Prova P1 **Disciplina: Organização de Computadores**

Professor: Ney Laert Vilar Calazans

Aluno: 15/abril/2016

Lista de associação de números e mnemônicos para os registradores do MIPS

Número (Decimal)	Nome
0	$zero
1	$at
2	$v0
3	$v1
4	$a0
5	$a1
6	$a2
7	$a3
8	$t0
9	$t1
10	$t2
11	$t3
12	$t4
13	$t5
14	$t6
15	$t7

Número (Decimal)	Nome
16	$s0
17	$s1
18	$s2
19	$s3
20	$s4
21	$s5
22	$s6
23	$s7
24	$t8
25	$t9
26	$k0
27	$k1
28	$gp
29	$sp
30	$fp
31	$ra

1. (3,0 pontos) Montagem/Desmontagem de código objeto. Abaixo se mostra parte de uma listagem gerada pelo ambiente MARS como resultado da montagem de um programa. Pede-se: (a) Substituir as triplas interrogações pelo texto que deveria estar em seu lugar (existem 6 triplas ???). Em alguns casos, isto implica gerar código objeto, enquanto em outros implica gerar código intermediário e/ou código fonte. Caso uma instrução seja de salto, expresse o exato endereço para onde ela salta (em hexa ou com o rótulo associado à linha), caso isto seja parte das interrogações.

   Dica 1: Dêem muita atenção ao tratamento de endereços e rótulos.
   Dica 2: Tomem muito cuidado com a mistura de representações numéricas: hexa, binário, complemento de 2, etc.
   Obrigatório: Mostre os desenvolvimentos para obter os resultados, justificando.

```
        Endereço   Cód.Objeto  Cód.Intermediário          Cód. Fonte
[1]     0x00400068 0x8fa80004  lw $8,0x00000004($29)    59 gera_lista:lw $t0,4($sp)
[2]     0x0040006c ???         lw $9,0x00000008($29)     60          lw   $t1,8($sp)
[3]     0x00400070 0x00005021  addu $10,$0,$0           61         addu  $t2,$zero,$zero
[4]     0x00400074 0x1120000e  beq $9,$0,0x0000000e     64 l_gera:beq   $t1,$zero,end_gera
[5]     0x00400078 0x8d0b0000  lw $11,0x00000000($8)    66          lw   $t3,0($t0)
[6]     0x0040007c 0x240c0000  addiu $12,$0,0x00000000 67          li   $t4,0
[7]     0x00400080 0x11400008  beq $10,$0,0x00000008    68          beq  $t2,$zero,prim
[8]     0x00400084 0xafcafffc  sw $10,0xfffffffc($30)   69          sw   $t2,-4($fp)
[9]     0x00400088 0x27ca0008  addiu $10,$30,0x0000008 70              ???
[10]    0x0040008c 0x27de0008  addiu $30,$30,0x0000008 72 cont:  addiu $fp,$fp,8
[11]    0x00400090 0xafcbfff8  sw $11,0xfffffff8($30)   73          sw   $t3,-8($fp)
[12]    0x00400094 0xafccfffc  sw $12,0xfffffffc($30)   74          sw   $t4,-4($fp)
[13]    0x00400098 0x2529ffff  addiu $9,$9,0xffffffff   75         addiu $t1,$t1,-1
[14]    0x0040009c 0x25080004  addiu $8,$8,0x00000004   76         addiu $t0,$t0,4
[15]    0x004000a0 ???         ???                      77          beq  $t0,$t0,l_gera
[16]    0x004000a4 0x27ca0008  addiu $10,$30,0x0000008 79 prim:  addiu $t2,$fp,8
[17]    0x004000a8 0x001e1021  addu $2,$0,$30           80          move $v0,$fp
[18]    0x004000ac 0x08100023  ???                      81              ???
[19]    0x004000b0 0x03e00008  jr $31                   84 end_gera: jr $ra
```

2. (1,0 ponto) No trecho de programa da questão 1. acima, pelo menos uma instrução executa operação com funcionalidade redundante, que eventualmente poderia ser substituída por uma instrução equivalente sem esta redundância funcional. Pede-se: (1) Identifique que instrução é esta (em que linha do trecho se encontra), comente sua redundância funcional e sugira uma substituição de código fonte para eliminar a redundância. (2) Gere código objeto para esta sua substituição.

