**Curso:** LCC/LEI **Exame 1ª Chamada** 25/Jun/07
**Disciplina:** Sistemas de Computação **Duração (máx):** 2h30m

## Avisos

1. Este exame é constituído por 3 partes:
   - **Parte 1**: Representação de Informação – **obrigatória**,
     corresponde aos resultados de aprendizagem avaliados no Teste1;
   - **Parte 2**: Estrutura de um computador e execução de instruções – **obrigatória**,
     corresponde aos resultados de aprendizagem avaliados nos Testes2 e 3;
   - **Parte 3**: Classificativa – **opcional**,
     para definição da classificação final (10 a 20), apenas para os Aprovados,
     i.e., apenas para quem não falhou mais que 1 questão obrigatória.

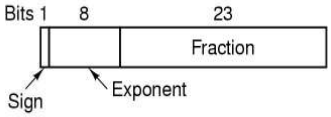
2. Duração do exame:
   **Parte 1 e Parte 3: 1h 50m**
   **Parte 2 e Parte 3: 2h 10m**
   **Apenas a Parte 3: 1h 30m**
   **Global: 2h 30m**

3. Apresente sempre o raciocínio ou os cálculos que efectuar; o não cumprimento desta regra equivale à não resolução do exercício. Use o verso das folhas do enunciado do exame como papel de rascunho.

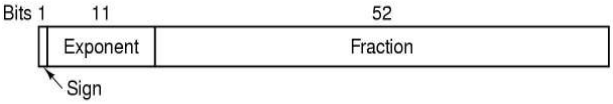
**Notas de apoio** (norma IEEE 754)

| | | | |
|---|---|---|---|
| Normalized | ± | 0 < Exp < Max | Any bit pattern |
| Denormalized | ± | 0 | Any nonzero bit pattern |
| Zero | ± | 0 | 0 |
| Infinity | ± | 1 1 1…1 | 0 |
| Not a number | ± | 1 1 1…1 | Any nonzero bit pattern |

Sign bit




(a)




Valor decimal de um fp em binário:
precisão simples, normalizado: $V = (-1)^S * (1.F) * 2^{E-127}$
precisão simples, desnormalizado: $V = (-1)^S * (0.F) * 2^{-126}$

| **Curso:** | **Nº** | **Nome** |
|---|---|---|

# Parte 1

**1.** **(A)** Considere a página de entrada na Web da disciplina de Sistemas de Computação (com a lista de avisos). O ficheiro que o *browser* acede tem o nome `default.html`. **Caracterize** o conteúdo desse ficheiro, **explicitando** a forma como os diversos tipos de informação são representados.

**2.** **(A)** Considere que o registo `%ax` (IA-32) contém o conjunto de bits correspondente ao valor `0xb8`. Sabendo que esse valor é um `short int` (codificado em complemento para 2), **represente** em decimal o valor que está lá guardado.

**3.** **(A)** Considere a representação de valores em vírgula flutuante, precisão simples, de acordo com a norma IEEE 754. **Represente** em binário, no formato especificado por esta norma, o maior valor inteiro ímpar que pode ser exactamente representado com esta notação.

| **Curso:** | **Nº** | **Nome** |
|---|---|---|

# Parte 2

Considere a figura fornecida com os códigos dum programa para PC (que procura números perfeitos).

**4.** **(A)** **Identifique** sucintamente os 3 níveis de abstracção dum computador que estão representados nessa figura, **indicando e caracterizando** os utilitários que fazem a conversão entre esses níveis.

**5.** **(A)** Analise os dados apresentados pelo depurador logo a seguir a uma paragem da execução do código no *breakpoint* introduzido a meio da função `perfeito`. **Indique**, justificando, em que instrução foi inserido o *breakpoint* (indique o endereço da instrução em código máquina).

.

**6.** **(A)** **Identifique** as principais diferenças entre as instruções em código simbólico (*assembly*) nas linhas **32** e **42**, para além da diferença nas designações dos registos.

| **Curso:** | **Nº** | **Nome** |
|---|---|---|

**7.** **(A)** Considere a execução da instrução (em *assembly*) na linha 15, após a descodificação da instrução, e assuma que o conteúdo dos registos é o indicado pelo depurador na figura anexa. Apresente, por ordem cronológica e em hexadecimal, toda a informação que circula apenas no barramento de dados.

**8.** **(A)** **Rescreva** aqui todas as instruções do código simbólico (*assembly*) que implementam a estrutura de controlo de ciclos na linha **45** do código fonte, devidamente anotadas, e apenas estas.

