Prova P1 **Disciplina: Organização de Computadores**

Professor: Ney Laert Vilar Calazans

Aluno: 19/abril/2013

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 parte de uma listagem gerada pelo ambiente MARS como resultado da montagem de um programa. Pede-se: (a) Substituir as triplas interrogações pelo texto que deveria estar em seu lugar (existem 10 triplas ??? a substituir, mas as das linhas [7] e [21] correspondem ao mesmo código objeto. Resolver o problema para uma das linhas resolve o da outra). Em alguns casos, isto implica gerar código objeto, enquanto em outros implica gerar código intermediário e/ou código fonte e/ou endereços. Caso uma instrução seja de salto, expresse o exato endereço para onde ela salta em hexadecimal, 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 os desenvolvimentos para obter os resultados em folha(s) anexa(s), justificando este desenvolvimento.
   Obrigatório: Mostre os desenvolvimentos para obter os resultados, justificando.

```
[1]   Endereço   Cód.Objeto  Cód.Intermediário          Cód. Fonte
[2]  0x00400000 0x3c011001 lui $1,0x00001001      main:la     $t0,num
[3]  0x00400004 0x34280000 ori $8,$1,0x00000000
[4]  0x00400008     ???    lw $4,0x00000000($8)         lw     $a0,0($t0)
[5]  0x0040000c     ???           ???                   addiu  $sp,$sp,-4
[6]  0x00400010 0xafbf0000 sw $31,0x00000000($29)       sw     $ra,0($sp)
[7]  0x00400014 0x0c100009        ???                        ???
[8]  0x00400018 0x8fbf0000 lw $31,0x00000000($29)       lw     $ra,0($sp)
[9]  0x0040001c 0x27bd0004 addiu $29,$29,0x000000       addiu  $sp,$sp, 4
[10] 0x00400020 0x03e00008 jr $31                       jr     $ra
[11] 0x00400024 0x27bdfff8 addiu $29,$29,0xffffff fact:addiu  $sp,$sp,-8
[12] 0x00400028 0xafbf0000 sw $31,0x00000000($29)       sw     $ra,0($sp)
[13] 0x0040002c     ???    sw $4,0x00000004($29)        sw     $a0,4($sp)
[14] 0x00400030 0x2c880001 sltiu $8,$4,0x00000001       sltiu  $t0,$a0, 1
[15] 0x00400034     ???           ???                   beq    $t0,$zero,rec
[16] 0x00400038 0x24020001 addiu $2,$0,0x00000001       addiu  $v0,$zero,1
[17] 0x0040003c 0x8fbf0000 lw $31,0x00000000($29)       lw     $ra, 0($sp)
[18] 0x00400040 0x27bd0008 addiu $29,$29,0x000000       addiu  $sp,$sp, 8
[19] 0x00400044 0x03e00008 jr $31                       jr     $ra
[20] 0x00400048 0x2484ffff addiu $4,$4,0xffffffff  rec:addiu  $a0 $a0,-1
[21] 0x0040004c 0x0c100009        ???                        ???
[22] 0x00400050 0x8fa40004 lw $4,0x00000004($29)        lw     $a0,4($sp)
[23] 0x00400054 0x8fbf0000 lw $31,0x00000000($29)       lw     $ra,0($sp)
[24] 0x00400058 0x27bd0008 addiu $29,$29,0x000000       addiu  $sp,$sp,8
[25] 0x0040005c 0x00440019 multu $2,$4                  multu  $v0, $a0
[26] 0x00400060 0x00001012 mflo $2                      mflo   $v0
[27] 0x00400064 0x03e00008 jr $31                       jr     $ra
```

2. (3,0 pontos) O fragmento de código do MIPS a seguir processa um vetor de inteiros e calcula dois valores importantes, que ao final do fragmento estão armazenados nos registradores $v0 e $v1. Suponha que o vetor possui 5.000 elementos, cada um ocupando uma palavra de memória e estes são indexados de 0 a 4.999. Suponha ainda que o endereço base do vetor está armazenado inicialmente em $a0 e o tamanho do mesmo (5.000) está inicialmente em $a1. Descreva em uma frase o que este trecho de código faz, mencionando o que retorna em $v0 e $v1.

```
1.         addu   $a1,$a1,$a1
2.         addu   $a1,$a1,$a1
3.         addu   $v0,$zero,$zero
4.         addu   $t0,$zero,$zero
5.  outer:addu    $t4,$a0,$t0
6.         lw     $t4,0($t4)
7.         addu   $t5,$zero,$zero
8.         addu   $t1,$zero,$zero
9.  inner: addu   $t3,$a0,$t1
10.        lw     $t3,0($t3)
11.        bne    $t3,$t4,skip
12.        addiu  $t5,$t5,1
13. skip:  addiu  $t1,$t1,4
14.        bne    $t1,$a1,inner
15.        slt    $t2,$t5,$v0
16.        bne    $t2,$zero,next
17.        addu   $v0,$t5,$zero
18.        addu   $v1,$t4,$zero
19. next:  addiu  $t0,$t0,4
20.        bne    $t0,$a1,outer
```

