Prova P1 **Disciplina: Organização de Computadores**

Professor: Ney Laert Vilar Calazans

Aluno: 16/setembro/2015

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. 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 os desenvolvimentos para obter os resultados, justificando.

```
        Endereço   Cód.Objeto  Cód.Intermediário          Cód. Fonte
[1]     0x00400070  ???         ???                          66 f:  addi   $t0,$zero,1
[2]     0x00400074  0x1088000c  beq $4,$8,0x0000000c          67     beq    $a0,$t0,menig1
[3]     0x00400078  0x0088482a  slt $9,$4,$8                 68     ???
[4]     0x0040007c  ???         ???                          69     bne    $t1,$zero,menig1
[5]     0x00400080  0x2084ffff  addi $4,$4,0xffffffff        70     addi   $a0,$a0,-1
[6]     0x00400084  0x23bdfffc  addi $29,$29,0xfffffffc      71     addi   $sp,$sp,-4
[7]     0x00400088  ???         sw $31,0x00000000($29)       72     sw     $ra,0($sp)
[8]     0x0040008c  0x0c10001c  jal 0x00400070               73     jal    f
[9]     0x00400090  0x8fbf0000  lw $31,0x00000000($29)       74     lw     $ra,0($sp)
[10]    0x00400094  0x23bd0004  addi $29,$29,0x0000000       75     addi   $sp,$sp,4
[11]    0x00400098  0x20840001  addi $4,$4,0x00000001        76     addi   $a0,$a0,1
[12]    0x0040009c  0x00440018  mult $2,$4                   78     mult   $v0,$a0
[13]    0x004000a0  0x00001012  mflo $2                      79     mflo   $v0
[14]    0x004000a4  0x03e00008  jr $31                       80     jr     $ra
[15]    0x004000a8  0x20020001  addi $2,$0,0x00000001        82 menig1:addi $v0,$zero,1
```

2. (4,0 pontos) O programa em linguagem de montagem do MIPS abaixo faz um certo processamento bem específico. (a) Descreva em uma frase o que este trecho de código faz. (b) Comente o programa semanticamente. (c) Diga o que acontece com a área de dados do programa após a execução do mesmo.

```
1          .text
2          .globl  main
3  main:   la     $t0,v
4          la     $t1,s
5          lw     $t1,0($t1)
6          la     $t2,c
7          lw     $t2,0($t2)
8  l:      blez   $t1,f
9          lw     $t3,0($t0)
10         addu   $t3,$t3,$t3
11         addu   $t3,$t3,$t2
12         sw     $t3,0($t0)
13         addiu  $t0,$t0,4
14         addiu  $t1,$t1,-1
15         j      l
16  f:     li     $v0,10
17         syscall
18
19         .data
20  v:     .word   0x35 0xabff 0x80 0x14 0x456 0xa31  0x6b 0x10 0x5 0x16 # o vetor
21  s:     .word   10
22  c:     .word   0x100
```

3. (3,0 pontos) Verdadeiro ou Falso. Abaixo aparecem 10 afirmativas. Marque com V as afirmativas verdadeiras e com F as falsas. Se não souber a resposta correta, deixe em branco, pois cada resposta correta vale 0,3 pontos, mas cada resposta incorreta desconta 0,2 pontos do total positivo de pontos. Não é possível que a questão produza uma nota menor do que 0 pontos.

