Curso: LCC/LEI **Exame 2ª Chamada** 06/Jul/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.

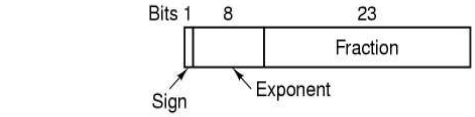
2. Duração do exame:
   Parte 1 e Parte 3: 1h 50m
   Parte 2 e Parte 3: 2h 10m
   Apenas a Parte 3: 1h 30m
   Global: 2h 30m

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 um ficheiro contendo o texto completo dos Lusíadas. Um colega seu disse-lhe que o ficheiro poderia utilizar como codificação o código ASCII de 7 bits. **Comente** essa afirmação **identificando** as formas que considerar adequadas para representar a informação em causa.

2. **(A)** Considere uma função que devolve um `float` (precisão simples e de acordo com a norma IEEE 754), colocando no registo `%eax` (IA-32) o conjunto de bits correspondente ao valor `0xc9`. **Represente** em decimal, o número que foi devolvido pela função.

3. **(A)** Considere que a função enunciada na alínea anterior foi invocada noutro módulo que não possuía informação sobre o protótipo da função. Assim sendo, o compilador assumiu erradamente que a função devolvia um valor do tipo `int` (codificado em complemento para 2). **Represente** em decimal, o valor correspondente.

Curso:	**Nº**	**Nome**

Parte 2

Considere a figura fornecida com os códigos dum programa para PC (IA-32), que conta o número de bits de uma linha que lê do teclado.

4. **(A)** **Identifique e caracterize** sucintamente todos os passos por que passou o programa fonte (em C) para chegar ao programa executável (apresentado em baixo nessa figura).

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 `contar_bits`. **Indique**, justificando, em que instrução foi inserido o *breakpoint* (indique o endereço da instrução em código máquina).

.

6. **(A)** **Explique** (em termos relacionados com o código fonte) o que faz a instrução em código simbólico (*assembly*) na linha **47**.

Curso:	**Nº**	**Nome**

7. **(A)** Considere que o PC acabou de executar a instrução na linha **14** (em *assembly*), que apanhou um *breakpoint* logo a seguir, e que o conteúdo dos registos é o indicado pelo depurador na figura anexa. **Apresente**, por ordem cronológica e em hexadecimal, toda a informação que irá circular apenas no barramento de endereços, assim que se mandar continuar a execução do código (a instrução na linha **15**).

8. **(A)** **Rescreva** todas as prováveis instruções do código simbólico (*assembly*) que faltam na listagem, devidamente anotadas, e que implementam a estrutura de controlo de ciclos na linha **7** do código fonte.

9. **(A)** Analisando o código simbólico da função `contar_bits`, **comente** (explicando a funcionalidade) todas as instruções desde o início da função até à 1ª instrução do corpo da função (exclusive).

10. **(A)** Analisando o código simbólico da função `contar_bits`, **indique**, justificando, qual o registo que o compilador reservou para representar a variável `c`.

Curso:	**Nº**	**Nome**

Parte 3

Continue a considerar a figura fornecida com os códigos dum programa para PC.

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

12. **(R/B)** **Indique** em hexadecimal, justificando, os conteúdos das seguintes células de memória durante a execução do código da função `contar_bits`: `0xbffff64f` e `0xbffff650`.

Curso:	**Nº**	**Nome**

13. **(B)** **Indique** em hexadecimal, justificando, a localização da célula de memória que irá conter o elemento do *array* (declarado na `main`), que contém o código ASCII de <`space`>, considerando a informação disponibilizada na figura anexa.

14. **(R/B)** **Explique** porque o compilador gerou as instruções em código simbólico das linhas **28** e **30**.

15. **(B)** Considere uma variante da instrução na linha **15** no código simbólico: `mov (%esi, %edi, 4),%ebx`. Considere agora 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 que seria gerado por um compilador para essa arquitectura RISC (use a sintaxe do IA-32).

