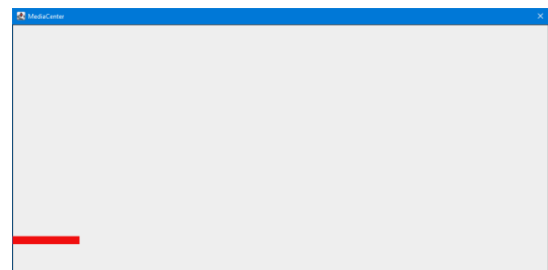
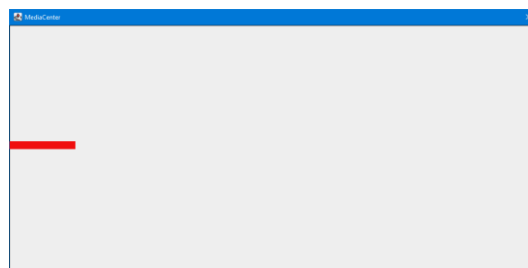
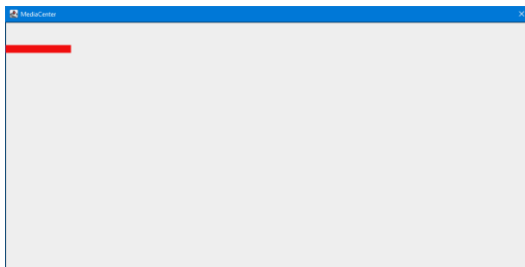


3 – Execução do trabalho de laboratório

3.1 – Temporizações por rotina de atraso

Passe para “Simulation” e faça clique no PEPE e no MediaCenter para abrir os respectivos painéis de simulação (se não estiverem já abertos).

Carregue o programa *assembly lab6-barra.asm*, com **Load source** (📁) ou *drag & drop*. Este programa anima uma barra com 8 pixels, que vai caindo até que chega ao fundo e recomeça do topo. A cor da barra é a da caneta, definida na janela de configuração do MediaCenter (em modo “Design”). As figuras seguintes ilustram três das sucessivas posições.



Por simplicidade, o programa escreve no ecrã ao nível do byte (com comandos), e não ao nível do pixel, pelo que a barra tem uma largura de 8 pixels.

Veja o código do programa num editor de texto, para perceber o seu funcionamento.

A cadência de queda foi implementada por meio de uma rotina que faz um ciclo de várias iterações, em número definido pela constante **DELAY**. O atraso que esta rotina provoca depende do desempenho do seu computador.

Execute o programa, carregando no botão **Start** (▶).

Afine a temporização (se necessário), aumentando ou diminuindo o valor de **DELAY** de modo a que a cadência de queda da barra seja aproximadamente de uma linha por cada segundo. Quanto mais rápido for o seu computador, menor terá de ser o valor do atraso.

De cada vez que quiser alterar, e também no fim, termine a execução do programa, carregando no botão **Stop** (⏏) do PEPE e depois no botão **Reload** (🔄), uma vez que o nome

do ficheiro do programa é o mesmo. Não se esqueça de antes fazer **Save** no editor de texto do programa após cada alteração.

3.2 – Temporizações múltiplas por rotina de atraso

Imagine agora que quer implementar quatro barras a cair, cada uma na sua coluna e cada uma com uma temporização duas vezes mais rápida (metade do atraso) do que a anterior. Para esse efeito, veja o código do programa *assembly lab6-quatro-barras.asm* num editor de texto, para perceber o seu funcionamento.

Verifique que agora a rotina que anima a barra é chamada quatro vezes no programa principal, uma para cada coluna de 8 pixels e com um tempo de atraso sucessivamente metade do da coluna anterior.

Carregue no PEPE o programa *assembly lab6-quatro-barras.asm*, com **Load source** (📁) ou *drag & drop*. Execute o programa, carregando no botão **Start** (▶).

Seria de esperar que as barras de deslocassem a velocidades diferentes, mas o facto é que tal não acontece!

Porquê? Consegue explicar o que observa?

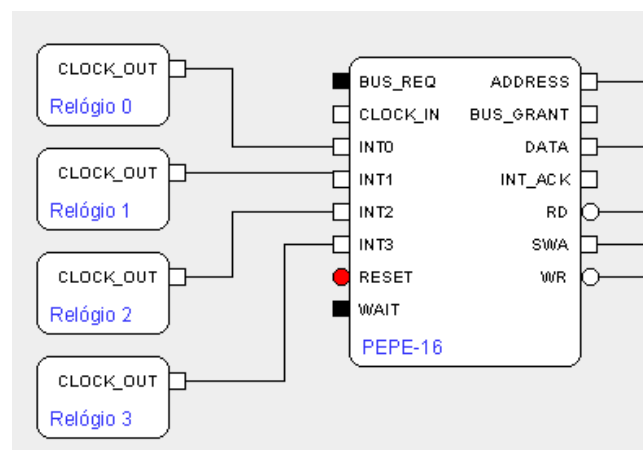
Termine a execução do programa, carregando no botão **Stop** (⏏) do PEPE.

