

# TPC6 e Guião Laboratorial

## Resolução dos exercícios

### 1. Anotação do código em assembly do ciclo while

O código gerado na compilação de ciclos pode ser complicado de analisar, devido aos diferentes tipos de otimização do código do ciclo que o compilador poderá optar, para além da dificuldade em mapear variáveis do programa a registos do CPU. Para se adquirir alguma técnica, nada como começar com um ciclo relativamente simples.

Eis o código *assembly* que o comando `gcc -S -O2` irá gerar na máquina remota, anotado manualmente, por vezes com alguma informação excessiva para um melhor esclarecimento:

```

while_loop:
    pushl %ebp          ; salva o frame pointer da função chamadora (%ebp) na stack
    movl %esp, %ebp     ; define o valor do frame pointer desta função: %esp -> %ebp
    movl 16(%ebp), %edx ; vai buscar o 3º argumento à stack e coloca-o em registo: n -> %edx
    testl %edx, %edx    ; testa o valor de n (fazendo n<AND>n, que é =n), alterando só as flags;
    pushl %ebx          ; salva o conteúdo de %ebx na stack, pois vai precisar deste registo
    movl 12(%ebp), %eax ; vai buscar o 2º argumento à stack e coloca-o em registo: y -> %eax
    movl 8(%ebp), %ebx  ; vai buscar o 1º argumento à stack e coloca-o em registo: x -> %ebx
    jle .L3             ; salta para o fim do while se as flags indicarem que n≤0 (less or equal)
    movl %edx, %ecx     ; cria uma variável temp em %ecx para calcular 16*n: temp= n
    sall $4, %ecx       ; um shift aritmético de 4 bits para a esq de temp é equivalente a temp=n*24
    cmpl %ecx, %eax     ; compara y com 16*n, i.e., altera as flags com a operação y-16*n
    jge .L3             ; salta p/ fim do while se as flags indicarem que y≥16*n (greater or equal)
    .p2align 2, , 3      ; pad-to-align: acrescentar instruções no-ops para que o endereço da próxima
                        ; instrução seja múltiplo de 4; compilação com -O2 pode ir mais longe
.L6:                  ; as instruções anteriores testaram a condição p/ executar o ciclo while
    addl %edx, %ebx     ; aqui começa o corpo do ciclo while com x += n
    imull %edx, %eax    ; y *= n
    decl %edx           ; n--
    subl $16, %ecx      ; p/ evitar calcular de novo 16*n o compilador optou por temp=temp-16
    testl %edx, %edx    ; estas 3 instruções repetem o cálculo da condição do ciclo while
    jle .L3             ; na próxima instrução, em vez de saltar para fora do ciclo se y≥16*n
    cmpl %ecx, %eax     ; salta para o início do ciclo while se ainda se mantiver y<16*n
    jl .L6              ; .L6:
    movl %ebx, %eax     ; coloca o valor a devolver (x) no registo convencionado para tal (%eax)
    popl %ebx           ; recupera o conteúdo de %ebx, que tinha sido salvaguardado no início
    leave               ; coloca o %esp a apontar para o mesmo endereço na stack que o frame
                        ; pointer (em %ebp) e recupera o frame pointer da função chamadora
    ret                 ; regressa à função chamadora, recuperando o IP que está no topo da pilha

```

A análise do modo como os argumentos são recuperados no código da função dá-nos uma boa pista de como o `gcc` usa os registos no cálculo de expressões de teste.

O valor de  $n * 16$ , utilizado numa das condições do ciclo, é pré-calculado e guardado em `%ecx`; em cada iteração do ciclo este valor é atualizado subtraindo 16, evitando-se assim a multiplicação  $n * 16$ , uma vez que  $n$  é decrementado de 1 unidade.

Utilização dos Registos		
Registo	Variável	Atribuição inicial
<b>%ebx</b>	x	valor recebido do 1º arg (x)
<b>%eax</b>	y	valor recebido do 2º arg (y)
<b>%edx</b>	n	valor recebido do 3º arg (n)
<b>%ecx</b>	temp ( $=n * 16$ )	o valor recebido do 3º arg (n) e logo a seguir $16 * n$

## 2. Código do main em C

```

1 int main() {
2     int a=4, b=3, c=2, x;
3     x = while_loop (a, b, c);
4     printf ("%d\n", x);
5     return x;
6 }
```

## 3. Validação da utilização dos registos

Para confirmar esta utilização dos registos, procede-se conforme sugerido na alínea **b)** do guião:

- i. coloca-se no mesmo ficheiro fonte onde estava o código do ciclo, um programa simples (`main`) que chame a função `while_loop`, passando como argumentos para a função os valores 4, 2 e 3, respetivamente;
- ii. compila-se com os *switches* indicados e desmonta-se o executável com o comando `objdump -d`;
- iii. analisando a parte do código desmontado com a função `while_loop` procura-se a 1ª instrução a seguir à inicialização de cada uma das variáveis e regista-se o endereço onde se encontram essas instruções (e ainda após `temp` passar a ser  $16 * n$ );
- iv. invocando o *debugger* (`gdb`), introduzem-se pontos de paragem (*breakpoints*) nesses endereços, os quais irão depois permitir que a execução do código seja interrompida nesses endereços para se verificar o conteúdo dos registos;
- v. com os comandos apropriados do `gdb` manda-se executar o programa (comando `run`) e ele vai sendo interrompido de cada vez que encontra um *breakpoint*;
- vi. uma vez a execução do programa interrompida num *breakpoint*, pode-se então validar o conteúdo dos registos (por ex., com `info reg`), confirmando as estimativas que foram indicadas atrás;
- vii. após cada *breakpoint* continuar a execução do código (comando `cont`) até terminar a execução do programa.

Código desmontado da função `while_loop` com indicação dos endereços para os *breakpoints*:

<b>Onde colocar breakpoints</b>	<pre> 0x08048354 &lt;while_loop&gt;: 0x08048354:    55          push    %ebp 0x08048355:    89 e5       mov     %esp,%ebp 0x08048357:    8b 55 10   mov     0x10(%ebp),%edx 0x0804835a:    85 d2       test    %edx,%edx 0x0804835c:    53          push    %ebx 0x0804835d:    8b 45 0c   mov     0xc(%ebp),%eax 0x08048360:    8b 5d 08   mov     0x8(%ebp),%ebx 0x08048363:    7e 1c       jle    0x8048381 &lt;while_loop+45&gt; 0x08048365:    89 d1       mov     %edx,%ecx 0x08048367:    c1 e1 04   shr    \$0x4,%ecx 0x0804836a:    39 c8       cmp    %ecx,%eax 0x0804836c:    7d 13       jge    0x8048381 &lt;while_loop+45&gt; 0x0804836e:    89 f6       mov     %esi,%esi 0x08048370:    01 d3       add    %edx,%ebx 0x08048372:    0f af c2   imul   %edx,%eax 0x08048375:    4a          dec    %edx 0x08048376:    83 e9 10   sub    \$0x10,%ecx 0x08048379:    85 d2       test   %edx,%edx 0x0804837b:    7e 04       jle    0x8048381 &lt;while_loop+45&gt; 0x0804837d:    39 c8       cmp    %ecx,%eax 0x0804837f:    7c ef       jl    0x8048370 &lt;while_loop+28&gt; 0x08048381:    89 d8       mov    %ebx,%eax 0x08048383:    5b          pop    %ebx 0x08048384:    c9          leave  0x08048385:    c3          ret </pre>	<div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> <b>Inicialização de registos com as variáveis x, y, n e temp</b> </div> <div style="border: 1px solid black; padding: 5px; margin-bottom: 10px;"> <b>Instrução nova inserida aqui pelo gcc ao criar o executável, que não faz nada de útil, apenas força que o início do corpo do ciclo (que está na instrução seguinte) comece num endereço que é múltiplo de 16</b> </div>
---------------------------------	---	--

Registo	Variável	Break1	Break2	Break3	Break4
%ebx	x	lixo	<b>4</b>	<b>4</b>	<b>4</b>
%eax	y	lixo	<b>2</b>	<b>2</b>	<b>2</b>
%edx	n	<b>3</b>	<b>3</b>	<b>3</b>	<b>3</b>
%ecx	temp (=n*16)	lixo	lixo	<b>3</b>	<b>48</b>

#### 4. A stack frame da função

Pretende-se neste exercício que se preencham 3 tipos de informação:

- i. à esquerda do desenho da *stack*, os endereços do início de algumas "caixas";
- ii. no interior das "caixas" o valor numérico que lá deveria estar (pode ser em hexadecimal);
- iii. à direita das "caixas", uma explicação do valor que se encontra na respectiva "caixa".