a) ( ) Se um processador possui um PC e um barramento de endereços ambos de 24 bits e usa endereçamento de memória a byte, ele possui um mapa de memória cujo tamanho é de 16 Megabytes.
b) ( ) O modo de endereçamento pseudo-absoluto, conforme usado no MIPS, obtém o operando de 32 bits (a somar com o registrador PC) acrescentando 2 bits em 0 à direita dos bits 25 a 0 da instrução que o emprega, e agregando à esquerda 4 vezes o bit 25 da mesma instrução (realizando a extensão de sinal do operando).
c) ( ) Suponha que se executa a instrução **ori $t0,$t0,0xFFFF**. Assuma que antes de executar esta instrução, $t0 contém 0xAAADF003. Após executar a instrução, $t0 conterá 0xAAADF002.
d) ( ) As instruções do MIPS **slt** e **sltu** diferenciam-se por realizar comparações com e sem sinal, respectivamente, e ambas usam extensão de sinal para gerar o dado imediato da comparação.
e) ( ) O modo de endereçamento imediato no MIPS é usado em instruções lógicas, aritméticas e de deslocamento de bits.
f) ( ) O código objeto **0x17008003** corresponde a uma instrução **bne** que quando saltar, o faz necessariamente para uma linha do programa anterior à linha onde há o **bne**.
g) ( ) A sequência de instruções **addi $t1,$zero,0xFFFE** seguida da instrução **and $t0,$t0, $t1** escrevem em $t0 o número natural par igual ou o mais próximo possível do valor que havia anteriormente em $t0.
h) ( ) Se o código objeto **0x08100007** estiver armazenado em memória iniciando no endereço **0x00400058**, ele corresponde a um salto incondicional para uma instrução que inicia necessariamente em uma posição de memória anterior à posição onde se encontra a instrução de salto.
i) ( ) A memória do MIPS é endereçada a byte, e usando uma única instrução do processador é possível escrever nela apenas dados de 8, 16 ou 32 bits.
j) ( ) Um vetor de 10.000 caracteres representados de forma convencional (ou seja, cada caracter usando o mínimo espaço necessário para armazená-lo) no MIPS ocupa 40.004 bytes.

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. 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 os desenvolvimentos para obter os resultados, justificando.

```
     Endereço    Cód.Objeto  Cód.Intermediário          Cód. Fonte
[1]  0x00400070  ???         ???                           66 f:  addi   $t0,$zero,1
[2]  0x00400074  0x1088000c  beq $4,$8,0x0000000c         67     beq    $a0,$t0,menig1
[3]  0x00400078  0x0088482a  slt $9,$4,$8                 68     ???
[4]  0x0040007c  ???         ???                           69     bne    $t1,$zero,menig1
[5]  0x00400080  0x2084ffff  addi $4,$4,0xffffffff         70     addi   $a0,$a0,-1
[6]  0x00400084  0x23bdfffc  addi $29,$29,0xfffffffc       71     addi   $sp,$sp,-4
[7]  0x00400088  ???         sw $31,0x00000000($29)        72     sw     $ra,0($sp)
[8]  0x0040008c  0x0c10001c  jal 0x00400070                73     jal    f
[9]  0x00400090  0x8fbf0000  lw $31,0x00000000($29)       74     lw     $ra,0($sp)
[10] 0x00400094  0x23bd0004  addi $29,$29,0x0000000       75     addi   $sp,$sp,4
[11] 0x00400098  0x20840001  addi $4,$4,0x00000001        76     addi   $a0,$a0,1
[12] 0x0040009c  0x00440018  mult $2,$4                   78     mult   $v0,$a0
[13] 0x004000a0  0x00001012  mflo $2                      79     mflo   $v0
[14] 0x004000a4  0x03e00008  jr $31                       80     jr     $ra
[15] 0x004000a8  0x20020001  addi $2,$0,0x00000001        82 menig1:addi $v0,$zero,1
```

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

```
[1]  0x00400070  ???         ???                           66 f:  addi   $t0,$zero,1
```

A única informação fornecida é o código fonte da instrução. A partir dele sabemos se tratar de instrução **addi**, cujo formato, retirado do Apêndice A é:

```
                              addi rt,rs,imm : linguagem de montagem
                               8   rs rt imm : formato do cód. objeto
      Número de bits/campo:   6    5  5  16 :
```

