Lab 4 – Utilização da pilha para passagem de argumentos e armazenamento de variáveis, limites de alocação e recursividade

1. Utilização da pilha

1.1 Escreva uma função que devolva a soma dos quadrados dos argumentos de entrada. Essa função deverá aceitar um número de argumentos variável e para isso a linguagem C define que o protótipo da mesma deverá ser da seguinte forma:

```
int addsquares(int n,...);
```

As reticências indicam que a seguir ao argumento n poderá haver um qualquer número de argumentos, sendo o próprio argumento **n** o número desses argumentos. Assim a função poderá ser chamada das seguintes formas:

```
int sum1=addsquares(2,5,3);       // 2 argumentos adicionais
int sum2=addsquares(4,2,3,5,2);   // 4 argumentos adicionais
```

Para resolver este problema não use as funções `va_start`, `va_arg`, `etc`.; use antes o acesso direto aos argumentos na pilha conhecendo o endereço do argumento `n`.

2. Recursividade e limites da pilha

2.1 Implemente um programa `fibonacci.c` que calcule os valores da sequência de Fibonacci, através de uma função `fibonacci_cycle()` usando um ciclo para obter a solução. O valor de `n` deve ser introduzido na linha de comandos. Para converter a `string` com o número indicado pelo utilizador num inteiro pode recorrer à função `atoi()`. Caso o utilizador não indique o número na linha de comando deve indicar como deve ser utilizado o programa `fibonacci`.

Exemplos de utilização:

```
aluno@mips:~$ ./fibonacci
 usage: fibonacci numero
```

```
aluno@mips:~$ ./fibonacci 7
fibo(0) = 0
fibo(1) = 1
fibo(2) = 1
fibo(3) = 2
fibo(4) = 3
fibo(5) = 5
fibo(6) = 8
fibo(7) = 13

aluno@mips:~$
```

2.2 Acrescente na função anterior uma implementação recursiva da sequência de Fibonacci em que a função `fibonacci_recursive()` se chama a si própria. Durante o cálculo o programa deve escrever no ecrã o número crescente de vezes que a função recursiva é invocada.

3. Limite de alocação ao espaço de alocação inicial

Escreva um programa em C para calcular o número de vezes que consegue alocar 1MB (1 megabyte = 2^{20} byte = 1.048.576 byte) na memória dinâmica.

O programa deve:

- imprimir para o ecrã o menor endereço da stack (lembre-se que a stack cresce de cima para baixo e, portanto, o topo da stack é o menor dos seus endereços). Para este trabalho, considere que o menor endereço é o endereço da primeira variável local (no trabalho laboratorial #9 veremos que o menor endereço é outro, embora próximo).
- imprimir para o ecrã o endereço inicial da heap (para tal faça a alocação dinâmica de 1 byte e imprima o endereço devolvido pela função `malloc()`).
- em ciclo, faça a alocação sucessiva de 1MB enquanto o endereço do topo da heap for inferior ao endereço do topo da stack.

No final, imprima para o ecrã o número de bytes que ainda podiam ser alocados na memória dinâmica.

Nota: quando a memória do espaço de alocação esgota, em sistemas operativos com mecanismos de Memória Virtual o sistema operativo pode estender a memória alocada a um programa, aumentando o número de alocações de 1MB um número muito elevado de vezes. Neste exercício não estamos a considerar esta hipótese.

4. Introdução ao Processador MIPS

Descarregue o simulador MARS de http://courses.missouristate.edu/KenVollmar/MARS/ que é uma versão em JAVA do simulador clássico do MIPS chamado SPIM.

4.1 Comece por abrir o programa `fibonacci.asm` que calcula a conhecida série de Fibonacci. Observe o código fonte na janela de edição e, apesar de ainda não conhecer todas as instruções, tente compreender o programa.

4.2 Passe para a janela de execução, carregue o código, e comece a fazer execução linha a linha. Observe a janela "Text Segment" e tente compreender o significado de cada coluna. Acompanhe a execução olhando para os registos e memória.