Versão: A Nota mínima: **7.5/20 valores** / Duração: 120 minutos

Número:_____ **Nome:**_____

Responda aos grupos III, IV e V em folhas A4 separadas. No final, deverá entregar 4 folhas assinadas, indicando a versão do enunciado.

[8v] **Grupo I - Assinale no seguinte grupo se as frases são verdadeiras ou falsas (uma resposta errada desconta 50% de uma correcta).**

	V	F
1) Em C, o *cast* de uma variável do tipo `int` para uma do tipo `unsigned int` altera o padrão de bits da variável	☐	☐
2) Em C, numa expressão com variáveis do tipo `int` com e sem sinal, todas as variáveis são convertidas para valores sem sinal	☐	☐
3) Em C, se tivermos um `char` com representação binária de `10011010`, o *cast* para um `short` resulta no valor `1111111110011010`	☐	☐
4) Admita um inteiro *x* com valor `0x01234567` e um valor dado por `&x` de `0x100`. Logo, o valor presente no byte `0x101` é `0x45`	☐	☐
5) Em C, a adição de duas variáveis do tipo `int` com valores positivos nunca pode resultar num valor negativo	☐	☐
6) Em C, a operação `u<<k` resulta em $u*2^k$, independentemente da variável `u` ter ou não sinal	☐	☐
7) Em Assembly, a instrução `movl %eax, (%esp)` pode ser usada para escrever o valor de `%eax` no topo da stack	☐	☐
8) Em Assembly, a instrução `cmp` não altera os seus parâmetros nem os bits do registo EFLAGS	☐	☐
9) Admita que o endereço de uma variável `x` está armazenado em `%edi`. É possível alterar o valor de `x` executando `movl $15,%edi`	☐	☐
10) A instrução `movl 4(%ebp),%eax` armazena em `%eax` o valor antigo de `%ebp` numa função que inicia com o prólogo estudado	☐	☐
11) Admita que o endereço de um vetor do tipo `int` está armazenado em `%esi` e que o valor de `%ecx` é 2. A instrução `leal (%esi,%ecx,4),%esi` armazena em `%esi` o endereço do terceiro elemento do vetor.	☐	☐
12) A execução da instrução `call` não implica a alteração do valor do registo `%esp`.	☐	☐
13) Não é possível passar o endereço de uma variável local para outra função chamada pela primeira usando a *stack*	☐	☐
14) Depois de um `call` de uma função com 3 parâmetros inteiros podemos executar `subl $12, %esp` para retirar os parâmetros da stack	☐	☐
15) A convenção de salvaguarda de registos indica que o registo `%esi` deve ser gerido pela função invocada	☐	☐
16) Admita a declaração da matriz `int m[4][5]`. Em Assembly, o endereço do elemento `m[i][j]` é dado pela expressão `m+20*i+4*j`	☐	☐
17) Uma estrutura, alinhada de acordo com as regras estudadas, com um vector de 2 `char`, 1 `int` e 1 `short` (por esta ordem) ocupa 8 bytes	☐	☐
18) O espaço ocupado por uma estutura alinhada é sempre o mesmo, independemente da ordem dos seus campos	☐	☐
19) O espaço ocupado por uma *union* é sempre o mesmo, independentemente da ordem dos seus campos	☐	☐
20) O compilador não tem qualquer dificuldade em mover a invocação de funções para outro local do programa com vista à sua optimização	☐	☐

[2v] **Grupo II – Para cada uma das expressões seguintes em C indique se a expressão é <u>sempre</u> verdadeira (uma resposta errada desconta 50% de uma correcta).**

Considere as seguintes declarações:

```
int x = f(); /* Retorna valor em [0, INT_MAX] */
int y = f(); /* Retorna valor em [0, INT_MAX] */

/* Conversão para outros formatos */
unsigned ux = (unsigned) x;
unsigned uy = (unsigned) y;
double   dx = (double) x;
double   dy = (double) y;
```

Expressão	Sempre verdade?
`x*x >= 0`	
`x+y == uy+ux`	
`x == (int)(float)x`	
`(double)(float)x == (double)ux`	
`dx + dy == (double)(y+x)`	
`(dx + dy) - dx == dy`	

[5v] **Grupo III – Responda numa folha A4 separada que deve assinar e entregar no final do exame. Justifique cada uma das respostas.**

Considere as seguintes declarações:

```
typedef struct {
    char c;
    double *p;
    int i;
    double d;
    short s;
} struct1;
```

```
typedef struct {
    float x;
    struct1 *s1;
    char e;
    int v[4];
} struct2;
```

[1.5v] **a)** Indique o alinhamento dos campos de uma estrutura do tipo `struct1`. Indique claramente, para cada campo, o seu endereço, bem como as partes alocadas mas não usadas para satisfazer as restrições de alinhamento. **Admita que a estrutura está colocada a partir do endereço `0x100`.**

[1.5v] **b)** Se definirmos os campos da estrutura `struct1` por outra ordem é possível reduzir o número de bytes necessários para o seu armazenamento? Justifique a sua resposta, indicando, em caso afirmativo, qual a ordem dos campos que garante o menor tamanho.

[2v] **c)** Para cada sequência de código em Assembly à esquerda, complete a função correspondente em C à direita, considerando as declarações iniciais das estruturas (**escreva a função completa na folha A4**).

```
f1:
   pushl %ebp
   movl %esp, %ebp
   movl 8(%ebp),%eax
   movl 16(%eax), %eax
   movl %ebp, %esp
   popl %ebp
   ret

f2:
   pushl %ebp
   movl %esp, %ebp
   movl 8(%ebp),%eax
   movl 4(%eax), %eax
   movl 8(%eax),%eax
   movl %ebp, %esp
   popl %ebp
   ret
```

```
___ f1(struct2 *s2)
 {
   return ____________________;
 }


___ f2(struct2 *s2)
 {
   return ____________________;
 }
```

[2v] **Grupo IV – Responda numa folha A4 separada que deve assinar e entregar no final do exame. Justifique a sua resposta.**

O código seguinte em C preenche a diagonal de uma matriz N*N com o valor passado no argumento `val`.

```
void fix_set_diag(int m[N][N], int val){
    int i;
    for(i = 0; i < N; i++)
        m[i][i] = val;
}
```

Quando compilado para Assembly, o GCC gera o seguinte código para um valor de N = 4

```
fix_set_diag:
   pushl %ebp
   movl %esp, %ebp
   movl 8(%ebp), %ecx
   movl 12(%ebp), %edx
   movl $0, %eax
.L14:
   movl %edx, (%ecx,%eax)
   addl $20, %eax
   cmpl $80, %eax
   jne .L14
   ret
```

Reescreva a função `fix_set_diag` em C usando uma optimização similar à demonstrada no código Assembly. Use expressões que recorram à variável N em vez de valores inteiros constantes, para que o seu código continue a funcionar se o valor de N for alterado.

[3v] **Grupo V – Responda numa folha A4 separada que deve assinar e entregar no final do exame. Justifique a sua resposta.**

Considere o seguinte código Assembly:

```
loop:
   pushl %ebp
   movl %esp,%ebp
   movl 8(%ebp),%ecx
   movl 12(%ebp),%edx
   xorl %eax,%eax
   cmpl %edx,%ecx
   jle .L4
.L6:
   decl %ecx
   incl %edx
   incl %eax
   cmpl %edx,%ecx
   jg .L6
.L4:
   incl %eax
   movl %ebp,%esp
   popl %ebp
   ret
```

Com base no código Assembly à esquerda, preencha os espaços em branco no código correspondente em C. Apenas pode usar as variáveis `x`, `y` e `result` nas expressões (*não use nomes de registos!*) (**escreva a função completa na folha A4**).

```
int loop(int x, int y){
    int result;

    for(_______; _______; result++){
        ___________;
        ___________;
    }

    ___________;

    return result;
}
```

- Consulta apenas das 2 folhas de consulta (C e Assembly).
- A infracção implica, no mínimo, a ANULAÇÃO da prova.
- Quando omissa a arquitectura, considere Linux/IA32.

Versão: A Nota mínima: **7.5/20 valores** / Duração: 120 minutos

Número:_____ **Nome:**_____

Responda aos grupos II, III, IV e V em folhas A4 separadas.

[8v] **Grupo I - Assinale no seguinte grupo se as frases são verdadeiras ou falsas (uma resposta errada desconta 50% de uma correcta).**

	V	F
1) Em C, o *cast* de uma variável do tipo `int` para uma do tipo `float` altera o padrão de bits da variável	☐	☐
2) Em C, o *cast* implícito em determinadas situações de variáveis com sinal para valores sem sinal pode levar a *bugs* no programa	☐	☐
3) Admita um `int x` com valor `0x01234567` e um valor dado por `&x` de `0x100`. Logo, o valor presente no byte `0x100` é `0x67`	☐	☐
4) Em C, se tivermos uma variável x do tipo `short` com o valor `0x1234`, o valor `-0x1234` pode ser obtido através de `~x + 1`	☐	☐
5) Em C, a adição de duas variáveis `u` e `v` do tipo `int` tem como resultado `(u+v) mod 32`	☐	☐
6) Em C, é garantido que o resultado de uma divisão inteira por 2^k, obtida através de `u >> k`, é correctamente arredondado se `u<0`	☐	☐
7) Admita que `ptr` é uma variável do tipo `char*`. Então, a expressão `(int*)ptr + 7` avança 28 bytes na memória	☐	☐
8) Em Assembly, a instrução `movb (%esi), (%edi)` permite copiar um byte para uma nova posição de memória numa única instrução	☐	☐
9) Em Assembly, o resultado das instruções de salto condicional depende do valor dos bits do registo EFLAGS	☐	☐
10) Admita que `%edi` e `int *ptr` armazenam o endereço do inteiro `x`. Então, `movl $1,(%edi)` é o equivalente a `*ptr = 1` em C	☐	☐
11) Os parâmetros de uma função não podem ser acedidos usando o registo `%esp` em vez do `%ebp` como base do endereçamento	☐	☐
12) Admita `0xF000` e `0x0100` em `%edx` e `%ecx`, respetivamente. `leal (%edx,%ecx,4),%esi` armazena em `%esi` o valor `0xF400`.	☐	☐
13) Em IA32 é usada a *stack* para armazenar o valor de retorno de uma função, à semelhança do que acontece com o seu endereço de retorno	☐	☐
14) Admita que o valor de `%esp` é `0x100C`. A execução da instrução `ret` coloca o valor de `%esp` em `0x1010`	☐	☐
15) Os registo `%eax` é local a cada uma das funções, o que dispensa qualquer cuidado no seu uso entre invocações de funções	☐	☐
16) Admita a matriz global `short int m[5][3]`. Em Assembly, acedemos ao valor de `m[3][0]` avançando 18 bytes a partir de `m`	☐	☐
17) Uma estrutura, alinhada de acordo com as regras estudadas, com um vector de 2 `char`, 1 `int` e 1 `short` (por esta ordem) ocupa 12 bytes.	☐	☐
18) É **sempre** possível diminuir o tamanho de um estrutura alinhada alterando a ordem dos seus campos	☐	☐
19) É possível redimensionar, com a função `realloc`, o tamanho um vetor de inteiros `vec` declarado estaticamente com `int vec[10]`	☐	☐
20) A possibilidade de existirem diversas referências para a mesma posição de memória dificulta a optimização efectuada pelo compilador	☐	☐

[2v] **Grupo II – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

Considere o código da função `sum` ao lado que pretende somar os elementos de um vetor `a`. O número de elementos do vetor é passado no parâmetro `unsigned int length`.

Quando invocada com o valor 0 no argumento `length`, a função deveria retornar 0.0. No entanto, é gerado um erro de acesso à memória.

```
float sum(float a[], unsigned int length){
   int i;
   float result = 0.0;

   for(i=0; i<= length-1; i++)
      result += a[i];
   return result;
}
```

[1v] **a)** Explique detalhadamente porque o erro acontece.
[1v] **b)** Demonstre como o código poderia ser corrigido. **Justifique a sua resposta.**

[5v] **Grupo III – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

Considere as seguintes declarações:

```
typedef struct {
    short int code;
    long int start;
    char raw[3];
    double data;
} OldSensor;
```

```
typedef struct {
    short int code;
    short int start;
    char raw[5];
    short int sense;
    short int ext;
    double data;
} NewSensor;
```

[1.5v] **a)** Indique o alinhamento dos campos de uma estrutura do tipo `OldSensor`. Indique claramente, para cada campo, o seu endereço, bem como as partes alocadas mas não usadas para satisfazer as restrições de alinhamento. Indique o tamanho total da estrutura. **Admita que a estrutura está colocada a partir do endereço `0x100`.**

[1.5v] **b)** Se definirmos os campos da estrutura `OldSensor` por outra ordem é possível reduzir o número de bytes necessários para o seu armazenamento? **Justifique a sua resposta.** Indique, em caso afirmativo, qual a ordem dos campos que garante o menor tamanho, o novo endereço de cada campo e das partes alocadas mas não usadas, bem como o novo tamanho total da estrutura.

