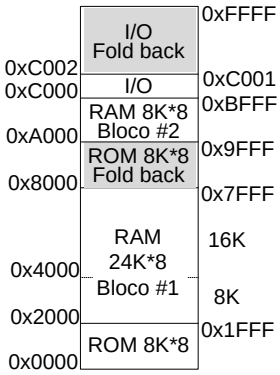


[3]

Bloco #1 12K*16=24K*8=8K+16K

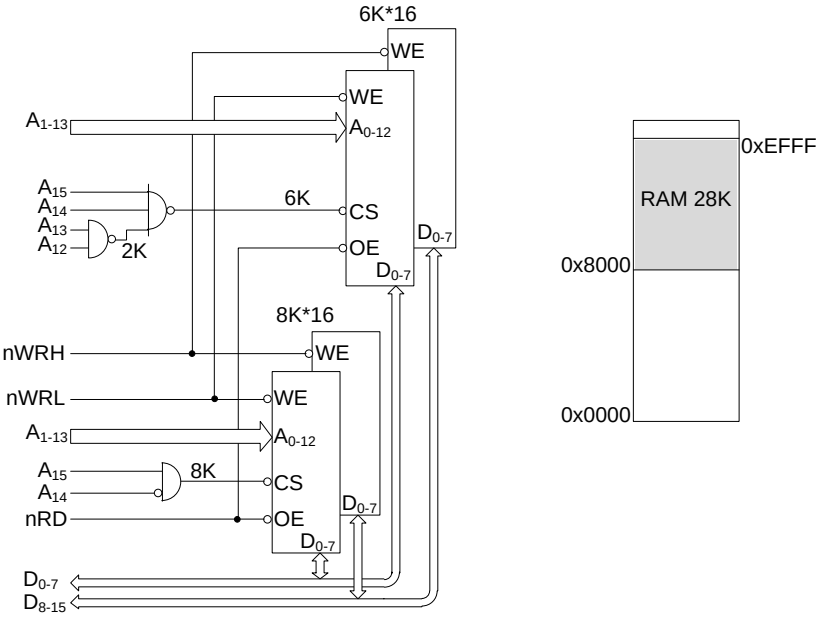
Bloco #2 4K*16=8K*8





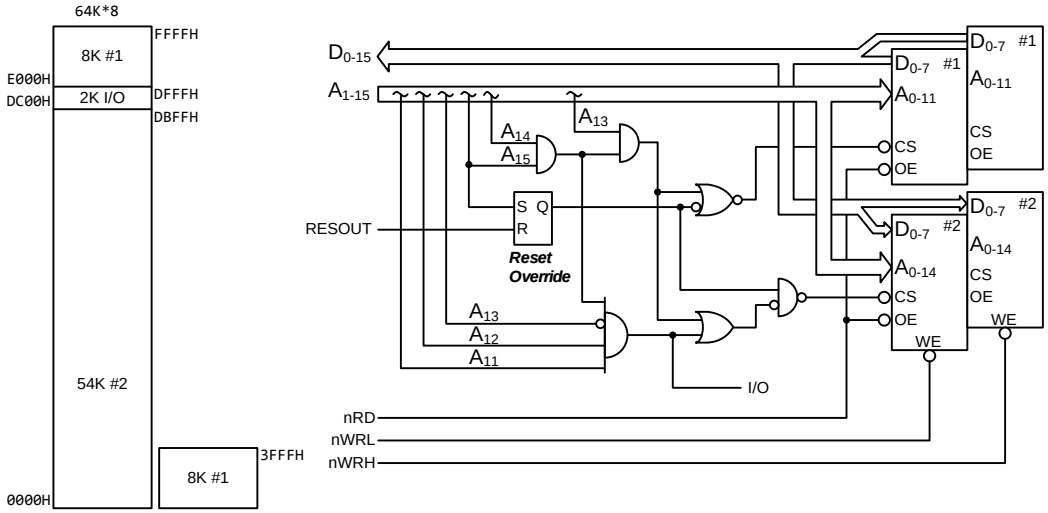
[4]

14K*16=8K+(8K-2K) => 28K*8=16K+(16K-4K)



[5]

RAM= 64K – (8K+2K)




```
        .section bios
        .org    0xe000
        ld      r7,[r7,#0]      ; long jump to the main
        .word   main
main:   ...
        ...
```