Atenção: Este exame tem 6 questões, num total de 200 pontos.

Análise de código

1. Considere o seguinte programa:

```
1  include \masm32\include\masm32rt.inc
2  func1 proto p: ptr sword, n: word, lim: sword
3  .data
4  T sword 3, -32, 7, 10, -5
5  k sword 4
6  .code
7  start:
8    invoke func1, offset T, lengthof T, k
9    add    eax, 0
10   print  ustr$(eax), 13, 10
11   inkey
12   exit
13
14 func1 proc uses ebx esi v: ptr sword, n: word, lim: sword
15      xor   ebx, ebx
16      mov   esi, v
17      movzx ecx, n
18 nx:  mov   ax, [esi]
19      cmp   ax, lim
20      jle   @F
21      call  func2
22      add   ebx, eax
23 @@:  add   esi, 2
24      loop  nx
25      mov   eax, ebx
26      ret
27 func1 endp
28
29 func2 proc
30   push ecx
31   imul ax
32   mov  ecx, eax
33   mov  ax, dx
34   sal  eax, 16
35   mov  edx, ecx
36   mov  ax, dx
37   pop  ecx
38   ret
39 func2 endp
40 end start
```

[10] (a) Descreva a sub-rotina `func2` e indique a sua funcionalidade.

> **Resposta:** A sub-rotina começa por preservar o conteúdo de `ECX`, guardando-o na pilha. Calcula o quadrado do valor de `AX`, resultado este que é armazenado em `DX, AX`. A metade menos significativa (`AX`) é copiada para `ECX` e a parte mais significativa é colocada em `EAX`. Depois, a parte menos significativa de `EAX` é ocupada pela metade menos significativa do produto, presente em `ECX`. Antes de terminar é reposto o valor em `ECX`.
>
> Em termos de funcionalidade, a sub-rotina calcula o quadrado de um número inteiro (de 16 bits). Não usa convenção *stdcall*: entrada em `AX` e saída em `EAX`.

[10] (b) Determine o valor apresentado no monitor no final da execução do programa. Justifique.

> **Resposta:** A sub-rotina `func1` calcula a soma dos quadrados de uma sequência de números inteiros superiores a um determinado valor. Como os valores nestas condições são o 7 e o 10, o valor resultante é $7^2 + 10^2 = 149$.

[10] (c) Indique, justificando, o valor que seria retornado pela sub-rotina `func1` se executasse `invoke func1, offset T, 3, 7`.

> **Resposta:** 0, porque só são percorridos os 3 primeiros valores e nenhum deles é superior ao limiar (7).

(d) Assuma que imediatamente antes da execução da instrução da linha 16 o valor de `ESP` é `18FF6C` e que o estado da pilha é o seguinte:

endereço (hex.)	conteúdo (hex.)
18FF84	00000004
18FF80	00000005
18FF7C	00403000
18FF78	0040103B
18FF74	0018FF94
18FF70	7EFDE000
18FF6C	00000000

Para cada uma das alíneas seguintes indique a resposta correcta.

[5] i. O valor contido no endereço `18FF6C` refere-se ao conteúdo de:
A. `EBX` B. `EAX` C. `EBP` **D.** `ESI`

> **Resposta:** D

[5] ii. O endereço de memória (em hexadecimal) a partir do qual se encontra codificada a instrução da linha 9 é:
A. `00403000` **B.** `0040103B` C. `7EFDE000` D. `0018FF94`

Resposta: B

Programação

2. Um número primo é um número inteiro positivo que só é divisível (i.e. tem resto zero) por 1 e por ele próprio. Um algoritmo simples para verificar se um número N é primo consiste em testar a divisibilidade de N por todos os inteiros no intervalo $[2, N/2]$.