[2v] **c)** Considere o seguinte fragmento de código em C, respeitando as declarações iniciais das estruturas.

```
void xpto(OldSensor *oldData){
   NewSensor *newData;

   /* zeros out all the space of oldData */
   bzero((void *)oldData,sizeof(OldSensor));

   oldData->code = 0x104f;
   oldData->start = 0x80501ab8;
   oldData->raw[0] = 0xe1;
   oldData->raw[1] = 0xe2;
   oldData->raw[2] = 0x8f;
   oldData->data = 1.5;

   newData = (NewSensor *) oldData;
   ...
 }
```

Admita que após estas linhas de código começamos a aceder aos campos da estrutura `NewSensor` através da variável `newData`. Indique, em hexadecimal, o valor de cada um dos campos de `newData` indicados a seguir. Tenha em atenção a ordenação dos bytes em memória em Linux/IA32!

a) `newData->code` = 0x_____
b) `newData->raw[0]` = 0x_____
c) `newData->raw[2]` = 0x_____
d) `newData->raw[4]` = 0x_____
e) `newData->sense` = 0x_____

[2v] **Grupo IV – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

Admita a existência de um vetor `v` preenchido com um número arbitrário de inteiros positivos e cuja última posição preenchida tem o valor -1. O código seguinte em C determina a soma de todos os seus valores positivos.

```
void sum_elements(int *v, int *sum){
    int i, val;
    *sum = 0;
    for(i = 0; i < vec_length(v); i++){
       get_element(v, i, &val);
       *sum += val;
    }
}
```

```
void get_element(int *v, int i, int *val){
   *val = v[i];
}

int vec_length(int *v){
   int i=0, length=0;
   while(v[i++]!= -1)
      length++;
   return length;
}
```

Reescreva a função `sum_elements` em C usando as técnicas de optimização estudadas nas aulas. **Indique claramente cada uma das optimizações usadas sob a forma de comentário no código.**

[3v] **Grupo V – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

Considere o seguinte código Assembly:

```
loop_func:
   pushl %ebp
   movl %esp,%ebp
   pushl %ebx
   movl 8(%ebp),%ecx
   movl 12(%ebp),%ebx
   movl $1, %eax
   cmpl %ecx,%ebx
   jle .L4
.L6:
   leal (%ebx,%ecx),%edx
   imull %edx,%eax
   shll %ecx
   cmpl %ecx,%ebx
   jg .L6
.L4:
   movl $0, %edx
   idivl %ebx
   popl %ebx
   movl %ebp,%esp
   popl %ebp
   ret
```

Com base no código Assembly à esquerda, preencha os espaços em branco no código correspondente em C. Apenas pode usar as variáveis `a`, `b` e `result` nas expressões (*não use nomes de registos!*) (**escreva a função completa na folha A4**).

```
int loop_func(int a, int b){
    int result = _________;

    while(_____){
        ___________;
        ___________;
    }

    __________;

    return result;
}
```

Versão: A Nota mínima: **7.5/20 valores** / Duração: 120 minutos

Número:_____ **Nome:**_____

Responda aos grupos II, III, IV e V em folhas A4 separadas.

[8v] **Grupo I - Assinale no seguinte grupo se as frases são verdadeiras ou falsas (uma resposta errada desconta 50% de uma correcta).**

	V	F
1) Admita a variável `unsigned char x` em C. O valor armazenado em `x` depois de executar "`x = -1; x = x >> 1;`" é 127.	☐	☐
2) As operações aritméticas de soma e subtracção de inteiros têm uma implementação diferente em *hardware* para valores com e sem sinal.	☐	☐
3) Em IA32, uma arquitetura *little-endian*, considerando o vetor `short x[10]`, o elemento `x[1]` está num endereço menor que `x[0]`	☐	☐
4) O vetor "`int *vec = (int*)malloc(16);`" pode armazenar 16 inteiros tal como se tivesse sido definido como "`int vec[16];`".	☐	☐
5) Em C, a multiplicação de duas variáveis `u` e `v` do tipo `int` pode resultar num valor menor do que os armazenados em `u` ou `v`	☐	☐
6) Em C, as expressões "`x * 35`" e "`(x<<5) + (x<<2) - x`" são sempre equivalentes para qualquer valor de `unsigned int x`	☐	☐
7) Admita que `ptr` é uma variável do tipo `char*`. Então, em C, a expressão `(short*)ptr + 7` avança 14 bytes na memória	☐	☐
8) Admita que declara a variável `int x` na função *main* em C. O compilador pode atribuir `x` a um registo ou a um endereço na *heap*	☐	☐
9) Em Assembly, a instrução "`imull %edx`" duplica o valor do registo usado como argumento.	☐	☐
10) Em Assembly, a instrução "`pushl %eax`" é equivalente a "`movl %eax, (%esp)`" seguido de "`addl $-4, %esp`"	☐	☐
11) A adição de dois bytes com sinal com valores 0xAC e 0x8A deixa as *flags* do registo EFLAGS com os valores ZF=0, SF=1, CF=1, OF=1	☐	☐
12) Em IA32, a instrução `test` compara o valor dos seus operandos através de um subtração.	☐	☐
13) Em IA32, a *stack* é usada para suportar a invocação de funções e o retorno para a função invocadora com `call` e `ret`, respetivamente.	☐	☐
14) Admita que o valor de `%esp` é `0x1000`. A execução da instrução `jmp` coloca o valor de `%esp` em `0xFFC`	☐	☐
15) De acordo com a convenção usada em Linux/IA32, a responsabilidade de salvaguarda e restauro de `%esi` é da função invocada	☐	☐
16) Admita a matriz global `short m[10][3]`. Em Assembly, acedemos ao valor de `m[3][1]` avançando 20 bytes a partir de `m`	☐	☐
17) Uma estrutura, alinhada de acordo com as regras estudadas, com 2 `int`, um vetor de 7 `char` e 1 `short` (por esta ordem) ocupa 20 bytes	☐	☐
18) O tamanho de uma *union* sujeita a alinhamento pode ser menor se indicarmos os seus campos por ordem crescente do seu tamanho	☐	☐
19) A fragmentação da *heap* pode impedir a alocação de um novo bloco mesmo que exista esse número de bytes livres	☐	☐
20) A invocação de funções introduz *overhead* e limita as possibilidades de otimização dos programas pelo compilador	☐	☐

[2v] **Grupo II – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

Pediram-lhe para implementar uma função que determine se a primeira *string* é maior do que a segunda. Decidiu usar a função `strlen` definida na biblioteca "`string.h`" com a seguinte declaração:

```
size_t strlen(const char *s);
```

A sua primeira tentativa resultou na função `strlonger` descrita ao lado. Quando a testou, verificou que os resultados nem sempre são os esperados. Depois de alguma investigação, descobriu que o tipo `size_t` está definido em "`stdio.h`" como sendo `unsigned int`.

```
int strlonger(char *s, char *t){

   return strlen(s) - strlen(t) > 0;

}
```

[1v] **a)** Em que casos irá a função definida por si produzir um resultado incorreto? Explique como é que esse resultado incorreto é possível.
[1v] **b)** Demonstre como o código poderia ser corrigido. **Justifique a sua resposta.**

[5v] **Grupo III – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

Considere as seguintes declarações:

```
typedef struct {
    char a[3];
    short int b;
    long long int c;
    int d;
    structB *ptrB;
    char e;
} structA;
```

```
typedef struct {
    int a;
    char b;
    short c;
    int d;
} structB;
```

[1.5v] **a)** Indique o alinhamento dos campos de uma estrutura do tipo `structA`. Indique claramente, para cada campo, o seu endereço, bem como as partes alocadas mas não usadas para satisfazer as restrições de alinhamento. Indique o tamanho total da estrutura. **Admita que a estrutura está colocada a partir do endereço `0x100`.**

[1.5v] **b)** Se definirmos os campos da estrutura `structA` por outra ordem é possível reduzir o número de bytes necessários para o seu armazenamento? **Justifique a sua resposta** indicando, em caso afirmativo, qual a ordem dos campos que garante o menor tamanho, o novo endereço de cada campo e das partes alocadas mas não usadas, bem como o novo tamanho total da estrutura.

[2v] **c)** Considere o seguinte fragmento de código em C:

```
structA matrix[4][5];

int return_structB_d(int i, int j){
   return matrix[i][j].ptrB->d;
}
```

Reescreva a função `return_structB_d` em Assembly. Na sua resolução tenha em consideração que a matriz `matrix` é global e respeite as declarações iniciais das estruturas. **Comente o seu código.**

[2v] **Grupo IV – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

Admita a seguinte função em C que recebe como primeiro parâmetro um vetor `strs` de apontadores para *strings* e como segundo parâmetro o endereço de um inteiro `res` no qual a função armazena o resultado.

```
void handle_strs(char *strs[20], int *res){
 int i, j;

 *res = 0;

 for(i = 0; i < 20; i++){
  if(strlen(strs[i]) < i*10){
   *res += strlen(strs[i]);
  }else{
   for(j=0; j < strlen(strs[i]); ++j)
    *res += (16*i + get_char_at(strs[i],j));
  }
 }
}
```

```
int get_char_at(char *str, int pos){
  return str[pos];
}
```

Apresente uma segunda versão da função `handle_strs` em C com a mesma funcionalidade, mas melhor desempenho. Admita que o compilador que é usado não efetua nenhuma otimização. **Indique claramente cada uma das otimizações usadas sob a forma de comentário no código.**

[3v] **Grupo V – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

```
fun:
  pushl %ebp
  movl  %esp,%ebp
  pushl %esi
  pushl %ebx
  movl  8(%ebp),%ebx
  movl  12(%ebp),%esi
  xorl  %edx,%edx
  xorl  %ecx,%ecx
  cmpl  %ebx,%ecx
  jge    .L3
.L1:
  movl  (%esi,%ecx,4),%eax
  cmpl  %edx,%eax
  jle   .L2
  movl  %eax,%edx
.L2:
  incl  %edx
  incl  %ecx
  cmpl  %ebx,%ecx
  jl    .L1
.L3:
  movl  %edx,%eax
  popl  %ebx
  popl  %esi
  movl  %ebp,%esp
  popl  %ebp
  ret
```

Com base no código Assembly à esquerda, preencha os espaços em branco no código correspondente em C. Apenas pode usar as variáveis `n`, `a`, `i` e `x` nas expressões (*não use nomes de registos!*) (**escreva a função completa na folha A4**).

```
int fun(int n, int *a){

  int i;
  int x = __________;

  for(i=___; ____; i++){

     if(___________)
        x = ________;

     ________;
  }

  return x;
}
```

- Consulta apenas das 2 folhas de consulta (C e Assembly).
- A infracção implica, no mínimo, a ANULAÇÃO da prova.
- Quando omissa a arquitectura, considere Linux/IA32.

Versão: A Nota mínima: **7.5/20 valores** / Duração: 120 minutos

Número:_____ **Nome:**_____

Responda aos grupos II, III, IV e V em folhas A4 separadas.