3.3 – Temporizações por interrupção

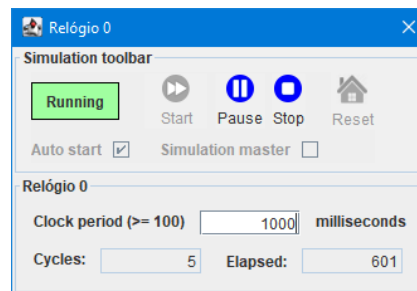
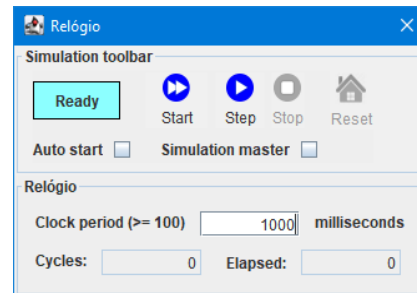
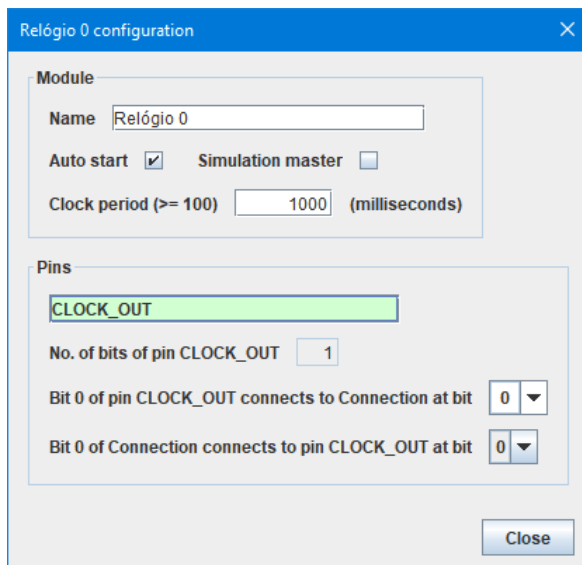
Com as temporizações por atraso com um ciclo de instruções não se conseguem tempos rigorosos (tempo real), além de que várias temporizações interferem entre si. O próximo atraso só pode começar depois de o anterior acabar. Ou seja, estão em série!

A forma de resolver este problema é usar temporizadores (relógios) de tempo real, circuitos externos ao PEPE que geram formas de onda periódicas cujo período pode ser definido em termos do tempo real, sem depender do desempenho do computador que simula o PEPE nem interferir com outras temporizações.

Para que o programa se possa sincronizar com essas formas de onda, o PEPE disponibiliza quatro pinos de interrupção. A figura seguinte reproduz a parte relevante do circuito da página 1 deste guião (ou do próprio simulador). Os quatro pinos de interrupção do PEPE (INT0 a INT3) ligam a outros tantos relógios de tempo real (Relógio 0 a Relógio 3).



Estes relógios geram uma forma de onda quadrada, cujo período é definido no seu painel de interface, quer em configuração (modo “Design”), quer em simulação. Por exemplo, o Relógio 0 tem definido um período de 1000 milissegundos, ou 1 segundo:



Com a interface de simulação (lado direito, em dois estados diferentes), é possível colocar o relógio em execução contínua (▶), em pausa (⏸) ou executar um ciclo de relógio de cada vez (▶), ou ainda pará-lo (⏹). A etiqueta antes dos botões indica o estado do relógio. A interface de simulação indica ainda quantos ciclos de relógio já foram gerados e quanto tempo já passou do ciclo de relógio corrente.

É também possível especificar se o relógio começa automaticamente quando a simulação se inicia (quando se executa o programa no PEPE) ou se o relógio tem de ser iniciado manualmente. Se o relógio tiver arranque automático, nem é preciso abrir a sua interface de simulação. Começa a funcionar mal se passe para “Simulation”.

No início de cada ciclo de relógio, quando o sinal gerado por um relógio passa de 0 para 1, é gerada uma interrupção para o PEPE, relativa ao pino a que o relógio estiver ligado.

No caso da interrupção 0, causada pela transição de 0 para 1 no Relógio 0, a rotina invocada por esta interrupção irá fazer descer a 1ª barra (mais à esquerda) de uma linha. Como o sinal gerado pelo relógio é periódico, com período de 1 segundo, a barra acaba por descer uma linha por segundo de forma rigorosa, quer num computador mais rápido, quer num mais lento.

Com o editor de texto abra o programa *assembly* **lab6-barra-interruptcao.asm**. Este programa é idêntico ao do ficheiro **lab6-barra.asm**, exceto que a temporização agora é feita por interrupção e não por uma rotina de atraso. Verifique:

- A definição da tabela das rotinas de interrupção (só uma, neste caso);
- A inicialização do registo **BTE** (Base da Tabela de Exceções);
- A permissão da interrupção 0 (**EI0** e **EI**);

- A rotina de interrupção **rot_int_0** (que tem obrigatoriamente de terminar com **RFE**, e não **RET**), que invoca a rotina **anima_barra**, para fazer descer a barra de uma linha.

