Prova P1 **Disciplina: Organização de Computadores**

Professor: Ney Laert Vilar Calazans

Aluno: 02/setembro/2022

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. Abaixo se mostra duas partes de uma listagem gerada pelo ambiente MARS como resultado da montagem de um programa do MIPS. Substitua as triplas interrogações pelo texto que deveria estar em seu lugar (existem 6 triplas **???,** nas linhas 1, 6 e 8). Isto implica gerar código objeto, e/ou gerar código intermediário e/ou gerar código fonte. Caso a instrução a ser colocada no lugar das interrogações seja um salto, expresse na área do código fonte/intermediário respectivamente o rótulo/endereço associados.

   Dica 1: Deem 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 em folha anexa os desenvolvimentos para obter os resultados, justificando-os.

```
   Endereço   Cód.Objeto  Código Intermediário              Código Fonte
[1]  0x00400078  0x02484007  ???                     44   ???
[2]  0x0040007c  0x00084102  srl $8,$8,0x00000004  45   srl    $t0,$t0,4
[3]  0x00400080  0x3c011001  lui $1,0x00001001     47   la     $t0,array
[4]  0x00400084  0x34280000  ori $8,$1,0x00000000
[5]  0x00400088  0x91090006  lbu $9,0x00000006($8) 48   lbu    $t1,6($t0)
[6]  0x0040008c  ???         ???                   49   xori   $t1,$t1,0xff
[7]  ...
                                                        loop:
[8]  0x004000d0  0x19200006  ???                   67   ???
[9]  0x004000d4  0x8d0b0000  lw $11,0x00000000($8) 68   lw     $t3,0($t0)
[10] 0x004000d8  0x016a5821  addu $11,$11,$10      69   addu   $t3,$t3,$t2
[11] 0x004000dc  0xad0b0000  sw $11,0x00000000($8) 70   sw     $t3,0($t0)
[12] 0x004000e0  0x25080004  addiu $8,$8,0x0000000471   addiu  $t0,$t0,4
[13] 0x004000e4  0x2529ffff  addiu $9,$9,0xffffffff72   addiu  $t1,$t1,-1
[14] 0x004000e8  0x08100034  j 0x004000d0          73   j      loop
                                                        end_add:
[15] 0x004000ec  0x27bdfffc  addiu $29,$29,0xffffff77   addiu  $sp,$sp,-4
```

2. (3,0 pontos) O programa em linguagem de montagem do MIPS abaixo faz um processamento bem específico. Observe a área de dados, analise a área de programa e responda o que se pede:

   (a) (1,5 pontos) Descreva em uma frase o que faz este programa, do ponto de vista de sua semântica. Comente (semanticamente) as linhas do programa (todas, ou pelo menos as mais relevantes);

   (b) (1,5 pontos) Observe as linhas [11] e [12]. Elas definem uma limitação para este programa. Identifique e discuta que limitação é esta. Seria possível resolvê-la incrementando o código do programa? Se sim, esboce com palavras uma maneira de fazê-lo, propondo uma modificação do laço existente entre as linhas [10]-[14].

```
[1]         .data
[2]   n:    .word  12
[3]   r:    .word  0
[4]         .text
[5]         la     $t0,n
[6]         lw     $t0,0($t0)
[7]         addiu  $t2,$zero,1
[8]         addiu  $t3,$zero,1
[9]         beq    $t0,$zero,f
[10]  den:  beq    $t0,$t3,f
[11]        mult   $t0,$t2
[12]        mflo   $t2
[13]        addiu  $t0,$t0,-1
[14]        j      den
[15]  f:    la     $t1,r
[16]        sw     $t2,0($t1)
[17]        li     $v0,10
[18]        syscall
```

3. (4,0 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 0,4 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 0,0 pontos.

a) ( ) A diretiva de montagem **.data** aceita pelo montador do ambiente MARS tem como efeito alocar um espaço na memória de dados exatamente do tamanho de uma palavra do processador MIPS, ou seja, 4 bytes.

