Nº	Nome:	Turma:

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.

```
while_loop:
    pushl %ebp              //Coloca o base pointer na stack (salvaguarda)
    movl %esp,%ebp          //Atualiza frame pointer
    movl 16(%ebp),%edx      //edx = n
    testl %edx,%edx         //edx & edx (Atualiza flags)
    pushl %ebx              //coloca n na stack (salvaguarda)
    movl 12(%ebp),%eax      //eax = y
    movl 8(%ebp),%ebx       //ebx = x
    jle .L3                 //salta se menor ou igual para L3
    movl %edx,%ecx          //ecx = edx = n
    sall $4,%ecx            //ecx = 16*n
    cmpl %ecx,%eax          //compara 16*n e y
    jge .L3                 //salta para L3 se for maior ou igual
    .p2align 2,,3           //?
.L6:
    addl %edx,%ebx          //x += n
    imull %edx,%eax         //y *= n
    decl %edx               //n--;
    subl $16,%ecx           //ecx = ecx - 16
    testl %edx,%edx         //edx & edx?
    jle .L3                 //salta para fora do ciclo se n<=0
    cmpl %ecx,%eax          //compara ecx e eax
    jl .L6                  //salta de novo para o ciclo se y<16*n
.L3 nas outras folhas
```

Variável	Registo	Atribuição inicial
x	%ebx	1º argumento
y	%eax	2º argumento
n	%edx	3º argumento
ecx	%ecx	igual a n

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

```
int main() {
    while_loop(2,4,6);
    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.

```
                8048313: 8b 55 10    mov 0x10(%ebp),%edx  //edx=n
break ---->     8048316:             test %edx,%edx
                                     push %ebx
                8048319: 8b 45 0c    mov 0xc(%ebp),%eax   //eax=y
                804831c: 8b 5d 08    mov 0x8(%ebp),%ebx   //ebx=x
break ---->     804831f:
                8048321: 89 d1       mov %edx,%ecx        //ecx=edx=n
break ---->     8048323:
```

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

```
0x8048316, 0x804831f, 0x8048323,
breakpoint: 1          2          3
```

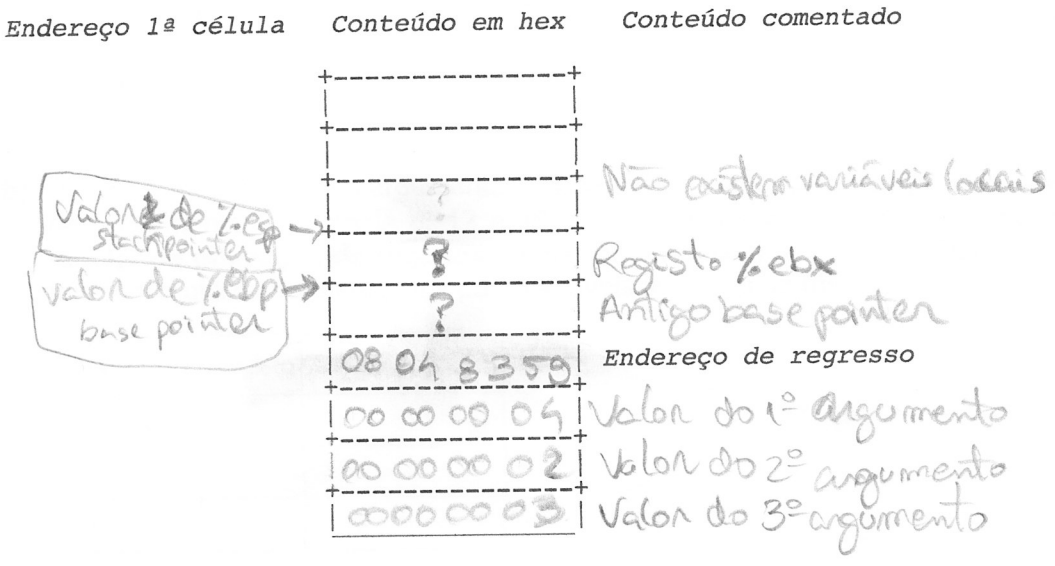
Antes de executar qualquer código, coloque aqui a sua estimativa do que irá estar nos registos após cada ponto de paragem:

Variável	Registo	Break1	Break2	Break3	Break_	Break_
x	%ebx		4	4		
y	%eax		2	2		
n	%edx	3	3	3		
ecx	%ecx			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	Registo	Break1	Break2	Break3	Break_	Break_
x	%ebx	...	4	4		
y	%eax	1	2	2		
n	%edx	3	3	3		
ecx	%ecx	...	...	3		

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.



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

① continuação:

Término:

```
.LB:
    movl %ebx, %eax   // devolve o resultado da função: x
    popl %ebx         // Liberta ebx (x)
    leave             // Recupera o stack pointer e base pointer
    ret               // regressa à função chamadora
```