Guião VII - Resolução

Exercício 1 *(Funções)*

Considere o trecho de código de montagem abaixo, o qual resultou da compilação da função **proc**.

```
int proc(void)
{
     int x,y;
     scanf("%x %x", &y, &x);
     return x-y;
}
```

```
1   proc:                       ; %esp=0x800040 %ebp=0x800060
2   pushl  %ebp                 ; %esp=%esp-4=0x80003C,
                                ; M[%esp] = M[0x80003C] = 0x00800060
3   movl   %esp,%ebp            ; %ebp = %esp = 0x80003C
4   subl   $24,%esp             ; %esp = %esp-24 = 0x80003C-0x18  = 0x800024
5   addl   $-4,%esp             ; %esp = %esp+(-4) = 0x800024-0x4 = 0x800020
6   leal   -4(%ebp),%eax        ; %eax = %ebp-4 = 0x80003C-4 = 0x800038 = &x
7   pushl  %eax                 ; %esp = %esp-4 = 0x800020-4 = 0x80001C
                                ; 3ºarg = &x=0x800038, guardado em M[0x80001C]
8   leal   -8(%ebp),%eax        ; %eax = %ebp-8= 0x80003C-8 = 0x800034 = &y
9   pushl  %eax                 ; %esp = %esp-4 = 0x80001C-4 = 0x800018
                                ; 2ºarg = &y=0x800034, guardado em M[0x800018]
10  pushl  $.LC0                ; %esp = %esp-4 = 0x800018-4 = 0x800014
                                ; 1ºarg = apontador (0x300070) para a
                                ; sequência "%x %x", guardado em M[0x800014]
11  call   scanf                ; (1º) pushl %eip⇔%esp=%esp-4=0x800014-4=0x800010
                                ;   guarda endereço da instrução 12 em M[0x800010]
                                ; (2º) %eip=endereço de scanf
                                ; (3º) devolve 0x46 e 0x53
                                ; repõe %esp no valor antes do call, %esp=0x800014
12  movl   -8(%ebp),%eax        ; %eax=y ->
13  movl   -4(%ebp),%edx        ; %edx=x
14  subl   %eax,%edx            ; %edx=x-y
15  movl   %edx,%eax            ; %eax = valor de retorno da função = x-y
16  movl   %ebp,%esp            ; %esp = %ebp = 0x80003C
17  popl   %ebp                 ; (1º) %ebp = M[%esp]=M[0x80003C]= 0x800060
                                ; (2º) %esp=%esp+4=0x80003C+4=0x800040
18  ret                         ; pop %eip ⇔
                                ; (1º) %eip = M[%esp]=M[0x800040]= endereço de
                                ;      retorno à função que chamou proc()
                                ; (2º) %esp=%esp+4=0x800040+4=0x800044
```

Tendo em atenção que:

- imediatamente, antes da execução (linha 1) %esp=0x800040 e %ebp=0x800060;
- a chamada de scanf (linha 11), retorna, da entrada de dados, os valores 0x46 e 0x53;
- a sequência de caracteres "%x %x" passada como argumento a `scanf` foi armazenada a partir da posição de memória 0x300070.

a) Que valor é colocado no registo **%ebp**, na linha 3?

%ebp = %esp = 0x800040-4 = **0x80003C**

b) Em que endereços estão localizadas as variáveis locais **x** e **y**?

&x ⇔ %eax da linha 6 → **x** está guardada em **0x800038**
&y ⇔ %eax da linha 8 → **y** está guardada em **0x800034**

c) Qual é o valor de **%esp** antes da chamada a `scanf` (antes de executar a linha 11)?

Após a execução da instrução 10 o valor de **%esp = 0x800014**

d) Desenhe a área de ativação da pilha (*stack frame)* de proc, imediatamente, após o regresso de scanf (linha 12) incluindo toda a informação útil relevante, nomeadamente, as posições e os conteúdos de memória associadas às:

- variáveis,

- estruturas de demarcação e de retorno da própria função,
- regiões desperdiçadas (alinhamento) para melhorar o desempenho da *cache*.

O **desenho da pilha** revela a estrutura e conteúdo que segue.
As instruções 16 e 17 são equivalentes a um LEAVE.

```
            |          |
            +----------+
0x800060    |          | <-- %ebp (início e após instrução 17)
            +----------+
            |          |

            |          |
            +----------+
0x800044    |          | <-- %esp (após instrução 18)
            +----------+
0x800040    | end.ret. | <-- %esp (início e após inst. 17) (end.ret. função que chamou proc)
            +----------+
0x80003C    | 0x800060 | <-- %ebp (após instrução 3) e %esp (após as instrução 2 e 16)
            +----------+
0x800038    |   0x53   |    (x) (o conteúdo é colocado pelo scanf e indicado no enunciado)
            +----------+
0x800034    |   0x46   |    (y) (o conteúdo é colocado pelo scanf e indicado no enunciado)
            +----------+
0x800030    |          |
            +----------+
0x80002C    |          |
            +----------+
0x800028    |          |
            +----------+
0x800024    |          | <-- %esp (após instrução 4)
            +----------+
0x800020    |          | <-- %esp (após instrução 5)
            +----------+
0x80001C    | 0x800038 | <-- %esp (após instrução 7) (o conteúdo é o 3º arg de scanf)
            +----------+
0x800018    | 0x800034 | <-- %esp (após instrução 9) (o conteúdo é o 2º arg de scanf)
            +----------+
0x800014    | 0x300070 | <-- %esp (após instrução 10 e após a instrução 12 ⇔ CALL) (1º arg)
            +----------+
0x800010    | end.ret  |     (Endereço da instrução 12)
            +----------+
```

