

Laboratório de Arquitetura de Computadores

Guião 9

Descodificação de endereços

1 Objetivos

Com este guião pretende-se exercitar e demonstrar a descodificação de endereços.

2 O circuito de simulação

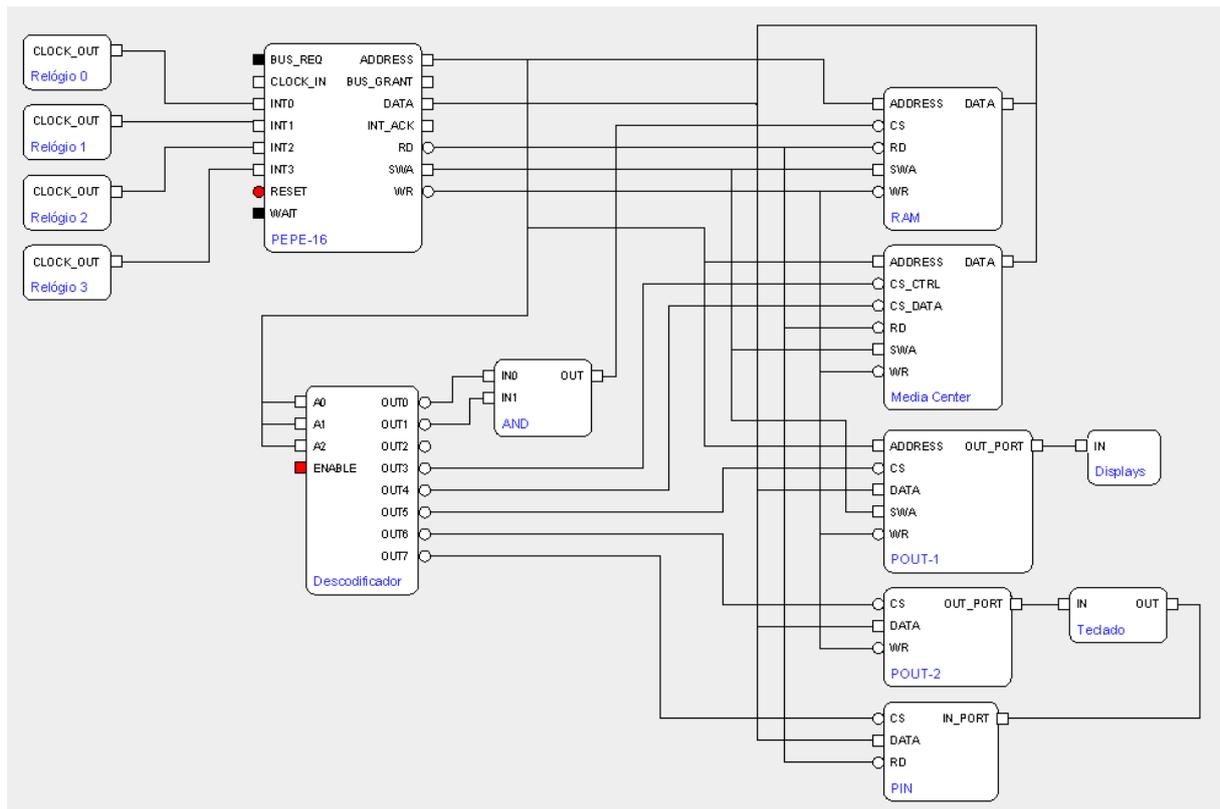
Os guiões de laboratório 6, 7 e 8 utilizaram o circuito da figura da página seguinte, constituído por:

- Processador PEPE-16;
- Uma memória (RAM);
- Um ecrã (MediaCenter);
- Dois periféricos de saída de 8 bits (POUT-1, ligado a dois displays de 7 segmentos, e POUT-2, ligado às linhas do teclado);
- Um periférico de entrada de 8 bits (PIN, ligado às colunas do teclado);
- Quatro relógios de tempo real, para temporizações.

Neste guião utiliza-se um circuito quase igual, contido no ficheiro **lab9.cir**. A diferença está no periférico POUT-1, que agora é de 16 bits (para ilustrar periféricos de 16 bits) e está ligado a 4 displays hexadecimais. Por ser de 16 bits ocupa agora 2 endereços, pelo que o periférico precisa agora de um pino **ADDRESS** e de outro **SWA** (porque o periférico suporta endereçamento de byte).