Cada "caixa" não é mais que um bloco de 32 bits armazenado em 4 células de 1 byte cada, em que o conteúdo da célula com menor endereço é o byte mais à direita de um valor de 32 bits (*little endian*).

A resolução completa implica uma análise detalhada do código gerado pelo `gcc`, do conteúdo de alguns registos e da localização em que deverá ser executado no PC, entre outros aspetos. Por exemplo, no `gdb` é possível visualizar o código da função `main` usando o comando `disas main`:

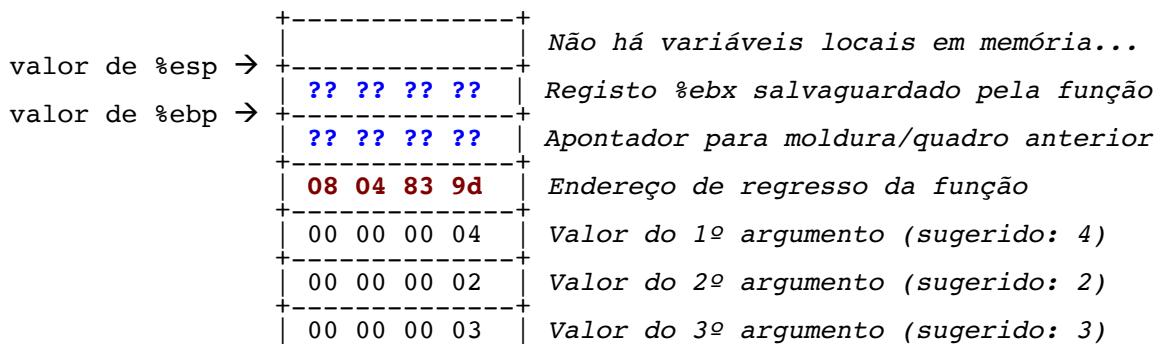
```

0x08048388 <main+0>:    push    %ebp
0x08048389 <main+1>:    mov     %esp,%ebp
0x0804838b <main+3>:    sub     $0x8,%esp
0x0804838e <main+6>:    and     $0xffffffff0,%esp
0x08048391 <main+9>:    push    %eax
0x08048392 <main+10>:   push    $0x3
0x08048394 <main+12>:   push    $0x2
0x08048396 <main+14>:   push    $0x4
0x08048398 <main+16>:   call    0x8048354 <while_loop>
0x0804839d <main+21>:  leave
0x0804839e <main+22>:  ret

```

Neste caso, o valor do endereço de regresso que deverá estar na *stack* deverá ser o endereço da instrução na `main` imediatamente a seguir à invocação da função (após a instrução `call`), ou seja `0x0804839d` (note que os endereços podem variar ligeiramente no seu caso, dependendo da forma como o código C está escrito).

Esta é a estimativa do conteúdo da *stack frame* associada à função `while_loop`, quando é invocada com os argumentos sugeridos e antes de se confirmar com o `gdb`:



O valor dos registos salvaguardados, incluindo o apontador para o quadro (*frame pointer*) da `main` na `stack`, i.e., tudo o que está com ?? na figura) pode ser obtido no `gdb`, parando a execução do código logo na 1<sup>a</sup> instrução da função (coloque aí um *breakpoint*), e analisando o conteúdo desses registos (que ainda não foram colocados na `stack`).

Para confirmar os conteúdos destas 24 posições de memória na `stack`, coloca-se outro *breakpoint* no fim da parte de arranque da função, i.e., após a salvaguarda do registo `%ebx`; quando o programa parar aí pode-se então usar um dos comandos para examinar dados do *debugger* e visualizar o conteúdo das 24 células com início no topo da pilha (valor em `%esp`), quer byte a byte (dá para ver o funcionamento *little endian*), quer em 6 blocos de 4 bytes.

#### Sugestão de procedimento a seguir, passo a passo:

(i) Escrevendo disas `while_loop` tem-se acesso novamente ao código da função:

```

0x08048354 <while_loop+0>:    push    %ebp
0x08048355 <while_loop+1>:    mov     %esp, %ebp
0x08048357 <while_loop+3>:    mov     0x10(%ebp), %edx
0x0804835a <while_loop+6>:    test    %edx, %edx
0x0804835c <while_loop+8>:    push    %ebx
0x0804835d <while_loop+9>:    mov     0xc(%ebp), %eax
0x08048360 <while_loop+12>:   mov     0x8(%ebp), %ebx
0x08048363 <while_loop+15>:   jle    0x8048381 <while_loop+45>
0x08048365 <while_loop+17>:   mov     %edx, %ecx
0x08048367 <while_loop+19>:   shl    $0x4, %ecx
0x0804836a <while_loop+22>:   cmp    %ecx, %eax
0x0804836c <while_loop+24>:   jge    0x8048381 <while_loop+45>
0x0804836e <while_loop+26>:   xchg   %ax, %ax
0x08048370 <while_loop+28>:   add    %edx, %ebx
0x08048372 <while_loop+30>:   imul   %edx, %eax
0x08048375 <while_loop+33>:   dec    %edx
0x08048376 <while_loop+34>:   sub    $0x10, %ecx
0x08048379 <while_loop+37>:   test   %edx, %edx
0x0804837b <while_loop+39>:   jle    0x8048381 <while_loop+45>
0x0804837d <while_loop+41>:   cmp    %ecx, %eax
0x0804837f <while_loop+43>:   jl    0x8048370 <while_loop+28>
0x08048381 <while_loop+45>:   mov    %ebx, %eax
0x08048383 <while_loop+47>:   pop    %ebx
0x08048384 <while_loop+48>:   leave
0x08048385 <while_loop+49>:   ret

```

(ii) coloca-se um *breakpoint* após `push %ebx` (assinalado em cima):

```
break *0x804835d
```

(iii) executa-se o programa (`run`) e ele parará no *breakpoint*; visualiza-se o conteúdo dos registos escrevendo `info reg`; para responder integralmente a esta questão precisamos de saber o valor do apontador para o topo da pilha (registo `%esp`) e o apontador para o quadro da função, o *frame pointer* (registo `%ebp`).

(iv) examinam-se 6 "palavras" na memória a partir do endereço em `%esp`:

```
x/6wx $esp
```

0xbffffe874:	0x007d3fff4	0xbffffe898	0x0804839d	0x00000004
0xbffffe884:	0x00000002	0x00000003		

(v) preenche-se agora a figura com estes valores, conforme pedido:

	+-----+	<i>Não há variáveis locais em memória...</i>
0xbffffe874 →	00 7d 3f f4	<i>Registo %ebx salvaguardado pela função</i>
0xbffffe878 →	bf ff e8 98	<i>Apontador para moldura/quadro anterior</i>
	08 04 83 9d	<i>Endereço de regresso da função</i>
	00 00 00 04	<i>Valor do 1º argumento (sugerido: 4)</i>
	00 00 00 02	<i>Valor do 2º argumento (sugerido: 2)</i>
	00 00 00 03	<i>Valor do 3º argumento (sugerido: 3)</i>

## 5. Estrutura do ciclo while

A expressão de teste é implementada em dois blocos. O primeiro bloco verifica se o ciclo deve ser executado na 1ª iteração:

```
testl    %edx, %edx      ; testa se n é 0; n <AND> n só é zero se n for zero...
jle     .L3                ; salta para fim do ciclo se n <= 0
movl    %edx, %ecx      ; temp = n
sal    $4, %ecx        ; temp = n*16
cmpl   %ecx, %eax      ; compara y com n*16
jge     .L3                ; salta para fim do ciclo se y >= 16
```

O segundo bloco, no fim do corpo do ciclo, verifica se este deve iterar:

```
decl    %edx          ; n--
subl    $16, %ecx      ; temp = temp-16 (=n*16)
testl    %edx, %edx      ; testa se n é 0
jle     .L3                ; salta para o fim do ciclo se n <= 0
cmpl   %ecx, %eax      ; compara y com n*16
jl      .L6                ; salta para o fim do ciclo se y < n*16
```

## 6. Versão do código com goto

Versão do tipo `goto` (em C) da função, com uma estrutura semelhante ao do código *assembly* (tal como foi feito para a série Fibonacci):

```
1 int while_loop_goto(int x, int y, int n)
2 {
3     if (!((n > 0) && (y < n*16))) goto done;
4     loop:
5         x += n;
6         y *= n;
7         n--;
8         if ((n > 0) && (y < n*16)) goto loop;
9     done:
10    return x;
11 }
```

## Adenda

A resolução do TPC6 mostra como deve ser uma resolução correta deste TPC. Contudo, houve muitas resoluções submetidas para avaliação que tinham os exercícios resolvidos de maneira diferente, na maioria dos quais com falhas, lacunas ou erros.

