Este exame tem 4 questões, num total de 200 pontos. **RESPONDA A CADA QUESTÃO EM FOLHAS SEPARADAS.**

1. Uma estação de medida colocada numa boia em alto-mar efetua um conjunto de medidas (atmosféricas, qualidade da água, etc.) periodicamente (i.e., uma vez a cada P segundos). Os dados de cada conjunto de medidas ocupam 64 KiB. A estação armazena a informação num disco magnético com as seguintes caraterísticas:

   - 1200 RPM, setores de 4 KiB;
   - tempo máximo de busca: 20 ms;
   - latência do controlador é nula;
   - taxa de transferência: 1 MB/s.

   Assumir que todos os dados do mesmo conjunto de medidas são guardados em setores adjacentes.

   [Nota: Para simplificar os cálculos, assumir: 1 KiB $\approx$ 1 kB, 1 MiB $\approx$ 1 MB.]

[20] (a) Assumir que um veículo submarino autónomo visita a estação a cada 30 dias. Determinar a quantidade de dados armazenada em função de P durante 30 dias.

Resposta: 30 dias = 30 x 24 x 3600 s = 2592000 s

Número de medidas: N = 2592000/P

Quantidade de dados armazenada em função de P: 64 KB * 2592000 / P = 165888000 KB/P

[30] (b) Um veículo submarino autónomo (VSA) visita a estação para recolher os dados armazenados. O sistema de comunicação com a estação de medida consegue atingir uma taxa de 1 MB/s. Determine o tempo máximo que o VSA demora a recolher os dados para o caso $P = 60$. Considerar que as tarefas de leitura dos dados do disco e a sua transferência para o VSA são executadas sequencialmente.

Nota: se não resolveu a alínea anterior, assuma que a quantidade de dados armazenada em 30 dias é dada por $1{,}7 \times 10^8$/P (em bytes).

Resposta:

Caso tenha resolvido a alinea a:

Total de dados a transferir: 165888000 / 60 = 2764800 KB

Conjuntos de medidas a transferir: 2764800 KB / 64 KB = 43200

Tempo de acesso ao disco por conjunto de medidas: Tbusca + Trot + Ttrans -> 20 ms + 0.5 * (60 / 1200) + 64 KB / 1 MB = 20 ms + 25 ms + 64 ms = 109 ms Tempo de transferência por conjunto de medidas (sistema de comunicação): 64 KB / 1 MB = 64 ms

Tempo total (acesso + transferencia) por conjunto de medidas: 109 ms + 64 ms = 173 ms

Tempo máximo para descarregar os dados = 43200 * 173 ms = 7473,6 s (2 horas, 4 minutos e 33,6 segundos)

Caso tenha utilizado o valor $1{,}7 \times 10^8$/P (em bytes):

Total de dados a transferir: 170000000 / 60 = 2833333 B -> 2833 KB

Conjuntos de medidas a transferir: 2833 KB / 64 KB = 44

Tempo de acesso ao disco por conjunto de medidas: Tbusca + Trot + Ttrans -> 20 ms + 0.5 * (60 / 1200) + 64 KB / 1 MB = 20 ms + 25 ms + 64 ms = 109 ms

Tempo de transferência por conjunto de medidas (sistema de comunicação): 64 KB / 1 MB = 64 ms

Tempo total (acesso + transferencia) por conjunto de medidas: 109 ms + 64 ms = 173 ms

Tempo máximo para descarregar os dados = 44 * 173 ms = 7,6 s

---

[50] 2. Dada uma sequência não ordenada de N valores inteiros com sinal ($N > 0$), designa-se por *mínimo local* um valor que é menor que os seus dois vizinhos. O primeiro e último números não podem ser mínimos locais.

Exemplo: A sequência [-2; 4; -6; -5; 9; -8; 8; 4; 12; 16; -5; -7] tem três mínimos locais: -6, -8, e 4.

Escrever em *assembly* a sub-rotina `Maxlocalminpos`, que, dada uma sequência de números inteiros com sinal de 32 bits, retorna a posição (o índice) do maior mínimo local. Caso exista mais que um valor com as caraterísticas indicadas, a sub-rotina deve retornar a posição do que está colocado mais à esquerda (índice mais baixo); caso não exista nenhum mínimo local, a sub-rotina deve retornar o valor -1. A sub-rotina poderia ser usada da seguinte forma:

```
extern int Maxlocalminpos(int v[], unsigned int N);
int vect[]={-2, 4, -6, -5, 9, -8, 8, 4,  12, 16, -5, -7};
main()
{   int pos = Maxlocalminpos(vect, 12);
    if (pos < 0) printf("Sem minimo local.\n");
    else  printf("Minimo local na posicao %d\n", pos);
}
```

O valor apresentado seria 7 (a posição do segundo número 4).