**9.** Analisando o código simbólico da função `perfeito`:

a) **(A)** **Comente** (explicando a funcionalidade) todas as instruções desde o início da função até à 1ª instrução do corpo da função (exclusive).

b) **(A)** **Indique** todas os registos e todas as células de memória que foram modificadas com a execução destas instruções.

| **Curso:** | **Nº** | **Nome** |
|---|---|---|

# Parte 3

Continue a considerar a figura fornecida com os códigos dum programa para PC (sobre números perfeitos).

**10.** **(R)** **Apresente** o quadro de activação na pilha (*stack frame*) da função `perfeito`, **indicando** claramente todos os campos pertinentes e respectivos endereços de início de cada campo.

**11.** **(R/B)** **Indique** em hexadecimal, justificando, os conteúdos das seguintes células de memória durante a execução do código da função `perfeito`: `0xbffff8e3` e `0xbffff8e4`.

| **Curso:** | **Nº** | **Nome** |
|---|---|---|

**12.** **(R/B)** **Indique** em hexadecimal, justificando, a localização das 4 células de memória que irão conter o 2º elemento do *array* declarado na função `perfeito`.

**13.** **(R)** A função `divisores` (linha **18** no código fonte; devolve um inteiro) é invocada várias vezes na execução deste programa. **Indique**, justificando, o valor em decimal que ela vai devolver da 1ª vez que for invocada.

**14.** **(B)** A instrução na linha **46** no código fonte é implementada num processador IA-32 com as instruções nas linhas **52** e **54** (no código simbólico). Considere a implementação desta mesma instrução numa arquitectura RISC, com a possibilidade de especificar 3 operandos no formato da instrução, e com apenas um único modo de endereçamento à memória (conteúdo de registo mais uma constante). **Apresente** o código gerado por um compilador para essa arquitectura RISC (use a sintaxe do IA-32).

**15.** **(R/B)** **Proponha**, justificando, alterações ao código fonte que irão certamente melhorar os tempos de execução da função `perfeito`, indicando as que terão maior e menor impacto.

| **Curso:** | **Nº** | **Nome** |
|---|---|---|

```c
// Este programa tenta achar o primeiro número perfeito maior do que 80.
// Um número é perfeito se for igual à soma dos seus divisores.


#define TAM        100

// Devolve verdadeiro se um número é perfeito
// Isto é, se é igual à sua soma

int e_perfeito(unsigned int *p, unsigned int n) {
        return *p == n - 1;
}

// Instancia o array div com os divisores de n
// Devolve o número de divisores de n

int divisores(unsigned int *div, unsigned int n) {
        unsigned int i;
        int j = 1;
        div[0] = 1; // O primeiro divisor é sempre 1

        for(i = 2; i < n; i++)
                if(n % i == 0) {
                        div[j] = i;
                        j++;
                }
        return j;
}

// Procura o primeiro número perfeito maior do que inicial

void perfeito(unsigned int *p, unsigned int inicial) {
        int i;
        unsigned int n;
        unsigned int div[TAM];
        int tam;

        // A condição de paragem do ciclo é o número ser igual à soma dos seus divisores

        for(n = inicial; !e_perfeito(p, n); n++) {
                 tam = divisores(div, n);
                // Calcular a soma dos divisores do número n
                *p = 0;
                for(i = 0; i < tam; i++)
                        *p += div[i];
        }
        // Neste momento *p contém o primeiro número perfeito maior do que inicial
}

main() {
        unsigned int p, n;
        n = 80;
        p = 0;

        perfeito(&p, n);

        printf("%u\n", p);
}
```