Infelizmente foram submetidas algumas resoluções que mostram que os alunos foram pouco cautelosos na sua resolução (algo que tem justificação quando a resolução é feita numa prova de avaliação com tempo limitado, mas neste caso...), e outras ainda em que os alunos tentaram iludir a equipa docente na parte de identificarem os *breakpoints* e exibirem o conteúdo dos registos quando a execução do código era interrompida (apresentaram o código assembly não do executável desmontado, mas do programa compilado para assembly).

Alguns dos alunos continuam na dúvida se as suas resoluções afinal seriam ou não aceitáveis e, quando não o fossem, porque razão os seus resultados não estariam totalmente corretos.

Serve esta adenda para:

- mostrar algumas resoluções que a equipa docente considera menos corretas (ou mesmo incorretas), cada uma com um comentário sobre o assunto;
- apresentar no fim uma versão mais detalhada da resolução do 1º exercício, explicando integralmente o funcionamento do código associado a uma função e sua anotação.

Para começar, um exemplo em como não foi lido sequer a indicação inicial a vermelho e a *bold*, o que conduziu a que a resolução fosse classificada com zero:

Resolução dos exercícios ( <b>deve ser redigido manualmente</b> )															
<b>1. Código em assembly</b>															
<p>Escreva aqui o código otimizado em assembly (tal como está no ficheiro devidamente anotado, i.e., com comentários à frente de cada instrução, comentários que expliquem o que está a acontecer se for a fase de arranque ou término duma função, que parte do código C essa instrução em assembly está a executar. De seguida, em assembly e preencha a tabela).</p>															
<pre>while_loop:     pushl %ebp; ;(salvaguarda %ebp (geral antes do corpo)     movl %esp, %ebp ;(salvaguarda %ebp (geral antes do corpo)     movl 16(%ebp), %edx ;(coloca 3º argumento em (n) %edx)     testl %edx, %edx ;(testa n)     pushl %ebx ;(salvaguarda de %ebx (geral antes do corpo)     movl 12(%ebp), %eax ;(coloca 2º argumento (y) em %eax (não afeta a flag)     movl 8(%ebp), %ebx ;(coloca 1º argumento (x) em %ebx (não afeta a flag)     jle .L3 ;Ciclo     movl %edx, %ecx     sall \$4, %ecx     cmpl %ecx, %eax</pre>	<table border="1"> <tr> <td>Registo</td> <td>V</td> </tr> <tr> <td></td> <td></td> </tr> </table>	Registo	V												
Registo	V														

### A1 . Anotação do código em assembly do ciclo while

A anotação de código é para nos ser útil quando analisamos o código assembly, não deve ser usado apenas para explicar o que faz a instrução em assembly!

Precisamos de uma indicação sobre o que está de facto a acontecer, se é algo relacionado com uma instrução do código fonte (em C, por exemplo) ou se é para implementar uma estrutura de controlo, ou ainda se é para o arranque ou término duma função (e o que é que de facto está a acontecer).

#### Exemplo 1:

*movl %esp,%ebp -> copia a informação da fonte (2º argumento, norte (coro ebp)) para o destino (1º argumento, norte coro ebp)*

A anotação introduzida aqui não nos diz nada de útil, apenas explica como funciona uma instrução de *mov!* Quando a dica importante a colocar aqui seria “criar o apontador para o quadro da função, em %ebp, usando o valor do apontador para o topo da pilha, em %esp”.

Aliás o resto das anotações na resolução deste aluno tem sempre o mesmo erro. Em todas as instruções foi introduzida apenas uma explicação para o modo de funcionamento da instrução especificada em assembly, algo totalmente inútil para quem conhece o *instruction set* mas deseja perceber o que faz o código.

### Exemplo 2:

# apresenta 4 bits a zero no final de n. (int d)

A anotação aqui também não introduz qualquer informação útil, até porque não há nada no código fonte que diga para introduzir “4 bits a zero no final de n”. Sabendo-se que em `%ecx` está a variável temporária `temp` com o valor de `n`, que cada deslocação de 1 bit à esquerda de um valor em binário é equivalente a multiplicar esse valor por 2 e que uma deslocação de 4 bits para a esquerda é equivalente a multiplicar por  $2^4$ , poderia ficar anotado que “ $\text{temp} = 2^4 * n$  ou  $\text{temp} = 16 * n$ ”.

### Exemplo 3:

decl %ecx -> n--;  
 sub \$16, %ecx -> retira 16 de ecx

É óbvio que a instrução `sub $16` “retira 16” ao valor do registo!... Mas aqui o que se pretendia saber era o porquê desta subtração, e esta anotação não ajuda.

O que o compilador gerou foi código para evitar novamente a multiplicação  $16 * n$ , uma vez que o valor de `n` acabou de ser decrementado de uma unidade. Portanto se o `temp` (em `%ecx`) tinha o valor de  $16 * n$  e agora pretende-se atualizar o valor de `temp` para  $16 * (n-1)$ , poupa-se essa multiplicação ao subtrair-se de `temp` o valor 16.

E uma subtração é em princípio mais rápida que uma multiplicação...

### Exemplo 4:

pushl %ecx -> puxa o endereço para o topo da stack

Em 1º lugar o verbo “push” em inglês quer dizer “empurrar” e não “puxar” (a tradução desta é “pull”). Portanto aqui seria mais um “empurrar para o topo da stack”, ou simplesmente “empilhar”.

Mas o mais grave é dizer que está a empilhar um “endereço” quando não se sabe o conteúdo desse registo! Mas mesmo trocando a palavra “endereço” por simplesmente “registo”, esta anotação continua a ser inútil. Porque vamos empilhar o valor deste registo?

O que era importante anotar aqui é que o código vai salvaguardar o conteúdo deste registo (porque a função precisa deste registo e a função chamadora poderá ter lá algo útil, competindo à função chamada não alterar o seu conteúdo); e que no fim o seu valor vai ser recuperado, antes de regressar à função chamadora.

### Exemplo 5:

testl %edx, %edx  
 .L3  
 tenta n (verifica os flags)  
 saltar se verificarem certas flags para .L3

Esta instrução de `test` dentro do ciclo `while_loop` vai operar com 2 operandos para alterar `flags` (não é para “verificar”). Ela vai, de facto, realizar a operação  $n \text{ AND } n$  e sabe-se que  $n \text{ AND } n = n$ .

Por isso se diz muitas vezes que se está a testar o valor de  $n$ . E este teste é para saber se  $n$  é zero, positivo, negativo...

Na instrução seguinte a anotação lá colocada não ajuda, pois não diz que flags vai verificar (e mesmo que dissesse, não iria ajudar muito) e que salta para .L3 (que tem de especial este sítio?).

O que deveria lá estar seria algo do género “*salta para fora do ciclo se  $n \leq 0$* ”.

### Exemplo 6:

~~popl %esi  
leave // reverte as ações da instrução entre~~  
~~... - obviamente~~

Alguém conhece a instrução “enter”? Nem com a folha do *instruction set* à mão dá para responder? Há mais que uma resolução com este disparate, dá para pensar se não houve aqui fraude...

