Universidade do Minho

Módulo 1 - Consolidação

Ficha 2

Estruturas de dados e controlo de fluxo

1. Introdução

Pretende-se com esta sessão teórico-prática que os alunos consolidem o conhecimento previamente adquirido sobre o suporte dado pelo conjunto de instruções à implementação de estruturas de dados e controlo de fluxo comuns em linguagens imperativas de alto nível. Recorre-se à arquitectura IA32 e ao compilador `gcc` como ferramentas de trabalho.

Recomenda-se a leitura do Capítulo 3 do livro "Computer Systems: A Programmer's Perspective" (http://csapp.cs.cmu.edu/). Este capítulo também está disponível na página da disciplina.

2. Endereçamento de vectores

O endereço de um elemento de um vector obedece à regra

```
endereço_de_vec[i] = endereço_base_de_v + i * factor_de_escala
```

onde o endereço base é o endereço da primeira posição de memória ocupada pelo vector `v`, `i` é o índice do elemento em questão e `factor_de_escala` é o tamanho, em *bytes*, do tipo de dados de cada elemento de `v`.

Esta expressão assenta no facto de todos os elementos do vector serem armazenados numa zona contínua de memória, de forma contígua e ordenada, começando pelo elemento índice `0` até ao elemento `n-1`, sendo `n` o número de elementos.

No caso de matrizes, vectores bi-dimensionais, é necessário conhecer uma das dimensões para poder calcular o endereço de `m[i][j]`. No caso de uma linguagem que armazena os elementos de cada linha de forma consecutiva a dimensão a conhecer é o número de colunas, `c`. A linguagem C armazena os vectores multidimensionais desta forma, designada por *row major order*.

Seja `m` uma matriz declarada como

```
tipo _de_dados m[l][c];
```

onde `l` é o número de linhas, `c` o número de colunas. Seja `factor_de_escala` o tamanho, em *bytes,* de cada elemento desta matriz. Então o endereço base de uma linha `i` é dado por:

$$m[i] = i * c * \ factordeescala, \forall_{i:0 \le i < l}$$

O endereço de um elemento `m[i][j]` é dado por

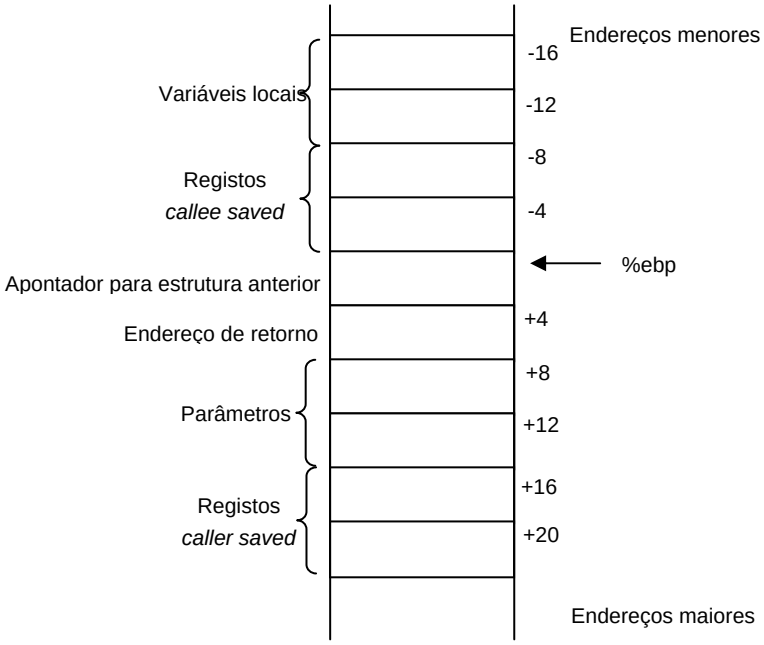
$$\& m[i][j] = m[i] + j * \ factordeescala = (i * c + j) * \ factordeescala, \forall_{i,j:0 \le i < l, 0 \le j < c}$$

3. Estrutura de activação

Cada função em C tem um contexto próprio constituído por vários elementos tais como, por exemplo, o valor dos seus parâmetros, as suas variáveis locais e o seu endereço de retorno. Para manter este contexto é usada uma estrutura de activação. esta estrutura de dados é criada na pilha (*stack*) e inclui os seguintes campos pela seguinte ordem: valor dos registos *caller saved*, parâmetros, endereço de retorno, valor anterior do apontador para a estrutura de activação

(registo %ebp), valor dos registos *callee saved* e variáveis locais. Esta estrutura de activação é apontada pelo registo %ebp; deslocamentos relativos a %ebp permitem aceder a qualquer campo da estrutura. A estrutura é criada parcialmente pela função que invoca (valor dos registos *caller saved*, parâmetros e endereço de retorno) e pela função invocada (restantes campos). Quando uma função termina, i.e., retorna, a respectiva estrutura de activação é destruída.