[8v] **Grupo I - Assinale no seguinte grupo se as frases são verdadeiras ou falsas (uma resposta errada desconta 50% de uma correcta).**

	V	F
1) Em C, o maior valor que podemos armazenar numa variável do tipo `int` é 2^{32}	☐	☐
2) Admita a variável `unsigned int a = 0xFFFFFFFF` em C. Então, a variável `unsigned int b = a + 1` tem o valor zero	☐	☐
3) Admita a variável `short a = 0x0123` e um valor dado por `&a` de `0x200` em C. Então, o valor presente no byte `0x201` é `0x01`	☐	☐
4) Em C, ao truncarmos uma variável do tipo `int` que armazena um valor positivo para um `short` podemos ficar com um valor negativo	☐	☐
5) Em C, o operador lógico || (OR) termina a avaliação da expressão se encontrar uma condição que seja avaliada como verdade	☐	☐
6) Em C, as divisões de inteiros usando deslocamentos podem exigir a alteração do dividendo para que o arredondamento seja correto	☐	☐
7) Admita que `int *ptr` armazena endereço inicial de um vetor de `int`. Logo, `(short*)ptr + 8` aponta para o quinto elemento	☐	☐
8) Admita que declara a variável `int x` como variável global em C. Logo, os 4 bytes são reservados na *stack*	☐	☐
9) Em C, é possível retornar como valor de saída de uma função o endereço de um vetor `short *vec = (short*)malloc(20)`	☐	☐
10) Em Assembly, a instrução `popl %eax` é equivalente a executar `movl (%esp),%eax` seguido de `addl $4, %esp`	☐	☐
11) Em Assembly, a instrução `leal (%eax,%eax,4),%eax` pode ser usada para multiplicar por 5 o valor em `%eax`	☐	☐
12) Se `%ecx` for 4 e `%esi` o endereço inicial de um vetor de `short`, `movw 6(%esi,%ecx,2),%ax` coloca em `%ax` o oitavo elemento.	☐	☐
13) Em IA32, a *stack* é usada para suportar a passagem do valor de retorno de uma função invocada à função invocadora	☐	☐
14) Admita que o valor de `%esp` é `0x1004`. A execução da instrução `ret` coloca o valor de `%esp` em `0x1000`	☐	☐
15) Em IA32, reservamos espaço para as variáveis locais de uma função subtraindo o número de bytes necessários ao valor atual de `%ebp`	☐	☐
16) Admita a matriz global `int m[4][5]`. Em Assembly, acedemos ao endereço de `m[i]` calculando `m + i*20`	☐	☐
17) Uma estrutura, alinhada de acordo com as regras estudadas, com 1 `char`, 1 `double` e um 1 `char*` (por esta ordem) ocupa 16 bytes	☐	☐
18) O tamanho de uma estrutura é garantidamente menor se indicarmos os seus campos por ordem decrescente do seu tamanho	☐	☐
19) A fragmentação interna dos blocos reservados na *heap* é originada pelas regras de alinhamento e *overhead* da gestão dos blocos	☐	☐
20) A possibilidade de existirem diversas referências para a mesma posição de memória em C dificulta a otimização efetuada pelo compilador	☐	☐

[2v] **Grupo II – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

No seguinte excerto de código em C foram omitidos os valores das constantes `M` e `N`:

```
#define M       /* Número mistério 1 */
#define N       /* Número mistério 2 */

int arith(int x, int y) {
  int result = 0;
  result = x*M + y/N;
  return result;
}
```

Admita agora que a função foi compilada para valores específicos de `M` e `N`. O compilador otimizou a multiplicação e divisão pelas constantes `M` e `N` usando deslocamentos, tal como estudado nas aulas. O código seguinte em C é uma tradução do código Assembly gerado:

```
int optarith(int x, int y) {
  int t = x;
  x <<= 5;
  x -= t;
  if (y < 0) y += 7;
  y >>= 3;
  return x+y;
}
```

Quais os valores de `M` e `N`? **Justifique a sua resposta**

[5v] **Grupo III – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

Considere as seguintes declarações:

```
typedef struct {
    short x;
    int y;
} structA;
```

```
typedef struct {
    structA a;
    structA *b;
    int x;
    char c;
    int y;
    char e[3];
    short z;
} structB;
```

[1.5v] **a)** Indique o alinhamento dos campos de uma estrutura do tipo `structB`. Indique claramente, para cada campo, o seu endereço, bem como as partes alocadas mas não usadas para satisfazer as restrições de alinhamento. Indique o tamanho total da estrutura. **Admita que a estrutura está colocada a partir do endereço `0x100`.**

[1.5v] **b)** Se definirmos os campos da estrutura `structB` por outra ordem é possível reduzir o número de bytes necessários para o seu armazenamento? **Justifique a sua resposta.** Indique, em caso afirmativo, qual a ordem dos campos que garante o menor tamanho, o novo endereço de cada campo e das partes alocadas mas não usadas, bem como o novo tamanho total da estrutura.

[2v] **c)** Considere as seguintes funções em C, respeitando as declarações iniciais das estruturas:

```
short fun1(structB *s){
  return s->a.x;
}

short* fun2(structB *s){
  return &s->z;
}
```

```
short fun3(structB *s){
  return s->z;
}

short fun4(structB *s){
  return s->b->x;
}
```

Indique a que funções (`fun1`, `fun2`, `fun3` ou `fun4`) correspondem os seguintes excertos de código em Assembly**. Escreva as funções completas na folha A4.**

```
________:
  pushl   %ebp
  movl    %esp,%ebp
  movl    8(%ebp),%eax
  addl    $28,%eax
  popl    %ebp
  ret
________:
  pushl   %ebp
  movl    %esp,%ebp
  movl    8(%ebp),%eax
  movl    8(%eax),%eax
  movswl  (%eax),%eax
  popl    %ebp
  ret
```

```
________:
  pushl   %ebp
  movl    %esp,%ebp
  movl    8(%ebp),%eax
  movl    28(%eax),%eax
  popl    %ebp
  ret
________:
  pushl   %ebp
  movl    %esp,%ebp
  movl    8(%ebp),%eax
  movswl  (%eax),%eax
  popl    %ebp
  ret
```

[2v] **Grupo IV – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

Admita a seguinte função em C que recebe como primeiro parâmetro o endereço de uma *string* `str` e como segundo parâmetro o endereço de um inteiro `hash` no qual a função armazena o resultado.

```
void calcHash(char *str, int *hash){
 int i, j;

 *hash = 0;
 for(i = 0; i < strlen(str); i++){
   j = strlen(str) / 2;
   *hash += secret(str,i) * 32 + j);
 }
}
```

```
int secret(char *str, int pos){
  return str[pos] % 26;
}
```

Apresente uma segunda versão da função `calcHash` em C com a mesma funcionalidade, mas melhor desempenho. Admita que o compilador que é usado não efetua nenhuma otimização. **Indique claramente cada uma das otimizações usadas sob a forma de comentário no código.**

[3v] **Grupo V – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

```
fun:
  pushl %ebp
  movl %esp,%ebp
  movl 16(%ebp),%ecx
  movl 12(%ebp),%eax
  movl 8(%ebp),%edx
  cmpl %ecx,%edx
  jl .L1
.L2:
  addl %edx,%eax
  decl %edx
  cmpl %ecx,%edx
  jge .L2
.L1:
  movl %ebp,%esp
  popl %ebp
  ret
```

Com base no código Assembly à esquerda, preencha os espaços em branco no código correspondente em C. Apenas pode usar as variáveis `x`, `y`, `z`, `i` e `result` nas expressões (*não use nomes de registos!*) (**escreva a função completa na folha A4**).

```
int fun(int x, int y, int z){

  int i, result = __________;

  for(i=____; ____; ____){
    result = ________;
  }

  return ________;
}
```

- Consulta apenas das 2 folhas de consulta (C e Assembly).
- A infracção implica, no mínimo, a ANULAÇÃO da prova.
- Quando omissa a arquitectura, considere Linux/IA32.

Versão: A Nota mínima: **7.5/20 valores** / Duração: 120 minutos

Número:_____ **Nome:**_____

Responda aos grupos II, III, IV e V em folhas A4 separadas.

[8v] **Grupo I - Assinale no seguinte grupo se as frases são verdadeiras ou falsas (uma resposta errada desconta 50% de uma correcta).**

V F

1) Em C, admita a variável "`char x = -1;` ". Logo, o valor armazenado em "`char y = (unsigned)x >> 1;`" é 127. ☐☐

2) Em C, a operação `u << k` tem sempre como resultado `u * 2^k`, para valores inteiros de `u` com ou sem sinal e `0 < k <= 31` ☐☐

3) Em C, admita a variável "`unsigned int x=0x12345678;`" cujo endereço é `0x100`. Logo, o valor presente no byte `0x102` é `0x34` ☐☐

4) Em C, a função `malloc` permite-nos reservar blocos de memória na *stack* em tempo de execução que podem ser depois redimensionados ☐☐

5) Em C, quando a soma aritmética de duas variáveis `u` e `v` do tipo `int` é superior a 2^{31} o valor obtido é equivalente a `u + v - 2^31` ☐☐

6) Em C, a divisão correta de um inteiro negativo `x` por 2^k através de um deslocamento deve ser obtida com "`(x+(1<<k)-1)>>k`" ☐☐

7) Em C, admita um vetor "`int vec[10];`" e um apontador "`short *ptr = (short*)vec`". Então, `ptr + 4` avança para `vec[2]` ☐☐

8) Em C, é correto retornar como valor de saída de uma função o endereço de um bloco de memória reservado na *heap* dentro da função ☐☐

9) Em Assembly, o equivalente a "`*ptr1 = *ptr2`", apontadores do tipo `int*` em C, pode ser obtido com "`movl (%eax),(%ebx)`" ☐☐

10) Em Assembly, a instrução "`popl %eax`" é equivalente a "`movl (%esp),%eax`" seguido de "`addl $4,%esp`" ☐☐

11) Em Assembly, as operações de multiplicação e divisão de inteiros têm instruções diferentes para valores com e sem sinal ☐☐

12) A adição de dois bytes com sinal com valores `$127` e `$10` deixa as *flags* do registo EFLAGS com os valores ZF=0, SF=1, CF=0, OF=1 ☐☐

13) Em IA32, a *stack* é usada para suportar o retorno do valor de saída de uma função, tal como acontece com o controlo de fluxo ☐☐

14) Em IA32, admita que o valor de `%esp` é `0x1004`. A execução da instrução "`call func`" coloca o valor de `%esp` em `0x1000` ☐☐

15) De acordo com a convenção usada em Linux/IA32, a responsabilidade da salvaguarda e restauro de `%edx` é da função invocadora ☐☐

16) Admita a matriz global `int m[10][3]`. Em Assembly, acedemos ao valor de `m[3][1]` avançando 40 bytes a partir de `m` ☐☐

17) Uma estrutura, alinhada de acordo com as regras estudadas, com 2 `char`, um vetor de 5 `int` e 1 `short` (por esta ordem) ocupa 24 bytes ☐☐

18) O tamanho de uma estrutura sujeita a alinhamento tem de ser múltiplo da menor restrição de alinhamento dos seus campos ☐☐

19) O tamanho de um bloco reservado na *heap* pode ser maior do que o número de bytes passados por parâmetro na função `malloc` ☐☐

20) A invocação de funções introduz *overhead* e limita as possibilidades de otimização dos programas por parte do compilador ☐☐

[2v] **Grupo II – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

Considere o seguinte código em C em que `num_t` é um tipo de dados declarado através de `typedef`:

```
void product(num_t *dest, unsigned int x, num_t y){
   *dest = x * y;
}
```

Admita que o GCC gerou o bloco de código em Assembly descrito ao lado, correspondente ao corpo da função.

```
movl 12(%ebp), %eax
movl 20(%ebp), %ecx
imull %eax, %ecx
mull 16(%ebp)
leal (%ecx, %edx), %edx
movl 8(%ebp), %ecx
movl %eax, (%ecx)
movl %edx, 4(%ecx)
```

[1v] **a)** Qual o tipo de dados de `num_t`? **Justifique a sua resposta.**
[1v] **b)** Descreva o algoritmo usado na multiplicação. **Argumente porque está correto.**

[5v] **Grupo III – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

Considere as seguintes declarações:

```
typedef struct {
  short int a[3];
  char b;
  long long int c;
  int d;
  unionB ub;
  char e;
}structA;
```

```
typedef union {
  int a;
  char b;
  short c;
  long int d;
}unionB;
```

[1.5v] **a)** Indique o alinhamento dos campos de uma estrutura do tipo `structA`. Indique claramente, para cada campo, o seu endereço, bem como as partes alocadas mas não usadas para satisfazer as restrições de alinhamento. Indique o tamanho total da estrutura. **Admita que a estrutura está colocada a partir do endereço `0x100`.**

[1.5v] **b)** Se definirmos os campos da estrutura `structA` por outra ordem é possível reduzir o número de bytes necessários para o seu armazenamento? **Justifique a sua resposta** indicando, em caso afirmativo, qual a ordem dos campos que garante o menor tamanho, o novo endereço de cada campo e das partes alocadas mas não usadas, bem como o novo tamanho total da estrutura.