3. (3,0 pontos) O programa em linguagem de montagem do MIPS abaixo faz um certo processamento bem específico. (a) Descreva em uma frase o que este trecho de código faz. (b) Aponte no código fonte **todas** as pseudo-instruções que nele existem. (c) Diga o que acontece com a área de dados do programa após a execução do mesmo, especificando se algo é escrito na memória, onde e que valor é escrito. Dicas: Na linha 7 a constante imediata é especificada como um caracter ASCIIE, o que é aceito por montadores do MIPS.

```
1          .text
2          .globl  main
3    main: la     $t0,s
4          li     $t1,0
5    l:    lbu    $t2,0($t0)
6          beq    $t2,$zero,f
7          bne    $t2,'a',na
8          addiu  $t1,$t1,1
9  na:     addiu  $t0,$t0,1
10         j      l
11   f:    la     $t0,c
12         sw     $t1,0($t0)
13         li     $v0,10
14         syscall
15
16         .data
17   s:    .asciiz      "Mamae me ama!"
18   c:    .word        0
```

4. (3,0 pontos) Verdadeiro ou Falso. Abaixo aparecem 10 afirmativas. Marque com V as afirmativas verdadeiras e com F as falsas. Se não souber a resposta correta, deixe em branco, pois cada resposta correta vale 0,3 pontos, mas cada resposta incorreta desconta 0,2 pontos do total positivo de pontos. Não é possível que a questão produza uma nota menor do que 0 pontos.

a) ( ) Suponha que no MIPS se executa a instrução **xori $t0,$t0,0xFFFF**. Assuma que antes de executar esta instrução, **$t0** contém **0xABADF003**. Após executar a instrução, **$t0** conterá **0xABAD0FFC**.

b) ( ) Se um processador possui um PC e um barramento de endereços ambos de 22 bits e usa endereçamento de memória a palavras de 16 bits, ele possui um mapa de memória cujo tamanho é de 8 Megabytes.

c) ( ) O modo de endereçamento pseudo-absoluto, conforme usado no MIPS, parte de um operando de 26 bits, e acrescenta 6 bits em 0 à esquerda dos primeiros, gerando assim o endereço efetivo a ser usado como operando na instrução que o emprega.

d) ( ) Um vetor de 5.000 caracteres ASCIIE representados de forma convencional (ou seja, cada caracter usando o mínimo espaço necessário para armazená-lo) no MIPS ocupa 5001 bytes na memória.

e) ( ) As instruções do MIPS **slti** e **sltiu** diferenciam-se por realizar comparações com e sem sinal, respectivamente. Ambas instruções usam extensão de sinal para gerar o operando especificado pelo dado imediato constante no código objeto da instrução.

f) ( ) Nenhuma instrução que não seja lógica ou aritmética no MIPS usa uma constante imediata como campo de seu código objeto.

g) ( ) O código objeto **0x1509FFFE** corresponde a uma instrução **bne** que quando saltar, o faz necessariamente para uma linha do programa anterior à linha onde se encontra a instrução **bne** em questão.

h) ( ) O mesmo código objeto do item g) acima (**0x1509FFFE**) compara os registradores **$t1** e **$t2**.

i) ( ) Suponha que ao executar uma instrução **lw** que leu dados a partir do endereço de memória **0x10010014**, escreveu-se no registrador $t0 o número **0xABCEDEAA**. Assumindo-se que a implementação do MIPS onde a instrução foi executada é *little endian*, os valores que estão armazenados nos endereços de memória **0x10010014**, **0x10010015**, **0x10010016** e **0x10010017** são, respectivamente **0xAA**, **0xED**, **0xEC** e **0xBA**.

j) ( ) O maior numeral inteiro que se pode armazenar em um registrador de dados do MIPS é maior do que 3 bilhões.

Prova P1 Disciplina: Organização de Computadores

Professor: Ney Laert Vilar Calazans

Aluno: 15/abril/2016

Gabarito

Lista de associação de números e mnemônicos para os registradores do MIPS

Número (Decimal)	Nome
0	$zero
1	$at
2	$v0
3	$v1
4	$a0
5	$a1
6	$a2
7	$a3
8	$t0
9	$t1
10	$t2
11	$t3
12	$t4
13	$t5
14	$t6
15	$t7

Número (Decimal)	Nome
16	$s0
17	$s1
18	$s2
19	$s3
20	$s4
21	$s5
22	$s6
23	$s7
24	$t8
25	$t9
26	$k0
27	$k1
28	$gp
29	$sp
30	$fp
31	$ra

