Prova P1 **Disciplina: Organização de Computadores**

Professor: Ney Laert Vilar Calazans

Aluno: 14/setembro/2017

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]   0x004000e4 ???         addu $2,$0,$3           105  e_g:   move   $v0,$v1
[2]   0x004000e8 0x03e00008  ???                     106  ???
[3]   0x004000ec 0x8fa80004  lw $8,0x00000004($29)  109         lw     $t0,4($sp)
[4]   0x004000f0 0x8fa90008  lw $9,0x00000008($29)  110         lw     $t1,8($sp)
[5]   0x004000f4 0x240a0000  addiu $10,$0,0x0000000 111         li     $t2,0
[6]   0x004000f8 0x24050075  addiu $5,$0,0x00000075 112         addiu  $a1,$zero,117
[7]   0x004000fc 0x24020028  addiu $2,$0,0x00000028 114         addiu  $v0,$zero,40
[8]   0x00400100 0x0000000c  syscall                 116         syscall
[9]   0x00400104 0x1120fff7  ???                     118  l_g:   ???
[10]  0x00400108 0x8fa50008  lw $5,0x00000008($29)  122         lw     $a1,8($sp)
[11]  0x0040010c 0x24a5ffff  addiu $5,$5,0xffffffff 123         addiu  $a1,$a1,-1
[12]  0x00400110 0x2402002a  addiu $2,$0,0x0000002a 125         addiu  $v0,$zero,42
[13]  0x00400114 0x0000000c  syscall                 127         syscall
[14]  0x00400118 0x00042080  sll $4,$4,0x00000002    128         sll    $a0,$a0,2
[15]  0x0040011c 0x00882021  addu $4,$4,$8           129         addu   $a0,$a0,$t0
[16]  0x00400120 0x8c8b0000  lw $11,0x00000000($4)   130         lw     $t3,0($a0)
[17]  0x00400124 0x240c0000  addiu $12,$0,0x0000000 136         li     $t4,0
[18]  0x00400128 0x11400006  beq $10,$0,0x00000006  137         beq    $t2,$zero,prim
[19]  0x0040012c ???         sw $30,0xfffffffc($30) 138         sw     $fp,-4($fp)
[20]  0x00400130 0x27de0008  addiu $30,$30,0x000000 140  cont:  addiu  $fp,$fp,8
[21]  0x00400134 0xafcbfff8  sw $11,0xfffffff8($30) 141         sw     $t3,-8($fp)
[22]  0x00400138 0xafccfffc  sw $12,0xfffffffc($30) 142         sw     $t4,-4($fp)
[23]  0x0040013c 0x2529ffff  addiu $9,$9,0xffffffff 143         addiu  $t1,$t1,-1
[24]  0x00400140 ???         ???                     144         j      l_g
[25]  0x00400144 0x001e1821  addu $3,$0,$30          147  prim:  move   $v1,$fp
[26]  0x00400148 0x240a0001  addiu $10,$0,0x0000000 148         li     $t2,1
[27]  0x0040014c 0x0810004c  j 0x00400130           149         j      cont
[28]  0x00400150 0x8fa80004  lw $8,0x00000004($29)  159         lw     $t0,4($sp)
```

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 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 de memória escreve e que valores escreve.