O mapa de endereços (em que os dispositivos podem ser acedidos pelo PEPE-16) é o seguinte (já vem do guião de laboratório 4, com a diferença de que o POUT-1 ocupa agora dois endereços):

Dispositivo	Endereços
RAM	0000H a 3FFFH
MediaCenter (acesso aos comandos)	6000H a 6063H (ver guião 4)
MediaCenter (acesso à sua memória)	8000H a 8FFFH
POUT-1 (periférico de saída de 16 bits)	0A000H a 0A001H
POUT-2 (periférico de saída de 8 bits)	0C000H
PIN (periférico de entrada de 8 bits)	0E000H



3 Acesso em byte e em word

O guião de laboratório 7 ilustrou duas soluções para o problema de implementar várias atividades concorrentes, aparentemente simultâneas:

- Rotinas cooperativas. O programa principal contém um ciclo infinito, que invoca todas as rotinas que implementam estas atividades;
- Processos cooperativos. O programa principal cria vários processos, que depois correm de forma independente e autónoma.

Os programas mais elaborados do guião 7 que ilustram cada uma destas soluções estão contidos nos seguintes ficheiros (aqui incluídos para facilidade de referência):

- Rotinas cooperativas. **lab7-cooperativas-teclado-OK.asm** (este programa anima barras no ecrã e aumenta o valor dos displays de forma contínua, exceto se se mantiver carregada uma tecla da última linha, em que diminui os displays);
- Processos cooperativos. **lab7-processos-quatro-bonecos-displays-teclado.asm** (este programa anima bonecos no ecrã e aumenta ou diminui o valor dos displays quando se carrega no teclado na tecla **C** ou **D**, respetivamente);

Se quiser, releia o guião de laboratório 7 e lembre a funcionalidade destes programas. Escolha um deles (para o efeito tanto faz), e vamos agora usá-lo na perspetiva da descodificação de endereços.

Carregue o programa que escolheu, carregando em **Load source** (📁) ou *drag & drop*.

Abra o ecrã, os displays, o teclado e os painéis de todos os relógios.

Execute o programa, carregando no botão **Start** (▶) do PEPE-16.

Verifique que tudo funciona, nomeadamente os displays, que vão aumentando ou diminuindo o seu valor, consoante se carregue na tecla **C** ou **D**, respetivamente.

No entanto, há uma diferença: agora há 4 displays, e os que contam são os da esquerda! Deveriam ser os da direita, correspondentes às unidades e dezenas.

Nos guiões de laboratório anteriores só havia dois displays, e eram esses que variavam.

Tal deve-se ao facto de o PEPE-16 ser um processador *Big-Endian*, e como tal o endereço 0A000H, usado para atualizar os displays com **MOVB** em ambos os programas, refere-se ao byte de maior peso do registo escrito no periférico que liga aos displays, ou seja, aos dois displays da esquerda.

Há duas soluções para corrigir esta situação:

1. Redefinir a constante **DISPLAYS** (definida nos programas com **EQU 0A000H**) para **0A001H**. Mas isso não permite alterar os dois displays da esquerda, pois o **MOVB** só escreve um byte;
2. Usar **MOV** em vez de **MOVB**, uma vez que temos 4 displays e o periférico **POUT-1** é agora de 16 bits. A constante **DISPLAYS** continua definida com **0A000H**. Qualquer escrita nos displays altera agora os 4 displays.

A melhor solução é a 2. Com periféricos de 16 bits devemos usar **MOV** e não **MOVB**, a menos que haja uma necessidade específica de aceder apenas a um dos dois bytes do periférico.

Pode experimentar a solução 2 substituindo a instrução **MOVB [R0] , R2** pela instrução de acesso em 16 bits **MOV [R0] , R2**, em qualquer ou em ambos os programas:

- Rotinas cooperativas. **lab7-cooperativas-teclado-OK.asm** (na rotina **anima_displays**);
- Processos cooperativos. **lab7-processos-quatro-bonecos-displays-teclado.asm** (no programa principal, a seguir ao label **atualiza_display**);

Com esta alteração, pode verificar que os 4 displays já contam nos displays do lado direito. Isto ilustra também a diferença de comportamento destas duas instruções, **MOVB** e **MOV**.

