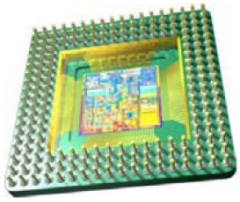

Lab 8
Funções, *Stack Frames* e Recursividade

***Objetivo*: Neste trabalho de laboratório pretende-se ver como funciona o mecanismo de chamada de funções e a utilização da pilha (*stack frames*), permitindo a correta interligação de funções em linguagem *assembly* e em C, bem como a instanciação independente de cada chamada a uma função que permite a ocorrência de recursividade.**

1. Pilha (*Stack*)

A pilha é uma zona de memória utilizada para armazenar dados temporários, incluindo variáveis locais e passagem de parâmetros para funções. O programador precisa de ter o cuidado de, em cada função, deixar a pilha inalterada. Além disso, quando uma função chama outra função necessita de guardar o seu próprio endereço de retorno (`$ra`) de forma a poder regressar ao código que a chamou. A passagem de parâmetros faz-se através dos registos `$a0` a `$a3` (`$4` a `$7`) e a devolução será feita pelos registos `$v0` e `$v1` (`$2` e `$3`).

Inspirado no conhecido conjunto de Mandelbrot, considere o seguinte programa em C que imprime o valor de *z* ao fim de *n* iterações, sendo o valor inicial de *z* e o valor de n indicados pelo utilizador.

$z(0) = z_0$
$z(n) = z^2(n-1) + z_0$

```
#include <stdio.h>

int Mandelbrot (int z0, n);

int main ()
{
     int z0, n, z;
     printf ("Introduza o valor inicial (z0): ");
     scanf ("%d", &z0);
     printf ("Indique o número de iterações (n): ");
     scanf ("%d", &n);
     z = Mandelbrot (z0, n);
     printf ("O resultado é %d \n",z);
     return 0;
}
```

a) Usando a convenção para chamada de funções, escreva em linguagem *assembly* a função `Mandelbrot()` que irá calcular o resultado de *z* para a *n-ésima* iteração solicitada pelo utilizador.
Nota: *poderá admitir que o resultado da função não ultrapassa os 32 bits.*

b) Uma qualquer **função** bem programada pode ser usada numa chamada **recursiva** a si própria, desde que tenha uma **condição de paragem bem definida**. Estas funções são muito úteis para alguns tipos de cálculo numérico ou outros. Baseando-se no código em C e na função `Mandelbrot()` do ponto 1.a), escreva uma nova função que permita o cálculo da *n-ésima* iteração da expressão de Mandelbrot, desta vez de forma recursiva, tendo em atenção as expressões anteriormente indicadas.

2. Passagem de mais do que quatro parâmetros

A passagem de parâmetros para uma função pode ter mais do que os 4 parâmetros passados através dos registos `$a0` a `$a3`. Quando tal acontece, os restantes parâmetros têm de ser passados pela pilha na ordem inversa à do cabeçalho. Assim, ao chamar uma função que tem mais do que 4 parâmetros de entrada, é necessário alocar espaço na pilha para os parâmetros adicionais, chamar a função e, quando esta retornar, remover da pilha os parâmetros adicionais.

a) Pretende-se então implementar um programa que permita determinar o valor dado pela expressão:

$$f = 2(x_1 + 3x_2)(x_3 - 2x_4x_5)$$

Escreva uma função `program_asm()` em linguagem *assembly* que é invocada a partir de uma rotina `main()` em C com a seguinte declaração:

```
int program_asm(int *tab);
```

A função `program_asm()` recebe um vector com os elementos `x1, x2, ..., x5` que são introduzidos pelo utilizador e devolve o valor de f. O cálculo será feito por uma outra função em *assembly* `PolyCalc()` invocada por `program_asm()` que obedece ao seguinte protótipo:

```
int PolyCalc(int x1, int x2, int x3, int x4, int x5);
```

b) Usando agora o que aprendeu sobre a utilização de instruções de deslocamento para a implementação de multiplicações por múltiplos de 2, programe uma nova função em assembly `PolyCalc2()` que implemente a seguinte função da forma mais eficiente possível:

$$f = 32x_1 + 8x_2 + 4x_3 + 2x_4 + x_5$$

3. Chamadas de funções C em *assembly*

O algoritmo *bubble sort* é um algoritmo simples de ordenamento de elementos que percorre repetidamente a lista a ser ordenada, comparando cada conjunto de pares adjacentes, permutando-os caso se encontrem fora de ordem. A lista é percorrida até que não se verifique nenhuma permuta, indicador de que a lista está finalmente ordenada.

Desenvolva uma função `BubbleSort()` em C que permita ordenar os elementos contidos no vetor do ponto anterior, devolvendo no final o número de permutas que foram efetuadas. A declaração da função deverá ser

```
int BubbleSort (int *v);
```

Esta função deverá ser chamada a partir do código *assembly* escrito no ponto anterior (a função `program_asm`).