

Laboratório de Introdução à Arquitetura de Computadores

IST - Taguspark

2020/2021

Rotinas

Guião 4

9 a 13 de novembro de 2020

(Semana 6)

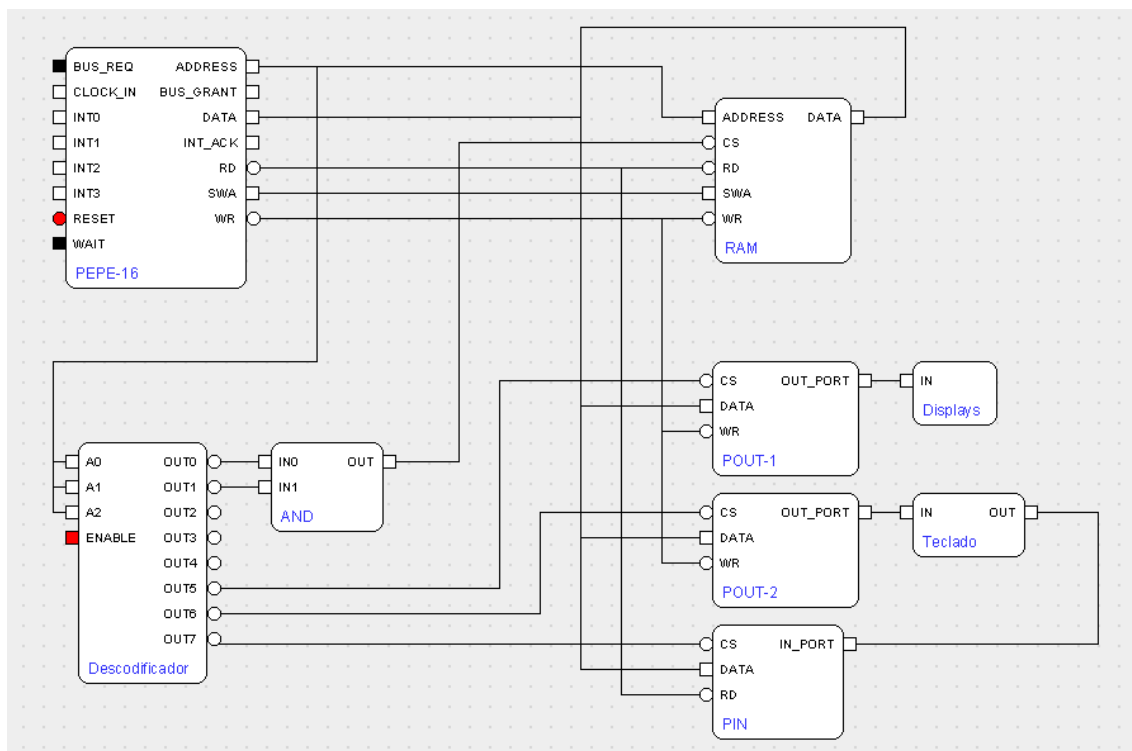
1 – Objetivos

Com este trabalho pretende-se que os alunos pratiquem programação em linguagem *assembly* do PEPE usando rotinas, para processamento e interação com periféricos.

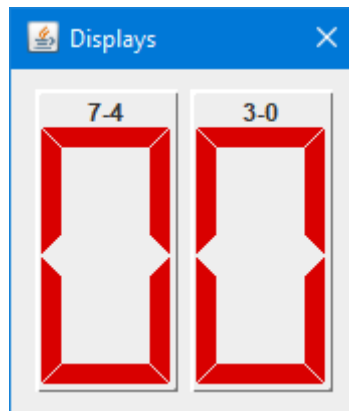
2 – Rotinas em linguagem assembly

2.1 – O circuito de simulação

Clique em **Load** (📁) e carregue o circuito contido no ficheiro **lab4-1.cir**. Este circuito é idêntico ao já usado no Guião 3 e será novamente usado para interação com os displays, mas agora com rotinas.



O periférico POUT-1 é de 8 bits e está ligado a dois displays de 7 segmentos, um ligado aos bits 7-4 (designado High) e o outro ligado aos bits 3-0 (Low).



Para escrever um valor nos dois displays, basta executar uma instrução **MOVB** no endereço **A000H**, em que o periférico POUT-1 está disponível, tal como no guião de laboratório 3:

| 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 |

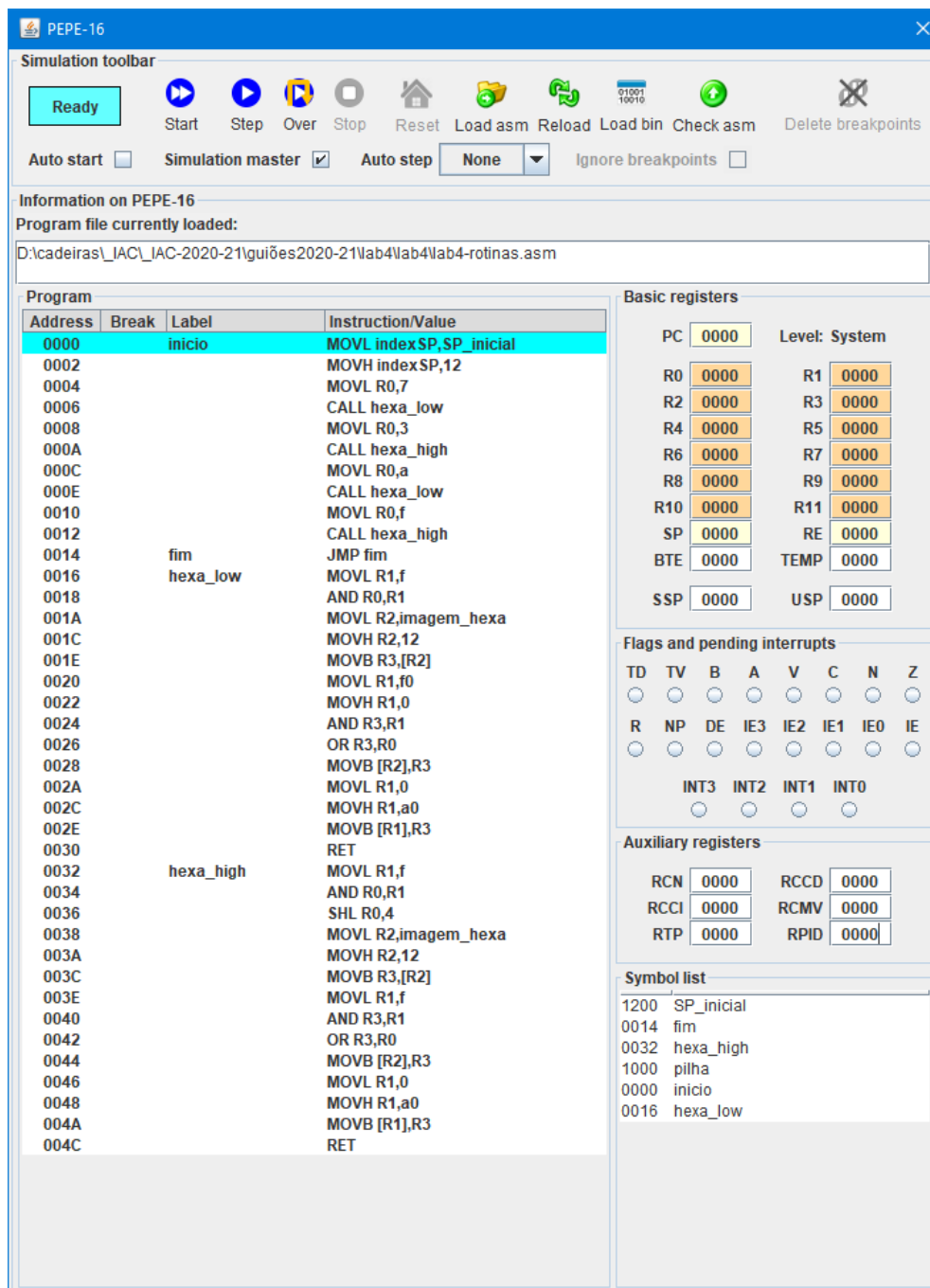
2.2 – Invocação de rotinas

Passe para “Simulation” e faça duplo clique no PEPE e nos displays para abrir os respetivos painéis de simulação.

Carregue o programa *assembly lab4-rotinas.asm*, com **Load asm** (📁) ou *drag & drop*. Este programa contém duas rotinas, uma para escrever um valor entre 0 e FH no display Low e outra correspondente para o display High.

Observe o programa e tente perceber o seu funcionamento. Note:

- A declaração do *stack* (pilha) na zona de dados e a etiqueta no fim dessa área;
- A inicialização do *stack pointer* (**SP**), no início do código, com essa etiqueta (a pilha usa-se a partir do fim, para trás). Esta inicialização é obrigatória;
- As instruções **CALL**, seguidas da etiqueta do endereço de uma rotina, para a invocar;
- As instruções **RET**, no fim de cada rotina, para regressar à instrução seguinte ao **CALL** que invocou essa rotina;
- A inicialização do registo que serve de parâmetro para passar a cada rotina.



Execute o programa passo a passo, com o botão **Step** (🔍), SEM carregar em **Start** (▶️).
Verifique que o **CALL** transfere o controlo para a rotina e que o **RET** faz regressar à instrução a seguir ao **CALL**.


Verifique também a evolução do **SP** (que na janela do PEPE é o **SSP**). Um **CALL** reduz o **SP** de 2 unidades e um **RET** incrementa-o de 2 unidades.



Vá observando a evolução das instruções (nomeadamente a chamada e retorno das rotinas) e os displays, até chegar à última instrução do programa.


Este exemplo é o suficiente para ilustrar o funcionamento das rotinas.


Termine a execução do programa, carregando no botão **Stop** (⏏️) do PEPE.

2.3 – Execução passo a passo com *Step Over*

Faça *reset* ao PEPE, com clique no botão **Reset** ()

Execute novamente o programa em passo a passo, mas desta vez usando o botão **Over** () em vez do botão **Step** () e tendo particular atenção às instruções **CALL**.

Note que, ao fazer *step-over* num **CALL**, o processador faz pausa na instrução com o endereço seguinte, e não na primeira instrução da rotina invocada, que é o que aconteceria com o botão **Step** (). Isto significa que a rotina foi executada à velocidade máxima, como se de uma só instrução se tratasse.

O botão **Over** () é muito útil para seguir um programa sem ter de entrar dentro de rotinas, interpolando os detalhes desnecessários.


Tirando as instruções **CALL**, os botões **Step** () e **Over** () são equivalentes.

2.4 – Preservação de registos nas rotinas

Um **CALL** esconde as instruções da rotina invocada. O problema é que esta pode alterar valores de registos, sem que o código onde está o **CALL** se aperceba. Desta forma, os valores desses registos podem ser uns antes do **CALL** e outros logo a seguir, após o retorno. Como uma rotina pode ser invocada de vários sítios, a situação fica descontrolada porque já não se sabe que registos são ou não importantes e cujos valores a rotina não deve mesmo alterar.



Para resolver este problema, a regra geral é que uma rotina não deve alterar o valor de nenhum registo (exceto os de resultado). Mas a rotina precisa de usar registos para poder trabalhar, e com isso destrói os valores anteriores destes registos!

A solução é a rotina, no início, guardar na pilha (com **PUSH**) os valores dos registos que vai alterar e no fim repor todos esses registos a partir da pilha (com **POP**).

Faça agora **Load asm** () ou *drag & drop* do programa *assembly lab4-registos.asm*. Este programa é igual ao anterior, com a exceção de que os registos usados pelas rotinas (**R1**, **R2** e **R3**) são preservados na pilha pelas próprias rotinas, com instruções **PUSH** e **POP**.

Estes registos são inicializados com valores para que se possa observar que após o retorno de uma rotina esses valores são repostos.

Note que a ordem dos **POPs** é inversa da dos **PUSHs**, pois a pilha funciona num regime LIFO (*Last In, First Out*).

Execute o programa do ficheiro passo a passo, com o botão o botão **Step** () SEM carregar no botão **Start** () . Verifique que os **POPs** repõem os valores guardados pelos **PUSHs** e que **PUSHs**, **POPs**, **CALLs** e **RETs** funcionam bem em conjunto (todos usam a pilha e alteram o **SP** – o *Stack Pointer*).

Vá observando a evolução das instruções e dos registos (nomeadamente o **SP**, **R1**, **R2** e **R3**). A funcionalidade do programa é igual à versão anterior.

Note que o **SP** desce agora mais, por causa dos **PUSHs**, mas no fim de cada rotina volta ao valor inicial. Cada **PUSH** reduz o **SP** de 2 unidades e cada **POP** incrementa-o de 2 unidades.

Como boa prática de programação, cada rotina deve preservar os valores de todos os registos que altera (mas apenas desses, e não de todos os registos!).

Termine a execução do programa, carregando no botão **Stop** () do PEPE.

2.5 – Rotinas que chamam rotinas

Faça agora **Load asm** (📁) ou *drag & drop* do programa *assembly lab4-variante.asm*. Este programa mantém a funcionalidade dos anteriores, mas inclui uma rotina (**hexa_display**) que recebe um byte e o escreve nos dois displays, chamando as outras duas rotinas (claro que o poderia fazer escrevendo simplesmente o byte no periférico...).

Note o uso das instruções de deslocamento para eliminar os bits que devem ser irrelevantes e para colocar os bits certos no lugar certo.

