

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 “short x=-1;”. A atribuição “unsigned int y=x;” primeiro altera o sinal de x e depois o tamanho..... | <input type="checkbox"/> <input checked="" type="checkbox"/> |
| 2) Em C, o apontador “int *ptr;” declarado na função main() é alocado na heap..... | <input type="checkbox"/> <input checked="" type="checkbox"/> |
| 3) Em C, admita as variáveis “unsigned char a=0;” e “short b=-1;”. A comparação “if (b<a)” é verdadeira | <input type="checkbox"/> <input checked="" type="checkbox"/> |
| 4) Em C, a função realloc permite-nos redimensionar blocos de memória reservados com a função malloc() mas não com calloc() | <input type="checkbox"/> <input checked="" type="checkbox"/> |
| 5) Em C, as operações aritméticas com tipos inteiros seguem as regras da aritmética modular, como consequência do seu número finito de bits... | <input checked="" type="checkbox"/> <input type="checkbox"/> |
| 6) Em C, para que o tamanho de uma union seja o menor possível, devemos declarar os seus campos por ordem decrescente de tamanho..... | <input type="checkbox"/> <input checked="" type="checkbox"/> |
| 7) Em C, admita as variáveis “char str[30];” e “int* ptr=str;”. Logo, “ptr=ptr+2;” avança para o nono elemento de str..... | <input checked="" type="checkbox"/> <input type="checkbox"/> |
| 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 | <input checked="" type="checkbox"/> <input type="checkbox"/> |
| 9) Em IA32, a instrução “call func” não altera o estado atual da stack, apenas o valor do registo %eip com o endereço da etiqueta func | <input type="checkbox"/> <input checked="" type="checkbox"/> |
| 10) Em IA32, a instrução “idivw %cx” assume que o dividendo se encontra em %eax, deixando o quociente em %ax e o resto em %dx..... | <input type="checkbox"/> <input checked="" type="checkbox"/> |
| 11) Em IA32, a instrução adc só permite adicionar aos seus operandos o valor da flag de carry quando aplicada a valores com sinal..... | <input type="checkbox"/> <input checked="" type="checkbox"/> |
| 12) Em IA32, “testl \$-1, %ecx” seguido de “jz xpto” permite saltar para a etiqueta xpto se o valor de %ecx for zero..... | <input checked="" type="checkbox"/> <input type="checkbox"/> |
| 13) Admita o vetor global “int a[5];” em C. “movl \$2, %ecx” seguido de “movl a(, %ecx, 8), %eax” coloca a[4] em %eax..... | <input checked="" type="checkbox"/> <input type="checkbox"/> |
| 14) Em IA32, é possível usar “leal (%edx, %ecx, 4), %eax” para ler um valor de 4 bytes da memória e colocá-lo em %eax..... | <input type="checkbox"/> <input checked="" type="checkbox"/> |
| 15) Em IA32, a instrução “pushl %eax” é equivalente a “movl (%esp), %eax” seguido de “subl \$4, %esp” | <input type="checkbox"/> <input checked="" type="checkbox"/> |
| 16) Admita a matriz dinâmica “int **m”, com 10 elementos por linha. Em IA32, acedemos a m[2][3] avançando 92 bytes a partir de m | <input type="checkbox"/> <input checked="" type="checkbox"/> |
| 17) O bloco de código “for(i=0; i<N; i++) for(j=0; j<M; j++) sum+=m[i][j];” exhibe boa localidade espacial mas não temporal | <input type="checkbox"/> <input checked="" type="checkbox"/> |
| 18) A fragmentação da heap pode impedir que a função realloc() consiga redimensionar um bloco existente para um tamanho menor..... | <input type="checkbox"/> <input checked="" type="checkbox"/> |
| 19) Na hierarquia de memória, à medida que nos afastamos do CPU abdicamos da performance em favor do custo por byte..... | <input checked="" type="checkbox"/> <input type="checkbox"/> |
| 20) Uma das otimizações efetuadas pelos compiladores de C é substituição da invocação de uma função pelo seu código..... | <input checked="" type="checkbox"/> <input type="checkbox"/> |

[2v] **Grupo II – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

[1v] **a)** Foi-lhe dada a tarefa de escrever código que multiplique a variável int x por vários valores constantes K diferentes. Para que o seu código seja eficiente, apenas deve usar as operações +, - e <<. Assuma os seguintes valores de K e escreva as expressões em C que realizem a multiplicação pretendida, usando no máximo até três operações:

- | | |
|--|---|
| <p>(a) K = 17 (x << 4) + x</p> <p>(b) K = -7 -(x << 3) + x</p> | <p>(c) K = 30 (x << 5) - (x << 1)</p> <p>(d) K = -56 -(x << 6) + (x << 3)</p> |
|--|---|

[1v] **b)** Implemente em C a função int is_big_endian() que deve retornar 1 quando compilada e executada numa arquitetura big-endian ou 0, caso seja compilada e executada numa arquitetura little-endian. Deve ser possível compilar e executar a sua função independentemente do número de bytes usados para representar um inteiro.

