Nº.) a93280
Nome) Miguel Ângelo Machado Martins
Turma) PL5

① Um possível algoritmo

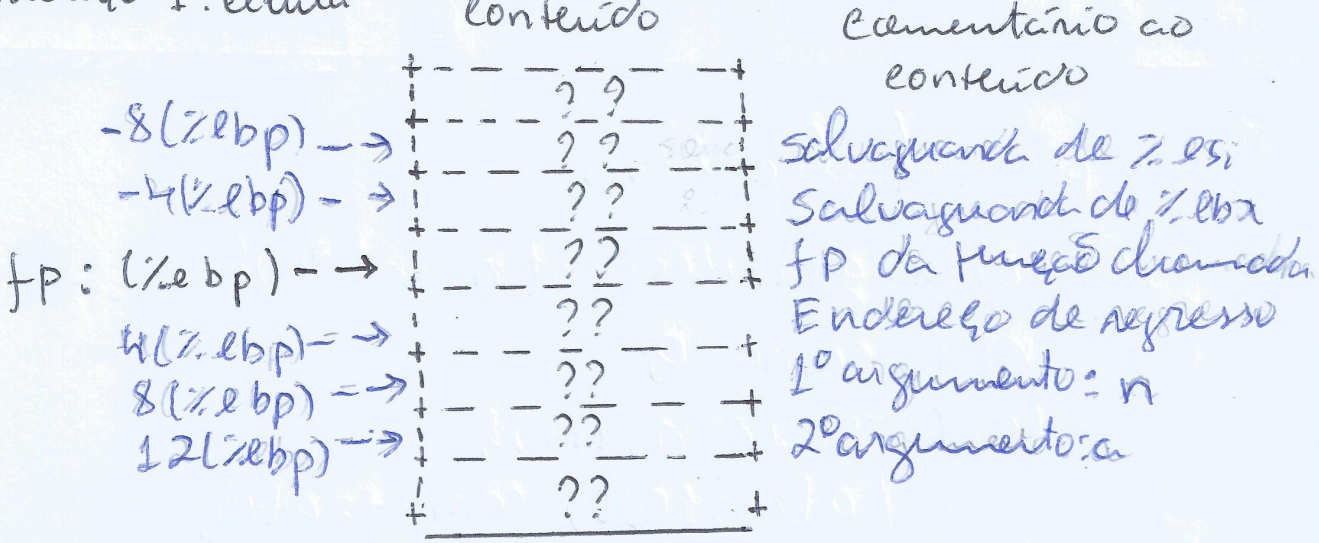
```
int soma_grandes(int n, int *a){
    int i, soma=0;
    for(i=0; i<=n, i++){
        if(a[i]>1000) soma+=a[i];
    }
    return soma;
}
```

② 2. a) ↓

Utilização dos Registos

Registo	Variável	Comentários
soma	%eax	Registo eax devolve a soma ↗return
i	%ecx	Escolha aleatória
a	%ebx	Esc. aleat.
n	%esi	Esc. aleat.
a[i]	%edx	Esc. aleat.
		Esc. aleat

2.b)



2.c)

soma	%eax	i	%ecx	a	%ebx	n	%esi	a[i]	%edx

```
soma_grandes:
        pushl   %ebp
        movl    %esp, %ebp
        pushl   %ebx                #salvaguarda o regist
        pushl   %esi                #salvaguarda o registo
        movl    8(%ebp), %esi       #n (1º argumento)
        movl    12(%ebp), %ebx      # a (2º argumento)
        xorl    %eax, %eax          # soma=0 (inicializa soma a zero)
        xorl    %ecx, %ecx          #i=0

TEST:
        cmpl    %esi, %ecx          #compara n e i
        jge     END_CICLE           # se i>=n (sai do for e vai para END_CICLE)
        movl    (%ebx, %ecx, 4), %edx   #edx=a[i]
        cmpl    $1000, %edx         #compara constante 1000 com a[i]
        jle     END_IF              #se a[i]<=1000 salta por cima do if (não adiciona
        addl    %edx, %eax          #soma+=a[i]
```

```
END-IF:
    incl %eax          # i++
    jmp TEST           # repete o ciclo for

END-CICLE:
    popl %esi          # recupera registo esi
    popl %ebx          # recupera registo ebx

    leave
    ret
```

2.d)

```
Soma_grandes:
        pushl   %ebp
        movl    %esp, %ebp
        pushl   %esi
        pushl   %ebx
        xorl    %eax, %eax
        movl    8(%ebp), %ebx
        xorl    %eax, %eax
        empl    %ebx, %eax
        movl    12(%ebp), %esi
        jg      .L9
        .p2align 2,,3
.L7:
        movl    (%esi, %eax, 4), %edx
        empl    $1000, %edx
        jle     .L4
        addl    %edx, %eax
```

```
.L4
    incl    %ecx
    cmpl    %ebx, %ecx
    jle     .L7

.L9
    popl    %ebx
    popl    %esi
    leave
    ret
```

(3) ./somaG

A sua função será testada 3 vezes!!

Teste 0: 20 elementos: Resultado correcto = 64469; Resultado obtido = 64469!

Teste 1: 50 elementos; Resultado correcto = 255581; Resultado obtido = 255581!

Teste 2: 100 elementos: Resultado correcto = 500123; Resultado obtido = 500123!

Teve sucesso 3 vezes!!

That's all, folks!