**********

```
(gdb) info register
eax            0x9      9
ecx            0x50     80
edx            0x1      1
ebx            0x50     80
esp            0xbffff720       0xbffff720
ebp            0xbffff8d8       0xbffff8d8
esi            0x0      0
edi            0xbffff8f4       -1073743628
eip            0x8048434        0x8048434
eflags         0x200246 2097734
cs             0x23     35
ss             0x2b     43
ds             0x2b     43
es             0x2b     43
fs             0x0      0
gs             0x0      0
```

```
080483b0 <divisores>:
 80483b0:       ...
 80483b4:       b9 02 00 00 00          mov    $0x2,%ecx
 80483b9:       8b 7d 08                mov    0x8(%ebp),%edi
 80483bd:       be 01 00 00 00          mov    $0x1,%esi
 80483c3:       8b 5d 0c                mov    0xc(%ebp),%ebx
 80483c6:       c7 07 01 00 00 00       movl   $0x1,(%edi)
 80483cc:       39 d9                   cmp    %ebx,%ecx
 80483ce:       73 13                   jae    80483e3 <divisores+0x33>
 80483d0:       31 d2                   xor    %edx,%edx
 80483d2:       89 d8                   mov    %ebx,%eax
 80483d4:       f7 f1                   div    %ecx
 80483d6:       85 d2                   test   %edx,%edx
 80483d8:       75 04                   jne    80483de <divisores+0x2e>
 80483da:       89 0c b7                mov    %ecx,(%edi,%esi,4)
 80483dd:       46                      inc    %esi
 80483de:       41                      inc    %ecx
 80483df:       39 d9                   cmp    %ebx,%ecx
 80483e1:       72 ed                   jb     80483d0 <divisores+0x20>
 80483e4:       89 f0                   mov    %esi,%eax
 80483e6:       ...
 80483e9:       c3                      ret

080483f0 <perfeito>:
 80483f0:       55                      push   %ebp
 80483f1:       89 e5                   mov    %esp,%ebp
 80483f3:       57                      push   %edi
 80483f4:       56                      push   %esi
 80483f5:       53                      push   %ebx
 80483f6:       81 ec ac 01 00 00       sub    $0x1ac,%esp
 80483fc:       8b 7d 08                mov    0x8(%ebp),%edi
 80483ff:       8b 5d 0c                mov    0xc(%ebp),%ebx
 8048402:       8d b4 26 00 00 00 00    lea    0x0(%esi),%esi
 8048409:       8d bc 27 00 00 00 00    lea    0x0(%edi),%edi
 8048410:       89 5c 24 04             mov    %ebx,0x4(%esp)
 8048414:       89 3c 24                mov    %edi,(%esp)
 8048417:       e8 74 ff ff ff          call   8048390 <e_perfeito>
 804841c:       85 c0                   test   %eax,%eax
 804841e:       89 c6                   mov    %eax,%esi
 8048420:       75 41                   jne    8048463 <perfeito+0x73>
 8048422:       89 5c 24 04             mov    %ebx,0x4(%esp)
 8048426:       8d 85 58 fe ff ff       lea    0xfffffe58(%ebp),%eax
 804842c:       89 04 24                mov    %eax,(%esp)
 804842f:       e8 7c ff ff ff          call   80483b0 <divisores>
 8048434:       c7 07 00 00 00 00       movl   $0x0,(%edi)
 804843a:       31 d2                   xor    %edx,%edx
 804843c:       39 c6                   cmp    %eax,%esi
 804843e:       7d 20                   jge    8048460 <perfeito+0x70>
 8048440:       31 c9                   xor    %ecx,%ecx
 8048442:       ...
 8048449:       ...
 8048450:       8b b4 95 58 fe ff ff    mov    0xfffffe58(%ebp,%edx,4),%esi
 8048457:       42                      inc    %edx
 8048458:       01 f1                   add    %esi,%ecx
 804845a:       39 c2                   cmp    %eax,%edx
 804845c:       7c f2                   jl     8048450 <perfeito+0x60>
 804845e:       89 0f                   mov    %ecx,(%edi)
 8048460:       43                      inc    %ebx
 8048461:       eb ad                   jmp    8048410 <perfeito+0x20>
 8048463:       81 c4 ac 01 00 00       add    $0x1ac,%esp
 8048469:       5b                      pop    %ebx
 804846a:       5e                      pop    %esi
 804846b:       5f                      pop    %edi
 804846c:       5d                      pop    %ebp
 804846d:       c3                      ret

08048470 <main>:
 8048470:       ...
 8048473:       b8 50 00 00 00          mov    $0x50,%eax
 8048478:       83 ec 18                sub    $0x18,%esp
 804847b:       83 e4 f0                and    $0xfffffff0,%esp
 804847e:       89 44 24 04             mov    %eax,0x4(%esp)
 8048482:       8d 45 fc                lea    0xfffffffc(%ebp),%eax
 8048485:       c7 45 fc 00 00 00 00    movl   $0x0,0xfffffffc(%ebp)
 804848c:       89 04 24                mov    %eax,(%esp)
 804848f:       e8 5c ff ff ff          call   80483f0 <perfeito>
 8048494:       c7 04 24 c4 85 04 08    movl   $0x80485c4,(%esp)
 804849b:       8b 45 fc                mov    0xfffffffc(%ebp),%eax
 804849e:       89 44 24 04             mov    %eax,0x4(%esp)
 80484a2:       e8 09 fe ff ff          call   80482b0 <_init+0x38>
 80484a7:       ...
 80484aa:       c3                      ret
```