3. (3,0 pontos) Considerando o modo de endereçamento relativo ao PC no MIPS, considere o trecho de código abaixo:

```
1.  here:         beq    $t1,$t2,there
2.  ... muitas linhas de código
3.  there:        add    $t1,$t1,$t1
```

Pede-se sobre este código:

- (1,0 ponto) Em que condições um montador poderia ter problemas para montar este código corretamente?
- (2,0 pontos) Se "**`muitas linhas de código`**" são muitas mesmo (mais do que quantas linhas? Calcule! 1 linha não dá problema, 2 linhas também não, mas e 1000? E 1 bilhão?), como o montador poderia resolver o problema de gerar código correto para o trecho?

Prova P1 **Disciplina: Organização de Computadores**

Gabarito **Professor: Ney Laert Vilar Calazans**

19/abril/2013

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 parte de uma listagem gerada pelo ambiente MARS como resultado da montagem de um programa. Pede-se: (a) Substituir as triplas interrogações pelo texto que deveria estar em seu lugar (existem 10 triplas ??? a substituir, mas as das linhas [7] e [21] correspondem ao mesmo código objeto. Resolver o problema para uma das linhas resolve o da outra). Em alguns casos, isto implica gerar código objeto, enquanto em outros implica gerar código intermediário e/ou código fonte e/ou endereços. Caso uma instrução seja de salto, expresse o exato endereço para onde ela salta em hexadecimal, 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 os desenvolvimentos para obter os resultados em folha(s) anexa(s), justificando este desenvolvimento.
   Obrigatório: Mostre os desenvolvimentos para obter os resultados, justificando.

```
[1]   Endereço   Cód.Objeto  Cód.Intermediário          Cód. Fonte
[2]  0x00400000 0x3c011001 lui $1,0x00001001      main:la     $t0,num
[3]  0x00400004 0x34280000 ori $8,$1,0x00000000
[4]  0x00400008     ???    lw $4,0x00000000($8)         lw     $a0,0($t0)
[5]  0x0040000c     ???           ???                   addiu  $sp,$sp,-4
[6]  0x00400010 0xafbf0000 sw $31,0x00000000($29)       sw     $ra,0($sp)
[7]  0x00400014 0x0c100009        ???                        ???
[8]  0x00400018 0x8fbf0000 lw $31,0x00000000($29)       lw     $ra,0($sp)
[9]  0x0040001c 0x27bd0004 addiu $29,$29,0x000000       addiu  $sp,$sp, 4
[10] 0x00400020 0x03e00008 jr $31                       jr     $ra
[11] 0x00400024 0x27bdfff8 addiu $29,$29,0xffffff fact:addiu  $sp,$sp,-8
[12] 0x00400028 0xafbf0000 sw $31,0x00000000($29)       sw     $ra,0($sp)
[13] 0x0040002c     ???    sw $4,0x00000004($29)        sw     $a0,4($sp)
[14] 0x00400030 0x2c880001 sltiu $8,$4,0x00000001       sltiu  $t0,$a0, 1
[15] 0x00400034     ???           ???                   beq    $t0,$zero,rec
[16] 0x00400038 0x24020001 addiu $2,$0,0x00000001       addiu  $v0,$zero,1
[17] 0x0040003c 0x8fbf0000 lw $31,0x00000000($29)       lw     $ra, 0($sp)
[18] 0x00400040 0x27bd0008 addiu $29,$29,0x000000       addiu  $sp,$sp, 8
[19] 0x00400044 0x03e00008 jr $31                       jr     $ra
[20] 0x00400048 0x2484ffff addiu $4,$4,0xffffffff  rec:addiu  $a0 $a0,-1
[21] 0x0040004c 0x0c100009        ???                        ???
[22] 0x00400050 0x8fa40004 lw $4,0x00000004($29)        lw     $a0,4($sp)
[23] 0x00400054 0x8fbf0000 lw $31,0x00000000($29)       lw     $ra,0($sp)
[24] 0x00400058 0x27bd0008 addiu $29,$29,0x000000       addiu  $sp,$sp,8
[25] 0x0040005c 0x00440019 multu $2,$4                  multu  $v0, $a0
[26] 0x00400060 0x00001012 mflo $2                      mflo   $v0
[27] 0x00400064 0x03e00008 jr $31                       jr     $ra
```