[2v] **c)** Considere o seguinte fragmento de código em C:

```
char return_unionB_b(structA **matrix, int i, int j){
  return matrix[i][j].ub.b;
}
```

Reescreva a função `return_unionB_b` em Assembly. Na sua resolução tenha em consideração que `matrix` é uma matriz de estruturas criada dinamicamente na `heap` através da função `malloc`. Respeite a declaração inicial da estrutura usada na alínea a. **Comente o seu código.**

[3v] **Grupo IV – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

Considere o seguinte código em Assembly:

```
func:
  pushl %ebp
  movl %esp,%ebp
  movl 8(%ebp), %eax
  movl 12(%ebp), %ecx
  movl 16(%ebp), %edx
  cmpl %ecx, %edx
  jle .L2
  movl 8(%eax), %eax

.L1:
  shrw $2, 4(%eax)
  incl %ecx
  cmpl %ecx, %edx
  jg .L1

.L2:
  movl %esp, %ebp
  popl %ebp
  ret
```

Com base no código Assembly à esquerda, preencha os espaços em branco no código correspondente em C. (**escreva a estrutura e a função completas na folha A4**).

```
typedef struct node{
  int    x;
  ______ y;
  struct node *next;
  struct node *prev;
}node_t;


void func(______, int a, int b){
  node_t *m;

  m = ____________;

  while(________){
    m->y = __________;
    a++;
  }
}
```

[2v] **Grupo V – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

Admita o seguinte excerto de código em C. A função `calc_matrix` recebe como primeiro parâmetro o endereço de uma estrutura, onde está armazenado o endereço de uma matriz dinâmica de inteiros e o seu tamanho atual, e como segundo parâmetro o endereço de um inteiro `res` no qual a função armazena o resultado computado.

```
typedef struct{
  int lines;
  int columns;
  int **m;
}data_t;

void calc_matrix(data_t *matrix, int *res){
 int i, j;

 *res = 0;

 for(j= 0; j < num_columns(matrix); j++){
  for(i = 0; i< num_lines(matrix); i++){
   *res += 16*i + get_element(matrix,i,j);
  }
 }
}
```

```
int num_lines(data_t *matrix){
  return matrix->lines;
}

int num_columns(data_t *matrix){
  return matrix->columns;
}

int get_element(data_t *matrix, int i, int j){
  return matrix->m[i][j];
}
```

Apresente uma segunda versão da função `calc_matrix` em C com a mesma funcionalidade, mas melhor desempenho. Admita que o compilador que é usado não efetua nenhuma otimização. **Indique claramente cada uma das otimizações usadas sob a forma de comentário no código.**

isep Instituto Superior de Engenharia do Porto

DEI / Licenciatura em Engenharia Informática
Arquitectura de Computadores
Exame Época Recurso – Fevereiro 2018

- Consulta apenas das 2 folhas de consulta (C e Assembly).
- A infracção implica, no mínimo, a ANULAÇÃO da prova.
- Quando omissa a arquitectura, considere Linux/IA32.

Versão: A Nota mínima: **7.5/20 valores** / Duração: 120 minutos

Número:_____ **Nome:**_____

Responda aos grupos II, III, IV e V em folhas A4 separadas.

[8v] **Grupo I - Assinale no seguinte grupo se as frases são verdadeiras ou falsas (uma resposta errada desconta 50% de uma correcta).**

	V	F
1) Em C, o valor de um apontador é o endereço do primeiro byte do bloco de memória para o qual aponta	☐	☐
2) Em C, o tipo de dados do apontador determina o espaço em memória necessário para o armazenar	☐	☐
3) Em C, admita a variável "`unsigned char x;`". O maior valor positivo que podemos armazenar em `x` é 2^7 - 1	☐	☐
4) Em C, um *cast* para `char` de uma variável do tipo `unsigned short` com um valor positivo pode resultar num valor negativo	☐	☐
5) Em C, as operações aritméticas com qualquer tipo de dados para valores inteiros seguem as regras da aritmética modular	☐	☐
6) Em C, admita as variáveis "`int x=0xABCD;`" e "`char *ptr=&x`". Logo, "`printf("%hhX",*(ptr+1));`" imprime o valor `0xCD`	☐	☐
7) Em C, "`x >> 2`" aplica um deslocamento lógico para a direita se `x` for do tipo `unsigned int` e um aritmético se `x` for do tipo `int`	☐	☐
8) Em C, admita o vetor "`short vec[5];`". A função `realloc` permite alterar o tamanho de `vec` para armazenar mais elementos	☐	☐
9) Em Assembly, qualquer que seja o valor armazenado em `%eax`, o resultado de "`sall $4,%eax`" e "`shll $4,%eax`" é o mesmo	☐	☐
10) Em Assembly, depois do prólogo de uma função, o valor antigo de `%ebp` pode ser encontrado em `(%esp)`	☐	☐
11) Em Assembly, reservar 8 bytes para variáveis locais de uma função pode ser conseguido com "`addl $8,%esp`"	☐	☐
12) Em IA32, a *stack* é usada para suportar o retorno do valor de saída de uma função, tal como acontece com o controlo de fluxo	☐	☐
13) Em IA32, a execução da instrução `ret` não altera o valor de qualquer registo	☐	☐
14) Em IA32, o resultado da instrução "`jmp func`" depende do valor dos bits do registo EFLAGS	☐	☐
15) De acordo com a convenção usada em Linux/IA32, uma função pode usar `%edx` sem a necessidade de o salvaguardar e restaurar	☐	☐
16) Admita uma matriz de inteiros alocada na *heap* dentro de uma função. O seu espaço é automaticamente libertado no fim da função	☐	☐
17) As restrições de alinhamento em memória contribuem para a possível fragmentação interna de um bloco reservado na *heap*	☐	☐
18) O tamanho de uma estrutura sujeita a alinhamento é sempre o mesmo em IA32 e x86-64, independentemente dos seus campos	☐	☐
19) O endereço inicial de uma estrutura sujeita a alinhamento depende dos tipos de dados dos seus campos	☐	☐
20) A técnica de otimização de programas que move código para fora de um ciclo é denominada "*loop unrolling*"	☐	☐

[2v] **Grupo II – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

Foi contratado para implementar um conjunto de funções que operam com um tipo de dados que agrupa **4 bytes com sinal num único valor de 32 bits sem sinal**:

```
typedef unsigned int packed_t;
```

O seu antecessor (que foi despedido por incompetência) produziu a função ao lado para extrair o byte indicado e expandi-lo para um valor de 32 bits com sinal. Os bytes são numerados de 0 (menos significativo) a 3 (mais significativo).

```
/* Extract byte from word. Return as signed
integer */

int xbyte(packed_t word, char byte_num){
   return (word >> (byte_num << 3)) & 0xFF;
}
```

[1v] **a)** Qual o problema da função desenvolvida? **Justifique a sua resposta.**

[1v] **b)** Escreva uma versão correta da função. **Comente o seu código, descrevendo a abordagem seguida.**

[5v] **Grupo III – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

Considere as seguintes declarações:

```
typedef struct {
  char a;
  char b[2];
  int c;
  unsigned short d;
  structB *e;
  char f;
}structA;
```

```
typedef struct {
  int a;
  char b;
  short c;
  long int d;
}structB;
```

```
typedef union{
  int a;
  int b;
  int c[2];
  unsigned char d[8];
}unionC;
```

[1.5v] **a)** Indique o alinhamento dos campos de uma estrutura do tipo `structA`. Indique claramente, para cada campo, o seu endereço, bem como as partes alocadas mas não usadas para satisfazer as restrições de alinhamento. Indique o tamanho total da estrutura. **Admita que a estrutura está colocada a partir do endereço `0x100`.**

[1.5v] **b)** Considerando o seguinte fragmento de código em C, que valores irão ser impressos? **Justifique a sua resposta.**

```
unionC u;
u.a = 0x01020304;
u.b = 0x05060708;

u.d[4] = 0x0A;   u.d[5] = 0x0B;
u.d[6] = 0x0C;   u.d[7] = 0x0D;
```

```
printf("%d\n",sizeof(u));
printf("%X\n",u.c[0]);
printf("%X\n",u.c[1]);
```

[2v] **c)** Considere o seguinte fragmento de código em C:

```
structA matrix[4][5];

void fill_structA_b(int i, int j){
   matrix[i][j].b[0] = matrix[i][j].a;
   matrix[i][j].b[1] = matrix[i][j].f;
}
```

Reescreva a função `fill_structA_b` em Assembly. Na sua resolução tenha em consideração que `matrix` é uma matriz de estruturas global definida estaticamente. **Comente o seu código.**

[3v] **Grupo IV – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

[1v] **a)** Mostre como os seguintes valores são armazenados em memória em IA32. Apenas preencha os bytes que se aplicam (isto é, se um valor não usar todos os bytes, assegure-se que não preenche nada nos bytes não ocupados). Assuma que os valores têm como endereço 0x100.

Valor	0x100	0x101	0x102	0x103
"ABC"				
0xABCD				

[1v] **b)** Use os seguintes valores iniciais em memória e nos registos da arquitetura para responder a cada uma das questões (isto é, cada questão não é afetada pela execução das instruções anteriores).

Endereço	Valor
0x100	0xFF
0x104	0xAB
0x108	0x13
0x10C	0x11

Registo	Valor
%eax	0x100
%ecx	0x1
%edx	0x3
%ebx	0x4

A. Qual o novo valor de `%eax` após "`movl 0x100,%eax`"? _____

B. Qual o novo valor de `%ecx` após "`movl (%eax,%edx,4),%ecx`"? _____

C. Qual o endereço que é alterado com "`subl %edx,4(%eax)`"? _____

D. Qual o novo valor de `%ebx` após "`leal 0x100(,%ebx,2),%ebx`"? _____

[1v] **c)** Converta a instrução "`leal 0x4(%eax,%ecx,8),%ebx`" num conjunto equivalente de instruções Assembly. **Assegure-se de que o código convertido e a instrução `leal` original deixam os registos `%eax`, `%ecx` e `%ebx` com os mesmos valores.** Na sua solução não pode usar outros registos para além destes 3.

[2v] **Grupo V – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

Um grupo de alunos do DEI decidiu participar num concurso de programação e há ainda uma vaga na equipa. Para ser admitido terá de desenvolver a versão mais otimizada do cálculo do fatorial de um número. A sua primeira abordagem foi reescrever a versão recursiva de forma iterativa, obtendo a função descrita em `fact_iter`:

```
int fact_iter(int n){
 int i, result = 1;

 for(i = n; i > 1; i--){
   result = result * i;
 }
 return result;
}
```

```
int fact_unroll2(int n){
 int i, result = 1;

 for(i = n; i > 0; i-=2){
  result = (result*i)*(i-1);
 }
 return result;
}
```

Ciclos de relógio Por Elem/Op (CPE)

Operação (inteiros)	Latência Operação	Débito Operações
Adição	1	1
Multiplicação	4	1
Divisão	36	36

[1v] **a)** Ao testar a performance da versão `fact_iter` num processador superescalar com as características da tabela acima verificou que reduziu o número de CPE de 64, obtidos pela versão recursiva, para cerca de 4. Explique o aumento de performance e o valor encontrado.

[1v] **b)** A seguir, decidiu aplicar a técnica de "*loop unrolling*" para continuar a melhorar a performance da função num processador superescalar e chegou à versão descrita em `fact_unroll2`. No entanto, ficou desapontado quando verificou que a performance não é melhor do que a versão `fact_iter`. Um colega sugeriu-lhe que alterasse a linha dentro do ciclo para "`result = result * (i * (i-1));`" e a performance da sua função passou para um CPE de 2.5. Como explica esta melhoria na performance?

- Autorizada apenas a consulta da folha de consulta oficial.
- A infração implica, no mínimo, a **ANULAÇÃO** da prova.
- Quando omissa a arquitectura, considere Linux/IA32.

Versão: A Nota mínima: **7.5/20 valores** / Duração: 120 minutos

Número:_____ **Nome:**_____

Responda aos grupos II, III, IV e V em folhas A4 separadas.