Para verificar o comportamento das interrupções:

- Carregue o programa *assembly lab6-barra-interruptao.asm*, com **Load source** (📁) ou *drag & drop*;
- Execute o programa, carregando no botão **Start** (▶) do PEPE.
- Abra a interface de simulação do Relógio 0, fazendo clique nele.

Verifique que a barra desce uma linha por segundo, e o programa já nem sequer tem a rotina de atraso. Verifique também, na interface do Relógio 0, que os ciclos do relógio já efetuados aumentam de segundo a segundo.

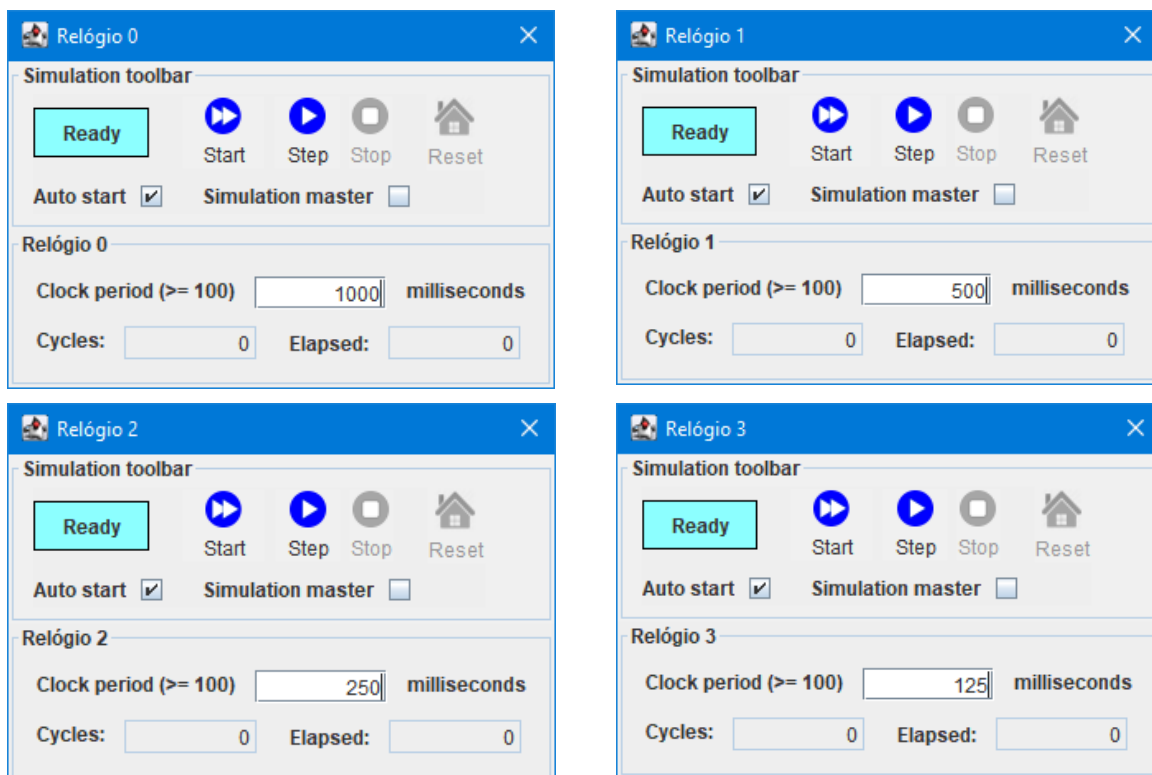
Termine a execução do programa, carregando no botão **Stop** (⏏) do PEPE.

3.4 – Temporizações múltiplas por interrupções

Usando várias interrupções (o PEPE permite até quatro) é possível implementar várias temporizações diferentes, sem interferência mútua.

Vamos usar os quatro relógios do circuito, cujos períodos estão definidos para 1000, 500, 250 e 125 milissegundos.

Estes relógios ligam às interrupções INT0 a INT3, respetivamente, sendo a primeira a mais lenta e a última a mais rápida.

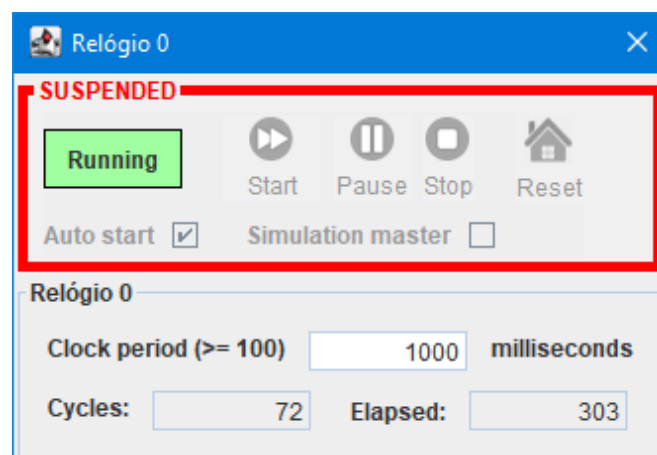






Com o editor de texto abra o programa *assembly lab6-quatro-barras-interruptoes.asm*. Verifique:

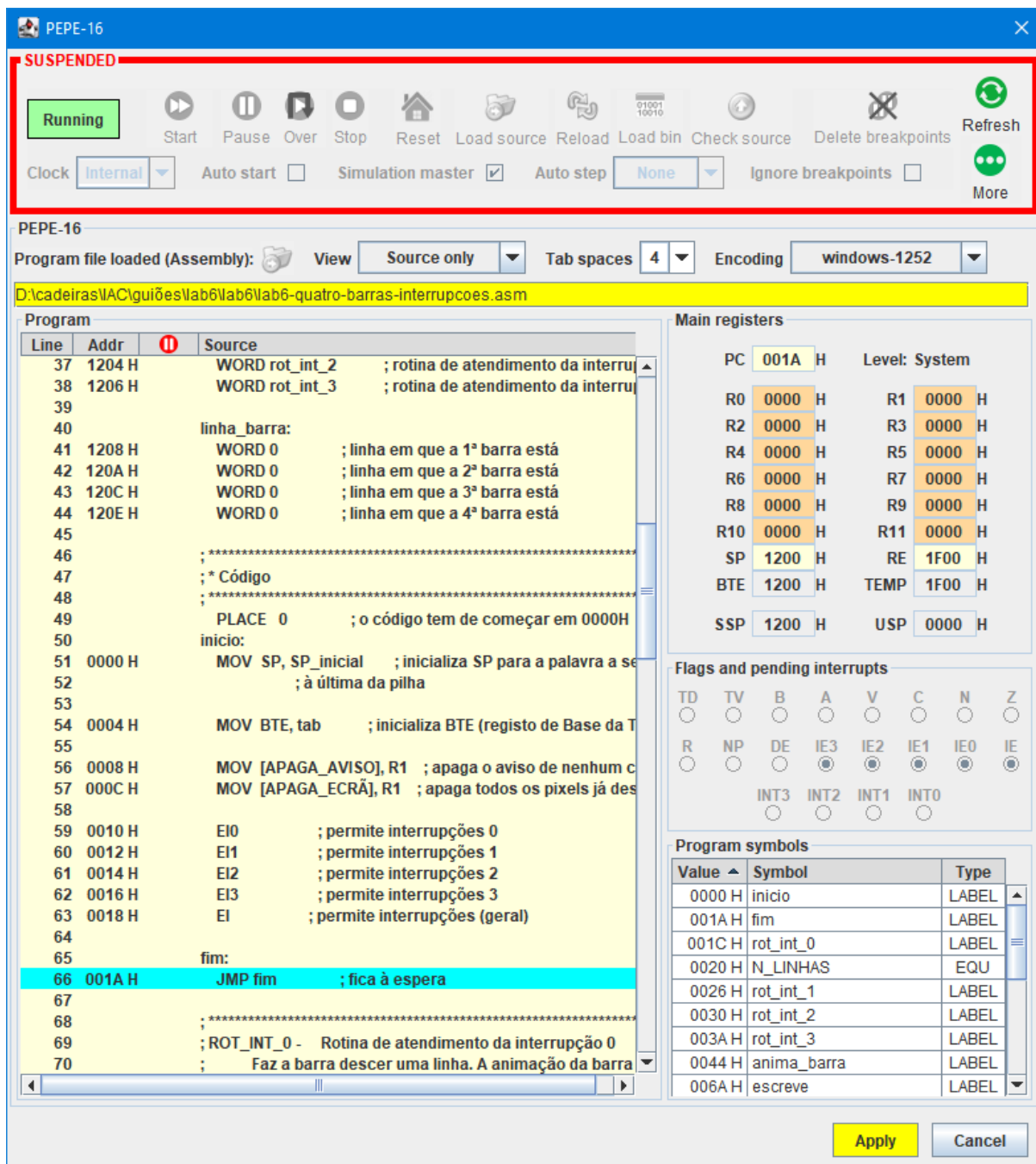
- A definição da tabela das rotinas de interrupção (agora são quatro);
- A inicialização do registo **BTE** (Base da Tabela de Exceções);
- A permissão das quatro interrupções 0 (**EI0** até **EI3**, e **EI**);
- As quatro rotinas de interrupção **rot_int_0** até **rot_int_3**. Cada uma destas rotinas invoca a rotina **anima_barra**, com um parâmetro (**R3**) que indica qual o número da barra (0 a 3) para fazer descer essa barra de uma linha;
- A definição da variável **linha_barra** como uma tabela de quatro **WORDS**, uma para cada uma das quatro barras (tabela indexada pelo número da barra, **R3**).