b) ( ) Pseudo-instruções aceitas como válidas pelo montador do ambiente MARS podem gerar apenas uma ou duas instruções do MIPS como resultado de sua tradução durante o processo de montagem de um programa.

c) ( ) O modo de endereçamento pseudo-absoluto no MIPS é usado exclusivamente em instruções de salto para subrotina.

d) ( ) O código objeto **0x1E20FFF9** corresponde a uma instrução de salto condicional que, quando saltar, o faz para uma instrução que está exatamente **seis palavras da memória de programa** acima do salto.

e) ( ) A instrução **SLT $t0, $t1, $t2** é uma instrução de controle de fluxo de execução de instruções.

f) ( ) O modo de endereçamento base-deslocamento é a única maneira que instruções do processador MIPS usam para especificar endereços de acesso à memória de dados.

g) ( ) Somente existem instruções na arquitetura MIPS que escrevem em **0** ou **1** registrador do banco. Nenhuma instrução escreve em **2** ou mais registradores do banco.

h) ( ) O modo de endereçamento imediato, conforme usado no MIPS, obtém o operando constante de 32 bits a usar na instrução sempre acrescentando 16 bits em 0 à esquerda dos 16 bits menos significativos (bits 15 a 0, ou a metade direita) do código objeto da instrução.

i) ( ) Suponha que se executa a instrução **andi $t0,$t0,0xFFFF**. Assuma que antes de executar esta instrução, $t0 contém 0xAB7FF003. Após executar a instrução mencionada, $t0 conterá 0x0000F003.

j) ( ) Um vetor de 1000 caracteres imprimíveis representado de forma convencional (ou seja, cada caractere usando o mínimo espaço necessário para armazená-lo) no MIPS ocupa 1001 bytes.

Prova P1 **Disciplina: Organização de Computadores**

Professor: Ney Laert Vilar Calazans

Aluno: 02/setembro/2022

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. Abaixo se mostra duas partes de uma listagem gerada pelo ambiente MARS como resultado da montagem de um programa do MIPS. Substitua as triplas interrogações pelo texto que deveria estar em seu lugar (existem 6 triplas ???, nas linhas 1, 6 e 8). Isto implica gerar código objeto, e/ou gerar código intermediário e/ou gerar código fonte. Caso a instrução a ser colocada no lugar das interrogações seja um salto, expresse na área do código fonte/intermediário respectivamente o rótulo/endereço associados.

   Dica 1: Deem 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]   0x00400078  0x02484007  ???                    44        ???
[2]   0x0040007c  0x00084102  srl $8,$8,0x00000004  45        srl    $t0,$t0,4
[3]   0x00400080  0x3c011001  lui $1,0x00001001     47        la     $t0,array
[4]   0x00400084  0x34280000  ori $8,$1,0x00000000
[5]   0x00400088  0x91090006  lbu $9,0x00000006($8) 48        lbu    $t1,6($t0)
[6]   0x0040008c  ???         ???                    49        xori   $t1,$t1,0xff
[7]   ...
                                                              loop:
[8]   0x004000d0  0x19200006  ???                    67    ???
[9]   0x004000d4  0x8d0b0000  lw $11,0x00000000($8) 68        lw     $t3,0($t0)
[10]  0x004000d8  0x016a5821  addu $11,$11,$10      69        addu   $t3,$t3,$t2
[11]  0x004000dc  0xad0b0000  sw $11,0x00000000($8) 70        sw     $t3,0($t0)
[12]  0x004000e0  0x25080004  addiu $8,$8,0x0000000471        addiu  $t0,$t0,4
[13]  0x004000e4  0x2529ffff  addiu $9,$9,0xffffffff72        addiu  $t1,$t1,-1
[14]  0x004000e8  0x08100034  j 0x004000d0          73        j      loop
                                                              end_add:
[15]  0x004000ec  0x27bdfffc  addiu $29,$29,0xffffff77        addiu  $sp,$sp,-4
```

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

