Notas:
1. Para cada uma das **5 questões de resposta satisfatória obrigatória**, numeradas de **1** a **5**, são-lhe oferecidas duas hipóteses para responder e/ou comentar; deverá optar por responder a **apenas uma** delas.
3. Para cada uma das hipóteses que optar, deverá apresentar **a justificação da solução**, incluindo o raciocínio ou os cálculos que efectuar.
3. Correção de cada questão: não-satisfaz (**0**), satisfaz com erros (**0.8**), certa com falhas (**1.0**) e completamente certa (**1.2**).

1.

a) Pretende-se codificar números romanos (de 1 até 3.999), usando os símbolos I, V, X, L, C, D e M. Sugira uma codificação binária para esses símbolos e apresente a forma como se representa o número 675 (DCLXXV)

b) Um ficheiro HTML tem armazenada a data de hoje, com conteúdo e formato iguais aos que se encontram no cabeçalho desta prova (ver canto superior direito desta página). Diga quantos bits são necessários para o dia do mês.

2.

a) Considere o seguinte fragmento de código em C:

```
int i=0;
while (i!=8) {
  j=i+j;
  i--;
}
```

Complete o mesmo fragmento compilado para *assembly* e introduza comentários.

```
        movl   ____, %ebx
.L3:    cmpl   $8, ____
        je .   L2
        addl   ____, %ecx
        _____  _____
        jmp    .L3
.L2:
```

b) Represente a estrutura genérica de código C a que corresponde o seguinte pedaço de código *assembly*:

```
.L1:
        # bloco de código
        subl $2, %ebx
        cmpl $10,%ebx
        jge .L1
```

Nota: Para as seguintes 3 questões escolha apenas uma afirmação, indique se é **Verdadeira** ou **Falsa**, e justifique a sua resposta.

3. Considere que o registo `%ax` (IA-32) contém um `short int` (codificado em complemento para 2).

a) Se calcular a soma das distâncias em metros de Braga-Barcelos e Barcelos-Esposende, e colocar o resultado nesse registo, fico com uma distancia negativa (<0).

b) Se nesse registo estiver o valor `0xd8`, então ele contém o valor `-40` em decimal.

4. Considere a norma IEEE 754 para representar valores reais de precisão simples (com 8 bits para o expoente em excesso de 127; não esquecer os casos de excepção). O valor decimal de um nº normalizado representado com este formato vem dado por $\mathbf{V= (-1)^S * 1.F * 2^{(Exp-127)}}$

a) Para representar neste formato de precisão simples a massa do electrão em kg (da ordem de grandeza de $\mathbf{10^{-30}}$) é necessário desnormalizar o valor da massa.

b) O operando $\mathbf{11111111\ 10000000\ 00000000\ 00000000_2}$ representa exatamente o valor $\mathbf{-2^{128}}_{10}$ que é aproximadamente $\mathbf{-1/4*10^{39}}_{10}$

5. Considere a execução duma instrução do IA-32(*little endian*) no corpo da função `conta_ai` (ver figura anexa) representada em *assembly* por:

```
movl   -20(%ebp,%edx,4),%ebx.
```

Considere que a instrução em binário contém 4 bytes (`8b`, `06`, `1c`, `f6`) e que está em memória a partir do endereço `0x80483b3` (estes valores não são verdadeiros).
Considere os conteúdo de registos indicados na figura anexa (obtidos no GDB).

Considere ainda este conteúdo das 16 células no topo da *stack* (endereços por ordem crescente, em hexadecimal): `00, 00, 00, 00, 88, 00, 01, 00, 00, 00, 00, 00, 00, 00, 00, 00`

a) Esta instrução distingue-se de `leal -20(%ebp,%edx,4),%ebx` porque nesta só é preciso aceder à memória para ir buscar o valor dos registos `%ebp` e `%edx`.

b) Após a descodificação da instrução, toda a informação que circula no barramento de endereços é:
`0xbfffe878, 0x00000001, 0xbfffe878, 0x80484c0`

Programa do teste de SC de 3-junho-2011:

```
main()
{
 printf("Total: %d\n",conta_ai("a UC Sistemas
                 de Computacao e' cool!"));
 return 0;
}

conta_ai(char *s)
{
 int i, count=0;
 for (i=0; s[i]!='\0'; i++)
   if (s[i]=='a' || s[i]=='i')
     count++;
 return (count);
}
```