Para verificar o comportamento das interrupções:

- Carregue o programa *assembly lab6-quatro-barras-interruptoes.asm*, com **Load source** (📁) ou *drag & drop*;
- Execute o programa, carregando no botão **Start** (▶) do PEPE;
- Abra o painel de simulação dos quatro relógios;
- Verifique que as quatro barras descem, mas cada uma ao seu ritmo, de acordo com o período do respetivo relógio;
- Experimente colocar em pausa um dos relógios (botão ⏸ do relógio) e verifique que apenas a barra correspondente deixa de descer. Carregando no botão **Resume** (▶), a barra recomeça a descer;
- Experimente agora colocar de novo um dos relógios em pausa (botão ⏸ desse relógio). Vá carregando no botão de um só ciclo (🔄) e verifique que consegue descer manualmente a barra respetiva, que desce uma posição por cada vez que carrega na tecla do impulso. Pode recomeçar o funcionamento contínuo desse relógio carregando no botão **Resume** (▶);
- Experimente agora colocar o PEPE em pausa (botão ⏸ do PEPE). Note que todos os relógios ficam não em Pausa mas sim Suspensos, indicando que pararam por uma razão que lhes é externa. Recomeçando o PEPE, todos os relógios recomeçam também. A figura seguinte ilustra o aspeto de um relógio suspenso;



- Experimente fazer ir fazendo Step no PEPE (botão **Step**  do PEPE). Note que os relógios recomeçam e param logo, com o tempo “Elapsed” a evoluir um milissegundo de vez em quando. Isto quer dizer que o Step pausa também o tempo real nos relógios do simulador, permitindo fazer simulação passo a passo sem afetar significativamente as temporizações de tempo real do programa;
- Isto funciona assim porque o PEPE está definido como “Simulation master” e controla a simulação dos restantes dispositivos (esta definição pode alterar-se). Qualquer dispositivo com uma toolbar de simulação pode ser “Simulation master”;
- Experimente por exemplo fazer **Stop** () no Relógio 0 e coloque-o como “Simulation master”. Agora, sempre no Relógio 0, faça **Restart** () e a seguir **Pause** (). Note que o próprio PEPE é suspenso:



- Agora faça **Step** (▶) no Relógio 0 e verifique que o PEPE corre durante 1 segundo, que é o tempo do período (e por conseguinte de um *step*) do relógio, e é suspenso no fim desse período (é o relógio como “Simulation master”, a controlar a simulação).

No fim, termine a execução do programa, carregando no botão **Stop** (■) do PEPE. Retire a indicação de “Simulation master” do Relógio 0.

As interrupções são independentes. Quando uma ocorre, invoca a rotina correspondente, sem interferência de temporização com as restantes, ao contrário do que sucedia com as temporizações ativas (por ciclo de instruções, numa rotina de atraso) na secção 3.2.

Se duas interrupções sucederem simultaneamente, a mais prioritária é invocada primeiro e a outra logo a seguir. INT0 é a mais prioritária e INT3 a menos prioritária.

3.5 – Interrupção sensível ao nível

Por omissão, os pinos de interrupção (INT0 a INT3) são sensíveis ao flanco ascendente, o que significa que a interrupção é pedida ao PEPE quando o sinal num destes pinos tem a transição de 0 para 1.

Desta forma, consegue-se pedir apenas uma interrupção por cada período do sinal de relógio ligado ao um pino de interrupção. No entanto, o PEPE permite também configurar a sensibilidade de cada pino de interrupção para o flanco descendente e ainda para o nível 0 ou nível 1.

Quando um pino é sensível ao nível 1, por exemplo, a interrupção mantém-se pedida enquanto o sinal estiver a 1. Desta forma, a rotina de interrupção é invocada e, se o pino de interrupção continuar a 1 quando retorna, a rotina de interrupção é invocada novamente, e assim sucessivamente enquanto o pino estiver a 1.

Como exemplo, para configurar a interrupção 0, tornando-a agora sensível ao nível 1, basta escrever 2 no **RCN** (Registo de Configuração do Núcleo), antes de permitir as interrupções. As restantes interrupções mantêm-se sensíveis ao flanco. Os detalhes podem ser vistos na tabela A.4 do livro.

Carregue o programa *assembly lab6-quatro-barras-int0-nivel.asm*, com **Load source** (📁) ou *drag & drop*. Este programa faz esta configuração no início do programa principal.