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

Curso:	**Nº**	**Nome**

```
1.  #include <stdio.h>

2.  #define SIZE    512

3.  int contar_bits(char *buf, unsigned int *dig) {
4.      int i, j;

5.      for(i = 0; i < strlen(buf); i++) {
6.          char c = buf[i];
7.          for(j = 0; j < 8; j++)
8.              *dig += (c >> j) & 1;
9.      }
10. }

11. int main(int argc, char **argv) {
12.     unsigned int dig = 0;
13.     char buf[SIZE];

14.     fgets(buf, SIZE, stdin);
15.     contar_bits(buf, &dig);
16.     printf("%d\n", dig);

17.     return 0;
18. }
```

```
1.  08048420 <contar_bits>:
2.  8048420:    55                      push   %ebp
3.  8048421:    89 e5                   mov    %esp,%ebp
4.  8048423:    57                      push   %edi
5.  8048424:    56                      push   %esi
6.  8048425:    53                      push   %ebx
7.  8048426:    31 f6                   xor    %esi,%esi
8.  8048428:    83 ec 0c                sub    $0xc,%esp
9.  804842b:    8b 7d 08                mov    0x8(%ebp),%edi
10. 804842e:    89 f6                   mov    %esi,%esi
11. 8048430:    89 3c 24                mov    %edi,(%esp)
12. 8048433:    e8 e8 fe ff ff          call   8048320 <_init+0x38>
13. 8048438:    39 c6                   cmp    %eax,%esi
14. 804843a:    73 2b                   jae    8048467 <contar_bits+0x47>
15. 804843c:    0f be 1c 3e             movsbl (%esi,%edi,1),%ebx
16. 8048440:    31 c9
17. 8048442:    8b 45 0c                mov    0xc(%ebp),%eax
18. 8048445:    8b 10                   mov    (%eax),%edx
19. 8048447:    89 f6                   mov    %esi,%esi
20. 8048449:    8d bc 27 00 00 00 00    lea    0x0(%edi),%edi
21. 8048450:    89 d8                   mov    %ebx,%eax
22. 8048452:    d3 f8                   sar    %cl,%eax
23. 8048454:    83 e0 01                and    $0x1,%eax
24. 8048457:    41
25. 8048458:    01 c2                   add    %eax,%edx
26. 804845a:    83 f9 07
27. 804845d:    7e f1
28. 804845f:    8b 45 0c                mov    0xc(%ebp),%eax
29. 8048462:    46                      inc    %esi
30. 8048463:    89 10                   mov    %edx,(%eax)
31. 8048465:    eb c9                   jmp    8048430 <contar_bits+0x10>
32. 8048467:    83 c4 0c                add    $0xc,%esp
33. 804846a:    5b                      pop    %ebx
34. 804846b:    5e                      pop    %esi
35. 804846c:    5f                      pop    %edi
36. 804846d:    5d                      pop    %ebp
37. 804846e:    c3                      ret
38. 804846f:    90                      nop
```

```
08048470 <main>:
8048470:       55                      push   %ebp
8048471:       31 d2                   xor    %edx,%edx
8048473:       89 e5                   mov    %esp,%ebp
8048475:       81 ec 28 02 00 00       sub    $0x228,%esp
804847b:       a1 04 97 04 08          mov    0x8049704,%eax
8048480:       83 e4 f0                and    $0xfffffff0,%esp
8048483:       89 5d fc                mov    %ebx,0xfffffffc(%ebp)
8048486:       8d 9d f8 fd ff ff       lea    0xfffffdf8(%ebp),%ebx
804848c:       89 44 24 08             mov    %eax,0x8(%esp)
8048490:       b8 00 02 00 00          mov    $0x200,%eax
8048495:       89 95 f4 fd ff ff       mov    %edx,0xfffffdf4(%ebp)
804849b:       89 1c 24                mov    %ebx,(%esp)
804849e:       89 44 24 04             mov    %eax,0x4(%esp)
80484a2:       e8 69 fe ff ff          call   8048310 <_init+0x28>
80484a7:       89 1c 24                mov    %ebx,(%esp)
80484aa:       8d 85 f4 fd ff ff       lea    0xfffffdf4(%ebp),%eax
80484b0:       89 44 24 04             mov    %eax,0x4(%esp)
80484b4:       e8 67 ff ff ff          call   8048420 <contar_bits>
80484b9:       c7 04 24 f4 85 04 08    movl   $0x80485f4,(%esp)
80484c0:       8b 85 f4 fd ff ff       mov    0xfffffdf4(%ebp),%eax
80484c6:       89 44 24 04             mov    %eax,0x4(%esp)
80484ca:       e8 71 fe ff ff          call   8048340 <_init+0x58>
80484cf:       8b 5d fc                mov    0xfffffffc(%ebp),%ebx
80484d2:       31 c0                   xor    %eax,%eax
80484d4:       89 ec                   mov    %ebp,%esp
80484d6:       5d                      pop    %ebp
80484d7:       c3                      ret
```

```
(gdb) x /10xb ....
0x........:     0x6f    0x6c    0x61    0x20    0x6d    0x75    0x6e    0x64
0x........:     0x6f    0x0a
```

```
(gdb) info registers
eax            0x0      0
ecx            0x5      5
edx            0x8      8
ebx            0x6c     108
esp            0xbffff630       0xbffff630
ebp            0xbffff648       0xbffff648
esi            0x1      1
edi            0xbffff670       -1073744272
eip            0x8048450        0x8048450
eflags         0x200293 2097811
cs             0x23     35
ss             0x2b     43
ds             0x2b     43
es             0x2b     43
fs             0x0      0
gs             0x0      0
```