Prova P1 **Disciplina: Organização de Computadores**

Professor: Ney Laert Vilar Calazans

Aluno: 17/abril/2018

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. (4,0 pontos) Montagem/Desmontagem de código objeto. Abaixo se mostra uma listagem gerada pelo ambiente MARS como resultado da montagem de um programa. Pede-se que se substituam as triplas interrogações pelo texto que deveria estar em seu lugar (existem 8 triplas ???). Em alguns casos, isto implica gerar código objeto, e/ou gerar código intermediário e/ou gerar código fonte. Caso uma instrução a ser colocada no lugar das interrogações seja um salto, expresse na área do código fonte/intermediário o exato endereço para onde ela salta (em hexa e/ou com o rótulo associado à linha).
   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  Código Intermediário      Código Fonte
[1]   0x00400094 0x00001021 addu $2,$0,$0     65   isq:  move   $v0,$zero
[2]   0x00400098 ???        ???               66         move   $t1,$a0
[3]   0x0040009c 0x20080001 addi $8,$0,1      67         addi   $t0,$zero,1
[4]   0x004000a0 0x00084780 ???               68         ???
[5]   0x004000a4 0x0128502a slt  $10,$9,$8    70   isqb: slt    $t2,$t1,$t0
[6]   0x004000a8 0x11400002 beq  $10,$0,2     71         beq    $t2,$zero,isql
[7]   0x004000ac 0x00084082 srl  $8,$8,2      72         srl    $t0,$t0,2
[8]   0x004000b0 0x08100029 j    0x004000a4   73         j      isqb
[9]   0x004000b4 0x1100000a beq  $8,$0,10     75   isql: beq    $t0,$zero,isqr
[10]  0x004000b8 0x00485820 add  $11,$2,$8    76         add    $t3,$v0,$t0
[11]  0x004000bc 0x012b502a slt  $10,$9,$11   77         slt    $t2,$t1,$t3
[12]  0x004000c0 0x11400002 beq  $10,$0,2     78         ???
[13]  0x004000c4 0x00021042 srl  $2,$2,1      79         srl    $v0,$v0,1
[14]  0x004000c8 ???        j    0x004000d8   80         j      isqf
[15]  0x004000cc 0x012b4822 sub  $9,$9,$11    82   isqe: sub    $t1,$t1,$t3
[16]  0x004000d0 0x00021042 srl  $2,$2,1      83         srl    $v0,$v0,1
[17]  0x004000d4 0x00481020 add  $2,$2,$8     84         add    $v0,$v0,$t0
[18]  0x004000d8 0x00084082 srl  $8,$8,2      86   isqf: srl    $t0,$t0,2
[19]  0x004000dc 0x1000fff5 ???               87         ???
[20]  0x004000e0 0x03e00008 jr   $31          89   isqr: jr     $ra
```

2. (3,0 pontos) O programa em linguagem de montagem do MIPS abaixo faz um processamento bem específico. Assuma que o rótulo **`isqrt`** (ver linha 23 do programa) é o início do código de uma função que calcula o valor inteiro da raiz quadrada do número passado como parâmetro no registrador $a0, e que o valor da raiz retorna no registrador $v0 ao final da execução da função **`isqrt`**. Pede-se: (a) Descreva em uma frase o que este trecho de código faz, do ponto de vista semântico. (b) Este programa escreve algo na memória de dados do processador? Caso afirmativo, diga em que posição(ões) de memória ele escreve e que valor(es) escreve.