NOTA – Com **MOV**, já é possível usar a constante **DISPLAYS** diretamente na instrução:
MOV [DISPLAYS] , R2.

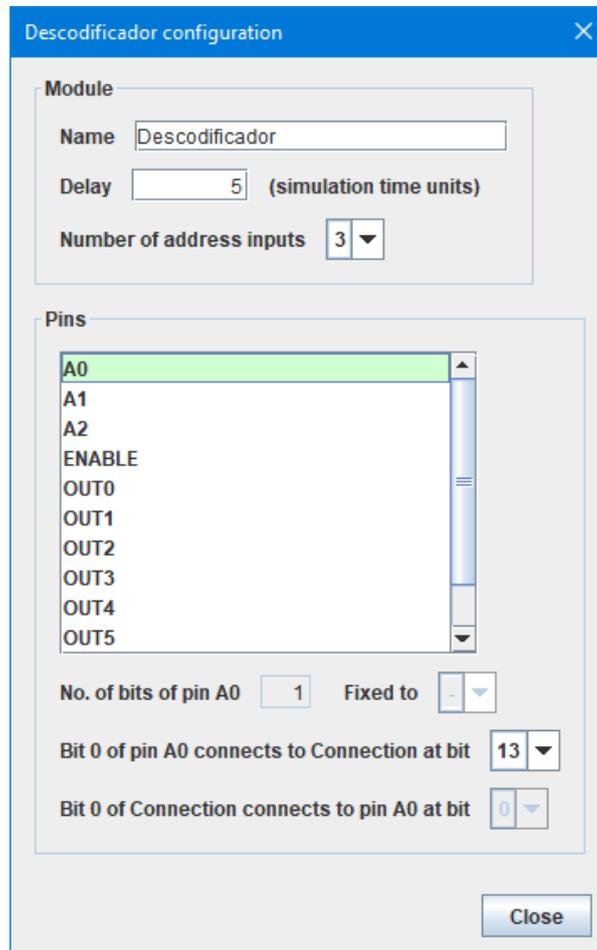
4 Verificação do mapa de endereços

Com o editor de texto, verifique no programa que escolheu:

- Quais são as linhas de programa em que se definem os endereços em que os dispositivos estão acessíveis;
- Quais as constantes usadas para os definir;
- Que os endereços definidos são os do mapa acima.

Verifiquemos agora no circuito que o mapa de endereços implementado é o indicado acima.

Com o simulador em modo “Design”, faça duplo clique no decodificador. Deverá abrir-se a janela ilustrada pela figura seguinte.



O pino **A0** que se observa é o de menor peso dos três bits que especificam qual das saídas do decodificador está ativa em cada instante. Estes três pinos ligam a bits do bus de endereços do PEPE-16 (ver circuito na secção 2). Note-se que as designações **A0** a **A2** do decodificador são internas e não ligam aos bits 0 a 2 do bus de endereços do PEPE-16.

NOTA – O decodificador tem pinos separados para cada bit de decodificação, mas o PEPE-16 tem os seus 16 bits do bus de endereços num só pino (**ADDRESS**).

A linha “**Bit 0 of pin A0 connects to Connection at bit 13**” indica qual o bit do bus de endereços do PEPE-16 (pino **ADDRESS**) a que o **A0** do decodificador liga (neste caso o bit 13). Faça clique no **A1** e no **A2** do Descodificador e verifique que o pino **A0** do decodificador liga ao bit 13 do bus de endereços, o pino **A1** ao bit 14 e o pino **A2** ao bit 15:

Pino do decodificador	Bit do bus de endereços do PEPE-16
A0	13
A1	14
A2	15

A fatia de endereços em que cada saída do decodificador (que liga ao *Chip Select* de um dispositivo) se mantém ativa é de 2000H (ou 8 Ki) endereços, o que corresponde aos 13 bits de menor peso do bus de endereços do PEPE-16 (bits 0 a 12) que podem ligar a cada dispositivo ($2^{13} = 2000H$, ou 8 Ki), tal como indicado na tabela seguinte:

Saída do decodificador	Endereço inicial	Endereço final
OUT0	0000H	1FFFH
OUT1	2000H	3FFFH
OUT2	4000H	5FFFH
OUT3	6000H	7FFFH
OUT4	8000H	9FFFH
OUT5	A000H	BFFFH
OUT6	C000H	DFFFH
OUT7	E000H	FFFFH

A generalidade dos periféricos requer apenas uma fração de cada fatia de endereços. Por exemplo, o MediaCenter não precisa de todos na parte dos comandos.

