Guião VI - Resolução

Exercício 1 *(Ciclo **Do-While**)*

a) A compilação da função **dw_loop** com o `gcc`

```
1   int dw_loop(int x, int y, int n)
2   {
3       do {
4           x += n;
5           y *= n;
6           n--;
7       } while ((n > 0) & (y < n)); // O operador usado é o E-lógico '&'
8                                    // e não a conjunção '&&'
9       return x;
10  }
```

resultou no código de montagem seguinte. Adicione em cada uma das linhas os comentários necessários à sua compreensão.

```
1   movl    8(%ebp),%esi      ; %esi = x
2   movl    12(%ebp),%ebx     ; %ebx = y
3   movl    16(%ebp),%ecx     ; %ecx = n
4   .p2align 4,,7             ; alinha o código na memória para otimizar a cache
5 .L6:                        ; início do ciclo do-while
6   imull   %ecx,%ebx         ; (%ebx=) y = y * n
7   addl    %ecx,%esi         ; (%esi=) x = x + n
8   decl    %ecx              ; n--
9   testl   %ecx,%ecx         ; n & n -> afeta flags ⇔ operação E-lógico bit a bit
10  setg    %al               ; se (n > 0) %al=1 senão %al=0
11  cmpl    %ecx,%ebx         ; Faz y-n -> afeta flags
12  setl    %dl               ; se (y < n) %dl=1 senão %dl=0 (8 bits)
13  andl    %edx,%eax         ; (n > 0) & (y < n) operação E-lógico bit a bit
14  testb   $1,%al            ; verifica se bit 0 de (n > 0) & (y < n) = 1
15  jne     .L6               ; Salta para início do ciclo do-while se bit 0 != 0
```

Nota 1: Os registos %al e %dl correspondem aos *8 bits* menos significativos dos registos %eax e %edx, respetivamente.
Nota 2: O compilador usa uma forma pouco usual de avaliar a expressão de teste. Com efeito, no pressuposto que as duas condições de saída (n > 0) e (y < n) apenas podem tomar os valores de 0 ou 1, basta averiguar o valor (0/1) do bit menos significativo do resultado do `E-lógico`. Em alternativa poderia ter sido usada a instrução testb para efetuar a operação E-lógico.

b) Construa uma tabela de utilização de registos:

Registo	Variável	Atribuição inicial
%esi	x	x
%ebx	y	y
%ecx	n	n
%al	--- (guarda cálculos intermédios)	(n > 0)
%dl	--- (guarda cálculos intermédios)	(y < n)

c) Identifique a expressão de **teste** e o **corpo do ciclo** incluído no código fonte C da função, estabelecendo a correspondência com as linhas no código de montagem produzido pela compilação.

O corpo do ciclo **do-while** encontra-se nas linhas 4 a 6 no código C e nas linhas 6 a 8 do código de montagem. A expressão de teste está na linha 7 do código C e corresponde, no código de montagem, às instruções das linhas 9 a 14 e à instrução de salto da linha 15.

Exercício 2 *(Ciclo **While**)*

Para a função **`loop_while`** e código de montagem que resulta da sua compilação, resposta às mesmas perguntas do exercício anterior.

```
1 int loop_while(int a, int b)
2 {
3   int i = 0;
4   int result = a;
5   while (i < 256) {
6       result += a;
7       a -= b;
8       i += b;
9    }
10   return result;}
```

a) Adicione em cada uma das linhas os comentários necessários à sua compreensão.

```
1    movl   8(%ebp),%eax     ; %eax = a
2    movl   12(%ebp),%ebx    ; %ebx = b
3    xorl   %ecx,%ecx        ; (%ecx=) i = 0
4    movl   %eax,%edx        ; (%edx=) result = a
5    .p2align 4,,7
6 .L5:                       ; início do ciclo while
7    addl   %eax,%edx        ; result = result + a
8    subl   %ebx,%eax        ; a = a - b
9    addl   %ebx,%ecx        ; i = i + b
10   cmpl   $255,%ecx        ; Faz i-255 -> afeta as flags
11   jle    .L5              ; Se i <= 255 salta para o início do ciclo while
12   movl   %edx,%eax        ; Prepara o retorno da função ⇔ coloca result em %eax
```

b) Construa uma tabela de utilização de registos.

Registo	Variável	Atribuição inicial
%eax	a	a
%ebx	b	b
%ecx	i	0
%edx	result	a

c) Identifique a expressão de **teste** e o **corpo do ciclo** incluído no código fonte C da função, estabelecendo a correspondência com as linhas no código de montagem produzido pela compilação.