Execute o programa do ficheiro passo a passo, com o botão o botão **Step** (▶), SEM carregar no botão **Start** (▶). Vá observando a evolução das instruções e dos registos (em particular, o **SP** – o *Stack Pointer*). Note que a pilha mantém o estado da rotina **hexa_display** quando esta invoca outra. O **SP** desce ainda mais baixo que no caso anterior (porque agora há uma rotina que invoca outras), mas no fim do programa volta ao valor inicial.

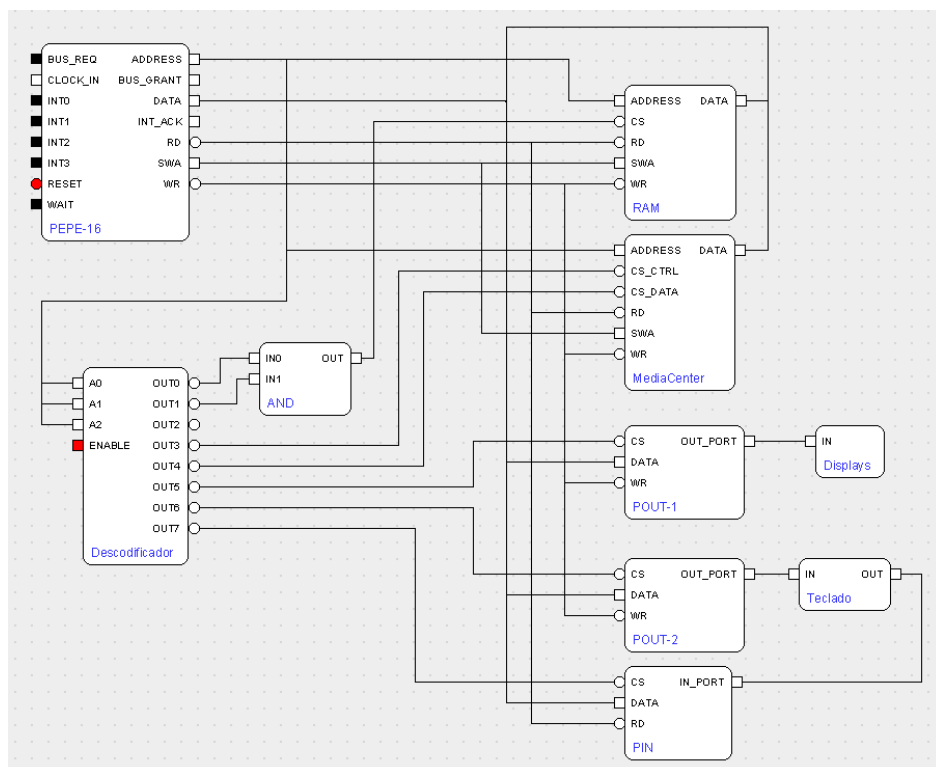
Termine a execução do programa, carregando no botão **Stop** (⏏) do PEPE.

Qualquer programa, seja em *assembly* ou numa linguagem de alto nível como Python, consiste num conjunto de rotinas (em alto nível, chamam-se funções ou métodos). Existe uma que é principal e que invoca outras, que por sua vez invocam outras, até chegar a rotinas que já não invocam mais nenhuma e que apenas retornam, quando terminam. A pilha mantém a história de invocação, para que cada rotina possa retornar para aquela que a invocou.

3 – Rotinas para manipular o ecrã

3.1 – O módulo MediaCenter

Passe o simulador para “Design” e faça **Load** (📁) do circuito contido no ficheiro **lab4-2.cir**. Este circuito é semelhante ao anterior, mas tem agora mais um periférico (MediaCenter, um ecrã gráfico de 32 x 64 pixels, com suporte para áudio e vídeo), e constitui mais um passo em direção ao circuito do projeto, permitindo desenhar objetos, pixel a pixel.



Cada pixel pode ter uma cor diferente, em 4 componentes (*Alpha*, *Red*, *Green* e *Blue*, ou ARGB), todas com 4 bits (valores não negativos, de 0 a 15), pois o PEPE tem 16 bits. A componente *Alpha* define a opacidade (0 é totalmente transparente, 15 totalmente opaco). O pixel pode estar ligado (com a cor definida pelas 4 componentes) ou desligado (todas as componentes a 0, caso em que não se vê).

As coordenadas de um dado pixel no ecrã são dadas em linha (0 a 31) e coluna (0 a 63). É possível alterar a cor de um dado pixel por dois métodos diferentes:

- Enviar comandos específicos para o MediaCenter, para seleccionar a linha, a coluna e a cor do pixel;
- Dado que o MediaCenter guarda os pixels numa memória de $32 \times 64 = 2048$ palavras (cada pixel ocupa 16 bits, ou uma palavra do PEPE), converter a linha e coluna para o endereço interno em que o pixel se encontra nesta memória, somar-lhe o endereço de base do MediaCenter e fazer um acesso de escrita normal (como se de uma RAM se tratasse), escrevendo directamente a cor do pixel (incluindo 0000H, para o apagar).

A figura seguinte representa a memória do MediaCenter, com 4096 bytes, ou 2048 palavras. Cada quadradinho é um pixel, que ocupa 2 bytes, ou 1 palavra:

- No topo, está indicado o endereço relativo (dentro de cada linha) da primeira palavra de cada 8 pixels e a ordem dos 64 pixels de cada coluna (nem todos, para ser mais simples). Note que os endereços das palavras são sempre pares;
- No lado esquerdo, o endereço relativo (dentro da memória do MediaCenter) da primeira palavra de cada linha;
- No lado direito, a ordem dos 32 pixels de cada linha (cresce de cima para baixo).

| | 0H | | | | | | | 10H | | | | | | | 20H | | | | | | | 30H | | | | | | | 40H | | | | | | | 50H | | | | | | | 60H | | | | | | | 70H | | | | | | | |
|------|----|---|---|---|---|---|---|-----|---|---|----|----|----|----|-----|----|----|--|----|----|--|-----|--|----|----|--|--|--|-----|----|--|--|--|----|----|-----|--|--|----|----|--|--|-----|----|--|--|--|----|--|-----|--|--|--|--|--|--|--|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | | 23 | 24 | | | | 31 | 32 | | | | 39 | 40 | | | | 47 | 48 | | | | 55 | 56 | | | | 63 | | | | | | | | | | | | | |
| 0H | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 0 | | | | | | | | | |
| 80H | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 1 | | | | | | | | | |
| 100H | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 2 | | | | | | | | | |
| 180H | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 3 | | | | | | | | | |
| 200H | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 4 | | | | | | | | | |
| 280H | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 5 | | | | | | | | | |
| 300H | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 6 | | | | | | | | | |
| 380H | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 7 | | | | | | | | | |
| 400H | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 8 | | | | | | | | | |
| 480H | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 9 | | | | | | | | | |
| 500H | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 10 | | | | | | | | | |
| 580H | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 11 | | | | | | | | | |
| 600H | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 12 | | | | | | | | | |
| 680H | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 13 | | | | | | | | | |
| 700H | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 14 | | | | | | | | | |
| 780H | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 15 | | | | | | | | | |
| 800H | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 16 | | | | | | | | | |
| 880H | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 17 | | | | | | | | | |
| 900H | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 18 | | | | | | | | | |
| 980H | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 19 | | | | | | | | | |
| A00H | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 20 | | | | | | | | | |
| A80H | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 21 | | | | | | | | | |
| B00H | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 22 | | | | | | | | | |
| B80H | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 23 | | | | | | | | | |
| C00H | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 24 | | | | | | | | | |
| C80H | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Desta forma, há uma correspondência directa entre a imagem do MediaCenter onde aparecem os pixels e a memória cujas palavras indicam a cor de cada pixel. Os endereços das palavras variam entre 000H e FFEH (sempre de 2 em 2, pois têm de ser pares). Isto dá 4096 endereços, ou 2048 palavras.

