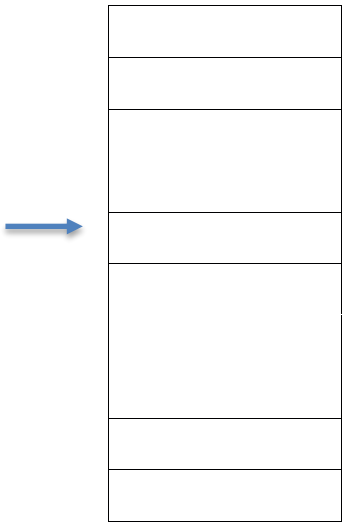
1. As 10 questões nesta prova para 20 valores estão cotadas para 2 valores cada (duração: 1h30m).
2. A figura anexa contém a listagem de uma função em C e o seu estado após uma interrupção (*breakpoint*) da sua execução no servidor de apoio às sessões laboratoriais (IA-32).
3. É permitido o uso de uma nota auxiliar de memória, manuscrita, com dimensão máxima 1 folha A4.
4. Não é permitido o uso de qualquer equipamento eletrónico para além do relógio.

1. A figura em baixo representa algumas posições de memória onde se encontra o quadro de ativação (*stack frame*) associada à função `soma_conta`. Os endereços crescem para baixo e cada retângulo representa 4 células de memória (i.e, 4 *bytes*).
Para a versão compilada com `-O2` **identifique** claramente todos os campos do quadro de ativação (à direita de cada retângulo na figura), considerando que a posição apontada pelo *frame pointer* está representada com uma seta à esquerda da figura.
Nota: não necessita de indicar os conteúdos de cada célula (i.e., não escreva nada dentro dos retângulos!).



2. Com base na análise das 2 versões dos códigos em *assembly* (com e sem otimização) **indique os endereços** das instruções que inicializam a zero as variáveis locais (para cada uma das versões).

3. Para a versão compilada com otimização `-O2` **preencha** a tabela abaixo, indicando a associação dos registos às variáveis locais e parâmetros do código C da função `soma_conta`. **Complemente** a resposta indicando as instruções responsáveis pela sua inicialização.

Reg	Variável / Argumento	Instrução *assembly*
`%eax`		
`%ebx`		
`%ecx`		
`%edx`		
`%esi`		
`%edi`		

4. Pretende-se modificar a expressão no corpo do ciclo `for` no código fonte

```
*s += a[i];
```

para

```
*s = 1 + conta + a[i] * 4;
```

Codifique esta expressão numa única instrução *assembly* (IA-32).

5. Pretende-se modificar a estrutura do ciclo `for` para um `do..while`.
Preencha os espaços em branco do código *assembly* em baixo, equivalente ao ciclo `for` do código C apresentado.

Código C

```
do{
  ...
   i++;
}
while(i < n);
```

Código *assembly*

```
while:  ...
        _____  %ecx
        cmpl   _____, %esi
        ______ out
        ______ while
out:  ...
```

6. **Indique** os valores armazenados nas posições da pilha referenciadas por `%ebp+12` e `%ebp+16`.

7. **Indique** o valor do *frame pointer* da função que chamou esta função e o número de *bytes* que a função chamadora tem reservados para variáveis locais e para salvaguarda de registos (excluindo `%ebp`).

8. A função `soma_conta` contém pelo menos uma ineficiência em termos de desempenho nesta implementação em C. **Identifique-a** e **sugira** alterações ao código para melhorar a sua *performance*.

9. **Introduza** comentários/anotações que explicitem o papel de cada uma das instruções no código *assembly* que foram destacadas a **negrito**, na figura do código *assembly* obtido com `-O2`.