```
[1]   0x00400078  0x02484007  ???                    44     ???
```

O que se quer aqui é partir do código objeto dado e gerar os códigos intermediário e fonte da instrução na linha [1]. O ponto de partida é separar os 6 bits mais significativos do código objeto (os 6 bits mais à esquerda de 0x02 convertido para binário), o que dá 0000 00, ou 0. Isto indica que a instrução é do tipo R e que precisamos então, segundo a Tabela de apoio da

página A-36 do Apêndice A, observar o valor dos 6 bits menos significativos do código objeto dado (os 6 bits mais à direta de 0x07 convertido para binário), o que dá 000111, ou 7. Segundo a mesma Tabela, isto identifica que a instrução procurada é **SRAV**. Na página A-41 do mesmo Apêndice A acha-se o formato desta instrução, que é:

```
                          srav  Rd,Rt,Rs: ling. de montagem
                           0x0 Rs Rt Rd  0 7: cód. objeto
Número de bits/campo:       6   5  5  5  5 6
```

A partir daí basta extrair os valores dos três campos de 5 bits que definem os registradores operandos da **SRAV**. Partindo do código objeto em hexadecimal e convertendo-o para binário e separando os bits nos campos do formato, obtém-se o seguinte:

Hexa: 0x02484007

Binário: 0000 0010 0100 1000 0100 0000 0000 0111

Binário em campos: 000000 (0) 10010(Rs) 01000(Rt) 01000(Rd) 00000(0) 000111(7)

Agora ficam claros os valores de Rs (18 ou $s2), Rt (01000 ou 8 ou $t0) e Rd (01000 ou 8 ou $t0). Com estes valores a geração do código intermediário/fonte é direta. O intermediário é srav $8,$8,$18. O código fonte também é obtido de forma direta: srav $t0, $t0,$s2.

Resposta final:

```
[1]   0x00400078  0x02484007  srav $8,$8,$18      44     srav $t0,$t0,$s2
```

```
[6]   0x0040008c  ???         ???                 49     xori   $t1,$t1,0xff
```

O que se quer aqui é partir do código fonte dado e gerar os códigos intermediário e objeto da instrução na linha [6]. O ponto de partida é o formato da instrução XORI. Do Apêndice A (na página A-42) se extrai este formato:

```
                          xori  Rt,Rs,Imm: ling. de montagem
                          0xE Rs Rt  Imm: cód. objeto
Número de bits/campo:       6   5  5  16
```

Com este formato e com a tabela de correspondência de nomes de registradores, vemos que se pode obter o valor dos campos Rs, Rt e Imm de forma mais ou menos direta. Rs e Rt são o mesmo registrador ($t1) que é indicado pelo número 9 em 5 bits, ou seja, 01001. O campo Imm do código fonte corresponde ao valor 0xFF, que expresso em 16 bits fica 0000 0000 1111 1111. O campo mais à esquerda é a constante hexadecimal 0xE expressa em 6 bits, ou seja 001110. Com estes valores binários obtém-se facilmente os códigos objeto e intermediário. O código objeto em binário é: 001110 01001 01001 0000 0000 1111 1111. Juntando estes 32 bits e separando-os em grupos de 4 bits, a conversão para hexadecimal fica óbvia, gerando 0x392900FF. O código intermediário é obtido trocando nomes simbólicos de registradores pelos respectivos nomes numéricos, e expressando o dado imediato em hexadecimal 16 bits, gerando xori $9,$9,0x00FF

Resposta Final:

```
[6]  0x0040008c  0x392900ff   xori   $9,$9,0x00ff          49  xori   $t1,$t1,0xff
```

```
[8]   0x004000d0  0x19200006  ???                 67   ???
```

O que se quer aqui é partir do código objeto dado e gerar os códigos intermediário e fonte da instrução na linha [8]. O ponto de partida é separar os 6 bits mais significativos do código objeto (os 6 bits mais à esquerda de 0x19 convertido para binário), o que dá 0001 10, ou 6. Segundo a Tabela de apoio da página A-36 do Apêndice A, trata-se então da instrução BLEZ (a única que possui no 6 bits mais à esquerda de seu código objeto o valor 6). Na página A-45 do mesmo Apêndice A acha-se o formato desta instrução, que é:

```
                          blez  Rs, label: ling. de montagem
                           6  Rs 0 Offset: cód. objeto
Número de bits/campo:       6   5 5    16
```

Então, é necessário determinar qual registrador RS a instrução usa, tomando os 5 bits depois do valor 6 nos 6 bits mais à esquerda do código objeto. Isto fornece 01 001 (os 2 bits mais à esquerda são os dois bits mais à direita do dígito hexa 9 e os três bits seguinte vêm da conversão do bit hexa 2 do código objeto (0x2 correspondem em binário a 0010). Ou seja, Rs é o registrador $9 ou $t1. Quanto ao Offset, sabemos da semântica da instrução que ele é o número de linhas para atingir a linha contendo o label da instrução a partir da instrução que segue o BLEZ (que,

claro, usa o modo de endereçamento relativo para expressar para onde saltar). Os 16 bits mais à direita do código objeto contém a constante hexa 0x0006 que corresponde a 6 em decimal. Como o número é positivo, o salto é para a frente seis linhas a partir da linha [9] (alinha abaixo do BLEZ), ou seja, linha [15] é o destino, que contém o rótulo **end_add**. Agora fica simples gerar a resposta final

Resposta final:

```
[8] 0x004000d0  0x19200006  blez    $9,0x0006    67    blez   $t1,end_add
```

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

2. (3,0 pontos) O programa em linguagem de montagem do MIPS abaixo faz um processamento bem específico. Observe a área de dados, analise a área de programa e responda o que se pede:

   (a) (1,5 pontos) Descreva em uma frase o que faz este programa, do ponto de vista de sua semântica. Comente (semanticamente) as linhas do programa (todas, ou pelo menos as mais relevantes);

   (b) (1,5 pontos) Observe as linhas [11] e [12]. Elas definem uma limitação para este programa. Identifique e discuta que limitação é esta. Seria possível resolvê-la incrementando o código do programa? Se sim, esboce com palavras uma maneira de fazê-lo, propondo uma modificação do laço existente entre as linhas [10]-[14].

