

# Laboratório de Introdução à Arquitetura de Computadores

IST - Taguspark

2017/2018

## Introdução ao processador

### Guião 3

9 a 13 de outubro 2017

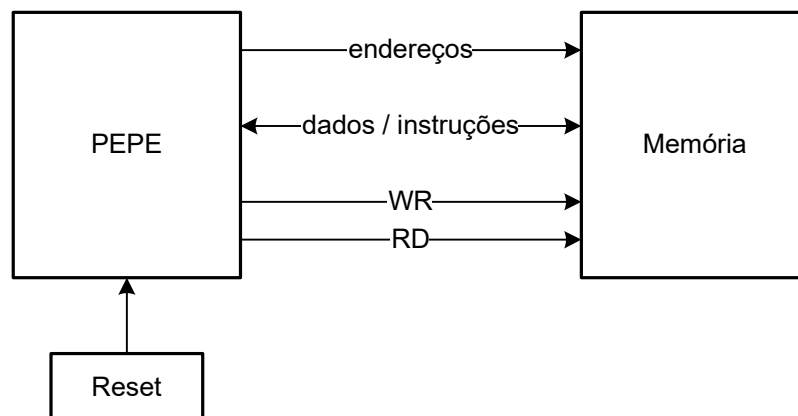
(Semana 4)

#### 1 – Objetivos

Com este trabalho pretende-se que os alunos se familiarizem com a interface do processador (PEPE) no simulador, por meio de um conjunto de instruções simples.

#### 2 – O circuito de simulação

Neste trabalho está em causa executar instruções que envolvam apenas os recursos internos do PEPE. Por conseguinte, o circuito a usar é muito simples e está representado na figura seguinte.



O módulo **Reset** destina-se a efetuar a inicialização do PEPE quando a simulação é iniciada.

O relógio do PEPE é gerado internamente, por omissão (embora possa ser configurado para ligar a um circuito exterior).


Os restantes pinos de entrada do PEPE e que não estão a ser usados estão forçados a 0.

A memória é de 16 bits e está adaptada ao PEPE.

### 3 – Execução do trabalho de laboratório

Inicie o simulador, carregue (no menu File, seguido de Load) o ficheiro **lab3.cmod** (que contém a descrição do circuito da figura da página anterior) e passe para Simulation. Um clique duplo no processador faz abrir a interface do PEPE, onde se pode ver o estado dos seus recursos internos (registos, nomeadamente, incluindo os bits de estado do processador).

#### 3.1 – Exemplos de instruções simples

Carregue no botão Compile&Load (  ) e abra o ficheiro **lab3-1.asm**, cujo conteúdo é o seguinte:


```
; *****
; *
; * Modulo:      lab3-1.asm
; * Descrição: Exemplos de instruções simples no PEPE.
; *             Ilustra igualmente o estilo de programação a usar em IAC.
; *
; *****
MOV R1, 5678H    ; constante de dois bytes
ROR R1, 8        ; roda até o byte mudar de sítio
MOV R2, 45H      ; constante de um byte
ADD R1, R2       ; exemplo de adição
MOV R3, R1       ; guarda resultado em R3
MOV R6, 8        ; para ciclo de 8 iterações
ciclo:
SHL R1, 1        ; desloca 1 bit para a esquerda
SUB R6, 1        ; decrementa contador e afeta bits de estado
JNZ ciclo        ; salta se ainda não tiver chegado a 0
SHL R3, 8        ; deslocamento de 8 bits para a esquerda
SUB R1, R3        ; o resultado dá sempre 0. Porquê?
fim:
JMP fim          ; forma expedita de "terminar"
```




Para ver o programa tal como aqui está (com indentação e comentários) use um editor de texto, como por exemplo o NotePad++ para PC ou o Brackets para Mac).

O assembler compila este código-fonte (texto) e gera código-máquina (números binários). Na janela de interface do PEPE aparece o código-máquina convertido de novo para instruções assembly, mas sem comentários.

