Nº A92846	Nome: Carlos Miguel Passos Ferreira	Turma: Quinta de tarde (MIEFIS)

Resolução dos exercícios (deve ser redigido manualmente)

Sugestões para esta resolução em baixo.

1. Um possível algoritmo

Escreva um possível algoritmo da função numa linguagem de alto nível (C, por ex.)

```
int soma_grandes (int n , int *a)
{ int i ;
  int result ;
  result = 0 ;
for (i=0 ; i<n ; i++ )
  { if ( a[i] > 1000)
      { result += a [i] ;}
  }
return result; }
```

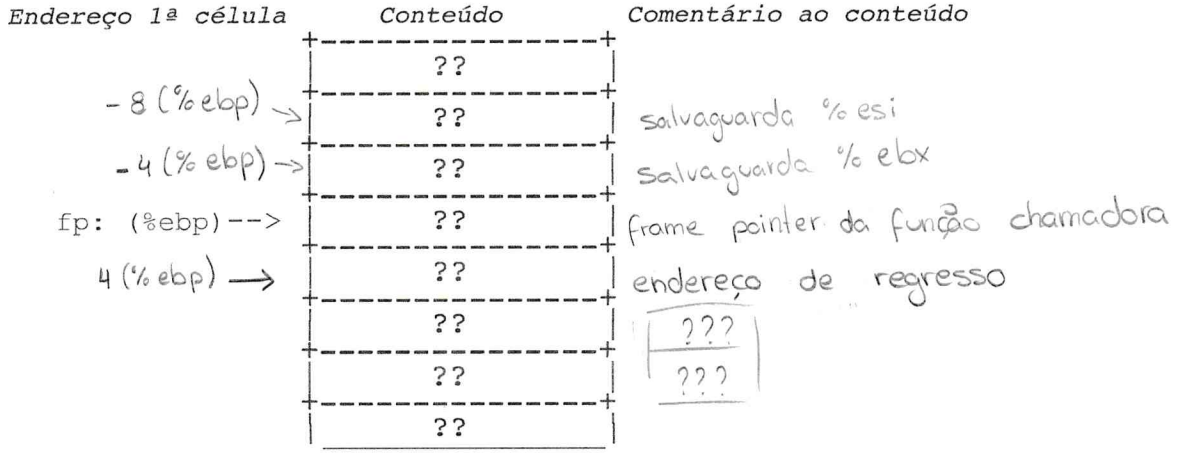
} guardado em soma_grande.c

2. Reconstrução do ficheiro `soma_grandes.s` devidamente anotado

2.a) Preencha a tabela de alocação de registos a argumentos, variáveis locais e eventuais variáveis de caráter mais temporário, que julgue serem necessários armazenar (com comentários)

Utilização dos Registos		
Variável	**Registo**	**Comentários**
n	% esi	"calle saved" pk são argumentos
* a	% ebx	"calle saved" pk são argumentos
i	% ecx	registo escolhido
a [i]	% edx	registo escolhido
return	% eax	valor retornado (com a soma dos valores)

2.b) Represente o quadro de ativação da função, colocando lá também o *stack pointer*.
Nota: não é ainda possível saber o valor do conteúdo das células na *stack frame*.



2.c) Com base na informação já obtida, construa agora o código *assembly* que substitui a parte "danificada". Considere que `soma_grandes.s` foi obtido com `gcc -O2 -S`.
Não esquecer de anotar devidamente este código (com comentários curtos e objetivos).

Nota 1:
Se o código tiver um ciclo, o controlo de permanência no ciclo na resolução desta alínea deve ser feito no início de cada iteração do ciclo e não no fim.

Nota 2:
Se compilar um código C desta função e usar essa versão de *assembly*, será considerado fraude e o trabalho receberá classificação negativa.