```
1           .text
2           .globl main
3  main:    move   $t0, $zero
4           move   $t1, $zero
5  loop:    mul    $t2, $t0, $t0
6           addu   $t1, $t1, $t2
7           addiu  $t0, $t0, 1
8           ble    $t0, 50, loop
9           la     $a0, str
10          jal    Ps
11          move   $a0, $t1
12          jal    Pi
13          li     $v0, 10
14          syscall
15  Ps:     li     $v0, 4
16          syscall
17          jr     $ra
18  Pi:     li     $v0, 1
19          syscall
20          jr     $ra
21          .data
22  str:    .asciiz "\n O resultado é: "
```

3. (3,0 pontos) No programa da Questão 2 existem diversas linhas que contêm pseudo-instruções. Algumas destas são bem conhecidas e foram usadas em aulas práticas, mas existe exatamente uma que não foi discutida, denominada **ble** (*branch if less or equal to*, **salta se menor ou igual a**). Em relação a esta nova pseudo-instrução, pede-se:
   a) Justifique porque se trata de uma pseudo-instrução e não de uma instrução;
   b) Traduza esta pseudo-instrução para uma ou para uma sequência de instruções do MIPS que são equivalentes a ela.

Prova P1 **Disciplina: Organização de Computadores**

Professor: Ney Laert Vilar Calazans

Aluno: 14/setembro/2017

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]     0x004000e4 ???         addu $2,$0,$3           105  e_g:   move   $v0,$v1
[2]     0x004000e8 0x03e00008  ???                     106  ???
[3]     0x004000ec 0x8fa80004  lw $8,0x00000004($29)   109         lw     $t0,4($sp)
[4]     0x004000f0 0x8fa90008  lw $9,0x00000008($29)   110         lw     $t1,8($sp)
[5]     0x004000f4 0x240a0000  addiu $10,$0,0x0000000 111         li     $t2,0
[6]     0x004000f8 0x24050075  addiu $5,$0,0x00000075 112         addiu  $a1,$zero,117
[7]     0x004000fc 0x24020028  addiu $2,$0,0x00000028 114         addiu  $v0,$zero,40
[8]     0x00400100 0x0000000c  syscall                 116         syscall
[9]     0x00400104 0x1120fff7  ???                     118  l_g:   ???
[10]    0x00400108 0x8fa50008  lw $5,0x00000008($29)   122         lw     $a1,8($sp)
[11]    0x0040010c 0x24a5ffff  addiu $5,$5,0xffffffff 123         addiu  $a1,$a1,-1
[12]    0x00400110 0x2402002a  addiu $2,$0,0x0000002a 125         addiu  $v0,$zero,42
[13]    0x00400114 0x0000000c  syscall                 127         syscall
[14]    0x00400118 0x00042080  sll $4,$4,0x00000002    128         sll    $a0,$a0,2
[15]    0x0040011c 0x00882021  addu $4,$4,$8           129         addu   $a0,$a0,$t0
[16]    0x00400120 0x8c8b0000  lw $11,0x00000000($4)   130         lw     $t3,0($a0)
[17]    0x00400124 0x240c0000  addiu $12,$0,0x0000000 136         li     $t4,0
[18]    0x00400128 0x11400006  beq $10,$0,0x00000006  137         beq    $t2,$zero,prim
[19]    0x0040012c ???         sw $30,0xfffffffc($30) 138         sw     $fp,-4($fp)
[20]    0x00400130 0x27de0008  addiu $30,$30,0x000000 140  cont:  addiu  $fp,$fp,8
[21]    0x00400134 0xafcbfff8  sw $11,0xfffffff8($30) 141         sw     $t3,-8($fp)
[22]    0x00400138 0xafccfffc  sw $12,0xfffffffc($30) 142         sw     $t4,-4($fp)
[23]    0x0040013c 0x2529ffff  addiu $9,$9,0xffffffff 143         addiu  $t1,$t1,-1
[24]    0x00400140 ???         ???                     144         j      l_g
[25]    0x00400144 0x001e1821  addu $3,$0,$30          147  prim:  move   $v1,$fp
[26]    0x00400148 0x240a0001  addiu $10,$0,0x0000000 148         li     $t2,1
[27]    0x0040014c 0x0810004c  j 0x00400130            149         j      cont
[28]    0x00400150 0x8fa80004  lw $8,0x00000004($29)   159         lw     $t0,4($sp)
```

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

```
[1]    0x004000e4  ???          addu $2,$0,$3          105  e_g:   move   $v0,$v1
```

O que se quer aqui é gerar o código objeto da instrução addu $2,$0,$3, dados seus códigos fonte (pseudo-instrução move $v0,$v1) e intermediário, ou seja, uma operação de montagem de código. Basta extrair do Apêndice A para o formato da instrução, 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 é imediata. Temos que rs=$0→00000, rt=$3→00011 e rt=$2→00010. Juntando todos os valores dos 6 campos, tem-se 000000 00000 00011 00010 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 0x00031021.

