Prova P1 **Disciplina: Organização de Computadores**

Professor: Ney Laert Vilar Calazans

Aluno: 16/abril/2019

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 programa. Pede-se que se substituam as triplas interrogações pelo texto que deveria estar em seu lugar (existem 6 triplas ???, nas linhas 4, 9, 14 e 20). 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: 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]  0x00400094  0x00001021  addu $2,$0,$0          65   isq:   move   $v0,$zero
[2]  0x00400098  0x00044821  addu $9,$0,$4          66          move   $t1,$a0
[3]  0x0040009c  0x20080001  addi $8,$0,1          67          addi   $t0,$zero,1
[4]  0x004000a0  0x00084780  ???                  68          ???
[5]  0x004000a4  0x0128502a  slt $10,$9,$8         70 isq_b:   slt    $t2,$t1,$t0
[6]  0x004000a8  0x11400002  beq $10,$0,2          71          beq    $t2,$zero,isq_l
[7]  0x004000ac  0x00084082  srl $8,$8,2           72          srl    $t0,$t0,2
[8]  0x004000b0  0x08100029  j 0x004000a4          73          j      isq_b
[9]  0x004000b4  0x1100000a  ???                  75 isq_l:   ???
[10] 0x004000b8  0x00485820  add $11,$2,$8         76          add    $t3,$v0,$t0
[11] 0x004000bc  0x012b502a  slt $10,$9,$11        77          slt    $t2,$t1,$t3
[12] 0x004000c0  0x11400002  beq $10,$0,2          78          beq    $t2,$zero,isq_e
[13] 0x004000c4  0x00021042  srl $2,$2,1           79          srl    $v0,$v0,1
[14] 0x004000c8  ???         j 0x004000d8         80          j      isq_d
[15] 0x004000cc  0x012b4822  sub $9,$9,$11         82 isq_e:   sub    $t1,$t1,$t3
[16] 0x004000d0  0x00021042  srl $2,$2,1           83          srl    $v0,$v0,1
[17] 0x004000d4  0x00481020  add $2,$2,$8          84          add    $v0,$v0,$t0
[18] 0x004000d8  0x00084082  srl $8,$8,2           86 isq_d:   srl    $t0,$t0,2
[19] 0x004000dc  0x0810002d  j 0x004000b4          87          j      isq_l
[20] 0x004000e0  ???         jr $31               89 isq_r:   jr     $ra
```

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 este trecho de código faz, do ponto de vista semântico, comentando as linhas semanticamente (todas, ou pelo menos as mais relevantes);

(b) (1,0 ponto) Este programa escreve algo na memória de dados do processador? Caso afirmativo, diga em que posição(ões) de memória ele escreve e que valor(es) escreve;

(c) (0,5 pontos) O programa contém alguma sub-rotina? Se sim, diga onde ela(s) se encontra(m) e que linhas do código ela(s) abrange(m).