[8v] **Grupo I - Assinale no seguinte grupo se as frases são verdadeiras ou falsas (uma resposta errada desconta 50% de uma correcta).**

V F

1) Em C, admita a variável "`unsigned short x=0xABFF;`". O valor armazenado em "`char y = (char)x | 0x0;`" é -1 ☐☐

2) Em C, qualquer que seja o valor atribuído à variável "`int y;`", a atribuição "`int x = -y;`" é equivalente a "`int x = ~y + 1;`" ☐☐

3) Em C, admita as variáveis "`unsigned char x,y;`". Se `x` for par, "`y = (x|1) & 0xFF;`" atribui um valor ímpar a `y` ☐☐

4) Em C, a função `realloc` permite-nos redimensionar em tempo de execução blocos de memória reservados na *stack* ☐☐

5) Em C, admita a variável "`short x;`" à qual é atribuída um valor negativo. Logo, "`short y = x*2;`" será sempre menor do que zero ☐☐

6) Em C, o maior valor positivo que é possível armazenar na variável "`unsigned short x;`" é $2^{16} - 1$ ☐☐

7) Em C, admita que "`int *ptr=(int*)malloc(20);`" é declarado numa função. É correto terminar a função com "`return ptr;`" ☐☐

8) O Assembler é o programa que recebe como *input* código escrito numa linguagem de alto nível como o C e o traduz para Assembly ☐☐

9) Em IA32, a instrução "`leal (%eax,%eax,4),%eax`" multiplica por 5 o valor presente em `%eax` ☐☐

10) Em IA32, a instrução "`idivb %cl`" assume que o dividendo se encontra em `%ax`, deixando o quociente em `%al` e o resto em `%ah` ☐☐

11) Em IA32, se pretendermos dividir valores inteiros sem sinal podemos usar as instruções `div` ou `idiv` obtendo sempre o mesmo resultado ☐☐

12) Em IA32, "`shrl %eax`" seguido de "`jnc xpto`" permite saltar para a linha `xpto` se o valor presente em `%eax` for par. ☐☐

13) Admita o vetor global "`short a[10];`" em C. "`movl $2,%ecx`" seguido de "`movw a(,%ecx,4),%ax`" coloca `a[4]` em `%ax` ☐☐

14) Em IA32, admita que o valor de `%esp` é `0x1004`. A execução da instrução "`call func`" coloca o valor de `%esp` em `0x1008` ☐☐

15) De acordo com a convenção usada em Linux/IA32, a responsabilidade da salvaguarda e restauro de `%ebp` é apenas da função invocadora ☐☐

16) Admita que M e N são valores grandes. "`for(j=0;j<N;j++)for(i=0;i<M;i++)sum+=m[i][j];`" terá a melhor performance ☐☐

17) O tempo de acesso a um setor num disco é dominado pelo tempo de pesquisa e latência de rotação da cabeça de leitura ☐☐

18) Na hierarquia de memória à medida que nos afastamos do processador, a capacidade de armazenamento aumenta bem como a performance ☐☐

19) Diz-se que um bloco de código possui boa localidade espacial quando não existem intervalos de alinhamento entre as variáveis usadas ☐☐

20) A invocação de funções introduz *overhead* e limita as possibilidades de otimização dos programas por parte do compilador ☐☐

[2v] **Grupo II – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

[1v] **a)** Considere o seguinte excerto de código. Qual dos outputs indicados do lado direito é produzido? **Justifique a sua resposta.**

```
int x = 0x15213F10 >> 4;
char y = (char)x;
unsigned char z = (unsigned char)x;
printf("y = %d, z = %u", y, z);
```

(a) y = -241, z = 15
(b) y= -15, z = 241
(c) y= -241, z = 241
(d) y = -15, z = 15

[1v] **b)** Considere o seguinte excerto de código. Qual dos outputs indicados do lado direito é produzido, assumindo que o utilizador insere corretamente um inteiro? **Justifique a sua resposta.**

```
int x;
printf("Please input an integer:");
scanf("%d",&x);
printf("%d", (!!x)<<31);
```

(a) 0, qualquer que seja o inteiro lido do teclado
(b) `INT_MIN`, qualquer que seja o inteiro lido do teclado
(c) 0 ou `INT_MIN`, dependendo do inteiro lido do teclado
(d) Um valor dependente do inteiro lido do teclado

[5v] **Grupo III – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

Considere as seguintes declarações:

```
struct s1{
   char a[3];
   union u1 *b;
   int c;
};
```

```
struct s2{
   struct s1 d;
   struct s1 *e;
   struct s2 *f;
   long long int g;
   short h[3];
};
```

```
union u1 {
  int i;
  struct s2 j;
  struct s1 *k;
};
```

[1.5v] **a)** Indique o alinhamento dos campos de uma estrutura do tipo `struct s2`. Indique claramente, para cada campo, o seu endereço, bem como as partes alocadas mas não usadas para satisfazer as restrições de alinhamento. Indique o tamanho total da estrutura. **Admita que a estrutura está colocada a partir do endereço `0x100`.**

[1.5v] **b)** Se definirmos os campos da estrutura `struct s1` por outra ordem é possível reduzir o número de bytes necessários para o seu armazenamento? **Justifique a sua resposta** indicando, em caso afirmativo, qual a ordem dos campos que garante o menor tamanho, o novo endereço de cada campo e das partes alocadas mas não usadas, bem como o novo tamanho total da estrutura.

[2v] **c)** Considere o seguinte fragmento de código em C:

```
int return_u1_i(struct s2 matrix[10][20], int i, int j){
  return matrix[i][j].e->b->i;
}
```

Reescreva a função `return_u1_i` em Assembly. Na sua resolução tenha em consideração que `matrix` é uma matriz de estruturas estática. Respeite a declaração inicial da estrutura usada na alínea a. **Comente o seu código.**

[3v] **Grupo IV – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

Considerando as declarações do exercício anterior, preencha os espaços em branco nas funções à direita, considerando o código correspondente à esquerda. (**escreva as funções completas na folha A4**)

```
proc1:
  pushl %ebp
  movl %esp,%ebp
  movl 8(%ebp),%eax
  movl 4(%eax),%eax
  movw 32(%eax),%ax
  movl %ebp, %esp
  popl %ebp
  ret
```

```
short proc1(struct s1 *x){
   return x->__________;
}
```

```
int proc2(struct s2 *x){
   return x->e->c;
}
```

```
proc2:
   pushl %ebp
   movl %esp,%ebp
   ____________________
   ____________________
   ____________________
   movl %ebp,%esp
   popl %ebp
   ret
```

```
proc3:
   pushl %ebp
   movl %esp,%ebp
   movl 8(%ebp),%eax
   movl (%eax),%eax
   movl 8(%eax),%eax
   movl %ebp,%esp
   popl %ebp
   ret
```

```
int proc3(union u1 *x){
   return x->_________;
}
```

[2v] **Grupo V – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

Admita o seguinte excerto de código em C para manipulação de uma lista simplesmente ligada não circular com medições de temperaturas em graus Fahrenheit. Assuma que o campo `next` com o valor `NULL` indica o fim da lista e que existe uma função `int length(List *p)` para determinar o número de elementos da lista. A função `count_positives_celsius` recebe em `p` o endereço de uma lista e armazena no endereço dado por `k` o número de elementos dessa lista cujas temperaturas em Celsius são maiores do que 0.

```
typedef struct LIST{
  int data;
  struct LIST *next;
}List;
```

```
void count_positives_celsius(List *p, int *k){
   int i;
   *k = 0;
   for (i = 0; i < length(p); i++){
      if ((p->data-32)*5/8 > 0)
         (*k)++;
      p = p->next;
   }
}
```

Apresente uma segunda versão da função `count_positives_celsius` em C com a mesma funcionalidade, mas melhor desempenho. Admita que o compilador que é usado não efetua nenhuma otimização. **Indique claramente cada uma das otimizações usadas sob a forma de comentário no código.**

- Autorizada apenas a consulta da folha de consulta oficial.
- A infração implica, no mínimo, a **ANULAÇÃO** da prova.
- Quando omissa a arquitectura, considere Linux/IA32.

Versão: A Nota mínima: **7.5/20 valores** / Duração: 120 minutos

Número:_____ **Nome:**_____

Responda aos grupos II, III, IV e V em folhas A4 separadas.

[8v] **Grupo I - Assinale no seguinte grupo se as frases são verdadeiras ou falsas (uma resposta errada desconta 50% de uma correcta).**

V F

1) Em C, admita "`unsigned short x=0xABFF;`" e "`char *p=(char)&x;`". "`printf("%hhx",*p);`" imprime o valor "`AB`"..... ☐☐

2) Em C, admita "`int x = 0xCFC7;`" e "`short y = (short)x;`". Logo, "`int z = (int)y;`" atribui o valor `0xCFC7` a `z` ..... ☐☐

3) Em C, admita "`short x = 0x1234;`". Logo, "`unsigned short y = (x && 0x00FF);`" atribui o valor `0x34` a `y`..... ☐☐

4) Em C, usamos memória dinâmica porque a *heap* é uma zona de memória com um tempo de acesso menor do que a *stack*..... ☐☐

5) Em C, admita a variável "`int x;`" à qual é atribuída um valor positivo. Logo, "`short y=(short)x*2;`" será sempre positivo..... ☐☐

6) Em C, o maior valor positivo que é possível armazenar na variável "`char x;`" é $2^8 - 1$ ..... ☐☐

7) Em C, admita que `ptr` é uma variável do tipo `char*`. Então, a expressão `(short*)ptr + 7` avança 14 bytes na memória..... ☐☐

8) O compilador é o programa que recebe como *input* código escrito numa linguagem de alto nível como o C e o traduz para Assembly ..... ☐☐

9) Em IA32, é possível usar a instrução "`leal (%eax,%eax,49),%eax`" para multiplicar por 50 o valor presente em `%eax` ..... ☐☐

10) Em IA32, reservamos espaço para as variáveis locais de uma função somando o número de bytes necessários ao valor atual de `%esp` ..... ☐☐

11) Em IA32, é possível retornar de um bloco de código com a instrução `ret` quando a sua invocação/salto foi efetuada com a instrução `jmp` ... ☐☐

12) Em IA32, são usados registos para suportar a passagem do valor de retorno de uma função invocada à função invocadora ..... ☐☐

13) Admita o vetor global "`int a[10];`" em C. "`movl $2,%ecx`" seguido de "`movl a(,%ecx,4),%eax`" coloca `a[4]` em `%eax`..... ☐☐

14) Em IA32, podemos substituir "`popl %eax`" por "`movl (%esp),%eax`" seguido de "`addl $4, %esp`" ..... ☐☐

15) De acordo com a convenção usada em Linux/IA32, a responsabilidade da salvaguarda e restauro de `%ebp` é apenas da função invocadora .... ☐☐

16) O tamanho de uma *union* é o menor possível se declararmos os seus campos por ordem decrescente de tamanho do tipo de dados..... ☐☐

17) O tempo de acesso a um setor num disco é dominado pelo tempo de pesquisa e latência de rotação da cabeça de leitura..... ☐☐

18) Na hierarquia de memória à medida que nos afastamos do processador a capacidade de armazenamento aumenta e diminui a performance.... ☐☐

19) A localidade espacial indica a probabilidade de acesso a dados e instruções em endereços próximos daqueles acedidos recentemente..... ☐☐

20) Uma das otimizações efetuadas pelos compiladores é a substituição da invocação de uma função pelo seu código ..... ☐☐

[2v] **Grupo II – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

[1v] **a)** Escreva em C o código da função `unsigned int replace_byte(unsigned int x, int i, unsigned char b)` que deverá retornar um valor em que o byte `i` do parâmetro `x` foi substituído pelo byte `b`. Os bytes são numerados de 0 (menos significativo) a 3 (mais significativo). Alguns exemplos de como a função deve operar:

```
replace_byte(0x12345678, 2, 0xAB) --> 0x12AB5678
replace_byte(0x12345678, 0, 0xAB) --> 0x123456AB
```

[1v] **b)** Preencha a seguinte tabela mostrando o efeito das instruções seguintes, quer em termos de localização dos resultados (registo ou endereço de memória), quer dos respetivos valores. (**cada instrução não é afetada pela execução das instruções anteriores**)

Endereço	Valor
0x100	0xFF
0x104	0xAB
0x108	0x13
0x10C	0x11

Registo	Valor
%eax	0x100
%ecx	0x1
%edx	0x3
%ebx	0x4

Instrução	Destino	Valor
`addl %ecx,(%eax)`		
`subl %edx,4(%eax)`		
`shll $4,(%eax,%edx,4)`		
`incl 8(%eax)`		
`subl %edx,%eax`		

[5v] **Grupo III – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

Considere as seguintes declarações:

```
struct s1{
   int *a;
   struct s2 b;
   struct s1 *c;
   char d;
};
```

```
struct s2{
   int d;
   union u1 e;
   struct s1 *f;
   short g[3];
};
```

```
union u1 {
  int h;
  char i[3];
  struct s2 *j;
};
```

[1.5v] **a)** Indique o alinhamento dos campos de uma estrutura do tipo `struct s1`. Indique claramente, para cada campo, o seu endereço, bem como as partes alocadas mas não usadas para satisfazer as restrições de alinhamento. Indique o tamanho total da estrutura. **Admita que a estrutura está colocada a partir do endereço `0x100`.**

[1.5v] **b)** Se definirmos os campos da estrutura `struct s2` por outra ordem é possível reduzir o número de bytes necessários para o seu armazenamento? **Justifique a sua resposta** indicando, em caso afirmativo, qual a ordem dos campos que garante o menor tamanho, o novo endereço de cada campo e das partes alocadas mas não usadas, bem como o novo tamanho total da estrutura.

