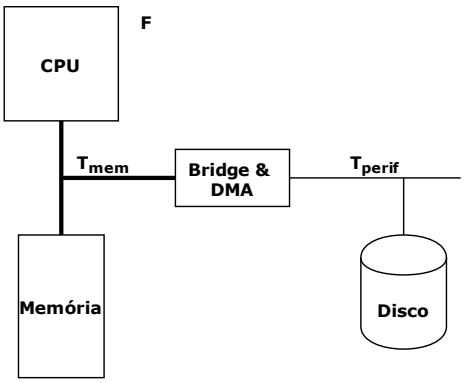
Este exame tem 4 questões, num total de 200 pontos. **RESPONDA A CADA QUESTÃO EM FOLHAS SEPARADAS.**

1. Considerar o sistema computacional indicado na figura.



Os parâmetros do sistema são os seguintes:

- Frequência F=1 GHz, CPI=1.
- Taxa máxima de transferência dos barramentos: T_{mem}=1 GB/s, T_{perif}=50 MB/s;
- Disco: velocidade de rotação=1000 RPM, setores de 4 KB, 120 setores/pista, t_{seek}=5 ms, $t_{overhead}$=0 s, taxa de transferência: 100 MB/s

A tarefa do sistema consiste em ler blocos de 128 KB do disco (situados em setores adjacentes) para a memória, cifrá-los e guardá-los em disco (mantendo o tamanho e setores adjacentes). As etapas são realizadas sequencialmente; leituras e escritas são feitas por DMA.

Cifrar os dados requer 10^6 instruções; configurar uma transferência de DMA e atender uma interrupção gerada pelo controlador de DMA requer 10^5 instruções.

Ignorar a possibilidade de existir contenção no barramento de memória.

[20] (a) Determinar o tempo de leitura ou escrita de/em disco para blocos de 128 KB guardados em setores adjacentes.

Resposta:

$$t = 5\,\text{ms} + \frac{128 \times 1 \times 10^3}{100 \times 1 \times 10^6}\text{s} + \frac{60}{1000 \times 2}\text{s} = 5\,\text{ms} + 1{,}28\,\text{ms} + 30\,\text{ms} = 36{,}28\,\text{ms}$$

[20] (b) Determinar o número máximo de tarefas que o sistema é capaz de executar por segundo.

Resposta: Tempo na transferência: $2 \times 36{,}28\,\text{ms}$ (leitura e escrita).
Disco: $\dfrac{1000}{36{,}28\,\text{ms}} = 27{,}56$ blocos/s
Taxa de dados que tem de ser suportada no barramento periférico:

$$27{,}56 \times 128\,\text{kB/s} = 3{,}53\,\text{MB/s} < 50\,\text{MB/s} = T_{\text{perif}} \qquad \text{(OK)}$$

Barramento de memória: OK porque 3,53 MB/s < 1 GB/s (tráfego para disco)
Barramento de memória (leitura/escrita de dados pelo CPU durante o processamento):

$$2 \times 128\,\text{kB/tarefa} = 256\,\text{kB/tarefa}$$

(em princípio, este fator também não afeta o desempenho)
CPU: 1 GHz e CPI= 1 $\rightarrow 1 \times 10^9$ instruções/s.

$$1 \times 10^6 + (2 \times 1 \times 10^5)\,\text{instruções} = 1{,}2 \times 10^6\,\text{instruções} \times 1\,\text{ns/instrução} = 1{,}2\,\text{ms}$$

(também não é o gargalo de desempenho)
1 tarefa demora = $2 \times 36{,}28\,\text{ms} + 1{,}2\,\text{ms} = 73{,}76\,\text{ms}$
Número total de tarefas: $\dfrac{1000}{73{,}76\,\text{ms/tarefa}} = 13{,}55\,\text{tarefas/s}$.
Verificação (barramento de memória):
$3{,}53\,\text{MB/s} + 13{,}55\,\text{tarefas/s} \times 256\,\text{kB/tarefa} = 6{,}95\,\text{MB/s} \ll 1\,\text{GB/s}$

[10] (c) Determinar a percentagem de tempo que o CPU necessita de dedicar a estas tarefas para as executar o mais rapidamente possível.
[Nota: se não resolveu a alínea (b), assuma que o resultado respetivo é 60.]

Resposta: O tempo de CPU gasto por uma tarefa: 1,2 ms (alínea anterior)
Em 1 segundo: $1{,}2\,\text{ms} \times 13{,}55 = 16{,}26\,\text{ms}$ de tempo de CPU $\rightarrow \frac{16{,}26}{1000} = 1{,}63\%$ do tempo.
Usando o valar indicado na nota:
calcular tempo de CPU gasto por uma tarefa $1{,}2\,\text{ms} \times 60 = 72\,\text{ms} \rightarrow \frac{72}{1000} = 7.2\,\%$ do tempo.

---

[50] 2. Dada uma sequência não ordenada de N valores sem sinal ($N > 0$), designa-se por subsequência crescente qualquer subsequência de elementos contíguos em que cada um é maior que o precedente.

