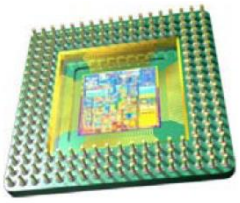
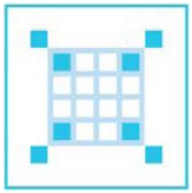


Arquitectura de Computadores

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



Lab 9 - Operações *bitwise* em MIPS – Aplicações

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 sua representação binária. Caso ambos os operandos sejam strings, essas operações irão trabalhar com os valores ASCII dos seus caracteres.

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 do tipo *I* do mesmo género:

- *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 tem um código em C que dado um array de inteiros chamado *tab* com *n* números escreve no ecrã a quantidade de números ímpares desse array. Essa quantidade é devolvida

pela função *oddnumber* que terá de desenvolver. A função *oddnumber* é escrita em *Assembly* e ***recorre apenas a máscaras*** para determinar a paridade de certo número. Os argumentos de entrada da função são o array *tab* e o seu número de elementos *n.* ***Dica***: aproveite o facto de poder pensar em binário! Qual a propriedade fundamental de um número inteiro ímpar quando escrito em binário?

ii) Pretendemos agora que os números pares e ímpares do array *tab* sejam imprimidos em linhas distintas. Nesse sentido, comece por modificar a função em *Assembly* desenvolvida na alínea anterior para que os números pares sejam armazenados no array *tabeven* e os números ímpares no array *tabodd*. Ambos os arrays são passados como argumentos de entrada da função *oddnumber*. Note que o programa em C já tem desenvolvido o código necessário para imprimir todos os números ímpares do array *tab*. Complete o código para que também sejam apresentados os números pares no ecrã numa linha distinta.

iii) Implemente uma função que receba uma *string* e que permita converter eventuais letras minúsculas em letras maiúsculas. Para isso terá em primeiro lugar que verificar se cada caracter é uma letra minúscula (código entre 0x61 'a' e 0x7A 'z'). No caso de se tratar de uma letra minúscula, converta o caracter para maiúsculo, sabendo que os códigos desses caracteres irão agora variar entre 0x41 'A' e 0x5A 'Z'. Identifique a operação bitwise que terá que utilizar para fazer essa conversão.

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 caracter com uma chave de encriptação fixa e conhecida quer pelo emissor, quer pelo receptor. Considere a título de exemplo o caracter '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 caracter '}' )
```

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

```
      01111101
xor   00111100
      01000001 (0x41 que representa o caracter 'A' )
```

Faça um programa em C que permita encriptar e desencriptar uma frase introduzida pelo utilizador. Para isso implemente uma função em *assembly* que receba uma *string* e uma chave de encriptação (um caracter) como parâmetros e que permita encriptar/desencriptar a *string*:

```
void codificastring(char *frase, char chave);
```

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*:

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

Os parâmetros x_0 a x_5 são lidos de um vetor armazenado em memória que é enviado como parâmetro de entrada para a função em *Assembly*. Como anteriormente, deverá correr o programa a partir da função `main()` de um programa em C; a função `main()`,deverá incluir a declaração do vetor e imprimir no ecrã o valor devolvido pela função em *assembly*.