

Laboratório de Arquitetura de Computadores

Guião 3

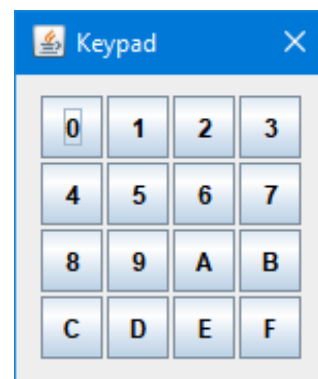
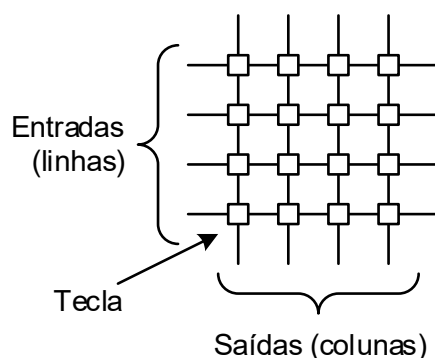
Interação do processador com memória e periféricos

1 Objetivos

Com este guião pretende-se que o leitor pratique a utilização do microprocessador PEPE-16 (Processador Especial Para Ensino com 16 bits) para acesso a periféricos (um teclado e dois displays de 7 segmentos).

2 Funcionamento de um teclado

O teclado de 16 teclas a usar neste guião está organizado internamente como uma matriz de 4 linhas por 4 colunas. Neste caso as teclas disponíveis são os números 0 a 9 e ainda as letras de A a F. O que se pretende fazer do ponto de vista do microprocessador é saber qual a tecla que foi premida (com o cursor em cima da tecla e fazendo clique, simulando o carregar na tecla com um dedo).



O teclado não indica diretamente qual a tecla que foi premida. Para descobrir qual foi, o programa tem de testar sucessivamente (varrer) as várias linhas do teclado, para ver se alguma tecla foi premida.

Quando o utilizador carrega numa tecla, é feito um curto-circuito entre a linha e a coluna que definem essa tecla, fazendo com que o bit introduzido na linha (seja 0, seja 1) apareça na coluna (saída). Coluna a que nenhuma linha ligue vale 0 no bit de saída.

NOTA – Como este teclado não é físico, o premir fisicamente uma tecla é substituído pelo clique no ícon dessa tecla.

Para varrer o teclado e testar todas as teclas, aplica-se sucessivamente um 1 a apenas uma das linhas. Ou seja, colocam-se os valores “0001”, “0010”, “0100” e “1000” (em formato binário) nas entradas do teclado (linhas), lendo as saídas do teclado (colunas) em cada caso.

NOTA – A tecla “0” (topo esquerdo) corresponde ao bit de menor peso dos 4 bits quer das linhas, quer das colunas. Por exemplo, aplicar 0001b nas linhas corresponde a injetar 1 na primeira linha, em que a tecla “0” está, e 0 nas restantes linhas.

Com o valor 0001b aplicado nas linhas, se se obtiver nas colunas o valor 0001b, quer dizer que a tecla que foi premida foi a tecla “0”. Se o valor for 0010b, quer dizer que foi a tecla “1” que foi premida. Se for 0100b ou 1000b, a tecla premida terá sido “2” ou “3”, respetivamente.

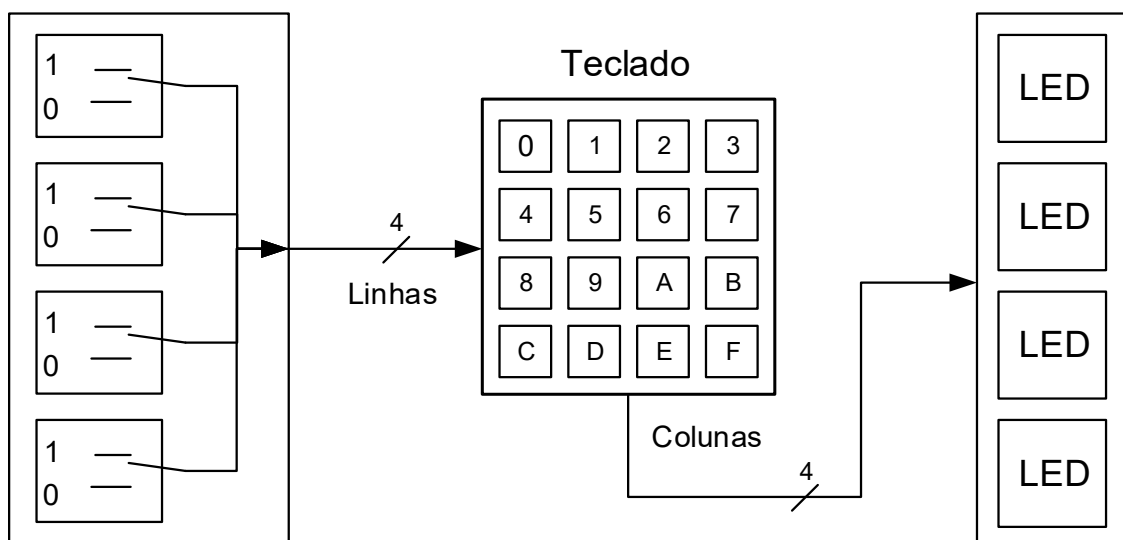
Com o valor 0010b aplicado nas linhas (o 1 está agora na segunda linha), é possível detetar se uma das teclas “4”, “5”, “6” e “7” foi premida, se se ler das colunas o valor 0001b, 0010b, 0100b ou 1000b, respetivamente.