A figura seguinte apresenta o formato desta estrutura. Neste exemplo supõe que cada campo tem 4 *bytes* de tamanho; isto é verdade, no IA32, para os apontadores, inteiros e vírgula flutuante de precisão simples. De realçar que os deslocamentos relativamente ao %ebp para aceder a campos criados pela função que invoca são sempre positivos, enquanto que para aceder aos campos criados pela própria função são negativos. Isto é consequência directa do facto de a pilha crescer para endereços menores.



4. Exercício 1

Considere o código apresentado na tabela seguinte.

sum.c

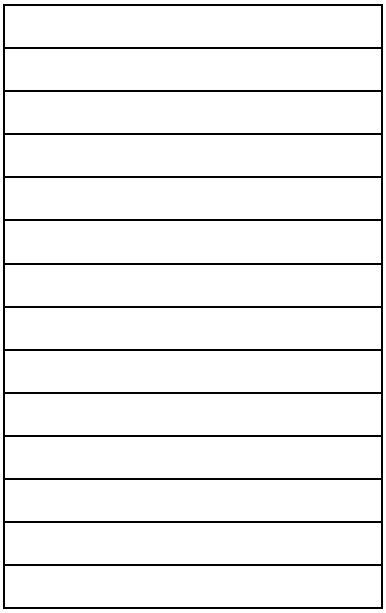
```
int sum (int *a, int n) {
   int i, s;

   s=0 ;
   for (i=0 ; i<n ; i++)
     s += a[i];
   return s;
}
```

Digite este código e guarde-o num ficheiro designado sum.c. Compile esta função sem qualquer optimização, gerando o ficheiro com o código *assembly* usando os seguintes comandos:

```
gcc –Wall –S sum.c
mv sum.s sum_O0.s
```

Questão 4.1 – Usando a tabela a seguir apresente um esquema da estrutura de activação. Tenha em consideração que:

- os parâmetros são colocados na pilha por ordem inversa aquela com que aparecem na definição da função;
- na versão não optimizada a estrutura de activação corresponde exactamente aquilo que as regras atrás enunciadas ditam;
- o compilador pode guardar alguns registos na pilha – registos *callee saved*. A única forma de saber quais é analisando o código *assembly*.



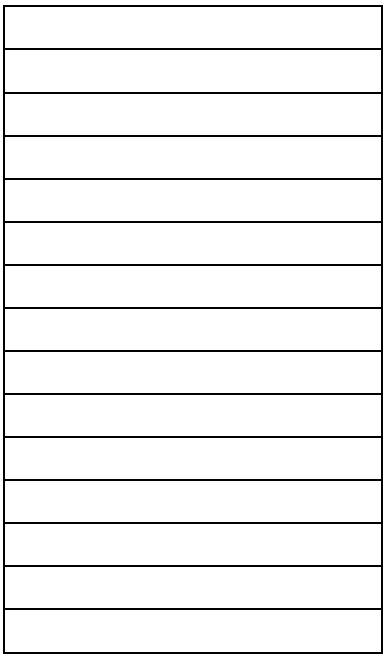
Questão 4.2 – Analise cuidadosamente o código *assembly* gerado pelo compilador, identificando a função de cada instrução.

Compile novamente esta função com nível de optimização 1, gerando o ficheiro com o código *assembly* usando os seguintes comandos:

```
gcc –Wall –O1 –S sum.c
mv sum.s sum_O1.s
```

Questão 3 – Analise cuidadosamente o código *assembly* gerado pelo compilador, identificando a função de cada instrução.

Esta tarefa é facilitada se for construindo um esquema da estrutura de activação enquanto analisa o código. Os campos criados pela função que invoca estão nas mesmas posições (relativamente ao %ebp) que na versão anterior, i.e., tudo até ao endereço de retorno é igual. Os campos criados pela função sum() poderão ser diferentes. Use a próxima tabela para representar esta estrutura.



