Prova P1 **Disciplina: Organização de Computadores**

Professor: Ney Laert Vilar Calazans

Aluno: 14/setembro/2018

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. 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 ???, nas linhas 5, 10, 13, 14 e 24). 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]   0x00400000 0x3c011001 lui $1,0x00001001     15  main:  la    $t1,s
[2]   0x00400004 0x34290000 ori $9,$1,0x00000000
[3]   0x00400008 0x3c011001 lui $1,0x00001001     16          la    $t2,s
[4]   0x0040000c 0x342a0000 ori $10,$1,0x00000000
[5]   0x00400010 ???        ???                   17  l_l:    lb    $t3,($t2)
[6]   0x00400014 0x11600004 beq $11,$0,0x00000004 18          beqz  $t3,e_ll
[7]   0x00400018 0x3c010000 lui $1,0x00000000     19          addu  $t2,$t2,1
[8]   0x0040001c 0x34210001 ori $1,$1,0x00000001
[9]   0x00400020 0x01415021 addu $10,$10,$1
[10]  0x00400024 ???        bgez $0,0xfffffffa    20          b     l_l
[11]  0x00400028 0x254affff addiu $10,$10,0xffffff 21 e_ll:  addiu $t2,$t2,-1
[12]  0x0040002c 0x00092021 addu $4,$0,$9         22          addu  $a0,$zero,$t1
[13]  0x00400030 ???        addu $5,$0,$10        23          addu  $a1,$zero,$t2
[14]  0x00400034 0x0c100013 ???                   24          ???
[15]  0x00400038 0x3c011001 lui $1,0x00001001     25          la    $t0,r
[16]  0x0040003c 0x34280008 ori $8,$1,0x00000008
[17]  0x00400040 0xad020000 sw $2,0x00000000($8)  26          sw    $v0,0($t0)
[18]  0x00400044 0x2402000a addiu $2,$0,0x0000000a 27         li    $v0,10
[19]  0x00400048 0x0000000c syscall               28          syscall
[20]  0x0040004c 0x0085082a slt $1,$4,$5          29  pd:     bge   $a0,$a1,i_p
[21]  0x00400050 0x10200006 beq $1,$0,0x00000006
[22]  0x00400054 0x808b0000 lb $11,0x00000000($4) 30          lb    $t3,($a0)
[23]  0x00400058 0x80ac0000 lb $12,0x00000000($5) 31          lb    $t4,($a1)
[24]  0x0040005c 0x156c0005 ???                   32          ???
[25]  0x00400060 0x24840001 addiu $4,$4,0x00000001 33         addiu $a0,$a0,1
[26]  0x00400064 0x24a5ffff addiu $5,$5,0xffffffff 34         addiu $a1,$a1,-1
[27]  0x00400068 0x08100013 j 0x0040004c          35          j     pd
[28]  0x0040006c 0x24020001 addiu $2,$0,0x00000001 36 i_p:    li    $v0,1
[29]  0x00400070 0x03e00008 jr $31                37          jr    $ra
[30]  0x00400074 0x24020001 addiu $2,$0,0x00000001 38 n_p:    li    $v0,0
[31]  0x00400078 0x03e00008 jr $31                39          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 programas 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 (todas, ou pelo menos as mais relevantes, semanticamente);