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

```
[4] 0x00400008     ???    lw $4,0x00000000($8)         lw     $a0,0($t0)
```

Obviamente, a questão consiste em demonstrar conhecimento de como se monta o código objeto da instrução lw. Um aspecto que facilita a montagem é a existência de um código intermediário, onde valores de operandos são já expressos de forma numérica. Para produzir o código objeto, inicia-se consultando o Apêndice A, para obter o formato de instrução, o que fornece:

```
                    lw rd, address
                    0x23 rs rt offset
      Número de bits/campo: 6    5  5   16
```

A relação entre address e os campos do código objeto consiste em que address é calculado como a soma dos conteúdos de rs e do offset, este considerado como um número representado em complemento de 2. A partir daí e da linha do código fonte, já se pode obter diretamente os 32 bits do código objeto, seguindo o formato. O que se obtém é 100011 (0x23 em 6 bits), 01000 ($t0=$8, logo 8 em binário 5 bits, Rs), 00100 ($a0=$4, logo 4 em binário 5 bits, Rt), e 0000 0000 0000 0000 (0 em binário, 16 bits). Juntando os 4 campos e agrupando os bits em grupos de 4, obtém-se 1000 1101 0000 0100 0000 0000 0000 0000, ou em hexadecimal: **0x8D040000**, que é o código objeto da instrução.

Resposta final:

```
[4] 0x00400008 0x8D040000  lw $4,0x0000($8)              lw     $a0,0($t0)

 [5] 0x0040000c      ???            ???                  addiu  $sp,$sp,-4
```

Neste item, o objetivo é gerar o código objeto é intermediário para uma instância da instrução addiu. Parte-se neste caso mais uma vez do formato da instrução contido no Apêndice A do livro-texto, que é:

```
                    addiu  rt, rs, imm
                    0x9  rs rt  imm
      Número de bits/campo:  6   5  5   16
```

Note-se que imm é um valor que deve ser representado em complemento de 2 com 16 bits. Como o operando é o negativo -4, o vetor de 16 bits correspondente é 1111 1111 1111 1100. Note-se ainda a inversão de posição dos operandos registrador entre o código fonte e o código objeto. Com estas informações e cuidados, pode-se então montar os códigos intermediário e objeto. O objeto será 001001 (9 em 6 bits) concatenado com 11101 duas vezes(29 decimal representado em binário 6 bits, o valor numérico do registrador $sp, Rs e Rt) concatenado com 1111 1111 1111 1100 (-4 representado em complemento de 2, 16 bits). O código objeto de 32 bits que resulta é 0010 0111 1011 1101 1111 1111 1111 1100, ou em hexadecimal: **0x27BDFFFC**, que é o código objeto da instrução em pauta. O código intermediário é obtido pela substituição dos valores simbólicos por numéricos nos nomes dos registradores e pela troca do operando -4 pelo seu equivalente em complemento de 2 16 bits representado em hexadecimal, como na resposta final abaixo.

Resposta final:

```
[5]    0x0040000c  0x27BDFFFC       addiu $29,$29,0xFFFC          addiu  $sp,$sp,-4

 [7(e 21)] 0x00400014  0x0c100009         ???                          ???
```

Trata-se de desmontagem de uma instrução. Expressando os 32 bits do código objeto em binário e separando-se os seis primeiros, tem-se 000011 00000100000000000000001001. Como os 6 primeiros bits respresentam o número 3 (em decimal ou hexadecimal), isto corresponde a uma instrução JAL, cujo formato é (acha-se este consultando-se a tabela A.10.2 do Apêndice A e o formato do JAL na página A-47 do mesmo documento):

```
                    jal   target
                    0x3   target
      Número de bits/campo:    6      26
```

Para montar os códigos intermediários e fonte, toma-se os 26 bits do campo target, acrescenta-se dois bits 00 a sua direita (para multiplicar o target por 4, e os quatro bits mais significativos do endereço da linha seguinte ao JAL (linha [8] ou [22], 0x**0**0400018 ou 0x**0**0400050) são acrescentados a sua esquerda. O resultado é 0000 0000 0100 0000 0000 0000 0010 0100, que corresponde em hexadecimal ao endereço 0x00400024. Olhando-se o código fornecido, nota-se este endereço corresponde à linha do programa que contém o rótulo **fact**, para onde se está saltando. Logo estão assim definidos os valores simbólico e numérico do rótulo e se pode dar a resposta final como segue abaixo.

