- 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];
      }
   }
}
```