Raciocínio idêntico aplica-se às teclas da 3ª e 4ª linhas do teclado.

Desta forma, fazendo um ciclo de “varrimento” às quatro linhas consegue-se detetar qualquer tecla em qualquer linha.

3 Teste do teclado


Primeiro vamos testar e verificar o funcionamento do teclado com um circuito muito simples, descrito pela figura seguinte. Um conjunto de 4 interruptores permite injetar valores nas linhas do teclado (entradas) e um conjunto de 4 leds permite ver o estado das colunas (saídas).

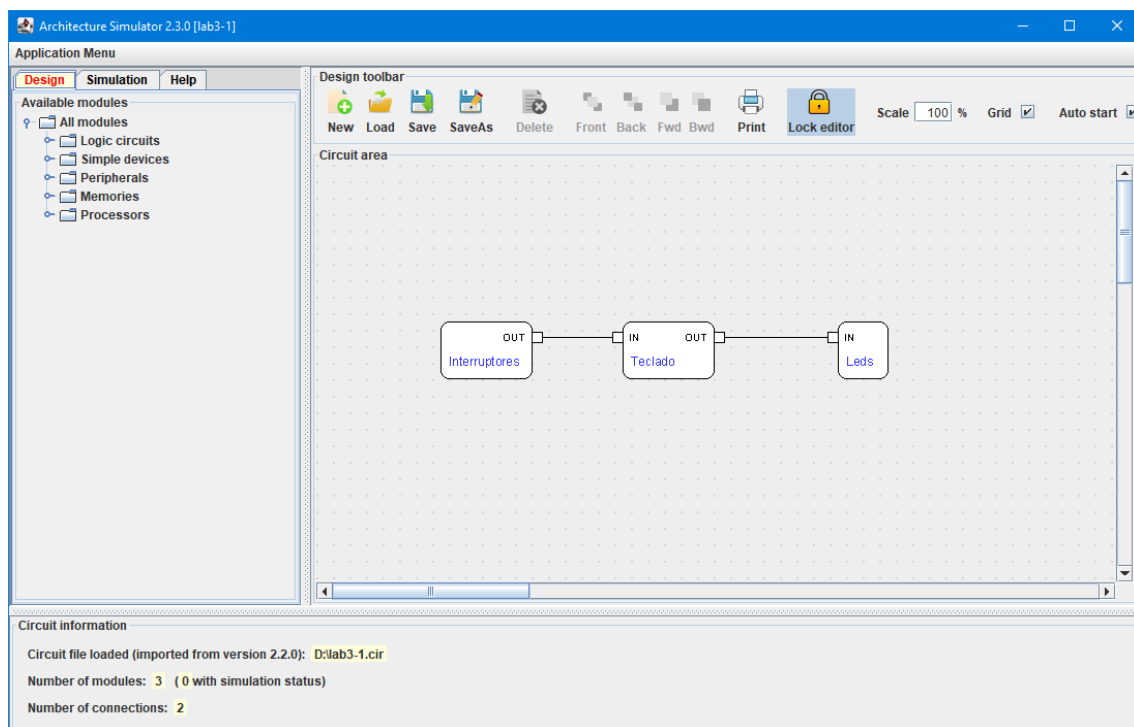


Um led liga apenas quando uma tecla é carregada e o bit injetado na linha dessa tecla está a 1.

O circuito que implementa este diagrama é dado já montado (ficheiro **lab3-1.cir**).

Faça o seguinte:

- Carregue este circuito no simulador (ou pelo botão **Load source**, , ou por *drag & drop*);
- Passe para “Simulation”;
- Abra as janelas de controlo de cada um destes dispositivos (com clique).



