Prova P1 **Disciplina: Organização de Computadores**

Professor: Ney Laert Vilar Calazans

Aluno: 29/abril/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 uma listagem gerada pelo ambiente MARS como resultado da montagem de um trecho de um programa. Substitua as triplas interrogações pelo texto que deveria estar em seu lugar (existem 6 triplas ???, nas linhas 2, 12 e 14). 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 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]  0x00400010  0xafbf0000  sw $31,0x0000($29)     9       sw     $ra,0($sp)
[2]  0x00400014  ???         ???                   10       jal    f
[3]  0x00400018  0x8fbf0000  lw $31,0x0000($29)    11       lw     $ra,0($sp)
[4]  0x0040001c  0x27bd0004  addiu $29,$29,0x00    12       addiu  $sp,$sp, 4
[5]  0x00400020  0x3c011001  lui $1,0x1001         13       la     $t0,r
[6]  0x00400024  0x34280004  ori $8,$1,0x0004
[7]  0x00400028  0xad020000  sw $2,0x000($8)       14       sw     $v0,0($t0)
[8]  0x0040002c  0x2402000a  addiu $2,$0,0x00a     15       li     $v0, 10
[9]  0x00400030  0x0000000c  syscall              16              syscall
[10] 0x00400034  0x27bdfff8  addiu $29,$29,0xffff 17   f:  addiu $sp,$sp,-8
[11] 0x00400038  0xafbf0000  sw $31,0x0000($29)   18       sw     $ra,0($sp)
[12] 0x0040003c  0xafa40004  ???                  19       ???
[13] 0x00400040  0x2c880001  sltiu $8,$4,0x0001   20       sltiu  $t0,$a0, 1
[14] 0x00400044  0x11000004  ???                  21       ???
[15] 0x00400048  0x24020001  addiu $2,$0,0x0001   22       addiu  $v0,$zero,1
[16] 0x0040004c  0x8fbf0000  lw $31,0x0000($29)   23       lw     $ra, 0($sp)
[17] 0x00400050  0x27bd0008  addiu $29,$29,0x0000 24       addiu  $sp,$sp,8
[18] 0x00400054  0x03e00008  jr $31               25       jr     $ra
[19] 0x00400058  0x2484ffff  addiu $4,$4,0xffff   26   re: addiu  $a0 $a0,-1
```

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) A linha **14** do código contém a pseudo-instrução **`blt`**, do inglês "*branch if less than*", ou "*salta se menor que*". Esta pseudo salta para o rótulo expresso pelo seu último operando se o primeiro registrador contiver valor menor que o segundo (considerando os números contidos nos registradores como inteiros representados em complemento de 2). Gere

um conjunto de uma ou mais instruções que substitua corretamente esta instância da pseudo-instrução na linha 14.