Estes endereços são internos ao MediaCenter. No entanto, o PEPE “vê” a memória do MediaCenter a partir do endereço 8000H (determinado pelo sistema de descodificação de endereços deste circuito), pelo que estes endereços, do ponto de vista do PEPE, correspondem ao intervalo 8000H a e 8FFFH (endereço do último pixel: 8FFEH).

Tal como a RAM do sistema, a memória do MediaCenter pode ser acedida em *byte* (com **MOVB**) ou em *word* (com **MOV**). No entanto, é mais simples lidar com uma palavra de cada vez, com **MOV**, pois assim lida-se logo com um pixel inteiro.

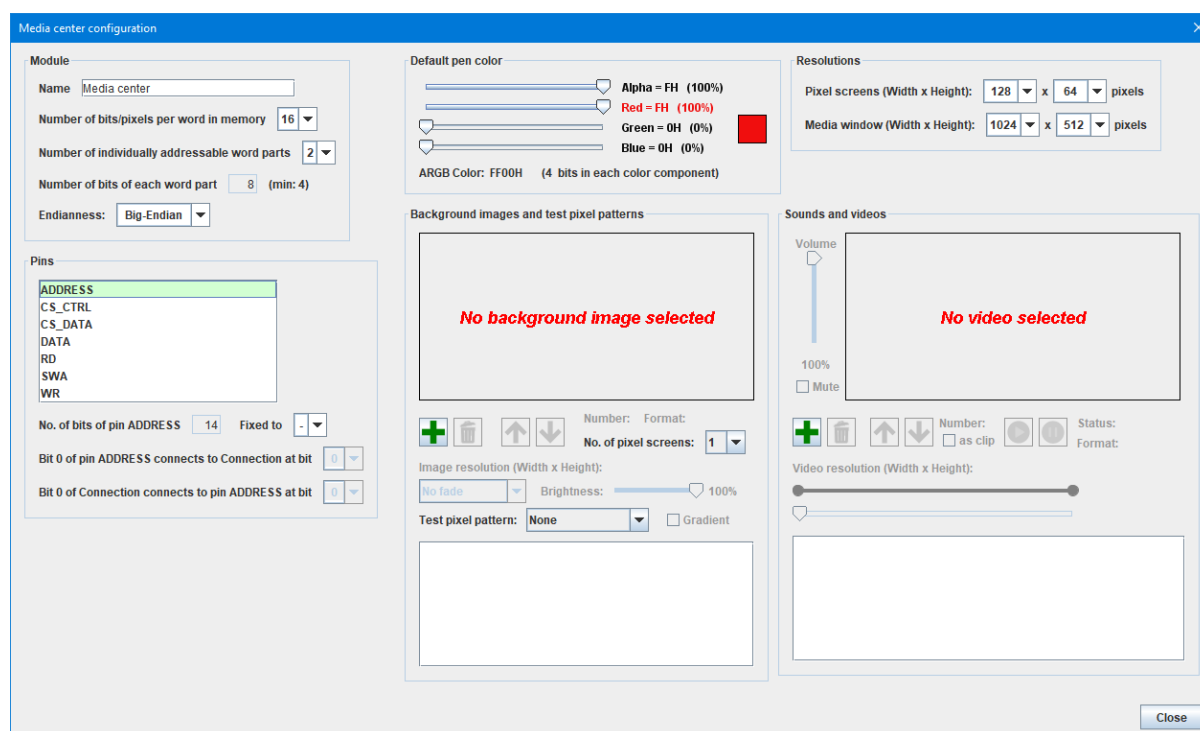
Em vez de escrever diretamente na memória dos pixels, o PEPE pode dar comandos ao MediaCenter, escrevendo em endereços a partir de 6000H, da forma explicada mais adiante. O MediaCenter tem dois *chip selects*, um para a memória de pixels e outra para os comandos.

Desta forma, o mapa de endereços (em que os dispositivos podem ser acedidos pelo PEPE) disponível neste circuito é o seguinte:

| Dispositivo | Endereços |
|---|--------------------------------|
| RAM | 0000H a 3FFFH |
| MediaCenter (acesso aos comandos) | 6000H a 6063H (ver secção 3.3) |
| MediaCenter (acesso à sua memória) | 8000H a 8FFFH |
| POUT-1 (periférico de saída de 16 bits) | 0A000H |
| POUT-2 (periférico de saída de 8 bits) | 0C000H |
| PIN (periférico de entrada de 8 bits) | 0E000H |

3.2 – Configuração do ecrã

A figura seguinte ilustra a interface de configuração do MediaCenter (accedida com clique duplo no MediaCenter em modo “Design”).



Para além da configuração módulo e dos pinos, já usada noutros módulos, esta interface permite configurar:

- A cor por omissão da caneta, incluindo opacidade (*Alpha*), com que os pixels serão desenhados no ecrã, em formato ARGB (*Alpha*, *Red*, *Green* e *Blue*);
- A resolução do ecrã de simulação, aquele em que o PEPE escreve os pixels. Na realidade, é possível definir até 16 ecrãs de pixels, suportando sobreposição de objetos diferentes sem interferência;
- A resolução dos cenários de fundo, imagens/vídeos definidos em ficheiros que, se definidos, aparecem por trás dos pixels desligados (que são 100% transparentes). Esta resolução é geralmente bastante maior que a anterior, mas a imagem de menor dimensão é esticada para se sobrepor à outra;
- Os ficheiros com imagens de cenário (tipicamente, jpegs), havendo um painel para a pré-visualização. Podem ser adicionadas (carregando no “+” da categoria “Background images and test pixel patterns”, apagadas ou mudadas de ordem). Para serem guardadas de forma portátil no ficheiro do circuito, de modo a funcionarem noutro computador, os ficheiros de imagem devem estar no mesmo diretório (ou subdiretório dele) que o ficheiro do circuito (caso contrário, é guardado o *path* absoluto de cada ficheiro);
- Padrões de teste, que sobrepõem à imagem para se ter uma ideia de como ficam os objetos em cima das imagens), e até padrões de *fading* ao mudar de cenário ou vídeo;
- Os ficheiros com sons (MP3 e WAV) e vídeos (MP4). Podem ser adicionados (carregando no “+” da categoria “Sounds and videos”, apagados ou mudados de ordem). Para serem guardados de forma portátil no ficheiro do circuito, de modo a funcionarem noutro computador, os ficheiros de som devem estar no mesmo diretório (ou subdiretório dele) que o ficheiro do circuito (caso contrário, é guardado o *path* absoluto de cada ficheiro). Quando houver pelo menos um ficheiro, definido, há um botão para se poder tocar e é possível ver o seu formato e duração. Também é possível reduzir o seu volume de som e até definir os pontos de início e fim de reprodução.