Após compilado para *assembly* sem otimização:

```
conta_ai:
    pushl  %ebp
    movl   %esp, %ebp
    subl   $8, %esp
    movl   $0, -8(%ebp)
    movl   $0, -4(%ebp)
.L3:
    movl   -4(%ebp), %eax
    addl   8(%ebp), %eax
    cmpb   $0, (%eax)
    jne    .L6
    jmp    .L4
.L6:
    movl   -4(%ebp), %eax
    addl   8(%ebp), %eax
    cmpb   $97, (%eax)
    je     .L8
    movl   -4(%ebp), %eax
    addl   8(%ebp), %eax
    cmpb   $105, (%eax)
    je     .L8
    jmp    .L5
.L8:
    leal   -8(%ebp), %eax
    incl   (%eax)
.L5:
    leal   -4(%ebp), %eax
    incl   (%eax)
    jmp    .L3
.L4:
    movl   -8(%ebp), %eax
    leave
    ret
```

Dados obtidos após uma paragem no corpo da função:

```
(gdb) info registers
eax            0xbfffe924   -1073747676
ecx            0xb97677c9   -1183418423
edx            0x1          1
ebx            0x80484c0    134513856
esp            0xbfffe874   0xbfffe874
ebp            0xbfffe878   0xbfffe878
esi            0x573ca0     5717152
edi            0x0          0
eip            0x80483b3    0x80483b3 <conta_ai+7>
eflags         0x296        [ PF AF SF IF ]
cs             0x73         115
ss             0x7b         123
ds             0x7b         123
es             0x7b         123
fs             0x0          0
gs             0x33         51
```

Executável desmontado, depois de ter sido compilado com otimização `-O2`:

```
 [lei-1011@sc doc]$ objdump -d a.out
...
08048384 <main>:
 8048384:  55            push   %ebp
 8048385:  89 e5         mov    %esp,%ebp
 8048387:  83 ec 08      sub    $0x8,%esp
 804838a:  83 e4 f0      and    $0xfffffff0,%esp
 804838d:  83 ec 0c      sub    $0xc,%esp
 8048390:  68 c0 84 04 08   push   $0x80484c0
 8048395:  e8 12 00 00 00   call   80483ac
 804839a:  5a            pop    %edx
 804839b:  59            pop    %ecx
 804839c:  50            push   %eax
 804839d:  68 e0 84 04 08   push   $0x80484e0
 80483a2:  e8 19 ff ff ff   call   80482c0
                               <printf@plt>
 80483a7:  31 c0         xor    %eax,%eax
 80483a9:  c9            leave
 80483aa:  c3            ret
 80483ab:  90            nop

080483ac <conta_ai>:
 80483ac:  55            push   %ebp
 80483ad:  89 e5         mov    %esp,%ebp
 80483af:  53            push   %ebx
 80483b0:  8b 5d 08      mov    0x8(%ebp),%ebx
 80483b3:  8a 03         mov    (%ebx),%al
 80483b5:  31 c9         xor    %ecx,%ecx
 80483b7:  31 d2         xor    %edx,%edx
 80483b9:  84 c0         test   %al,%al
 80483bb:  74 13         je     80483d0
 80483bd:  8d 76 00      lea    0x0(%esi),%esi
 80483c0:  3c 61         cmp    $0x61,%al
 80483c2:  74 14         je     80483d8
 80483c4:  3c 69         cmp    $0x69,%al
 80483c6:  74 10         je     80483d8
 80483c8:  42            inc    %edx
 80483c9:  8a 04 1a      mov  (%edx,%ebx,1),%al
 80483cc:  84 c0         test   %al,%al
 80483ce:  75 ??         jne    80483c0
 80483d0:  89 c8         mov    %ecx,%eax
 80483d2:  5b            pop    %ebx
 80483d3:  c9            leave
 80483d4:  c3            ret
 80483d5:  8d 76 00      lea    0x0(%esi),%esi
 80483d8:  41            inc    %ecx
 80483d9:  eb ed         jmp    80483c8
```

Comentários à resolução esperada desta prova

1.

a) Pretende-se codificar números romanos (de 1 até 3.999), usando os símbolos I, V, X, L, C, D e M. Sugira uma codificação binária para esses símbolos e apresente a forma como se representa o número 675 (DCLXXV)

Havendo no total 7 símbolos, uma forma compacta de codificar estes símbolos seria usar 3 bits, propondo depois uma codificação para cada símbolo. O resto da resolução não oferece dificuldades.

