Prova P1 **Disciplina: Organização de Computadores**

Professor: Ney Laert Vilar Calazans

Aluno: 09/abril/2014

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 parte de uma listagem gerada pelo ambiente MARS como resultado da montagem de um programa (dois trechos separados). Pede-se: (a) Substituir 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 seja de salto, expresse o exato endereço para onde ela salta (em hexa ou com o rótulo associado à linha), caso isto seja parte das interrogações.
   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 o desenvolvimento para obter os resultados em folha(s) anexa(s), justificando este.
   Obrigatório: Mostre os desenvolvimentos para obter os resultados, justificando.

```
[1]     Endereço    Cód.Objeto  Cód.Intermediário          Cód. Fonte
[2]    0x00400064     ???      beq $9,$0,0x00000012   57        ???
[3]    0x00400068  0x8fa50008   lw $5,0x00000008($29)  61    lw     $a1,8($sp)
[4]    0x0040006c  0x24a5ffff   addiu $5,$5,0xffffffff 62    addiu  $a1,$a1,-1
[5]    0x00400070  0x2402002a   addiu $2,$0,0x0000002a 64    addiu  $v0,$zero,42
[6]    0x00400074  0x0000000c   syscall                 66    syscall
[7]    0x00400078     ???      sll $4,$4,0x00000002   67    sll    $a0,$a0,2
[8]    0x0040007c  0x00882021   addu $4,$4,$8          68    addu   $a0,$a0,$t0
[9]    0x00400080  0x8c8b0000   lw $11,0x00000000($4)  69    lw     $t3,0($a0)
[10]   0x00400084  0x240c0000   addiu $12,$0,0x0000000 75    li     $t4,0
[11]   0x00400088  0x11400006   beq $10,$0,0x00000006  76    beq    $t2,$zero,p
[12]   0x0040008c  0xafdefffc          ???              77         ???
[13]   0x00400090  0x27de0008   addiu $30,$30,0x000000 79 c:addiu  $fp,$fp,8
[14]   0x00400094  0xafcbfff8   sw $11,0xfffffff8($30) 80    sw     $t3,-8($fp)
[15]   0x00400098  0xafccfffc   sw $12,0xfffffffc($30) 81    sw     $t4,-4($fp)
[16]   0x0040009c  0x2529ffff   addiu $9,$9,0xffffffff 82    addiu  $t1,$t1,-1
[17]   0x004000a0  0x08100019   j 0x00400064           83    j      l_gera
[18]   0x004000a4  0x001e1821   addu $3,$0,$30         86 p:move   $v1,$fp
[19]   0x004000a8  0x240a0001   addiu $10,$0,0x0000000 87    li     $t2,1
[20]   0x004000ac  0x08100024   j 0x00400090          88    j      c
[21]   0x004000b0  0x00031021   addu $2,$0,$3          90 eg:move  $v0,$v1
[22]   0x004000b4     ???       jr $31                91    jr     $ra
```

2. (4,0 pontos) O programa em linguagem de montagem do MIPS abaixo faz um certo processamento bem específico. Descreva em uma frase o que este trecho de código faz. Comente o programa semanticamente. Existe(m) alguma(s) sub-rotina(s) que este programa chama? Lembre-se que chamadas do sistema não devem ser consideradas como chamadas de subrotina. Se de fato existe(m) subrotina(s), identifique em que linhas esta(s) se encontra(m), separando o código desta(s) do código do programa principal, pela citação do intervalo de linhas que cada um ocupa.

```
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, 100, 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) Na programa da Questão 2 existem diversas linhas que contêm pseudo-instruções. Algumas destas são bem conhecidas e bastante usadas em aula, mas existe exatamente uma que não foi discutida. Em relação a esta “nova” pseudo-instrução, pede-se:
   a) Identifique a linha que contém esta nova pseudo-instrução, justificando 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: 09/abril/2014

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 parte de uma listagem gerada pelo ambiente MARS como resultado da montagem de um programa (dois trechos separados). Pede-se: (a) Substituir 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 seja de salto, expresse o exato endereço para onde ela salta (em hexa ou com o rótulo associado à linha), caso isto seja parte das interrogações.
   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 o desenvolvimento para obter os resultados em folha(s) anexa(s), justificando este.
   Obrigatório: Mostre os desenvolvimentos para obter os resultados, justificando.