O código objeto é muito fácil de gerar: 001000 (8 em seis bits) concatenado com o endereço do **rs** no banco, 00000, concatenado com o endereço do **rt** no banco, 01000 ($t0 é $8, ver Tabela no

início desta prova), concatenado com o terceiro operando em 16 bits, ou seja 0000 0000 0000 0001. Juntando os 32 bits e agrupando-os de 4 em 4 temos: 0010 0000 0000 1000 0000 0000 0000 0001. Traduzindo cada grupo de 4 bits em um valor hexadecimal, obtém-se 0x20080001, que é o código objeto da instrução. Para gerar o código intermediário, é mais simples ainda. Usa-se o mesmo mnemônico e traduz-se os nomes dos registradores para suas versões numéricas, e produz-se a representação do dado imediato em hexadecimal, o que fornece addi $8,$0, 0x00000001

Resposta final:

```
[1] 0x00400070 0x20080001  addi $8,$0,0x00000001          addi $t0,$zero,1

[3]  0x00400078  0x0088482a  slt $9,$4,$8                  68     ???
```

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

```
                                slt   rd,rs,rt
                                0x0   rs  rt  rd  0  0x2a
      Número de bits/campo:      6    5   5   5   5   6
```

A geração do código fonte é trivial, basta traduzir o nome dos registradores, mantendo a mesma ordem do código intermediário. O resultado é **slt $t1, Sa0, $t0**.

Resposta Final:

```
[3] 0x00400078  0x0088482a    slt $9,$4, $8               slt    $t1,$a0,$t0

[4]  0x0040007c  ???          ???                         69     bne    $t1,$zero,menig1
```

O que se deseja aqui é gerar os códigos objeto e intermediário de uma instrução, dado apenas o código fonte. Do Apêndice A descobre-se o formato da instrução **bne**, que é:

```
                                bne    rs,rt,rótulo
                                 5     rs   rt offset
      Número de bits/campo:      6     5    5    16
```

Para gerar o código intermediário, precisamos converter os nomes simbólicos dos registradores em nomes numéricos correspondentes, ou seja $t1=$9 e $zero=$0. O rótulo é transformado em offset considerando quantas instruções existem entre a instrução imediatamente subsequente ao **bne** (**addi** na linha 70) e a instrução associada ao rótulo da **bne** (menig1). Como são 10 linhas e o rótulo está abaixo da bne, o campo de offset deve conter a constante 10 em decimal ou 0x0000000a em hexa. Assim o código intermediário é **bne $9,$0,0x0000000a**. O código objeto pode ser produzido a partir dos campos do formato e do valor do offset calculado para o código intermediário. Assim temos 000101 (5 em binário 6 bits), 01001 (9 em binário 5 bits), 00000 (0 em binário 5 bits) e 0000000000001010 (10 em 16 bits). Juntando os campos e agrupando de 4 em 4 bits, tem-se: 0001 0101 0010 0000 0000 0000 0000 1010 ou 0x1520000A, que é o código objeto.

Resposta Final:

```
[4] 0x0040007c  0x1520000abne    $9,$0,0x0000000a    bne    $t1,$zero,menig1

[7] 0x00400088  ???      sw $31,0x00000000($29)       72     sw     $ra,0($sp)
```

O que se deseja aqui é apenas gerar o código objeto de uma instrução sw, dados o código fonte e intermediário desta. Do Apêndice A descobre-se o formato da instrução **sw**, que é:

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

Gerar o código objeto é possível usando os valores numéricos dos registradores e o offset já dados no código intermediário e montando tudo a partir do formato. Logo, o código objeto é a junção de 4 campos: 101011 (constante 0x2b em 6 bits) 11101 (29 em 5 bits, valor do campo **rs**), 11111 (31 em 5 bits, valor do campo **rt**) e 0000000000000000 (0 em 16 bits, o **offset**). Logo, o código objeto é 1010 1111 1011 1111 0000 0000 0000 0000 ou em hexa 0xafbf0000.