Note que a instrução `MOV R1, 5678H` aparece dividida em duas `MOVL` e `MOVH`. Após a execução das duas, o valor correto fica em R1.

Pretende-se que:

- Execute as instruções uma a uma (passo a passo), com o botão STEP (  );
- Após a execução de cada instrução, verifique as alterações nos registos e se a instrução fez o esperado;
- Verifique a evolução do registo PC (*Program Counter*);
- Verifique os bits de estado, nomeadamente o bit Z;

- Sequenciamento da execução das instruções e os saltos, condicionais e incondicionais (verifique que `JNZ ciclo` salta para `ciclo`, mas apenas até R6 chegar a zero, e verifique que após chegar a `fim` não sai de lá);
- Termine a execução do programa, carregando no botão  do PEPE;
- Carregue de novo o programa, carregando no botão de Reload (). Coloque um *breakpoint* (ponto de paragem) na instrução com a etiqueta `ciclo` (`SHL`), simplesmente fazendo clique na linha da instrução, que fica a roxo) e outro na instrução `JNZ`. Execute o programa, no botão , e verifique que o programa para nos *breakpoints*, no mesmo estado que quando fez STEP.
- O que é que o programa faz globalmente? Qual o conteúdo de R3? O valor de R1 é sempre zero no fim do programa. Porquê?

Repita o que fez com o exemplo anterior agora com o ficheiro **lab3-2.asm**, cujo conteúdo é:

```
; *****
; *
; * Modulo:      lab3-2.asm
; * Descrição: Exemplos de instruções simples no PEPE.
; *             Ilustra igualmente o estilo de programação a usar em IAC.
; *
; *****
MOV R1, 4356H    ; constante de dois bytes
MOV R2, 21H      ; constante de um byte
SUB R1, R2       ; exemplo de subtração
MOV R5, 00FFH    ; máscara
AND R1, R5       ; elimina bits 8 a 15
XOR R1, R5       ; nega os bits 0 a 7
MOV R4, 0H       ; inicializa R4
ciclo:
MOV R6, 01H      ; máscara
AND R6, R1       ; força a 0 todos menos o bit de menor peso
JZ  salta        ; salta se for 0
ADD R4, 1        ; se não der zero adiciona 1
salta:
SHR R1, 1        ; deslocamento de um bit a direita
JNZ ciclo        ; salta se ainda não tiver chegado a 0
fim:
JMP fim          ; forma expedita de "terminar"
```

### 3.2 – Exemplos de programas com funcionalidade específica

De seguida carregue, compile, execute (passo a passo e com *breakpoints*) e tente perceber o funcionamento do programa 4.11 do livro, contido no ficheiro **programa4-11.asm**. Os comentários permitem perceber melhor cada instrução e ilustram o estilo de programação em linguagem assembly. O livro explica estes exemplos em maior detalhe.

A tabela A.9 do livro contém informação sobre cada instrução do PEPE.

```

; *****
; *
; * Modulo:      programa4-11.asm (do livro)
; * Descrição : Algoritmo para calcular o fatorial.
; *      Utilização dos registos:
; *      R1 - Produto dos vários fatores (valor do fatorial no fim)
; *      R2 - fator auxiliar que começa com N-1, depois N-2, ...
; *      ... até ser 2 (1 já não vale a pena).
; *
; * Nota : Verifique como se declaram constantes
; *****

; *****
; * Constantes
; *****
N    EQU 6      ; número de que se pretende calcular o fatorial

; *****
; * Código
; *****

início:
    MOV R1, N    ; valor inicial do produto
    MOV R2, R1   ; valor auxiliar
maisUm:
    SUB R2, 1     ; decrementa factor
    MUL R1, R2    ; acumula produto de fatores
    JV erro       ; se houve excesso, o fatorial tem de acabar aqui
    CMP R2, 2     ; verifica se o fator diminuiu até 2
    JGT maisUm    ; se ainda é maior do que 2, deve continuar
fim:
    JMP fim       ; acabou. R1 com o valor do fatorial
erro:
    JMP erro      ; termina com erro