Menos do que a fatia não tem problema, mas no caso da RAM quer-se uma capacidade que é o dobro (4000H) do tamanho da fatia, e é por isso que se usa um AND das primeiras duas saídas. Como estas são ativas a 0, basta uma delas estar ativa para ativar o *Chip Select* da RAM (também ativo a 0).

O mapa real de endereços é então (ver de novo o circuito na secção 2):

Saída do decodificador	Dispositivo	Endereço inicial	Endereço final
OUT0	RAM	0000H	1FFFH
OUT1	RAM (devido ao AND)	2000H	3FFFH
OUT2	Não usado	4000H	5FFFH
OUT3	MediaCenter (comandos)	6000H	6069H (ver guião 4)
OUT4	MediaCenter (ecrã)	8000H	8FFFH
OUT5	POUT-1 (saída, 16 bits)	A000H	A001H
OUT6	POUT-2 (saída, 8 bits)	C000H	C000H
OUT7	PIN (entrada, 8 bits)	E000H	E000H

O POUT-1 é um periférico de saída de 16 bits, pelo que gasta dois endereços. O POUT-2 e o PIN são periféricos de apenas 8 bits, pelo que só gastam um endereço.

Quando um dispositivo requer menos endereços do que a fatia do decodificador, os endereços em que o dispositivo está ativo repetem-se (espelhos).

Por exemplo, o periférico POUT-1 está também disponível de A002H a A003H, de A004H a A005H, etc. De 2 em 2, repete-se, ao longo de toda a gama de endereços da respetiva fatia.

Como os periféricos POUT-2 e PIN são periféricos de 8 bits, só usam os endereços pares, ou seja, só usam metade do barramento de dados. Por isso, os espelhos são sempre em endereços pares. Assim, o POUT-2 está acessível nos endereços C000H, C002H, C004H, etc., enquanto o PIN está acessível nos endereços E000H, E002H, E004H, etc.

5 Alteração do mapa de endereços

Pretende-se agora alterar o mapa de endereços do circuito e verificar que basta alterar também os endereços no software em concordância, para que tudo continue a funcionar.

Carregue o circuito contido no ficheiro **lab9-novo-mapa-endereços.cir** (por **Load** (📁) ou por *drag & drop*). Este circuito é quase igual ao **lab9.cir**, com a diferença de que o mapa de endereços é agora o seguinte:

