Escola Superior de Tecnologia e Gestão
Instituto Politécnico de Viana do Castelo

Arquitectura e Sistemas de Computadores

2020/2021

Trabalho Prático Laboratorial 2

Introdução ao simulador MARS
Introdução à programação em assembly

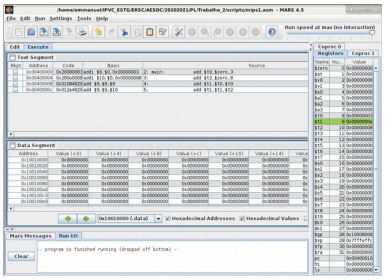
Emanuel Lomba
emanuellomba@estg.ipvc.pt

Novembro 2020
(adaptação da *Ficha Prática nº 2* de ASC-ERSC, sil@estg.ipvc.pt, 2019)

1. Objectivos

Introdução ao simulador MARS
Entender os conceitos básicos da arquitectura de um processador MIPS
Introdução à programação em assembly - Estrutura de um programa e instruções básicas no MIPS

2. Material necessário (além do computador)



2.1 **Simulador MARS**: O MARS (*MIPS Assembler and Runtime Simulador*) é um simulador da arquitectura MIPS, desenvolvido em JAVA, na Universidade do Missouri (EUA). O download do simulador pode ser feito na seguinte ligação:

http://courses.missouristate.edu/KenVollmar/MARS/index.htm

Para mais informação relativa ao MARS, recomenda-se a leitura do seguinte artigo.

Por ter sido desenvolvido em JAVA, este simulador corre em sistemas Linux, Mac ou Windows.

3. Aritmética simples em *Assembly*

3.1 Pretende-se escrever um programa em assembly que implemente a seguinte expressão aritmética:

$$y = 2x + 8$$

Supondo que o valor de **x** é passado através do registo **$t0**, e que o resultado e colocado no registo **$t1**, uma possível solução é:

```
                .text

main:           add   $t0,$zero,val_x       # $t0=x (substituir val_x pelo
                                            # valor de x pretendido)
                add   $t2,$zero,8           # $t2=8
                add   $t1,$t0,$t0           # $t1=x+x=2x
                add   $t1,$t1,$t2           # $t1=y=2x+8
```

3.1.a Edite o programa com o editor do MARS e substitua "val_x" pelo valor com que pretende efectuar o cálculo (por exemplo **3**).

3.1.b Compile o programa (opção *Run* → *Assemble*). Se for assinalado algum erro de sintaxe, corrija o erro e repita a compilação.

3.1.c Execute o programa (opção *Run* → *Go*). Observe e anote no seu caderno, o resultado presente no registo **$t1**. Repita os procedimentos anteriores para outros dois valores de **x**.

3.1.d Preencha a tabela seguinte com o endereço de memória e o código máquina de cada uma das instruções do programa.

Endereço de Memória	Código Máquina	Instrução *Assembly*

3.1.e Coloque um *breakpoint* na segunda instrução do programa. Faça o reset ao sistema (opção *Run* → *Reset*)e execute novamente o programa - a execução vai parar na instrução onde colocou o *breakpoint*. Execute a parte restante do programa passo a passo (opção Run → Step) e preenche a tabela com os valores que os vários registos vão tomando.

PC	Instrução *Assembly*	$t0	$t1	$t2

3.2 Altere o programa que escreveu em 3.1, de modo a implementar a expressão aritmética:

$$y = 2x - 8$$

3.2.a Execute o programa para **x=2**, **3**, **4** e **5** e observe os resultados no registo **$t1**. Interprete o resultado de **y** para **x=2** e **x=3**. Anote os resultados no seu caderno.

3.2.b Proceda do modo descrito na alínea *3.1.e* e preencha a tabela seguinte na situação em que x=3

PC	Instrução *Assembly*	$t0	$t1	$t2

4. *System calls* no MARS

Na solução adoptada nos exercícios anteriores, a atribuição do valor de **x** faz parte da codificação do programa. A alteração do valor de **x** pressupõe a edição do código fonte e a geração de novo código máquina, ou seja, **x** é encarado pelo programa como uma constante. Também a observação do resultado tem que ser efectuada directamente no registo do CPU.

Neste exercício vão ser utilizadas funções de interacção com o utilizador (normalmente designadas por *system calls*) para permitir a leitura do valor de **x** a partir do teclado (durante a execução do programa) e a apresentação do correspondente valor de **y**.

O MARS disponibiliza cerca de 50 *system calls*, com diferentes funcionalidades (ver lista de *system calls* aqui). As *system calls* são chamadas através da colocação no registo **$v0** do CPU, do número identificador respectivo, seguida da instrução *syscall*. Por exemplo, para a leitura de um valor inteiro do teclado, pode ser usada a *system call* **read_int()** através da seguinte sequência de instruções:

```
add $v0,$zero,5 # a system call read_int() é identificada
                # com o número 5 (ver tabela de instruções)
syscall         # a system call read_int() é chamada
```

Para a *system call* **read_int()**, o resultado do valor lido do utilizador é devolvido através do registo **$v0** do CPU. Para visualizar o conteúdo de um registo do CPU no ecrã pode ser usada a *system call* **print_int10()**; nesse caso o valor que se pretende visualizar no ecrã é passado através do registo **$a0**, pelo que, para além da inicialização do registo **$v0** com o valor identificador do **print_int10()** é necessário copiar para o registo **$a0** o valor a imprimir. Por exemplo, a impressão do valor do registo **$t5** no ecrã pode ser efectuada através da seguinte sequência de instruções:

```
add  $a0,$zero,$t5   # copia o registo $t5 para o registo $a0
addi $v0,$zero,1     # a system call print_int10() é identificada
                     # com o número 1 (ver tabela de instruções)
syscall              # a system call print_int10() é chamada
```

4.1.a Faça as alterações ao programa que escreveu no ponto 3.2, de modo a ler do teclado o valor de **x** e a imprimir no ecrã o resultado do cálculo de **y**.

4.1.b Execute o programa para diferentes valores de **x** e observe, em particular, o resultado para x=2 e x=3. Anote os resultados no seu caderno.

4.1.c Altere novamente o programa de modo a usar a *system call* **print_int16()**. Execute o programa para diferentes valores de **x** e observe, em particular, o resultado para x=2, 3, 4 e 5. Anote os resultados no seu caderno.

4.1.d Utilize, finalmente, a *system call* **print_intu10()**. Execute o programa e observe o resultado para x=2, 3, 4 e 5. Anote os resultados no seu caderno.

FIM.