A expressão de teste aparece na linha 5 do código C e no código de montagem nas linhas 10 e 11 (salto).
O corpo do ciclo **while** está nas linhas 6 a 8 do código C, ao qual correspondem as linhas 7 a 9 no código de montagem. O compilador detetou que o teste inicial do ciclo **while** é sempre verdadeiro, uma vez que sendo i iniciado a 0 (zero) o seu valor é sempre inferior a 256. Nesta assunção, o teste inicial normalmente associado a um **while** pode ser evitado. O código C (usando um **goto)** equivalente ao código de montagem anterior é:

```
1 int loop_while_goto(int a, int b)
2 {
3    int i = 0;
4    int result = a;
5    loop:
6    result += a;
7    a -= b;
8    i += b;
9    if (i <= 255)
10     goto loop;
11   return result;}
```

d) Que otimizações foram feitas pelo compilador?

- Utilização de registos (%eax, %ebx, %ecx e %edx) para guardar as variáveis (a, b, i, result) por forma a evitar acessos desnecessários à memória.
- Transformação do while num "do while". O compilador detetou que o teste inicial do ciclo seria executado pelo menos uma vez já que i = 0 é obviamente inferior a 256.
- Uso da instrução xorl %ecx, %ecx em vez de movl $0,%ecx que é mais eficiente porque a instrução não necessita de bytes extras (valor imediato) para representar a constante 0. Esta instrução permite pôr o registo %ecx a 0, através da operação lógica OU exclusivo, por exemplo: %ecx^ %ecx = 00...00 qualquer que seja o valor de 'i', no código C corresponde à expressão i = 0; a instrução `xorl` da arquitetura IA32 requer 2 bytes (xorl %ecx,%ecx ⇔ 31 c0) enquanto a instrução movl $0,%ecx requer 5 bytes ⇔ b8 00 00 00 00.

Exercício 3 ***(Apontadores)*:**

Considere que o apontador para o início do *vector* **S** (do tipo *short int*) e o índice ***i*** (do tipo *int*) estão armazenados nos registos ***%edx*** e ***%ecx***, respetivamente.

Apresente, para cada uma das expressões abaixo:

i) a respetiva declaração de **tipo de dados**;

ii) uma fórmula de cálculo do **valor**;

iii) uma **instrução em IA32** que coloca aquele resultado, no registo %eax (no caso do tipo *short* *) ou em alternativa no registo %ax (no caso do tipo *short*).

Considerando que **S** corresponde ao valor do apontador para o vetor temos:

Expressão	Tipo de Dados	Valor	Instrução
`S+1`	`short *`	`S + 1*2 = S+2`	`leal 2(%edx),%eax`
`S[3]`	`short`	`M[S + 3*2] = M[S+6]`	`movw 6(%edx),%ax`
`&S[i]`	`short *`	`S + 2*i = S+2*i`	`leal (%edx,%ecx,2),%eax`
`S[4*i+1]`	`short`	`M[S+4*2*i+1*2] = M[S+8*i+2]`	`movw 2(%edx,%ecx,8),%ax`
`S+i-5`	`short *`	`S + 2*i - 5*2 = S+2*i-10`	`leal -10(%edx,%ecx,2),%eax`

Nota : O compilador pode substituir instruções como movw 6(%edx),%ax `(e.g S[3])` pela instrução movswl 6(%edx),%eax**.** Esta instrução faz a extensão com o sinal dos 2 bytes originais em memória para obter os 4 bytes do registo de destino %eax.

Exercício 4 (*Estruturas*):

O procedimento **`sp_init`** (com algumas expressões omitidas) trabalha com o tipo de dados **`struct prob`**:

```
struct prob {
    int *p;
    struct {
        int x;
        int y;
    } s;
    struct prob *next;
};

void sp_init(struct prob *sp)
{
    sp->s.x  = _________;
    sp->p    = _________;
    sp->next = _________;
}
```

a) Quantos bytes são necessários para representar a estrutura?

2 * 4 (inteiro) + 2 *4 (apontador) = 16 bytes

b) Qual o valor do deslocamento em relação ao início do vetor (em número de bytes) de cada campo?

A organização da memória correspondente à estrutura ***prob*** é:

Deslocamento	0	4	8	12
Campo	**p**	**s.x**	**s.y**	**next**

c) Considerando que após compilar **`sp_init`**, o código de montagem correspondente ao corpo dessa função é o que se segue:

```
1   movl  8(%ebp),%eax    ; %eax=sp (endereço da estrutura em memória)
2   movl  8(%eax),%edx    ; Obter sp->s.y (sp+8)
3   movl  %edx,4(%eax)    ; Copiar sp->s.y para sp->s.x (sp+4)
4   leal  4(%eax),%edx    ; Obter &(sp->s.x)
5   movl  %edx,(%eax)     ; Copiar &(sp->s.x) para sp->p (sp+0)
6   movl  %eax,12(%eax)   ; Copiar sp para sp->next (sp+12)
```

Preencher as expressões em falta, assinaladas com "_____", no código C da função.

```
void sp_init(struct prob *sp)
{
     sp->s.x  = sp->s.y ;
     sp->p    = &(sp->s.x) ;
     sp->next = sp ;
}
```