1. (3,0 pontos) Montagem/Desmontagem de código objeto. Abaixo se mostra parte de uma listagem gerada pelo ambiente MARS como resultado da montagem de um programa. Pede-se: (a) Substituir as triplas interrogações pelo texto que deveria estar em seu lugar (existem 6 triplas ???). Em alguns casos, isto implica gerar código objeto, enquanto em outros implica gerar código intermediário e/ou código fonte. Caso uma instrução seja de salto, expresse o exato endereço para onde ela salta (em hexa ou com o rótulo associado à linha), caso isto seja parte das interrogações.
   Dica 1: Dêem muita atenção ao tratamento de endereços e rótulos.
   Dica 2: Tomem muito cuidado com a mistura de representações numéricas: hexa, binário, complemento de 2, etc.
   Obrigatório: Mostre os desenvolvimentos para obter os resultados, justificando.

```
     Endereço   Cód.Objeto  Cód.Intermediário          Cód. Fonte
[1]  0x00400068 0x8fa80004 lw $8,0x00000004($29)   59 gera_lista:lw $t0,4($sp)
[2]  0x0040006c ???        lw $9,0x00000008($29)    60          lw   $t1,8($sp)
[3]  0x00400070 0x00005021 addu $10,$0,$0           61        addu  $t2,$zero,$zero
[4]  0x00400074 0x1120000e beq $9,$0,0x0000000e     64 l_gera:beq   $t1,$zero,end_gera
[5]  0x00400078 0x8d0b0000 lw $11,0x00000000($8)    66          lw   $t3,0($t0)
[6]  0x0040007c 0x240c0000 addiu $12,$0,0x00000000  67          li   $t4,0
[7]  0x00400080 0x11400008 beq $10,$0,0x00000008    68          beq  $t2,$zero,prim
[8]  0x00400084 0xafcafffc sw $10,0xfffffffc($30)   69          sw   $t2,-4($fp)
[9]  0x00400088 0x27ca0008 addiu $10,$30,0x0000008  70              ???
[10] 0x0040008c 0x27de0008 addiu $30,$30,0x0000008  72 cont:  addiu $fp,$fp,8
[11] 0x00400090 0xafcbfff8 sw $11,0xfffffff8($30)   73          sw   $t3,-8($fp)
[12] 0x00400094 0xafccfffc sw $12,0xfffffffc($30)   74          sw   $t4,-4($fp)
[13] 0x00400098 0x2529ffff addiu $9,$9,0xffffffff   75        addiu $t1,$t1,-1
[14] 0x0040009c 0x25080004 addiu $8,$8,0x00000004   76        addiu $t0,$t0,4
[15] 0x004000a0 ???        ???                      77         beq  $t0,$t0,l_gera
[16] 0x004000a4 0x27ca0008 addiu $10,$30,0x0000008  79 prim:  addiu $t2,$fp,8
[17] 0x004000a8 0x001e1021 addu $2,$0,$30           80          move $v0,$fp
[18] 0x004000ac 0x08100023 ???                      81              ???
[19] 0x004000b0 0x03e00008 jr $31                   84 end_gera: jr $ra
```

Solução da Questão 1 (3,0 pontos). Cada ??? vale 0,5 pontos

```
[2]  0x0040006c ???        lw $9,0x00000008($29)    60          lw   $t1,8($sp)
```

A única informação a gerar é o código objeto da instrução. A partir dos demais campos, sabemos se tratar de instrução **lw**, cujo formato, retirado do Apêndice A é:

```
lw   rt, Offset(rs): linguagem de montagem
0x23  rs rt imm : formato do cód. objeto
Número de bits/campo:     6    5  5  16 :
```

O código objeto é muito fácil de gerar: 100011 (0x23 em seis bits) concatenado com o endereço do **rs** no banco, 11101 (29 em binário 5 bits código do registrador $sp), concatenado com o

endereço do **rt** no banco, 01001 ($t1 é o mesmo que $9, ver Tabela no início desta prova), concatenado com o terceiro operando em 16 bits, ou seja 0000 0000 0000 1000 (8 em 16 bits). Juntando estes 32 bits e agrupando-os de 4 em 4 temos: 1000 1111 1010 1001 0000 0000 0000 1000. Traduzindo cada grupo de 4 bits em um valor hexadecimal, obtém-se 0x8FA90008, que é o código objeto da instrução.