1.

b) Um ficheiro HTML tem armazenada a data de hoje, com conteúdo e formato iguais aos que se encontram no cabeçalho desta prova (ver canto superior direito desta página). Diga quantos bits são necessários para o dia do mês.

A principal fonte de informação aqui é a indicação do tipo de ficheiro que contém esta informação: HTML.
Como foi referenciado nas aulas e distribuído em slides, um ficheiro HTML é um ficheiro de texto puro, onde cada carater é representado por uma extensão para 8 bits do código ASCII original.
Ora os algarismos com o dia do mês são também carateres, e não são precisos mais que 2 algarismos.
Logo, para representar o dia do mês são precisos 2 *bytes* ou 16 bits.
Infelizmente, uma grande percentagem das provas corrigidas continha disparates graves nesta questão, que não podem ser admissíveis a quem queira ter sucesso na UC.

2.

Resolução quase imediata, feita sem grande dificuldade por quase todos quantos tentaram seriamente.

3. Considere que o registo `%ax` (IA-32) contém um `short int` (codificado em complemento para 2).

Informação útil e pertinente que não foi devidamente analisada por uma grande maioria de estudantes: a dimensão do registo (16 bits) e a dimensão do tipo de variável (tb 16 bits no IA-32).
Não é admissível ignorar esta informação ou considerar que o valor é apenas de 8 bits.

a) Se calcular a soma das distâncias em metros de Braga-Barcelos e Barcelos-Esposende, e colocar o resultado nesse registo, fico com uma distancia negativa (<0).

Sendo um valor de 16 bits, a gama de representação em complemento para 2 está em $[-2^{15}, 2^{15}[$, i.e., no intervalo [-32k, 32k[.
Como o resultado desta adição conduz a um valor da ordem de grandeza dos 40k, se ele for representado com 16 bits, vai ter o bit mais à esquerda a 1, o que significa que é um valor negativo na representação em complemento para 2, em consequência do *overflow* ocorrido.

3. Considere que o registo `%ax` (IA-32) contém um `short int` (codificado em complemento para 2).

b) Se nesse registo estiver o valor `0xd8`, então ele contém o valor `-40` em decimal.

Para representar em 16 bits este valor com 8 bits, acrescentam-se 8 bits que não alterem o seu valor, i.e., **à esquerda e zeros**. Logo, este valor em 16 bits e em complemento para 2 representa uma quantidade positiva, e como tal não pode ser `-40`.

4. Considere a norma IEEE 754 para representar valores reais de precisão simples (com 8 bits para o expoente em excesso de 127; não esquecer os casos de excepção). O valor decimal de um nº normalizado representado com este formato vem dado por $\mathbf{V= (-1)^S * 1.F * 2^{(Exp-127)}}$

a) Para representar neste formato de precisão simples a massa do electrão em kg (da ordem de grandeza de 10^{-30}) é necessário desnormalizar o valor da massa.

Só será necessário desnormalizar se ele for tão pequeno que não possa ser representado no formato normalizado.
Assim calcula-se o menor normalizado que se pode representar com esta norma, que dá 2^{-126}, que por sua vez é $\sim(1/64)\times10^{-36}$. Logo, sendo este valor muito menor que a massa do electrão, é possível representá-la em formato normalizado.

4. Considere a norma IEEE 754 para representar valores reais de precisão simples (com 8 bits para o expoente em excesso de 127; não esquecer os casos de excepção). O valor decimal de um nº normalizado representado com este formato vem dado por $\mathbf{V= (-1)^S * 1.F * 2^{(Exp-127)}}$

b) O operando `11111111 10000000 00000000 00000000`$_2$ representa exatamente o valor $-2^{128}{}_{10}$ que é aproximadamente $-1/4*10^{39}{}_{10}$

Olhando para os 8 bits do expoente (tudo "1"s), rapidamente se conclui que este valor binário representa uma das excepções da norma IEEE, a que indica que representará ±∞ ou NaN. Como neste caso o campo da mantissa é zero, é a 1ª opção.