### Exemplo 7:

9	pushl %edi	; salvaguarda de %edi
10	pushl %esi	; salvaguarda de %esi
11	movl 16(%ebp), %eax	; coloca o 3º argumento (n) em %eax
12	testl %eax, %eax	; testa $n > 0$
13	pushl %eax	; salvaguarda de %eax
14	movl %eax, %ebx	; coloca o 3º argumento em %ebx
15	setl %dl	; coloca em %dl o valor lógico de ( $n > 0$ )
16	movl 18(%ebp), %esi	; coloca o 2º argumento (y) em %esi
17	Scall \$4, %ebx	; $y = 16 * n$
18	cmpl %ebx, %esi	; calcula $y - (16 * n)$ (afeta as flags em bits de sinalização)
19	setl %al	; coloca em %al o valor lógico de ( $y < 16 * n$ )
20	andl %eax, %eax	; coloca em %eax o valor lógico de ( $(n > 0) \& \& (y < 16 * n)$ )
21	andl \$1, %eax	; limpa %eax exato o bit menos significativo
		; limpa %eax exato o bit menos significativo
		; limpa %eax exato o bit menos significativo

Um exemplo claro em como a compilação não foi feita conforme indicada, i.e., com o nível 2 de otimização (-O2): o código da função precisa de mais um registo (salvaguarda também o %esi) e depois para o cálculo da condição (que era do tipo  $a \& b$ ) calculou e guardou o valor booleano de  $a$  ( $n > 0$ ), depois calculou e guardou o valor booleano de  $b$  ( $y < 16 * n$ ) e depois fez o  $\&$  desses 2 valores.

Se o compilador tivesse gerado código mais eficiente, testava 1º o  $a$ , se falhasse saltava logo para depois do ciclo; senão, testava então o  $b$  e se falhasse fazia o mesmo.

## A2 . Utilização dos registos

Nesta fase do problema pretendia-se uma indicação de como o compilador iria alojar os registos às variáveis do código fonte (quer as explícitas, quer as usadas para cálculos intermédios de expressões aritméticas) e uma estimativa dos valores iniciais de cada variável nos registos.

### Exemplo 1:

Registo	Variável	Atribuição inicial
%eax	x	8 (%ebp)
%ebx	y	12 (%ebp)
%edx	n	16 (%ebp)
%ecx	d	9(%edx)

A lista dos registos está correta, mas os valores que estão nesta tabela como sendo os inicialmente atribuídos não dizem nada a quem não está dentro das regras de especificação de endereços de memória no assembly da GNU. Apenas estão lá endereço de memória especificados à la GNU...

Esta resposta não é aceitável numa prova de avaliação.

### Exemplo 2:

Registo	Variável	Atribuição inicial
%ebx	x	x
%eax	y	y
%edx	n	n
%ecx	16*m	16*m

ou

Registo	Variável	Atribuição inicial
%ebx	x	valor atribuído a x
%eax	y	valor atribuído a y
%edx	n	valor atribuído a n
%ebx	16*m	16 vezes o valor de m

Estas respostas nem merecem ser comentadas...

### A3. O código do main em C

Vão ser apresentados aqui alguns exemplos de código C menos correto, mas que a versão antiga do compilador instalado no servidor remoto não rejeitou.

### Exemplo 1:

```
int main()
{
    while_loop(4,2,3);
    return 0;
}
```

O `while_loop` é uma função, não é um procedimento, logo comporta-se como sendo uma variável (cujo valor é o que a função devolve depois de ser executada).

Quando se escreve código C não é boa prática colocar numa linha apenas o nome duma variável (ou duma função, que é este caso). Um compilador mais atual que o instalado no servidor remoto provavelmente apresentaria aqui um erro de compilação.

Adicionalmente os compiladores devem ter também a preocupação de gerar código eficiente e a maioria dos mais recentes conseguem identificar as partes de código fonte que estão a mais, i.e., que não realizam qualquer trabalho útil, e simplesmente não geram código para essa parte.

Neste caso, o valor devolvido pela função não é nunca usado, portanto não faz sentido gerar código para a função...

### Exemplo 2:

```
int main()
{
    int x;
    x = while_loop(4,2,3);
    return x;
}
```

Aparentemente este código já não devia ser rejeitado pelo compilador, pois o valor devolvido pela função é atribuído à variável `x` e esta é devolvida pela “função” `main`. Mas a função `main`, tal como o próprio nome indica, é o código da parte principal do programa, que por acaso em C se chama também de “função”. Não é pressuposto que o código do `main` devolva um valor...

#### A4 . Validação experimental das atribuições iniciais aos registos com variáveis

Tal como foi sugerido no guião, era necessário criar um executável e, a partir deste, identificar quando os valores das variáveis eram colocados nos registos alocados pelo compilador, de modo a se inserir um ponto de paragem imediatamente a seguir e depois ir ver o que estava nos registos.

Mas isto exigia que se criasse um executável, se desmontasse esse executável (para ver o código executável transformado em assembly) e a partir daí fazer então a análise.

Mas...

#### Exemplo 1:

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

```

0000000000004f0 < main>:
4f0: ba 03 00 00 00    mov    $0x3,%edx
4f5: be 02 00 00 00    mov    $0x2,%esp      |break point|
4fa: bf 04 00 00 00    mov    $0x4,%edi      |break point|
4ff: e9 dc 01 00 00    jmpq   620 <while-loop>|break point|
? |Instruções que se seguem à leitura de conta 4 dos|
? |argumentos da Stack|

```

000000000000620 <while-loop>:

```

620: 85 d2             test   %edx,%edx
622: ff 34             jle    658 <while-loop+0x38>
624: 89 d4             mov    %edx,%ecx(primeira utilização)
626: c1 e1 04             shl   $0x4,%ecx  do registo %ecx
628: 39 ce             cmp    %ecx,%esi |break point|

```

Uma longa lista de falsidades e erros:

- esta listagem não é do código executável desmontado; vê-se logo pelos endereços das instruções;
- esta listagem mistura a `main` com a função `white_loop`, quando apenas se pretendia desta última;
- os *breakpoints* estão todos mal colocados, não percebeu sequer a explicação dada!

