Prova P1 **Disciplina: Organização de Computadores**

Professor: Ney Laert Vilar Calazans

Aluno: 13/setembro/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 (dois trechos separados). Pede-se: (a) Substituir as triplas interrogações pelo texto que deveria estar em seu lugar (existem 10 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]    0x00400000  0x3c011001  lui $1,0x00001001      main: la    $t0,num
[3]    0x00400060  0x11000020    ???                        ???
[4]    0x00400064   ???          ???                       beq   $v0,$zero,cont0
[5]    0x00400068   ???          ???                       addi  $sp,$sp,-16
[6]    0x0040006c  0xafa2000c  sw $2,0x0000000c($29)       sw    $v0,12($sp)
[7]    0x00400070  0x3c011001  lui $1,0x00001001           la    $t0,sequences
...
[8]    0x00400098  0x01094004  sllv $8,$9,$8               sllv  $t0,$t1,$t0
[9]    0x0040009c  0x8faa0004  lw $10,0x00000004($29)      lw    $t2,4($sp)
[10]   0x004000a0  0x010a0018  mult $8,$10                 mult  $t0,$t2
[11]   0x004000a4  0x00004012    ???                       ???
[12]   0x004000a8  0x21080001  addi $8,$8,0x00000001       addi  $t0,$t0,1
[13]   0x004000ac  0x3c011001  lui $1,0x00001001           la    $a0,sequences
[14]   0x004000b0  0x3424008c  ori $4,$1,0x0000008c
[15]   0x004000b4  0x01044020  add $8,$8,$4                add   $t0,$t0,$a0
[16]   0x004000b8  0xa1000000  sb $0,0x00000000($8)       sb    $zero,0($t0)
[17]   0x004000bc   ???        lw $31,0x00000000($29)     lw    $ra,0($sp)
[18]   0x004000c0  0x23bd0010  addi $29,$29,0x0000010     addi  $sp,$sp,16
[19]   0x004000c4  0x24020004  addiu $2,$0,0x00000004     li    $v0,4
[20]   0x004000c8  0x0000000c  syscall                    syscall
[21]   0x004000cc  0x08100002  j 0x00400008               j     while
[22]   0x004000d0  0x3c011001  lui $1,0x00001001    cont0: la    $a0,n_j_exec
[23]   0x004000d4  0x3424005f  ori $4,$1,0x0000005f
[24]   0x004000d8  0x24020004  addiu $2,$0,0x00000004     li    $v0,4
[25]   0x004000dc  0x0000000c  syscall                    syscall
[26]   0x004000e0   ???        j 0x00400008               j     while
[27]   0x004000e4  0x3c011001  lui $1,0x00001001  contfim: la    $t0,cont
```

2. (3,0 pontos) O fragmento de código do MIPS a seguir processa uma cadeia de caracteres Descreva em uma frase o que este trecho de código faz, mencionando o que ele armazena em memória ao final de sua execução. Comente o programa semanticamente.

