

## AULA PRÁTICA N.º 2

### Objectivos:

Utilização de instruções lógicas e de deslocamento sobre inteiros no MIPS. Utilização de directivas *assembler*.

### Conceitos básicos:

- Lógica *bitwise* e operações com máscaras. Instruções lógicas.
- Deslocamento (*shift*) lógico e aritmético. Instruções de deslocamento.
- Directivas do *assembler*.

### Guião:

#### 1. Instruções lógicas

- a) Codifique um programa em *assembly* do MIPS que determine o resultado das operações lógicas bit a bit (*bitwise*) AND<sup>1</sup>, OR, NOR e XOR, considerando como operandos os registos **\$t0** e **\$t1**; os resultados devem ser armazenados nos registos **\$t2**, **\$t3**, **\$t4** e **\$t5**, respectivamente.
- b) Execute o programa no MARS, introduzindo previamente os valores dos operandos nos registos **\$t0** e **\$t1**, para os seguintes pares:

```
(0x12345678, 0x0000000F)
(0x12345678, 0x000FF000)
(0x762A5C1B, 0x89D5A3E4)
```

- c) Preencha (no seu *logbook*) a tabela seguinte e confirme manualmente os resultados para cada um dos pares de valores de entrada.

\$t0	\$t1	\$t2 (AND)	\$t3 (OR)	\$t4 (NOR)	\$t5 (XOR)
0x12345678	0x0000000F				
0x12345678	0x000FF000				
0x762A5C1B	0x89D5A3E4				

- d) O MIPS não disponibiliza uma instrução de negação bit a bit. Usando as instruções lógicas disponíveis, sugira uma forma de efectuar a negação bit a bit do conteúdo de um registo e implemente-a. Teste o seu programa com os seguintes valores e confirme manualmente os resultados obtidos:

```
0x0000FF1E, 0xF5A30614, 0x1ABCE543
```

#### 2. Instruções de deslocamento

- a) Para além das instruções que implementam operações lógicas bit a bit, o MIPS disponibiliza ainda operações de deslocamento<sup>2</sup> (*shift*), nomeadamente, deslocamento à esquerda lógico, deslocamento à direita lógico e deslocamento à direita aritmético. Em todas estas instruções o número de bits a deslocar é especificado na instrução (campo **Imm**):

```
sll Rdst,Rsrc,Imm    # Shift left logical
srl Rdst,Rsrc,Imm    # Shift right logical
sra Rdst,Rsrc,Imm    # Shift right arithmetic
```

<sup>1</sup> Em linguagem C, os operadores lógicos *bitwise* representam-se por: AND - **&**; OR - **|**; XOR - **^**; NOT - **~**

<sup>2</sup> Em linguagem C o deslocamento à direita representa-se por **>>** e o deslocamento à esquerda por **<<**

- b) Escreva um programa que efectue as 3 operações de deslocamento, considerando como operandos os registos **\$t0** e a constante **Imm** (valor e número de bits a deslocar, respectivamente) e colocando os resultados nos registos **\$t2**, **\$t3** e **\$t4**. Execute o programa para os seguintes pares de valores:

```
(0x12345678, 1)
(0x12345678, 4)
(0x12345678, 16)
(0x862A5C1B, 2)
(0x862A5C1B, 4)
```

- c) Preencha (no seu *logbook*) a tabela seguinte e confirme manualmente os resultados para cada um dos pares de valores de entrada.

\$t0	Imm	\$t2 (sll)	\$t3 (srl)	\$t4 (sra)
0x12345678	1			
0x12345678	4			
0x12345678	16			
0x862A5C1B	2			
0x862A5C1B	4			

- d) Usando máscaras e deslocamentos, escreva e teste um programa que imprima separadamente no ecrã, em hexadecimal, cada um dos 8 dígitos da quantidade armazenada no registo **\$t0**.

```
print_int16((val & 0xF0000000) >> 28); print_char('\n');
print_int16((val & 0x0F000000) >> 24); print_char('\n');
...
print_int16((val & 0x000000F0) >> 4); print_char('\n');
print_int16( val & 0x0000000F);
```

### 3. Directivas do *assembler*

Os programas que efectuam a tradução de código *assembly* para código máquina (designados em inglês por *assemblers*) disponibilizam um conjunto de instruções que permitem ao programador controlar alguns aspectos do processo de tradução. Estas instruções (não confundir com as instruções do CPU) são normalmente designadas por directivas e são executadas exclusivamente pelo *assembler* durante o processo de tradução do código.

