Prova P1 **Disciplina: Organização de Computadores**

Professor: Ney Laert Vilar Calazans

Aluno: 15/setembro/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 um trecho de 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 6 triplas ???). Em alguns casos, isto implica gerar código objeto, enquanto em outros implica gerar código intermediário e/ou código fonte. Caso uma instrução a ser colocada no lugar das interrogações seja um salto, expresse 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  Cód.Intermediário          Cód. Fonte
[1]     ...
[2]     0x004000e4  ???         addu $2,$0,$3          105 e_g:    move   $v0,$v1
[3]     0x004000e8  0x03e00008  jr $31                106         jr     $ra
[4]     0x004000ec  0x8fa80004  lw $8,0x0004($29)      109 g_l:    lw     $t0,4($sp)
[5]     0x004000f0  0x8fa90008  lw $9,0x0008($29)      110         lw     $t1,8($sp)
[6]     0x004000f4  0x240a0000  addiu $10,$0,0x0000    111         li     $t2,0
[7]     0x004000f8  0x24050075  ???                    112         ???
[8]     0x004000fc  0x24020028  addiu $2,$0,0x0028     114         addiu  $v0,$zero,40
[9]     0x00400100  0x0000000c  syscall                116         syscall
[10]    0x00400104  0x1120fff7  ???                    118 l_g:    ???
[11]    0x00400108  0x8fa50008  lw $5,0x0008($29)      122         lw     $a1,8($sp)
[12]    0x0040010c  0x24a5ffff  addiu $5,$5,0xffff     123         addiu  $a1,$a1,-1
[13]    0x00400110  0x2402002a  addiu $2,$0,0x002a     125         addiu  $v0,$zero,42
[14]    0x00400114  0x0000000c  syscall                127         syscall
[15]    0x00400118  ???         sll $4,$4,0x0002       128         sll    $a0,$a0,2
[16]    0x0040011c  0x00882021  addu $4,$4,$8          129         addu   $a0,$a0,$t0
[17]    0x00400120  0x8c8b0000  lw $11,0x000($4)       130         lw     $t3,0($a0)
[18]    ...
```

2. (3,0 pontos) O programa em linguagem de montagem do MIPS abaixo faz um processamento bem específico. (a) Descreva em uma frase o que ele faz. (b) Aponte no código fonte **todas** as pseudo-instruções que nele existem. (c) Diga o que acontece com a área de memória que contém os dados do programa após a execução do mesmo, especificando se algo é escrito na memória, onde e que valor(es) é(são) escrito(s).

   Dicas: Nas linhas 7 e 8 as constantes imediatas são especificadas como caracteres ASCIIE, o que é aceito pelo montador MARS.

```
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         beq    $t2,'M',mM
8         bne    $t2,'m',nmM
9   mM:   addiu  $t2,$zero,'p'
10        sb     $t2,0($t0)
11  nmM:  addiu  $t0,$t0,1
12        j      l
13  f:    li     $v0,10
14        syscall
15
16        .data
17  s:    .asciiz      "Mamae me ama!"
```

3. (2,0 pontos) Defina uma sequência de instruções válidas do MIPS (pode ser apenas uma, ou pode ser uma sequência de 2 ou mais instruções) para executar cada um dos comportamentos descritos pelas duas pseudo-instruções abaixo.
   a) Pseudo 1:

```
beq   $t0, 10, xuxu     # Esta pseudo salta para o rótulo xuxu se o conteúdo do registrador $t0 for igual à constante 10 (em decimal claro!).
```

   b) Pseudo 2:

```
ori   $t1, $t2, 0xffff0000    # Esta pseudo carrega em $t1 o valor que existe em $t2, depois de transformar os 16 bits mais significativos do valor em $t2 em uns. O valor original em $t2 não é alterado pela instrução. Alternativamente pode-se descrever a funcionalidade desta pseudo como realizando o OU lógico da constante 0xFFFF0000 com o conteúdo de $t2, sendo o resultado escrito no registrador $t1.
```

4. (2,0 pontos) Verdadeiro ou Falso. Abaixo aparecem 5 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 pontos.

a) ( ) O modo de endereçamento pseudo-absoluto, conforme usado no MIPS, parte de um operando de 26 bits, acrescenta 4 bits 0 à esquerda e 2 bits 0 à direita do operando, gerando assim o endereço efetivo de memória a ser usado como operando na instrução que o emprega.

b) ( ) Um vetor de 5.000 inteiros representados de forma convencional no MIPS ocupa 20.000 bytes em memória.

c) ( ) Suponha que ao executar uma instrução **lw** que leu dados a partir do endereço de memória **0x10010018**, escreveu-se no registrador $t0 o número **0xA8C7D3A4**. Sabendo que a implementação do MIPS onde a instrução foi executada é *little endian*, os valores que estão armazenados nos endereços de memória **0x10010018**, **0x10010019**, **0x1001001A** e **0x1001001B** são, respectivamente **0x4A**, **0x3D**, **0x7C** e **0x8A**.

d) ( ) Suponha que no MIPS se executa a instrução **xori $t5,$t5,0xFFFF**. Assuma que antes de executar esta instrução, **$t5** contém **0xABADABAD**. Após executar a instrução, **$t5** conterá **0xABAD5452**.

e) ( ) O numeral inteiro decimal mais negativo que se pode armazenar em um registrador de dados do MIPS é -2.147.483.648.

Prova P1 Disciplina: Organização de Computadores

Professor: Ney Laert Vilar Calazans

Aluno: 15/setembro/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 um trecho de 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 6 triplas ???). Em alguns casos, isto implica gerar código objeto, enquanto em outros implica gerar código intermediário e/ou código fonte. Caso uma instrução a ser colocada no lugar das interrogações seja um salto, expresse 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  Cód.Intermediário          Cód. Fonte
[1]       ...
[2]       0x004000e4  ???         addu $2,$0,$3          105 e_g:    move   $v0,$v1
[3]       0x004000e8  0x03e00008  jr $31                 106         jr     $ra
[4]       0x004000ec  0x8fa80004  lw $8,0x0004($29)      109 g_l:    lw     $t0,4($sp)
[5]       0x004000f0  0x8fa90008  lw $9,0x0008($29)      110         lw     $t1,8($sp)
[6]       0x004000f4  0x240a0000  addiu $10,$0,0x0000    111         li     $t2,0
[7]       0x004000f8  0x24050075  ???                    112         ???
[8]       0x004000fc  0x24020028  addiu $2,$0,0x0028     114         addiu  $v0,$zero,40
[9]       0x00400100  0x0000000c  syscall                116         syscall
[10]      0x00400104  0x1120fff7  ???                    118 l_g:    ???
[11]      0x00400108  0x8fa50008  lw $5,0x0008($29)      122         lw     $a1,8($sp)
[12]      0x0040010c  0x24a5ffff  addiu $5,$5,0xffff     123         addiu  $a1,$a1,-1
[13]      0x00400110  0x2402002a  addiu $2,$0,0x002a     125         addiu  $v0,$zero,42
[14]      0x00400114  0x0000000c  syscall                127         syscall
[15]      0x00400118  ???         sll $4,$4,0x0002       128         sll    $a0,$a0,2
[16]      0x0040011c  0x00882021  addu $4,$4,$8          129         addu   $a0,$a0,$t0
[17]      0x00400120  0x8c8b0000  lw $11,0x000($4)       130         lw     $t3,0($a0)
[18]      ...
```

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

```
[2]       0x004000e4  ???         addu $2,$0,$3          105 e_g:    move   $v0,$v1
```

O código fonte contém uma pseudo-instrução (move $v0,$v1), que foi traduzida no código intermediário para uma única instrução (addu $2,$0,$3). Assim, basta gerar o código objeto desta instrução. O formato da instrução **addu**, retirado do Apêndice A é:

```
                              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   :
```

