

Nº 95076

Nome: Pedro Marcelo Bogas Oliveira

Turma: MEEI

**Resolução dos exercícios (deve ser redigido manualmente)****1. Código em assembly**

Escreva aqui o código otimizado em *assembly* (tal como está no ficheiro `while_loop.s`) devidamente anotado, i.e., com comentários à frente de cada instrução, comentários esses que (i) expliquem o que está a acontecer se for a fase de arranque ou término duma função e (ii) mostrem que parte do código C essa instrução em *assembly* está a executar. De seguida, analise o código em *assembly* e preencha a tabela.

```

6 while_loop:
7     pushl    %ebp
8     movl     %esp, %ebp
9     movl     16(%ebp), %edx
10    testl    %edx, %edx
11    pushl    %ebx
12    movl     12(%ebp), %eax
13    movl     8(%ebp), %ebx
14    jle      .L3
15    movl     %edx, %ecx
16    sall     $4, %ecx
17    cmpl     %ecx, %eax
18    jge      .L3
19    .p2align 2,,3
20 .L6:
21    addl     %edx, %ebx
22    imull    %edx, %eax
23    decl     %edx
24    subl     $16, %ecx
25    testl    %edx, %edx
26    jle      .L3
27    cmpl     %ecx, %eax
28    jl       .L6
29 .L3:
30    movl     %ebx, %eax
31    popl     %ebx
32    leave
33    ret

```

*arranque* → atualiza o base pointer

carregar o argumento n em %edx

carregar o argumento y em %eax

carregar o argumento x em %ebx

Variável	Registro	Atribuição inicial
x	%ebx	valor do argumento x
y	%eax	valor do argumento y
n	%edx	valor do argumento n

```

1 int while_loop(int x, int y, int n)
2 {
3     while((n>0)&&(y<16*n)){
4         x+=n;
5         y*=n;
6         n--;
7     }
8     return x;
9 }

```

*termino*

atualiza o stack pointer e recupera o valor anterior do base pointer

Escreva aqui o código C de um programa simples (main) que usa a função `while_loop`:

```

int while_loop(int x, int y, int n)
{
    while((n>0)&&(y<16*n)){
        x+=n;
        y*=n;
        n--;
    }
    return x;
}

int main () {
    int i;
    i = while_loop(4,2,3);

    return 0;
}

```

Apresente aqui o código executável depois de desmontado com o comando `objdump -d`.

Assinale neste pedaço de código as instruções que vão buscar à *stack* cada um dos 3 argumentos que foram passados para a função, para os colocar em registos.

Registe (marque) os endereços das instruções imediatamente a seguir a essas, para que sejam os pontos de paragem a introduzir na execução do código.

Uma vez parada a execução do código nesses endereços, vamos poder ver o conteúdo dos registos que receberam os argumentos, confirmando parte da tabela da página anterior.

```

08048310 <while_loop>:
08048310: 55          push    %ebp
08048311: 89 e5       mov     %esp,%ebp
08048313: 90          nop
08048314: 83 7d 10 00  cmpl   $0x0,0x10(%ebp) = n>0
08048318: 7e 24       jle     0804833e <while_loop+0x2e>
0804831a: 8b 45 10     mov     0x10(%ebp),%eax ← argumento n → %eax
0804831d: c1 e0 04     shl     $0x4,%eax = 16*n
08048320: 39 45 0c     cmp     %eax,0xc(%ebp) = y < 16*n
08048323: 7c 02       jl      08048327 <while_loop+0x17>
08048325: eb 17       jmp     0804833e <while_loop+0x2e>
08048327: 8b 45 10     mov     0x10(%ebp),%eax ← argumento n → %eax
0804832a: 01 45 08     add     n,%eax,0x8(%ebp) ← argumento x → 0x8(%ebp)
0804832d: 01 45 0c     mov     0xc(%ebp),%eax ← argumento y → %eax
08048330: 0f af 45 10  imul    n,0x10(%ebp),%eax ← y*=n
08048334: 89 45 0c     mov     %eax,0xc(%ebp) ← y → 0xc(%ebp)
08048337: 8d 45 10     lea     0x10(%ebp),%eax ← argumento n → %eax
0804833a: ff 08       decl    (%eax)n ← n--
0804833c: eb d6       jmp     08048314 <while_loop+0x4>
0804833e: 8b 45 08     mov     0x8(%ebp),%eax ← n → %eax
08048341: c9          leave   4(%eax)
08048342: c3          ret

08048343 <main>:
08048343: 55          push    %ebp
08048344: 89 e5       mov     %esp,%ebp
08048346: 83 ec 08     sub     $0x8,%esp
08048349: 83 e4 f0     and     $0xffffffff0,%esp
0804834c: b8 00 00 00 00  mov     $0x0,%eax
08048351: 29 c4       sub     %eax,%esp
08048353: 83 ec 04     sub     $0x4,%esp
08048356: 6a 03       push    $0x3
08048358: 6a 02       push    $0x2
0804835a: 6a 04       push    $0x4
0804835c: e8 af ff ff ff  call    08048310 <while_loop>
08048361: 83 c4 10     add     $0x10,%esp
08048364: 89 45 fc     mov     %eax,0xffffffffc(%ebp)
08048367: b8 00 00 00 00  mov     $0x0,%eax
0804836c: c9          leave   4(%eax)
0804836d: c3          ret
0804836e: 90          nop
0804836f: 90          nop