Resposta final:

```
[2] 0x0040006c  0x8fa90008    lw $9,0x00000008($29)    60         lw   $t1,8($sp)

[9] 0x00400088  0x27ca0008  addiu $10,$30,0x0000008 70          ???
```

O que se deseja aqui é gerar o código fonte de uma instrução **addiu**, dados o código objeto e intermediário desta. Para realizar a desmontagem, consulta-se o formato da instrução **addiu** no Apêndice A, o que fornece:

	addiu	rt,rs,imm		
	0x9	rs	rt	0	0x2a
Número de bits/campo:	6	5	5	10	6

A geração do código fonte é trivial, basta traduzir o nome dos registradores, mantendo a mesma ordem do código intermediário, e traduzindo a constante do mesmo código intermediário, por exemplo, para decimal. O resultado é **addiu $t2, $fp, 8**.

Resposta Final:

```
[9] 0x00400088  0x27ca0008  addiu $10,$30,0x0000008 70       addiu      $t2,$fp,8

[15] 0x004000a0  ???    ???                      77        beq  $t0,$t0,l_gera
```

O que se deseja aqui é gerar os códigos objeto e intermediário de uma instrução, dado apenas o código fonte. Do Apêndice A descobre-se o formato da instrução **beq**, que é:

	beq	rs,rt,rótulo		
	4	rs	rt	offset
Número de bits/campo:	6	5	5	16

Para gerar o código intermediário, precisamos converter os nomes simbólicos dos registradores em nomes numéricos correspondentes, ou seja $t0=$8. O rótulo é transformado em offset considerando quantas instruções existem entre a instrução imediatamente subsequente ao **beq** (**addiu** na linha [16]) e a instrução associada ao rótulo da **beq** (l_gera). Como são 12 linhas e o rótulo está acima da instrução **beq**, o campo de offset deve conter a constante -12 representada em complemento de 2, 16 bits (ou seja, 0xFFF4). Assim, o código intermediário é **beq $8,$8,0xFFF4**. O código objeto pode ser produzido a partir dos campos do formato e do valor do offset calculado para o código intermediário. Assim temos 000100 (4 em binário 6 bits), 01000 (8 em binário 5 bits, Rs=$t0), 01000 (8 em binário 5 bits, Rt=$t0) e 1111111111110100 (-12 em 16 bits). Juntando os campos e agrupando de 4 em 4 bits, tem-se: 0001 0001 0000 1000 1111 1111 1111 0100 ou **0x1108FFF4**, que é o código objeto.

Resposta Final:

```
[15] 0x004000a0  0x1108fff4     beq $8,$8,0xfff4     77        beq  $t0,$t0,l_gera

[18]   0x004000ac  0x08100023  ???                    81            ???
```

O que se deseja aqui é gerar os códigos intermediário e fonte, dado apenas o código objeto da instrução. Usando a Figura A.10.2 do Apêndice A pode-se descobrir a instrução que se tem aqui, observando-se inicialmente os 6 bits mais significativos. Como estes são 000010 (2 em hexa e decimal), nota-se que se trata da instrução **j**. O formato da instrução **j** é:

	j	rótulo
	2	pseudo-endereço
Número de bits/campo:	6	26

Para obter os códigos fonte e intermediário, basta obter o endereço que corresponde ao pseudo-endereço contido nos 26 bits menos significativos do código objeto. Estes são 00 0001 0000 0000 0000 0010 0011. Se Junta a estes dois 0s à direita e à esquerda os quatro bits mais significativos do valor que o PC terá no momento da execução da instrução **j**. Como ao executar esta instrução o PC estará apontando para a instrução seguinte, ele conterá 0x004000B0. Logo os quatro bits mais significativios são 0000. Isto gera o endereço 0000 0000 0100 0000 0000 0000 1000 1100, ou **0x0040008C**, que corresponde ao endereço do rótulo **cont**.

Resposta Final:

```
[18]   0x004000ac  0x08100023  j   0x0040008c   81        j cont
```

Fim da Solução da Questão 1 (3,0 pontos)

2. (1,0 ponto) No trecho de programa da questão 1. acima, pelo menos uma instrução executa operação com funcionalidade redundante, que eventualmente poderia ser substituída por uma instrução equivalente sem esta redundância funcional. Pede-se: (1) Identifique que instrução é esta (em que linha do trecho se encontra), comente sua redundância funcional e sugira uma substituição de código fonte para eliminar a redundância. (2) Gere código objeto para esta sua substituição.