O código objeto em 32 bits então é facilmente gerado, **atentando apenas** para a ordem de apresentação dos registradores, que é diferente nos formatos fonte (ling. de montagem) e objeto. Concatena-se 000000 (0 em 6 bits) com o endereço do **rs** no banco, 00000 (0 em binário 5 bits código do registrador $0 ou $zero), com o endereço do **rt** no banco, 00011 ($3=$v1, ou 3 em 5 bits), com o o endereço do **rd** no banco, 00010 ($2=$v0 ou 2 em 5 bits), com 00000 (constante 0 em 5 bits) e com 100001 (correspondendo a 0x21 em 6 bits). O resultado é então 0000 0000 0000 0011 0001 0000 0010 0001. Convertendo este código de 32 bits em hexadecimal, tem-se o formato final do código objeto, solução desta questão: 0x00031021.

Resposta final:

```
[2] 0x004000e4  0x00031021    addu $2,$0,$3    105 e_g:  move   $v0,$v1

[7]    0x004000f8  0x24050075  ???                    112         ???
```

O que se deseja 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, entra-se com os 6 primeiros bits do código objeto (001001, ou 9) na Tabela da Figura A.10.2 do Apêndice A, o que identifica a instrução como sendo um **addiu**. Com esta descoberta, usa-se novamente o Apêndice A para dele extrair o formato da instrução, qual seja:

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

A geração do código intermediário a partir daqui é simples, basta extrair os bits dos campos do formato, obter o número dos registradores e o valor da constante imediata. Depois, pode-se usar a tabela no início da prova para relacionar o número dos registradores com seu nome e daí gerar o código fonte. Assim, tem-se **rs**=00000 (bits 25-21 do código objeto), **rt**=00101 (bits 20-16 do código objeto) e **imm**=0x0075=117.

Resposta Final:

```
[7] 0x004000f8  0x24050075  addiu $5,$0,0x0075      112      addiu      $a1,$zero,117

[10]    0x00400104  0x1120fff7  ???                    118 l_g:    ???
```

Novamente, o que se deseja aqui é gerar os códigos intermediário e fonte de uma instrução, dado apenas seu código objeto, outra operação de desmontagem de código. Do Apêndice A com os seis primeiros bits do código objeto (000100) descobre-se que se trata de uma instrução **beq**. Partindo deste fato, obtém-se o formato da instrução do mesmo Apêndice A, qual seja:

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

A geração do código intermediário começa extraindo os bits dos campos do formato, obtendo-se o número dos registradores (01001, que gera $9 e 00000, que gera $0) e o valor do deslocamento (o **offset**, que é 0xFFF7, corresponde ao número -9 em complemento de 2, 16 bits). Logo, o código intermediário é **beq $9,$0,0xFFF7**. O offset serve para obter o rótulo para onde a instrução salta (especificado no código fonte). Partindo da linha abaixo do beq em questão, conta-se 9 instruções para cima (deslocamento negativo de nove linhas), o que termina por apontar para a linha [1] do trecho de código, onde existe o rótulo **e_g**.

Resposta Final:

```
[10] 0x00400104  0x1120fff7     beq $9,$0,0xfff7     118        beq  $t1,$zero,e_g

[15]    0x00400118  ???         sll $4,$4,0x0002       128         sll    $a0,$a0,2
```

O que se deseja aqui é apenas gerar o código objeto, dados os códigos fonte e intermediário de uma instrução. Usando o Apêndice A obtém-se o formato da instrução **sll**, que é:

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

Para obter o código objeto, há que se gerar os código para cada um dos campos não constantes. O código de 32 bits é formado pela concatenação de 000000 (0 em 6 bits) com 00000 (0 em 5 bits), com 00100 (4 em 5 bits, o índice do registrador **rt**, que é $4=$a0), com 00100 (4 em 5 bits, o índice do registrador **rd**, que é $4=$a0), com 00010 (2 em 5 bits, o campo **shamt** ou quantidade de posições de bits a deslocar) concatenado com 000000 (0 em 6 bits). Isto produz o código 0000 0000 0000 0100 0010 0000 1000 0000, ou **0x00042080**, que é a resposta da questão.

