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

Hipótese b.

A `int x` é atribuído o valor `0x015213F1`. O cast de `x` para `char` atribui a `y` o valor `0xF1`, equivalente a `-15` em decimal. Se esse mesmo valor for interpretado como um valor sem sinal, o bit mais significativo passa a fazer parte do número, o que resulta no valor `241` atribuído a `z`.

[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

Hipótese c.

O operador ! é um operador lógico, tratando 0 como falso e um valor diferente de 0 como verdade. Assim, `!!x` será sempre 0 se `x` for 0, ou 1 se `x` tiver um qualquer valor diferente de 0. Ao deslocarmos 31 bits para a esquerda o valor resultante, os únicos outputs possíveis serão 0 ou `INT_MIN`, uma vez que serão sempre adicionados zeros pelo lado direito, independentemente do valor que pretendemos deslocar para a esquerda.

[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`.**

```
struct s2{
   struct s1 d;            0x100: 12 bytes
   struct s1 *e;           0x10C: 4 bytes
   struct s2 *f;           0x110: 4 bytes
   long long int g;        0x114: 8 bytes
   short h[3];             0x11C: 6 bytes
   [gap]                   0x122: 2 bytes
};
```

Tamanho da estrutura: 36 bytes

[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.

Não. Mesmo que indiquemos os campos por ordem decrescente de tamanho de tipo de dados, a restrição de alinhamento que obriga a que o tamanho total da estrutura seja múltiplo da maior restrição de alinhamento dos seus campos (neste caso K=4) faz com que a estrutura tenha sempre 12 bytes.

[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.**

```
return_u1_i:
   pushl %ebp
   movl %esp, %ebp
   movl 8(%ebp), %eax
   movl 12(%ebp), %ecx
   imull $720, %ecx        # i * 20 * sizeof(struct s2)
   addl %ecx, %eax         # matrix[i]
   movl 16(%ebp), %edx
   imull $36, %edx         # j * sizeof(struct s2)
   addl %edx, %eax         # &matrix[i][j]
   movl 12(%eax), %eax     # matrix[i][j].e
   movl 4(%eax), %eax      # &matrix[i][j].e->b
   movl (%eax), %eax       # matrix[i][j].e->b->i
   movl %ebp, %esp
   popl %ebp
   ret
```

Nota: o código apresentado pode ser otimizado, mas é aceite uma versão sem otimização.

[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->b->j.h[2]__;
}
```

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

```
proc2:
   pushl %ebp
   movl %esp,%ebp
   __8(%ebp),%eax___
   __12(%eax),%eax__
   __8(%eax),%eax___
   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->k->c__;
}
```

[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.**

```
void count_positives_celsius(List *p, int *k){
   // Acumular resultados num registo (maior probabilidade)
   int acc = 0;
   int temp;

   // Eliminar invocação de funções
   while(p!=NULL){
     // Reutilização de expressões
     temp = p->data-32;

     // Reduzir custo das operações
     if ( ((tmp<<2)+tmp)>>3 > 0 )
         acc++;
      p = p->next;
   }
   // Apenas uma escrita na memória
   *k = acc;
}
```