```
1          .text
2          .data
```

```
3   x:    .word       1
4   y:    .word       5
5   z:    .word       4
6   k1:   .word       0
7   k2:   .word       0
8   erst: .asciiz     "Error!"
9         .text
10        .globl      main
11  main: la          $s0,x
12        lw          $s0,0($s0)
13        la          $s1,y
14        lw          $s1,0($s1)
15        la          $s2,z
16        lw          $s2,0($s2)
17        mul         $s3,$s1,$s1
18        sll         $s4,$s0,2
19        mul         $s4,$s4,$s2
20        sub         $s5,$s3,$s4
21        bltz        $s5,error
22        move        $a0,$s5
23        jal         isqrt
24        sub         $s6,$v0,$s1
25        sll         $s7,$s0,1
26        div         $s6,$s7
27        mflo        $s6
28        la          $t0,k1
29        sw          $s6,0($t0)
30        sub         $s6,$zero,$s1
31        sub         $s6,$s6,$v0
32        div         $s6,$s7
33        mflo        $s6
34        la          $t0,k2
35        sw          $s6,0($t0)
36  end:  li          $v0,10
37        syscall
38  error:li          $v0,4
39        la          $a0,erst
40        syscall
41        j           end
```

3. (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 uma resposta correta vale 0,3 pontos, mas uma incorreta desconta 0,2 pontos do total positivo de pontos. Não é possível que a questão produza uma nota menor do que 0,0 pontos.
   a) ( ) Suponha que existem duas arquiteturas, **X** e **Y**, de 16 bits que usam endereços de 24 bits, mas **X** usa endereçamento a byte e **Y** usa endereçamento a palavra. **X** tem um mapa de memória de **16 Mbytes** e **Y** tem um mapa de memória de **32 Mbytes**.
   b) ( ) O modo de endereçamento base-deslocamento do MIPS permite acessar qualquer dado do tamanho desejado em uma região de **256 Kbytes** em torno do endereço base.
   c) ( ) De acordo com a Tabela A.10.2 do Apêndice A do livro-texto, o vetor binário **011001** nos bits 31 a 26 de uma palavra de 32 bits corresponde a um código inválido de instrução na arquitetura MIPS R2000.
   d) ( ) Sempre que se escrever em memória um valor de um registrador do MIPS, necessariamente se escreve na memória todo o conteúdo deste registrador.
   e) ( ) A instrução **slt** do MIPS não realiza um salto, mas pode servir para implementar ou estender operações de controle de fluxo em um programa.
   f) ( ) Após executar a instrução **addi $t0,$zero,0xB573**, o conteúdo do registrador **$t0** será **0xFFFFB573**, independente do seu valor inicial.
   g) ( ) A instrução **xori $t0, $t0, 0xFFFF** escreve em **$t0** um número onde os 16 bits mais significativos (à esquerda) são os mesmos de antes da execução do **xori**, mas onde os 16 bits menos significativos de **$t0** são todos invertidos em relação ao valor original.
   h) ( ) O código objeto **0x06A00012** corresponde a um salto condicional para uma posição de memória necessariamente abaixo (em endereço maior) da posição onde está o salto.
   i) ( ) A cadeia de caracteres definida abaixo ocupa 20 bytes em memória:
      ```
      cadeia:    asciiz     "Titia me idolatra!!!"
      ```
   j) ( ) A instrução **andi $t0, $t0, 3** seguida de uma instrução de salto condicional permite identificar se um número é múltiplo de 4 ou não.

Prova P1 **Disciplina: Organização de Computadores**

Professor: Ney Laert Vilar Calazans

Aluno: 17/abril/2018

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. (4,0 pontos) Montagem/Desmontagem de código objeto. Abaixo se mostra uma listagem gerada pelo ambiente MARS como resultado da montagem de um programa. Pede-se que se substituam as triplas interrogações pelo texto que deveria estar em seu lugar (existem 8 triplas ???). Em alguns casos, isto implica gerar código objeto, e/ou gerar código intermediário e/ou gerar código fonte. Caso uma instrução a ser colocada no lugar das interrogações seja um salto, expresse na área do código fonte/intermediário o exato endereço para onde ela salta (em hexa e/ou com o rótulo associado à linha).
   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  Código Intermediário      Código Fonte
[1]   0x00400094  0x00001021  addu $2,$0,$0     65   isq:  move   $v0,$zero
[2]   0x00400098  ???         ???               66         move   $t1,$a0
[3]   0x0040009c  0x20080001  addi $8,$0,1      67         addi   $t0,$zero,1
[4]   0x004000a0  0x00084780  ???               68         ???
[5]   0x004000a4  0x0128502a  slt  $10,$9,$8    70   isqb: slt    $t2,$t1,$t0
[6]   0x004000a8  0x11400002  beq  $10,$0,2     71         beq    $t2,$zero,isql
[7]   0x004000ac  0x00084082  srl  $8,$8,2      72         srl    $t0,$t0,2
[8]   0x004000b0  0x08100029  j    0x004000a4   73         j      isqb
[9]   0x004000b4  0x1100000a  beq  $8,$0,10     75   isql: beq    $t0,$zero,isqr
[10]  0x004000b8  0x00485820  add  $11,$2,$8    76         add    $t3,$v0,$t0
[11]  0x004000bc  0x012b502a  slt  $10,$9,$11   77         slt    $t2,$t1,$t3
[12]  0x004000c0  0x11400002  beq  $10,$0,2     78         ???
[13]  0x004000c4  0x00021042  srl  $2,$2,1      79         srl    $v0,$v0,1
[14]  0x004000c8  ???         j    0x004000d8   80         j      isqf
[15]  0x004000cc  0x012b4822  sub  $9,$9,$11    82   isqe: sub    $t1,$t1,$t3
[16]  0x004000d0  0x00021042  srl  $2,$2,1      83         srl    $v0,$v0,1
[17]  0x004000d4  0x00481020  add  $2,$2,$8     84         add    $v0,$v0,$t0
[18]  0x004000d8  0x00084082  srl  $8,$8,2      86   isqf: srl    $t0,$t0,2
[19]  0x004000dc  0x1000fff5  ???               87         ???
[20]  0x004000e0  0x03e00008  jr   $31          89   isqr: jr     $ra
```