Resposta Final:

```
[15]   0x00400118  0x00042080  sll   $4, $4, 0x0002     128       sll    $a0, $a0, 2
```

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. (a) Descreva em uma frase o que ele faz. (b) Aponte no código fonte **todas** as pseudo-instruções que nele existem. (c) Diga o que acontece com a área de memória que contém os dados do programa após a execução do mesmo, especificando se algo é escrito na memória, onde e que valor(es) é(são) escrito(s).

   Dicas: Nas linhas 7 e 8 as constantes imediatas são especificadas como caracteres ASCIIE, o que é aceito pelo montador MARS.

```
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          beq     $t2,'M',mM
8          bne     $t2,'m',nmM
9    mM:   addiu   $t2,$zero,'p'
10         sb      $t2,0($t0)
11   nmM:  addiu   $t0,$t0,1
12         j       l
13   f:    li      $v0,10
14         syscall
15
16         .data
17   s:    .asciiz       "Mamae me ama!"
```

Solução da Questão 2 (3,0 pontos)

a) Este programa substitui todas as instâncias de caracteres `M´ e `m´ na cadeia s pelo caracter `p´.
b) As cinco linhas que contêm pseudo-instruções estão salientadas no código acima (linhas 3, 4, 7, 8 e 13). Os mnemônicos la e li sempre são pseudo-instruções. Embora **beq** e **bne** sejam mnemônicos de instruções, estas pressupõem que os seus dois primeiros operandos sejam registradores do banco (como na linha 6) e não um registrador e uma constante, como aparece nas linhas 7 e 8. Logo, estas últimas linhas contêm pseudo-instruções.
c) A área de dados do programa é alterada nas posições da cadeia **s** onde existem caracteres `M´ ou `m´. A cadeia s ao final do processamento contém “papae pe apa!”.

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

3. (2,0 pontos) Defina uma sequência de instruções válidas do MIPS (pode ser apenas uma, ou pode ser uma sequência de 2 ou mais instruções) para executar cada um dos comportamentos descritos pelas duas pseudo-instruções abaixo.
   a) Pseudo 1:

```
beq   $t0, 10, xuxu     # Esta pseudo salta para o rótulo xuxu se o
conteúdo do registrador $t0 for igual à constante 10 (em decimal claro!).
```

   b) Pseudo 2:

```
ori   $t1, $t2, 0xffff0000    # Esta pseudo carrega em $t1 o valor
que existe em $t2, depois de transformar os 16 bits mais significativos do
valor em $t2 em uns. O valor original em $t2 não é alterado pela instrução.
Alternativamente pode-se descrever a funcionalidade desta pseudo como
realizando o OU lógico da constante 0xFFFF0000 com o conteúdo de $t2,
sendo o resultado escrito no registrador $t1.
```

Solução da Questão 3 (2,0 pontos)

a) A funcionalidade da pseudo `beq $t0,10, xuxu` pode ser implementada pela sequência de duas instruções:

```
addiu $at, $zero, 10
beq   $t0, $at, xuxu
```

b) A funcionalidade da pseudo `ori $t1, $t2, 0xffff0000` pode ser implementada pela sequência de duas instruções:

```
lui   $at, 0xffff
or    $t1, $t2, $at
```

Observações: Em a), basta usar o registrador temporário do montador (`$at`), nele inserindo a constante 10 com a primeira instrução (`addiu`). Em seguida, basta realizar o `beq`, trocando a constante da pseudo pelo registrador `$at`. Isto faz com que o `beq` seja agora uma instrução reconhecível pelo MIPS. Em b), a instrução `lui` sozinha pode gerar a constante de 32 bits `0xffff0000` no mesmo registrador `$at`, pois o `lui` zera a parte baixa e carrega seu operando na parte alta de seu registrador destino. Depois disto, usa-se uma instrução `or` no lugar do `ori` para gerar o resultado e escrevê-lo no registrador destino `$t1`.