Experimente colocar o valor 0001b nos interruptores, fazendo clique no bit 0 dos interruptores.

As janelas de controlo dos dispositivos deverão ter o seguinte aspeto:



Carregue agora nas várias teclas do teclado e verifique que quando carrega numa tecla da primeira linha o led respetivo se liga. Nas teclas das outras linhas, nada acontece.

Note que:

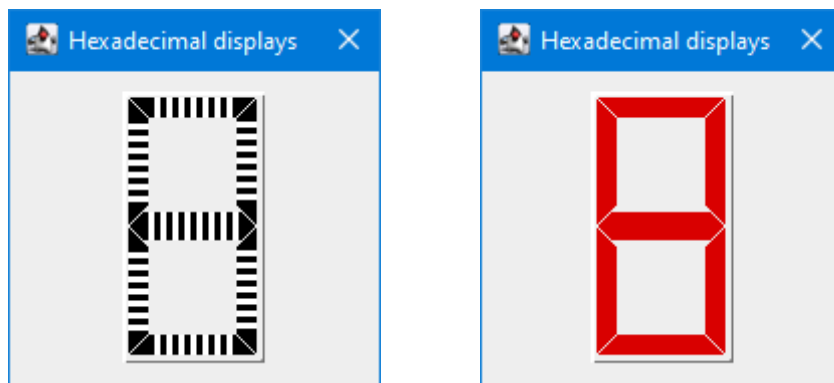
- Só as teclas da primeira linha são detetadas, pois só a primeira linha tem um 1;
- A tecla 0 está à esquerda, enquanto o bit de ordem 0 nas colunas está à direita. É apenas efeito visual. O que interessa é a ordem dos bits lidos das colunas;


Agora experimente sucessivamente com os valores 0010b, 0100b e 1000b nos interruptores, e verifique que teclas consegue detetar em cada caso. Por fim, experimente com o valor 1111b nos interruptores, clique nas várias teclas de uma mesma coluna e verifique que assim não se consegue distingui-las.

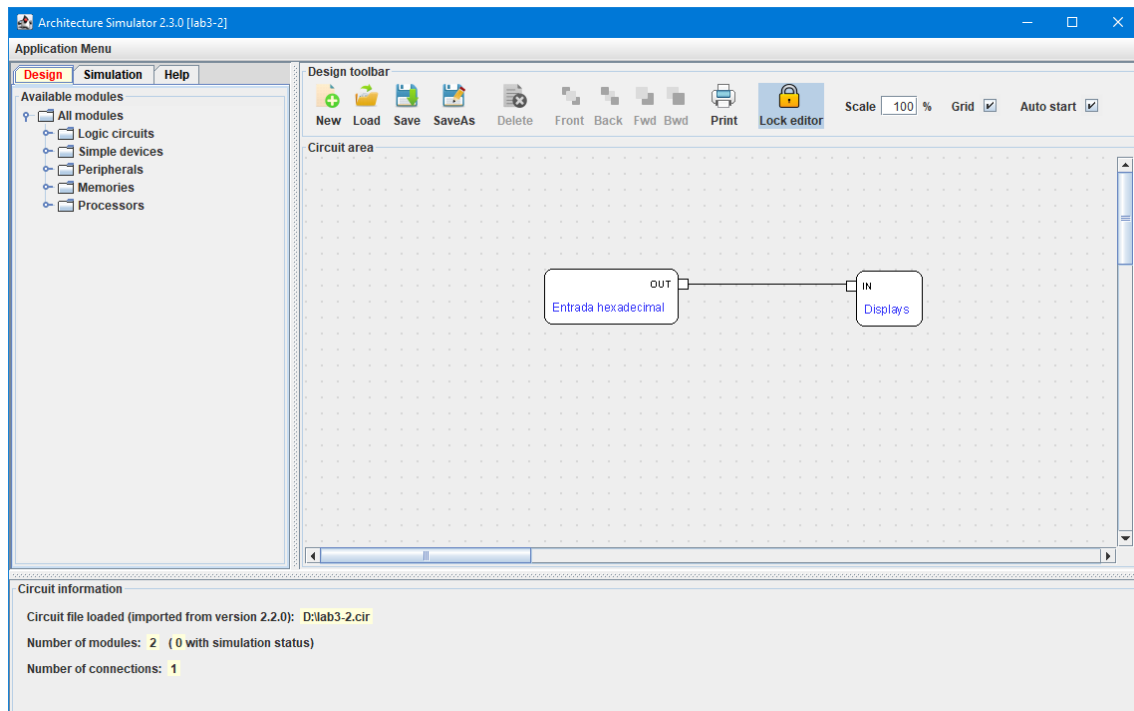
Conclusão: para detetar uma tecla qualquer tem de se testar as diversas linhas, mas só uma de cada vez.