Solução da Questão 1 (4,0 pontos). Cada ??? vale 0,5 pontos

```
[2]   0x00400098  ???         ???               66         move   $t1,$a0
```

O que se quer aqui é gerar os códigos intermediário e objeto da pseudo-instrução move $t1,$a0, dado seu código fonte, ou seja, uma operação de montagem de código. Note que a linha [1] do trecho de código da questão também contém um **move**. Dela se pode depreender de forma óbvia que o código intermediário é addu $9, $0, $4 ou, equivalentemente, addu $9, $4, $0. Note que as duas opções são funcionalmente equivalentes, mas geram códigos objeto distintos. Assumamos

doravante a primeira opção. Basta extrair do Apêndice A para o formato da instrução addu, qual seja:

```
                          addu   rd,rs,rt     : ling. de montagem
                          0x0 rs rt rd 0 0x21 : cód. objeto
Número de bits/campo:       6  5  5  5  5  6  :
```

A geração do código objeto a partir daqui é imediata. Temos que rs=$0→00000, rt=$4→00100 e rt=$9→01001. Juntando todos os valores dos 6 campos, tem-se 000000 00000 00100 01001 00000 100001. Agrupando estes 32 bits de 4 em 4 bits e convertendo cada grupo de 4 bits em hexa, o código objeto é então 0x00044821.

Resposta Final:

```
[2]    0x00400098  0x00044821  addu  $9,$0,$4        105 e_g: move  $t1,$a0
```

```
[4]    0x004000a0  0x00084780  ???                   68     ???
```

O que se quer aqui é gerar os códigos intermediário e fonte de uma instrução, dado apenas seu código objeto, ou seja, uma operação de desmontagem de código. Para realizar a desmontagem, separa-se os 6 bits mais à esquerda do código objeto (000000, ou 0) e usa-se este na Tabela da Figura A.10.2 do Apêndice A, o que identifica uma instrução tipo R. Para descobrir qual, deve-se tomar os 6 bits mais à direita da instrução (000000), e usá-lo como índice na sexta coluna da mesma Tabela, o que identifica a instrução SLL. Com esta descoberta, usa-se novamente o Apêndice A para dele extrair o formato da instrução, qual seja:

```
                          sll   rd,rt,shamt
                         0x0    0     rt    rd    shamt      0
Número de bits/campo:    6      5     5     5     5          6.
```

Os bits 25-21 não interessam (valem 0) e dos bits 20-16 se extrai o registrador rt, que é 01000 ou $8 ou $t0. Dos bits 15-11 se extrai o registrador rd, que é 01000 ou $8 ou $t0. O campo shamt (bits 10-6) valem 11110 ou em decimal 30. Assim, tudo está pronto para gerar os códigos intermediário e fonte.