```
[1]            .data
[2]     A:     .word  38
[3]     R:     .word  0
[4]            .text
[5]            li     $t0,1
[6]            la     $t1,A
[7]            lw     $t1,0($t1)
[8]            li     $t2, 1
[9]     loop:  addi   $t2, $t2,1
[10]           beq    $t2, $t1,fim
[11]           or     $t3, $t1,$zero
[12]    div:   subu   $t3, $t3,$t2
[13]           beq    $t3, $t2,n_pr
[14]           blt    $t3, $t2,loop
[15]           j      div
[16]    n_pr:  li     $t0,0
[17]    fim:   la     $t4,R
[18]           sw     $t0,0($t4)
[19]           li     $v0,10
[20]           syscall
```

3. (2,0 pontos) Suponha que você possui uma descrição arquitetural completa do processador MIPS 2000, conforme descrito no Apêndice A do livro texto, e que é necessário estender esta arquitetura acrescentando uma nova instrução. A nova instrução chama-se **SUBU2**, e é especificada da seguinte forma:
   - Especificação da instrução **SUBU2**: trata-se de uma instrução similar à instrução **SUBU** do MIPS 2000 mas que, ao invés de subtrair um valor contido no registrador subtraendo do valor contido no registrador minuendo, ela subtrai **dois** valores armazenados em dois registradores subtraendos, especificados pelo programador, do registrador minuendo (também especificado pelo programador) e coloca o resultado em um quarto registrador (o registrador destino, igualmente especificado pelo programador).

   Pede-se: (1) proponha para a nova instrução um **formato fonte** e um **formato de código objeto** que façam com que sua codificação (da instrução **SUBU2**) não entre em conflito com o código de qualquer instrução existente na arquitetura MIPS 2000 original, descrita no Apêndice A; (2) mostre pelo menos **um exemplo de uma linha de programa em linguagem de montagem** do MIPS 2000 com esta nova instrução no formato fonte; (3) gere o **código objeto para a linha exemplo**, em hexadecimal.
   Observação: Siga estritamente os pressupostos da arquitetura. Isto significa que **o código objeto da nova instrução deve ocupar exatamente 32 bits**, e que **cada campo associado a um registrador deve ser de 5 bits**, de forma a permitir que o programador escolha cada registrador da instrução como qualquer registrador do banco de registradores do MIPS 2000.

4. (2,0 pontos) Verdadeiro ou Falso. Abaixo aparecem 4 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 0,5 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.
   a) ( ) Um processador com palavra de 16 bits possui um barramento de endereços de 26 fios para se comunicar com a memória e um registrador *program counter* (PC) com o mesmo número de bits, 26. Assuma que o processador emprega um modelo de memória que usa endereçamento a byte. Logo, o mapa de memória acessível aos programas deste processador é de 64Mbytes.
   b) ( ) Suponha que o registrador **$t1** contém o valor **0xFA45778D**. Nesta situação, após executar a instrução **andi $t1,$t1,0x80FF**, o conteúdo do registrador **$t1** passará a ser **0x0000808D**.
   c) ( ) Suponha que se executa a instrução **sh $t0,4($t1)**. Suponha também que os valores dos registradores **$t0** e **$t1** no momento que esta instrução é executada são, respectivamente **0x10010ABF**, e **0x10010060**. As únicas posições de memória que serão alteradas pela execução de **sh** serão os endereços de memória **0x10010064** e **0x10010065**, que terão como novos valores **0xBF** e **0X0A**.
   d) ( ) As instruções **lb** e **lbu** diferem pelo fato de ambas carregarem um byte no byte menos significativo do seu registrador destino, mas **lbu** faz extensão de 0s para os 24 bits mais significativos, enquanto **lb** mantém o valor original dos 24 bits superiores do registrador destino envolvido.

Prova P1 **Disciplina: Organização de Computadores**

Professor: Ney Laert Vilar Calazans

Aluno: 29/abril/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 uma listagem gerada pelo ambiente MARS como resultado da montagem de um trecho de um programa. Substitua as triplas interrogações pelo texto que deveria estar em seu lugar (existem 6 triplas ???, nas linhas 2, 12 e 14). 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 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]  0x00400010  0xafbf0000  sw $31,0x0000($29)      9   sw     $ra,0($sp)
[2]  0x00400014  ???            ???                 10   jal    f
[3]  0x00400018  0x8fbf0000  lw $31,0x0000($29)     11   lw     $ra,0($sp)
[4]  0x0040001c  0x27bd0004  addiu $29,$29,0x00     12   addiu  $sp,$sp, 4
[5]  0x00400020  0x3c011001  lui $1,0x1001          13   la     $t0,r
[6]  0x00400024  0x34280004  ori $8,$1,0x0004
[7]  0x00400028  0xad020000  sw $2,0x000($8)        14   sw     $v0,0($t0)
[8]  0x0040002c  0x2402000a  addiu $2,$0,0x00a      15   li     $v0, 10
[9]  0x00400030  0x0000000c  syscall               16   syscall
[10] 0x00400034  0x27bdfff8  addiu $29,$29,0xffff  17 f:addiu $sp,$sp,-8
[11] 0x00400038  0xafbf0000  sw $31,0x0000($29)     18   sw     $ra,0($sp)
[12] 0x0040003c  0xafa40004  ???                   19   ???
[13] 0x00400040  0x2c880001  sltiu $8,$4,0x0001     20   sltiu  $t0,$a0, 1
[14] 0x00400044  0x11000004  ???                   21   ???
[15] 0x00400048  0x24020001  addiu $2,$0,0x0001     22   addiu  $v0,$zero,1
[16] 0x0040004c  0x8fbf0000  lw $31,0x0000($29)     23   lw     $ra, 0($sp)
[17] 0x00400050  0x27bd0008  addiu $29,$29,0x0000  24   addiu  $sp,$sp,8
[18] 0x00400054  0x03e00008  jr $31                25   jr     $ra
[19] 0x00400058  0x2484ffff  addiu $4,$4,0xffff    26 re:addiu $a0 $a0,-1
```

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

```
[2]  0x00400014  ???          ???                 10   jal    f
```

O que se quer aqui é partir do código fonte dado gerar os códigos intermediário e objeto da instrução na linha [2]. O ponto de partida é ver o endereço de destino da instrução **jal**, que é 0x00400034 onde está o rótulo **f**. Este endereço vai para o código intermediário e serve para gerar os 26 bits do pseudo-endereço a constar no código objeto do jal, eliminando os dois bits mnais à direita e os quatro bits mais à esquerda, o que dá, em binário, 0000 0100 0000 0000 0000 0011 01. A estes 26 bits acrescente-se, à esquerda, os 6 bits que designam que a instrução é um **jal**.