Existem diversas formas de resolver este problema. A ideia básica é criar um valor com mais do que um byte com valores diferentes para os bytes mais e menos significativos. Depois, basta ler o byte menos significativo e determinar o seu valor. Esta solução cria um inteiro com o valor inicial 1, acedendo depois ao seu primeiro byte e convertendo-o para um inteiro. Este inteiro terá o valor 0 numa arquitetura big-endian e 1 numa arquitetura little-endian.

```
int is_big_endian() {
    /*MSB=0, LSB=1*/
    int x = 1;

    /* MSB (0) when big-endian, LSB (1) when little-endian */
    char byte = *(char *) &x;

    return !byte;
}
```

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

Considere as seguintes declarações:

<pre>struct s1{ int a; short b[3]; union ul c; char d; };</pre>	<pre>struct s2{ char e; short f[2]; long long g; struct s3 *h; };</pre>	<pre>struct s3{ struct s1 *i; struct s2 *j; struct s3 *k; };</pre>	<pre>union ul{ char l; short m; struct s2 n; struct s1 *o; };</pre>
---	---	--	---

[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{
    int a;           0x100: 4 bytes
    short b[3];      0x104: 6 bytes
    [gap]            0x10A: 2 bytes
    union ul c;      0x10C: 20 bytes
    char d;          0x120: 1 byte
    [gap]            0x121: 3 bytes
};
```

Tamanho da estrutura: 36 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{
    long long g;      0x200: 8 bytes
    struct s3 *h;     0x208: 4 bytes
    short f[2];       0x20C: 4 bytes
    char e;           0x210: 1 byte
    [gap]             0x211: 3 bytes
};
```

Tamanho da estrutura: 20 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 20 bytes.

[5v] **Grupo IV – Responda numa folha A4 separada que deve assinar e entregar no final do exame.**

[3v] **a)** Considere o seguinte fragmento de código em C:

```
short return_s1_b2(struct s2 **matrix, int i, int j){
    return matrix[i][j].h->i->b[2];
}
```

Reescreva a função `return_s1_b2` em Assembly. Na sua resolução tenha em consideração que `matrix` é uma matriz de estruturas dinâmica. Assuma que os valores de `i` e `j` estão dentro dos limites reservados. Respeite a declaração inicial da estrutura usada na alínea a) do grupo anterior. **Comente o seu código.**

```
return_s1_b2:
    pushl %ebp
    movl %esp, %ebp

    movl 8(%ebp), %edx      # **matrix
    movl 12(%ebp), %ecx     # i
    movl (%edx,%ecx,4), %edx # matrix[i]
    movl 16(%ebp), %ecx     # j
    imull $20, %ecx         # j*sizeof(struct s2)
    movl 16(%edx,%ecx), %edx # matrix[i][j]->h
    movl (%edx), %edx       # matrix[i][j]->h->i
    movw 8(%edx), %ax       # matrix[i][j]->h->i->b[2]

    movl %ebp, %esp
    popl %ebp
    ret
```

[1v] b) Admita o seguinte excerto de código. Indique os valores que irão aparecer no ecrã. **Justifique a sua resposta.**

```
unsigned int data[3] = {0x11223344, 0x55667788, 0x99AABBCC};
char *p=(char*)data;
printf("0x%x\n", *p);
printf("0x%x\n", *(short*) (p+2));
printf("0x%x\n", *(int*) &data[1]);
```

0x44
0x1122
0x55667788

IA32 é uma arquitetura little-endian. Logo, no endereço armazenado em p temos o byte 0x44. Ao avançarmos 2 bytes, e indicarmos que queremos imprimir 2 bytes com o cast para short, o valor impresso é 0x1122, uma vez que ao imprimirmos valores com mais do que um byte, a sua apresentação no ecrã é sempre igual, independentemente de estarmos numa arquitetura little-endian ou big-endian. O mesmo conceito se aplica no terceiro printf.

[1v] c) 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:

```
movl $vec, %edx
movl $2, %ecx
leal (%edx, %ecx, 4), %eax
movl (%eax, %ecx, 4), %eax
```

No final, que valor (em hexadecimal) fica em %eax?

Justifique a sua resposta.

0x1008. A primeira linha armazena em %edx o valor 0x1000. A instrução leal armazena em %eax o valor 0x1008. A instrução movl armazena em %eax os 4 bytes existentes na memória a partir do endereço 0x1010, o que corresponde ao valor 0x1008.

[2v] Grupo V – Responda numa folha A4 separada que deve assinar e entregar no final do exame.

O processamento de imagens oferece diversos exemplos de funções que beneficiam com a otimização de código com acessos intensivos à memória. Neste exercício iremos considerar uma função smooth que aplica um efeito de “blur” a uma imagem representada por um vetor estático bidimensional com N*N pixéis.

```
#define N 25

typedef struct {
    unsigned short red;
    unsigned short green;
    unsigned short blue;
} pixel;

unsigned short avg(pixel src[N][N], int i, int j){
    unsigned short res;
    res=(src[i+1][j]+src[i-1][j]+src[i][j-1]+src[i][j+1])/4;
    return res;
}

void smooth(pixel src[N][N], pixel dest[N][N]){
    int i,j;

    for (j = 1; j < N-1; j++)
        for (i = 1; i < N-1; i++)
            dst[i,j].red = avg(src,i,j);
}
```

A função avg retorna a média aritmética simples dos pixéis vizinhos do pixel na posição [i,j], isto é, os pixéis [i+1][j], [i-1][j], [i][j-1] e [i][j+1]. Para simplificar, considere apenas o cálculo da média da componente vermelha da cor em cada pixel. Apresente uma segunda versão da função smooth 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 smooth(pixel src[N][N], pixel dest[N][N]){
    int i,j;
    /* Redução do custo das operações no endereçamento, evitando multiplicações */
    int ni = N;
    /* Aceder aos elementos do vetor por linha, aproveitando a localidade espacial */
    for (i = 1; i < N-1; i++)
        for (j = 1; j < N-1; j++){
            /* Remover a invocação de funções */
            /* Partilhar expressões comuns */
            int nij = ni + j;
            int up, down, left, right;
            up = src[nij-N];
            down = src[nij+N];
            left = src[nij-1];
            right = src[nij+1];
            /* Redução do custo das operações */
            dst[ni].red = (up+down+left+right)>>2;
            ni+=N;
        }
}
```