3.3 – Escrita de pixels por comandos

Uma forma de escrever pixels no ecrã do MediaCenter é usando a sua interface de comandos. Para alterar algo no ecrã (a cor de um pixel, selecionar uma linha ou uma coluna, etc.), o PEPE faz uma escrita num endereço a partir do 6000H, com um valor que corresponde ao argumento do comando. Para ler informação do ecrã (cor de um pixel, que linha está atualmente selecionada, etc.), o PEPE faz uma leitura de um endereço a partir do 6000H. O valor lido é a informação pretendida.

Portanto, os endereços usados identificam os comandos e os valores (escritos ou lidos) indicam a nova informação, para alterar no ecrã, ou a informação lida do ecrã. Um dado endereço pode ter significados diferentes, consoante seja usado num acesso de escrita ou leitura. Estes acessos pelo PEPE não acedem realmente a uma memória, apenas executam comandos.

IMPORTANTE – Porque alguns comandos podem envolver valores que não cabem num byte, todos os comandos são pares. Para usar os comandos deve-se usar instruções de **MOV** (acesso em *word*), somando 6000H ao comando.

Os comandos de escrita atualmente disponíveis no MediaCenter são os seguintes:

| Endereço | Descrição | Valor a escrever |
|----------|--|--|
| 6000H | Apaga todos os pixels do ecrã especificado | 0 .. n.º ecrãs-1 |
| 6002H | Apaga todos os pixels de todos os ecrãs | --- |
| 6004H | Seleciona o ecrã especificado | 0 .. n.º ecrãs-1 |
| 6006H | Mostra o ecrã especificado | 0 .. n.º ecrãs-1 |
| 6008H | Esconde o ecrã especificado | 0 .. n.º ecrãs-1 |
| 600AH | Especifica a linha a usar no próximo comando | 0 .. n.º linhas-1 |
| 600CH | Especifica a coluna a usar no próximo comando | 0 .. n.º colunas-1 |
| 600EH | Especifica o n.º do pixel a usar no próximo comando | 0 .. n.º pixels-1 |
| 6010H | Especifica o auto-increment. Se ligado, os comandos que alteram a cor de pixels (com *) avançam automaticamente para o(s) pixel(s) seguinte(s) | 0 - desliga; 1 - liga (desligado por omissão) |
| 6012H | (*) Altera a cor do pixel na posição corrente | Palavra com ARGB |
| 6014H | Especifica a cor da caneta | Palavra com ARGB |
| 6016H | (*) Altera a cor do pixel indicado para a cor da caneta | 0 .. n.º pixels-1 |
| 6018H | (*) Apaga o pixel indicado | 0 .. n.º pixels-1 |
| 601AH | (*) Altera a cor do pixel na posição corrente | 0 - apaga; 1 - caneta |
| 601CH | (*) Altera a cor de 8 pixels, a partir da maior posição múltipla de 8 não é maior que a posição corrente | Cada bit: 0 - apaga; 1 - caneta |
| 601EH | (*) Altera a cor de 16 pixels, a partir da maior posição múltipla de 16 não é maior que a posição corrente | Cada bit: 0 - apaga; 1 - caneta |
| 6020H | Desenha um padrão (um dos 12 que se podem indicar na janela de configuração do MediaCenter) | 0 .. 11 |
| 6022H | Igual ao anterior, mas com gradiente na cor | 0 .. 11 |
| --- | 6024H a 603EH reservados para expansão futura | --- |
| 6040H | Apaga o cenário de fundo (e elimina o aviso: "No background image selected") | --- |
| 6042H | Seleciona o n.º do cenário de fundo a visualizar | 0 .. n.º imagens-1 |
| 6044H | Apaga o cenário frontal | --- |
| 6046H | Seleciona o n.º do cenário frontal a visualizar | 0 .. n.º imagens -1 |
| 6048H | Seleciona um som/vídeo para comandos seguintes | 0 .. n.º sons/videos-1 |
| 604AH | Especifica o volume do som (percentagem) do som/vídeo selecionado | 0 .. 100 |
| 604CH | Corta o volume (<i>mute</i>) do som/vídeo especificado | 0 .. n.º sons/videos-1 |
| 604EH | Retoma o volume do som/vídeo especificado | 0 .. n.º sons/videos-1 |
| 6050H | Corta o volume (<i>mute</i>) de todos os sons/vídeos a reproduzir | --- |
| 6052H | Retoma o volume de todos os sons/vídeos a reproduzir | --- |
| 6054H | Especifica o brilho dos vídeos e do cenário de fundo | 0 .. 100 |
| 6056H | Especifica um padrão de transição entre vídeos (um dos 5 que se podem indicar na janela de configuração do MediaCenter) | 0- no fading; 1 - very slow; 2 - slow; 3 - fast; 4 - very fast |
| 6058H | Especifica o número de vezes que um som/vídeo deve ser reproduzido | 0 – repetição até ser parado |
| 605AH | Inicia a reprodução do som/vídeo especificado | 0 .. n.º sons/videos-1 |
| 605CH | Reproduz o som/vídeo especificado em ciclo até ser parado | 0 .. n.º sons/videos-1 |



| | | |
|-------|---|------------------------|
| 605EH | Pausa a reprodução do som/vídeo especificado | 0 .. n.º sons/videos-1 |
| 6060H | Continua a reprodução do som/vídeo especificado | 0 .. n.º sons/videos-1 |
| 6062H | Pausa a reprodução de todos os sons/vídeos a reproduzir | --- |
| 6064H | Continua a reprodução de todos os sons/vídeos em pausa | --- |
| 6066H | Termina a reprodução do som/vídeo especificado | 0 .. n.º sons/videos-1 |
| 6068H | Termina a reprodução de todos os sons/vídeos a reproduzir | --- |

Os comandos de leitura atualmente disponíveis no MediaCenter são os seguintes:

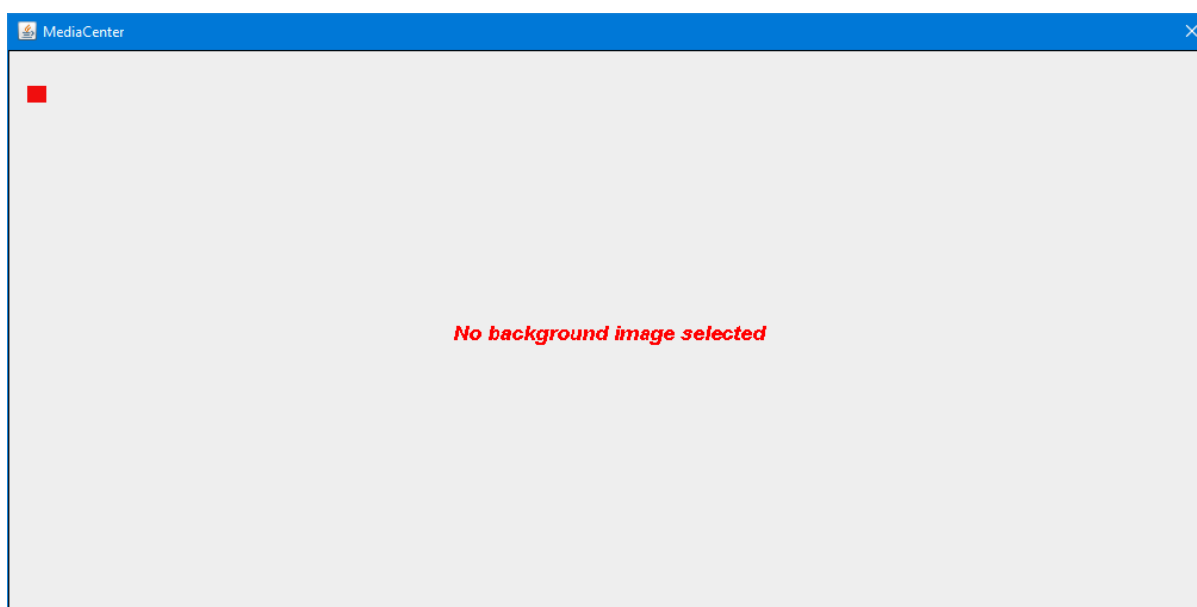
| Endereço | Descrição | Valor lido |
|----------|---|---|
| 6000H | Obtém o n.º de colunas do ecrã (potência de 2) | 0 .. 2048 |
| 6002H | Obtém o n.º de linhas do ecrã (potência de 2) | 0 .. 1024 |
| 6004H | Obtém o ecrã atualmente selecionado | 0 .. n.º ecrãs-1 |
| 6006H | Obtém a visibilidade do ecrã atualmente selecionado | 0 - escondido; 1 - visível |
| 6008H | Obtém a linha da posição corrente | 0 .. n.º linhas-1 |
| 600AH | Obtém a coluna da posição corrente | 0 .. n.º colunas-1 |
| 600CH | Obtém o n.º do pixel da posição corrente | 0 .. n.º pixels-1 |
| 600EH | Obtém o estado do auto-increment. Se ligado, os comandos que alteram a cor de pixels (com *) avançam automaticamente para o(s) pixel(s) seguinte(s) | 0 - desligado; 1 - ligado |
| 6010H | (*) Obtém a cor do pixel na posição corrente | Palavra com ARGB |
| 6012H | Obtém a cor atual da caneta | Palavra com ARGB |
| 6014H | (*) Obtém o estado da cor do pixel corrente | 0 - apagado; 1 – cor não zero |
| 6016H | (*) Obtém o estado da cor de 8 pixels, a começar na maior posição múltipla de 8 que não é maior que a posição corrente | Byte, com cada bit: 0 - apagado; 1 – cor não zero |
| 6018H | (*) Obtém o estado da cor de 16 pixels, a começar na maior posição múltipla de 16 que não é maior que a posição corrente | Palavra, com cada bit: 0 - apagado; 1 – cor não zero |
| --- | 601AH a 603EH reservados para expansão futura | --- |
| 6040H | Obtém n.º de imagens definidas | 0 .. 32 |
| 6042H | Obtém o n.º do cenário de fundo selecionado | -1 .. n.º imagens-1 (-1 se nenhum) |
| 6044H | Obtém o n.º do cenário frontal selecionado | -1 .. n.º imagens-1 (-1 se nenhum) |
| 6046H | Obtém n.º de sons/vídeos definidos | 0 .. 32 |
| 6048H | Obtém o n.º do som/vídeo selecionado | -1 .. n.º sons/videos-1 (-1 se nenhum) |
| 604AH | Obtém o volume do som (percentagem) do som/vídeo selecionado | 0 .. 100 |
| 604CH | Obtém o estado de <i>mute</i> do som/vídeo especificado | 0 - normal; 1 - muted |
| 604EH | Obtém o valor do brilho (percentagem) | 0 .. 100 |
| 6050H | Obtém o n.º do padrão de transição entre vídeos (um dos 5 que se podem indicar na janela de configuração do MediaCenter) | 0- no fading; 1 - very slow; 2 - slow; 3 - fast; 4 - very fast |

| | | |
|-------|---|---|
| 6052H | Obtém o estado do som/vídeo selecionado | 0 - Unknown; 1 - Ready; 2 - Paused; 3 - Playing; 4 - Stopped; 5 - Error |
| 6054H | Obtém o n.º dos sons/vídeos a reproduzir | 0 .. n.º sons/videos-1 |
| 6056H | Obtém o n.º de vezes que o som/vídeo selecionado será reproduzido | 0 - repetição até ser parado |

Para exemplificar o uso de comandos com o ecrã, edite e observe o programa contido no ficheiro **lab4-comandos.asm**. Contém uma rotina que recebe a linha, coluna e a cor do pixel (logo, também serve para apagar um pixel).

Passe o simulador para “Simulation” e carregue o ficheiro **lab4-comandos.asm** no PEPE (com o botão **Load asm**, , ou por *drag & drop*). Execute-o, com o botão **Start** () e verifique que o pixel da linha 2, coluna 1, se liga. Note que tanto as linhas como as colunas começam em 0 (canto superior esquerdo do ecrã).

Termine a execução do programa, carregando no botão **Stop** () do PEPE.



O aviso “No background image selected” existe para lembrar que os cenários de fundo têm de ser explicitamente selecionados, com o comando de escrita 6042H. No entanto, se quiser eliminar o aviso sem selecionar nenhum cenário, basta executar o comando de escrita 6040H.