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

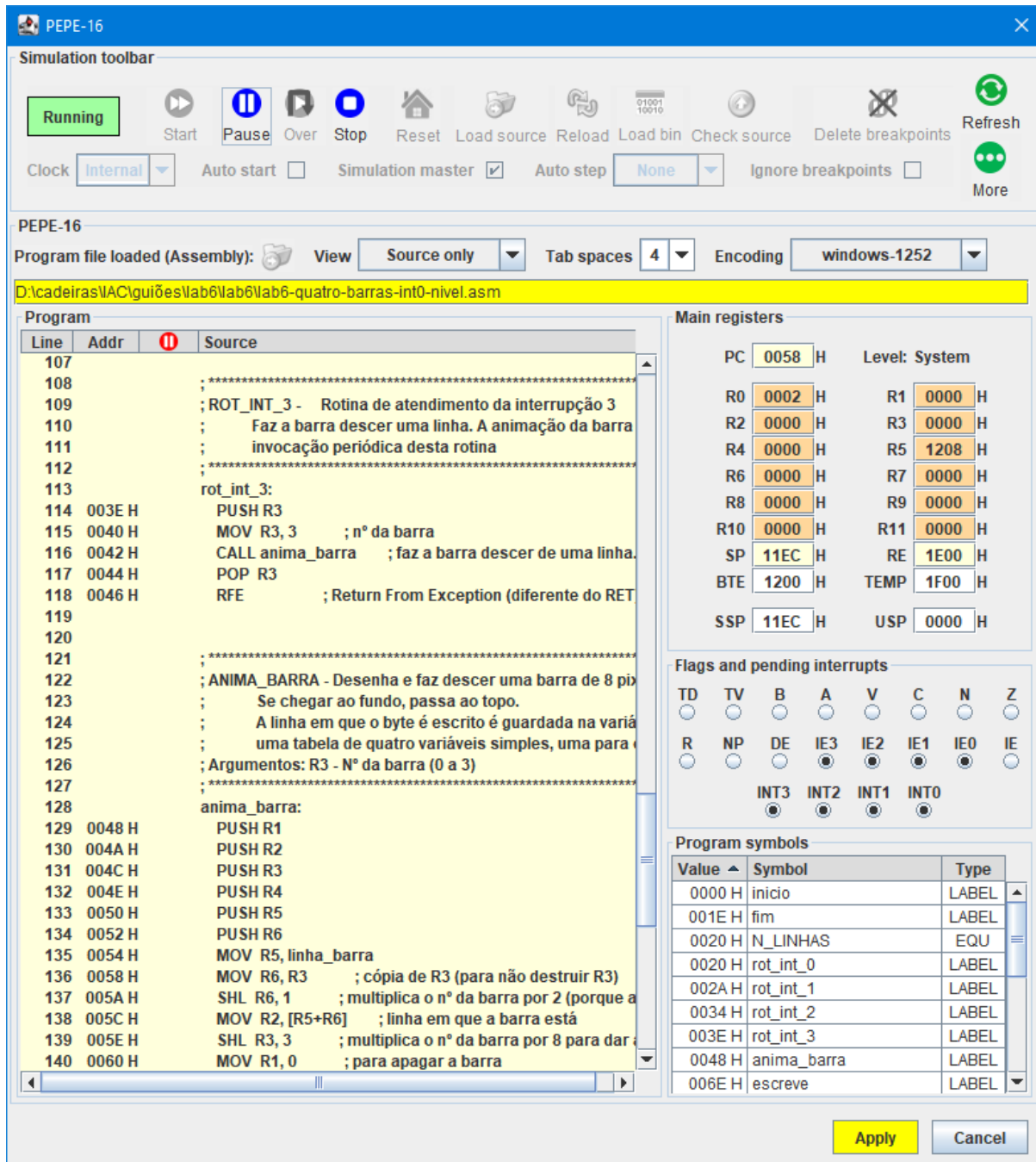
Explique o que é que se passa com a barra mais à esquerda, correspondente à interrupção 0.

NOTA – Enquanto a barra mais à esquerda se move rápido, as restantes param. Tal é uma consequência de a interrupção 0 ser mais prioritária que as restantes, pelo que é atendida em primeiro lugar, desde que esteja pedida. As restantes pedidas têm de esperar.

Este detalhe pode também ser visto na janela do PEPE, do lado direito. Por baixo das flags, está indicado o estado de cada um dos pedidos de interrupção (INT3 a INT0). Se uma destas flags estiver ativa, isso significa que foi feito o respetivo pedido de interrupção, mas este ainda não foi atendido.

Note que, enquanto o Relógio 0 tem a sua saída a 1 (durante 500 milissegundos), a flag INT0 está sempre ativa (mesmo que seja atendida, continua pedida, pois é sensível ao nível 1).

Das outras, a primeira a aparecer é tipicamente a 3 (mais frequente), seguida das restantes, e ficam a 1, pois sendo a interrupção 1 mais prioritária e estando sempre pedida, não conseguem ser atendidas. Quando o Relógio 0 passa para 0, já não há impedimento.



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

3.6 – Conjugação entre o programa principal e as interrupções

Nos programas anteriores com interrupções (secções 3.3 a 3.5), o programa principal limita-se a um salto para a própria instrução (**fim: JMP fim**).

No entanto, o programa principal pode continuar a fazer coisas úteis. A ideia é que normalmente o programa principal está a ser executado. Quando uma interrupção surge, a respetiva rotina é invocada, e quando retorna continua no ponto em que o programa principal tinha sido interrompido.

