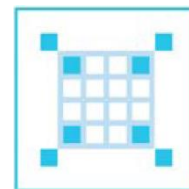


Arquitectura de Computadores

LICENCIATURA EM ENG³ INFORMÁTICA
FACULDADE DE CIÊNCIA E TECNOLOGIA
UNIVERSIDADE DE COIMBRA



Lab 5 – Funções e Aplicações de Operações *bitwise* em MIPS

Neste trabalho de laboratório pretende-se aprender o que são operações *bitwise*, como usá-las em MIPS e qual a sua utilidade.

1. Introdução

As operações *bitwise* são utilizadas quando precisamos realizar operações ao nível do bit em números inteiros, ou seja, trabalhar com a sua representação binária. Caso ambos os operandos sejam *strings*, essas operações irão trabalhar com o código ASCII do carácter correspondente.

2. Máscaras

Dentre as operações *bitwise* no MIPS, temos o importante conjunto das *operações lógicas*. Estas aplicam funções lógicas entre cada um dos bits correspondentes dos seus operandos (que, como sabem, são registos de 32 bits). O MIPS tem 4 instruções do tipo *R* que implementam operações lógicas:

- *and* – “e” lógico entre dois registos, bit a bit
- *or* – “ou” lógico entre dois registos, bit a bit
- *nor* – “ou” negado lógico entre dois registos, bit a bit
- *xor* – “ou” exclusivo entre dois registos, bit a bit

O MIPS tem também 3 instruções imediatas equivalentes:

- *andi* – “e” lógico entre um registo e um valor imediato, bit a bit
- *ori* – “ou” lógico entre um registo e um valor imediato, bit a bit
- *xori* – “ou” exclusivo lógico entre um registo e um valor imediato, bit a bit

Como aprendeu nas aulas teóricas, uma das aplicações destas operações é a implementação de *máscaras*, em que os bits de um registo são processados independentemente até que se obtém um resultado pretendido de transformação desse valor (por exemplo, a colocação de alguns bits a 0 ou a 1, e a manutenção dos restantes inalterados).

- i) Para aplicar esta ideia num caso concreto, que exemplifica o uso de máscaras para produzir resultados condicionais eficientes, considere que pretende desenvolver um programa que declara uma tabela de inteiros chamado **tab** com **n** números. Esse programa deverá chamar uma função que devolva a quantidade

de números ímpares dessa tabela. A função deverá aceitar como parâmetros o endereço da tabela e o seu comprimento. A função deverá chamar-se *oddnumber* e deverá apenas **recorrer a máscaras** para determinar a paridade de cada número. **Dica:** aproveite o facto de poder pensar em binário! Qual a propriedade fundamental de um número inteiro ímpar quando escrito em binário?

- ii) Baseada na função anterior, pretendemos agora implementar uma função que transforme todos os números ímpares da tabela no número par imediatamente anterior. Tente identificar qual a operação *bitwise* que poderá utilizar para efectuar o pretendido. Na função principal imprima a tabela resultante utilizando a *Syscall* adequada.
- iii) Implemente uma função que receba como parâmetro o endereço de uma *string* e que converta eventuais letras minúsculas existentes nessa *string* em letras maiúsculas. Para isso terá em primeiro lugar que verificar se cada carácter é uma letra minúscula (código entre 0x61 'a' e 0x7A 'z'). No caso de se tratar de uma letra minúscula, converta o carácter para maiúsculo, sabendo que os códigos desses caracteres irão agora variar entre 0x41 'A' e 0x5A 'Z'. Identifique a operação *bitwise* que terá de utilizar para fazer essa conversão. No programa principal imprima a *string* antes e depois de ser convertida (utilizando a *Syscall* apropriado).

3. Utilização da operação *bitwise* XOR para implementar um sistema simples de encriptação de texto

Uma das formas mais simples (e menos segura) de encriptar uma mensagem de texto é utilizara operação XOR de cada carácter com uma chave de encriptação fixa e conhecida quer pelo emissor, quer pelo receptor. Considere a título de exemplo o carácter 'A' que tem o código 0x41, ou 01000001 em binário. Imagine que se utilizava a seguinte chave de encriptação: 00111100 (0x3C). Utilizando a operação XOR o resultado da encriptação seria o seguinte:

```

          01000001
xor      00111100
          -----
          01111101 (0x7D que representa o carácter '}' )

```

O carácter 'A' seria transformado no carácter '}'. Para desencriptar, bastará aplicar de novo a operação XOR com a mesma chave de encriptação:

```

          01111101
xor      00111100
          -----
          01000001 (0x41 que representa o carácter 'A' )

```

Faça um programa que permita encriptar e desencriptar uma frase introduzida pelo utilizador. Para isso implemente em *assembly* uma função *codificaString* que receba como parâmetro uma *string* e uma chave de encriptação (um carácter) e que permita encriptar/desencriptar a *string*. Para testar a sua implementação, no programa principal, utilizando a função *read_string* do Syscall, peça ao utilizador para introduzir uma frase e um carácter. Depois chame a função que desenvolveu para encriptar e uma segunda vez para desencriptar. Após cada chamada da função imprima a *string* no ecrã para verificar o resultado obtido.

4. Instruções de deslocamento

Usando o que aprendeu sobre as instruções de deslocamento nas aulas teóricas, programe uma função *Polycalc* em *Assembly* que implemente a seguinte função *da forma mais eficiente* (sem recorrer à instrução de divisão):

$$f = \sum_{i=0}^5 \frac{x_i}{2^i}$$

Os parâmetros x_0 a x_5 são lidos de uma tabela armazenada em memória que é passada como parâmetro de entrada para a função em *Assembly*. No programa principal deverá incluir a declaração da tabela e imprimir no ecrã o valor devolvido pela função em *assembly*.

Exemplo: se a tabela de entrada for:

Tab: .word 10,24,32,40,64,128

Então o resultado da função deverá ser 43.