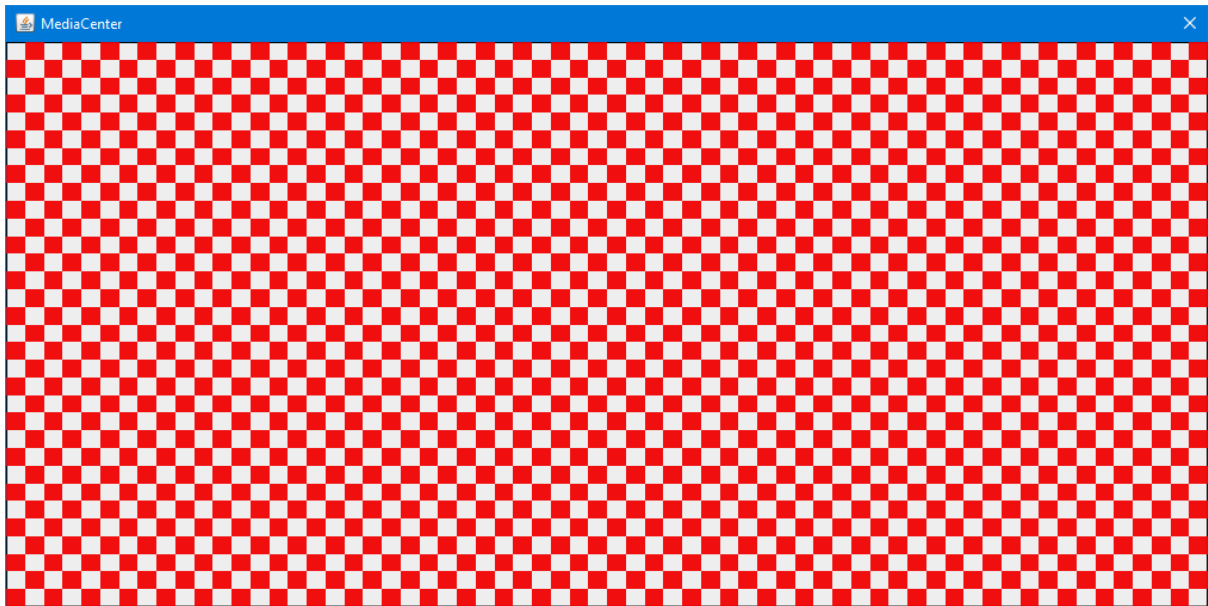
Carregue agora o programa contido no ficheiro **lab4-comandos-xadrez.asm**.

Este programa usa a rotina do programa anterior para escrever um pixel numa dada linha e coluna para implementar uma rotina que escreve uma linha de pixels, que por sua vez é usada para escrever todas as linhas do ecrã. O interessante é que de cada vez que escreve um pixel ou uma linha troca o valor do pixel (entre desligado e vermelho), e o resultado é um padrão em xadrez, tal como se pode ver na figura seguinte.

O programa também ilustra a chamada de rotinas de mais baixo nível por outras de mais alto nível.

O programa poderia ser otimizado mudando a definição de linha apenas quando se muda de linha e não em todos os pixels. Seria uma solução mais eficiente, mas menos genérica.

Termine a execução do programa, carregando no botão **Stop** (■) do PEPE.



Usando o mesmo mecanismo, é possível por mudar a cor (incluindo a transparência) pixel a pixel. De cada vez que se muda a cor, os pixels escritos a partir daí passam a ter essa nova cor.

Também é possível alterar 8 ou 16 pixels de uma só vez, com um só comando, mas todos têm de ter a mesma cor, a da caneta, previamente definida. Cada bit a 0 desliga o pixel e cada bit a 1 coloca o pixel respetivo com a cor da caneta. Os pixels alterados contam-se a partir da maior posição múltipla de 8 ou 16 que não é maior do que a posição corrente.

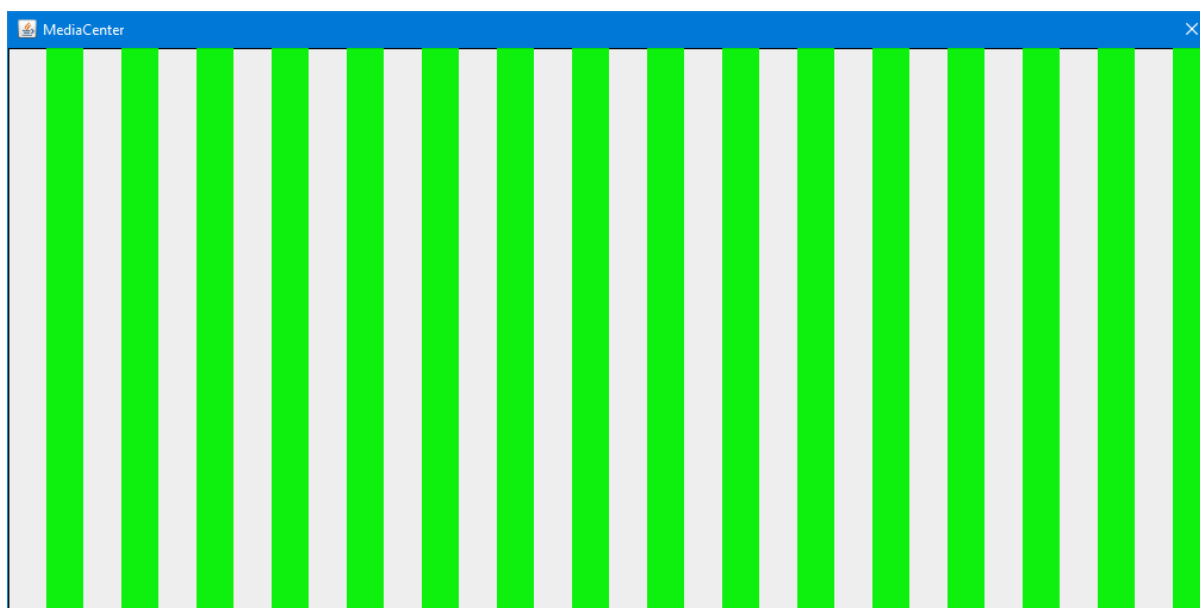
Para ilustrar esta funcionalidade, carregue agora o programa **lab4-comandos-barras.asm**. Este programa define a posição inicial onde começar a escrever os pixels e liga o mecanismo de *auto-increment*, que avança automaticamente a posição corrente do número de pixels escritos. Em cada iteração, escreve o byte 0CCH (1100 1100, em binário), e faz 256 iterações, dado que o ecrã tem 256 conjuntos de 8 pixels (32 linhas * 8 x 8 pixels por linha).

Execute o programa e verifique que obtém o ecrã da figura seguinte (a cor foi mudada para verde). Note que é bastante mais rápido a executar do que no caso anterior, pois escrevem-se 8 pixels (um byte) de cada vez, enquanto no caso anterior se tinha de escrever pixel a pixel.

Note também que os dois 0s de menor peso do byte estão à esquerda, pois a coluna cresce da esquerda para a direita e o bit de menor peso de cada byte é escrito no pixel de menor ordem no ecrã (mais à esquerda).