#### Exemplo 2:

Disassembly of Section `text`:

```

00000000 <while_loop>:
0: 55              push   %ebp
1: 89 e5            mov    %esp,%ebp
2: 90              nop
3: 83 f7 10        cmp    $0x10,%eax(%ebp)
4: 83 e8 2e        jle    2e <while_loop+0x2e>
5: 90              nop
6: 90              nop
7: 90              nop
8: 90              nop
9: 90              nop
10: 90              nop
11: 90              nop
12: 90              nop
13: 90              nop
14: 90              nop
15: 90              nop
16: 90              nop
17: 90              nop
18: 90              nop
19: 90              nop
20: 90              nop
21: 90              nop
22: 90              nop
23: 90              nop
24: 90              nop
25: 90              nop
26: 90              nop
27: 90              nop
28: 90              nop
29: 90              nop
30: 90              nop
31: 90              nop
32: 90              nop
33: 90              nop
34: 90              nop
35: 90              nop
36: 90              nop
37: 90              nop
38: 90              nop
39: 90              nop
40: 90              nop
41: 90              nop
42: 90              nop
43: 90              nop
44: 90              nop
45: 90              nop
46: 90              nop
47: 90              nop
48: 90              nop
49: 90              nop
50: 90              nop
51: 90              nop
52: 90              nop
53: 90              nop
54: 90              nop
55: 90              nop
56: 90              nop
57: 90              nop
58: 90              nop
59: 90              nop
60: 90              nop
61: 90              nop
62: 90              nop
63: 90              nop
64: 90              nop
65: 90              nop
66: 90              nop
67: 90              nop
68: 90              nop
69: 90              nop
70: 90              nop
71: 90              nop
72: 90              nop
73: 90              nop
74: 90              nop
75: 90              nop
76: 90              nop
77: 90              nop
78: 90              nop
79: 90              nop
80: 90              nop
81: 90              nop
82: 90              nop
83: 90              nop
84: 90              nop
85: 90              nop
86: 90              nop
87: 90              nop
88: 90              nop
89: 90              nop
90: 90              nop
91: 90              nop
92: 90              nop
93: 90              nop
94: 90              nop
95: 90              nop
96: 90              nop
97: 90              nop
98: 90              nop
99: 90              nop
100: 90              nop
101: 90              nop
102: 90              nop
103: 90              nop
104: 90              nop
105: 90              nop
106: 90              nop
107: 90              nop
108: 90              nop
109: 90              nop
110: 90              nop
111: 90              nop
112: 90              nop
113: 90              nop
114: 90              nop
115: 90              nop
116: 90              nop
117: 90              nop
118: 90              nop
119: 90              nop
120: 90              nop
121: 90              nop
122: 90              nop
123: 90              nop
124: 90              nop
125: 90              nop
126: 90              nop
127: 90              nop
128: 90              nop
129: 90              nop
130: 90              nop
131: 90              nop
132: 90              nop
133: 90              nop
134: 90              nop
135: 90              nop
136: 90              nop
137: 90              nop
138: 90              nop
139: 90              nop
140: 90              nop
141: 90              nop
142: 90              nop
143: 90              nop
144: 90              nop
145: 90              nop
146: 90              nop
147: 90              nop
148: 90              nop
149: 90              nop
150: 90              nop
151: 90              nop
152: 90              nop
153: 90              nop
154: 90              nop
155: 90              nop
156: 90              nop
157: 90              nop
158: 90              nop
159: 90              nop
160: 90              nop
161: 90              nop
162: 90              nop
163: 90              nop
164: 90              nop
165: 90              nop
166: 90              nop
167: 90              nop
168: 90              nop
169: 90              nop
170: 90              nop
171: 90              nop
172: 90              nop
173: 90              nop
174: 90              nop
175: 90              nop
176: 90              nop
177: 90              nop
178: 90              nop
179: 90              nop
180: 90              nop
181: 90              nop
182: 90              nop
183: 90              nop
184: 90              nop
185: 90              nop
186: 90              nop
187: 90              nop
188: 90              nop
189: 90              nop
190: 90              nop
191: 90              nop
192: 90              nop
193: 90              nop
194: 90              nop
195: 90              nop
196: 90              nop
197: 90              nop
198: 90              nop
199: 90              nop
200: 90              nop
201: 90              nop
202: 90              nop
203: 90              nop
204: 90              nop
205: 90              nop
206: 90              nop
207: 90              nop
208: 90              nop
209: 90              nop
210: 90              nop
211: 90              nop
212: 90              nop
213: 90              nop
214: 90              nop
215: 90              nop
216: 90              nop
217: 90              nop
218: 90              nop
219: 90              nop
220: 90              nop
221: 90              nop
222: 90              nop
223: 90              nop
224: 90              nop
225: 90              nop
226: 90              nop
227: 90              nop
228: 90              nop
229: 90              nop
230: 90              nop
231: 90              nop
232: 90              nop
233: 90              nop
234: 90              nop
235: 90              nop
236: 90              nop
237: 90              nop
238: 90              nop
239: 90              nop
240: 90              nop
241: 90              nop
242: 90              nop
243: 90              nop
244: 90              nop
245: 90              nop
246: 90              nop
247: 90              nop
248: 90              nop
249: 90              nop
250: 90              nop
251: 90              nop
252: 90              nop
253: 90              nop
254: 90              nop
255: 90              nop
256: 90              nop
257: 90              nop
258: 90              nop
259: 90              nop
260: 90              nop
261: 90              nop
262: 90              nop
263: 90              nop
264: 90              nop
265: 90              nop
266: 90              nop
267: 90              nop
268: 90              nop
269: 90              nop
270: 90              nop
271: 90              nop
272: 90              nop
273: 90              nop
274: 90              nop
275: 90              nop
276: 90              nop
277: 90              nop
278: 90              nop
279: 90              nop
280: 90              nop
281: 90              nop
282: 90              nop
283: 90              nop
284: 90              nop
285: 90              nop
286: 90              nop
287: 90              nop
288: 90              nop
289: 90              nop
290: 90              nop
291: 90              nop
292: 90              nop
293: 90              nop
294: 90              nop
295: 90              nop
296: 90              nop
297: 90              nop
298: 90              nop
299: 90              nop
300: 90              nop
301: 90              nop
302: 90              nop
303: 90              nop
304: 90              nop
305: 90              nop
306: 90              nop
307: 90              nop
308: 90              nop
309: 90              nop
310: 90              nop
311: 90              nop
312: 90              nop
313: 90              nop
314: 90              nop
315: 90              nop
316: 90              nop
317: 90              nop
318: 90              nop
319: 90              nop
320: 90              nop
321: 90              nop
322: 90              nop
323: 90              nop
324: 90              nop
325: 90              nop
326: 90              nop
327: 90              nop
328: 90              nop
329: 90              nop
330: 90              nop
331: 90              nop
332: 90              nop
333: 90              nop
334: 90              nop
335: 90              nop
336: 90              nop
337: 90              nop
338: 90              nop
339: 90              nop
340: 90              nop
341: 90              nop
342: 90              nop
343: 90              nop
344: 90              nop
345: 90              nop
346: 90              nop
347: 90              nop
348: 90              nop
349: 90              nop
350: 90              nop
351: 90              nop
352: 90              nop
353: 90              nop
354: 90              nop
355: 90              nop
356: 90              nop
357: 90              nop
358: 90              nop
359: 90              nop
360: 90              nop
361: 90              nop
362: 90              nop
363: 90              nop
364: 90              nop
365: 90              nop
366: 90              nop
367: 90              nop
368: 90              nop
369: 90              nop
370: 90              nop
371: 90              nop
372: 90              nop
373: 90              nop
374: 90              nop
375: 90              nop
376: 90              nop
377: 90              nop
378: 90              nop
379: 90              nop
380: 90              nop
381: 90              nop
382: 90              nop
383: 90              nop
384: 90              nop
385: 90              nop
386: 90              nop
387: 90              nop
388: 90              nop
389: 90              nop
390: 90              nop
391: 90              nop
392: 90              nop
393: 90              nop
394: 90              nop
395: 90              nop
396: 90              nop
397: 90              nop
398: 90              nop
399: 90              nop
400: 90              nop
401: 90              nop
402: 90              nop
403: 90              nop
404: 90              nop
405: 90              nop
406: 90              nop
407: 90              nop
408: 90              nop
409: 90              nop
410: 90              nop
411: 90              nop
412: 90              nop
413: 90              nop
414: 90              nop
415: 90              nop
416: 90              nop
417: 90              nop
418: 90              nop
419: 90              nop
420: 90              nop
421: 90              nop
422: 90              nop
423: 90              nop
424: 90              nop
425: 90              nop
426: 90              nop
427: 90              nop
428: 90              nop
429: 90              nop
430: 90              nop
431: 90              nop
432: 90              nop
433: 90              nop
434: 90              nop
435: 90              nop
436: 90              nop
437: 90              nop
438: 90              nop
439: 90              nop
440: 90              nop
441: 90              nop
442: 90              nop
443: 90              nop
444: 90              nop
445: 90              nop
446: 90              nop
447: 90              nop
448: 90              nop
449: 90              nop
450: 90              nop
451: 90              nop
452: 90              nop
453: 90              nop
454: 90              nop
455: 90              nop
456: 90              nop
457: 90              nop
458: 90              nop
459: 90              nop
460: 90              nop
461: 90              nop
462: 90              nop
463: 90              nop
464: 90              nop
465: 90              nop
466: 90              nop
467: 90              nop
468: 90              nop
469: 90              nop
470: 90              nop
471: 90              nop
472: 90              nop
473: 90              nop
474: 90              nop
475: 90              nop
476: 90              nop
477: 90              nop
478: 90              nop
479: 90              nop
480: 90              nop
481: 90              nop
482: 90              nop
483: 90              nop
484: 90              nop
485: 90              nop
486: 90              nop
487: 90              nop
488: 90              nop
489: 90              nop
490: 90              nop
491: 90              nop
492: 90              nop
493: 90              nop
494: 90              nop
495: 90              nop
496: 90              nop
497: 90              nop
498: 90              nop
499: 90              nop