   (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!"
3
4          .text
5          .globl main
6    main: la     $t0,t
7    l:    lbu    $t1,0($t0)
8          beq    $t1,$zero,f
9          slti   $t2,$t1,0x5B
10         beq    $t2,$zero,nelm
11         slti   $t2,$t1,0x41
12         bne    $t2,$zero,nelm
13         addiu  $t1,$t1,0x20
14         sb     $t1,0($t0)
15   nelm: addiu  $t0,$t0,1
16         j      l
17   f:    la     $a0,t
18         li     $v0,4
19         syscall
20         li     $v0,10
21         syscall
```

3. (3,0 pontos) Esta questão se refere ao código do programa mostrado na Questão 1 desta prova. Note que todas as linhas do código fonte do programa original estão presentes nesta listagem, gerada pelo montador. Pede-se o seguinte sobre este código:
   a) (1 ponto) O que falta para que este programa esteja completo? Mostre uma possível estrutura da área de dados para o programa;
   b) (1 ponto) Existem pseudo-instruções neste programa? Se sim, diga quantas são ao todo e a quantas instruções cada uma corresponde (pode usar uma tabela para mostrar a resposta, se achar conveniente);
   c) (1 ponto) Existem sub-rotinas neste código? Se sim, quantas são e que linhas ocupam?

Prova P1 **Disciplina: Organização de Computadores**

Professor: Ney Laert Vilar Calazans

Aluno: 14/setembro/2018

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. 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 ???, nas linhas 5, 10, 13, 14 e 24). 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]  0x00400000  0x3c011001  lui $1,0x00001001      15  main:  la    $t1,s  #*1
[2]  0x00400004  0x34290000  ori $9,$1,0x00000000
[3]  0x00400008  0x3c011001  lui $1,0x00001001      16         la    $t2,s  #*2
[4]  0x0040000c  0x342a0000  ori $10,$1,0x00000000
[5]  0x00400010  0x814b0000  lb $11,0x00000000($10) 17  l_l:  lb    $t3,($t2)
[6]  0x00400014  0x11600004  beq $11,$0,0x00000004  18         beqz  $t3,e_ll #*3
[7]  0x00400018  0x3c010000  lui $1,0x00000000     19         addu  $t2,$t2,1 #*4
[8]  0x0040001c  0x34210001  ori $1,$1,0x00000001
[9]  0x00400020  0x01415021  addu $10,$10,$1
[10] 0x00400024  0x0401fffa  bgez $0,0xfffffffa    20         b     l_l #*5
[11] 0x00400028  0x254affff  addiu $10,$10,0xffffff 21  e_ll:  addiu $t2,$t2,-1
[12] 0x0040002c  0x00092021  addu $4,$0,$9         22         addu  $a0,$zero,$t1
[13] 0x00400030  0x000a2821  addu $5,$0,$10        23         addu  $a1,$zero,$t2
[14] 0x00400034  0x0c100013  jal 0x0040004c        24         jal   pd
[15] 0x00400038  0x3c011001  lui $1,0x00001001      25         la    $t0,r  #*6
[16] 0x0040003c  0x34280008  ori $8,$1,0x00000008
[17] 0x00400040  0xad020000  sw $2,0x00000000($8)  26         sw    $v0,0($t0)
[18] 0x00400044  0x2402000a  addiu $2,$0,0x0000000a 27        li    $v0,10 #*7
[19] 0x00400048  0x0000000c  syscall               28         syscall
[20] 0x0040004c  0x0085082a  slt $1,$4,$5          29  pd:    bge   $a0,$a1,i_p #*8
[21] 0x00400050  0x10200006  beq $1,$0,0x00000006
[22] 0x00400054  0x808b0000  lb $11,0x00000000($4) 30         lb    $t3,($a0)
[23] 0x00400058  0x80ac0000  lb $12,0x00000000($5) 31         lb    $t4,($a1)
[24] 0x0040005c  0x156c0005  bne $11,$12,0x00000005 32        bne   $t3,$t4,n_p
[25] 0x00400060  0x24840001  addiu $4,$4,0x00000001 33        addiu $a0,$a0,1
[26] 0x00400064  0x24a5ffff  addiu $5,$5,0xffffffff 34       addiu $a1,$a1,-1
[27] 0x00400068  0x08100013  j 0x0040004c          35         j     pd
[28] 0x0040006c  0x24020001  addiu $2,$0,0x00000001 36  i_p:  li    $v0,1 #*9
[29] 0x00400070  0x03e00008  jr $31                37         jr    $ra
[30] 0x00400074  0x24020001  addiu $2,$0,0x00000001 38  n_p:  li    $v0,0 #*10
[31] 0x00400078  0x03e00008  jr $31                39         jr    $ra
```

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

```
[5]    0x00400010  ???  ??? 17   l_1:   lb    $t3,($t2)
```

O que se quer aqui é gerar os códigos intermediário e objeto da instrução lb $t3,0 ($t2), dado seu código fonte, ou seja, uma operação de montagem de código. Basta extrair do Apêndice A o formato da instrução lb, qual seja:

```
                              lb   rt,offset(rs)  : ling. de montagem
                              0x20 rs rt offset   : cód. objeto
Número de bits/campo:          6   5  5   16
```

A geração do código objeto a partir daqui é imediata. Temos que rs=$t2=$10→01010 e rt=$t3=$11→01011, e o offset é 0=0x0000. Juntando todos os valores dos 4 campos, tem-se 100000 01010 01011 0000000000000000. Agrupando estes 32 bits de 4 em 4 (1000 0001 0100 1011 0000 0000 0000 0000) e convertendo cada grupo de 4 bits em hexa, o código objeto é então 0x814B0000.