Experimente executar de novo o programa, mas depois de mudar o valor da constante **BYTE_PADRAO** para, por exemplo, 55H, ou 0AAH. Explique o padrão resultante.

NOTA – No *assembly* do PEPE, as constantes hexadecimais começadas por uma letra têm de levar um “0” antes, para que sejam interpretadas como um valor numérico e não como um nome.



3.4 – Escrita de pixels por acesso à memória do ecrã

Em vez de dar comandos ao ecrã, também é possível escrever diretamente na sua memória, que neste ecrã de 64 colunas por 32 linhas é constituída por 2048 palavras (uma palavra por cada um dos 64 x 32 pixels). A primeira palavra está localizada no endereço 8000H e a última em 8FFEh, de acordo com o mapa de endereços indicado atrás.

Os programas equivalentes ao **lab4-comandos.asm** e **lab4-comandos-xadrez.asm**, mas com escrita dos pixels na memória, estão contidos nos ficheiros **lab4-memoria.asm** e **lab4-memoria-xadrez.asm**, respetivamente, que também são fornecidos.

No entanto, os comandos são essenciais para diversas funcionalidades, como apagar o aviso de nenhum cenário de fundo selecionado ou até para limpar o ecrã dos pixels já escritos. O programa do ficheiro **lab4-memoria-xadrez.asm**, embora escreva os pixels na memória, usa comandos para estas funcionalidades. Edite os programas para observar o código e execute-os no PEPE para verificar o seu funcionamento.

Com escrita direta em memória apenas é possível escrever um pixel de cada vez, razão pela qual o programa das barras não tem equivalente com escrita em memória. O mesmo padrão pode naturalmente ser conseguido, mas escrevendo um pixel de cada vez em memória.

3.5 – Múltiplos ecrãs de pixels

Na janela de configuração do módulo MediaCenter (accedida com clique duplo no MediaCenter em modo “Design” – ver secção 3.2), é possível especificar o número de ecrãs que se pretendem (até 16).

Cada ecrã está num plano diferente e a vantagem é que os bonecos que se desenhavam num ecrã podem “passar” por cima ou por baixo de bonecos noutros ecrãs, sem interferência.

Depois de definir o número de ecrãs, é possível o programa seleccionar o ecrã onde os próximos pixels irão ser desenhados, bem como esconder ou mostrar um ecrã inteiro. Tal faz-se por meio de comandos (ver secção 3.3).

3.6 – Cenários de fundo e frontal

Por trás de todos os ecrãs está uma janela que pode ter uma imagem, que atua como cenário de fundo. Na frente está outro

Na janela de configuração do módulo MediaCenter (accedida com clique duplo no MediaCenter em modo “Design” – ver secção 3.2), é possível as imagens que se pretendem (até 32), que são automaticamente esticadas para acompanhar a resolução da janela de simulação do MediaCenter. Depois no programa é possível seleccionar a que se quer como cenário, e ir mudando de acordo com os requisitos do programa, por meio de comandos (ver secção 3.3).

As extensões de imagem suportadas são as seguintes: "jpg", "png", "bmp" e "gif". No entanto, outras poderão ser suportadas, dependendo da plataforma.

É ainda possível definir um padrão de transição entre cenários, com fade-out do anterior e fade-in do novo seleccionado.

3.7 – Sons e vídeos

Na janela de configuração do módulo MediaCenter (accedida com clique duplo no MediaCenter em modo “Design” – ver secção 3.2), é também possível definir sons e vídeos (até 32). Os vídeos são automaticamente esticados para acompanhar a resolução da janela de simulação do MediaCenter e estão em planos entre o cenário de fundo e os ecrãs de pixels.

Depois no programa é possível seleccionar os sons e os vídeos e proceder à respetiva reprodução. Existem vários comandos para este efeito (ver secção 3.3).

As extensões de áudio e vídeo suportadas são as seguintes: "aif", "aiff", "mp3", "wav", "mp4", "m4a" e "m4v".

Na janela de configuração é ainda possível obter informação sobre cada som e vídeo, e até definir os pontos de início e fim do tempo de reprodução.

A escolha “*as clip*” deve ser reservada para pequenos sons, tipicamente efeitos sonoros muito curtos (1 segundo, ou assim). Ocupam mais memória que os sons “não clip”, mas a grande vantagem é a latência (começam a reproduzir mais rapidamente) e podem ser recomeçados antes de acabarem. Em compensação, não se podem terminar a meio, nem fazer pausa, como os outros.

Os ficheiros de áudio e vídeo são carregados quando se passa o simulador para “Simulation”. Embora raro, acontece por vezes o carregamento de um destes ficheiros falhar, caso em que aparece uma mensagem de erro e esse ficheiro não poderá ser reproduzido. Basta passar para “Design” e de novo para “Simulation”.

4 – Considerações em relação ao Projeto

O módulo MediaCenter é fundamental para o projeto, com imagens para os cenários de fundo, sons, vídeos e objetos desenhados nos ecrãs de pixels.

Imagens, sons e vídeo são ficheiros que basta, essencialmente, seleccionar/reproduzir.

Os objetos nos ecrãs de pixels, no entanto, têm de ser desenhados, pixel a pixel, de acordo com formas e cores previamente definidas. Alguns desses objetos até têm de se movimentar no ecrã!

Para gerir o desenhar e movimentar destes objetos, ficam as seguintes sugestões:

- Cada objeto tem em cada instante uma dada posição no ecrã (linha, coluna), que deve memorizar;
- Mover um objeto é apagá-lo na sua posição corrente, alterar essa posição e voltar a desenhá-lo na nova posição;
- A posição de um objeto é na realidade a posição de um dos seus pixels, que é tomado como referência (por exemplo, o canto superior esquerdo). Sabendo essa posição, os restantes pixels são desenhados relativamente a ela (na coluna a seguir, na linha de baixo, etc.). Mudar a posição de um objeto é mudar a posição do seu pixel de referência;
- Não desenhe um objeto “à mão”, com comandos ou escritas de pixels sucessivas na memória, com mudança manual da posição dos vários pixels do objeto!
- Descreva os vários pixels de um objeto por meio de uma tabela, que inclua a largura, altura e cores de cada um dos pixels. Programe uma rotina genérica que receba o endereço desta tabela, bem como a linha e a coluna do pixel de referência. A rotina deve então percorrer a tabela e ir desenhando o objeto, pixel a pixel. Esta rotina serve para desenhar qualquer objeto cuja descrição siga este formato;
- Note que as variáveis que memorizam a linha e a coluna do pixel de referência não devem, em regra, estar nesta tabela, pois o valor da posição pode variar, ao contrário da forma e cores. Também pode haver várias instâncias do mesmo objeto (descritas pela mesma tabela, mas em posições diferentes).