```
[1]     Endereço   Cód.Objeto  Cód.Intermediário          Cód. Fonte
[2]     0x00400064     ???      beq $9,$0,0x00000012   57        ???
[3]     0x00400068  0x8fa50008  lw $5,0x00000008($29)  61    lw     $a1,8($sp)
[4]     0x0040006c  0x24a5ffff  addiu $5,$5,0xffffffff 62    addiu  $a1,$a1,-1
[5]     0x00400070  0x2402002a  addiu $2,$0,0x0000002a 64    addiu  $v0,$zero,42
[6]     0x00400074  0x0000000c  syscall                66    syscall
[7]     0x00400078     ???      sll $4,$4,0x00000002   67    sll    $a0,$a0,2
[8]     0x0040007c  0x00882021  addu $4,$4,$8          68    addu   $a0,$a0,$t0
[9]     0x00400080  0x8c8b0000  lw $11,0x00000000($4)  69    lw     $t3,0($a0)
[10]    0x00400084  0x240c0000  addiu $12,$0,0x0000000 75    li     $t4,0
[11]    0x00400088  0x11400006  beq $10,$0,0x00000006  76    beq    $t2,$zero,p
[12]    0x0040008c  0xafdefffc          ???            77          ???
[13]    0x00400090  0x27de0008  addiu $30,$30,0x000000 79 c:addiu  $fp,$fp,8
[14]    0x00400094  0xafcbfff8  sw $11,0xfffffff8($30) 80    sw     $t3,-8($fp)
[15]    0x00400098  0xafccfffc  sw $12,0xfffffffc($30) 81    sw     $t4,-4($fp)
[16]    0x0040009c  0x2529ffff  addiu $9,$9,0xffffffff 82    addiu  $t1,$t1,-1
[17]    0x004000a0  0x08100019  j 0x00400064           83    j      l_gera
[18]    0x004000a4  0x001e1821  addu $3,$0,$30          86 p:move   $v1,$fp
[19]    0x004000a8  0x240a0001  addiu $10,$0,0x0000000 87    li     $t2,1
[20]    0x004000ac  0x08100024  j 0x00400090           88    j      c
[21]    0x004000b0  0x00031021  addu $2,$0,$3          90 eg:move  $v0,$v1
[22]    0x004000b4     ???      jr $31                 91    jr     $ra
```

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

```
[2]     0x00400064     ???      beq $9,$0,0x00000012   57        ???
```

A única informação fornecida é o código intermediário. Sabemos se tratar de instrução BEQ, cujo formato é:

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

O código objeto é muito fácil de gerar: 000100 (4 em seis bits) concatenado com o endereço do Rs no banco, 01001, concatenado com o endereço do Rt no banco, 00000, concatenado com o terceiro operando em 16 bits, ou seja 0000 0000 0001 0010. Juntando os 32 bits e traduzindo-os de 4 em 4 em valores hexadecimais, obtém-se 0x11200012. Para gerar o código fonte, é necessário descobrir o rótulo para onde o salto remete. Como o operando é 0x12=18 na base 10, a linha para onde se deve saltar está 18 linhas abaixo da linha abaixo do beq, ou seja, trata-se da linha 21, que possui o rótulo eg. Assim, tem-se

Resposta final:

```
[2] 0x00400064 0x11200012  beq $9,$0,0x0012              beq $t1,$zer0,eg
```

```
[7]    0x00400078     ???      sll $4,$4,0x00000002   67    sll    $a0,$a0,2
```

O que se deseja aqui é gerar o código objeto de uma instrução sll, dados o código fonte e intermediário desta. Para realizar a montagem, consulta-se o formato da instrução SLL no Apêndice A, o que fornece:

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

Do código intermediário obtém-se os valores em decimal dos registradores, rt=rd=4=00100. O campo shamt é dado como sendo 2 ou 00010 em cinco bits. Logo, o código objeto é 000000 00000 00100 00100 00010 000000, ou em hexadecimal 0x00042080.

Resposta Final:

```
[7] 0x00400078  0x00042080    sll $4,$4, 0x2              sll    $a0,$a0,2
```

```
[12]   0x0040008c  0xafdefffc          ???          77        ???
```

O que se deseja aqui é gerar o código fonte e intermediário de uma instrução, dado apenas o código objeto. Observando-se os seis bits mais à esquerda deste código objeto (os seis primeiros bits de 0xaf), tem-se em binário 101011, que em hexadecimal corresponde a 0x2B, ou 43 em decimal. Entrando com um destes valores na Figura A.10.2 do Apêndice A descobre-se que se trata da instrução SW, cujo formato é:

```
                               sw   rt, offset(rs)
                             0x2b  rs   rt   offset
      Número de bits/campo:    6   5    5      16
```

Os 5 bits que correspondem ao Rs são (tirados em parte de 0xfd) 11110 ou em decimal 30 que corresponde ao registrador $fp. Os cinco bits seguintes correspondem ao registrador Rt e são 11110, ou seja outra referência ao mesmo registrador $fp. Finalmente, os 16 bits que correspondem ao offset são em hexa 0xfffc, ou seja 1111 1111 1111 1100 em binário. Isto corresponde a um número negativo em complemento de 2. Invertendo todos os bits e somando 1 obtém-se 0000 0000 0000 0100 ou seja +4. Então o offset é -4. Temos então todos os ingredientes para gerar a

Resposta Final:

```
[12] 0x0040008c  0xafdefffc      sw     $30,0xfffc($30)     sw     $fp,-4($fp)
```