Para ilustrar este aspeto, com o editor de texto abra o programa *assembly lab6-quatro-barras-displays.asm*. Este programa é idêntico ao **lab6-quatro-barras-interruptoes.asm** (da secção 3.4), mas agora inclui um programa principal que faz contar um valor nos displays de 7 segmentos (Displays, na figura da página 1 deste guião). A temporização de contagem volta a ser feita por um ciclo de instruções, pela rotina **atraso**, que foi de novo introduzida, tal como na secção 3.1.

Carregue o programa *assembly lab6-quatro-barras-displays.asm*, com **Load source** (📁) ou *drag & drop*.

Abra o ecrã do MediaCenter, os displays de 7 segmentos e os painéis de todos os relógios.

Execute o programa, carregando no botão **Start** (▶) do PEPE. Note que os displays começam logo a contar e os relógios a gerar ciclos nos seus sinais de saída.

Coloque sucessivamente cada um dos relógios em pausa (botão ⏸ de cada relógio). Note que as respetivas barras deixam de descer. Quando todos os relógios estiverem em pausa, nenhuma barra desce, mas os displays continuam a contar.

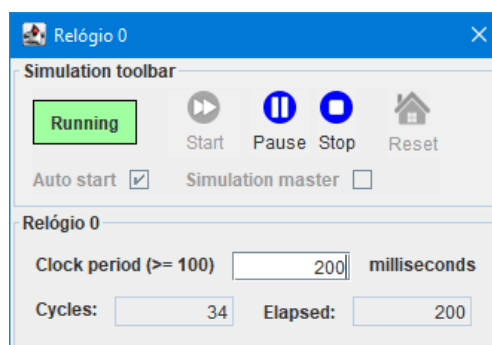
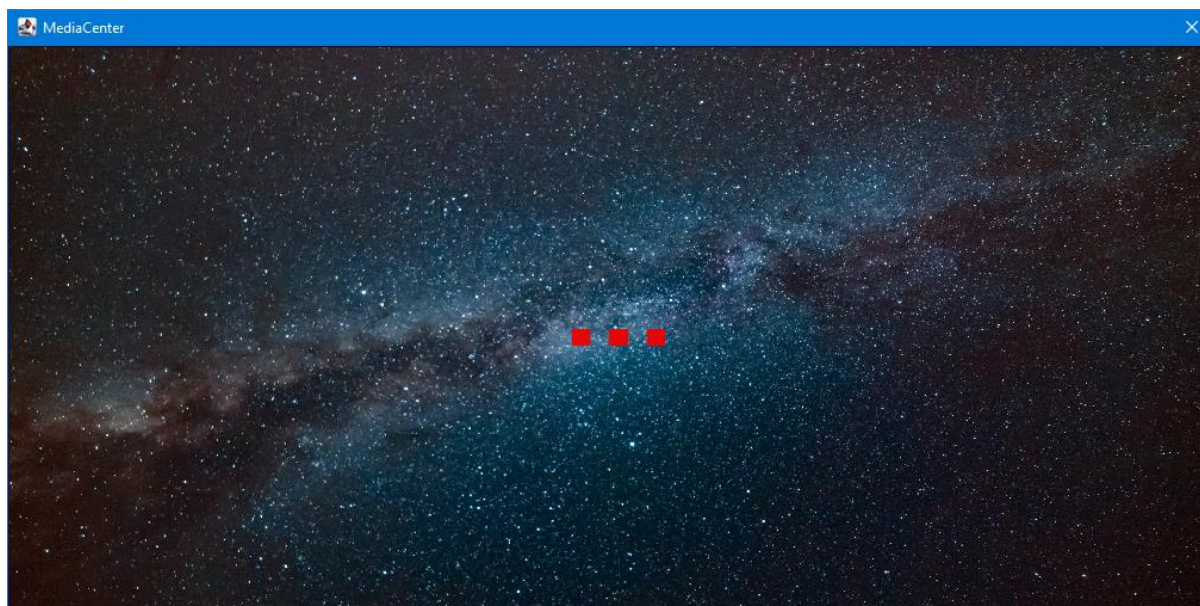
Volte a executar os relógios todos (botão ▶ nos relógios). Note que a rapidez de contagem dos displays praticamente não se altera, o que indica que o PEPE perde pouco tempo com a execução das rotinas de interrupção. A maior parte do tempo de execução é gasto com o programa principal, em particular no ciclo de espera da rotina **atraso**.

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

3.7 – Movimentar bonecos com interrupções

As barras são simples de desenhar, mas têm interesse visual reduzido. Carregue agora o programa *assembly lab6-move-boneco.asm*, com **Load source** (📁) ou *drag & drop*. Este programa implementa a funcionalidade já vista no programa **lab5-move-boneco.asm**, mas agora com a velocidade de movimentação estabelecida pela interrupção 0.

Execute o programa, carregando no botão **Start** (▶) do PEPE. Note a velocidade de movimentação do boneco (um movimento por segundo). Pode diminuir o período do Relógio 0 (para 200 milissegundos, por exemplo) e ver que o boneco anda mais rápido.