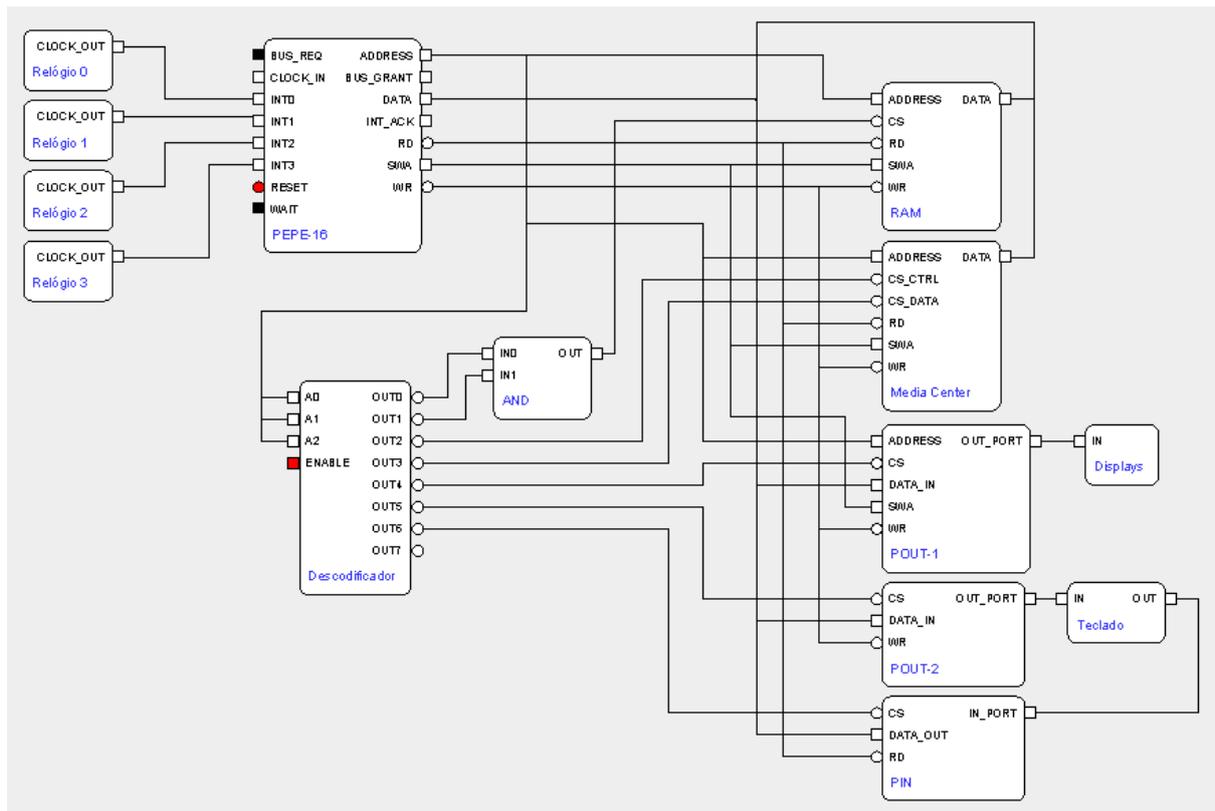
Dispositivo	Endereços
RAM	0000H a 1FFFH
MediaCenter (acesso aos comandos)	2000H a 2069H
MediaCenter (acesso à sua memória)	3000H a 3FFFH
POUT-1 (porto de saída de 16 bits)	4000 a 4001H
POUT-2 (porto de saída de 8 bits)	5000H
PIN (porto de entrada de 8 bits)	6000H

Ou seja, a fatia de endereços em que cada saída do decodificador (que liga ao *Chip Select* de um dispositivo) se mantém ativa é agora de 1000H (ou 4 Ki) endereços, o que corresponde aos 12 bits de menor peso do bus de endereços do PEPE-16 (0 a 11) que podem ligar a cada dispositivo ($2^{12} = 1000H$, ou 4 Ki). Isto é metade do que era no circuito **lab9.cir**.

As alterações efetuadas no circuito foram as seguintes:

- Os pinos **A0** a **A2** do decodificador passaram a ligar aos bits 12 a 14 do bus de endereços do PEPE-16;
- A capacidade de endereçamento da RAM foi reduzida para 13 bits (2^{13} bytes);
- As saídas do decodificador que ligam aos dispositivos foram também alteradas, de acordo com o mapa pretendido.

O circuito ficou agora como ilustrado pela figura seguinte, onde apenas são aparentes as alterações das saídas do decodificador.



Agora é preciso alterar também as definições no programa, para ficar consistente com o novo mapa de endereços. Pretende-se que faça o seguinte:

- Faça uma cópia do ficheiro *assembly* que escolheu atrás;
- Nessa cópia, altere os endereços necessários para que o programa bata certo com este novo mapa de endereços. Deverá apenas alterar as constantes que definem os endereços dos periféricos;
- Carregue no PEPE-16 o ficheiro com o programa alterado, com **Load source** (🟢) ou *drag & drop* na janela de programa do PEPE-16;
- Abra o ecrã, os displays, o teclado e os painéis de todos os relógios;
- Execute o programa, carregando no botão **Start** (▶) do PEPE-16;
- Verifique que tudo funciona como dantes, e que os displays vão aumentando ou diminuindo o seu valor, consoante se carregue na tecla **C** ou **D**, respetivamente;
- Termine o programa, carregando no botão **Stop** (⏹) do PEPE-16.

Conclusão: é possível alterar o mapa de endereços do hardware e manter a funcionalidade do software, desde que se altere no programa, de forma correspondente, as constantes que definem os endereços.