Exemplo: A sequência [2; 4; 6; 5; 9; 8; 8] tem as seguintes subsequências crescentes:

[2; 4; 6] [5; 9] [8] [8]

Escrever em *assembly* a sub-rotina `Maxsumsubseq`, que, dada uma sequência de números inteiros sem sinal de 32 bits, retorna a maior de entre as somas dos elementos das suas subsequências crescentes. Ignorar a possibilidade de *overflow*. A sub-rotina poderia ser usada da seguinte forma:

```
extern int Maxsumsubseq(unsigned int v[], unsigned int N);
unsigned int vect[]={2, 4, 6, 5, 9, 8, 8};
main()
{  printf("%u\n", Maxsumsubseq(vect, 7));    }
```

O valor apresentado seria 14 (a soma dos dois elementos da 2ª subsequência crescente).

Resposta: Uma possível alternativa:

```
// X0: vetor
// W1: número de elementos
Maxsumsubseq:
    mov     W9, 0 // menor número; máximo inicial
```

```
    ldr     W10, [X0], 4     // 1º número
    sub     W1, W1, 1
    mov     W11, W10        // W10: soma parcial, W11: número anterior
 L11:
    cbz     W1, terminar
    ldr     W12,[X0], 4
    sub     W1, W1, 1
    cmp     W12, W11         // terminar subsequência? (W11 contém valor anterior)
    bls     fim_seq
    add     W10, W10, W12
    mov     W11, W12
    b       L11

fim_seq:
    cmp   W10, W9
    csel  W9, W10, W9, HI
    mov   W10, W12
    mov     W11, W12
    b   L11

terminar:
    cmp   w10, W9
    csel  W0, W10, W9, HI
    ret
```

[30] 3. Escrever sub-rotina `apply_func` em *assembly* que recebe uma sequência de números em vírgula flutuante (precisão dupla) e o respetivo comprimento (N), substituindo cada elemento x pelo resultado da expressão indicada abaixo:

$$x \mapsto f(x,i) - \frac{x}{3} \qquad i = 0, 1, \ldots N-1 : \text{índice do valor na sequência.}$$

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

```
double func(double x, int i);
```

A sub-rotina `apply_func` não tem resultado, mas altera os elementos da sequência em memória. O seu código deve invocar a sub-rotina `func` corretamente. Não usar declarações de constantes. **Não escrever a sub-rotina func.** Exemplo de utilização:

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

Resposta: Uma solução possível:

```
// X0: endereço base do vector
/// W1: nº de elementos
// valores a guardar na pilha:
// W20 (+16), W21 (+20), X22 (+24) e D2 (+32) (24 bytes -> 32 bytes + 16 bytes = 48 bytes)
```

```
apply_func:
        // criar frame e guardar valores
        stp       X29, X30, [SP, -48]!
        mov       X29, SP       // frame pointer
        stp       W20, W21, [X29, 16] // preservar registos
        str       X22, [X29, 24]

        // este registos não serão alterados por func
        mov       W20, W1       // número de elementos
        mov       W21, 0         // índice
        mov       X22, X0       // endereço
        // processar
ciclo:
        cbz       W20, final
        ldr       D0, [X22]
        mov       W3, 3
        scvtf     D2, W3
        fdiv      D2, D0, D2           // D2 = x/3
        str       D2, [X29, 32]   // preservar D2
        mov       W0, W21
        add       W21, W21, 1
        sub       W20, W20, 1
        bl        func       // resultado em D0

        ldr       D2, [X29, 32] // recuperar D2
        fsub      D0, D0, D2
        str       D0, [X22], 8
        b         ciclo

final:
        // recuperar valores de registos
        ldp       W20, W21, [X29, 16]
        ldr       X22, [X29, 24]
        ldp       W20, W21, [X29, 16]
        ldp       X29, X30, [SP], 48
        ret
```

---

4. Considerar a sub-rotina apresentada a seguir, com três argumentos na seguinte ordem:
   1. número N de elementos (N é do tipo `int` e tem valor múltiplo de 4);
   2. endereço-base da sequência A de N elementos do tipo `float`;
   3. endereço-base da sequência B de N elementos do tipo `float`.

```
subrotina:
     fsub   S0, S0, S0
L1:  cbz    W0, L2
     ldr    S1, [X1], 4
     ldr    S2, [X2], 4
```

```
        fadd  S2, S2, S2
        fsub  S1, S1, S2
        fabs  S1, S1
        fmax  S0, S0, S1
        sub   W0, W0, 1
        b     L1
L2:     ret
```

[40] (a) Explicar o algoritmo implementado pela sub-rotina e o significado do resultado.

Resposta:

A sub-rotina possui um ciclo que é executado N vezes. Em cada iteração deste ciclo é acedido um elemento de cada sequência, A_i e B_i, por ordem (i=0, ..., N-1). De seguida é calculado o valor da expressão $|A_i - 2 \times B_i|$ atualizando-se o registo `S0` com o valor máximo entre `S0` atual e o valor da expressão calculada.

A inicialização de `S0` com 0.0 garante que, no final do ciclo, o resultado da sub-rotina seja o valor máximo que a expressão $|A_i - 2 \times B_i|$ $(0 \leq i < N)$ pode atingir para as sequências A e B.

[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:

```
subrotina_simd:
     fsub  S0, S0, S0
L1x: cbz   W0, L22
     ldr   Q1, [X1], 16
     ldr   Q2, [X2], 16
     sub   W0, W0, 4
     fadd  V2.4S, V2.4S, V2.4S
     fsub  V1.4S, V1.4S, V2.4S
     fabs  V1.4S, V1.4S
     fmaxv S1, V1.4S
     fmax  S0, S1, S0
     b     L1x
L22: ret
```

Fim.