Isto se obtém do Apêndice A (tabela A.10.2 ou na página A-47, onde consta o formato do **jal**). Estes 6 bits são o número 3, em binário 000011. O código objeto final obtém-se, em hexadecimal convertendo os 32 bits assim obtidos para hexadecimal, o que dá 0x0C10000D.

Resposta Final:

```
[2]   0x00400014   0x0c10000d          jal   0x00400034          10    jal    f
```

```
[12] 0x0040003c   0xafa40004   ???                  19    ???
```

O que se quer aqui é partir do código objeto dado e gerar os códigos intermediário e fonte da instrução na linha [12]. O ponto de partida é separar os 6 bits mais significativos do código objeto, o que dá 101011, ou 0x2B em hexadecimal ou 43 em decimal. Isto identifica que a instrução referente a esta linha é **sw**. Na página A-51 do Apêndice A acha-se o formato desta instrução, que é:

```
                          sw   Rs,Rt, endereço: ling. de montagem
                        0x2b Rs Rt offset   : cód. objeto
Número de bits/campo:    6   5  5  16
```

A partir daí basta extrair os valores dos três campos dos 26 bits restantes do código objeto. Partindo do código objeto em hexadecimal e convertendo-o para binário campo a campo, obtém-se os seguintes 26 bits: 11101 00100 0000000000000100. Do formato ficam agora claros os valores de Rs (11101, ou 29 ou $sp), Rt (00100 ou 4 ou $a0) e o offset ou deslocamento (0000000000000100 ou 4 em decimal ou 0x0004 em hexadecimal). Com estes valores a geração do código intermediário é direta. Este corresponde a sw $4,0x0004($29). Para o código fonte, a solução é direta, substituindo-se nomes numéricos de registradores pelos nomes simbólicos (opcional, claro) e (mais opcional ainda) substituindo o offset em hexa pelo decimal correspondente, ou seja, sw $a0,4($sp).

Resposta final:

```
[12]      0x0040003c   0xafa40004   sw      $4,0x0004($29)    19      sw     $a0,4($sp)
```

```
[14]      0x00400044   0x11000004   ???                  21    ???
```

O que se quer aqui é partir do código objeto dado e gerar os códigos intermediário e fonte da instrução na linha [14]. O ponto de partida é separar os 6 bits mais significativos do código objeto, o que dá 000100, ou 0x4 em hexadecimal ou 4 em decimal. Isto identifica que a instrução referente a esta linha é **beq**. Na página A-51 do Apêndice A acha-se o formato desta instrução, que é:

```
                          beq  Rs,Rt, label   : ling. de montagem
                          4    Rs Rt  offset  : cód. objeto
Número de bits/campo:     4    5  5     16
```

Dado o formato, extrai-se dos códigos numéricos dos registradores e o offset, que são respectivamente Rs= 01000 (ou 8 em hexa ou em decimal), Rt=00000 (ou 0 em hexa ou decimal) e 0x0004 (4 em decimal). Isto já permite produzir o código intermediário, que é **beq $8, $0, 0x0004**. Para o código fonte os nomes numéricos dos registradores são trocados pelos nomes simbólicos respectivos ($8=$t0 e $0=$zero) e calcula-se o endereço do rótulo (label) como aquele obtido pela soma do offset multiplicado por 4 (0x4*4 =0x10 ou 16 em decimal) com o endereço da instrução abaixo do **beq**, que em hexadecimal é 0x00400048, ou seja: 0x00400048+0x10 = 0x00400058, que é o endereço onde se encontra o rótulo **re**. Isto produz a resposta para o código fonte da linha 14.