5. Considere a execução duma instrução do IA-32(*little endian*) no corpo da função `conta_ai` (ver figura anexa) representada em *assembly* por:

```
movl    -20(%ebp,%edx,4),%ebx.
```

Considere que a instrução em binário contém 4 bytes (`8b, 06, 1c, f6`) e que está em memória a partir do endereço `0x80483b3` (estes valores não são verdadeiros).
Considere os conteúdo de registos indicados na figura anexa (obtidos no GDB).
Considere ainda este conteúdo das 16 células no topo da *stack* (endereços por ordem crescente, em hexadecimal): `00, 00, 00, 00, 88, 00, 01, 00, 00, 00, 00, 00, 00, 00, 00, 00`

Comentários iniciais a este problema:
- a informação a castanho, em cima, é desnecessária para a resolução deste problema;
- o banco de registos é um dos componentes essenciais do CPU, portanto a simples referência a que o CPU tem de ir à memória buscar/guardar valores de registos, é um disparate tão grave que por si só deveria impedir que algum alunos passasse à UC;
- sempre que numa instrução em *assembly* se indica o nome de um registo, isto significa que, ao nível do *hardware*, o CPU necessita de saber onde se encontra esse registo no banco de registos (se é o 1º, 2º, 3º, ou outro), e a isso se poderia designar o "endereço de um registo", mas dentro do banco de registos; que é diferente do "valor do registo", que é o seu conteúdo, o qual poderá ser o valor de uma variável (um inteiro, por exemplo), ou o apontador ou a localização de uma variável (um endereço de memória); misturar estes 2 conceitos - "endereço de um registo" e "valor do registo" - é um erro grave, e infelizmente muito comum.

a) Esta instrução distingue-se de `leal -20(%ebp,%edx,4),%ebx` porque nesta só é preciso aceder à memória para ir buscar o valor dos registos `%ebp` e `%edx`.

O primeiro erro grave nesta afirmação já foi referido: não se vai à memória buscar valores de registos!
E depois espera-se que se faça a distinção entre as 2 instruções, de maneira sucinta e clara: a instrução `mov` tem de aceder à memória para buscar/guardar um dos operandos (neste caso seria buscar), cujo endereço de memória está especificado na instrução; a instrução `lea` apenas calcula o endereço do 1º operando e carrega-o no operando destino, não havendo qualquer acesso à memória.

5. Considere a execução duma instrução do IA-32(*little endian*) no corpo da função `conta_ai` (ver figura anexa) representada em *assembly* por:

```
movl    -20(%ebp,%edx,4),%ebx.
```

Considere que a instrução em binário contém 4 bytes (`8b, 06, 1c, f6`) e que está em memória a partir do endereço `0x80483b3` (estes valores não são verdadeiros).
Considere os conteúdo de registos indicados na figura anexa (obtidos no GDB).
Considere ainda este conteúdo das 16 células no topo da *stack* (endereços por ordem crescente, em hexadecimal): `00, 00, 00, 00, 88, 00, 01, 00, 00, 00, 00, 00, 00, 00, 00, 00`

b) Após a descodificação da instrução, toda a informação que circula no barramento de endereços é:
`0xbfffe878, 0x00000001, 0xbfffe878, 0x80484c0`

O primeiro erro grave nesta afirmação já foi referido em cima: não se vai à memória buscar valores de registos! Logo os 1º, 2º e último valores não deveriam passar pelo barramento de endereços!
Apenas deveria circular no barramento de endereços, o endereço especificado no 1º operando, i.e., o resultado do cálculo da expressão indicada na instrução, `-20(%ebp,%edx,4)`, o que feitas as contas deveria dar `0xbfffe868`; não esquecer que o valor `-20` está na base decimal, e que os outros valores estão em hexadecimal...
Atenção também ao enunciado, quando afirma "Após a descodificação da instrução"; se a instrução já foi descodificada, não se vai à memória buscá-la depois, como alguns afirmaram...