10. Considere agora que este código C foi compilado para uma versão do MIPS também com 32-bits e que o *instruction set* deste MIPS **(i)** não tem instruções de `mov` (só tem `load` e `store` para acessos à memória), **(ii)** as instruções aritméticas e lógicas especificam 3 operandos (pela ordem `dest=fonte1<op>fonte2`) e **(iii)** tem o mesmo suporte a estruturas de controlo que o IA-32.
Reescreva o código das primeiras 7 instruções *assembly* para esta versão do MIPS **justificando** todas as alterações que introduzir.
(Nota: pode usar a sintaxe do GNU para o IA-32, substituindo apenas o nome dos registos para `%r0`, `%r1`, … e indicando eventualmente a convenção que adotou para o uso de alguns dos registos).

```
int soma_conta (int *a, int n, int *s)
{
  int conta=0;
  int i;


  for(i=0; i < n; i++)
  {
   if (a[i] > 20 && a[i] < 30)
     {
      *s += a[i];
      conta++;
     }
  }

  return conta;
}
```

— *objdump* parcial com `gcc -S -O0` ——

```
080483ac <soma_conta>:
 80483ac: 55          push  %ebp
 80483ad: 89 e5       mov   %esp,%ebp
 80483af: 53          push  %ebx
 80483b0: 83 ec 08    sub   $0x8,%esp
 80483b3: c7 45 f8 00 00 00 00
              movl  $0x0,0xfffffff8(%ebp)
 80483ba: c7 45 f4 00 00 00 00
              movl  $0x0,0xfffffff4(%ebp)
 80483c1: 8b 45 f4
              mov   0xfffffff4(%ebp),%eax
 80483c4: 3b 45 0c    cmp   0xc(%ebp),%eax
 80483c7: 7c 02       jl    80483cb
 80483c9: eb 4c       jmp   8048417
 80483cb: 8b 45 f4
              mov   0xfffffff4(%ebp),%eax
 80483ce: 8d 14 85 00 00 00 00
              lea   0x0(,%eax,4),%edx
 80483d5: 8b 45 08    mov   0x8(%ebp),%eax
 80483d8: 83 3c 10 14
              cmpl  $0x14,(%eax,%edx,1)
...
```

——— *objdump* parcial com `gcc -S -O2` — ——

```
08048390 <soma_conta>:
 8048390: 55          push  %ebp
 8048391: 89 e5       mov   %esp,%ebp
 8048393: 57          push  %edi
 8048394: 56          push  %esi
 8048395: 53          push  %ebx
 8048396: 8b 75 0c    mov   0xc(%ebp),%esi
 8048399: 31 db       xor   %ebx,%ebx
 804839b: 31 c9       xor   %ecx,%ecx
 804839d: 39 f3       cmp   %esi,%ebx
 804839f: 8b 7d 10    mov   0x10(%ebp),%edi
 80483a2: 7d 16       jge   80483ba <soma_conta+0x2a>
 80483a4:  8b 45 08   mov   0x8(%ebp),%eax
 80483a7:  8b 14 88   mov   (%eax,%ecx,4),%edx
 80483aa:  8d 42 eb   lea   0xffffffeb(%edx),%eax
 80483ad:  83 f8 08   cmp   $0x8,%eax
 80483b0:  77 03      ja    80483b5 <soma_conta+0x25>
 80483b2:  01 17      add   %edx,(%edi)
 80483b4:  43         inc   %ebx
 80483b5:  41         inc   %ecx
 80483b6:  39 f1      cmp   %esi,%ecx
 80483b8:  7c ea      jl    80483a4 <soma_conta+0x14>
 ...
```

— *breakpoint* em `soma_conta` (executável com `-O2`) —

```
(gdb) info registers
eax    0xbfffd720
ecx    0x1
edx    0xc
ebx    0x1
esp    0xbfffd700
ebp    0xbfffd70c
esi    0x4
edi    0xbfffd730
eip    0x80483b5

 (gdb) x/8xw $esp
0xbfffd700:
 0x0016d51a  0x00241f60 0x00000005  0xbfffd738
0xbfffd710:
 0x0804841e
```