Resposta Final:

```
[14]      0x00400044   0x11000004   beq $8,$0,0x00000004   21    beq    $t0,$zero,re
```

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) A linha **14** do código contém a pseudo-instrução **blt**, do inglês "branch if less than", ou "salta se menor que". Esta pseudo salta para o rótulo expresso pelo seu último operando se o primeiro registrador contiver valor menor que o segundo (considerando os números representados em complemento de 2). Gere um conjunto de uma ou mais instruções que implementam corretamente esta pseudo neste contexto.

```
[1]             .data
[2]       A:    .word  38           # Número a examinar a primalidade
[3]       R:    .word  0            # R recebe 1 se A for primo, e 0 caso contrário
[4]             .text
[5]             li     $t0,1        # Inicialmente, assume que A é primo, 1 em $t0
[6]             la     $t1,A        #
[7]             lw     $t1,0($t1)   # Carrega valor de A em $t1
[8]             li     $t2,1        # $t2 vai conter sempre o divisor de A
[9]       loop: addi   $t2,$t2,1    # Primeiro divisor testado é 2, depois 3... até A
[10]            beq    $t2,$t1,fim  # Se não achou divisor inteiro exato, é primo, fim
[11]            or     $t3,$t1,$zero# Copia A em $t3, p/ser usado como temp para div
[12]      div:  subu   $t3,$t3,$t2  # Testa se divisor cabe em A mais uma vez
[13]            beq    $t3,$t2,n_pr # Se coube, nro inteiro de vezes, A não é primo
[14]            blt    $t3,$t2,loop # Se resto menor que contador ($t3), tst novo div
[15]            j      div          # Senão, continua dividindo por este divisor
[16]      n_pr: li     $t0,0        # Chegando aqui, número não é primo, põe 0 em $t0
[17]      fim:  la     $t4,R
[18]            sw     $t0,0($t4)   # O que há em $t0 é escrito em R (0 ou 1)
[19]            li     $v0,10       # Linhas para terminar
[20]            syscall             # o programa
```

Solução da Questão 2 (3,0 pontos)

a) Este programa testa se o número inicialmente contido na posição de memória A é primo ou não, escrevendo 1 na posição de memória R quando ele for primo e escrevendo 0 em R, caso contrário. A detecção consiste em tentar dividir A sucessivamente por divisores inteiros entre 2 e ele (2, 3, 4, 5, ..., A-1). A divisão é realizada por subtrações sucessivas do divisor de A. Se alguma das subtrações sucessivas por um divisor >=2 e <A resultar em 0, é porque A é divisível pelo divisor em questão, logo não pode ser primo.

b) A linha `blt $t3, $t2, loop` pode ser transformada em duas instruções que lhe equivalem:

```
slt   $at, $t3, $t2       # coloca 1 em $at se $t3<$t2 (slt compara dois inteiros)
bne   $at, $zero, loop    # salta para loop $at = 0 (ou seja, se $at é 1)
```

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

3. (2,0 pontos) Suponha que você possui uma descrição arquitetural completa do processador MIPS 2000, conforme descrito no Apêndice A do livro texto, e que é necessário estender esta arquitetura acrescentando uma nova instrução. A nova instrução chama-se **SUBU2**, e é especificada da seguinte forma:

Especificação da instrução **SUBU2**: trata-se de uma instrução similar à instrução **SUBU** do MIPS 2000 mas que, ao invés de subtrair um valor contido no registrador subtraendo do valor contido no registrador minuendo, ela subtrai **dois** valores armazenados em dois registradores subtraendos, especificados pelo programador, do registrador minuendo (também especificado pelo programador) e coloca o resultado em um quarto registrador (o registrador destino, igualmente especificado pelo programador).

Pede-se: (1) proponha para a nova instrução um **formato fonte** e um **formato de código objeto** que façam com que sua codificação (da instrução **SUBU2**) não entre em conflito com o código de qualquer instrução existente na arquitetura MIPS 2000 original, descrita no Apêndice A; (2) mostre pelo menos **um exemplo de uma linha de programa** em linguagem de montagem do MIPS 2000 com esta nova instrução no formato fonte; (3) gere o **código objeto para a linha exemplo**, em hexadecimal.

Observação: Siga estritamente os pressupostos da arquitetura. Isto significa que **o código objeto da nova instrução deve ocupar exatamente 32 bits**, e que **cada campo associado a um registrador deve ser de 5 bits**, de forma a permitir que o programador escolha cada registrador da instrução como qualquer registrador do banco de registradores do MIPS 2000.