```
[1]          .data                 # Programa de Cálculo do Fatorial de n
[2]   n:     .word  12             # n é o número cujo fatorial será calculado
[3]   r:     .word  0              # r conterá o resultado, o fatorial de n (fat)
[4]          .text                 # O programa começa aqui
[5]          la     $t0,n
[6]          lw     $t0,0($t0)     # primeiro, $t0 recebe valor de n(depois n-1, n-2,...)
[7]          addiu  $t2,$zero,1    # inicializa $t2 com 1, o fatorial de 0
[8]          addiu  $t3,$zero,1    # inicializa $t3 com 1 o último multiplicando do fat
[9]          beq    $t0,$zero,f    # Se n=0, 0!=1 e vai para o fim
[10]  den:   beq    $t0,$t3,f      # Senão, se $t0 chegou a 1, vai para o fim, $t2 tem res
[11]         mult   $t0,$t2        # multiplica resultado até agora (em $t2) pelo fator,
                                   # que vai valendo n, n-1, n-2, ... ,2
[12]         mflo   $t2            # $t2 recebe a parte menos significativa da
                                   # multiplicação
[13]         addiu  $t0,$t0,-1     # gera novo fator em $t0
[14]         j      den            # e continua a realizar as sucessivas multiplicações
[15]  f:     la     $t1,r          # Quando chegar aqui, $t2 tem o fatorial de n
[16]         sw     $t2,0($t1)     # escreve o fatorial de n em r
[17]         li     $v0,10         # Para terminar, sai fora do programa
[18]         syscall
```

Solução da Questão 2 (3,0 pontos)

a) Este programa calcula o fatorial de n e escreve o resultado em r. Os comentários semânticos se encontram após as linhas de código acima.

b) A limitação dada pelas linhas [11] e [12] deriva do fato de que a multiplicação de 2 números de 32 bits necessita, no pior caso, de 64 bits para representar o produto. A instrução mult coloca o resultado em dois registradores de 32 bits especiais (localizados fora do banco de registradores) denominados HI e LO (que guardam, respectivamente os 32 bits mais significativos do produto e os 32 bits menos significativos do produto). Ora, como a linha [12] usa a instrução MFLO, que move para $t2 apenas a parte baixa do resultado da multiplicação, qualquer valor em HI é ignorado nos cálculos parciais. Isto faz com que o resultado do fatorial seja correto apenas para os número 0-12 (12! = 479.001.600, mas 13! = 6.227.020.800, enquanto o maior número natural representável com 32 bits é 4.294.967.295). Esboçando tratamentos para solucionar a questão, temos (quem esboçar uma ou mais destas opções tem a nota completa neste item):

   - Pior, mesmo que HI e LO fossem usados a cada passo, a faixa de valores corretos do cálculo do fatorial seria apenas de 0-20, pois o fatorial de 21 não cabe em 64 bits (20! = 2.432.902.008.176.640.000, mas 21! = 51.090.942.171.709.440.000, enquanto o maior número natural representável com 64 bits é 18.446.744.073.709.551.615)!!

- A solução para poder ter a faixa mais ampla de resultados corretos seriam um tanto complexa, pois seria necessário tomar cada resultado de 64 bits e multiplicar este pelo fator de 32 bits em $t0.
- O laço seria bem mais longo pelo processo de implementar a multiplicação de um número de 64 bits por um de 32 bits usando apenas os registradores de 32 bits que o MIPS possui.
- Alternativamente, seria possível testar se n está entre 13-20 para ver se o tratamento especial é necessário, calculando quando empregar os 64 bits da multiplicação.
- Também é possível no laço usar a instrução MFHI e testar quando este valor é diferente de 0, indicando a necessidade de tratamento especial.

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

3. (4,0 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 0,4 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 0,0 pontos.

Solução da Questão 3 (4,0 pontos)

a) (**F**) A diretiva de montagem .data aceita pelo montador do ambiente MARS tem como efeito alocar um espaço na memória de dados exatamente do tamanho de uma palavra do processador MIPS, ou seja, 4 bytes.
Justificativa: FALSO. A diretiva **.data** apenas define que as linhas após ela correspondem a definições do conteúdo da memória de dados; ela não reserva espaço nesta memória.

b) (**F**) Pseudo-instruções aceitas como válidas pelo montador do ambiente MARS podem gerar apenas uma ou duas instruções do MIPS como resultado de sua tradução durante o processo de montagem de um programa.
Justificativa: FALSO, sobretudo devido ao uso da palavra “apenas”. Foram vistos exemplos em aula dos dois casos citados (pseudo-instruções gerando 1 ou 2 instruções, tais como li e la), mas foi também visto em aula exemplos de pseudo-instruções que geram 3 instruções. Um exemplo é addu $t1,$t2,0xFFFFF.

c) (**F**) O modo de endereçamento pseudo-absoluto no MIPS é usado exclusivamente em instruções de salto para subrotina.
