Este exame tem 7 questões, num total de 200 pontos. Responda em folhas separadas a cada um dos seguintes conjuntos de problemas: (1 e 2), (3 e 4), (5 e 6). O problema 7 deve ser respondido na folha de enunciado.

1. Considerar o seguinte programa, composto por dois ficheiros.

   Ficheiro `main.cpp`:

```
#include <iostream>
extern "C" void func(char seqA[], char seqB[], char seqC[]);
char stringA[] = "123456789", stringB[] = "abcde", stringC[30];
int main()
{
      func(stringA, stringB, stringC);
      std::cout << stringC << std::endl;
      return 0;
}
```

   Ficheiro `func.asm`:

```
1  include mpcp.inc
2      .code
3  func PROC C uses esi edi seqA: PTR BYTE,
4           seqB: PTR BYTE, seqC: PTR BYTE
5
6      mov   edi, seqC
7      mov   esi, seqA
8      mov   edx, seqB
9
10
11 @@:  mov   al, [esi]
12     .IF   al == 0
13           mov  ecx, edx
14           jmp  @F
15     .ENDIF
16      mov  [edi], al
17      inc  esi
18      inc  edi
19      mov  al, [edx]
20     .IF  al == 0
21         mov  ecx, esi
22         jmp  @F
23     .ENDIF
24      mov [edi], al
25      inc  edi
26      inc  edx
27      jmp  @B
28
29 @@:   mov  al, [ecx]
30      .IF  al != 0
31         mov  [edi], al
32         inc  edi
33         inc  ecx
34         jmp  @B
35       .ENDIF
36       ret
37 func ENDP
38 end
```

[20] (a) Considerar o ciclo composto pelas linhas 11 a 27. Para os dados de entrada indicados no ficheiro `main.cpp`, quantas vezes é este ciclo executado e quantas leituras de dados de memória são efetuadas durante a sua execução?

Explicar a finalidade do ciclo.

> **Resposta:** Nesta função, os registos `esi` e `edx` contêm o endereço inicial do primeiro e segundo argumentos, respetivamente (argumentos de entrada). O registo `edi` contém o apontador para o início da zona de memória onde ver ser guardado o resultado (terceiro parâmetro da sub-rotina).
>
> Em cada iteração, o ciclo das linhas 11–27 copia um caráter do primeiro argumento para a primeira posição livre do terceiro (e incrementa os respetivos apontadores) e

depois faz o mesmo para um caráter do segundo argumento. O efeito é construir no terceiro argumento uma sequência de carateres provindos alternadamente de cada um dos dois primeiros argumentos.

O ciclo termina quando uma das cadeias chega ao fim (i.e, quando é lido o caráter de código ASCII 0). Esse teste é feito na linha 12 para o primeiro argumento e na linha 30 para o segundo. Em qualquer dos casos a execução do ciclo termina, deixando no registo `ecx` o endereço da próxima posição livre do terceiro argumento.

O ciclo executa completamente tantas vazes quantos os carateres da sequência de entrada mais curta, que neste caso são 5. Como a sequência mais curta é a segunda, o programa inicia ainda mais uma iteração e copia um caráter da primeira sequência de entrada, mas já não copia nenhum da segunda, terminando portanto a meio da sexta iteração. Em cada iteração completa, o ciclo realiza duas *leituras de dados de memória* (linhas 11 e 29). Mesmo a sexta iteração realiza duas leituras, porque a terminação é provocada pelo fim da segunda sequência. Concluindo, são feitos $2 \times 6 = 12$ acessos de leitura de dados.

A finalidade do ciclo é criar na zona endereçada pelo terceiro argumento uma sequência de carateres provenientes alternadamente do primeiro e do segundo argumento. Para estes dados, o conteúdo da zona do resultado é `"1a2b3c4d5e6"`.

[Nota: A sub-rotina assume implicitamente que essa zona tem comprimento suficiente para o resultado e que está inicialmente preenchida por zeros.]

[20] (b) Indicar, justificando, o que é apresentado no monitor.

Resposta: O ficheiro `main.cpp` define três cadeias de carateres (`stringA, stringB` e `stringC`). [Nota: esta última está inicializada com o valor 0 em cada uma das 30 posições, já que é uma variável global.] A função `main` invoca a sub-rotina `func`, escrita em *assembly*. Esta sub-rotina é do tipo `void`, logo não retorna um valor. Em vez disso, a sub-rotina recebe os endereços da primeira posição de três cadeias de carateres: a relação entre as cadeias de carateres e os parâmetros `seqA, seqB` e `seqC` é estabelecida pela ordem das variáveis na invocação de `func`.

O efeito da sub-rotina `func` é alterar a zona de memória endereçada pelo terceiro argumento para conter, de forma entrelaçada, os carateres dos dois primeiros argumentos (cf. resposta à alinea anterior). Se um dos argumentos for mais comprido do que o outro, os carateres excedentes são copiados para o terceiro argumento pelo ciclo das linhas 29–34.