Resposta final:

```
[4]    0x004000a0  0x00084780  sll     $8,$8,30     68     sll    $t0,$t0,30
```

```
[12]   0x004000c0  0x11400002  beq  $10,$0,2     78        ???
```

Aqui, deseja-se gerar o código fonte de uma instrução beq com seu código objeto e intermediário dados. Usa-se a formato da instrução **beq**, retirado do Apêndice A que é:

```
                          beq   rs,rt,label   : ling. de montagem
                          0x4 rs rt offset    : cód. objeto
Número de bits/campo:       6  5  5    16     :
```

Em seguida, obtém-se facilmente do código intermediário da instrução os nomes dos registradores referenciados usando a tabela no início da prova, o que fornece rs=$t2 e rt=$zero. O último valor a descobrir é o rótulo para onde o beq salta, quando salta. Para descobrir este, basta olhar o valor do offset (2) e notar que duas instruções abaixo da instrução que segue o beq encontra-se o rótulo isqe, que é o rótulo desejado:

Resposta Final:

```
[12]   0x004000c0  0x11400002  beq  $10,$0,2     78        beq    $t2,$zero,isqe
```

```
[14]   0x004000c8  ???         j    0x004000d8   80        j      isqf
```

O que se quer aqui é gerar o código objeto de uma instrução j, dados seus códigos fonte e intermediário, ou seja, uma operação de montagem de código. Obtém-se o formato da instrução j do Apêndice A:

```
                          j    target         : ling. de montagem
                          0x2  target         : cód. objeto
Número de bits/campo:       6    26          :
```

Para gerar o código objeto já temos os seis primeiros bits (o opcode), o endereço para onde saltar, bastando agora transformar o mesmo nos 26 bits do campo target. Para tanto, basta remover os 4 bits mais significativos (0000) e os dois bits menos significativos (00). O que sobra é o campo target (em binário 0000 0100 0000 0000 0000 1101 10). A estes 26 se acrescenta os seis bits 000010 à esquerda, converte-se os 32 bits para hexa e isto dá o código objeto desejado.

Resposta Final:

```
[14]   0x004000c8  08100036      j    0x004000d8   80        j      isqf
```

```
[19]    0x004000dc  0x1000fff5  ???              87        ???
```

O que se quer aqui é gerar os códigos intermediário e fonte de uma instrução, dado apenas seu código objeto, ou seja, uma operação de desmontagem de código. Para realizar a desmontagem, separa-se os 6 bits mais à esquerda do código objeto (000100, ou 4) e usa-se este na Tabela da Figura A.10.2 do Apêndice A, o que identifica a instrução beq. Com esta descoberta, usa-se novamente o Apêndice A para dele extrair o formato da instrução, qual seja:

```
                          beq   rs,rt,label   : ling. de montagem
                          0x4 rs rt offset    : cód. objeto
Número de bits/campo:       6  5  5    16     :
```

Os campos rs e rt são extraídos respectivamente dos bits 25-21 (00000 ou seja $0, ou seja $zero) e 20-16 (os mesmos 00000 ou seja $0, ou seja $zero). O últimos valores a serem definidos são o offset (retirado direto dos bits 15-0 como sendo 0xFFF5) e o rótulo para onde salta. Para descobrir este rótulo, converte-se o número inteiro representado em complemento de 2 16 bits pelo hexadecimal 0xFFF5 para o decimal correspondente. Primeiro, trata-se de um número negativo (pois o bit mais significativo de 0xFFF5 é 1). Se trocarmos o sinal deste fica mais fácil enxergar o valor do número em módulo. Isto pode ser realizado invertendo-se todos os bits de 0xFFF5 (o que dá 0x000A) e somando 1 no resultado (o que dá 0x000B). Logo como 0x000B equivale a 11 em decimal o número original (0xFFF5) corresponde a -11 em decimal. Conta-se a partir da instrução abaixo do beq (na linha [20]) 11 instruções para cima, atingindo-se assim a linha [9], onde se encontra o rótulo procurado, **isql**. A resposta final é então:

Resposta Final:

```
[19]    0x004000dc  0x1000fff5  beq      $0,$0,0xFFF5 87     beq    $zero,$zero,isql
```

Fim da Solução da Questão 1 (4,0 pontos)