4 Displays hexadecimais

Permitem visualizar dígitos hexadecimais, de 0 a F, ligando um ou mais dos 7 segmentos que compõem cada display. Têm apenas uma entrada, de 4 bits (para especificar uma das 16 hipóteses). Permitem indicar também quando a sua entrada não tem um valor definido (figura do lado esquerdo). As cores são configuráveis (fazendo duplo clique no display em modo “Design”).

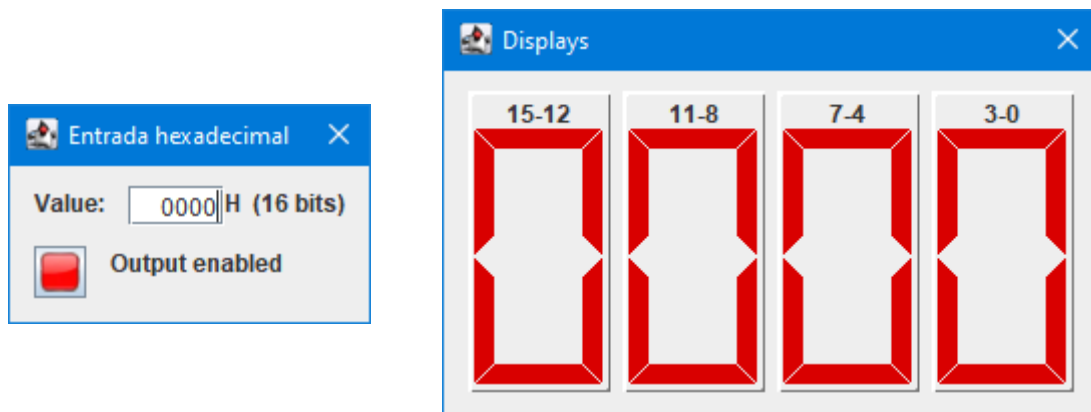


Para testar o funcionamento dos displays, carregue o circuito que implementa o diagrama seguinte, contido no ficheiro **lab3-2.cir** (com o botão **Load source**, , ou por *drag & drop*).



Este circuito contém um módulo Entrada hexadecimal, que permite especificar diretamente valores hexadecimais, e um módulo Displays. Ambos estão configurados para 16 bits, ou seja, 4 dígitos hexadecimais.

Passe para “Simulation” e faça clique em cada um dos módulos, o que abre os respectivos painéis de controlo, com o seguinte aspeto:

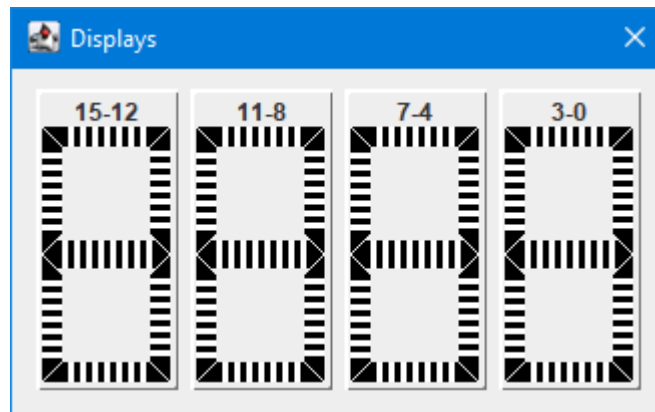


A entrada hexadecimal permite escrever (com o teclado do computador) 4 dígitos hexadecimais, ou 16 bits, como indicado (se um valor for inválido, fica vermelho).

O módulo Displays mostra 4 dígitos hexadecimais e, por cima de cada um, os bits que lhe correspondem no pino de entrada (IN) do módulo. Os dígitos hexadecimais introduzidos na entrada hexadecimal aparecem nos displays.

A Entrada hexadecimal tem um botão que permite desligar a sua saída, deixando de forçar um valor. Nestas circunstâncias, o pino de entrada dos Displays fica “no ar” (estado também

conhecido por Alta Impedância, ou valor Z). Experimente carregar nesse botão e veja que os displays ficam com aspecto de não inicializados.