500: 90              nop
501: 90              nop
502: 90              nop
503: 90              nop
504: 90              nop
505: 90              nop
506: 90              nop
507: 90              nop
508: 90              nop
509: 90              nop
510: 90              nop
511: 90              nop
512: 90              nop
513: 90              nop
514: 90              nop
515: 90              nop
516: 90              nop
517: 90              nop
518: 90              nop
519: 90              nop
520: 90              nop
521: 90              nop
522: 90              nop
523: 90              nop
524: 90              nop
525: 90              nop
526: 90              nop
527: 90              nop
528: 90              nop
529: 90              nop
530: 90              nop
531: 90              nop
532: 90              nop
533: 90              nop
534: 90              nop
535: 90              nop
536: 90              nop
537: 90              nop
538: 90              nop
539: 90              nop
540: 90              nop
541: 90              nop
542: 90              nop
543: 90              nop
544: 90              nop
545: 90              nop
546: 90              nop
547: 90              nop
548: 90              nop
549: 90              nop
550: 90              nop
551: 90              nop
552: 90              nop
553: 90              nop
554: 90              nop
555: 90              nop
556: 90              nop
557: 90              nop
558: 90              nop
559: 90              nop
560: 90              nop
561: 90              nop
562: 90              nop
563: 90              nop
564: 90              nop
565: 90              nop
566: 90              nop
567: 90              nop
568: 90              nop
569: 90              nop
570: 90              nop
571: 90              nop
572: 90              nop
573: 90              nop
574: 90              nop
575: 90              nop
576: 90              nop
577: 90              nop
578: 90              nop
579: 90              nop
580: 90              nop
581: 90              nop
582: 90              nop
583: 90              nop
584: 90              nop
585: 90              nop
586: 90              nop
587: 90              nop
588: 90              nop
589: 90              nop
590: 90              nop
591: 90              nop
592: 90              nop
593: 90              nop
594: 90              nop
595: 90              nop
596: 90              nop
597: 90              nop
598: 90              nop
599: 90              nop
600: 90              nop
601: 90              nop
602: 90              nop
603: 90              nop
604: 90              nop
605: 90              nop
606: 90              nop
607: 90              nop
608: 90              nop
609: 90              nop
610: 90              nop
611: 90              nop
612: 90              nop
613: 90              nop
614: 90              nop
615: 90              nop
616: 90              nop
617: 90              nop
618: 90              nop
619: 90              nop
620: 90              nop
621: 90              nop
622: 90              nop
623: 90              nop
624: 90              nop
625: 90              nop
626: 90              nop
627: 90              nop
628: 90              nop
629: 90              nop
630: 90              nop
631: 90              nop
632: 90              nop
633: 90              nop
634: 90              nop
635: 90              nop
636: 90              nop
637: 90              nop
638: 90              nop
639: 90              nop
640: 90              nop
641: 90              nop
642: 90              nop
643: 90              nop
644: 90              nop
645: 90              nop
646: 90              nop
647: 90              nop
648: 90              nop
649: 90              nop
650: 90              nop
651: 90              nop
652: 90              nop
653: 90              nop
654: 90              nop
655: 90              nop
656: 90              nop
657: 90              nop
658: 90              nop
659: 90              nop
660: 90              nop
661: 90              nop
662: 90              nop
663: 90              nop
664: 90              nop
665: 90              nop
666: 90              nop
667: 90              nop
668: 90              nop
669: 90              nop
670: 90              nop
671: 90              nop
672: 90              nop
673: 90              nop
674: 90              nop
675: 90              nop
676: 90              nop
677: 90              nop
678: 90              nop
679: 90              nop
680: 90              nop
681: 90              nop
682: 90              nop
683: 90              nop
684: 90              nop
685: 90              nop
686: 90              nop
687: 90              nop
688: 90              nop
689: 90              nop
690: 90              nop
691: 90              nop
692: 90              nop
693: 90              nop
694: 90              nop
695: 90              nop
696: 90              nop
697: 90              nop
698: 90              nop
699: 90              nop
700: 90              nop
701: 90              nop
702: 90              nop
703: 90              nop
704: 90              nop
705: 90              nop
706: 90              nop
707: 90              nop
708: 90              nop
709: 90              nop
710: 90              nop
711: 90              nop
712: 90              nop
713: 90              nop
714: 90              nop
715: 90              nop
716: 90              nop
717: 90              nop
718: 90              nop
719: 90              nop
720: 90              nop
721: 90              nop
722: 90              nop
723: 90              nop
724: 90              nop
725: 90              nop
726: 90              nop
727: 90              nop
728: 90              nop
729: 90              nop
730: 90              nop
731: 90              nop
732: 90              nop
733: 90              nop
734: 90              nop
735: 90              nop
736: 90              nop
737: 90              nop
738: 90              nop
739: 90              nop
740: 90              nop
741: 90              nop
742: 90              nop
743: 90              nop
744: 90              nop
745: 90              nop
746: 90              nop
747: 90              nop
748: 90              nop
749: 90              nop
750: 90              nop
751: 90              nop
752: 90              nop
753: 90              nop
754: 90              nop
755: 90              nop
756: 90              nop
757: 90              nop
758: 90              nop
759: 90              nop
760: 90              nop
761: 90              nop
762: 90              nop
763: 90              nop
764: 90              nop
765: 90              nop
766: 90              nop
767: 90              nop
768: 90              nop
769: 90              nop
770: 90              nop
771: 90              nop
772: 90              nop
773: 90              nop
774: 90              nop
775: 90              nop
776: 90              nop
777: 90              nop
778: 90              nop
779: 90              nop
780: 90              nop
781: 90              nop
782: 90              nop
783: 90              nop
784: 90              nop
785: 90              nop
786: 90              nop
787: 90              nop
788: 90              nop
789: 90              nop
790: 90              nop
791: 90              nop
792: 90              nop
793: 90              nop
794: 90              nop
795: 90              nop
796: 90              nop
797: 90              nop
798: 90              nop
799: 90              nop
800: 90              nop
801: 90              nop
802: 90              nop
803: 90              nop
804: 90              nop
805: 90              nop
806: 90              nop
807: 90              nop
808: 90              nop
809: 90              nop
810: 90              nop
811: 90              nop
812: 90              nop
813: 90              nop
814: 90              nop
815: 90              nop
816: 90              nop
817: 90              nop
818: 90              nop
819: 90              nop
820: 90              nop
821: 90              nop
822: 90              nop
823: 90              nop
824: 90              nop
825: 90              nop
826: 90              nop
827: 90              nop
828: 90              nop
829: 90              nop
830: 90              nop
831: 90              nop
832: 90              nop
833: 90              nop
834: 90              nop
835: 90              nop
836: 90              nop
837: 90              nop
838: 90              nop
839: 90              nop
840: 90              nop
841: 90              nop
842: 90              nop
843: 90              nop
844: 90              nop
845: 90              nop
846: 90              nop
847: 90              nop
848: 90              nop
849: 90              nop
850: 90              nop
851: 90              nop
852: 90              nop
853: 90              nop
854: 90              nop
855: 90              nop
856: 90              nop
857: 90              nop
858: 90              nop
859: 90              nop
860: 90              nop
861: 90              nop
862: 90              nop
863: 90              nop
864: 90              nop
865: 90              nop
866: 90              nop
867: 90              nop
868: 90              nop
869: 90              nop
870: 90              nop
871: 90              nop
872: 90              nop
873: 90              nop
874: 90              nop
875: 90              nop
876: 90              nop
877: 90              nop
878: 90              nop
879: 90              nop
880: 90              nop
881: 90              nop
882: 90              nop
883: 90              nop
884: 90              nop
885: 90              nop
886: 90              nop
887: 90              nop
888: 90              nop
889: 90              nop
890: 90              nop
891: 90              nop
892: 90              nop
893: 90              nop
894: 90              nop
895: 90              nop
896: 90              nop
897: 90              nop
898: 90              nop
899: 90              nop
900: 90              nop
901: 90              nop
902: 90              nop
903: 90              nop
904: 90              nop
905: 90              nop
906: 90              nop
907: 90              nop
908: 90              nop
909: 90              nop
910: 90              nop
911: 90              nop
912: 90              nop
913: 90              nop
914: 90              nop
915: 90              nop
916: 90              nop
917: 90              nop
918: 90              nop
919: 90              nop
920: 90              nop
921: 90              nop
922: 90              nop
923: 90              nop
924: 90              nop
925: 90              nop
926: 90              nop
927: 90              nop
928: 90              nop
929: 90              nop
930: 90              nop
931: 90              nop
932: 90              nop
933: 90              nop
934: 90              nop
935: 90              nop
936: 90              nop
937: 90              nop
938: 90              nop
939: 90              nop
940: 90              nop
941: 90              nop
942: 90              nop
943: 90              nop
944: 90              nop
945: 90              nop
946: 90              nop
947: 90              nop
948: 90              nop
949: 90              nop
950: 90              nop
951: 90              nop
952: 90              nop
953: 90              nop
954: 90              nop
955: 90              nop
956: 90              nop
957: 90              nop
958: 90              nop
959: 90              nop
960: 90              nop
961: 90              nop
962: 90              nop
963: 90              nop
964: 90              nop
965: 90              nop
966: 90              nop
967: 90              nop
968: 90              nop
969: 90              nop
970: 90              nop
971: 90              nop
972: 90              nop
973: 90              nop
974: 90              nop
975: 90              nop
976: 90              nop
977: 90              nop
978: 90              nop
979: 90              nop
980: 90              nop
981: 90              nop
982: 90              nop
983: 90              nop
984: 90              nop
985: 90              nop
986: 90              nop
987: 90              nop
988: 90              nop
989: 90              nop
990: 90              nop
991: 90              nop
992: 90              nop
993: 90              nop
994: 90              nop
995: 90              nop
996: 90              nop
997: 90              nop
998: 90              nop
999: 90              nop
1000: 90              nop