Questão 4.4 – Identifique as optimizações feitas pelo compilador.

5. Exercício 2

Considere o código apresentado na tabela seguinte.

mm.c

```
void mm (int A[][MAX_SIDE], int B[][MAX_SIDE], int C[][MAX_SIDE], int n)
{
   int i, j, k, r;

   for (k=0 ; k<n ; k++) { // A column, B row
      for (j=0 ; j<n ; j++) { // B & C column
         r = B[k][j];
         for (i=0 ; i<n ; i++) { // A & C row
            C[i][j] += A[i][k] * r;
         }
      }
   }
}
```

Esta função calcula o produto C = A * B de duas matrizes quadradas de dimensão n, sendo C[i][j] dado por

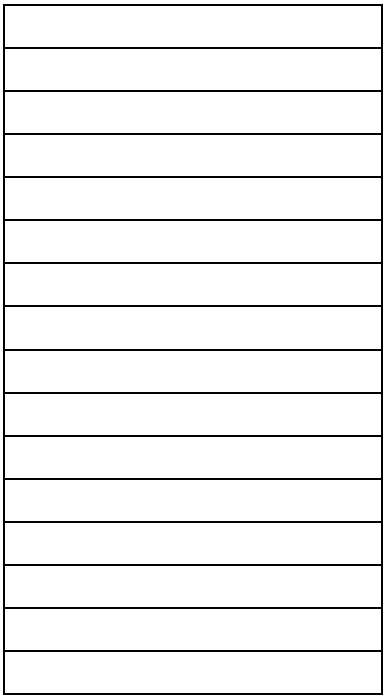
$$C[i][j] = \sum_{k=0}^{n-1} A[i][k] * B[k][j]$$

Descarregue o ficheiro mm.c a partir da página da disciplina. Compile esta função sem qualquer optimização, gerando o ficheiro com o código *assembly* usando os seguintes comandos:

```
gcc –Wall –S mm.c
mv mm.s mm_O0.s
```

Questão 5.1 – Usando a tabela a seguir apresente um esquema da estrutura de activação. Tenha em consideração que:

- os parâmetros são colocados na pilha por ordem inversa aquela com que aparecem na definição da função;
- na versão não optimizada a estrutura de activação corresponde exactamente aquilo que as regras atrás enunciadas ditam;
- o compilador pode guardar alguns registos na pilha – registos *callee saved*. A única forma de saber quais é analisando o código *assembly*.



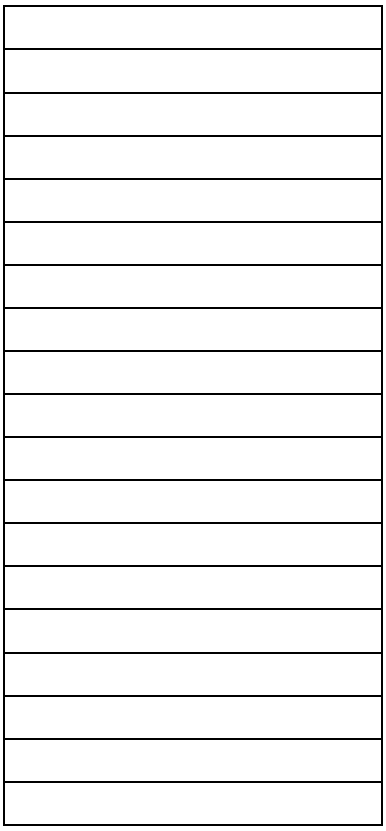
Questão 5.2 – Analise cuidadosamente o código *assembly* gerado pelo compilador, identificando a função de cada instrução.

Compile novamente esta função com nível de optimização 1, gerando o ficheiro com o código *assembly* usando os seguintes comandos:

```
gcc –Wall –O1 –S mm.c
mv mm.s mm_O1.s
```

Questão 5.3 – Analise cuidadosamente o código *assembly* gerado pelo compilador, identificando a função de cada instrução.

Esta tarefa é facilitada se for construindo um esquema da estrutura de activação enquanto analisa o código. Os campos criados pela função que invoca estão nas mesmas posições (relativamente ao %ebp) que na versão anterior, i.e., tudo até ao endereço de retorno é igual. Os campos criados pela função mm() poderão ser diferentes. Use a próxima tabela para representar esta estrutura.



Questão 5.4 – Identifique as optimizações feitas pelo compilador.