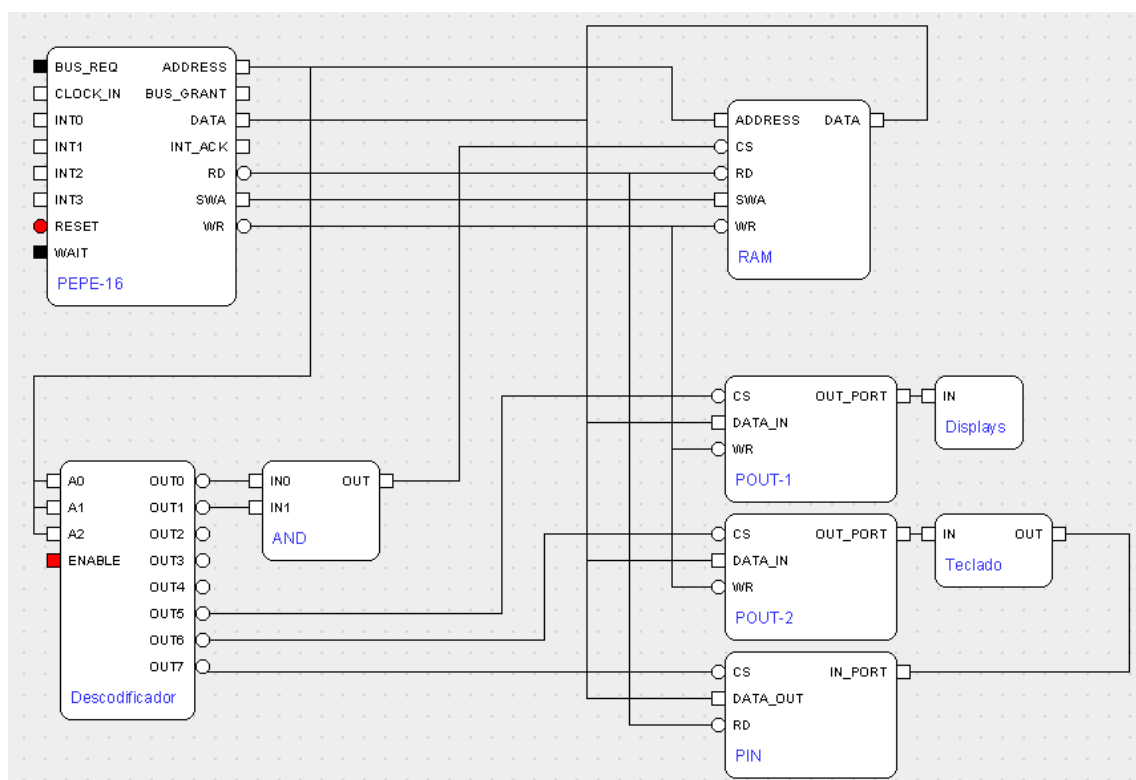


Por esta razão, diz-se que o pino de saída da Entrada hexadecimal é Tristate (cada bit tem 3 estados, 0, 1 e Z).

5 Leitura do teclado e escrita nos displays com o PEPE-16

5.1 Circuito

O circuito para leitura do teclado com um programa é fornecido já montado (ficheiro **lab3-3.cir**).



São necessários os seguintes módulos:

- **PEPE-16** (cujo relógio é gerado internamente);
- **RAM** (uma memória de 16 bits de dados, 14 bits de endereço e com capacidade de endereçamento de byte);
- **POUT-1** (periférico de saída – liga aos displays. Este circuito tem 2 dígitos hexadecimais);
- **POUT-2** (periférico de saída – liga às linhas do Teclado);
- **PIN** (periférico de entrada – liga às colunas do Teclado);
- **Teclado** (teclado de 4 linhas e 4 colunas);
- **Descodificador e AND** (para permitir aceder aos dispositivos. Neste momento não é relevante. A descodificação de endereços será tratada no guião de laboratório 8).

NOTAS IMPORTANTES:

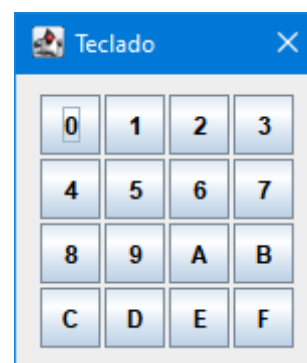
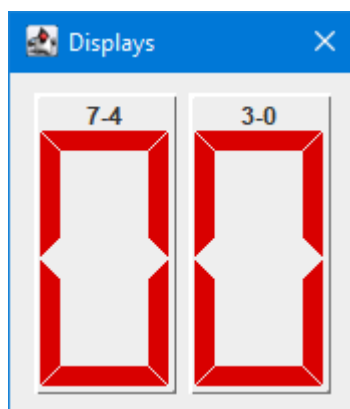
- Os acessos aos periféricos (quer em escrita quer em leitura) devem ser feitos com a instrução **MOVB**, pois estes periféricos são de apenas 8 bits;
- Os acessos à memória podem ser feitos com **MOV** (16 bits) ou **MOVB** (8 bits);
- A memória e periféricos estão disponíveis nos seguintes endereços:

Dispositivo	Endereços
RAM (memória)	0000H a 3FFFH
POUT-1 (periférico de saída de 8 bits)	A000H
POUT-2 (periférico de saída de 8 bits)	C000H
PIN (periférico de entrada de 8 bits)	E000H

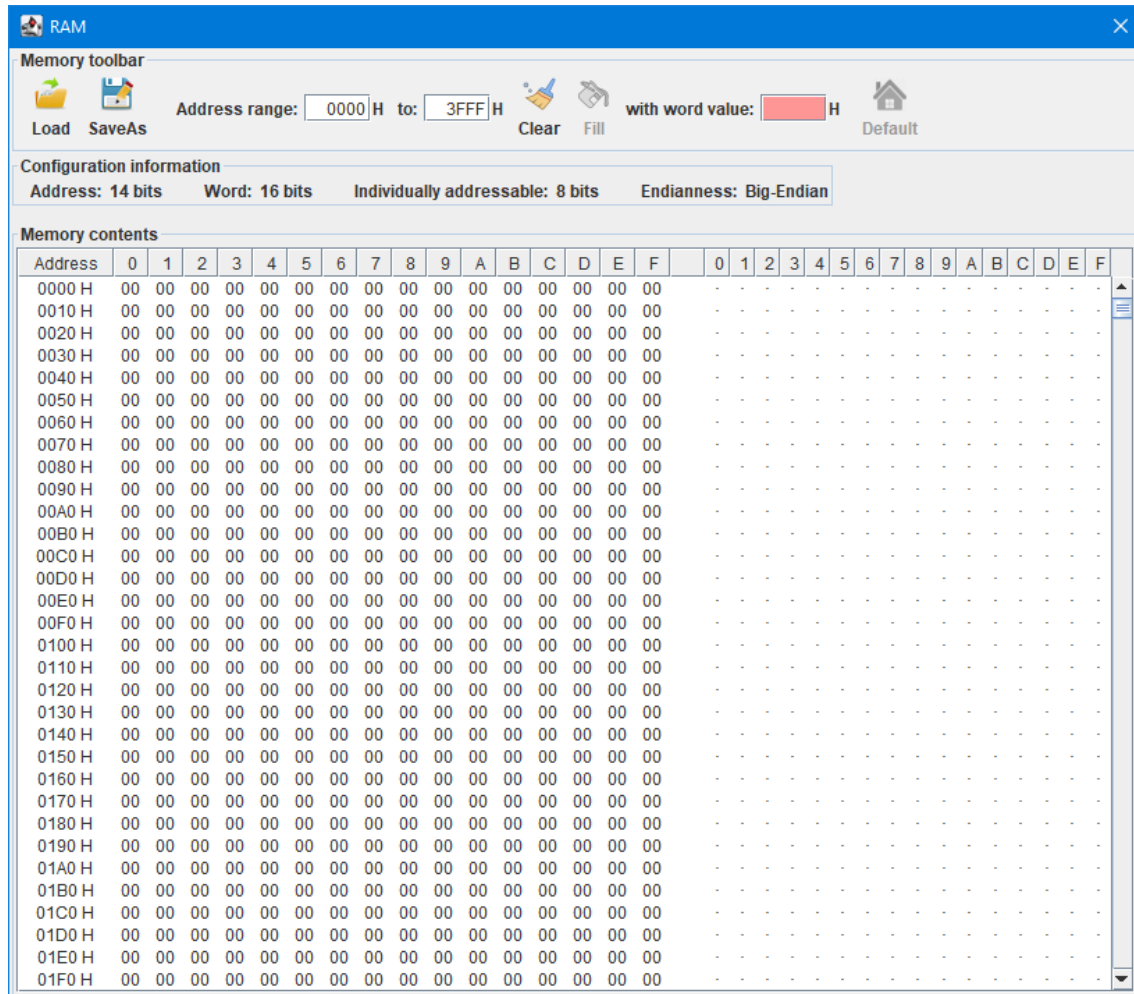
5.2 Simulação do circuito

Passe o simulador para “Simulation”.

Faça clique nos displays e no teclado, para abrir os respetivos painéis, e desloque-os para uma zona do ecrã fora da janela do simulador.