```

Há aqui várias pistas da batota feita, já que este pedaço de código *assembly* não é uma visão de um código executável desmontado (senão os endereços não começavam em zero, para além de outras pistas) e este código não foi compilado com a opção de otimização -O2. E isto é fácil de ver, porque está a fazer a comparação com o argumento em memória (na *stack*) em vez de ter copiado primeiro os argumentos para registo!

### Exemplo 3:

```
804832c: add %edx,%ebx # x=m (BP1)
804832e: imul %edx,%eax # y=m (BP2)
8048331: dec %edx      # m- (BP3)
```

As variáveis *x*, *y* e *m* são os argumentos recebidos pela função e o compilador alocou para cada argumento um registo diferente. O que se pretendia saber no preenchimento da tabela era quais seriam os conteúdos (numéricos) que seriam colocados nesses registo pela 1<sup>a</sup> vez, a que designámos por “atribuição inicial”. Ou seja, quando seriam colocados nos registo os valores dos argumentos recebidos, que valores eram esses?

E para se ter a certeza dos valores que foram de facto lá colocados, temos de procurar no código quando se vai buscar à *stack* os valores dos argumentos (que é assim que funciona no IA-32), de modo a se inserir pontos de paragem na instrução seguinte. Assim, quando fossemos executar o código dentro do *debugger*, a execução do programa seria interrompida em cada *breakpoint* e nós poderíamos ir analisar os conteúdos dos registo.

A sua resolução vai colocar o 1º *breakpoint* antes da instrução que vai alterar o valor de *x* (portanto não seria a sua 1<sup>a</sup> atribuição...) para além de que se colocar o *breakpoint* nesse endereço, a instrução que iria alterar o valor de *x* (uma adição) não seria executada antes de parar no *breakpoint*.

### A5. A *stack frame* da função

Este exercício é um dos 2 que aparece sempre em provas de avaliação (o outro é sobre anotação de código assembly) e, para além de vários alunos não terem sequer tentado resolver, vários outros apresentam ainda resultados incompreensíveis, o que mostra uma falta de estudo e concentração na resolução deste TPC.

E mesmo tendo acesso integral aos slides e vídeos das aulas teóricas de exposição deste tema!

### Exemplo 1:

Endereço 1ª célula	Conteúdo em hex	Conteúdo comentado
		push %ebp guardando %ebp
		mov %esp,%ebp
		push %ebp salvaguardando o antigo
		mov 0x10(%ebp),%edx (Valor de
		mov 0xc(%ebp),%eax (valor de y)
<i>Endereço de regresso</i>		
0x11(%ebp)	0x04	mov valor de %eax
(0xe(%ebp))	0x02	mov valor de %edx
0x10(%ebp)	0x03	
	0x01	antigo%eax

Vejam só a longa lista de disparates nesta resolução:

- Nas primeiras 5 caixas de 4 células cada: o que está lá escrito não é certamente o "Conteúdo em hex" conforme está indicado em cima;
- O comentário de cada uma destas 5 caixas não bate certo com que o que foi escrito no conteúdo das mesmas;
- O endereço de regresso é um nº que não faz qualquer sentido;
- O conteúdo das últimas células já é numérico, mas também não bate certo com o comentário;
- O "Endereço da 1ª célula" de cada caixa deveria também ser um valor numérico!

### Exemplo 2:

<code>0x00000000</code>	Variável m
<code>0x00000000</code>	Variável y
<code>0x00000000</code>	Variável x
<code>0x08048325</code>	Endereço de regresso (antigo %ebp)
<code>0xbfff0394</code>	ebx
<code>0xbfff0398</code>	ebp } valores que me aparecem quando uso "info registers", como "Save registers".
<code>0xbfff039e</code>	esp

Outra lista de disparates, a começar com "Endereço de regresso (antigo %ebp)". E os valores que aparecem no "info reg" é naquele sítio da stack que vão ficar? Porquê?

### Exemplo 3:

?		Como olhar 2º breakpoint? Como fazer breakpoint?
nt ←	C3	Que é preciso colocar?
bx ←	C9 ??	
	08048359	→ Endereço de regresso → endereço a seguir de call
valor de x ←	6a 04	
valor de y ←	6a 02	
valor de m ←	6a 03	
	50	→ push %eax (guarda a var)
epoca para a var	83 e4 f0	→ and \$0xffffffff,%esp
	83 ec 08	→ sub \$0x8,%esp (subtraindo 8 de esp)
	89 e5	→ mov %esp,%ebp (valor esp no end esp)
	55	→ push %ebp ↳ base point

Esta resolução tem tanto disparate que não tem por onde se pegue. É uma falta de estudo gritante. Se uma resolução como esta aparece numa prova de avaliação e o resultado final está na zona de fronteira entre o passar e o reprovar, só este exercício é o suficiente para influenciar negativamente o docente...

**Exemplo 4:**

[ 08 04 83 44 ]	-> Fim de main
[ 08 04 83 59 ]	-> retorno de função
[ 00 00 00 04 ]	-> 1º argumento
[ 00 00 00 02 ]	-> 2º argumento

Vê-se nesta resolução que o aluno não percebeu ainda como é a estrutura típica do quadro duma função (na *stack*) no IA-32. Entendeu o conceito de “Endereço de regresso” e traduziu para “retorno de função” (está correto), mas já não entendeu o que era o *frame pointer* da função chamadora *main* (que deveria estar imediatamente logo a seguir na *stack*) e colocou lá em substituição o endereço do início do código da *main*!...

**A6. Explicação mais detalhada do código em assembly do ciclo while**

Nova análise do código *assembly* da função *while\_loop*, anotado (a **bold**) e comentado:

```

pushl %ebp          ; salvaguarda valor do %ebp
                      %ebp aponta nesta altura para quadro da função chamadora na stack, mas é
                      necessário definir um novo apontador para o quadro na stack (frame pointer) da
                      função chamada (while_loop); guarda-se na stack o valor do frame pointer
                      que se encontra no registo %ebp, para o %ebp poder alojar o valor do frame
                      pointer da função chamada (while_loop).

movl %esp, %ebp      ; cria novo frame pointer usando o endereço em %esp
movl 16(%ebp), %edx ; copia n (o 3º arg de while_loop) da stack p/ %edx
                      os argumentos da função while_loop foram colocados na stack da função
                      chamadora antes da execução desta função; assumindo argumentos de 4 bytes de
                      tamanho em IA-32, o primeiro encontra-se sempre à distância de 8 células do
                      endereço no frame pointer (em %ebp) da função chamada (while_loop)
                      durante a sua execução; o 2º argumento encontra-se a 12 células de %ebp, o 3º a
                      16, e assim por diante.

testl %edx, %edx    ; testa n (faz & lógico de %edx com %edx)
                      esta instrução implementa um & lógico entre os operandos, ativando as flags ZF,
                      CF, OF, e SF de acordo com o resultado; quando os 2 operandos são iguais,
                      tudo se passa como se o teste fosse sobre um dos operandos, já que o & lógico
                      de um valor consigo mesmo dá sempre o próprio valor; as instruções de salto
                      consultam estas flags para a tomada de decisão; portanto esta instrução vai então
                      testar se n é zero, positivo ou negativo.

pushl %ebx          ; salvaguarda conteúdo do %ebx
                      garantir que os conteúdos dos registos %eax, %ecx, e %edx não são
                      alterados quando se chama uma função é uma responsabilidade da função
                      chamadora; se contiverem informação relevante para essa função ela terá de
                      salvaguardar os seus conteúdos (na stack) antes de chamar a função
                      while_loop de modo a que esta os possa usar, e depois terá de repor esses
                      valores; a responsabilidade de salvaguardar os conteúdos dos registos %ebx,
                      %edi, e %esi é da função chamada, caso ela necessite de usar esses