Resposta Final:

```
[7] 0x00400088  0xafbf0000       sw $31,0x00000000($29)          72     sw     $ra,0($sp)
```

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. (a) Descreva em uma frase o que este trecho de código faz. (b) Comente o programa semanticamente. (c) Diga o que acontece com a área de dados do programa após a execução do mesmo.

```
1            .text
```

```
2          .globl  main          #
3  main:   la      $t0,v         # $t0 contém pomteiro para próximo elemento de v
4          la      $t1,s         #
5          lw      $t1,0($t1)    # $t1 contém no. de elementos de v que resta processar
6          la      $t2,c         #
7          lw      $t2,0($t2)    # $t2 contém a constate c
8  l:      blez    $t1,f         # Se nada mais resta processar, termine programa
9          lw      $t3,0($t0)    # senão toma próximo elemento v(i) de v e coloca em $t3
10         addu    $t3,$t3,$t3   # v(i) ← 2*v(i)
11         addu    $t3,$t3,$t2   # v(i) ← 2*v(i)+c
12         sw      $t3,0($t0)    # escreve novo valor calculado de v(i) de volta em v
13         addiu   $t0,$t0,4     # avança ponteiro para próximo elemento de v
14         addiu   $t1,$t1,-1    # reduz número de elementos que falta tratar
15         j       l             # volta para testar fim do laço
16  f:     li      $v0,10        # quando processamemto acabou , apenas termina o
17         syscall               # o programa
18
19         .data
20  v:     .word   0x35 0xabff 0x80 0x14 0x456 0xa31  0x6b 0x10 0x5 0x16 # o vetor
21  s:     .word   10            # o tamanho do vetor
22  c:     .word   0x100         # a constante
```

Solução da Questão 2 (4,0 pontos)

(a) Este programa toma os elementos do vetor numérico **v** (com 10 elementos no exemplo de área de dados, tamanho informado no elemento **s** da área de dados) e transforma cada elemento **v(i)** da seguinte maneira: **v(i) ← 2*v(i) + c**, onde **c** é a constante na área de dados do programa.

(b) Ver comentários no programa acima.

(c) Na área de dados o vetor v é alterado para conter os novos valores mencionados em (a) acima. Por exemplo o primeiro elemento de v se torna 0x170, o segundo se torna 0x158FE e assim por diante.

.

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

3. (3,0 pontos) Verdadeiro ou Falso. Abaixo aparecem 10 afirmativas. Marque com V as afirmativas verdadeiras e com F as falsas. Se não souber a resposta correta, deixe em branco, pois cada resposta correta vale 0,3 pontos, mas cada resposta incorreta desconta 0,2 pontos do total positivo de pontos. Não é possível que a questão produza uma nota menor do que 0 pontos.

Solução da Questão 3 (3,0 pontos)

a) (**V**) Se um processador possui um PC e um barramento de endereços ambos de 24 bits e usa endereçamento de memória a byte, ele possui um mapa de memória cujo tamanho é de 16 Megabytes.
Como sabe-se 2**(número de bits do PC) dá a extensão do mapa em posições. Ora, 2**16 = 16 Mega. Como o endereçamento é a byte, cada endereço armazena exatamente um byte e o mapa de memória tem o tamanho citado na afrmativa. **V**

b) (**F**) O modo de endereçamento pseudo-absoluto, conforme usado no MIPS, obtém o operando de 32 bits (a somar com o registrador PC) acrescentando 2 bits em 0 à direita dos bits 25 a 0 da instrução que o emprega, e agregando à esquerda 4 vezes o bit 25 da mesma instrução (realizando a extensão de sinal do operando).
Bem errado, pois o modo citado (pseudo-absoluto) do MIPS monta o endereço acrescentando 2 bits em 0 à direita dos 26 bits do operando, e usando os 4 bits mais significativos do valor do PC no momento da execução da instrução como os 4 bits mais significativos do endereço para onde saltar. **F**

c) (**F**) Suponha que se executa a instrução **ori $t0,$t0,0xFFFF**. Assuma que antes de executar esta instrução, $t0 contém 0xAAADF003. Após executar a instrução, $t0 conterá 0xAAADF002.
Instruções lógicas com dado imediato no MIPS usam extensão de 0 e não de sinal. Logo. O dado a ser “oreado” com 0xAAADF003 é 0x0000FFFF, o que tem como efeito manter os 16 bits superiores de $t0 inalterados e mudar todos os 16 bits inferiores para 1, gerando em $t0 0xAAADFFFF como novo valor, e não 0xAAADF002. **F**

d) (**F**) As instruções do MIPS **slt** e **sltu** diferenciam-se por realizar comparações com e sem sinal, respectivamente, e ambas usam extensão de sinal para gerar o dado imediato da comparação.