(4) a

SEM OTIMIZAÇÃO

9 acessos à memória

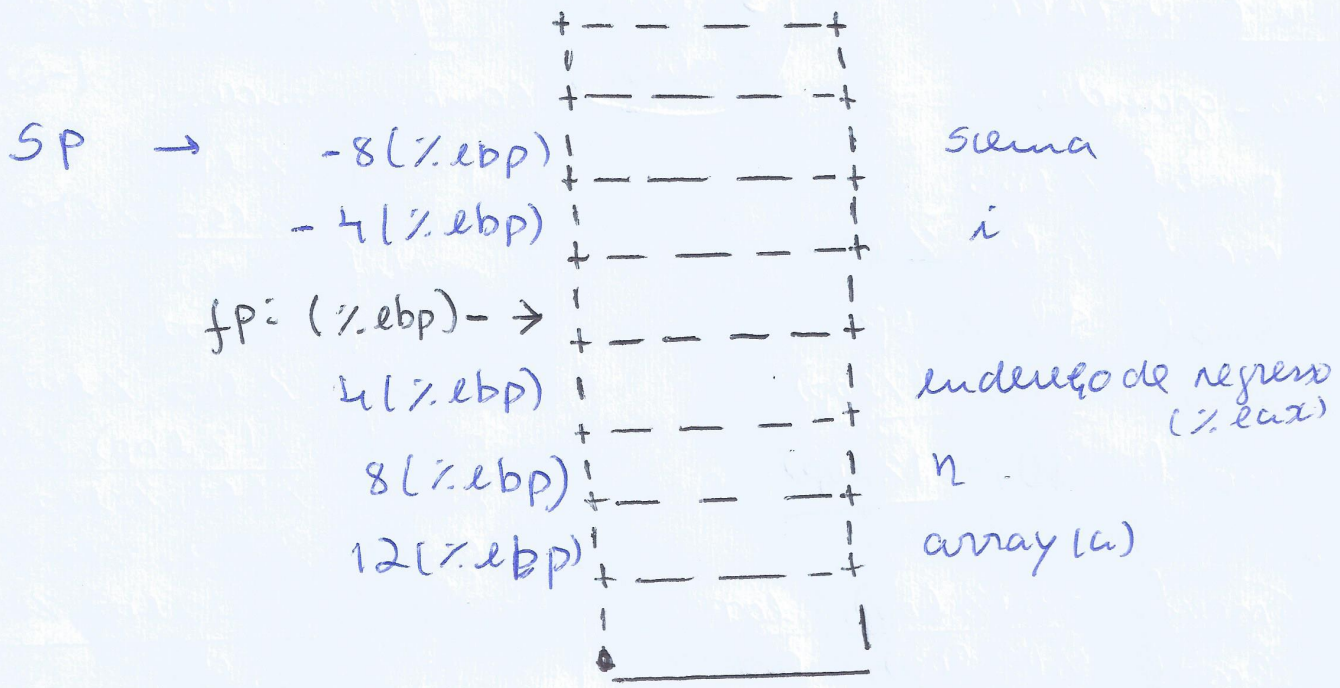
```
soma-grandes
        pushl   %ebp
        movl    %esp, %ebp
        subl    $8, %esp
        movl    $0, -8(%ebp)
        movl    $0, -4(%ebp)
.L2:
        movl    -4(%ebp), %eax
        cmpl    8(%ebp), %eax
        jle     .L5
        jmp     .L3
.L5:
        movl    -4(%ebp), %eax
        leal    0, (, %eax, 4), %edx
        movl    12(%ebp), %eax
        cmpl    $1000, (%eax, %edx)
        jle     .L4
        movl    -4(%ebp), %eax
        leal    0, (, %eax, 4) %edx
        movl    12(%ebp), %eax
        movl    (%eax, %edx), %edx
        leal    -8(%ebp), %eax
        addl    %edx, (%eax)
.L4:
        leal    -4(%ebp), %eax
        incl    (%eax)
        jmp     .L2
.L3:
        movl    -8(%ebp), %eax
        leave
        ret
```

COM OTIMIZAÇÃO (-O2)

4 acessos à memória

```
soma-grandes
        pushl   %ebp
        movl    %esp, %ebp
        pushl   %esi
        pushl   %ebx
        xorl    %eax, %eax
        movl    8(%ebp), %ebx
        xorl    %ecx, %ecx
        cmpl    %ebx, %eax
        movl    12(%ebp), %esi
        jg      .L9
.L7:
        movl    (%esi, %ecx, 4), %edx
        cmpl    $1000, %edx
        jle     .L4
        addl    %edx, %eax
.L4:
        incl    %ecx
        cmpl    %ebx, %ecx
        jle     .L7
.L9:
        popl    %ebx
        popl    %esi
        leave
        ret
```

Endereço 1ª célula	conteúdo	Comentário ao conteúdo



→ Uma diferença grande entre as duas versões é o acesso à memória:

Existem muitos ⊕ acessos à memória na versão sem otimização (tal acesso é feito por exemplo com a movl).

→ Na versão com otimização (-O2) existem ⊕ registos. (1er ⊕ registos é caro. No entanto compensa no ponto de não se ter de aceder tantas vezes à memória.

4b

 → ciclo for

 → estruturas de controlo (if...)

 → Acesso a instruções

As estruturas de controlo na versão sem otimização acedem diretamente à memória, sendo menos eficientes.