# Assembly Challenge

#### TPC9 e Guião laboratorial

Luís Paulo Santos, André Martins Pereira e Alberto José Proença

## Introdução

Este documento é o guião para apoio ao funcionamento da última sessão laboratorial de SC.

**Não esquecer** que estes trabalhos experimentais <u>deverão ser realizados no servidor Unix de SC</u>, à semelhança dos trabalhos anteriores.

Na diretoria /home/shared/TPC9 do servidor remoto Unix de SC encontram-se disponíveis os diversos ficheiros auxiliares de apoio à realização deste trabalho laboratorial: copie-os para a sua diretoria de trabalho no mesmo servidor remoto.

#### O Desafio

O ficheiro soma\_grandes.s continha o código de uma função em assembly com a seguinte assinatura em C:

```
int soma grandes (int n, int *a);
```

Esta função, obtida após compilação com -02, percorria os n primeiros elementos do vetor de inteiros a e calculava a soma de todos aqueles que eram maiores do que 1000. Terminava devolvendo o resultado dessa adição.

Visualizando o ficheiro soma\_grandes.s constata-se que este foi corrompido; mantiveram-se inalteradas apenas as diretivas pertinentes para o assembler e algumas instruções do início e do término da função soma grandes (). Pretende-se recuperar esse ficheiro.

- 1. Escreva um possível algoritmo dessa função usando linguagem de alto nível (C, por ex.).
- 2. Reconstrua o ficheiro soma\_grandes.s, escrevendo o código em falta (em assembly), devidamente anotado, de modo a que ele implemente a funcionalidade descrita acima.
- 3. Crie um executável usando o seguinte comando de compilação e o ficheiro reconstruído:

```
gcc -Wall -02 -o somaG main.o soma grandes.s
```

O ficheiro com código objeto (main.o) foi desenvolvido com o objetivo de testar e validar uma versão reconstruída de soma\_grandes.s: cria 3 arrays aleatórios de inteiros, de dimensões diferentes, e compara os resultados obtidos usando a versão original do executável da função soma grandes() com os resultados da nova função reconstruída no exercício 1.

Ponha em execução o ficheiro somaG. Mostre o que aparece no monitor como resultado.

Se o seu programa abortar a meio ou devolver um resultado incorreto, <u>use o gdb</u> para detetar e localizar o(s) erro(s); depois <u>edite</u> o ficheiro <u>soma\_grandes.s</u>, corrigindo o(s) erro(s), e <u>teste</u> <u>de novo</u> a sua solução. **Repita** estes passos até que apareça no monitor o valor esperado.

TPC9: Assembly do IA-32

- 4. Crie um novo ficheiro assembly, compilando sem (-02) o algoritmo feito em 1.
  - **4.a) Compare** o ficheiro *assembly* resultante, com o reconstruído em **2.**, **indicando e caracterizando** as principais diferenças na alocação e manuseamento dos argumentos e das variáveis locais, bem como as respetivas consequências na estrutura do quadro da função.
  - **4.b) Quantifique** o impacto no desempenho devido aos acessos à memória para ler/escrever os dados a processar, no código compilado sem otimização comparativamente ao código compilado com otimização.
- **5.** A diretoria /home/shared/TPC9\_b contém um executável, hackme, resultante da compilação de dois ficheiros através do comando:

```
gcc -Wall -02 main.c autentica.c -o hackme.
```

Esta diretoria contém ainda o ficheiro objeto de main.c e o ficheiro com o código assembly autentica.s. Copie o conteúdo desta diretoria (ficheiros hackme, main.o e autentica.s) para a sua diretoria de trabalho.

O executável hackme recebe uma *password* introduzida pelo utilizador e compara-a com uma *password* guardada internamente.

**Produza** uma versão modificada do executável hackme que "imprima" a *password* no monitor, através da modificação do ficheiro autentica.s.

### Sugestão de resolução:

- i. Testar o funcionamento do executável hackme; note que a password tem de ser um número inteiro.
- ii. Analisar a estrutura da função de autenticação, no ficheiro autentica.s.
- iii. Modificar a função de autenticação para que devolva a *password* interna de modo a que a função chamadora a possa enviar para o monitor.
- iv. Criar um novo executável hacked.
- v. Verificar se a password aparece no monitor através da execução de hacked.