Notem que as instruções em questão não possuem a letra i no mnemônico. De fato, estas são instruções tipo R que não usam nenhum dado imediato. Logo, a afirmativa não faz sentido As comparações em ambos os casos são entre valores de 32 bits armazenados em registradores. **F**

e) (**F-V**) O modo de endereçamento imediato no MIPS é usado em instruções lógicas, aritméticas e de deslocamento de bits.
Ok, ao se considerar apenas o formato I de instruções, as instruções de deslocamento não usam tal campo, pois são tipo **R**, onde um campo especial de 5 bits contém o número de bits a deslocar. Neste caso, **F** é a resposta esperada. No entanto o dado imediato no fundo existe, na forma destes 5 bits e a afirmativa se justifica. Assim, aceitarei também **V**

f) (**V**) O código objeto **0x17008003** corresponde a uma instrução **bne** que quando saltar, o faz necessariamente para uma linha do programa anterior à linha onde há o **bne**.
Faz sentido, pois o deslocamento (offset) da instrução, 0x8003 é um número negativo em complemento de 2 16 bits. Assim o salto, quando acontecer, será para cima (ou para trás, linha anterior, do programa) **V**

g) (**V**) A sequência de instruções **addi $t1,$zero,0xFFFE** seguida da instrução **and $t0,$t0, $t1** escrevem em $t0 o número natural par igual ou o mais próximo possível do valor que havia anteriormente em $t0.
Instruções aritméticas com dado imediato usam sempre extensão de sinal para produzir a versão de 32 bits do dado imediato para uso na soma. Logo, **addi** acima escreve em $t1 o valor 0xFFFFFFFE, que é um padrão com os 31 bits mais significativos em 1 e o bit 0 (mais à direita) em 0. Ao fazer o **AND** deste padrão com o que quer que haja em $t0 e escrevendo o resultado em $t0, teremos como resultado a manutenção de todos os bits de $t0, exceto o último que será forçado para o valor 0, não importando seu valor original. Assim, isto corresponde a transformar o númeral em $t0 no numeral par mais próximo dele. **V**

h) (**F**) Se o código objeto **0x08100007** estiver armazenado em memória iniciando no endereço **0x00400058**, ele corresponde a um salto incondicional para uma instrução que inicia necessariamente em uma posição de memória anterior à posição onde se encontra a instrução de salto.
Analisando o código objeto fornecido, notamos tratar-se de uma instrução j (6 primeiros bits são 000010, ou 2). Os 26 bits restantes são 00 0001 0000 0000 0000 0000 0111. Acrescentando dois 0s à direita e os quatro bits mais significativos do endereço seguinte ao dado na questão (0x00400058, ou seja 0x0040005C), temos o endereço de salto, ou seja, 0000 0000 0100 0000 0000 0000 0001 1100, ou 0x0040001C. Ora, este endereço está depois de (posição posterior a) 0x00400058, não antes. **F**

i) (**V**) A memória do MIPS é endereçada a byte, e usando uma única instrução do processador é possível escrever nela apenas dados de 8, 16 ou 32 bits.
Sim, só existem três instruções de escrita na memória, sw, sh e sb que escrevem respectivamente 4, 2 e 1 byte, respectivamente. **V**

j) (**F**) Um vetor de 10.000 caracteres representados de forma convencional (ou seja, cada caracter usando o mínimo espaço necessário para armazená-lo) no MIPS ocupa 40.004 bytes.
Um vetor de 10.000 caracteres ASCIIE ocupam, cada um 1 byte de memória. Ao final deve existir 1 byte para indicar fim da cadeia (o caracter NULL). Assim para um vetor de 10.000 caracteres devemos usar 10.001 bytes, não 40.004. **F**

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