[20] (a) Escreva uma sub-rotina para determinar se um número N passado como argumento é primo. A rotina deve retornar 1 se N for primo e 0 em caso contrário. O protótipo é:

`primo PROTO n:DWORD`

Resposta:

```
primo PROC USES ebx edi esi N: DWORD
   mov esi, N ; esi:=n
   mov edi, N
   shr edi, 1 ; edi:=n/2

   mov ebx, 1 ; ebx holds retval
   mov ecx, 2 ; ecx holds divisor
@@:
   xor edx, edx
   mov eax, esi
   div ecx
   cmp edx, 0
   je _notprime
   cmp ecx, edi
   jae _ret
   inc ecx
   jmp @B

_notprime:
   mov ebx, 0

_ret:
   mov eax, ebx
   ret
primo endp
```

[20] (b) Assuma que tem um vector `Vect` de números naturais (DWORDs) declarado em memória. Use a sub-rotina da alínea anterior para escrever um fragmento de código que coloque em EAX o menor e em EDX o maior dos números primos presentes em `Vect`. Por exemplo, se `Vect` = [1, 3, 7, 21, 23, 25], no final da execução `EAX=1` e `EDX=23`.

Resposta:

```
        mov ebx, 0FFFFFFFFh ; smallest prime
        xor edx, edx ; largest prime
        mov esi, offset vect
        mov ecx, lengthof vect
_loop:
        jecxz _end
        mov edi, dword ptr [esi]
        push ecx
        push edx
        invoke isprime, edi
        pop edx
        pop ecx
        cmp eax, 1
        jne _next
        cmp edi, ebx ; is smallest?
        jae @F
        mov ebx, edi
@@:
        cmp edi, edx ; is largest?
        jbe _next
        mov edx, edi
_next:
        add esi, type dword
        dec ecx
        jmp _loop

_end:
        mov eax, ebx
```

3. A função erf(x) tem a seguinte aproximação racional para $x \geq 0$:

$$\text{erf}(x) \approx 1 - \frac{1}{(1 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4)^4}$$

com

$$a_1 = 0{,}278393 \qquad a_2 = 0{,}230389 \qquad a_3 = 0{,}000972 \qquad a_4 = 0{,}078108$$

[20] (a) Apresente uma rotina que calcula o valor de erf(x) usando a aproximação indicada. Assuma que $x \geq 0$. O protótipo da rotina é:

```
erfpos PROTO argX: REAL8
```

Resposta: (Uma solução de entre várias possíveis.)

```
.data
a1 REAL8 0.278393
a2 REAL8 0.230389
a3 REAL8 0.000972
a4 REAL8 0.078108

.code
erfpos PROC argX:REAL8
    fld  argX
    fld  st(0)
    fmul st(0), st(1)
    fld  st(1)
    fmul st(0), st(1)
    fld  st(2)
    fmul st(0 ), st(1)
    fmul a4
    fxch
    fmul a3
    fadd
    fxch
    fmul a2
    fadd
    fxch
    fmul a1
    fadd
    fld1
    fadd
    fmul st(0), st(0)
    fmul st(0), st(0)
    fld1
    fdivr
    fld1
    fsubr
    ret
erfpos ENDP
```

[20] (b) A função erf(x) é ímpar: $\text{erf}(-x) = -\text{erf}(x)$.

Apresente uma rotina que calcula erf(x) para qualquer valor de x com recurso à rotina da alínea anterior. O protótipo da nova rotina é:

```
erf PROTO argX: REAL8
```

Resposta: (Uma solução de entre várias possíveis.)

```
erf PROC argX:REAL8
    LOCAL tmp:REAL8
    fld argX
    ftst ; compara com 0.0
    fstsw ax
    sahf
    jb negativo
    invoke erfpos, argX
    jmp fim
negativo:
    fchs
    fstp tmp
    invoke erfpos, tmp
    fchs
fim:
    ret
erf ENDP
```

Sistemas de entrada/saída

4. Foi encarregado de comprar um servidor novo para um banco. A carga de trabalho esperada consiste em pequenas operações do seguinte tipo: leitura de um registo de 2 kB do disco, alteração da informação do registo (e.g. actualização de saldo) e escrita dos 2 kB de volta no disco. O seu fornecedor de material informático tem os seguintes itens disponíveis:

   - Placa-mãe Asas com suporte para até 6 processadores, FSB partilhado de 12 GB/s, barramento de memória de 6 GB/s, 8 GB de memória RAM embutida, controlador de discos com barramento de 40 MB/s partilhado por até 10 discos e suporte RAID 0, 1, 4 e 5.
   - CPU Interu 3 GHz (1 núcleo) por 250 € cada.
   - Discos Seegate por 250 € cada (15000 rpm, taxa de transferência de 4 MB/s, tempo de busca de 2 ms, sectores de 4 kB).