[2v] **c)** Considere a seguinte função em C:

```
void s1_init(struct s1 *p){
  p->b.d = _________;
  p->a   = _________;
  p->c   = _________;
}
```

O GCC gerou o código Assembly ao lado para `s1_init`. Com base nesse código, preencha as expressões em falta no código C. **Comente o seu código.**

```
s1_init:
  pushl %ebp
  movl %esp,%ebp
  movl 8(%ebp),%eax
  movl 8(%eax),%edx
  movl %edx,4(%eax)
  leal 4(%eax),%edx
  movl %edx,(%eax)
  movl 12(%eax),%edx
  movl %edx,24(%eax)
  movl %ebp,%esp
  popl %ebp
  ret
```

[3v] **Grupo IV – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

```
proc1:
  pushl %ebp
  movl %esp,%ebp
  movl 8(%ebp),%eax
  movl 12(%ebp),%edx
  shrl %eax
  jnc .L2
  movl 8(%ebp),%eax
  cmpl %edx,%eax
  jle .L3
  imull %edx,%eax
  jmp .L4
.L3:
  leal (%edx,%eax),%eax
  jmp .L4
.L2:
  movl 8(%ebp),%eax
  cmpl $20,%eax
  jg .L5
  leal (%eax,%edx,4),%eax
  jmp .L4
.L5
  subl %edx,%eax
.L4:
  movl %ebp,%esp
  popl %ebp
  ret
```

Considerando o código Assembly à esquerda, preencha os espaços em branco no código correspondente em C. (**escreva a função completa na folha A4**)

```
int proc1(int x, int y){
  int val = __________;

  if(____________){
    if(__________){
      val = _______;
    }else{
      val = _______;
    }
  }else{
   if(___________){
      val = ________;
    }
    else{
     val = ________;
    }
  }

  return val;
}
```

[2v] **Grupo V – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

Considere a seguinte função em C que permite multiplicar duas matrizes de inteiros. Apresente uma nova versão da função `matrix_mult` com a mesma funcionalidade, mas melhor desempenho. Admita que o compilador que é usado não efetua nenhuma otimização. Considere que as matrizes são quadradas. A constante `N` indica o número de linhas e colunas das matrizes. **Indique claramente cada uma das otimizações usadas sob a forma de comentário no código.**

```
void matrix_mult(int a[N][N], int b[N][N], int c[N][N]){
   int i,j,k;

   for (i = 0; i < N; i++){
      for (j = 0; j < N; j++){
         c[i][j] = 0;
         for (k = 0; k < N; k++)
            c[i][j]+=a[i][k]*b[k][j];
      }
   }
}
```

Versão: A Nota mínima: **7.5/20 valores** / Duração: 120 minutos

Número:_____ **Nome:**_____

Responda aos grupos II, III, IV e V em folhas A4 separadas. O grupo I deve ser respondido nesta folha.

[8v] **Grupo I - Assinale no seguinte grupo se as frases são verdadeiras ou falsas (uma resposta errada desconta 50% de uma correcta).**

V F

1) Em C, admita a variável "`short x=-1;`". A atribuição "`unsigned int y=x;`" primeiro altera o sinal de `x` e depois o tamanho ..... ☐☐

2) Em C, o apontador "`int *ptr;`" declarado na função `main()` é alocado na *heap* ..... ☐☐

3) Em C, admita as variáveis "`unsigned char a=0;`" e "`short b=-1;`". A comparação "`if (b<a)`" é verdadeira ..... ☐☐

4) Em C, a função `realloc` permite-nos redimensionar blocos de memória reservados com a função `malloc()` mas não com `calloc()` .... ☐☐

5) Em C, as operações aritméticas com tipos inteiros seguem as regras da aritmética modular, como consequência do seu número finito de bits .. ☐☐

6) Em C, para que o tamanho de uma *union* seja o menor possível, devemos declarar os seus campos por ordem decrescente de tamanho ..... ☐☐

7) Em C, admita as variáveis "`char str[30];`" e "`int* ptr=str;`". Logo, "`ptr=ptr+2;`" avança para o nono elemento de `str` ..... ☐☐

8) O compilador é o programa que recebe como *input* código escrito numa linguagem de alto nível como o C e o traduz para Assembly ..... ☐☐

9) Em IA32, a instrução "`call func`" não altera o estado atual da *stack*, apenas o valor do registo `%eip` com o endereço da etiqueta `func` .... ☐☐

10) Em IA32, a instrução "`idivw %cx`" assume que o dividendo se encontra em `%eax`, deixando o quociente em `%ax` e o resto em `%dx` ..... ☐☐

11) Em IA32, a instrução `adc` só permite adicionar aos seus operandos o valor da *flag* de *carry* quando aplicada a valores com sinal ..... ☐☐

12) Em IA32, "`testl $-1,%ecx`" seguido de "`jz xpto`" permite saltar para a etiqueta `xpto` se o valor de `%ecx` for zero. ..... ☐☐

13) Admita o vetor global "`int a[5];`" em C. "`movl $2,%ecx`" seguido de "`movl a(,%ecx,8),%eax`" coloca `a[4]` em `%eax` ..... ☐☐

14) Em IA32, é possível usar "`leal (%edx,%ecx,4), %eax`" para ler um valor de 4 bytes da memória e colocá-lo em `%eax` ..... ☐☐

15) Em IA32, a instrução "`pushl %eax`" é equivalente a "`movl (%esp),%eax`" seguido de "`subl $4,%esp`" ..... ☐☐

16) Admita a matriz dinâmica "`int **m`", com 10 elementos por linha. Em IA32, acedemos a `m[2][3]` avançando 92 bytes a partir de `m` ..... ☐☐

17) O bloco de código "`for(i=0;i<N;i++)for(j=0;j<M;j++)sum+=m[i][j];`" exibe boa localidade espacial mas não temporal .... ☐☐

18) A fragmentação da *heap* pode impedir que a função `realloc()` consiga redimensionar um bloco existente para um tamanho menor ..... ☐☐

19) Na hierarquia de memória, à medida que nos afastamos do CPU abdicamos da performance em favor do custo por byte ..... ☐☐

20) Uma das otimizações efetuadas pelos compiladores de C é substituição da invocação de uma função pelo seu código ..... ☐☐

[2v] **Grupo II – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

[1v] **a)** Foi-lhe dada a tarefa de escrever código que multiplique a variável `int x` por vários valores constantes `K` diferentes. Para que o seu código seja eficiente, apenas deve usar as operações +, − e <<. Assuma os seguintes valores de `K` e escreva as expressões em C que realizem a multiplicação pretendida, usando no máximo até três operações:

(a) `K` = 17
(b) `K` = -7
(c) `K` = 30
(d) `K` = -56

[1v] **b)** Implemente em C a função `int is_big_endian()` que deve retornar 1 quando compilada e executada numa arquitetura *big-endian* ou 0, caso seja compilada e executada numa arquitetura *little-endian*. Deve ser possível compilar e executar a sua função independentemente do número de bytes usados para representar um inteiro.

[3v] **Grupo III – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

Considere as seguintes declarações:

```
struct s1{
   int a;
   short b[3];
   union u1 c;
   char d;
};
```

```
struct s2{
   char e;
   short f[2];
   long long g;
   struct s3 *h;
};
```

```
struct s3{
   struct s1 *i;
   struct s2 *j;
   struct s3 *k;
};
```

```
union u1{
  char l;
  short m;
  struct s2 n;
  struct s1 *o;
};
```

[1.5v] **a)** Indique o alinhamento dos campos de uma estrutura do tipo `struct s1`. Indique claramente, para cada campo, o seu endereço, bem como as partes alocadas, mas não usadas, para satisfazer as restrições de alinhamento. Indique o tamanho total da estrutura. **Admita que a estrutura está colocada a partir do endereço `0x100`.**

[1.5v] **b)** Se definirmos os campos da estrutura `struct s2` por outra ordem é possível reduzir o número de bytes necessários para o seu armazenamento? **Justifique a sua resposta** indicando, em caso afirmativo, qual a ordem dos campos que garante o menor tamanho, o novo endereço de cada campo e das partes alocadas, mas não usadas, para satisfazer as restrições de alinhamento, bem como o novo tamanho total da estrutura. **Admita que a estrutura está colocada a partir do endereço `0x200`.**

[5v] **Grupo IV – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

[3v] **a)** Considere o seguinte fragmento de código em C:

```
short return_s1_b2(struct s2 **matrix, int i, int j){
  return matrix[i][j].h->i->b[2];
}
```

Reescreva a função `return_s1_b2` em Assembly. Na sua resolução tenha em consideração que `matrix` é uma matriz de estruturas dinâmica. Assuma que os valores de `i` e `j` estão dentro dos limites reservados. Respeite a declaração inicial da estrutura usada na alínea a) do grupo anterior. **Comente o seu código.**

[1v] **b)** Admita o seguinte excerto de código. Indique os valores que irão aparecer no ecrã. **Justifique a sua resposta.**

```
unsigned int data[3] = {0x11223344,0x55667788,0x99AABBCC};
char *p=(char*)data;
printf("0x%x\n",*p);
printf("0x%x\n",*(short*)(p+2));
printf("0x%x\n",*(int*)&data[1]);
```

[1v] **c)** Admita os seguintes endereços e conteúdo da memória:

Endereço	Conteúdo
0x1000	0x1018
0x1004	0x1014
0x1008	0x1010
0x100C	0x100C
0x1010	0x1008
0x1014	0x1004
0x1018	0x1000

Admita que o endereço do vetor `vec` é `0x1000` e são executadas as seguintes instruções:

```
movl $vec, %edx
movl $2, %ecx
leal (%edx, %ecx, 4), %eax
movl (%eax, %ecx, 4), %eax
```

No final, que valor (em hexadecimal) fica em `%eax`?
Justifique a sua resposta.

[2v] **Grupo V – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

O processamento de imagens oferece diversos exemplos de funções que beneficiam com a otimização de código com acessos intensivos à memória. Neste exercício iremos considerar uma função `smooth` que aplica um efeito de "blur" a uma imagem representada por um vetor estático bidimensional com `N*N` pixéis.

```
#define N 25

typedef struct {
  unsigned short red;
  unsigned short green;
  unsigned short blue;
} pixel;
```

```
unsigned short avg(pixel src[N][N], int i, int j){
   unsigned short res;
   res=(src[i+1][j]+ src[i-1][j]+src[i][j-1]+src[i][j+1])/4;
   return res;
}

void smooth(pixel src[N][N], pixel dest[N][N]){
   int i,j;

   for (j = 1; j < N-1; j++)
     for (i = 1; i < N-1; i++)
       dst[i,j].red = avg(src,i,j);
}
```

A função `avg` retorna a média aritmética simples dos pixés vizinhos do pixel na posição `[i,j]`, isto é, os pixéis `[i+1][j]`, `[i-1][j]`, `[i][j-1]` e `[i][j+1]`. Para simplificar, considere apenas o cálculo da média da componente vermelha da cor em cada pixel. Apresente uma segunda versão da função `smooth` em C com a mesma funcionalidade, mas melhor desempenho. Admita que o compilador que é usado não efetua nenhuma otimização. **Indique claramente cada uma das otimizações usadas sob a forma de comentário no código.**

Versão: A Nota mínima: **7.5/20 valores** / Duração: 120 minutos

Número:_____ **Nome:**_____

Responda aos grupos II, III, IV e V em folhas A4 separadas. O grupo I deve ser respondido nesta folha.