Resposta Final:

```
[7 (e 21)] 0x00400014  0x0c100009       jal 0x00400024    22    jal    fact
```

```
[13] 0x0040002c      ???     sw $4,0x00000004($29)        sw     $a0,4($sp)
```

Neste item, o objetivo é a montagem do código objeto para a instrução sw. Para obter o formato do código objeto, consulta-se o Apêndice A, onde se obtém para sw:

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

A partir deste formato, e do código intermediário monta-se facilmente o código objeto. A relação entre address e os campos do código objeto consiste como no primeiro item desta questão (para instrução lw) em que address é calculado como a soma dos conteúdos de rs e do offset, este considerado como um número representado em complemento de 2. Em binário tem-se: 101011 (0x2b representado em binário, 6 bits) 11101 (29 decimal representado em binário 5 bits, o número que corresponde ao registrador $sp, o Rs) 00100 (4 decimal representado em binário 5 bits, o número que corresponde ao registrador $a0, o Rt) e 0000 0000 0000 0100 (o deslocamento, 4 representado em binário, 16 bits). Agrupando em 32 bits (na ordem prevista no formato) e reagrupando de 4 em 4 bits, obtém-se 1010 1111 1010 0100 0000 0000 0000 0100, que em hex decimal corresponde a 0xAFA40004.

Resposta Final:

```
[13] 0x0040002c      0xAFA40004  sw $4,4($29)          sw      $a0,4($sp)
```

```
[15] 0x00400034      ???           ???                  beq    $t0,$zero,rec
```

O que se deseja aqui é gerar os códigos intermediário objeto da instrução beq, dado apenas o código fonte desta e o contexto (formado pelas demais instruções do trecho de programa). O formato da instrução beq é:

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

A partir deste formato os três primeiros campos são obtidos de maneira muito simples: 000100 (4 expresso em 6 bits, o código da operação beq), 01000 (8 expresso em 5 bits, o código do registrador Rs, $t0) e 00000 (0 expresso em 5 bits, o código do registrador Rt, $zero). O problema maior consiste em definir o valor do offset a partir do rótulo fornecido na instrução. Segundo o Apêndice A, o offset corresponde ao número de instruções a saltar para chegar ao lugar onde se encontra o rótulo. Como o rótulo **rec** se encontra cinco linhas abaixo da linha do beq, e considerando que o PC está apontando para a linha abaixo do beq no momento da execução deste, o offset corresponde ao valor 4. Em binário tem-se como código objeto então: 0001 0001 0000 0000 0000 0000 0000 0100. Traduzindo-se para hexadecimal tem-se o código objeto, que é 0x11000004. O código intermediário consiste em representar registradores com seu código numérico e o offset em hexadecimal, conforme mostra a resposta final abaixo.

Resposta Final:

```
[15] 0x00400034      0x11000004  beq $8,$0,0x0004          beq   $t0,$zero,rec
```

Fim da Solução da Questão 1 (4,0 pontos)