Resposta Final:

```
[5]    0x00400010  0x814b0000  lb $11,0x00000000($10) 17   l_1:   lb    $t3,($t2)
```

```
[10]   0x00400024  ???  bgez $0,0xfffffffa     20    b     l_1
```

O que se tem aqui é a pseudo-instrução b, que é traduzida para uma instrução bgez $0 (ou seja, uma instrução que sempre salta, pois $0 é sempre maior ou igual a 0, uma vez que se trata da constante 0). O que se quer é gerar código objeto para esta bgez, cujo código intermediário já contém o deslocamento em hexadecimal. Usa-se novamente o Apêndice A para dele extrair o formato da instrução bgez, qual seja:

```
                              bgez  rs,label
                               1    rs   1    offset
Número de bits/campo:          6    5    5      16
```

O rs é $0=00000, o offset (em 16 bits) vem do código intermediário 0xFFFA=1111 1111 1111 1111 1010. Com isto, monta-se o código objeto assim: 000001 00000 00001 1111111111111010. Agrupando os 32 bits de 4 em 4 (0000 0100 0000 0001 1111 1111 1111 1010) e convertendo cada grupo de 4 bits em hexa, o código objeto fica então 0x0401FFFA.

Resposta final:

```
[10]   0x00400024  0x0401fffa  bgez $0,0xfffffffa     20    b     l_1
```

```
[13]   0x00400030  ???  addu $5,$0,$10         23          addu   $a1,$zero,$t2
```

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

```
                              addu  rd,rs,rt      : ling. de montagem
                               0 rs rt rd 0 0x21  : cód. objeto
Número de bits/campo:          6 5  5  5  5  6    :
```

Em seguida, obtém-se direto do código intermediário da instrução os códigos dos registradores rs=$0=00000, rt=$10=01010 e rd=$5=00101. Com isto, pode-se montar o código objeto da instrução, que é 000000 00000 01010 00101 00000 100001. Agrupando os 32 bits de 4 em 4 (0000 0000 0000 1010 0010 1000 0010 0001) e convertendo cada grupo de 4 bits em hexa, o código objeto fica então 0x000A2821.

Resposta Final:

```
[13]   0x00400030  0x000a2821  addu $5,$0,$10         23          addu   $a1,$zero,$t2
```

```
[14]   0x00400034  0x0c100013  ???                    24          ???
```

O que se quer aqui é partir do código objeto dado e gerar os códigos intermediário e fonte da mesma. O ponto de partida é separar os 6 bits mais significativos do código objeto, o que dá 000011. Isto corresponde em decimal ou hexa ao numeral 3. Com este valor, entra-se na primeira coluna da Figura A.10.2 do Apêndice A e descobre-se que se trata da instrução jal. Obtém-se então o formato da instrução jal do Apêndice A:

```
                              jal   target        : ling. de montagem
                               3    target        : cód. objeto
Número de bits/campo:          6      26          :
```

Para gerar o código objeto basta usar os 26 bits menos significativos do código objeto e a partir dele montar o endereço de salto, sabendo que se trata de uma instrução cujo único operando usa o modo de endereçamento pseudo-absoluto. Para tanto, aos 26 bits dados (que convertidos para binário fornecem 00 0001 0000 0000 0000 0001 0011) acrescenta-se dois bits em 0 à direita destes (gerando 00 0001 0000 0000 0000 0001 0011 00) e acrescenta-se à esquerda dos 26 bits os 4 bits mais significativos do endereço da instrução que segue o jal (isto é o valor do PC incrementado após a busca da instrução, que aponta para a instrução lui da linha [15], que está localizada a partir do endereço 0x00400038). Isto fornece os 32 bits do endereço para onde a instrução jal salta (0000 0000 0100 0000 0000 0000 0100 1100, ou em hexa 0x0040004C). Este endereço é o operando do código intermediário. Notando-se que este endereço aparece na linha [20] do programa, nota-se que esta linha contém o rótulo pd, que é o operando do código fonte da instrução.

Resposta Final:

```
[14]   0x00400034  0x0c100013  jal 0x0040004c        24          jal   pd
```

```
[24]   0x0040005c  0x156c0005  ??? 32                ???
```

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 (000101, ou 5) e usa-se este na Tabela da Figura A.10.2 do Apêndice A, o que identifica a instrução bne. Com esta descoberta, usa-se novamente o Apêndice A para dele extrair o formato da instrução, qual seja:

```
                          bne   rs,rt,label   : ling. de montagem
                          0x5 rs rt offset    : cód. objeto
Número de bits/campo:       6  5  5    16     :
```

