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`?

Para cada valor de K podemos calcular dois múltiplos. 2^k (quando B é zero) e $2^k + 1$ (quando B é igual a A). Logo, podemos calcular os múltiplos 1, 2, 3, 4, 5, 8 e 9.

[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	(x<<3) - x
30	4	3	(x<<4) + (x<<3) + (x<<2) + (x<<1)
28	2	1	(x<<5) - (x<<2)
55	2	2	(x<<6) - (x<<3) - x

[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`.**

```
struct s1{
    short a;                0x100 : 2 bytes
    char b;                 0x102 : 1 byte
    [gap]                   0x103 : 5 bytes
    struct s2 c;            0x108 : 56 bytes
    union u1 d;             0x140 : 8 bytes
    long e;                 0x148 : 8 bytes
};
```

Tamanho da estrutura: 80 bytes

[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.

Não. Independentemente da ordem em que sejam declarados os campos da estrutura, as regras de alinhamento determinam que o tamanho total da estrutura seja de 56 bytes. Mesmo que o campo *i* seja o último, como o tamanho total da estrutura tem de ser múltiplo de K=8, serão acrescentados 4 bytes após o campo *i*.

```
struct s2{
   long *f;                 8 bytes
   struct s1 *g;            8 bytes
   struct s2 *h;            8 bytes
   union u1 *j[3];          24 bytes
   int i;                   4 bytes
   [gap]                    4 bytes
};
```

[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.**

```
return_s2_c_i:
      #matrix in %rdi, i in %esi, j in %edx
      movq (%rdi,%rsi,8), %rdi      # matrix[i]
      imulq $56, %rdx               # j * sizeof(struct s2)
      addq %rdx, %rdi               # matrix[i][j]
      movq 8(%rdi), %rdi            # matrix[i][j].g
      movl 32(%rdi), %eax           # matrix[i][j].g->c.i
      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 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**

```
sum_element:
  leaq 0(,%rdi,8), %rdx          # %rdx = 8 * i
  subq %rdi, %rdx                # %rdx = 7 * i
  addq %rsi, %rdx                # %rdx = 7 * i + j
  leaq (%rsi,%rsi,4), %rax       # %rax = 5 * j
  addq %rax, %rdi                # %rdi = 5 * j + i
  leaq Q(%rip), %r8              # %r8 = &Q[0]
  leaq P(%rip), %r9              # %r9 = &P[0]
  movq (%r8,%rdi,8), %rax        # %rax = Q[5 * j + i]
  addq (%r9,%rdx,8), %rax        # %rax = %rax + P[7 * i + j]
  ret
```

Podemos ver que a referência à posição da matriz Q é dada pelo deslocamento de 8*(5*j + i) bytes em relação a &Q[0], enquanto que a referência à posição da matriz P é dada pelo deslocamento de 8*(7*i + j) bytes em relação a &P[0]. Logo, podemos determinar que P tem 7 colunas, enquanto Q tem 5, obtendo assim os valores M = 5 e N = 7.

[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.

```
leaq vec(%rip), %rdx           # %rdx = 0x1000
movl $3, %ecx                  # %ecx = 3
leaq (%rdx, %rcx, 4), %rax     # %rax = %rdx + 4 * %rcx = 0x100C
movl (%rax, %rcx, 4), %eax     # %eax = *(%rax + 4 * %rcx) = *(0x1018) = 0x1000
```

[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
  subq $8, %rsp

  movl %edi, -4(%rbp)
  movl %esi, -8(%rbp)
  movl $3, %edx
  leaq -4(%rbp), %rdi
  leaq -8(%rbp), %rsi

  call incrk

  movl -4(%rbp), %eax
  addl -8(%rbp), %eax

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