Justificativa: FALSO. Embora este modo seja usado na instrução JAL que é de salto para subrotina, existe pelo menos mais uma instrução onde ele é usado que não é de salto para subrotina, a instrução J.

d) (**V**) O código objeto 0x1E20FFF9 corresponde a uma instrução de salto condicional que, quando saltar, o faz para uma instrução que está exatamente **seis palavras da memória de programa acima do salto**.
Justificativa: VERDADEIRO. Os 6 primeiros bits do código objeto são, em binário, 000111, ou seja 7 em decimal/hexadecimal. Isto corresponde de fato ao código de uma instrução BGTZ (um tipo de salto condicional). Observando os 16 bits inferiores (0xFFF9), nota-se que se trata de um numeral que em complemento de 2 expressa o binário 1111111111111001 o número -7. Como este valor é somado ao valor do PC apontando para a palavra seguinte à instrução BGTZ, ao multiplicar -7 por 4 e somar ao PC, obtém-se o endereço da posição localizada exatamente 6 palavras acima da BGTZ.

e) (**F**) A instrução SLT $t0, $t1, $t2 é uma instrução de controle de fluxo de execução de instruções.
Justificativa: FALSO. Trata-se de uma instrução que testa se o valor em $t1 é menor que o valor em $t2, anotando o resultado do teste em $t0. Claro que isto pode depois ser usado com instruções de salto condicional para realizar controle de fluxo, mas SLT em si não se caracteriza como tal tipo de instrução.

f) (**V**) O modo de endereçamento base-deslocamento é a única maneira que instruções do processador MIPS usam para especificar endereços de acesso à memória de dados.
Justificativa: VERDADEIRO. Como a arquitetura do MIPS define que instruções não podem ocupar mais do que 32 bits e que endereços de memória ocupam 32 bits, é impossível armazenar um endereço inteiro em uma instrução. Além do mais a arquitetura do MIPS é do tipo LOAD-STORE. Embora haja outros modos de endereçamento para fazer acesso à memória de instruções, somente o base-deslocamento está disponível para acesso à memória de dados.

g) (**V**) Somente existem instruções na arquitetura MIPS que escrevem em 0 ou 1 registrador do banco. Nenhuma instrução escreve em 2 ou mais registradores do banco.
Justificativa: VERDADEIRO. Embora existam no MIPS instruções que escrevem informação útil em mais de um registrador (como a instrução JAL), isto nunca é feito para dois registradores do banco.

h) (**F**) O modo de endereçamento imediato, conforme usado no MIPS, obtém o operando constante de 32 bits a usar na instrução sempre acrescentando 16 bits em 0 à esquerda dos 16 bits menos significativos (bits 15 a 0, ou a metade direita) do código objeto da instrução.
Justificativa: FALSO. Este modo opera dependendo da instrução específica. Existem duas formas de gerar os 16 bits mais significativos do operando constante: por extensão de ZERO, como usado nas instruções lógicas como ANDI e ORI e por extensão de SINAL, como ocorre nas instruções ADDI e SLTIU. A extensão de ZERO funciona quase como a questão enuncia (acrescentando 16 ou mais bits em 0 à esquerda dos 16 bits menos significativos do código objeto da instrução). A extensão de SINAL, por outro lado, acrescenta ou 16 (ou mais) bits em 0 ou 16 (ou mais) bits em 1, sendo os 16 bits idênticos ao bit mais significativo presente no código objeto da instrução. A justificativa para esta funcionalidade é que a extensão de sinal preserva o sinal da constante de original na representação de 32 bits, caso estes sejam interpretados como números expressos na codificação complemento de 2.

i) (**V**) Suponha que se executa a instrução andi $t0,$t0,0xFFFF. Assuma que antes de executar esta instrução, $t0 contém 0xAB7FF003. Após executar a instrução mencionada, $t0 conterá 0x0000F003.
Justificativa: VERDADEIRO, pois **andi** trabalha, segundo o Apêndice A com extensão de 0 para produzir a constante de 32 bits a usar na operação lógica AND com $t0. Assim, a constante usada será 0x0000FFFF, que combinada com o conteúdo citado de $t0 vai zerar os 16 bits mais significativos da palavra e manter inalterados os 16 bits menos significativos, ou seja, escreverá 0x0000F003 em $t0, conforme diz a afirmativa.

j) (**V**) Um vetor de 1000 caracteres imprimíveis representado de forma convencional (ou seja, cada caractere usando o mínimo espaço necessário para armazená-lo) no MIPS ocupa 1001 bytes.
Justificativa: VERDADEIRO. Cada caractere imprimível vem do código ASCIIE, que usa 1 byte para representar cada caractere. Toda cadeia deve ter um caractere adicional no final para informar quando ela termina, tipicamente o caractere ASCIIE NULL (representado pelo byte por 0x00). Assim, o espaço ocupado em memória será 1000+1=1001 bytes.

Fim da Solução da Questão 3 (4,0 pontos)