Solução da Questão 2 (1,0 ponto)

a) Isto acontece na linha [15], onde a instrução beq $t0, $t0, l_gera realiza um teste que sempre resulta verdadeiro, pois compara o conteúdo de $t0 com ele mesmo. Logo, isto se configura como um teste inútil e redundante. No seu lugar poderia existir apenas uma instrução **j l_gera**.

b) Se a instrução fosse l_gera, seu código objeto seria (ver formato da instrução **j** no Apêndice A) 0x2 em seis bits, ou 000010, seguido dos bits 25 a 2 do endereço associado ao rótulo l_gera, que é 0x00400074. Isto fornece os 26 bits seguintes: 0000 0100 0000 0000 0000 0111 01. Logo, o código objeto da instrução e questão seria 0000 1000 0001 0000 0000 0000 0001 1101, ou **0x0810001D**.

.

Fim da Solução da Questão 2 (1,0 ponto)

3. (3,0 pontos) O programa em linguagem de montagem do MIPS abaixo faz um certo processamento bem específico. (a) Descreva em uma frase o que este trecho de código faz. (b) Aponte no código fonte **todas** as pseudo-instruções que nele existem. (c) Diga o que acontece com a área de dados do programa após a execução do mesmo, especificando se algo é escrito na memória, onde e que valor é escrito. Dicas: Na linha 7 a constante imediata é especificada como um caracter ASCIIE, o que é aceito por montadores do MIPS.

```
1            .text
2            .globl  main
3       main: la     $t0,s
4             li     $t1,0
5       l:    lbu    $t2,0($t0)
6             beq    $t2,$zero,f
7             bne    $t2,'a',na
8             addiu  $t1,$t1,1
9       na:   addiu  $t0,$t0,1
10            j      l
11      f:    la     $t0,c
12            sw     $t1,0($t0)
13            li     $v0,10
14            syscall
15
16            .data
17      s:    .asciiz       "Mamae me ama!"
18      c:    .word         0
```

Solução da Questão 3 (3,0 pontos)

a) Este programa conta o número de vezes que o caracter **'a'** aparece na cadeia **s**, escrevendo este valor, ao final do processamento, na variável **c** em memória.

b) As cinco linhas que contêm pseudo-instruções estão salientadas no código acima (linhas 3, 4, 7, 11 e 13). Note-se que embora **bne** seja o mnemônico de uma instrução, esta pressupõe que os seus dois primeiros operandos sejam registradores do banco e não um registrador e uma constante, como aparece aqui. Logo, esta linha contém uma pseudo-instrução.

c) A área de dados do programa é lida nas posições da cadeia **s** byte a byte, e ao final do processamento, escreve-se o valor do contador de **'a's** (em **$t1**) na posição de memória **c**. No caso, o valor escrito é 4.

Fim da Solução da Questão 3 (3,0 pontos)

4. (3,0 pontos) Verdadeiro ou Falso. Abaixo aparecem 10 afirmativas. Marque com V as afirmativas verdadeiras e com F as falsas. Se não souber a resposta correta, deixe em branco, pois cada resposta correta vale 0,3 pontos, mas cada resposta incorreta desconta 0,2 pontos

do total positivo de pontos. Não é possível que a questão produza uma nota menor do que 0 pontos.

Solução da Questão 3 (3,0 pontos)

a) (**V**) Suponha que no MIPS se executa a instrução **xori $t0,$t0,0xFFFF**. Assuma que antes de executar esta instrução, **$t0** contém **0xABADF003**. Após executar a instrução, **$t0** conterá **0xABAD0FFC**.

A operação xor de um bit x com 0 dá como resultado o valor de x e o xor de 1 com o mesmo x gera o inverso de x. Como **xori** produz a constante de 32bits a partir de 0xFFFF fazendo extensão de 0, a constante usada é então **0x0000FFFF**. Ao fazer o xor deste valor com o valor em $t0, obtém-se como resultado um vetor onde os 16 bits mais significativos são iguais ao valor originalmente em $t0 e os 16 bits menos significativos contêm o inverso (bit a bit) dos 16 bits menos significativos de $t0. Logo, a afirmativa é verdadeira (o inverso de 0xF003 é 0x0FFC). **V**

b) (**V**) Se um processador possui um PC e um barramento de endereços ambos de 22 bits e usa endereçamento de memória a palavras de 16 bits, ele possui um mapa de memória cujo tamanho é de 8 Megabytes.