N° Nome:	Turma:
----------	--------

## Resolução dos exercícios (deve ser redigido manualmente)

Sugestões para esta resolução em baixo.

## 1. Um possível algoritmo

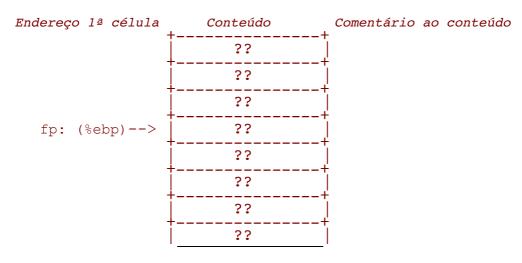
Escreva um possível algoritmo da função numa linguagem de alto nível (C, por ex.)

## 2. Reconstrução do ficheiro soma grandes.s devidamente anotado

**2.a)** Preencha a tabela de alocação de registos a argumentos, variáveis locais e eventuais variáveis de caráter mais temporário, que julgue serem necessários armazenar (com comentários)

Utilização dos Registos		
Variável	Registo	Comentários

**2.b)** Represente o quadro de ativação da função, colocando lá também o *stack pointer*. **Nota**: não é ainda possível saber o valor do conteúdo das células na *stack frame*.



TPC9: Assembly do IA-32

**2.c)** Com base na informação já obtida, construa agora o código assembly que substitui a parte "danificada". Considere que soma\_grandes.s foi obtido com gcc -02 -S.

Não esquecer de anotar devidamente este código (com comentários curtos e objetivos).

#### Nota 1:

Se o código tiver um ciclo, o controlo de permanência no ciclo <u>na resolução desta alínea deve</u> <u>ser feito no início de cada iteração do ciclo</u> e não no fim.

#### Nota 2:

Se compilar um código C desta função e usar essa versão de *assembly*, será considerado fraude e o trabalho receberá classificação negativa.

**2.d)** Compile para *assembly* com -O2 o programa C que criou no início para validar o código que criou. Escreva aqui o resultado e comente as diferenças com o seu código.

TPC9: Assembly do IA-32

### 3. Criação e validação dum executável

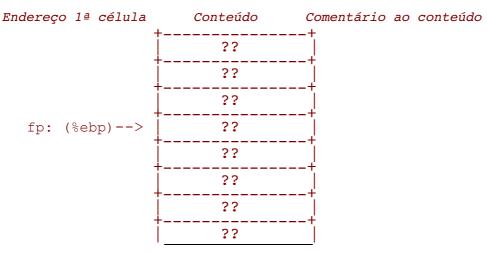
Crie o executável somaG e ponha-o a correr. Mostre o que aparece no monitor como resultado. Se não produzir o resultado esperado, use o gdb para corrigir o código assembly da função. Apresente numa folha separada a metodologia seguida para fazer o debugging, o código final e o resultado que apareceu no monitor quando o código estava correto.

## 4. Diferenças na compilação com e sem otimização

#### 4.a) Caracterização das diferenças

Apresente aqui as 2 versões de código assembly (com e sem otimização) e marque nas duas versões as instruções que são diferentes, explicando a diferença.

Complemente esta caracterização das diferenças mostrando como ficaria agora a *stack frame* da função.



#### 4.b) Quantificação das diferenças em performance

Apresente todas as instruções (para cada versão) que acedem à memória e indique à frente de cada instrução, quantos acessos à memória são feitos nessa instrução (não considere o acesso para ir buscar a instrução).

Em cada versão, desenhe uma caixa à volta do código do ciclo que percorre todos os elementos do *array*, pois esse código representa uma iteração do ciclo, e é normalmente aí que o código pode ter um impacto significativo na *performance*, em especial quando o *array* é muito grande e não cabe na *cache*.

Quando encontrar uma estrutura de controlo, tipo if...then, considere o percurso com mais acessos à memória.

Comente o resultado.