Faça também clique na RAM e abra o respetivo painel, que tem o aspeto indicado na página seguinte. Verifique, fazendo scroll na barra do lado direito, que o endereço máximo é 3FFFH, tal como deve ser atendendo a que esta memória tem 14 bits de endereço ($2^{14} = 4000H$).




5.3 Programa de leitura do teclado e escrita nos displays

Abra o ficheiro **lab3.asm** (com um editor de texto do tipo NotePad++ para PC ou Brackets para Mac) e tente perceber o que faz este programa, pelos comentários e pelo funcionamento descrito na secção 2.

NOTA – O programa tem dois ciclos separados: um espera que uma tecla seja premida, enquanto o outro espera que essa tecla seja largada (o clique na tecla acabe). Desta forma, caso o premir de uma tecla seja usada para implementar alguma funcionalidade, premir uma tecla só executa essa funcionalidade uma vez, e não repetidamente enquanto a tecla estiver premida.

De seguida, execute os seguintes passos:

1. Abra o painel de controlo do PEPE-16 (clique no módulo) e carregue o ficheiro **lab3.asm** (pelo botão **Load source**, , ou por *drag & drop*);

2. Execute o programa em modo contínuo, com o botão **Start** (▶), e carregue numa tecla da linha 4 (a de baixo). Verifique que os displays exibem a linha (8) e a coluna (1, 2, 4 ou 8) correspondentes à tecla em que carregou (ou seja, 81, 82, 84 ou 88);
3. Verifique também que estes valores são exibidos apenas enquanto a tecla estiver a ser carregada. Verifique no programa o que é que permite ter esta distinção de comportamento, consoante a tecla está carregada ou não;
4. Verifique que em qualquer altura pode fazer pausa à execução do programa, carregando no botão **Pause** (⏸) do PEPE-16. A partir daí pode então executar uma instrução de cada vez, ou passo a passo, carregando no botão **Step** (▶) do PEPE-16. Em qualquer altura, pode voltar a executar em modo contínuo, carregando no botão **Resume** (▶);
5. Note que não se conseguem detetar teclas carregadas em execução passo a passo, pois o cursor do rato é só um (e ou se carrega no botão **Step** (▶) ou no teclado). A funcionalidade de auto-step permite resolver este dilema. Selecione a velocidade mais rápida (“Very fast”) e carregue no botão **Step** (▶):

PEPE-16

Simulation toolbar

Stepping Resume Pause Over Stop Reset Load source Reload Load bin Check source Delete breakpoints

Clock Internal Auto start Simulation master Auto step Very fast Ignore breakpoints

PEPE-16

Program file loaded (Assembly): View Source only Tab spaces 4 Encoding windows-1252

D:\lab3.asm

Program

Line	Addr	Source
25		*****
26		PLACE 0
27		inicio:
28		; inicializações
29	0000 H	MOV R2, TEC_LIN ; endereço do periférico das linhas
30	0004 H	MOV R3, TEC_COL ; endereço do periférico das colun...
31	0008 H	MOV R4, DISPLAYS ; endereço do periférico dos displ...
32	000C H	MOV R5, MASCARA ; para isolar os 4 bits de menor p...
33		
34		; corpo principal do programa
35		ciclo:
36	000E H	MOV R1, 0
37	0010 H	MOVB [R4], R1 ; escreve linha e coluna a zero nos di...
38		
39		espera_tecla: ; neste ciclo espera-se até uma tecla ...
40	0012 H	MOV R1, LINHA ; testar a linha 4
41	0014 H	MOVB [R2], R1 ; escrever no periférico de saída (linh...
42	0016 H	MOVB R0, [R3] ; ler do periférico de entrada (colunas)
43	0018 H	AND R0, R5 ; elimina bits para além dos bits 0-3
44	001A H	CMP R0, 0 ; há tecla premida?
45	001C H	JZ espera_tecla ; se nenhuma tecla premida, repete
46		; vai mostrar a linha e a coluna da tecla
47	001E H	SHL R1, 4 ; coloca linha no nibble high
48	0020 H	OR R1, R0 ; junta coluna (nibble low)
49	0022 H	MOVB [R4], R1 ; escreve linha e coluna nos displays
50		
51		ha_tecla: ; neste ciclo espera-se até NENHUMA te...
52	0024 H	MOV R1, LINHA ; testar a linha 4 (R1 tinha sido altera...
53	0026 H	MOVB [R2], R1 ; escrever no periférico de saída (linh...
54	0028 H	MOVB R0, [R3] ; ler do periférico de entrada (colunas)
55	002A H	AND R0, R5 ; elimina bits para além dos bits 0-3