2. (3,0 pontos) O programa em linguagem de montagem do MIPS abaixo faz um processamento bem específico. Assuma que o rótulo **`isqrt`** (ver linha 23 do programa) é o início do código de uma função que calcula o valor inteiro da raiz quadrada do número passado como parâmetro no registrador $a0, e que o valor da raiz retorna no registrador $v0 ao final da execução da função **`isqrt`**. Pede-se: (a) Descreva em uma frase o que este trecho de código faz, do ponto de vista semântico. (b) Este programa escreve algo na memória de dados do processador? Caso afirmativo, diga em que posição(ões) de memória ele escreve e que valor(es) escreve.

```
42          .text
43          .data
44    x:    .word       1            # O parâmetro x da equação
45    y:    .word       5            # O parâmetro y da equação
46    z:    .word       4            # O parâmetro z da equação
47    k1:   .word       0            # O local onde guardar a raiz k´
48    k2:   .word       0            # O local onde guardar a raiz k´´
49    erst: .asciiz     "Error!"     # Mensagem de erro a ser emitada se discr < 0
50          .text
51          .globl      main
52    main: la          $s0,x        # Busca endereço de x
53          lw          $s0,0($s0)   # Coloca x em $s0
54          la          $s1,y        # Busca endereço de y
55          lw          $s1,0($s1)   # Coloca y em $s1
56          la          $s2,z        # Busca endereço de z
57          lw          $s2,0($s2)   # Coloca z em $s2
58          mul         $s3,$s1,$s1  # Gera y**2 em $s3
59          sll         $s4,$s0,2    # Gera 4*x em $s4
60          mul         $s4,$s4,$s2  # Gera 4*x*z
61          sub         $s5,$s3,$s4  # Gera o discrimante da equação em $s5
62          bltz        $s5,error    # Se discriminante (y**2 - 4*a*c) < 0, erro
63          move        $a0,$s5      # Se dsicriminante >= 0 raízes são reais
64          jal         isqrt        # Calcula raiz quadrada do discriminante
65          sub         $s6,$v0,$s1  # Gera -y + raiz quadrada (discrimante) em $s6
66          sll         $s7,$s0,1    # Gera 2*a em $s7
67          div         $s6,$s7      # lo <= (-y+raizquad(discrim))/(2*a)
68          mflo        $s6          # $s6 <= (-y+raizquad(discrim))/(2*a), raiz k´
69          la          $t0,k1       # Prepara escrita de k´
70          sw          $s6,0($t0)   # Escreve k´ na memória
71          sub         $s6,$zero,$s1# Gera -y em $s6
72          sub         $s6,$s6,$v0  # Gera -y - raiz quadrada (discrimante) em $s6
73          div         $s6,$s7      # lo <= (-y-raizquad(discrim))/(2*a), raiz k´´
74          mflo        $s6          # $s6 <= (-y-raizquad(discrim))/(2*a), raiz k´´
```

```
75        la            $t0,k2        # Prepara escrita de k´´
76        sw            $s6,0($t0)    # Escreve k´ na memória
77  end:  li            $v0,10        # Termina o programa
78        syscall                     # Tchau!
79  error:li            $v0,4         # Prepara impressão da mensagem de erro, passo 1
80        la            $a0,erst      # Prepara impressão da mensagem de erro, passo 2
81        Syscall                     # Imprime mensagem de erro
82        j             end           # E parte para acabar o programa.
```

Solução da Questão 2 (3,0 pontos)

a) Este programa aplica o cálculo da fórmula de Bhaskara para descobrir as duas raízes reais de uma equação quadrática x*k**2 + y*k + z = 0. Naturalmente a fórmula é (k´, k´´) = (-y +/- raiz_quadrada(y**2-4*x*z)) / (2*x). Caso o discriminante seja negativo, o programa termina sem calcular as raízes reais e imprimindo uma mensagem de erro na tela.

b) O programa escreve dois valores na memória de dados, que correspondem às raízes k´ e k´´ (respectivamente as posições de memória k1 e k2 do programa), se estas existirem. Se não existirem, nada é escrito na memória.