Para os dados do enunciado, a cadeia de carateres `stringC` vai ficar com o o valor `"1a2b3c4d5e6789"` (modificação via o endereço passado a `func`). O valor dessa variável é apresentado no monitor pela função principal usando `cout`. Logo, no monitor surge a sequência de carateres **1a2b3c4d5e6789** seguida de mudança de linha.

2. Relativamente à sub-rotina `SR` sabe-se que:

   - antes de a invocar `ESP=004AC07Ch`;
   - o estado da pilha após execução do prólogo é o apresentado ao lado (valores em hexadecimal).

Endereço	Conteúdo
004AC078	valor de ECX
004AC074	valor de ESI
004AC070	09FF15F0
004AC06C	valor de EBP
004AC068	valor de ESI

[15] (a) Escreva a linha de código com a invocação de `SR` que deu origem ao conteúdo da pilha.

Resposta:

A primeira instrução do prólogo escreve `EBP` na pilha, pelo que o valor que precede `EBP` se refere ao endereço de retorno da sub-rotina. Logo, os valores de `ECX` e `ESI` são os parâmetros com que a sub-rotina foi invocada.

Portanto, a invocação é: `invoke SR, ESI, ECX`

[10] (b) O valor de `EBP` após execução do prólogo é `004AC06Ch`. Justifique.

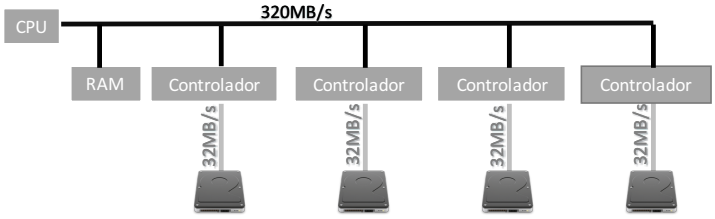
Resposta:

As duas primeiras instruções executadas no prólogo são:

```
push ebp
mov  ebp, esp
```

Quando a segunda instrução é executada, o endereço do topo da pilha (contido em `ESP`) é `004AC06Ch`. Logo, no final do prólogo `EBP=004AC06Ch`.

[20] 3. Considere o computador indicado na figura e que tem as seguintes caraterísticas:



- O CPU opera a 2GHz;
- O barramento de memória possui uma taxa de transferência de 320 MB/s;
- Ligados ao barramento de memória estão 4 controladores, cada um com o seu disco;
- O acesso aos discos é feito com uma largura de banda de 32 MB/s; o tempo médio de busca mais a latência de rotação é 6 ms;
- O acesso aos discos é feito em blocos de 128 kB, guardados em setores consecutivos;
- Em cada acesso, o programa do utilizador e o sistema operativo gastam, respetivamente, 1 milhão e 3 milhões de ciclos de relógio.

Determine qual dos recursos (CPU, barramento de memória e discos) limita o desempenho expresso em blocos processados por unidade de tempo. [Considere kB = 10^3 B, MB = 10^6 B.]

Resposta:

Programa: Leitura de 128kB + 1×10^6ciclos

S.O.: 2×10^6ciclos

CPU:

Tratamento de 1 bloco:

$$\frac{1 \times 10^6 + 3 \times 10^6}{2 \times 10^9} = 2\text{ms}$$

Por segundo: 500 blocos

Barramento de Memória:

$$\frac{320\text{MB/s}}{128\text{kB}} = 2500\text{blocos/s}$$

Discos:

Por disco:

$$6\text{ms} + \frac{128\text{kB}}{32\text{MB/s}} = 10\text{ms}$$

O que corresponde a 100 blocos por disco por segundo, no total temos então:

$$4 \times 100\text{blocos/s} = 400\text{blocos/s}$$

Como o CPU consegue processar 500 blocos/s e o barramento de memória suporta a transferência de 2500 blocos/s, são os discos que limitam o desempenho.

[20] 4. O disco de um computador transfere dados em grupos de 4 palavras (16 bytes cada). Este periférico é gerido por recurso a *polling* para realizar as transferências de dados. A operação de *polling* consome 600 ciclos e o CPU funciona a 2 GHz. Indique qual a taxa máxima de transferência que os discos podem ter para que a taxa de ocupação do CPU com a tarefa de *polling* não seja superior a 30 %. [Considere kB = 10^3 B, MB = 10^6 B.]

Resposta:

Número de ciclos por segundo:

$$\frac{x}{2 \times 10^9} = 3 \times 10^{-1} <=> x = 6 \times 10^8$$

Acessos por segundo:

$$\frac{6 \times 10^8}{600} = 10^6$$

Taxa de transferência:

4 palavras×16B = 64B

$$\frac{y}{64\text{B/acesso}} = 10^6 <=> y = 64 \times 10^6 = 64\text{MB/s}$$

[25] 5. Considere a função de variável real $f(x) = \frac{1}{\sqrt{1-x^2}}$. O domínio da função é]-1; 1[.

Implemente a sub-rotina `fx` que calcula o valor de $f(x)$. Caso x não pertença ao domínio deve ser devolvido o valor `0.0`.