Os campos rs e rt são extraídos respectivamente dos bits 25-21 (01011 ou seja $11, ou seja $t3) e 20-16 (os mesmos 01100 ou seja $12, ou seja $t4). Os últimos valores a serem definidos são o offset (retirado direto dos bits 15-0 como sendo 0x0005) e o rótulo para onde se salta, quando se saltar. Para descobrir este rótulo, toma-se o número inteiro representado em complemento de 2 16 bits pelo hexadecimal 0x0005. Claramente 0x0005 equivale a 5 em decimal. Conta-se a partir da instrução abaixo do bne (a da linha [25]) 5 instruções para baixo, atingindo-se assim a linha [30], onde se encontra o rótulo procurado, n_p. A resposta final é então:

Resposta Final:

```
[24]   0x0040005c  0x156c0005  bne $11,$12,0x00000005 32        bne   $t3,$t4, n_p
```

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 programas 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 (todas, ou pelo menos as mais relevantes, semanticamente);

   (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   $t2,$t1,0x5B          # Se não no fim, testa se char pode ser maiúscula
10         beq    $t2,$zero,nelm        # Se não pode ser maiúscula, vai p/ próximo char
11         slti   $t2,$t1,0x41          # Aqui testa se efetivamente é maiúscula
12         bne    $t2,$zero,nelm        # Não sendo maiúscula, vai p/ próximo char
13         addiu  $t1,$t1,0x20          # Aqui achou maiúscula, transforma em minúscula
14         sb     $t1,0($t0)            # Escreve minúscula sobre a  maiúscula
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 maiúsculas desta para as minúsculas correspondentes no alfabeto.

b) Sim, para cada caracter ASCIIE maiúsculo em t escreve-se no seu lugar o caracter minúsculo correspondente. Como t possui 4 letras maiúsculas, quatro 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) são 0x10010000 (‘a’), 0x10010006 (‘m’), 0x10010009 (‘a’) e 0x1001000B (‘a’).

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. (3,0 pontos) Esta questão se refere ao código do programa mostrado na Questão 1 desta prova. Note que todas as linhas do código fonte do programa original estão presentes nesta listagem, gerada pelo montador. Pede-se o seguinte sobre este código:
   a) (1 ponto) O que falta para que este programa esteja completo? Mostre uma possível estrutura da área de dados para o programa;
   b) (1 ponto) Existem pseudo-instruções neste programa? Se sim, diga quantas são ao todo e a quantas instruções cada uma corresponde (pode usar uma tabela para mostrar a resposta, se achar conveniente);
   c) (1 ponto) Existem sub-rotinas neste código? Se sim, quantas são e que linhas ocupam?

Solução da Questão 3 (3,0 pontos)

a) Dado o enunciado, faltaria apenas determinar a área de dados do programa. Para tanto, pode-se examinar o código fonte dado e verificar rótulos não definidos neste que devem corresponder a definições de dados. A análise revela que existem apenas dois rótulos nesta situação, “s” e “r”. Ainda, analisando como estes rótulos são usados, vemos que os registradores $t1 e $t2 começam com ponteiros para “s”, mas o laço entre as linhas [5] a [10], fazem $t2 avançar ao longo da estrutura apontada por “s” até atingir um byte com valor 0. Isto está a indicar que “s” é de fato uma cadeia de caracteres terminada pelo caracter ASCIIE null (0x00). Quanto a “r” trata-se de um valor que nunca é lido no programa, apenas escrito (na linha [17]), usando sw. Uma análise do valor escrito indica que o valor escrito é ou o número 0 ou o número 1, representado em 32 bits. Assim, uma estrutura da área de dados possível para o programa seria a dada abaixo:

```
1    .data
2    s:      .asciiz      "Qualquer coisa!!"
3    r:      .word        0
```

b) Sim, existem múltiplas pseudo-instruções (ao todo são 10) no código fonte: la (nas linhas [1], [3] e [15]), li (nas linhas [18], [28] e [30]), beqz na linha [6], addu com operando imediato (na linha [7]), a pseudo b (na linha [10]), e a pseudo bge (na linha [20]). No limite, aceita-se se os alunos disserem que instruções de leitura da memória sem o deslocamento explícito (offset, considerado como o valor 0) também são pseudo-instruções, pois estas não seguem as regras precisas da estrutura de instruções de leitura (possuir três operandos, dois registradores e o deslocamento). Isto aparece nas linhas [5], [22] e [23]. Neste caso, a resposta seria 13 pseudo-instruções.

c) Sim existe uma sub-rotina, denominada pd, que ocupa as linhas [20] a [31] do programa. O programa que a chama se encontra entre as linhas [1] a [19] e chama a sub-rotina apenas uma vez a cada execução, na linha [14]. A sub-rotina retorna ao programa que a chamou seja pela linha [29], seja pela linha [31]. Antes de retornar ela escreve ou 0 ou 1 no registrador $v0.

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