```

está a apontar para o primeiro elemento da stack

← argumento n → %eax

← argumento x → 0x8(%ebp)

← argumento y → %eax

← y\*=n

← y → 0xc(%ebp)

← argumento n → %eax

← n → %eax

return

(%esp) →

4	(x)
2	(y)
3	(n)

põe os argumentos na stack

Escreva aqui os endereços das instruções onde vai inserir pontos de paragem (*breakpoints*) quando usar o depurador `gdb`.

0x804832d  
0x8048330  
0x8048341

Antes de executar qualquer código, coloque aqui a sua estimativa do que irá estar nos registos após cada ponto de paragem:

Variável	Registo	Break1	Break2	Break3	Break_	Break_
x	%eax			10		
y	%eax		2			
n	%eax	3				

Após execução do código de modo controlado (dentro do depurador), preencha de novo essa tabela com os valores que efetivamente leu quanto ao conteúdo dos registos, após cada ponto de paragem.

Variável	Break_
x	
y	
n	

```

[01a95076@sc 01a95076]$ ls
a.out  while_loop.c  while_loop.o  while_loop.s
[01a95076@sc 01a95076]$ gdb while_loop.o
GNU gdb Red Hat Linux (6.3.0-1.138.el3rh)
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386-redhat-linux-gnu"...
(no debugging symbols found)
Using host libthread_db library "/lib/tls/libthread_db.so.1".

(gdb) break *0x804832d
Breakpoint 1 at 0x804832d
(gdb) break *0x8048330
Breakpoint 2 at 0x8048330
(gdb) break *0x8048341
Breakpoint 3 at 0x8048341
(gdb) run
Starting program: /home/01a95076/while_loop.o
/bin/bash: line 1: /home/01a95076/while_loop.o: Permission denied
/bin/bash: line 1: exec: /home/01a95076/while_loop.o: cannot execute: Success

Program exited with code 0176.
You can't do that without a process to debug.

```

Preencha aqui os valores pedidos no enunciado relativamente à *stack frame* desta função, nomeadamente alguns endereços relevantes, o conteúdo de cada conjunto de 4 células de memória e uma interpretação de cada um desses conteúdos.

Os valores a colocar aqui deverão ser os valores lidos da memória do servidor no 2º ponto de paragem.

Endereço 1ª célula      Conteúdo em hex      Conteúdo comentado

	%ebx	→ guardar antigo %ebx
	08 04 83 61	Endereço de regresso
	00 00 00 04	→ valor do 1º argumento
	00 00 00 02	→ valor do 2º argumento
	00 00 00 03	→ valor do 3º argumento

Nota: neste diagrama, cada caixa representa um bloco de 32-bits em 4 células.