```
1.            .data
3.  t:        .asciiz "Mamae Me AmA!"
4.
5.            .text
6.            .globl main
7.  main:     la     $t0,t
8.  l:        lbu    $t1,0($t0)
9.            beq    $t1,$zero,f
10.           slti   $t2,$t1,0x7B
11.           beq    $t2,$zero,nelm
12.           slti   $t2,$t1,0x61
13.           bne    $t2,$zero,nelm
14.           addiu  $t1,$t1,-32
15.           sb     $t1,0($t0)
16. nelm:     addiu  $t0,$t0,1
17.           j      l
18. f:        la     $a0,t
19.           li     $v0,4
20.           syscall
21.           li     $v0,10
22.           syscall
```

3. (2,0 pontos) Responda às seguintes questões sobre o efeito da execução de instruções do MIPS:

a) (1 ponto) Suponha que ao executar uma instrução **lw** leu-se dados a partir do endereço de memória **0x10010014**, e escreveu-se no registrador $t0 o número **0xA6B7C8D9**. Assumindo-se que a implementação do MIPS onde a instrução foi executada é *little endian*, diga que valores estão armazenados em todos os endereços de memória lidos pela instrução **lw**;

b) (1 ponto) Suponha que no MIPS se executa a instrução **xori $t0,$t0,0xFC3F**. Assuma que antes de executar esta instrução, **$t0** contém **0xABADF003**. Após executar a instrução, o que haverá em **$t0**?

4. (2,0 pontos) Suponha que você possui uma descrição arquitetural completa do processador MIPS, 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 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) propor para a nova instrução um formato que faça com que a codificação não entre em conflito com o código de qualquer instrução existente na arquitetura MIPS descrita no Apêndice A; (2) mostrar pelo menos um exemplo de uma linha de programa em linguagem de montagem do MIPS com esta nova instrução; (3) gerar o código objeto para a linha exemplo, em hexadecimal.

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.

Prova P1 **Disciplina: Organização de Computadores**

Professor: Ney Laert Vilar Calazans

Aluno: 16/abril/2019

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 programa. Pede-se que se substituam as triplas interrogações pelo texto que deveria estar em seu lugar (existem 6 triplas ???, nas linhas 4, 9, 14 e 20). 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: 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]  0x00400094  0x00001021  addu $2,$0,$0          65   isq:   move   $v0,$zero
[2]  0x00400098  0x00044821  addu $9,$0,$4          66          move   $t1,$a0
[3]  0x0040009c  0x20080001  addi $8,$0,1          67          addi   $t0,$zero,1
[4]  0x004000a0  0x00084780  ???                  68          ???
[5]  0x004000a4  0x0128502a  slt $10,$9,$8          70 isq_b:  slt    $t2,$t1,$t0
[6]  0x004000a8  0x11400002  beq $10,$0,2          71          beq    $t2,$zero,isq_l
[7]  0x004000ac  0x00084082  srl $8,$8,2           72          srl    $t0,$t0,2
[8]  0x004000b0  0x08100029  j 0x004000a4           73          j      isq_b
[9]  0x004000b4  0x1100000a  ???                  75 isq_l:   ???
[10] 0x004000b8  0x00485820  add $11,$2,$8          76          add    $t3,$v0,$t0
[11] 0x004000bc  0x012b502a  slt $10,$9,$11        77          slt    $t2,$t1,$t3
[12] 0x004000c0  0x11400002  beq $10,$0,2          78          beq    $t2,$zero,isq_e
[13] 0x004000c4  0x00021042  srl $2,$2,1           79          srl    $v0,$v0,1
[14] 0x004000c8  ???        j 0x004000d8           80          j      isq_d
[15] 0x004000cc  0x012b4822  sub $9,$9,$11          82 isq_e:  sub    $t1,$t1,$t3
[16] 0x004000d0  0x00021042  srl $2,$2,1           83          srl    $v0,$v0,1
[17] 0x004000d4  0x00481020  add $2,$2,$8           84          add    $v0,$v0,$t0
[18] 0x004000d8  0x00084082  srl $8,$8,2           86 isq_d:  srl    $t0,$t0,2
[19] 0x004000dc  0x0810002d  j 0x004000b4          87          j      isq_l
[20] 0x004000e0  ???        jr $31                 89 isq_r:  jr     $ra
```

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

```
[4]    0x004000a0   0x00084780 ???                68          ???
```

O que se quer aqui é partir do código objeto dado e gerar os códigos intermediário e fonte da instrução na linha [4]. O ponto de partida é separar os 6 bits mais significativos do código objeto, o que dá 000000. Isto corresponde em decimal ou hexa ao numeral 0. Com este valor, entra-se na primeira coluna da Figura A.10.2 do Apêndice A e descobre-se que é necessário observar os bits

(5 downto 0) do código objeto, que são 000000. Assim, descobre-se que o código objeto é o da instrução **sll**. Obtém-se então o formato da instrução **sll** do Apêndice A (página A-41):

```
                              sll  Rd, Rt, shamt  : ling. de montagem
                               0  0 Rt Rd shamt 0      : cód. objeto
Número de bits/campo:          6  5  5  5   5   6
```

A partir daí basta extrair os valores dos três campos dos 20 bits intermediários do código objeto. Partindo do código objeto em hexadecimal e convertendo-o para binário campo a campo, obtém-se: 000000 00000 01000 01000 11110 000000. Este formato deixa claros os valores de Rt (01000, ou 8 ou $t0), Rd (01000 ou 8 ou $t0) e shamt (11110 ou 30 em decimal). Com estes valores a geração dos códigos intermediário e fonte é direta, ver resposta abaixo.

Resposta Final:

```
[4]    0x004000a0  0x00084780  sll $8,$8,30        68            sll    $t0,$t0,30
```

```
[9]    0x004000b4  0x1100000a  ???                 75 isq_l:     ???
```

O que se quer aqui é novamente partir do código objeto dado e gerar os códigos intermediário e fonte da instrução na linha [9]. O ponto de partida é separar os 6 bits mais significativos do código objeto, o que dá 000100, ou 4 em decimal. Com este valor, entra-se na primeira coluna da Figura A.10.2 do Apêndice A e descobre-se que se trata da instrução **beq**. Obtém-se então o formato desta instrução do Apêndice A (página A-44):

```
                              beq  Rs, Rt, label  : ling. de montagem
                               4  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: 000100 01000 00000 0000000000001010. Este formato deixa claros os valores de Rs (01000, ou 8 ou $t0), Rt (00000 ou 0 ou $zero) e o offset ou deslocamento (0000000000001010 ou 10 em decimal ou 0x000A em hexadecimal). Com estes valores a geração do código intermediário é direta. Este corresponde a beq $8,$0, 10. Para o código fonte, deve-se descobrir o rótulo a que este se refere. Isto se obtém contando 10 (valor do deslocamento) linhas a partir da instrução que segue o beq, o que aponta para a linha [20] do código, onde se encontra o rótulo isq_r, que é o rótulo (label) procurado. Assim, o código fonte é beq $t0,$zero, isq_r.

Resposta final:

```
[9]    0x004000b4  0x1100000a  beq     $8,$0,10    75 isq_l:     beq $t0,$zero,isq_r
```

```
[14]   0x004000c8  ???         j 0x004000d8        80            j     isq_d
```

Aqui, deseja-se gerar o código objeto de uma instância da instrução j, dados seu código fonte e seu código intermediário. Usa-se a formato da instrução j, retirado do Apêndice A (página A-47) que é:

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

Note-se que o código intermediário já contém o endereço efetivo para onde o salto é realizado. Para gerar o código objeto, basta concatenar os 6 bits do código de operação ou opcode (2 ou 000010 em 6 bits) com os 26 bits obtidos do endereço constante no código intermediário (0x004000d8), pela remoção dos 2 bits menos significativos deste (divisão por 4) e dos 4 bits mais significativos do mesmo. Traduzindo o endereço para binário obtém-se 0000 0000 0100 0000 0000 0000 1101 1000. Removendo os bits mencionados obtém-se os seguintes 26 bits: 00000100000000000000110110. Concatenando à esquerda destes o opcode 000010, obtém-se 00001000000100000000000000110110. Agora, agrupando os 32 bits assim obtidos de 4 em 4 (0000 1000 0001 0000 0000 0000 0011 0110) e convertendo cada grupo de 4 bits para hexa obtém-se o código objeto final, 0x08100036.

Resposta Final:

```
[14]   0x004000c8  0x08100036      j 0x004000d8        80            j     isq_d
```

```
[20]   0x004000e0  ???         jr $31              89 isq_r:     jr    $ra
```

Aqui, deseja-se gerar o código objeto de uma instância da instrução **jr**, dados seu código fonte e seu código intermediário. Usa-se a formato da instrução **jr**, retirado do Apêndice A (página A-47) que é:

```
                          jr   Rs          : ling. de montagem
                          0   Rs   0    8   : cód. objeto
Número de bits/campo:     6    5   15   6   :
```

Para gerar o código objeto basta concatenar os valores dos 4 campos. Destes, 3 são constantes e o campo RS é dado nos códigos intermediário e fonte fornecidos na questão. Logo, gerar o código objeto é um processo direto de conversão de 4 valores para binário: 000000 11111 000000000000000 001000. Reagrupando os bits de 4 em 4, obtém-se 0000 0011 1110 0000 0000 0000 0000 1000. Convertendo os grupos de 4 bits em hexadecimal produz o código objeto procurado, qual seja, 0x03E00008.

Resposta Final:

```
[20]    0x004000e0  0x03e00008     jr $31                  89 isq_r:       jr     $ra
```

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. 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 este trecho de código faz, do ponto de vista semântico, comentando as linhas semanticamente (todas, ou pelo menos as mais relevantes);

   (b) (1,0 ponto) Este programa escreve algo na memória de dados do processador? Caso afirmativo, diga em que posição(ões) de memória ele escreve e que valor(es) escreve;

   (c) (0,5 pontos) O programa contém alguma sub-rotina? Se sim, diga onde ela(s) se encontra(m) e que linhas do código ela(s) abrange(m).