Main registers

PC 0016 H Level: System

R0 0000 H R1 0008 H

R2 C000 H R3 E000 H

R4 A000 H R5 000F H

R6 0000 H R7 0000 H

R8 0000 H R9 0000 H

R10 0000 H R11 0000 H

SP 0000 H RE 0001 H

BTE 0000 H TEMP 0000 H

SSP 0000 H USP 0000 H

Flags and pending interrupts

TD TV B A V C N Z

R NP DE IE3 IE2 IE1 IE0 IE

INT3 INT2 INT1 INTO

Program symbols

Value	Symbol	Type
0000 H	inicio	LABEL
0008 H	LINHA	EQU
000E H	ciclo	LABEL
000F H	MASCARA	EQU
0012 H	espera_tecla	LABEL
0024 H	ha_tecla	LABEL
A000 H	DISPLAYS	EQU

Apply Cancel

6. Verifique que a barra azul circula entre o *label* “espera_tecla” e pouco mais abaixo, mas que quando carrega no teclado numa tecla da última linha o programa vai até à última instrução (tem de deixar a tecla carregada tempo suficiente para o PEPE-16 ler o teclado na instrução **MOVB R0, [R3]**);

NOTA – O AND com R5 (0FH) destina-se a forçar a 0 os bits 7 a 4 lidos do periférico de entrada, pois estes estão “no ar” (uma vez que o teclado só liga aos bits 3 a 0) e dão valores aleatórios, pelo que têm de ser eliminados;

7. Termine a execução do programa, carregando no botão **Stop** (■) do PEPE-16;
8. No editor de texto, altere o valor da constante **LINHA** (no ficheiro **lab3.asm**) para 1, 2 ou 4. Guarde o ficheiro, faça **Reload** (↻) e volte ao passo 2 acima, verificando que agora o programa deteta teclas noutra linha (indicada pela constante **LINHA**).

6 Exercício

O programa do ficheiro **lab3.asm** dá apenas para uma tecla numa linha. Pretende-se detetar qualquer tecla em qualquer linha. O teclado é um componente essencial para a execução de programas que usem o teclado como elemento de entrada e/ou controlo.

Como exercício, faça um programa que faça o varrimento (em ciclo) das quatro linhas do teclado e detete qual a tecla premida, obtendo o seu valor hexadecimal (0 a FH), em vez de apenas os valores separados da linha e da coluna.

Para tal, deve fazer os passos seguintes:

- Faça um programa (pode usar o programa do ficheiro **lab3.asm** como base) que varre continuamente, em ciclo, as várias linhas do teclado, uma de cada vez, verificando se alguma das teclas foi premida e, quando tal acontecer, sai do ciclo com a linha e a coluna da tecla em dois registos (use *breakpoints* e/ou a funcionalidade de auto-step para verificar se os valores obtidos estão certos);
- Na realidade, o que quer obter é um valor entre 0 e FH, correspondente ao valor da tecla carregada, em vez de apenas informação separada sobre a linha e coluna da tecla. Acrescente código para converter esta informação no valor entre 0 e FH. Sugestão: o valor hexadecimal da tecla é igual a $4 * \text{linha} + \text{coluna}$, em que tanto linha como coluna são números entre 0 e 3. Logo, terá de converter 0001b, 0010b, 0100b e 1000b (isto é, 1, 2, 4 e 8, valores possíveis quer para a linha, quer para a coluna) em 0, 1, 2 e 3, respetivamente. Tal pode ser feito usando a instrução **SHR** (deslocamento à direita de 1 bit) dentro de um ciclo, contando o número de vezes que se tem de se fazer **SHR** ao valor da linha (ou da coluna) até este ser zero. Use um *breakpoint* para verificar se o valor obtido está certo, e se não estiver reinicie o programa e corra-o em *single-step* desde o *breakpoint* do ponto anterior, verificando o valor dos registos relevantes em cada passo;
- Acrescente código para mostrar o valor da última tecla carregada nos displays, em ciclo. Sempre que uma tecla é detetada e o seu valor mostrado, o programa deve voltar a esperar a deteção de uma nova tecla (e não terminar);

- Mas, mesmo realmente, o que quer é usar as teclas detetadas para executar funcionalidades (comandos) num dado programa. Simule esta capacidade fazendo com que os displays mostrem agora o valor de um contador hexadecimal (valor de um registo), que é incrementado quando se carrega numa dada tecla (escolha qual) e decrementado quando se carrega numa outra tecla (escolha qual). **Por cada tecla carregada, o contador só deve variar uma vez.**

O Help do simulador contém informação sobre cada instrução do PEPE-16.