[8v] **Grupo I - Assinale no seguinte grupo se as frases são verdadeiras ou falsas (uma resposta errada desconta 50% de uma correcta).**

V F

1) Em C, admita a variável "`unsigned int x;`". A expressão "`!(x&0x1)`" é avaliada em um se `x` for par ☐☐

2) Em C, admita a variável "`char x=-12;`". A atribuição "`short y=(short)x;`" armazena em `y` um valor diferente de `x`. ☐☐

3) Em C, admita as variáveis "`unsigned char *a;`" e "`int b;`". A comparação "`if (sizeof(a) < sizeof(b))`" é verdadeira ☐☐

4) Em C, um bloco de memória alocado com `malloc` durante a execução de uma função é automaticamente libertado no fim desta ☐☐

5) Em C, admita as variáveis "`int x,y;`". A comparação "`x < y`" pode ter um resultado diferente da comparação "`x - y < 0`" ☐☐

6) Em C, admita as variáveis "`int x=0x01234567;`" e "`short *ptr=(short*)&x`". Logo, "`*(ptr+1)`" equivale ao valor `0x4567` ☐☐

7) Em C, admita a variável "`char x=-128;`". A atribuição "`char y=-x;`" armazena o valor `128` em `y` ☐☐

8) O tamanho efetivo de um bloco de memória reservado com `malloc` pode ser maior do que o número de bytes passados por parâmetro ☐☐

9) Em IA32, a instrução "`cmpl %eax,%ebx`" armazena o resultado da comparação em `%ebx` e nos bits do registo EFLAGS ☐☐

10) Em IA32, as operações de multiplicação e divisão de inteiros exigem instruções distintas para valores com e sem sinal ☐☐

11) Em IA32, numa função, após o prólogo estudado nas aulas, o seu endereço de retorno pode ser encontrado em `4(%ebp)` ☐☐

12) Em IA32, "`subl $12,%esp`" permite remover da *stack* os três parâmetros inteiros de uma função invocada na linha anterior com `call` ☐☐

13) Admita o vetor global "`short a[5];`" em C. "`movl $3,%ecx`" seguido de "`movw a+2(,%ecx,2),%ax`" coloca `a[4]` em `%ax` ☐☐

14) Em IA32, é possível usar "`shll $3, %eax`" seguido de "`negl %eax`" para multiplicar por -8 o valor de `%eax` ☐☐

15) Em IA32, a instrução "`pushl %eax`" é equivalente a "`subl $4,%esp`" seguido de "`movl (%esp),%eax`" ☐☐

16) O bloco de código "`for(j=0;j<N;j++)for(i=0;i<M;i++)sum+=m[j][i];`" exibe boa localidade espacial ☐☐

17) O sistema operativo executa periodicamente uma desfragmentação da *heap* para melhorar o desempenho dos programas em C ☐☐

18) A fragmentação externa dos blocos reservados na *heap* é consequência das regras de alinhamento e *overhead* da gestão dos blocos ☐☐

19) Na hierarquia de memória, à medida que nos afastamos do CPU temos maior performance e menor custo por byte ☐☐

20) Uma das otimizações efetuadas pelos compiladores de C é a alocação de variáveis locais aos registos disponíveis da arquitetura ☐☐

[2v] **Grupo II – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

[1v] **a)** Escreva uma expressão em C que retorne um valor composto pelo byte menos significativo de `int x` e pelos três bytes mais significativos de `int y`. Por exemplo, para os valores de `x=0x89ABCDEF` e `y=0x76543210`, a expressão deverá retornar `0x765432EF`.

[1v] **b)** Implemente em C a função `int right_shifts_are_arithmetic()` que deve retornar 1 quando compilada e executada numa arquitetura que use deslocamentos aritméticos para a direita ou 0, caso contrário, isto é, que use deslocamentos lógicos. Assuma que a sua função apenas irá tratar variáveis do tipo `int` (com sinal). Deve ser possível compilar e executar a sua função independentemente do número de bytes usados para representar um inteiro.

[3v] **Grupo III – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

Considere as seguintes declarações:

```
struct s1{
   char a;
   int b[3];
   union u1 *c;
   struct s2 d;
};
```

```
struct s2{
   int e;
   short *f[2];
   long long *g;
   struct s3 h;
};
```

```
struct s3{
   short i;
   struct s2 *j;
   struct s3 *k;
};
```

```
union u1{
  short l;
  char m;
  struct s2 *n;
  struct s1 *o;
};
```

[1.5v] **a)** Indique o alinhamento dos campos de uma estrutura do tipo `struct s1`. Indique claramente, para cada campo, o seu endereço, bem como as partes alocadas, mas não usadas, para satisfazer as restrições de alinhamento. Indique o tamanho total da estrutura. **Admita que a estrutura está colocada a partir do endereço `0x100`.**

[1.5v] **b)** Se definirmos os campos da estrutura `struct s2` por outra ordem é possível reduzir o número de bytes necessários para o seu armazenamento? **Justifique a sua resposta** indicando, em caso afirmativo, qual a ordem dos campos que garante o menor tamanho, o novo endereço de cada campo e das partes alocadas, mas não usadas, para satisfazer as restrições de alinhamento, bem como o novo tamanho total da estrutura. **Admita que a estrutura está colocada a partir do endereço `0x200`.**

[5v] **Grupo IV – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

[1v] **a)** Admita os seguintes endereços e conteúdo da memória e registos. Que valor (em hexadecimal) é armazenado em `%eax` em cada uma das seguintes instruções? **Justifique as suas respostas**.

Endereço	Conteúdo
0x8000	0x5
0x8004	0xA
0x8008	0xF

Registo	Conteúdo
%edx	0x8000
%ebx	2

```
leal (%edx), %eax
movl (%edx), %eax
leal 4(%edx), %eax
movl 4(%edx), %eax
leal (%edx, %ebx, 4), %eax
movl (%edx, %ebx, 4), %eax
```

[2v] **b)** Admita a seguinte declaração de uma função em C: `void xpto(int *p1, int p2);`

Esta função terá de ser invocada dentro de uma função `void func1(int a, int b, int c)` que está a desenvolver em Assembly. A função `xpto` deverá ser invocada passando-lhe como primeiro parâmetro o endereço do parâmetro `b` e, como segundo parâmetro, o resultado da soma do parâmetro `a` com o parâmetro `c`. Apresente a sequência de instruções em Assembly que permitam realizar essa invocação da função `xpto`, garantindo que quer a *stack* quer os registos que usar terão o mesmo estado antes e depois desse bloco.

[2v] **c)** Considerando o código Assembly à esquerda otimizado pelo compilador, preencha os espaços em branco no código em C com a mesma funcionalidade, mas não otimizado. (**escreva a função completa na folha A4**)

```
func2:
   pushl %ebp
   movl %esp, %ebp
   movl 8(%ebp), %eax
   testl %eax, %eax
   je .L5
   xorl %edx, %edx
   testl %eax, %eax
   js .L3
.L4:
   addl $1, %edx
   shll %eax
   jns .L4
.L3:
   movl 12(%ebp), %ecx
   movl %edx, (%ecx)
   jmp .L2
.L5:
   movl $32, %eax
.L2:
   movl %ebp, %esp
   popl %ebp
   ret
```

```
int func2(int x, int *p){
   int n = ___________;

   if(___________)
      return ___________;

   while(___________){
      ___________;
      ___________;
   }

   ___________;
   return ___________;
}
```

[2v] **Grupo V – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

Admita o seguinte excerto de código em C. A função `calc_hash` recebe como primeiro parâmetro o endereço de uma estrutura onde está armazenado o endereço de um vetor de *strings* (`strs`), assim como o número de *strings* armazenadas nesse vetor (`num`). A função recebe como segundo parâmetro o endereço de um inteiro `hash` onde é armazenado o resultado computado.

```
typedef struct{
  int num;
  char **strs;
}data_t;

void calc_hash(data_t *src, int *hash){
 int i, j;
 *hash = 0;

 for(i = 0; i< get_num(src); i++)
  for(j= 0; j < strlen(src->strs[i]); j++)
   *hash += secret(src->strs[i],j) + strlen(src->strs[i])/2;
}
```

```
int get_num(data_t *src){
  return src->num;
}

int secret(char *str, int pos){
  return str[pos] % 26;
}
```

Apresente uma segunda versão da função `calc_hash` em C com a mesma funcionalidade, mas melhor desempenho. Admita que o compilador que é usado não efetua nenhuma otimização. **Indique claramente cada uma das otimizações usadas sob a forma de comentário no código.**

DEI / Licenciatura em Engenharia Informática
Arquitectura de Computadores
Exame Época Normal – Fevereiro 2022

- Autorizada apenas a consulta da folha de consulta oficial.
- A infração implica, no mínimo, a **ANULAÇÃO** da prova.
- Quando omissa a arquitectura, considere Linux/x86-64.

Versão: A Nota mínima: **7.5/20 valores** / Duração: 120 minutos

Número:_____ **Nome:**_____

Responda aos grupos II, III, IV e V em folhas A4 separadas.

[8v] **Grupo I - Assinale no seguinte grupo se as frases são verdadeiras ou falsas (uma resposta errada desconta 50% de uma correta).**

V F

1) Em C, considere "`unsigned char x=-1; short y=10;`". À variável "`short z=x+y;`" é atribuído o valor 265 ☐☐

2) Em C, os tipos de dados com sinal usam mais um bit para armazenar se o valor é positivo ou negativo do que os seus equivalentes sem sinal . ☐☐

3) Em C, considere "`int x=0xA0B0F0CC;` ". À variável "`short y=(short)x;`" é atribuído um valor interpretado como negativo ☐☐

4) Em C, o operador lógico || (OR) termina a avaliação da expressão logo que encontre uma condição que seja avaliada como verdade ☐☐

5) Em C, admita "`short v[]={0xAABB,0xCCDD}; int x=*(int*)v;`". Então, no inteiro `x` fica armazenado o valor `0xCCDDAABB` . ☐☐

6) Em C, "`x>>2`" aplica um deslocamento aritmético para a direita se `x` for do tipo `unsigned int` e um lógico se `x` for do tipo `int` ☐☐

7) Em C, é seguro retornar como valor de saída de uma função o endereço de um vetor "`short *vec=(short*)malloc(20)` " ☐☐

8) Em x86-64, a instrução "`pushq %rax`" é o equivalente a "`subq $8,%rsp`" seguido de "`movq %rax,(%rsp)`" ☐☐

9) Em x86-64, o valor final de `%rbx` após a instrução "`cmovg %rax,%rbx`" depende do valor dos bits do registo RFLAGS ☐☐

10) Em x86-64, "`testl $1,%ecx`" seguido de "`jz xpto`" permite saltar para a etiqueta `xpto` se o valor de `%ecx` for 1 ☐☐

11) Em x86-64, a *stack* nunca é usada para passar parâmetros a uma função ☐☐

12) Em x86-64, a instrução "`leaq (%rax,%rax,4),%rax`" pode ser usada para multiplicar por cinco o valor presente em `%rax` ☐☐

13) Em x86-64, é possível usar "`shll $3, %eax`" seguido de "`notl %eax`" para multiplicar por -8 o valor de `%eax` ☐☐

14) Em x86-64, admita que o valor de `%rsp` é `0x1008`. A execução da instrução `call` coloca o valor de `%rsp` em `0x1000` ☐☐

15) Em x86-64, qualquer instrução que altere os 4 bytes menos significativos de um registo coloca a zero os 4 bytes mais significativos ☐☐

16) Em x86-64, de acordo com a convenção de salvaguarda e restauro de registos estudada nas aulas, `%r12` é um registo *callee saved* ☐☐

17) O sistema operativo executa periodicamente uma desfragmentação da *heap* para melhorar o desempenho dos programas em C ☐☐

18) A fragmentação interna dos blocos reservados na *heap* é consequência das regras de alinhamento e *overhead* da gestão dos blocos ☐☐

19) Na hierarquia de memória à medida que nos afastamos do CPU, a capacidade de armazenamento aumenta, mas diminui a performance ☐☐

20) O bloco de código "`for(i=0;i<N;i++)for(j=0;j<M;j++)sum+=m[j][i];`" exibe boa localidade espacial e temporal ☐☐

[2v] **Grupo II – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

[1v] **a)** A instrução `leaq` pode ser usada para realizar operações do tipo `(A<<K) + B`, em que `K` é 0, 1, 2, ou 3. Por exemplo, podemos calcular `3*A` como `(A<<1) + A`, invocando `leaq (%rax,%rax,2), %rax`. Considerando apenas os casos `B = 0` ou `B = A`, e para todos os valores possíveis de `K`, que múltiplos de `A` podem ser calculados com uma única invocação da função `leaq`?

[1v] **b)** Para cada um dos valores de `K` indicados, como podemos calcular `X * K` usando apenas o número indicado de operações (deslocamentos e somas/subtrações)?