Resposta Final:

```
[1]    0x004000e4  0x00031021  addu  $2,$0,$3        105 e_g: move  $v0,$v1
```

```
[2]    0x004000e8  0x03e00008  ???                    106   ???
```

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 (001000), e usá-lo como índice na sexta coluna da mesma Tabela, o que identifica a instrução JR. Com esta descoberta, usa-se novamente o Apêndice A para dele extrair o formato da instrução, qual seja:

```
                                jr  rs
                           0x0  rs  0   8
Número de bits/campo:       6   5   15  6.
```

Dos bits 25-21 do código objeto extrai-se então **rs**=11111 (31, correspondendo ao registrador $31 ou **$ra**). Assim pode-se gerar os códigos intermediário e fonte.

Resposta final:

```
[2]    0x004000e8  0x03e00008    jr $31                106        jr    $ra
```

```
[9]    0x00400104  0x1120fff7  ???                    118  l_g:   ???
```

Novamente, solicita-se gerar os códigos intermediário e fonte desta instrução. 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**. O formato da instrução **beq**, retirado do Apêndice A é:

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

Em seguida, extrai-se os valores dos campos do código obeto: rs=01001=$9=$t1, rt=00000=$0=$zero e o deslocamento (offset) é 0xFFF7, ou seja -9 na base 10. Isto quer dizer que o salto é "para trás", ou seja, para uma linha acima do beq. Descobre-se que contado 9 linhas para cima a partir da linha seguinte ao beq chega-se à linha [1], endereço 0x004000e4, onde está o rótulo e_g do programa. Assim, podemos montar a resposta final como sendo:

Resposta Final:

```
[9]    0x00400104  0x1120fff7    beq    $9,$0,0xfff7   106        beq    $t1,$zero,e_g
```

```
[19]   0x0040012c  ???          sw $30,0xfffffffc($30) 138   sw     $fp,-4($fp)
```

O que se quer aqui é gerar o código objeto da instrução sw $fp,-4($fp), dados seus códigos fonte e intermediário, ou seja, uma operação de montagem de código. Obtém-se o formato da instrução do Apêndice A:

```
                              sw   rt,offset(rs)  : ling. de montagem
                              0x2b rs rt offset   : cód. objeto
Número de bits/campo:          6   5  5    16     :
```

Para gerar o código objeto já temos todos os valores no código intermediário, basta converter para binário o número do registrador $30 e os bits do código de operação. Assim, tem-se 0x2b= 101011, 30=11110 e o código objeto fica 1010 1111 1101 1110 (em hexa, isto é, 0xAFDE), que deve ser concatenado com o offset 0xFFFC. Isto produz a resposta final:

Resposta Final:

```
 [19]  0x0040012c  0xafdefffc    sw     $30,0xfffffffc($30)   138 sw     $fp,-4($fp)
```

```
--------------------------------------------------------------------------------
[24]    0x00400140  ???         ???                    144   j     l_g
```

Aqui, o objetivo é gerar os código intermediário e objeto da instrução j l_g. Parte-se do formato da instrução, que é:

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

Notando-se que target é um pseudo-endereço, e partindo-se do endereço completo associado ao rótulo l_g (linha [9] do trecho, endereço 0x00400104), pode-se gerar os 26 bits do pseudo-endereço removendo os 4 bits mais significativos e os 2 menos significativos do endereço original, o que dá, em binário, 0000 0100 0000 0000 0001 0000 01. Concatenando a esquerda destes 26 bits o código de operação convertido para binário tem-se os 32 bits do código objeto, qual seja, 000010 0000 0100 0000 0000 0001 0000 01. Agrupando estes de 4 em 4 e convertendo tudo para hexadecimal tem-se 0x08100041, o código objeto da instrução. A resposta final é então:

Resposta Final:

```
[24]    0x00400140  0x08100041     j 0x00400104            144    j      l_g
```

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. (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 de memória escreve e que valores escreve.

```
1          .text
2          .globl main
3    main: move   $t0, $zero          # $t0 começa com 0 e vai de 1 em 1 até 50
4          move   $t1, $zero          # $t1 inicia com 0 e armazena soma dos quadrados
5    loop: mul    $t2, $t0, $t0       # Aqui gera-se o quadrado a somar
6          addu   $t1, $t1, $t2       # Acumula quadrado gerado na soma
7          addiu  $t0, $t0, 1         # Gera novo número a calcular o quadrado
8          ble    $t0, 50, loop       # Se número não ultrapassou 50, volta a acumular
9          la     $a0, str            # $a0 ← endereço da mensagem inicial p/ imprimir
10         jal    Ps                  # Vai para rotina Ps, que imprime msg inicial
11         move   $a0, $t1            # Coloca soma de quadrados em $a0
12         jal    Pi                  # Vai para rotina Pi, que imprime a soma
13         li     $v0, 10             # Prepara-se para sair do programa
14         Syscall                    # E sai dele
15   Ps:   li     $v0, 4              # Rotina Ps, que chama serviço print-string
16         syscall                    # Executa rotina Ps
17         jr     $ra                 # Retorna para quem chamou
18   Pi:   li     $v0, 1              # Rotina Pi, que chama serviço print-int
19         Syscall                    # Executa rotina Pi
20         jr     $ra                 # Retorna para quem chamou
21         .data
22   str:  .asciiz "\n O resultado é: "     # Cadeia com texto explicativo
```

Solução da Questão 2 (3,0 pontos)

a) Este programa soma os quadrados de todos os naturais entre 0 e 50 e imprime o resultado desta soma.

b) O programa não escreve nada na memória de dados, notando-se que ele não possui nenhuma das instruções **sw**, **sh** ou **sb**.

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

3. (3,0 pontos) No programa da Questão 2 existem diversas linhas que contêm pseudo-instruções. Algumas destas são bem conhecidas e foram usadas em aulas práticas, mas existe exatamente uma que não foi discutida, denominada ble (*branch if less or equal to*, **salta se menor ou igual a**). Em relação a esta nova pseudo-instrução, pede-se:
   a) Justifique porque se trata de uma pseudo-instrução e não de uma instrução;
   b) Traduza esta pseudo-instrução para uma ou para uma sequência de instruções do MIPS que são equivalentes a ela.

Solução da Questão 3 (3,0 pontos)

a) Observando a estrutura das instruções de salto condicional (todas que começam com a letra ‘b’, tais como beq, bne, blez e bgez, estas operam sempre comparando ou dois registradores (caso de beq, bne) ou um registrador com a constante implícita 0 (caso do blez e bgez). A estrutura geral das instruções inclui 4 ou 3 campos, sendo o campo de código da operação (os 6 bits mais à esquerda, um ou dois campos de especificação do(s) registrador(es) e um campo de especificação do deslocamento em 16 bits. Ora, no caso da linha 8 do programa acima a comparação é entre um registrador e uma constante, o que implica que deveria haver mais um campo para conter uma constante, o que implicaria a necessidade de um formato diferente para esta instrução de salto condicional, o que normalmente não é usado em um processador RISC. Observe também que no Apêndice A não existe esta instrução como uma das instruções do MIPS.

b) Uma forma possível de produzir código equivalente a esta pseudo-instrução é primeiro gerar a constante no registrador temporário do montador ($at), realizar a comparação com uma instrução do grupo “set if less than”, para em seguida realizar o salto condicional. No caso específico em questão, um código que funciona assim é dado abaixo:

```
1  addiu        $at,$zero,51        # Note que gera-se 50+1=51 e não 50 em $at
2  slt          $at,$t0,$at         # $at← 1 se $t0 for <51, ou seja, <=50
3  bne          $at,$zero,loop      # salta para loop se $at=1, como se queria.
```

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