   [Nota: Para simplificar, assuma em todos os casos K=10^3, M=10^6 e G=10^9.]

[10] (a) Qual o modo de RAID que deve ser escolhido para maximizar a taxa de operações de E/S realizadas por segundo pelos discos que comprar? Justifique de forma breve.

> **Resposta:** RAID 0 porque maximiza o paralelismo de leitura e escrita.

(b) Depois de comprar a placa-mãe fica com 1000 € para gastar em discos e processadores. Pretende-se determinar a melhor forma de aplicar essa quantia.

[5] i. Quantas operações de E/S por segundo são suportadas por um CPU Interu? Considere que cada operação (leitura do disco + processamento + escrita no disco) gasta 10 milhões de ciclos do CPU.

> **Resposta:** 3000/10 = 300 IOPS

[5] ii. Quantas operações completas (leitura do disco + escrita no disco) por segundo são suportadas pelos barramentos (FSB e memória) da placa-mãe?

> **Resposta:** FSB: 12GB/4KB = 3 x 10^6 IOPS, Mem. bus: 6GB/4KB = 1.5 x 10^6 IOPS

[10] iii. Quantas operações completas (leitura do disco + escrita no disco) por segundo são suportadas por um disco Seegate? Considere que as leituras e escritas são realizadas em sítios aleatórias do disco.

> **Resposta:** Tempo por operação =
> 2 x Tempo de acesso a 2 kB =
> 2 x Tempo de acesso a um sector =

2 x (T_busca + 0.5 x T_rot + T_trans) =
2 x (2 + 0.5 x 4 + 4 kB/4 MB) = 10 ms
Operações por segundo: 1000 ms/10 ms = 100 IOPS

[10] iv. Com base nos cálculos anteriores, quantos CPUs e quantos discos deverá comprar com 1000 € de forma a maximizar a performance do sistema de E/S?

Resposta: Com um CPU e 3 discos obtém um sistema perfeitamente equilibrado e capaz de 300 IOPS. O barramento do controlador de discos não é um factor limitativo: 40 MB/4 kB = 10000 IOPS

Escolha múltipla

5. Para as alíneas seguintes, indique a única resposta correcta.

[10] (a) Considere a seguinte sequência de instruções:

```
    cld
@@: lodsw
    inc  ax
    stosw
    loop @B
```

Considerando que os registos usados pelas várias instruções foram devidamente inicializados com o objectivo de incrementar os N elementos de 16 bits de uma sequência, tem-se após a execução:

A. `EDI` possui o valor inicial mais 4×N

B. `ESI` possui o valor inicial menos 2×N

C. `EDI` possui o valor inicial mais N

D. `ESI=EDI`

Resposta: D

[10] (b) Um programa lê e processa um vídeo de 1 GB em 30 s. O vídeo está armazenado em sectores contíguos num disco com uma taxa de transferência de 50 MB/s. O tempo de busca e latência de rotação podem ser ignorados. Se o CPU for trocado por um duas vezes mais rápido, qual será o novo tempo total de execução do programa?

A. 25 s B. 30 s C. 15 s D. 20 s

Resposta: A

[10] (c) Indique qual dos fragmentos de código calcula o valor da tangente da variável X.

A.
```
fld X
fsin
fld X
fcos
fdivr
```

B.
```
fld X
fcos
fld X
fsin
fdiv
```

C.
```
fld1
fld X
fcos
fdiv
fld x
fsin
```

D.
```
fld X
fsin
fld X
fcos
fdiv
```

Resposta: D

[10] (d) Considere a sequência de instruções que se segue.

```
mov eax, 1
or  eax, 40000004h
```

De entre as opções disponíveis, escolha a instrução que pode substituir a instrução `or eax, 40000004h` produzindo o mesmo valor em `EAX`.

A. `sub eax,4` B. `or ax,4` C. `xor eax,40000005h` **D.** `add eax,40000004h`

Resposta: D

Fim do enunciado.