2. (3,0 pontos) O fragmento de código do MIPS a seguir processa um vetor de inteiros e calcula dois valores importantes, que ao final do fragmento estão armazenados nos registradores $v0 e $v1. Suponha que o vetor possui 5.000 elementos, cada um ocupando uma palavra de memória e estes são indexados de 0 a 4.999. Suponha ainda que o endereço base do vetor está armazenado inicialmente em $a0 e o tamanho do mesmo (5.000) está inicialmente em $a1. Descreva em uma frase o que este trecho de código faz, mencionando o que retorna em $v0 e $v1.

```
 1.          addu   $a1,$a1,$a1          # $a1 <= tam*2
2.           addu   $a1,$a1,$a1          # $a1 <= tam*4
3.           addu   $v0,$zero,$zero      # $v0 <= 0
4.           addu   $t0,$zero,$zero      # $t0 <= 0
5.   outer:addu   $t4,$a0,$t0           # $t4 <= endereço do próximo elemento do vetor
6.           lw     $t4,0($t4)           # $t4 <= próximo elemento do vetor
7.           addu   $t5,$zero,$zero      # $t5 <= 0
8.           addu   $t1,$zero,$zero      # $t1 <= 0
9.   inner: addu   $t3,$a0,$t1           # $t3 <= endereço do próximo elemento do vetor
10.          lw     $t3,0($t3)           # $t3 <= próximo elemento do vetor
11.          bne    $t3,$t4,skip         # Se elemento em $t4 dif de elemento em $t3, não inc
12.          addiu  $t5,$t5,1            # Se el em $t4 igual a el em $t3, inc $t5
```

```
13. skip:  addiu  $t1,$t1,4          # Avança $t1
14.        bne    $t1,$a1,inner      # Se $t1 ainda menor que 5.000*4, executa laço inner
15.        slt    $t2,$t5,$v0        # Se núm vezes que elemento atual foi achado no
                                     # no vetor ($t5) é menor que núm maximo atua ($v0)
                                     # $t2 recebe 1, senão, 0
16.        bne    $t2,$zero,next     # Se não achou novo máximo pula para próximo
17.        addu   $v0,$t5,$zero      # Se achou, este é novo máximo, copia valor para $v0
18.        addu   $v1,$t4,$zero      # Elemento que aparece mais vezes é guardado em $v1
19. next:  addiu  $t0,$t0,4          # Avança ponteiro para de laço externo para próximo
20.        bne    $t0,$a1,outer      # Se ainda não passou além do último volta a executar
                                     # pesquisa de número de vezes para o novo elemento.
```

Solução da Questão 2 (3,0 pontos)

O trecho em questão calcula o elemento do vetor de 5.000 elementos que aparece o maior número de vezes repetido no vetor. Em $v0 é guardado o número de vezes que tal elemento aparece no vetor, enquanto que em $v1 armazena-se o elemento em si. Os comentários acrescentados no trecho detalham sua operação.

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

3. (3,0 pontos) Considerando o modo de endereçamento relativo ao PC no MIPS, considere o trecho de código abaixo:

```
1. here:        beq    $t1,$t2,there
2. ... muitas linhas de código
3. there:       add    $t1,$t1,$t1
```

Pede-se sobre este código:

- (1,0 ponto) Em que condições um montador poderia ter problemas para montar este código corretamente?
- (2,0 pontos) Se "**`muitas linhas de código`**" são muitas mesmo (mais do que quantas linhas? Calcule! 1 linha não dá problema, 2 linhas também não, mas e 1000? E 1 bilhão?), como o montador poderia resolver o problema de gerar código correto para o trecho?

Solução da Questão 3 (3,0 pontos)

A instrução beq ocupa 32 bits, e trabalha com modo de endereçamento relativo ao PC para definir a posição para onde saltar. O formato da instrução beq é:

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

Ou seja, a partir do rótulo (**`label`**) determina-se o deslocamento (**`offset`**) necessário para atingir o rótulo a partir do valor do registrador PC durante a execução do beq. Este deslocamento é expresso em instruções à frente ou para trás da instrução seguinte ao **`beq`** (pois o PC aponta para esta ao se executar o **`beq`**). Como offset é um campo de 16 bits, e como valores positivos saltam para linhas à frente do **`beq`** (incluindo offset=0), o que é o caso do rótulo **`there`**, nota-se que a distância máxima (número máximo de instruções entre **`here`** e **`there`**) é especificada quando o **`offset`** for o maior número número positivo que se pode representar em 16 bits, ou seja, 0111 1111 1111 1111 em binário, equivalente a 0x7FFF em hexa, e equivalente a 32767 em decimal. Ou seja, se houver mais de 32767 instruções depois do **`beq`** para chegar a there, um único **`beq`** não pode ser usado para traduzir a funcionalidade da linha 1.

Uma forma que o montador teria para resolver o problema seria partir do endereço correspondente ao rótulo **`there`**, colocar este valor no registrador **`$at`** (por convenção reservado para uso do montador), e executar uma instrução **`jr`** para este endereço se a condição de teste for verdadeira. Assim, supondo que there esteja a mais de 32767 instruções de distância de here, o código abaixo poderia ser gerado pelo montador para substituir a linha 1:

```
     la  $at,there  # note a pseudo la, que deve ser transformada
                    # na sequência de instruções lui-ori
     bne  $t1,$t2,np# np (não pula) é rótulo criado pelo montador
     jr   $at       # se beq deve saltar, é aqui que se salta
np:  ....      # lugar onde vai a primeira instrução depois do beq
```

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