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

4. (2,0 pontos) Verdadeiro ou Falso. Abaixo aparecem 5 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 pontos.

a) ( ) O modo de endereçamento pseudo-absoluto, conforme usado no MIPS, parte de um operando de 26 bits, acrescenta 4 bits 0 à esquerda e 2 bits 0 à direita do operando, gerando assim o endereço efetivo de memória a ser usado como operando na instrução que o emprega.

b) ( ) Um vetor de 5.000 inteiros representados de forma convencional no MIPS ocupa 20.000 bytes em memória.

c) ( ) Suponha que ao executar uma instrução **lw** que leu dados a partir do endereço de memória **0x10010018**, escreveu-se no registrador $t0 o número **0xA8C7D3A4**. Sabendo que a implementação do MIPS onde a instrução foi executada é *little endian*, os valores que estão armazenados nos endereços de memória **0x10010018**, **0x10010019**, **0x1001001A** e **0x1001001B** são, respectivamente **0x4A**, **0x3D**, **0x7C** e **0x8A**.

d) ( ) Suponha que no MIPS se executa a instrução **xori $t5,$t5,0xFFFF**. Assuma que antes de executar esta instrução, **$t5** contém **0xABADABAD**. Após executar a instrução, **$t5** conterá **0xABAD5452**.

e) ( ) O numeral inteiro decimal mais negativo que se pode armazenar em um registrador de dados do MIPS é -2.147.483.648.

Solução da Questão 4 (2,0 pontos)

a) (**F**) O modo de endereçamento pseudo-absoluto, conforme usado no MIPS, parte de um operando de 26 bits, acrescenta 4 bits 0 à esquerda e 2 bits 0 à direita do operando, gerando assim o endereço efetivo de memória 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**

b) (**V**) Um vetor de 5.000 inteiros representados de forma convencional no MIPS ocupa 20.000 bytes em memória.

A afirmativa é Verdadeira, pois nas condições estabelecidas no item cada inteiro ocupa exatamente 4 byte. **V**

c) (**F**) Suponha que ao executar uma instrução lw que leu dados a partir do endereço de memória 0x10010018, escreveu-se no registrador $t0 o número 0xA8C7D3A4. Sabendo que a implementação do MIPS onde a instrução foi executada é little endian, os valores que estão armazenados nos endereços de memória 0x10010018, 0x10010019, 0x1001001A e 0x1001001B são, respectivamente 0x4A, 0x3D, 0x7C e 0x8A.

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 0x10010018, 0x10010019, 0x1001001A e 0x1001001B são, respectivamente 0xA4, 0xD3, 0xC7 e 0xA8. **F**

d) (**V**) Suponha que no MIPS se executa a instrução xori $t5,$t5,0xFFFF. Assuma que antes de executar esta instrução, $t5 contém 0xABADABAD. Após executar a instrução, $t5 conterá 0xABAD5452.

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 como operando 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 0xABAD – 1010 1011 1010 1101 bit a bit, é 0x5452 – 0101 0100 0101 0010). **V**

e) (**V**) O numeral inteiro decimal mais negativo que se pode armazenar em um registrador de dados do MIPS é -2.147.483.648.

Afirmativa verdadeira, pois com 32 bits os números inteiros possíveis de se representa vão de -2^{31} a $+2^{31}-1$. O número inteiro mais negativo representável é então 1000 0000 0000 0000 0000 0000 0000 0000 em binário, ou 0x80000000 em hexa ou, em decimal -2.147.483.648, que corresponde ao valor decimal de $-(2^{31})$. **V**

Fim da Solução da Questão 4 (2,0 pontos)