universidade de aveiro deti departamento de electrónica, telecomunicações e informática

Trabalho Prático 6

Instruções Lógicas e de Deslocamento

Objetivos

- Instruções lógicas bit-a-bit (*bitwise*).
- Instruções de deslocamento (*shift*) lógico e aritmético.
- Instrução *Syscall* e chamadas ao sistema

Introdução

Algumas operações de uso frequente em programação, tais como mascarar, fazer o *set*, o *reset* ou o *toggle* de um subconjunto dos bits dentro dum registo, são implementadas em *Assembly* do MIPS através das instruções lógicas básicas bit-a-bit (*bitwise*). Outro conjunto de instruções bastante usado, especialmente na manipulação de grandezas binárias, são as instruções de deslocamento (*shift*) aritmético e lógico.

O simulador MARS disponibiliza um vasto conjunto de *system calls* que suportam as operações de entrada e saída de e para a consola (ecrã). Destas selecionamos algumas cujo conhecimento será essencial na resolução de futuros exercícios mais complexos.

Guião

1. Instruções lógicas

1.1 Codifique um programa em *Assembly* que a partir do valor de dois operandos presentes em $t0 e $t1[1] calcule o resultado do AND, OR, NOR e XOR e guarde os resultados em $t2, $t3, $t4 e $t5, respetivamente.

1.2 Teste o programa e confirme manualmente os resultados para os seguintes pares de valores:

$t0 = 0x12345678 , $t1 = 0x0000000F
$t0 = 0x12345678 , $t1 = 0x0000F000
$t0 = 0x12345678 , $t1 = 0x0000ABCD

1.3 Como poderia implementar a operação de negação bit a bit do registo $t0 e armazenar o resultado em $t6?

[1] Ao escrever o programa considere que $t0 e $t1 já estão inicializados com os valores previstos. A inicialização pode ser feita editando no próprio simulador o conteúdo do registo antes de correr o programa.

2. Instruções de deslocamento lógico e aritmético

O MIPS disponibiliza três instruções de deslocamento (*shift*): deslocamento à esquerda (lógico) e deslocamento à direita lógico e aritmético[2].

2.1 Escreva um programa que efectue as três operações de deslocamento considerando como operando o registo $t0 e a constante **Imm** (número de bits a deslocar). Os resultados devem ser guardados nos registos $t1, $t2 e $t3.

2.2 Teste o programa com os seguintes pares de valores e verifique manualmente os resultados:

a) **0x12345678, 1**
b) **0x12345678, 4**
c) **0x12345678, 2**
d) **0xF0000003, 4**

3. Chamadas ao sistema

3.1 Traduza para *Asssembly* e teste a seguinte sequência de código C:

```
print_string( "Introduza dois números :" );
a = read_int();
b = read_int();
print_string( "A soma dos números e': " );
print_int10( a + b )
```

3.2 Teste o programa com valores **a** e **b** positivos e negativos.

3.3 Substitua a instrução **print_int10**(a + b) por **print_intu10**(a + b). Repita os testes da alínea anterior. O que acontece quando o resultado deveria ser negativo?

4. Usando máscaras e deslocamentos escreva um programa que imprima separadamente no ecrã em hexadecimal cada um dos dígitos hexadecimais do valor armazenado no registo $t1. Use como exemplo as instruções que se seguem:

```
print_int16((t1 & 0xF0000000) >> 28 ); print_char(' ');
print_int16((t1 & 0x0F000000) >> 24 ); print_char(' ');
…
```

4.1 Que alterações seriam necessárias para imprimir o valor de $t1 em base 8?

[2] A diferença entre o deslocamento à direita lógico e aritmético é que este preserva o sinal do número, isto é, se o número era originalmente positivo continua positivo se era negativo mantem-se também negativo, enquanto o deslocamento lógico ignora o sinal do número.