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, num `unsigned char` com representação binária de `10011010`, o *cast* para um `short` resulta no valor `1111111110011010`.	☐	☐
2) Em C, considere "`unsigned int x=0x01234567;`" com o endereço de `x` em `0x100`. Logo, o valor presente no byte `0x101` é `0x45`.	☐	☐
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 sem sinal	☐	☐
5) Em C, admita um vetor "`short vec[10];`" e um apontador "`int *ptr = (int*)vec`". Então, `ptr + 2` avança para `vec[4]`	☐	☐
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")+1)`" permite-nos reservar na *stack* 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 "`addq $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, à semelhança das operações de deslocamento de bits, as operações de rotação também perdem os bits da informação original	☐	☐
12) Em x86-64, a instrução "`leaq (%rax,%rax,8),%rax`" pode ser usada para multiplicar por nove 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 `0x1000`. A execução da instrução `ret` coloca o valor de `%rsp` em `0x1008`	☐	☐
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, `%rbx` é um registo *callee saved*	☐	☐
17) Em x86-64, o tamanho total de uma *struct* alinhada não depende da ordem 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* nunca é usada para suportar a passagem de parâmetros a uma função	☐	☐
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	☐	☐

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

[1,5v] **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`.

[1,5v] **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:

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

```
typedef struct {
    short code;
    short start;
    char  raw[5];
    short sense;
    short ext;
    long  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)** Considere o seguinte fragmento de código em C, respeitando as declarações das estruturas apresentadas acima.

```
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   = 15;

   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/x86-64!

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

[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.**

```
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
```