```

registos (e neste caso a função necessita de usar 4 registos, os 3 que não tem de salvaguardar e mais este).

```

movl 12(%ebp), %eax ; copia y (o 2º arg de while_loop) da stack p/ %eax
                      %eax toma o valor de y que foi passado como argumento.

movl 8(%ebp), %ebx ; copia x (o 1º arg de while_loop) da stack p/ %ebx
                      a salvaguarda do conteúdo deste registo foi feita anteriormente para agora poder ser usado; %ebx toma o valor de x que foi passado como argumento.

jle .L3              ; salta se n menor ou igual que zero para depois do ciclo
                      a instrução consulta as flags previamente ativadas pela instrução testl; (de notar que as instruções de transferência de dados não alteram as flags, tais como os movl e pushl que estão entre o testl e este jle); neste caso, essa instrução em conjunto com esta de salto condicional estão a avaliar se o conteúdo de %edx é menor ou igual a zero; caso n seja menor ou igual a zero o controlo do programa salta para o fim de while_loop, em .L3 (a primeira das duas condições é  $n > 0$ ), não executando nenhuma iteração do corpo do ciclo.

movl %edx, %ecx    ; coloca o valor de n numa variável temporária
                      o valor de n é copiado para esta nova variável temp (em %ecx) de modo a ficar depois com  $16 * n$ .

sall $4, %ecx       ; desloca 4 bits p/ a esquerda (shift left) no valor binário de temp, equivalente a calcular  $n * 2^4$ 
                      um deslocamento de 2 bits para a esquerda no nº binário  $111_2$  resulta em  $11100_2$ ; é equivalente a uma multiplicação de  $2^2$  pelo número em decimal; neste caso, está a ser feita uma multiplicação de  $2^4$  (16) pelo conteúdo de %ecx (que tem ainda o valor de n) que depois será útil para a avaliação de  $y < 16 * n$ ; poder-se-ia usar imull $16,%ecx, mas a instrução sall ocupa menos bytes em memória e shifts são mais rápidos de executar que multiplicações.

cmp l %ecx, %eax   ; compara y com  $16 * n$  fazendo  $y - 16 * n$  e ativando as flags:
                      as flags serão alteradas consoante o resultado da diferença é zero, negativo ou positivo.

jge .L3              ; salta para fora do ciclo se o resultado da operação anterior for maior ou igual que zero
                      esta instrução, em conjunto com o cmp l anterior, indica um salto para o fim do ciclo (em L3) caso y (em %eax) seja maior ou igual que temp (em %ecx, com  $16 * n$ ); caso  $y \geq 16 * n$  o controlo salta para L3, que se encontra no fim de while_loop, sem executar qualquer iteração do ciclo.

.p2align 2,,3        ; etiqueta vista na resolução do TPC anterior.

.L6:                ; etiqueta que indica o início do ciclo while
                      etiqueta usada como referência para saltos condicionais; indica o início do corpo do ciclo while do código C.

addl %edx, %ebx     ; calcula  $x = n$ 
                      é feita a soma de x (em %ebx) com n (em %edx) guardando o resultado em x; é equivalente à primeira instrução do corpo do ciclo  $x += n$ ;

imull %edx, %eax    ; calcula  $y = n$ 
                      multiplicação de n (em %edx) com y (em %eax) guardando o resultado em y; é equivalente à expressão  $y *= n$  no código C.

decl %edx           ; calcula  $n --$ 
                      decremento de n (em %edx), equivalente a  $n --$ .

```

```

subl    $16, %ecx      ; calcula o novo temp=16*(n-1) trocando a multiplicação pela subtração temp=16*n-16
esta instrução evita o uso da multiplicação novamente, já que o n foi decrementado e é necessário recalcular  $16 \cdot n$ ; assim, apenas se subtrai 16 a  $16 \cdot n$ , que é equivalente a subtrair uma unidade a n e voltar a fazer  $16 \cdot n$  com o novo n, quer usando um shift ou uma multiplicação; é mais rápido fazer uma subtração do que uma dessas duas instruções.

testl   %edx, %edx    ; testa n (faz & lógico de %edx com %edx)
preparação para a avaliação de  $n > 0$ , semelhante ao processo feito nas instruções antes do ciclo; no entanto, essa avaliação inicial foi feita para determinar se é possível entrar dentro do corpo do ciclo while; agora, esta avaliação verifica se é possível continuar dentro do ciclo na próxima iteração, sendo esta instrução repetida a cada iteração; poder-se-ia evitar a repetição destas avaliações com alguma reorganização do código, no entanto este é o padrão normalmente usado por esta versão do compilador da GNU.

jle     .L3            ; salta se n menor ou igual que zero p/ fora do ciclo
salto para fora do ciclo (em L3) caso n (em %edx) seja menor ou igual que zero, falhando a avaliação de  $n > 0$  do ciclo while.

cmpl   %ecx, %eax     ; compara y com  $16 \cdot n$  fazendo  $y-16 \cdot n$  e ativando as flags
comparação de y com  $16 \cdot n$ , já contemplando o facto de n ter sido decrementado; no entanto, a avaliação inicial idêntica a esta foi feita para determinar se era possível entrar dentro do corpo do ciclo while; agora, esta avaliação verifica se é possível continuar dentro do ciclo na próxima iteração (em conjunto com as instruções que avaliam se  $n > 0$ ), sendo esta instrução repetida a cada iteração; poder-se-ia evitar a repetição destas avaliações com alguma reorganização do código, no entanto este é o padrão normalmente usado por esta versão do compilador da GNU.

j1     .L6             ; salta para o início do ciclo se o resultado da operação anterior for menor que zero
salto para o início do ciclo (em L6) caso a condição  $y < 16 \cdot n$  seja válida; caso esta condição não se verifique não é possível continuar dentro do corpo do ciclo, sendo a instrução seguinte a próxima a ser executada.

.L3:          ; etiqueta que indica o início do bloco de código após o ciclo while
corresponde ao endereço da primeira instrução após o ciclo while; na função while_loop corresponde à preparação para devolver o valor x (return x no código C), mas que envolve uma série de passos antes de finalizar a execução desta função.

movl   %ebx, %eax     ; coloca x (o valor a devolver) no registo %eax
por convenção no IA-32, os valores devolvidos por funções são sempre colocados no registo %eax antes do seu término; poder-se-ia ter alocado logo o registo %eax à variável x, que é a que temos de devolver (return x no código C), eliminando esta instrução final.

popl   %ebx           ; recupera o valor original de %ebx
como foi feita a salvaguarda do conteúdo deste registo na fase de arranque da função agora é necessário recuperar esse valor, de modo à função chamadora manter o conteúdo que tinha inicialmente atribuído a este registo, antes de chamar while_loop.

```

---

```
leave ; recupera o frame pointer da função chamador
      agora que a função while_loop acabou a sua execução é necessário
      recuperar a frame pointer da função chamadora, que foi salvaguardada na stack
      logo na 1ª instrução desta função; basta garantir que o %esp está a apontar para
      o mesmo sítio que %ebp e fazer um popl %ebp, que é o processo
      implementado pela instrução leave.

ret ; regressa à função chamadora
     é necessário que a próxima instrução a ser executada (cuja localização na
     memória se encontra sempre em %eip) seja a instrução na função chamadora
     imediatamente a seguir à instrução call que chamou a função while_loop;
     o endereço dessa instrução foi colocado na stack na execução da instrução call
     while_loop (encontra-se neste momento no topo da pilha), e tem agora que
     ser colocado no %eip; esta instrução seria equivalente a um popl %eip.
```