Fim da Solução da Questão 2 (3,0 pontos)

3. (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 uma resposta correta vale 0,3 pontos, mas uma incorreta desconta 0,2 pontos do total positivo de pontos. Não é possível que a questão produza uma nota menor do que 0,0 pontos.
   a) (V) Suponha que existem duas arquiteturas, **X** e **Y**, de 16 bits que usam endereços de 24 bits, mas **X** usa endereçamento a byte e **Y** usa endereçamento a palavra. **X** tem um mapa de memória de **16 Mbytes** e **Y** tem um mapa de memória de **32 Mbytes**.
   b) (F) O modo de endereçamento base-deslocamento do MIPS permite acessar qualquer dado do tamanho desejado em uma região de **256 Kbytes** em torno do endereço base.
   c) (V) De acordo com a Tabela A.10.2 do Apêndice A do livro-texto, o vetor binário **011001** nos bits 31 a 26 de uma palavra de 32 bits corresponde a um código inválido de instrução na arquitetura MIPS R2000.
   d) (F) Sempre que se escrever em memória um valor de um registrador do MIPS, necessariamente se escreve na memória todo o conteúdo deste registrador.
   e) (V) A instrução **slt** do MIPS não realiza um salto, mas pode servir para implementar ou estender operações de controle de fluxo em um programa.
   f) (V) Após executar a instrução **addi $t0,$zero,0xB573**, o conteúdo do registrador **$t0** será **0xFFFFB573**, independente do seu valor inicial.
   g) (V) A instrução **xori $t0,$t0,0xFFFF** escreve em **$t0** um número onde os 16 bits mais significativos (à esquerda) são os mesmos de antes da execução do **xori**, mas onde os 16 bits menos significativos de **$t0** são todos invertidos em relação ao valor original.
   h) (V) O código objeto **0x06A00012** corresponde a um salto condicional para uma posição de memória necessariamente abaixo (em endereço maior) da posição onde está o salto.
   i) (F) A cadeia de caracteres definida como segue ocupa 20 bytes em memória:
      `cadeia:  asciiz  "Titia me idolatra!!!"`
   j) (V) A instrução **andi $t0,$t0,3** seguida de uma instrução de salto condicional permite identificar se um número é múltiplo de 4 ou não.

Solução da Questão 3 (3,0 pontos)

a) (V) Suponha que existem duas arquiteturas, X e Y, de 16 bits que usam endereços de 24 bits, mas X usa endereçamento a byte e Y usa endereçamento a palavra. X tem um mapa de memória de 16 Mbytes e Y tem um mapa de memória de 32 Mbytes.

Justificativa: Endereços de 24 bits correspondem a 2^{24} posições no mapa de memória, ou seja 16Mposições. Como X usa endereçamento a byte cada posição corresponde a 1 byte e o mapa total possui 16Mbytes. Y, por outro lado usa endereçamento a palavra. Como cada palavra é de 16 bits ou 2 bytes (tipo da arquitetura) o mapa de memória possui tamanho total de 16M * 2 bytes ou 32Mbytes. Afirmativa V.

b) (F) O modo de endereçamento base-deslocamento do MIPS permite acessar qualquer dado do tamanho desejado em uma região de **256 Kbytes** em torno do endereço base.

Justificativa: O modo base-deslocamento usa um valor em complemento de 2 em 16 bits para representar o deslocamento a partir do endereço base. Assim, como 2^{16} = 65536 ou 64K e o

MIPS usa endereçamento a byte o tamanho da faixa de endereços de dados abrangida pela instrução que usa este modo é 64Kbytes, e não 256Kbytes. Afirmativa F.

c) (V) De acordo com a Tabela A.10.2 do Apêndice A do livro-texto, o vetor binário **011001** nos bits 31 a 26 de uma palavra de 32 bits corresponde a um código inválido de instrução na arquitetura MIPS R2000.

Justificativa: Convertendo 011001 para decimal obtém-se 25. Entrando com este valor na primeira coluna da Tabela A.10.2 (bits 31-26) nota-se que a linha decimal 25 da Tabela não contém nem o nome de uma instrução e nem um apontador para outra coluna da Tabela, o que indica um código não atribuído a qualquer instrução existente. Afirmativa V.

d) (F) Sempre que se escrever em memória um valor de um registrador do MIPS, necessariamente se escreve na memória todo o conteúdo deste registrador.

Justificativa: Como existem instruções (por exemplo sh e sb) que escrevem valores menores que 32 bits na memória a afirmativa é claramente falsa. Afirmativa F.

e) (V) A instrução **slt** do MIPS não realiza um salto, mas pode servir para implementar ou estender operações de controle de fluxo em um programa.