1000 = 0x3e8

```
soma_grandes:
    push %ebp
    movl %esp, %ebp
    push %esi                   # salvaguarda (para o "n")
    push %ebx                   # salvaguarda (para o "a")
    mov 0xc(%ebp), %ebx         # ebx = a*
    mov 0x8(%ebp), %esi         # esi = n
    xor %eax, %eax              # result = 0
    xor %ecx, %ecx              # i = 0
cmp %ecx, %esi                  # i-n (para o próximo jump)
jge "para o final"              # se i>=n então salta para o fim
mov (%ebx,%ecx,4), %edx         # guardou em edx o elemento "a[i]"
cmp $0x3e8, %edx                # compara "a[i]" com 1000
jle "para o incremento do i"    # se a[i] <= 1000, salta
add %edx, %eax                  # result = result + a[i]
inc %ecx                        # i++ (incrementa o "%ecx"
jP "para o cmp %ecx,%esi"       # salta logo para a condição
    pop %ebx
    pop %esi
    leave
    ret
```

2.d) Compile para *assembly* com `-O2` o programa C que criou no início para validar o código que criou. Escreva aqui o resultado e comente as diferenças com o seu código.

(na folha à parte)

2.

(d)

```
0 :  55                                  push  %ebp
1 :  89  e5                              mov   %esp, %ebp
3 :  56                                  push  %esi
4 :  53                                  push  %ebx
5 :  31  c0                              xor   %eax, %eax
7 :  8b  5d  08                          mov   0x8(%ebp), %ebx        (*1)
a :  31  c9                              xor   %ecx, %ecx
c :  39  d8                              cmp   %ebx, %eax             (*2)
e :  8b  75  0c                          mov   0xc(%ebp), %esi        (*3)
11 : 7d  13                              jge   26 <soma_grandes+0x26>
13 : 90                                  nop                          (*4)
14 : 8b  14  8e                          mov   (%esi,%ecx,4), %edx
17 : 81  fa  e8  03  00  00              cmp   $0x3e8, %edx
1d : 7e  02                              jle   21 <soma_grandes+0x14>
1f : 01  d0                              add   %edx, %eax
21 : 41                                  inc   %ecx
22 : 39  d9                              cmp   %ebx, %ecx
24 : 7c  ee                              jl    14 <soma_grandes+0x14>  (*5)
26 : 5b                                  pop   %ebx
27 : 5e                                  pop   %esi
28 : c9                                  leave
29 : c3                                  ret
```

(*1) : %edx e %esi estão trocados : → %ebx = n

(*2) : %eax = i((?)) ??? → para comparar → i - n (???)

(*3) : %esi = *a ↳ não faz sentido !!! ↳ pk na linha "1f" contraria isso

(*4) : não coloquei este nop

(*5) : este controlo não foi colocado no meu código na 2.c)

```
%ebx ⇐ n
%esi ⇐ *a
%eax ⇐ ???
%edx ⇐ a[i]
%ecx ⇐ ???
```

3. Criação e validação dum executável

Crie o executável `somaG` e ponha-o a correr. Mostre o que aparece no monitor como resultado. Se não produzir o resultado esperado, use o `gdb` para corrigir o código *assembly* da função.
Apresente numa folha separada a metodologia seguida para fazer o *debugging*, o código final e o resultado que apareceu no monitor quando o código estava correto.

```
[03a92846@sc TPC9]$ ./somaG
A sua funcao sera testada 3 vezes!!
Teste 0: 20 elementos. Resultado correcto=64469; Resultado obtido=49!
Teste 1: 50 elementos. Resultado correcto=255581; Resultado obtido=50!
Teste 2: 100 elementos. Resultado correcto=500123; Resultado obtido=51!
Teve sucesso 0 vezes!!

That's all, folks!
```

Apenas consegui entender como fazer isto neste exercício.

4. Diferenças na compilação com e sem otimização

4.a) Caracterização das diferenças

Apresente aqui as 2 versões de código *assembly* (com e sem otimização) e marque nas duas versões as instruções que são diferentes, explicando a diferença.