Solução da Questão 3 (2,0 pontos)

1) Sendo uma instrução com 4 operandos do tipo registrador, faz sentido (embora não seja necessário nesta questão) tentar manter um formato parecido com o das instruções tipo R (ADDU, SUBU etc.). Uma forma de fazer isto é manter o *opcode* ocupando os bits 31 a 26 do código objeto com valor 000000 (como muitas instruções tipo R) e usando um valor do campo *funct* (bits 5 a 0 do código objeto) não usado por outra instrução tipo R. Ora facilmente se percebe que há diversos destes códigos na sexta coluna da Figura A.10.2 do Apêndice A. Podemos escolher o valor 40 (decimal) desta coluna, que corresponde a um valor binário do campo *funct* igual a 101000 (40 decimal convertido para binário 6 bits). O que sobra no formato são 20 bits que podem ser usados como os 4 campos de 5 bits que especificam os 4 registradores operandos da SUBU2. Assim os formatos fonte e objeto podem ser definidos assim (parecido com o formato da SUBU). **Note-se que esta NÃO É a única solução possível** para este item.

```
                          subu2 Rd,Rm,Rs1,Rs2      : formato fonte
                           0    Rm Rs1 Rs2 Rd 0x28 : formato cód obj
Número de bits/campo: 6    5  5  5   5   6
```

2) Um exemplo de uso correto da instrução SUBU2 seria:

```
exsubu2:      subu2  $t0,$t1,$t2,$t3    # $t0 ← $t1 - $t2 - $t3
```

Aqui, $t1 é o minuendo, $t2 e $t3 são os dois subtraendos e $t0 é o registrador destino.

3) Dado os formatos fonte e objeto do item 1), o código objeto em binário seria: 000000 01001 01010 01011 01000 101000. Reagrupando de 4 em 4 bits e convertendo o código objeto em binário para hexadecimal, resulta em 0x012A5A28, que é o código objeto resposta.

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

4. (2,0 pontos) Verdadeiro ou Falso. Abaixo aparecem 4 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 0,5 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) (V) Um processador com palavra de 16 bits possui um barramento de endereços para se comunicar com a memória de 26 fios e um registrador *program counter* (PC) com o mesmo número de bits, 26. Assuma que o processador emprega um modelo de memória que usa endereçamento a byte. Logo, o mapa de memória acessível aos programas deste processador é de 64Mbytes.
      Explicação: Com 26 bits no PC é possível apontar para 2^{26} valores distintos ou seja $64*2^{20}$ posições, ou 64Mposições. Como o endereçamento é a byte, cada endereço contém apenas 1 byte e o mapa é de 64Mbytes.
   b) (F) Suponha que o registrador **$t1** contém o valor **0xFA45778D**. Nesta situação, após executar a instrução **andi $t1,$t1,0x80FF**, o conteúdo do registrador **$t1** passará a ser **0x0000808D**.
      Explicação: andi, como toda instrução lógica com dado imediato, trabalha com extensão de 0. Assim a constante 0x80FF é transformada em 0x000080FF e se faz o **AND** bit a bit deste valor com o conteúdo de $t1 (0xFA45778D). **AND** com 0 sempre dá 0 e **AND** com 1 sempre dá o valor do outro lado do 1. Lembrando que 0x8 é 1000 em binário e que 0x7 é 0111 em binário, o resultado da andi será 0x0000008D e não 0x0000808D.
   c) (V) Suponha que se executa a instrução **sh $t0,4($t1)**. Suponha também que os valores dos registradores **$t0** e **$t1** no momento que esta instrução é executada são, respectivamente **0x10010ABF**, e **0x10010060**. As únicas posições de memória que serão alteradas pela execução de **sh** serão os endereços de memória **0x10010064** e **0x10010065** que terão como novos valores **0xBF** e **0X0A**.
      Explicação: $t1 contém o endereço base de escrita da sh, que é 0x10010060, que deve ser somado com o deslocamento 0x4, produzindo o endereço inicial de escrita 0x10010064. Como se trata de uma instrução sh (*store half-word*), serão escritos dois bytes nos endereços 0x10010064 e 0x10010065, conforme mencionado no próprio enunciado da questão. Como empregamos aqui apenas endereçamento *little-endian* e como sh usa como operando os dois bytes menos significativos do registrador fonte, 0xBF será escrito na primeira posição (0x10010064) e 0x0A será escrito na segunda posição (0x10010065).
   d) (F) As instruções **lb** e **lbu** diferem pelo fato de ambas carregarem um byte no byte menos significativo do seu registrador destino, mas **lbu** faz extensão de 0s para os 24 bits mais significativos, enquanto **lb** mantém o valor original dos 24 bits superiores do registrador destino envolvido.

Explicação: A definição de lbu está correta, mas a de lb está errada: segundo se pode verificar na descrição da **lb** (no Apêndice A) esta instrução realiza a extensão de sinal sobre os 24 bits mais significativos.