```
[22]   0x004000b4      ???     jr $31                91    jr     $ra
```

O que se deseja aqui é gerar o código objeto de uma instrução jr, dados o código fonte e intermediário desta. Para realizar a montagem, consulta-se o formato da instrução JR no Apêndice A, o que fornece:

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

Logo, é muito simples gerar o código objeto, pois tudo é constante, exceto o código do registrador que contém o endereço de salto, que é 31 ou 11111 em binário. Monta-se o código objeto concatenando todos os vetores de bits na ordem do formato, ou seja 000000 11111 000000000000000 001000, o que em hexadecimal fica 0x03E00008.

Resposta Final:

```
[22] 0x004000b4  0x03e00008     jr $31              91    jr     $ra
```

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

2. (4,0 pontos) O programa em linguagem de montagem do MIPS abaixo faz um certo processamento bem específico. Descreva em uma frase o que este trecho de código faz. Comente o programa semanticamente. Existe(m) alguma(s) sub-rotina(s) que este programa chama? Lembre-se que chamadas do sistema não devem ser consideradas como chamadas de subrotina. Se de fato existe(m) subrotina(s), identifique em que linhas esta(s) se encontra(m), separando o código desta(s) do código do programa principal, pela citação do intervalo de linhas que cada um ocupa.

```
1            .text
2            .globl main
3   main:    move   $t0, $zero      # $t0 receberá números 0-100, inicia c/ 0
4            move   $t1, $zero      # $t1 guardará a soma dos quadrados, inicia com 0
5   loop:    mul    $t2, $t0, $t0   # gera próximo quadrado em $t2
6            addu   $t1, $t1, $t2   # acumula quadrado à soma atual de quadrados
7            addiu  $t0, $t0, 1     # gera em $t0 novo número
8            ble    $t0, 100, loop # enquanto não passou de 100, gera/acumula
9            la     $a0, str        # Aqui processamento acabou, prepara saída:
10           jal    Ps              # Põe ponteiro para texto em $a0 e chama Ps
11           move   $a0, $t1        # Põe valor acumulado em $a0
12           jal    Pi              # e chama rotina de impressão de inteiro P1
13           li     $v0, 10         # Ao final da impressão, sai do programa
14           syscall                # última linha, retorno ao SO.
15  # Subrotina de impressão de cadeia de caracteres
16  Ps:      li     $v0, 4              # escolhe chamada 4, impressão de cadeia
17           syscall                    # chamada em si
18           jr     $ra                 # depois de imprimir retorna a quem chamou
19  # Subrotina de impressão de inteiro
20  Pi:      li     $v0, 1              # escolhe chamada 1, impressão de inteiro
21           syscall                    # chamada em si
22           jr     $ra                 # depois de imprimir retorna a quem chamou
23           .data
24  str:     .asciiz "\n O resultado é: "
```

Solução da Questão 2 (4,0 pontos)

Este programa soma os quadrados de todos os números naturais entre 0 e 100.
Existem duas subrotinas no código, que são ambas do tipo folha, cada uma chamada apenas uma vez pelo programa principal. São elas Ps, que ocupa as linhas 15-17 e Pi, que ocupa as linhas 18 a 20.

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

3. (3,0 pontos) Na programa da Questão 2 existem diversas linhas que contêm pseudo-instruções. Algumas destas são bem conhecidas e bastante usadas em aula, mas existe exatamente uma que não foi discutida. Em relação a esta "nova" pseudo-instrução, pede-se:
   a) Identifique a linha que contém esta nova pseudo-instrução, justificando 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) A linha que contém a pseudo-instrução "nova" é a linha 8 do programa e a pseudo-instrução é a ble. Esta pseudo compara o valor de um dado imediato (neste caso, 100) ao conteúdo de um registrador do MIPS, e caso o dado seja menor ou igual ao conteúdo do registrador, salta-se para o rótulo contido na pseudo (neste caso, "loop"). A justificativa para esta ser uma pseudo-instrução é a seguinte: São três operandos, sendo que um é um registrador, um é um dado imediato e um é rótulo a partir do qual se pode calcular o endereço de salto para a linha deste rótulo. Como nenhum dos formatos do MIPS (R, I ou J) possui uma estrutura de campos capaz de comportar estas três informações, esta só pode ser uma pseudo-instrução.

b) Existem várias formas de traduzir esta pseudo para um conjunto de instruções, explica-se uma delas a seguir. Primeiro, é possível comparar um registrador com uma constante usando a instrução slti. Segundo, deve ser lembrado que a pseudo é um salto com base em uma comparação de magnitude "menor ou igual a", ao passo que slti identifica o resultado de um teste "menor que". Logo, alguma manipulação dos valores deve transformar o teste em um teste "menor ou igual a". Uma forma que funciona seria

transformar a linha **`ble $t0, 100, loop`** na seguinte sequência de linhas contendo apenas instruções MIPS:

```
addi    $at, $t0, -1        # Subtrai 1 de $t0
slti    $at, $at, 100       # Escreve 1 em $at se ($t0-1)<100, ou seja se $t0<=100
bne     $at, $zero, loop    # Finalmente se $t0<=100 (i.e. se $at=1) salta para loop
```

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