Justificativa: É verdade que slt não realiza um salto (ela apenas escreve a constante 0 ou a constante 1 em um registrador, de acordo com o resultado de uma comparação de magnitude entre dois registradores). Mas, **slt** combinada com instruções de salto condicional (tais como **beq**/**bne**), que testam e saltam de acordo com a relação entre o valor do registrador escrito por **slt** e algum outro registrador (tipicamente, mas não necessariamente, o registrador $zero) permite realizar operações de controle do fluxo de execução de instruções. Afirmativa V.

f) (V) Após executar a instrução **addi $t0,$zero,0xB573**, o conteúdo do registrador **$t0** será **0xFFFFB573**, independente do seu valor inicial.

Justificativa: As instruções aritméticas com modo de endereçamento imediato usam extensão de sinal para transformar a constante de 16 bits em uma constante de 32 bits. Como 0xB573 possui o bit mais significativo em 1, este bits será estendido 16 vezes, gerando o operando 0xFFFFB573 a ser somado com $zero. Como este último possui sempre como conteúdo a constante 0 em 32 bits, o resultado da soma, escrito em $t0 é 0xFFFFB573. Afirmativa V.

g) (V) A instrução **xori $t0,$t0,0xFFFF** escreve em **$t0** um número onde os 16 bits mais significativos (à esquerda) são os mesmos de antes da execução do **xori**, mas onde os 16 bits menos significativos de **$t0** são todos invertidos em relação ao valor original.

Justificativa: A instrução **xori** trabalha com extensão de 0. Logo o operando 0xFFFF é transformado no valor e 32 bits 0x0000FFFF. Ao se realizar o XOR deste valor com o conteúdo de 32 bits do registrador $t0, pode-se usar as propriedades da operação Booleana XOR que dizem que 1) 0 XOR x = x e 2) 1 XOR x = not x. Daí deriva que a afirmativa é verdadeira. Afirmativa V.

h) (V) O código objeto **0x06A00012** corresponde a um salto condicional para uma posição de memória necessariamente abaixo (em endereço maior) da posição onde está o salto.

Justificativa: Como os 6 primeiros bits à esquerda do código objeto dado aqui são 000001, a Tabela A.10.2 diz para procurarmos na quinta coluna (bits 20-16) a instrução específica em questão usando os bits 20-16 do código objeto. Estes bits são 00000, e levam ao primeiro código da quinta coluna, onde se encontra a instrução **bltz**. De fato, trata-se de uma instrução de salto condicional, simular ao **beq**, mesmo modo de endereçamento relativo e com formato bem parecido. Como o campo de offset (bits 15-0) é 0x0012, quando o salto for tomado, o será para uma instrução que está 12 instruções abaixo da linha seguinte ao **bltz**. Afirmativa V.

i) (F) A cadeia de caracteres definida como segue ocupa 20 bytes em memória:

```
cadeia:    asciiz    "Titia me idolatra!!!"
```

Justificativa: A cadeia possui exatamente 20 caracteres ASCIIE, ocupando cda um exatamente 1byte. Contudo, como se usou a diretiva .asciiz, será acrescentado o caracter de fim de cadeia NULL, e a cadeia em si ocupará 21 bytes em memória. Afirmativa F.

j) (V) A instrução **andi $t0,$t0,3** seguida de uma instrução de salto condicional permite identificar se um número é múltiplo de 4 ou não.

Justificativa: A partir da instrução e do formato da **andi**, nota-se que a constante usada nesta em hexadecimal é 0x0003, ou, em binário 0000 0000 0000 0011. Primeiro, constante operada através de uma operação lógica E (AND) com o valor em $t0 é o resultado de fazer a extensão de zero de 0x0003 (conforme a semântica das operações lógicas com dado imediato) ou seja, 0x00000003. Segundo, as propriedades da operação Booleana E (AND) dizem que 1) 0 AND x = 0 e 2) 1 AND x = x. Assim fazer um AND bit a bit de qualquer coisa (conteúdo em $t0 na hora da execução) com 0x00000003 tem como efeito zerar todos os bits do resultado, exceto

os dois bits mais à direita que ficam como eram no número originalmente em **$t0**. Uma propriedade discutida em aula de números múltiplos de 4 representados em binário ou em complemento de 2 é que estes sempre têm os dois dígitos mais à direita sendo 00. Assim, o resultado de executar a instrução **andi $t0,$t0,3** será 0 se e somente se o número original for múltiplo de 4. Afirmativa V.

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