```

Repita o que fez para o programa 4.11, mas agora para o programa 4.21 do livro, contido no ficheiro **programa4-21.asm**, verificando a sua funcionalidade.

```

; *****
; *
; * Modulo:      programa4-21.asm
; * Descrição : Algoritmo para contar '1s' numa constante.
; *      Utilização dos registos:
; *      R1 - valor cujo número de '1s' deve ser contado
; *      R2 - contador
; *
; * Nota : Verifique como se declaram constantes
; *****

; *****
; * Constantes
; *****
valor EQU 6AC5H ; valor cujos bits a 1 vão ser contados

; *****
; * Código
; *****

início:

```

```

        MOV R1, valor    ; inicializa registo com o valor a analisar
        MOV R2, 0        ; inicializa contador de número de 1s
        MOV R3, 0
maisUm:
        ADD R1, 0        ; isto é só para atualizar os bits de estado
        JZ fim          ; se o valor já é zero, não há mais 1s para contar
        SHR R1, 1        ; retira o bit de menor peso (fica no bit C-Carry)
        ADDC R2, R3       ; soma mais 1 ao contador, se esse bit for 1
        JMP maisUm       ; vai analisar o próximo bit
fim:
        JMP fim          ; acabou. R2 tem o número de bits a 1 no valor

