Lab 10 – Funções no MIPS e Interligação com o C – Funções Recursivas

1. Funções Recursivas

Uma função recursiva é aquela que se chama a si mesma para resolver um subproblema menor do mesmo tipo. É uma técnica poderosa que pode levar a soluções elegantes e concisas para certos tipos de problemas. O exemplo clássico da utilização de uma função recursiva é o cálculo do fatorial de um número. No caso do fatorial, podemos definir o fatorial de um número a partir do cálculo do fatorial do número anterior:

$$n! = n \times (n-1)!$$

A função pode ser calculada através do recurso a uma versão mais simplificada do problema. O processo pode repetir-se até se chegar a um caso elementar cuja solução seja conhecida. No caso do cálculo do fatorial, se *n* for igual a 0, a função retorna 1 (este é o chamado caso elementar ou condição de paragem). O exemplo de uma implementação em C da função seria este:

```
#include <stdio.h>

int fatorial(int n) {
    // Caso elementar: o fatorial de 0 é 1
    if (n == 0) {
        return 1;
    } else {
    // Caso recursivo: chama a função simplificando o problema
        return n * fatorial(n - 1);
    }
}

int main() {
    // Vamos testar a função fatorial
    int numero = 5;
    printf("Fatorial de %d: %d\n", numero, fatorial(numero));
    return 0;
}
```

2. Implementação de Funções Recursivas em *Assembly* do MIPS

Escrever uma função recursiva em *Assembly* MIPS pode ser um pouco desafiador devido à arquitetura específica e à falta de suporte direto para chamadas recursivas. Como a função vai-se chamar a ela mesma será sempre necessário preservar o valor do registo `$ra`. Para além disso, para se respeitar a convenção de registos e preservar os valores dos parâmetros de entrada e dos cálculos intermédios realizados na função, pode ser necessário armazenar outros registos na pilha. A seguir é apresentada uma função que calcula o fatorial de um número, escrita em *Assembly* do MIPS:

```
.text
.globl fatorial

fatorial:
    # Função recursiva para calcular o fatorial
    # Argumento: $a0 (número n para calcular o fatorial)
    # Resultado: $v0 (fatorial de $a0)

    # Arranja espaço na pilha para o $ra e para preservar o
    # valor de $a0
    addi $sp,$sp,-8
    sw $ra, 4($sp)

    # Caso elementar: fatorial de 0 é 1?
    beq $a0, $zero, caso_elementar

    # Caso recursivo: chama a função fatorial(n-1)
    # primeiro preserva o valor de $a0
    sw $a0, 0($sp)

    addi $a0, $a0, -1 # n = n-1
    jal fatorial

    # recupera o valor de $a0
    lw $a0, 0($sp)

    # Multiplica o resultado da chamada recursiva por n
    mul $v0, $v0, $a0
    j fim

caso_elementar:
    # Caso elementar: fatorial de 0 é 1
    li $v0, 1

fim:
    # Recupera o registo $ra e repõe o valor de $sp
    lw $ra, 4($sp)
    addi $sp,$sp,8

    # Retorna da Função
    jr $ra
```

3. Exercícios

Resolva os seguintes exercícios:

i) Implemente uma função recursiva `soma(int n)` que calcule a soma dos números inteiros de 1 até n. Procure encontrar o caso elementar e o caso geral neste exemplo.

ii) Implemente uma função recursiva `potencia(int n, int e)` que permita calcular n^e. Lembre-se que $n^e = n \times n^{e-1}$ e que $n^0 = 1$.

iii) Implemente uma função recursiva para calcular o Máximo Divisor Comum (MDC) entre dois números: `mdc(int a, int b)`. Para isso pode utilizar o algoritmo de Euclides que é eficiente e se baseia na observação de que o MDC de dois números é o mesmo que o MDC do menor deles e do resto da divisão do maior pelo menor. O caso elementar acontece quando um dos números é zero. Nesse caso, o MDC é o outro número não zero. Se ambos os números não forem zero, então o $mdc(a, b) = mdc(b, a \ \% \ b)$, em que % denota o resto da divisão. Poderá testar a sua função com `mdc(48, 18)` que deverá dar o resultado igual a 6, que é o MDC desses dois números.

NOTA: Para simplificar o problema assuma que o primeiro parâmetro é sempre maior ou igual ao segundo.

iv) Implemente uma função recursiva `procura(int *tab, int low, int high, int num)` que utilize uma pesquisa binária para encontrar um número numa tabela ordenada de números inteiros. A pesquisa binária é um algoritmo eficiente para encontrar um elemento específico numa tabela ordenada. A ideia principal é reduzir para metade a porção da tabela onde o elemento pode estar a cada iteração.

Definem-se normalmente dois índices para identificar a porção da tabela onde vamos procurar: um índice inicial (geralmente chamado de *low*) e um índice final (geralmente chamado de *high*). O *low* é o índice do primeiro elemento da tabela a pesquisar, e o *high* é o índice do último elemento. Inicialmente estes índices são inicializados a `0` e `num-1` respectivamente.

Em cada interacção, calcula-se o índice médio (*mid*) da tabela calculando a média de *low* e *high* ($mid = \frac{low + high}{2}$). Isso divide a tabela em duas partes aproximadamente iguais. Compara-se o elemento que podemos encontrar nesse índice médio com o elemento que estamos a procurar. Existem três possíveis resultados para a comparação:

a. Se o elemento em *mid* for igual ao elemento procurado, então encontrámos o valor que procurávamos e podemos retornar o índice *mid* como resultado da função.
b. Se o elemento em *mid* for menor que o elemento procurado, então concluímos que o elemento que procuramos está na metade superior da tabela. Nesse caso, ajusta-se o índice *low* para *mid* + 1 e repetimos o processo.
c. Se o elemento em *mid* for maior que o elemento procurado, então concluímos que o elemento que procuramos está na metade inferior da tabela. Nesse caso, ajusta-se o índice *high* para *mid* - 1 e repetimos o processo.

Vamos repetir os passos anteriores até que o elemento seja encontrado ou então até que o índice *low* seja maior que *high*, o que significa que o elemento não está na tabela. Neste caso a função deverá devolver -1.

Utilize o ficheiro **main_4.c**, fornecido com o enunciado como ponto de partida para testar esta função.