

# Programação em IA32

## (Semana-11)

*Exercícios adaptados do livro CSPP  
Randal E. Bryant e David R. O'Hallaron*

Os exercícios deverão ser resolvidos em grupos de 2/3 pessoas e a resolução condensada numa única folha fornecida para o efeito a entregar no final da aula.

**Exercício 1. (Funções):** Considere o trecho de código montado, abaixo, resultante da compilação da função `proc`.

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

1 proc:
2   pushl   %ebp
3   movl    %esp,%ebp
4   subl    $24,%esp
5   addl    $-4,%esp
6   leal    -4(%ebp),%eax
7   pushl   %eax
8   leal    -8(%ebp),%eax
9   pushl   %eax
10  pushl   $.LC0           ;1º argumento usado como apontador para a sequência %x %x"
11  call    scanf
12  movl    -8(%ebp),%eax
13  movl    -4(%ebp),%edx
14  subl    %eax,%edx
15  movl    %edx,%eax
16  movl    %ebp,%esp
17  popl    %ebp
18  ret
```

Tendo em atenção que

- imediatamente, antes da execução (linha 1) `%esp=0x800040` e `%ebp=0x800060`;
- a chamada `scanf` (linha 12), 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`.

- Que valor é colocado no registo `%ebp`, na linha 3?
- Em que endereços estão localizadas as variáveis locais `x` e `y`?
- Qual é o valor de `%esp` antes da chamada a `scanf` (linha 11)?
- Desenhe a área de activação 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*.

**Exercício 2. (Vectores):** Complete a tabela, abaixo, considerando as declarações de tipos de dados que seguem: `short S[7]`; `short *T[3]`; `short **U[6]`; `long double V[8]`; `long double *W[4]`.

Vector	Espaço/elemento	Espaço total	Endereço Inicial	Expressão/elemento <i>i</i>
S			$x_S$	
T			$x_T$	
U			$x_U$	
V			$x_V$	
W			$x_W$	

**Exercício 3. (Estruturas):** Considerando que o registo `%edx` foi iniciado com o valor da variável `r` definida de acordo com as declarações que seguem, 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
2  movl    %eax,4(%edx)
3  leal   8(%edx,%eax,4),%ecx
```

b)

```
1  movl    4(%edx),%eax
2  addl    (%edx),%eax
3  leal   8(%edx,%eax,4),%eax
4  movl    %eax,20(%edx)
```

**Exercício 4. (Laço for):** Pretende-se completar a escrita da função `loop`, de que se conhece apenas a estrutura geral, de modo a obter por compilação, usando o `gcc`, o trecho de código montado, abaixo:

```
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
2  movl    16(%ebp),%edx
3  xorl    %eax,%eax
4  decl    %edx
5  js     .L4
6  movl    %ebx,%ecx
7  imull   12(%ebp),%ecx
8  .p2align 4,,7           ; alinha o código na memória para otimizar a cache

9  .L6:
10 addl    %ecx,%eax
11 subl    %ebx,%edx
12 jns    .L6

13 .L4:           ; terminação do laço
```

Para solucionar o problema sugere-se que comente o código montado por forma a estabelecer uma relação directa entre os registos IA32 e as variáveis na função, tendo em **atenção**:

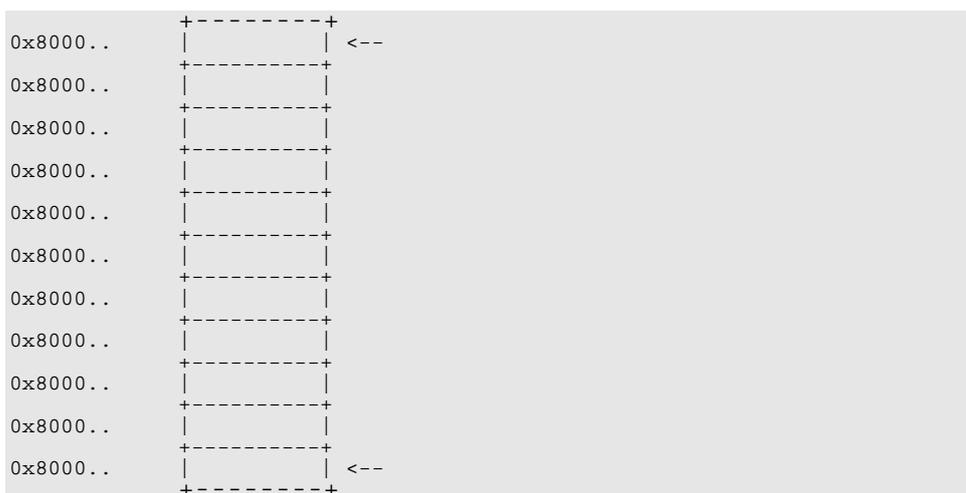
- a existência de uma estrutura de controlo (`for`);
- a atribuição de um valor inicial à variável `i`;
- que por convenção o valor de retorno de uma função é devolvido no registo `%eax`;
- o compilador retirou a expressão que incrementa a variável `result` do interior do ciclo;

**Exercício 1.**

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

```
1 proc:
2   pushl   %ebp
3   movl    %esp,%ebp
4   subl   $24,%esp
5   addl   $-4,%esp
6   leal   -4(%ebp),%eax
7   pushl  %eax
8   leal   -8(%ebp),%eax
9   pushl  %eax
10  pushl  $.LC0          ; 1º argumento usado como apontador para a sequência %x %x"
11  call   scanf
12  movl   -8(%ebp),%eax
13  movl   -4(%ebp),%edx
14  subl   %eax,%edx
15  movl   %edx,%eax
16  movl   %ebp,%esp
17  popl   %ebp
18  ret
```

O desenho da pilha deverá revelar uma estrutura e conteúdo que deverá corresponder ao preenchimento de um quadro como o que a seguir se apresenta. Notar que falta completar os endereços das células de memória.



**Exercício 2.**

Vector	Espaço/elemento	Espaço total	Endereço Inicial	Expressão/elemento <i>i</i>
S			$x_S$	$x_S + 2 i$
T			$x_T$	
U			$x_U$	
V			$x_V$	
W			$x_W$	

### Exercício 3.

a)

```

1  movl    (%edx), %eax
2  movl    %eax, 4(%edx)
3  leal   8(%edx, %eax, 4), %ecx

```

b)

```

1  movl    4(%edx), %eax
2  addl   (%edx), %eax
3  leal   8(%edx, %eax, 4), %eax
4  movl   %eax, 20(%edx)

```

### Exercício 4.

Notar que o processo de compilação de estruturas de controlo usado pelo *gcc* usa implicitamente um pré-processamento para uma versão equivalente em C que recorre à declaração de etiquetas e instruções *goto*, que não aparecem habitualmente no código original C.

```

1  movl    8(%ebp), %ebx
2  movl    16(%ebp), %edx
3  xorl   %eax, %eax
4  decl   %edx
5  js     .L4
6  movl   %ebx, %ecx
7  imull  12(%ebp), %ecx
8  .p2align 4,,7           ; alinha o código na memória para otimizar a cache

9  .L6:
10  addl   %ecx, %eax
11  subl   %ebx, %edx
12  jns    .L6

13 .L4:           ; terminação do laço

```