No caso do *assembler* para o MIPS usado no MARS, as directivas são constituídas por um identificador, cujo primeiro carácter é sempre o símbolo “.”, e, em alguns casos, por um ou mais parâmetros. Exemplos de directivas: **.text**, para definir o início da zona de código do programa; **.data**, para definir o início da zona de dados do programa.

Para além das duas directivas anteriores há uma outra que será usada com frequência e que permite a declaração de *strings* (sequências de caracteres delimitadas pelo carácter “”). Por exemplo, a declaração da *string* **"AC1 – aulas praticas"**, pode ser efectuada do seguinte modo:

```
.data
str1: .asciiz "AC1 – aulas praticas"
```

em que **str1** é um identificador (*label*), que é uma sequência de caracteres alfanuméricos, cujo primeiro carácter não pode ser um numérico.

A directiva **.asciiz** reserva, em memória, espaço para alojar todos os caracteres da *string*, e ainda para um carácter especial que explicita o fim da *string*, designado por terminador. Em *assembly* e em linguagem C o terminador é o carácter '**\0**', isto é, o byte **0x00**. De referir ainda que cada carácter é codificado, de acordo com o código ASCII, com 1 byte.

- a) Edite e compile no MARS, o seguinte código:

```

.data
str1: .asciiz "So para chatear"
str2: .asciiz "AC1 – aulas praticas"

.text
.globl main
main: jr $ra

```

- b) Sabendo que o segmento de dados tem início no endereço **0x10010000** da memória (os endereços no MIPS são quantidades de 32 bits), preencha a tabela seguinte com o código ASCII e o endereço onde está armazenado cada um dos caracteres da *string* **str2**. Confirme os códigos dos caracteres numa tabela ASCII e verifique a sua localização na memória através da janela de dados do MARS. Preencha (no *logbook*) a tabela seguinte com todos os endereços de memória ocupados pela *string* **str2** e respectivos valores.

Endereço	Valor	Endereço	Valor

- c) O identificador da *string* (*label*) permite que o endereço inicial dessa *string* seja referenciado por uma instrução *assembly*. Por exemplo, a utilização da *system call* **print\_str()** requer que, antes da chamada à função, o registo **\$a0** do CPU seja inicializado com o endereço inicial da *string* a imprimir. No MIPS, a obtenção do endereço a que corresponde o identificador da *string* pode ser feita através da instrução virtual **la**, iniciais de *load address* (o MIPS não disponibiliza, por razões que serão compreendidas mais tarde, uma única instrução que permita a inicialização de uma quantidade de 32 bits num registo do CPU).

O programa para imprimir a *string* **str2**, usando a *system call* **print\_str()**, fica então:

```

.data
str1: .asciiz "So para chatear"
str2: .asciiz "AC1 – aulas praticas"

.text
.globl main
main: la      $a0, str2 # instrução virtual, decomposta pelo
                        # assembler em 2 instruções nativas
      ori $v0, $0, 4 # $v0 = 4
      syscall      # print_str(str2);
      jr $ra      # fim do programa

```

Edite, compile e execute este código.

- d) Traduza para *assembly*, e teste no MARS a seguinte sequência de código C:

```

print_str("Introduza 2 numeros ");
a = read_int();
b = read_int();
print_str("A soma dos dois numeros e:");
print_int(a + b);

```

**Anexo:**

<b>u</b>	<b>v</b>	<b>w = u or v</b>
0	x	x
1	x	1
<b>u</b>	<b>v</b>	<b>w = u and v</b>
0	x	0
1	x	x
<b>u</b>	<b>v</b>	<b>w = u xor v</b>
0	x	x
1	x	x\
<b>u</b>	<b>v</b>	<b>w = u nor v</b>
0	x	x\
1	x	0

<b>U</b>	<b>V</b>	<b>W = U or V</b>
0	X	X
F	X	F
<b>U</b>	<b>V</b>	<b>W = U and V</b>
0	X	0
F	X	X
<b>U</b>	<b>V</b>	<b>W = U xor V</b>
0	X	X
F	X	X\
<b>U</b>	<b>V</b>	<b>W = U nor V</b>
0	X	X\
F	X	0

**Notas:**

1. Na tabela de esquerda apresentam-se alguns casos particulares com operandos de 1 bit das operações lógicas mais comuns (o símbolo \ significa negação).
2. Na tabela de direita apresentam-se alguns casos particulares com operandos de 4 bits (1 dígito hexadecimal) das operações lógicas mais comuns (o símbolo \ significa negação bit a bit, ou seja, complemento para 1 do operando;  $X + X\ = F$ ).