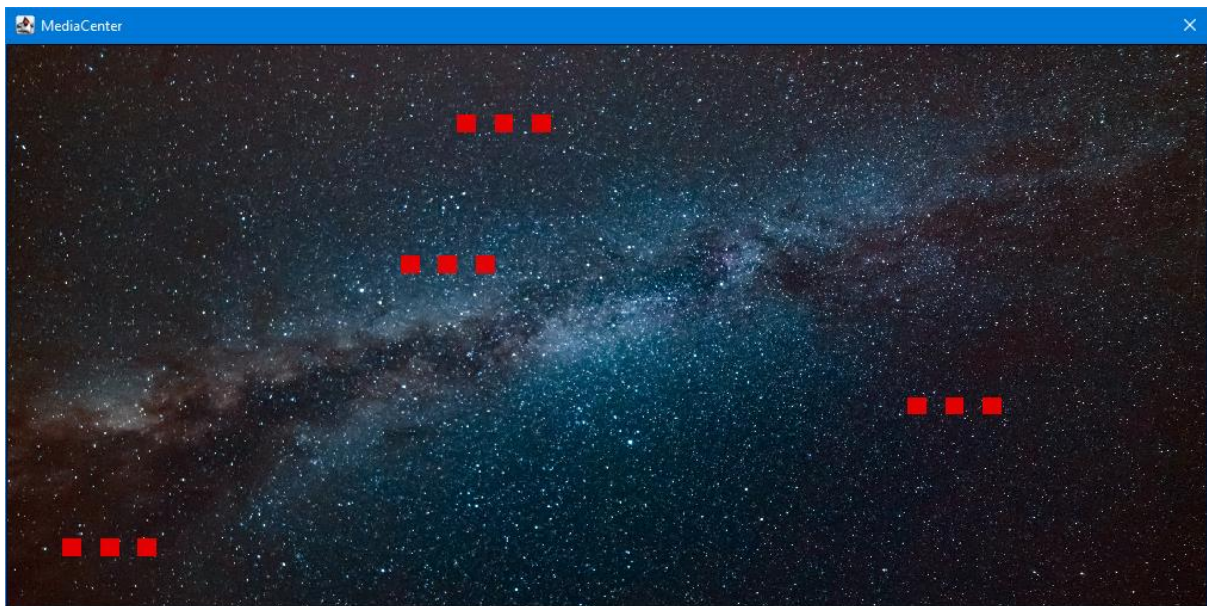


Termine a execução do programa, carregando no botão **Stop** (●) do PEPE.

3.8 – Movimentar vários bonecos com interrupções

Tal como foi possível animar várias barras, também é possível animar vários bonecos, usando as interrupções.

Carregue agora o programa *assembly* contido no ficheiro **lab6-move-quatro-bonecos.asm**, com **Load source** (📁) ou *drag & drop*. Este programa estende a funcionalidade do programa **lab6-move-boneco.asm**, movimentando um boneco por cada uma das interrupções, com períodos sucessivamente mais pequenos (por isso, o objeto de cima move-se uma vez por segundo, enquanto o de baixo se move aproximadamente 8 vezes por segundo).



Analise o programa e verifique que a rotina que anima cada um dos bonecos é sempre a mesma, tendo o número do boneco (0 a 3) como parâmetro e que é usado para aceder às tabelas que descrevem o estado de cada um dos bonecos (linha, coluna, sentido de movimento).

Seria até possível definir mais uma tabela com os endereços de 4 tabelas, cada uma a descrever um objeto de forma/cores diferentes, em vez de usar sempre a mesma tabela (`DEF_BONECO`).

3.9 – Nota importante sobre as interrupções

O mecanismo das interrupções é de muito baixo nível e deve usar-se apenas para detetar a ocorrência de eventos temporizados ou lidar diretamente com os periféricos (*device drivers*).

Assim, não devem ser as rotinas de interrupção a executar funcionalidade do programa, nomeadamente invocando rotinas de mais alto nível, como por exemplo as rotinas que animam as barras e os bonecos.

Note que, durante a execução da rotina de interrupção todas as interrupções são ignoradas. Se esta rotina fizer algo complexo que demore algum tempo, o programa pode deixar de fazer algumas coisas importantes, desencadeadas por outras interrupções.

Este guião não seguiu esta boa prática apenas por simplicidade, mas no guião de laboratório 7 mostrar-se-á como corrigir este problema.

No caso geral, uma rotina de interrupção deve apenas assinalar a ocorrência da interrupção, por exemplo colocando um dado valor (tipicamente, 1 ou algo diferente de 0) numa variável. Esta variável deve depois ser lida pelas rotinas de mais alto nível, que implementam a funcionalidade do programa. Desta forma, separam-se os mecanismos de baixo e alto nível e o programa fica melhor estruturado.

O guião de laboratório 7 mostra como implementar uma aplicação completa com processos (cada um implementando uma atividade, concorrente com outras) e qual a forma correta de implementar a ligação entre o baixo nível (periféricos, como o teclado, e os relógios, que geram interrupções) e o alto nível (processos que implementam a funcionalidade principal do programa).