```

### 3.3 – Exemplos de instruções de acesso à memória

Verifique agora o funcionamento das instruções de acesso à memória (MOV e MOVB), carregando no PEPE o ficheiro **lab3-3.asm**, cujo conteúdo é:

```

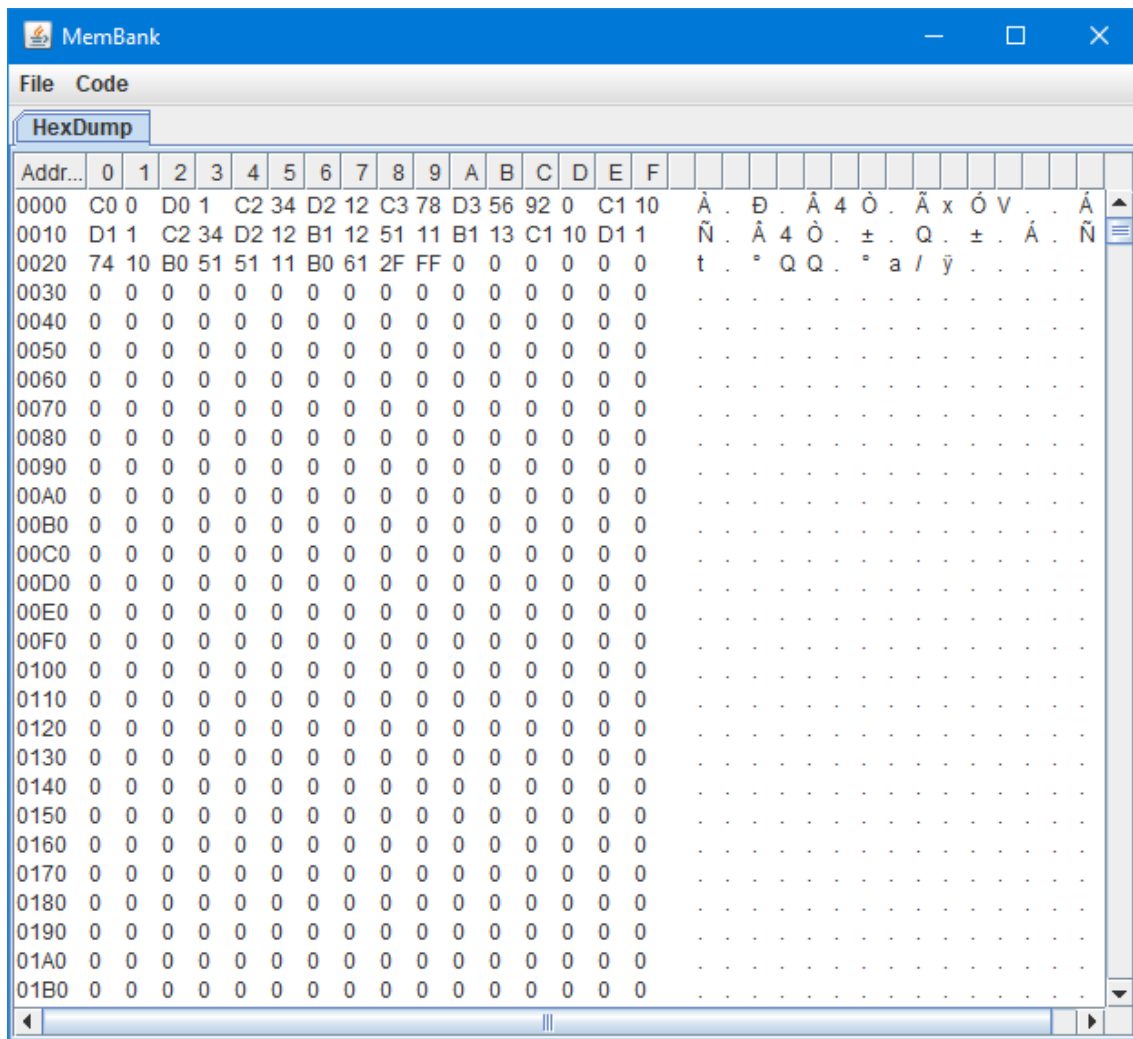
; *****
; *
; * Modulo:      lab3-3.asm
; * Descrição: Exemplos de instruções de acesso à memória,
; *             em word e em byte.
; *
; *****
        MOV R0, 0100H    ; endereço da célula de memória a aceder em word
        MOV R2, 1234H    ; constante de dois bytes
        MOV R3, 5678H    ; constante de dois bytes
        MOV [R0], R2     ; escreve uma word (16 bits) na memória
        MOV R1, 0110H    ; endereço da célula de memória a aceder em byte
        MOV R2, 1234H    ; constante de dois bytes
        MOVB [R1], R2    ; escreve um byte (8 bits) na memória (0110H)
        ADD R1, 1        ; endereço 0111H
        MOVB [R1], R3    ; escreve um byte (8 bits) na memória (0111H)
        MOV R1, 0110H    ; repõe endereço 0110H em R1
        MOV R4, [R1]     ; lê uma word (16 bits) da memória (endereço par)
        MOVB R5, [R1]    ; lê um byte (8 bits) da memória (0110H)
        ADD R1, 1        ; endereço 0111H
        MOVB R6, [R1]    ; lê um byte (8 bits) da memória (0111H)
fim:
        JMP fim          ; forma expedita de "terminar"


```

Este programa tem as suas instruções localizadas a partir do endereço 0000H (atribuído automaticamente pelo assembler do programa) e acede à memória nos endereços 0100H, 0110H e 0111H.

Os acessos em palavra (word, 16 bits, instrução MOV) só podem ser feitos a endereços pares, enquanto os acessos em byte (8 bits, instrução MOVB), podem ser feitos a qualquer endereço.

Faça duplo clique na memória (MemBank), o que faz abrir o respetivo painel de controlo (note os bytes do programa nos endereços a partir de 0000H):



Execute o programa passo a passo, com o botão STEP (  ). Após cada instrução, verifique o conteúdo dos registos envolvidos e o valor da memória (neste painel) nos endereços, 0100H, 0110H e 0111H, de forma a perceber o funcionamento das instruções MOV e MOVB.

Note que só as instruções com parênteses retos acedem realmente à memória. Os restantes MOVs destinam-se apenas a carregar constantes nos registos.