Como se sabe, 2**(número de bits do PC) dá a extensão do mapa em posições. Ora, 2**22 = 4Mega. Como o endereçamento é a palavra de 16 bits, cada endereço armazena exatamente dois bytes. Assim, o mapa de memória tem o tamanho citado na afirmativa (2bytes*4Mega=8Megabytes). **V**

c) (**F**) O modo de endereçamento pseudo-absoluto, conforme usado no MIPS, parte de um operando de 26 bits, e acrescenta 6 bits em 0 à esquerda dos primeiros, gerando assim o endereço efetivo a ser usado como operando na instrução que o emprega.

O modo pseudo-absoluto realmente gera 6 bits para acrescentar aos 26 bits do operando da instrução j. Contudo, isto não dá da forma descrita neste item. Acrescentam-se dois bits em 0 à direita e acrescentam-se à esquerda dos mesmos 26 bits os 4 bits mais significativos do registrador PC no momento da execução da instrução **j**. Logo, a afirmativa é Falsa. **F**

d) (**V**) Um vetor de 5.000 caracteres ASCIIE representados de forma convencional (ou seja, cada caracter usando o mínimo espaço necessário para armazená-lo) no MIPS ocupa 5001 bytes na memória.

A afirmativa é Verdadeira, pois nas condições estabelecidas no item cada caracter ASCIIE ocupa exatamente 1 byte. Além do mais, acrescenta-se sempre ao final da cadeia o caracter de controle NULL (0x00), que indica o fim da cadeia. **V**

e) (**V**) As instruções do MIPS **slti** e **sltiu** diferenciam-se por realizar comparações com e sem sinal, respectivamente. Ambas instruções usam extensão de sinal para gerar o operando especificado pelo dado imediato constante no código objeto da instrução.

A afirmativa é Verdadeira, o que se pode verificar na descrição destas duas instruções no Apêndice A. **V**

f) (**F**) Nenhuma instrução que não seja lógica ou aritmética no MIPS usa uma constante imediata como campo de seu código objeto.

A afirmativa é Falsa, um contra-exemplo claro são as instruções de leitura e escrita a memória, que usam um valor constante. Este é somado ao registrador base para obter o endereço efetivo de memória a usar na instrução. **F**

g) (**V**) O código objeto **0x1509FFFE** corresponde a uma instrução **bne** que quando saltar, o faz necessariamente para uma linha do programa anterior à linha onde se encontra a instrução **bne** em questão.

De fato, trata-se de uma instrução **bne**, pois se extraindo os seis bits mais significativos do código objeto, obtém-se 000101, ou seja 5 em decimal, que segundo a Tabela A.10.2 do Apêndice A identifica a instrução **bne**. além disto o deslocamento é 0xFFFE, que é o número negativo -2 em complemento de 2. Assim, o salto é para linha anterior ao bne no programa. **V**

h) (**F**) O mesmo código objeto do item g) acima (**0x1509FFFE**) compara os registradores **$t1** e **$t2**.

Extraindo-se os bits 25-21 e 20-16 do código objeto, obtém-se 01000 e 01001 ou seja, 8 e 9 em decimal. Isto corresponde aos registradores **$t0** e **$t1**, não a **$t1** e **$t2**. Logo, a afirmativa é falsa. **F**

i) (**F**) Suponha que ao executar uma instrução **lw** que leu dados a partir do endereço de memória **0x10010014**, escreveu-se no registrador $t0 o número **0xABCEDEAA**.

Assumindo-se que a implementação do MIPS onde a instrução foi executada é *little endian*, os valores que estão armazenados nos endereços de memória **0x10010014**, **0x10010015**, **0x10010016** e **0x10010017** são, respectivamente **0xAA**, **0xED**, **0xEC** e **0xBA**.

A afirmativa é falsa, pois a ordem de bytes é do menos significativo para o mais significativo, mas dentro de cada byte a ordem dos bits é mantida. Assim a resposta correta seria dizer que os valores que estão armazenados nos endereços de memória 0x10010014, 0x10010015, 0x10010016 e 0x10010017 são, respectivamente 0xAA, 0xDE, 0xCE e 0xAB. **F**

j) (**F**) O maior numeral inteiro que se pode armazenar em um registrador de dados do MIPS é maior do que 3 bilhões.

Afirmativa falsa, pois com 32 bits os números inteiros possíveis de se representa vão de -2^{31} a $+2^{31}-1$. O maior número inteiro representável é então 0111 1111 1111 1111 1111 1111 1111 1111 em binário, ou 0x7FFFFFFF em hexa ou, em decimal 2.147.483.647, um número menor que 3 bilhões. **F**

Fim da Solução da Questão 3 (3,0 pontos)