```
1.              .data
2.   t:         .asciiz "Mamae Me AmA!"    # O único dado do programa, um texto ASCIIE
3.                                         # terminado pelo caracter NULL (byte 0x00)
4.              .text
5.              .globl main
6.   main:      la      $t0,t              # $t0 aponta para o início do texto
7.   l:         lbu     $t1,0($t0)         # $t1 recebe próximo caracter
8.              beq     $t1,$zero,f        # Cai fora do laço se atingiu caracter NULL
9.              slti    $t2,$t1,0x7B       # Se não no fim, testa se char pode ser minúscula
10.             beq     $t2,$zero,nelm     # Se não pode ser minúscula, vai p/ próximo char
11.             slti    $t2,$t1,0x61       # Aqui testa se efetivamente é minúscula
12.             bne     $t2,$zero,nelm     # Não sendo minúscula, vai p/ próximo char
13.             addiu   $t1,$t1,-32        # Aqui achou minúscula, transforma em maiúscula
14.             sb      $t1,0($t0)         # Escreve maiúscula sobre a  minúscula
15.  nelm:      addiu   $t0,$t0,1          # Avança ponteiro para próximo char,
16.             j       l                  # e volta para o início do laço
17.  f:         la      $a0,t              # No final, põe ponteiro para texto final em $a0
18.             li      $v0,4              # Prepara impressão do texto final,
19.             syscall                    # e imprime
20.             li      $v0,10             # Cai fora do programa
21.             syscall                    # aqui
```

Solução da Questão 2 (3,0 pontos)

a) Este programa trata a cadeia ASCIIE t, modificando todas as letras minúsculas desta para as maiúsculas correspondentes no alfabeto.

b) Sim, para cada caracter ASCIIE minúsculo em t escreve-se no seu lugar o caracter maiúsculo correspondente. Como t possui 6 letras minúsculas, seis bytes serão escritos pelo programa na memória. As posições de memória escritas (lembrando que se assume aqui: uma organização *little endian* de memória e que t é armazenado a partir do endereço 0x10010000) os endereços onde algo é escrito são:

- 0x10010001 (onde se escreve ‘A’),
- 0x10010002 (onde se escreve ‘M’),
- 0x10010003 (onde se escreve ‘A’),
- 0x10010004 (onde se escreve ‘E’),
- 0x10010007 (onde se escreve ‘E’) e
- 0x1001000A (onde se escreve ‘M’).

c) O programa não possui instruções jal (ou similar) nem jr $ra (ou similar). Logo, ele não possui sub-rotinas.

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

3. (2,0 pontos) Responda às seguintes questões sobre o efeito da execução de instruções do MIPS:

a) (1 ponto) Suponha que ao executar uma instrução **lw** leu-se dados a partir do endereço de memória **0x10010014**, e escreveu-se no registrador **$t0** o número **0xA6B7C8D9**. Assumindo-se que a implementação do MIPS onde a instrução foi executada é *little endian*, diga que valores estão armazenados em todos os endereços de memória lidos pela instrução **lw**;

b) (1 ponto) Suponha que no MIPS se executa a instrução **xori $t0,$t0,0xFC3F**. Assuma que antes de executar esta instrução, **$t0** contém **0xABADF003**. Após executar a instrução, o que haverá em **$t0**?

Solução da Questão 3 (2,0 pontos)

a) Tendo o MIPS um modelo de memória endereçado a byte, os endereços lidos são 4 a partir do endereço inicial, ou seja, 0x10010014, 0x10010015, 0x10010016 e 0x10010017. Sendo a organização little endian os valores originalmente armazenados são:
- 0xD9 em 0x10010014,
- 0xC8 em 0x10010015,
- 0xB7 em 0x10010016,
- 0xA6 em 0x10010017.

b) A operação executada é um OU-exclusivo bit a bit entre o valor originalmente em $t0 (0xABADF003) com o valor de 32 bits produzido a partir dos 16 bits do dado imediato da instrução por extensão de 0, ou seja 0x0000FC3F como (qquer coisa xor 0 = qquer coisa) e (qquer coisa xor 1 = qquer coisa negada) o efeito consiste em manter os 16 bits originais iguais (xor com 16 0s) e dos restantes 16 bits inverter os 6 bits mais significativos e inverter os 6 menos significativos (0xFC = 11111100 e 0x3F = 00111111), mantendo os demais 4 bits iguais. O resultado escrito em $t0 será então 0xABAD0C3C.

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

4. (2,0 pontos) Suponha que você possui uma descrição arquitetural completa do processador MIPS, 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 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) propor para a nova instrução um formato que faça com que a codificação não entre em conflito com o código de qualquer instrução existente na arquitetura MIPS descrita no Apêndice A; (2) mostrar pelo menos um exemplo de uma linha de programa em linguagem de montagem do MIPS com esta nova instrução; (3) gerar o código objeto para a linha exemplo, em hexadecimal.

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.

Solução da Questão 4 (2,0 pontos)

1) Sendo uma instrução com 4 operandos do tipo registrador, faz sentido 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 de 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 4 campos de 5 bits que especificam os registradores. Assim os formatos fonte e objeto podem ser definidos assim (parecido com o formato da SUBU)

```
                                subu2 Rd,Rm,Rs1,Rs2      : l. mont.
                                 0    Rm Rs1 Rs2 Rd  0x28: cód. obj.
Número de bits/campo:            6     5  5  5   5   6
```

2) Um exemplo de uso 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 e o registrador destino.

3) O código objeto em binário seria: 000000 01001 01010 01011 01000 101000. Reagrupando de 4 em 4 bits e convertendo para hexadecimal, resulta em 0x012A5A28, que é o código objeto resposta.

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