K	Deslocamentos	Somas/Subtrações	Expressão
7	1	1	
30	4	3	
28	2	1	
55	2	2	

[4v] **Grupo III – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

```
struct s1{
    short a;
    char b;
    struct s2 c;
    union u1 d;
    long e;
};
```

```
struct s2{
   long *f;
   struct s1 *g;
   struct s2 *h;
   int i;
   union u1 *j[3];
};
```

```
union u1 {
  int k;
  char l;
  long m;
  struct s1 *n;
};
```

[1v] **a)** Indique o alinhamento dos campos de uma estrutura do tipo **`struct s1`**. Indique claramente, para cada campo, o seu endereço, bem como as partes alocadas, mas não usadas, para satisfazer as restrições de alinhamento. Indique o tamanho total da estrutura. **Admita que a estrutura está colocada a partir do endereço `0x100`.**

[1v] **b)** Se definirmos os campos da estrutura **`struct s2`** por outra ordem é possível reduzir o número de bytes necessários para o seu armazenamento? **Justifique a sua resposta** indicando, em caso afirmativo, qual a ordem dos campos que garante o menor tamanho, o novo endereço de cada campo e das partes alocadas mas não usadas, bem como o novo tamanho total da estrutura.

[2v] **c)** Considere o seguinte fragmento de código em C:

```
short return_s2_c_i(struct s2 **matrix, int i, int j){
  return matrix[i][j].g->c.i;
}
```

Reescreva a função `return_s2_c_i` em Assembly. Na sua resolução tenha em consideração que `matrix` é uma matriz dinâmica de estruturas do tipo `struct s2`. Assuma que os valores de `i` e `j` estão dentro dos limites reservados. Respeite a declaração da estrutura usada na alínea a). **Comente o seu código.**

[3v] **Grupo IV – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

[1,5v] **a)**No seguinte excerto de código em C foram omitidos os valores das constantes `M` e `N`:

```
#define M       /* Número mistério 1 */
#define N       /* Número mistério 2 */

long P[M][N];
long Q[N][M];

long sum_element(long i, long j){
  return P[i][j] + Q[j][i];
}
```

Admita que a função foi compilada para valores específicos de `M` e `N` e o compilador gerou o seguinte código em Assembly:

```
sum_element:
  leaq 0(,%rdi,8), %rdx
  subq %rdi, %rdx
  addq %rsi, %rdx
  leaq (%rsi,%rsi,4), %rax
  addq %rax, %rdi
  leaq Q(%rip), %r8
  leaq P(%rip), %r9
  movq (%r8,%rdi,8), %rax
  addq (%r9,%rdx,8), %rax
  ret
```

Quais os valores de `M` e `N`? **Justifique a sua resposta**

[1,5v] **b)** Admita os seguintes endereços e conteúdo da memória:

Endereço	Conteúdo
0x1000	0x1018
0x1004	0x1014
0x1008	0x1010
0x100C	0x100C
0x1010	0x1008
0x1014	0x1004
0x1018	0x1000

Admita que o endereço do vetor `vec` é `0x1000` e são executadas as seguintes instruções:

```
leaq vec(%rip), %rdx
movl $3, %ecx
leaq (%rdx, %rcx, 4), %rax
movl (%rax, %rcx, 4), %eax
```

No final, que valor (em hexadecimal) fica em `%eax`?
Justifique a sua resposta.

[3v] **Grupo V – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

```
/* increment values by k */
void incrk(int *v, int *z, int k){
   *v += k;
   *z += k;
}


/* compute x + 3 + y + 3 */
int fun(int x, int y){
   int localx = x;
   int localy = y;
   incrk(&localx, &localy, 3);
   return localx + localy;
}
```

Com base no código C acima, preencha os espaços em branco no código correspondente em Assembly ao lado. (**escreva a função completa na folha A4**).

```
fun:
  pushq %rbp
  movq %rsp,%rbp
  _____________________

  _____________________
  _____________________
  _____________________
  _____________________
  _____________________

  call incrk
  _____________________
  _____________________


  movq %rbp,%rsp
  popq %rbp
  ret
```

DEI / Licenciatura em Engenharia Informática
Arquitectura de Computadores
Exame Época Recurso – Fevereiro 2022

- Autorizada apenas a consulta da folha de consulta oficial.
- A infração implica, no mínimo, a **ANULAÇÃO** da prova.
- Quando omissa a arquitectura, considere Linux/x86-64.

Versão: A Nota mínima: **7.5/20 valores** / Duração: 120 minutos

Número:_____ **Nome:**_____

Responda aos grupos II, III, IV e V em folhas A4 separadas.

[8v] **Grupo I - Assinale no seguinte grupo se as frases são verdadeiras ou falsas (uma resposta errada desconta 50% de uma correta).**

V F

1) Em C, se tivermos um `char` com representação binária de `10011010`, o *cast* para um `short` resulta no valor `1111111110011010` ☐☐

2) Em C, considere "`int x=0x01234567;`" com o endereço de `x` em `0x100`. Logo, o valor presente no byte `0x101` é `0x23` ☐☐

3) Em C, considere "`short x=0x1234;`". O resultado da operação "`x && 0x0F0F`" é `0x0204` ☐☐

4) Em C, a avaliação de expressões com variáveis com e sem sinal interpreta todas as variáveis como sendo valores com sinal ☐☐

5) Em C, admita um vetor "`int vec[10];`" e um apontador "`short *ptr = (short*) vec`". Então, `ptr + 4` avança para `vec[2]` ☐☐

6) Em C, quando a soma de duas variáveis "`unsigned char u,v;`" é igual ou superior a 2^8 o valor obtido é equivalente a `u + v - 2^8` ☐☐

7) Em C, executar "`malloc(strlen("arqcp"))`" permite-nos reservar na *heap* os bytes suficientes para armazenar a string "`arqcp`" ☐☐

8) Em x86-64, a instrução "`popq %rax`" é o equivalente a "`movq %rax,(%rsp)`" seguido de "`subq $8,%rsp`" ☐☐

9) Em x86-64, se atribuirmos valores com sinal aos registos a somar, o resultado será incorreto se a *flag* de *carry* estiver ativa após a soma ☐☐

10) Em x86-64, "`idivq %rcx`" efetua a divisão (com sinal) entre `%rax` e `%rcx` colocando o quociente em `%rax` e o resto em `%rdx` ☐☐

11) Em x86-64, ao contrário das operações de deslocamento de bits, as rotações nunca perdem os bits da informação original ☐☐

12) Em x86-64, a instrução "`leaq (%rax,%rax,6),%rax`" pode ser usada para multiplicar por sete o valor presente em `%rax` ☐☐

13) Em x86-64, é possível obter o mesmo resultado com "`imull $-8,%eax`" e "`shll $3,%eax; notl %eax; incl %eax`" ☐☐

14) Em x86-64, admita que o valor de `%rsp` é `0x1008`. A execução da instrução `ret` coloca o valor de `%rsp` em `0x1000` ☐☐

15) Em x86-64, o equivalente a "`*ptr1 = *ptr2`", apontadores do tipo `int*` em C, pode ser obtido com "`movl (%rax),(%rcx)`" ☐☐

16) Em x86-64, de acordo com a convenção de salvaguarda e restauro de registos estudada nas aulas, `%r10` é um registo *caller saved* ☐☐

17) Em x86-64, o endereço inicial de uma *struct* alinhada depende das restrições de alinhamento dos seus campos ☐☐

18) Em x86-64, o espaço ocupado por uma *union* é sempre o mesmo, independentemente da ordem dos seus campos ☐☐

19) Em x86-64, a *stack* é usada para suportar o retorno do valor de saída de uma função, tal como acontece com o controlo de fluxo ☐☐

20) O bloco de código "`for(j=0;j<N;j++)for(i=0;i<M;i++)sum+=m[j][i];`" exibe boa localidade espacial e temporal ☐☐

[3v] **Grupo II – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

[1,5v] **a)** Cada uma das seguintes linhas de código gera um erro quando invocamos o *assembler*. Explique o que está errado em cada uma delas.

```
1. movb $0xF, (%ebx)
2. movl %rax, (%rsp)
3. movw (%rax), 4(%rsp)
4. movq %rax, $0x123
5. movl %eax, %rdx
6. movw %si, 8(%rdi,%rcx,9)
```

[1,5v] **b)** Assuma os apontadores `src_t *sp` e `dest_t *dp`, em que `src_t` e `dest_t` são tipos de dados declarados com `typedef`. Assuma que os endereços `sp` e `dp` são passados por parâmetro a uma função e estão, portanto, armazenados nos registos `%rdi` e `%rsi`, respetivamente. Para cada uma das entradas seguintes da tabela indique **as duas instruções em Assembly** que implementam o equivalente à operação `*dp = (dest_t) *sp` realizada dentro da função em C.

src_t	dest_t	Instruções Assembly
long	long	
char	int	
int	unsigned long	
unsigned char	long	
int	char	
unsigned int	unsigned short	

[3v] **Grupo III – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

```
struct s1{
    char a;
    short b;
    struct s2 *c;
    union u1 d;
    struct s2 e;
};
```

```
struct s2{
   long f;
   struct s1 *g;
   struct s2 *h;
   char i;
   char j[3];
};
```

```
union u1 {
  int *k;
  char l;
  long m[2];
  struct s1 *n;
};
```

[1,5v] **a)** Indique o alinhamento dos campos de uma estrutura do tipo **`struct s1`**. Indique claramente, para cada campo, o seu endereço, bem como as partes alocadas, mas não usadas, para satisfazer as restrições de alinhamento. Indique o tamanho total da estrutura. **Admita que a estrutura está colocada a partir do endereço `0x100`.**

[1,5v] **b)** Considere que a função `init` opera sobre uma estrutura do tipo `struct test` e que o compilador gerou o seguinte código Assembly. Com base nesta informação, preencha as expressões em falta no código em C para a função `init`. **Justifique as suas escolhas**.

```
struct test{
    short *p;
    struct s{
      short x;
      short y;
    }
    struct test *next;
};
```

```
void init(struct test *st){
   st->s.y  = _______;
   st->p    = _______;
   st->next = _______;
}
```

```
init:
   movw 8(%rdi), %ax
   movw %ax, 10(%rdi)
   leaq 10(%rdi), %rax
   movq %rax, (%rdi)
   movq %rdi, 16(%rdi)
   ret
```

[3v] **Grupo IV – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

[1,5v] **a)** No seguinte excerto de código em C foi omitido o valor da constante `M`:

```
#define M      /* Número mistério */

void transpose(long A[M][M]) {
  long i, j;
  for(i=0; i<M; i++){
   for(j=0; j<i; j++){
     long t =  A[i][j];
     A[i][j] = A[j][i];
     A[j][i] = t;
   }
}
```

Admita que a função foi compilada para um valor específico de `M` e o compilador gerou o seguinte código otimizado em Assembly para o ciclo interior da função:

```
.L6:
  movq (%rdx), %rcx
  movq (%rax), %rsi
  movq %rsi, (%rdx)
  movq %rcx, (%rax)
  addq $8, %rdx
  addq $120, %rax
  cmpq %rdi, %rax
  jne .L6
```

Qual o valor de `M`? **Justifique a sua resposta.**

[1,5v] **b)** Use os seguintes valores iniciais em memória e nos registos da arquitetura para responder a cada uma das questões (isto é, cada questão não é afetada pela execução das instruções anteriores).

Endereço	Valor
0x100	0xFF
0x104	0xAB
0x108	0x13
0x10C	0x11

Registo	Valor
%rax	0x100
%rcx	0x1
%rdx	0x3
%rbx	0x4

A. Qual o novo valor de `%eax` após "`movl 0x100,%eax`"? _____

B. Qual o novo valor de `%ecx` após "`movl (%rax,%rdx,4),%ecx`"? _____

C. Qual o endereço que é alterado com "`subl %edx,4(%rax)`"? _____

D. Qual o novo valor de `%rbx` após "`leaq 0x100(,%rbx,2),%rbx`"? _____

[3v] **Grupo V – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

```
long f1(long a, long b, long c,
       long d, long e, long f){
  return a + f2(a*b,b,c,d,e,f,10,-20);
}
```

Com base no código C acima, preencha os espaços em branco no código correspondente em Assembly ao lado. (**escreva a função completa na folha A4**).

```
f1:
  _______________
  movq %rdi, %rax
  imulq %rsi
  _______________
  _______________
  _______________
  call f2
  _______________
  _______________
  _______________
  ret
```