Resposta: Uma possível solução:

```
// endereço de vetor de int32: X0
// nº de elementos: W1
// Resultado: índice ou -1 W0
Maxlocalminpos:
    mov         W5, -1    // posição
    cmp         W1, 2
    b.ls    Lexit
    // pelo menos 3 elementos
    ldr         W2, [X0], 4
    ldr         W3, [X0], 4
    ldr         W4, [X0], 4
    sub         W1, W1, 3
    mov         W6, 1     // índice do elemento central atual
L1:
```

```
    cmp     W3, W2
    b.ge    Lnext
    cmp     W3, W4
    b.ge    Lnext
    // minimo local
    cmp     W5, -1   //1º mínimo
    b.ne    Lnewmin
    mov     W5, W6
    mov     W7, W3
    b       Lnext
Lnewmin:    // outro mínimo
    cmp          W3, W7
    b.le    Lnext
    mov      W5, W6
    mov          W7, W3

Lnext:
    cbz     W1,Lexit
    mov     W2, W3
    mov     W3, W4
    ldr     W4, [X0], 4
    sub     W1, W1, 1
    add     W6, W6, 1
    b       L1

Lexit:
    mov          W0, w5
    ret
```

[30] 3. Escrever sub-rotina `sum_select` em *assembly* que recebe uma sequência de números em vírgula flutuante (precisão dupla) e o respetivo comprimento ($N > 0$), calculando a expressão $\sum_i \sqrt{|x_i|}$ com os elementos x_i da sequência que verificam a condição

$$p(x, i) = 1 \qquad i = 0, 1, \ldots N-1 : \text{índice do valor na sequência.}$$

Assumir que a sub-rotina que calcula o valor da função $p(x, i)$ já existe e corresponde à sub-rotina

```
bool func_p(double x, int i);
```

A sub-rotina `sum_select` deve invocar a sub-rotina `func_p` corretamente. **Não escrever a sub-rotina `func_p`.** Exemplo de utilização:

```
extern double sum_select(double v[], int n);
double seq[]={1.23, -4.56, 1.675};
main()
{   double res = sum_elect(seq, 3); ...    }
```

Nota: o tipo `bool` tem apenas dois valores, 0 (falso) e 1 (verdadeiro).

Resposta: Uma solução possível:

```
 // X0: endereço base vetor de doubles
 // W1: número de elementos (> 0)
 // preservar (para além de SP (8) e X29 (8), X22 (8) [+16] , D15 (8) [+24] ,
 // W23 (4) [+32], W24 (4) [+36]-> 40 (-> 48)
sum_select:
    stp     X29, X30, [SP, -48]!
    mov     X29, SP      // frame pointer
    // preservar X22, D15, W23
    str     X22, [X29, 16]
    str     D15, [X29, 24]
    stp     W23, w24, [X29, 32]

    mov     W24, 0
    scvtf   D15, W24
    mov     X22, X0
    mov     W23, W1
L2:
    cbz     W23, Lfim
    ldr     D0, [X22], 8
    sub     W23, W23, 1
    mov     W0, W24
    bl      func_p
    add     W24, W24, 1
    cbz     W0, Lprox
    fabs    D0, D0
    fsqrt   D0, D0
    fadd    D15, D15, D0
Lprox:
    b       L2
Lfim:
    fmov    D0, D15    // resultado
    ldp     W23, w24, [X29, 32]
    ldr     D15, [X29, 24]
    ldr     X22, [X29, 16]
    ldp     X29, X30, [SP], 48
    ret
```

4. Considerar a sub-rotina apresentada a seguir, com 2 argumentos na seguinte ordem:
   1. endereço-base de uma sequência de carateres ASCII (elementos do tipo `char`);
   2. número N de elementos (N é do tipo `unsigned int` com valor múltiplo de 16);

   Esta sub-rotina não retorna um valor, mas altera elementos da sequência em memória.

```
subrX:
L3:     cbz     W1, Lend
        ldrb    W2, [X0]
        cmp     W2, 'A'
```

```
        b.lo    Lnx
        cmp     W2, 'Z'
        b.hi    Lnx
        orr     W2, W2, 32
        strb    W2, [X0]
Lnx:    add     X0, X0, 1
        sub     W1, W1, 1
        b       L3
Lend:   ret
```

[40] (a) Explicar o algoritmo implementado pela sub-rotina `subrX` e a alteração sofrida pela sequência. Ter em atenção a informação sobre o código ASCII apresentada na tabela seguinte:

letra	código ASCII em binário	letra	código ASCII em binário
A	01000001	a	01100001
B	01000010	b	01100010
...	...	...	...
Z	01011010	z	01111010

Resposta: A sub-rotina possui um ciclo que é executado N vezes. Em cada iteração deste ciclo é lido um carater da sequência, verificando-se de seguida se o carater é uma letra maiúscula. Em caso afirmativo, a letra maiúscula é convertida em minúscula, somando 32 ao código ASCII da maiúscula, substituindo-se o carater em memória pelo resultante. Caso contrário, nenhuma alteração é feita.

Portanto, a sub-rotina converte as letras maiúsculas existentes numa sequência de carateres para letras minúsculas.

[30] (b) Escrever uma sub-rotina alternativa que aceite os mesmos argumentos, mas realize o mesmo processamento de forma mais eficiente através do recurso a instruções SIMD.

Resposta: Uma alternativa possível:

```
//X0: endreço da sequência de carateres
//W1: número de carateres
subrX_simd:
    // preparação
    mov     W2, 32
    dup     V0.16B, W2      // para tratatmento
    mov     W2, 'A'
    dup     V1.16B, W2      // limite inferior
    mov     W2, 'Z'
    dup     V2.16B, W2      // limite inferior
    lsr     W1, W1, 4       // dividir por 16 (número de iterações)
    // processamento
Lp:
    cbz     W1, Lquit
    ldr     Q4, [X0]
    sub     W1, W1, 1
```

```
    cmhs   V5.16B, V4.16B, V1.16B
    cmhs   V6.16B, V2.16B, V4.16B
    and    V5.16B, V5.16B, V6.16B
    and    V5.16B, V5.16B, V0.16B
    orr    V4.16B, V4.16B, V5.16B
    str    Q4, [X0], 16
    b      Lp
Lquit:
    ret
```