O protótipo da sub-rotina é: `fx proto x:real8`

Resposta:

```
fx proc x:real8
    fld   x
    fmul  x
    fld1
    fcomi st(0), st(1)   ; 1>x^2 ?
    jbe   sai
    fxch                 ; st(0)=x^2, st(1)=1
    fsub  st(0), st(1)
    fchs                 ; st(0)=1-x^2, st(1)=1
    fsqrt
    fdiv
    jmp   @F
sai:fstp  st
    fstp  st
    fldz
@@: ret
fx endp
```

6. Cada uma das seguintes questões tem apenas uma resposta certa. Indique as respostas corretas **na folha de resposta** (e não na folha do enunciado).

[5] (a) Um sistema RAID é constituído por 20 discos de 1 TB e tem uma capacidade total útil de 16 TB. De que tipo de sistema RAID se trata?
A. RAID-0 B. RAID-1 **C. RAID-5 com 4 grupos** D. RAID-6 com 4 grupos

[5] (b) Inicialmente, o registo `BX` tem o valor `950Ah`. Qual das seguintes instruções altera esse valor para `6AF6h`?
A. `neg BX` B. `not BX` C. `xor BX, 0FFFFh` D. `and BX, 0FFFFh`

[5] (c) Qual é o valor correto dos indicadores *(flags)* `Carry (CF)`, `Zero (ZF)`, e `Sign (SF)` após as seguintes instruções?

```
mov al,00110011b
test al,2
```

A. `CF=1,ZF=0,SF=1` B. `CF=0,ZF=1,SF=0`
C. `CF=1,ZF=1,SF=1` **D.** `CF=0,ZF=0,SF=0`

[5] (d) Qual das instruções divide por 8 um número inteiro sem sinal guardado em `EBX`?
A. `shr EBX, 8` **B.** `shr EBX, 3` C. `sar EBX, 8` D. `sar EBX, 3`

[5] (e) A execução da linha de código `repnz scasd` altera sempre os registos:
A. `ECX` e `EAX` B. `EDI` e `EAX` **C.** `ECX` **e** `EDI` D. `EAX` e registo de *flags*

[5] (f) Qual é um tempo típico de acesso a uma memória DRAM?
A. 2 ns **B. 50 ns** C. 25 µs D. 10 ms

Nome (legível): _____

7. Uma sequência de elementos do tipo `sdword`, terminada pelo valor 100, contém valores de temperatura em ºC, recolhidos de minuto a minuto, no interior de uma câmara frigorífica. Pretende-se imprimir quantas vezes e por quanto tempo a temperatura esteve acima de um dado limiar. Por exemplo, se o limiar definido for -17 ºC, a sequência {-18, -17, -15, -13, -15, -18, -19, -16, -17, -15, -14, 100} representa um caso em que a temperatura esteve $3 + 1 + 2$ minutos acima de -17 ºC. O resultado a imprimir será então:

```
Limiar de temperatura: -17 C.
Minutos acima do limiar: 3 1 2. Total: 6 minutos.
```

[20] (a) Escrever a sub-rotina `seek PROTO lim:sdword, buf:ptr sdword` que conta quantos elementos consecutivos situados a partir do endereço `buf` são maiores que o limiar `lim`. Quando for encontrado o fim da sequência (valor 100) deve ser devolvido o valor -1.

Resposta:

```
seek proc uses esi lim:sdword, buf:ptr sdword
     mov  esi, buf
     xor  eax, eax       ;; inicia o contador a zero

@@:  mov  ecx, [esi]
     cmp  ecx, lim
     jle  fim            ;; sai se não está acima do limiar
     cmp  ecx, 100       ;; verifica se é o último valor
     jz   ult
     inc  eax            ;; incrementa contador
     add  esi, 4         ;; próximo valor da sequência
     jmp  @b

ult: and  eax,eax        ;; verifica se há contagem anterior
     jnz  fim            ;; nesse caso retorna a contagem
     mov  eax, -1
fim: ret
seek endp
```

[20] (b) Completar, nos locais assinalados, o programa principal, que imprime o resultado com o formato indicado no exemplo.

```
include mpcp.inc

seek proto lim:sdword, buf:ptr sdword

.data
limt  sdword -17
temps sdword -18,-17,-15,-13,-15,-18,-19,-16,-17,-15,-14,100
msg1  byte "Limiar de temperatura: %d C.",10,13,
           "Minutos acima do limiar:",0
msg2  byte " %d",0
msg3  byte ". Total: %d minutos.",10,13,0

.code
main: invoke printf, offset msg1, limt
      mov esi, offset temps  ;; endereço inicial
      xor ebx, ebx         ;; tempo total acima do limiar

ciclo:invoke seek, limt, esi
      cmp  eax, -1 ;; verifica se chegou ao fim da sequência
      jz   sai
      cmp  eax, 0      ;; se 0 não imprime
      jz   prox

      add  ebx, eax    ;; atualiza o tempo total
      push eax
      invoke printf, offset msg2, eax
      pop  eax
      jmp  salta

prox: add  eax, 1
salta:shl  eax, 2   ;; atualiza endereço para
      add  esi, eax          ;; a próxima iteração
      jmp  ciclo

sai:  invoke printf, offset msg3, ebx
      invoke _getch
      invoke ExitProcess, 0
end main
```