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`.

```
(x&0xFF) | (y&0xFFFFFF00)
```

[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.

Existem diversas soluções para esta questão. O principal desafio é obter uma solução que se adapte a diferentes tamanhos usados na representação de um inteiro. A solução apresentada aplica um deslocamento para a direita a um valor cuja representação binária tem todos os bits a 1. Se os deslocamentos forem aritméticos, o resultado do deslocamento ainda terá todos os bits a 1 e, por isso, será negativo.

```
int right_shifts_are_arithmetic() {
   int x = -1; /* Todos os bits a 1 */
   return (x >> 1) < 0;
}
```

[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`.**

```
struct s1{
   char a;          0x100: 1 byte
   [gap]            0x101: 3 bytes
   int b[3];        0x104: 12 bytes
   union u1 *c;     0x110: 4 bytes
   struct s2 d;     0x114: 28 bytes
};
```

Tamanho da estrutura: 48 bytes

[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`.**

```
struct s2{
   struct s3 h;     0x200: 12 bytes
   long long *g;    0x20C: 4 bytes
   short *f[2];     0x210: 8 bytes
   int e;           0x218: 4 byte
};
```

Tamanho da estrutura: 28 bytes

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 28 bytes.

[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                  0x8000
movl (%edx), %eax                  0x5
leal 4(%edx), %eax                 0x8004
movl 4(%edx), %eax                 0xA
leal (%edx, %ebx, 4), %eax         0x8008
movl (%edx, %ebx, 4), %eax         0xF
```

Enquanto que a instrução `leal` apenas determina o valor resultante da expressão de endereçamento indicada (sem aceder à memória), a instrução `mov` copia o conteúdo da memória presente no endereço indicado para o destino.

[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.

```
pushl %edx          # guardar %edx e %eax
pushl %eax
leal 12(%ebp),%eax  # &b
movl 8(%ebp),%edx   # a + c
addl 16(%ebp),%edx
pushl %edx          # passar parâmetros
```

```
pushl %eax
call xpto           # xpto(&b,a+c)
addl $8,%esp        # limpar parâmetros da
stack
popl %eax           # restaurar %edx e %eax
popl %edx
```

[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
   sall %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 = ____0____;

   if(___x==0___)
      return ____32____;

   while(___x >= 0___){
      ___n++____;
      _x = x<<1_;
   }

   ___*p = n___;
   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. 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.**

```
void calc_hash(data_t *src, int *hash){
 int i, j, tmp, num,size;

 /* Acumular resultados num registo (maior probabilidade) */
 tmp = 0;

 /* Remover invocação de funções */
 num = get_num(src);
 for(i = 0; i< num; i++){
  size = strlen(src->strs[i]);
  for(j= 0; j < size; j++)
   /* Remover invocação de funções, partilha de expressões comuns, redução do custo */
   tmp += (strs[i][j]%26) + (size<<1);
 }

 /* Diminuir acessos à memória */
 *hash = tmp;
}
```