Exercício 2 *(Vetores)*
Complete a tabela, abaixo, considerando as declarações de tipos de dados que se seguem:

- short S[7];
- short *T[3];
- short **U[6];
- long double V[8];
- long double *W[4].

Vector	Espaço dum elemento	Espaço total	Endereço inicial	Expressão para acesso ao elemento ***i***
S	2	14	xS	xS + 2*i
T	4	12	xT	xT + 4*i
U	4	24	xU	xU + 4*i
V	12 **(◊)**	96	xV	xV + 12*i
W	4	16	xW	xW + 4*i

(◊) **sizeof(long double)** é **12** na máquina sc.di.uminho.pt

Exercício 3 *(Estruturas)*
Considerando que o registo **%edx** foi iniciado com o valor da variável **r** definida de acordo com a declaração que se segue, explique o funcionamento dos fragmentos de código abaixo.

```
struct rec {
   int i;
   int j;
   int a[3];
   int *p;} *r;
```

a)

```
1    movl    (%edx),%eax           ; lê r->i para %eax
```

```
2    movl    %eax,4(%edx)          ; guarda em r->j o valor de r->i
3    leal    8(%edx,%eax,4),%ecx   ; %ecx = &r->a[r->i] (*)
```

(*) %edx+8 = r->a
Somando a %edx+8 o valor %eax*4, igual a r->i*4), equivale a indicar o elemento r->i do campo r->a, ou seja, **r->a[r->i]**.

b)

```
1    movl    4(%edx),%eax          ; lê r->j para %eax
2    addl    (%edx),%eax           ; soma r->j com r->i e guarda em %eax
3    leal    8(%edx,%eax,4),%eax   ; calcula &r->a[r->i + r->j]
4    movl    %eax,20(%edx)         ; guarda &r->a[r->i + r->j] em r->p
```

```
r->p = &r->a[r->i + r->j];
```

Exercício 4 *(ciclo **for**)*

Pretende-se completar o código C da função **loop**, de que se conhece apenas a estrutura geral, de modo a obter por compilação, usando o *gcc*, o trecho de código de montagem seguinte.

Para solucionar o problema sugere-se que comente o código de montagem de forma a estabelecer uma relação direta entre os registos IA32 e as variáveis na função, tendo em **atenção**:
• a existência de uma estrutura de controlo (ciclo ***for***);
• a atribuição de um valor inicial à variável **i (*);**
• que por convenção o valor de retorno de uma função é devolvido no registo %eax;

```
1 int loop(int x, int y, int n)
2 {
3    int result = 0;
4    int i;
5    for (i = _____; i _____; i = _____) {
6       result += _______;
7    }
8    return result;
9 }
```

```
1   movl  8(%ebp),%ebx    ; ebx=x
2   movl  16(%ebp),%edx   ; (edx⇔) i = n (i por (*) e por exclusão de partes)
3   xorl  %eax,%eax       ; (eax⇔) result=0
4   decl  %edx            ; i=n-1
5   js    .L4             ; salta para fim do ciclo se i<0
6   movl  %ebx,%ecx       ; ecx=x
7   .p2align 4,,7         ; alinha o código na memória para optimizar a cache
8   L6:                   ; início do ciclo
9   imull 12(%ebp),%ecx   ; ecx=x*y
10  addl  %ecx,%eax       ; result += x*y
11  subl  %ebx,%edx       ; i = i-x
12  jns   .L6             ; regressa ao início do ciclo se i >=0
13 .L4:                   ; fim do ciclo
```

Pode-se ver que:

- **i** em %edx, **i =n** (linha 2)
- **result=0** em %eax (linha 3)
- **i =n-1** (linha 4)
- testar se **i >=0** (linhas 5 e 12)
- **x*y** em %ecx (linha 9)
- atualização **result+=x*y** (linha 10)
- **i** é decrementado de **x** (linha 11)

```
1 int loop(int x, int y, int n)
2 {
3    int result = 0;
4    int i;
5    for (i = n-1; i >= 0; i = i-x) {
6       result += y * x;
7    }
8    return result;
9 }
```