```
1               .text
2               .globl   main
3   main:       li       $a0,'M'
4               xor      $v0,$v0,$v0
5               la       $s0,dados
6   w:          lw       $t1,0($s0)
7               addu     $t3,$zero,$zero
8   f:          bgt      $t3,3,ff
9               andi     $t0,$t1,0xff
10              beq      $t0,$zero,fc
11              bne      $t0,$a0,na
12              addiu    $v0,$v0,1
13  na:         srl      $t1,$t1,8
14              addiu    $t3,$t3,1
15              j        f
16  ff:         addiu    $s0,$s0,4
17              j        w
18  fc:         li       $v0,10
19              syscall
20              .data
21  dados:      .asciiz  "Minha mamãe me AMA!!!"
```

3. (3,0 pontos) Suponha que você possui uma descrição completa do processador MIPS, conforme descrito no Apêndice A do livro texto, e que é necessário estender esta arquitetura acrescentando uma nova instrução. Esta nova instrução chama-se ADDU3, é similar à instrução ADDU mas, ao invés de somar **dois** valores contidos em registradores e colocar o resultado em um terceiro, ela soma **três** valores contidos em registradores e coloca o resultado em um quarto registrador. Deve-se gerar para a nova instrução um formato que não entre em conflito com qualquer instrução existente. Proponha este novo formato e mostre pelo menos um exemplo de linha de programa com esta nova instrução. Gere código objeto para a linha exemplo, em hexadecimal. Você deve seguir os pressupostos da arquitetura. Logo, o código objeto da nova instrução deve ocupar exatamente 32 bits, e cada campo associado a um registrador deve ser de 5 bits.

Prova P1 **Disciplina: Organização de Computadores**

Gabarito **Professor: Ney Laert Vilar Calazans**

13/setembro/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 (dois trechos separados). Pede-se: (a) Substituir as triplas interrogações pelo texto que deveria estar em seu lugar (existem 10 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]    0x00400000  0x3c011001  lui $1,0x00001001      main: la    $t0,num
[3]    0x00400060  0x11000020    ???                        ???
[4]    0x00400064   ???          ???                       beq   $v0,$zero,cont0
[5]    0x00400068   ???          ???                       addi  $sp,$sp,-16
[6]    0x0040006c  0xafa2000c  sw $2,0x0000000c($29)       sw    $v0,12($sp)
[7]    0x00400070  0x3c011001  lui $1,0x00001001           la    $t0,sequences
...
[8]    0x00400098  0x01094004  sllv $8,$9,$8               sllv  $t0,$t1,$t0
[9]    0x0040009c  0x8faa0004  lw $10,0x00000004($29)      lw    $t2,4($sp)
[10]   0x004000a0  0x010a0018  mult $8,$10                 mult  $t0,$t2
[11]   0x004000a4  0x00004012    ???                       ???
[12]   0x004000a8  0x21080001  addi $8,$8,0x00000001       addi  $t0,$t0,1
[13]   0x004000ac  0x3c011001  lui $1,0x00001001           la    $a0,sequences
[14]   0x004000b0  0x3424008c  ori $4,$1,0x0000008c
[15]   0x004000b4  0x01044020  add $8,$8,$4                add   $t0,$t0,$a0
[16]   0x004000b8  0xa1000000  sb $0,0x00000000($8)       sb    $zero,0($t0)
[17]   0x004000bc   ???       lw $31,0x00000000($29)      lw    $ra,0($sp)
[18]   0x004000c0  0x23bd0010  addi $29,$29,0x0000010     addi  $sp,$sp,16
[19]   0x004000c4  0x24020004  addiu $2,$0,0x00000004     li    $v0,4
[20]   0x004000c8  0x0000000c  syscall                    syscall
[21]   0x004000cc  0x08100002  j 0x00400008               j     while
[22]   0x004000d0  0x3c011001  lui $1,0x00001001    cont0: la    $a0,n_j_exec
[23]   0x004000d4  0x3424005f  ori $4,$1,0x0000005f
[24]   0x004000d8  0x24020004  addiu $2,$0,0x00000004     li    $v0,4
[25]   0x004000dc  0x0000000c  syscall                    syscall
[26]   0x004000e0   ???        j 0x00400008               j     while
[27]   0x004000e4  0x3c011001  lui $1,0x00001001  contfim: la    $t0,cont
```

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

```
[3]    0x00400060  0x11000020    ???                          ???
```

Obviamente, a questão consiste em demonstrar conhecimento de como se desmonta o código objeto de uma instrução. Para descobrir que instrução é esta, usa-se a Tabela A.10.2 do Apêndice A. Separando-se os 6 bits mais significativos do código objeto (bits 31 a 26) tem-se 000100 (gerados a partir da conversão dos dois primeiros dígitos hexa do código objeto, 0x11, e tomando os seis bits mais à esquerda deste, ou seja 0001 0001 => 000100), o que corresponde ao número decimal 4. Consultando a tabela, nota-se que 4 é o código da instrução BEQ. Para continuar a desmontagem, consulta-se o formato da instrução BEQ, que é:

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

Dos 10 bits seguintes do código objeto se retira os designadores dos registradores rs e rt, que são respectivamente 01000 e 00000. O offset corresponde aos 4 dígitos hexadecimais do código objeto, ou seja 0x0020. Assim, o endereço para onde o BEQ salta (quando salta) pode ser obtido somando-se 0x0020*4=0x0080 ao endereço da linha seguinte ao BEQ, que é 0x00400064. O resultado desta soma dá 0x004000E4, a linha do rótulo contfim. Logo, temos todas as informações para produzir a resposta deste item:

Resposta final:

```
[3] 0x00400060 0x11000020  beq $8,$0,0x0020              beq $t0,$zer0,contfim

 [4] 0x00400064    ???          ???                      beq   $v0,$zero,cont0
```

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

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

Os códigos dos registradores em decimal e binário (5 bits) são, respectivamente, rs=2=00010 e rt=0=00000. A última informação necessária para gerar os códigos intermediários e objeto é a determinação do campo de offset. Partindo-se do rótulo para onde saltar, obtém-se seu endereço do enunciado, cont0=0x004000D0. O endereço da linha seguinte ao BEQ é 0x00400068. Pela definição do modo de endereçamento relativo, o offset é dado pela divisão por 4 da diferença entre estes dois endereços, ou seja offset=(0x004000D0-0x00400068)/4= 0x68/4=0x1A. Assim, tem-se como resposta o código objeto 0001 0000 0100 0000 0000 0000 0001 1010 em binário ou 0x1040001A em hexadecimal.

Resposta final:

```
[4]    0x00400063  0x1040001a       beq $2,$0,0x001a       beq  $v0,$zero,cont0

 [5]   0x00400068   ???          ???                      addi   $sp,$sp,-16
```

Trata-se de montagem de uma instrução addi. Parte-se do formato da instrução, retirado da Tabela A.10.2 do Apêndice A:

```
                        addi rt, rs, immed
                          8  rs rt immed
     Número de bits/campo: 6   5  5   16
```

O código do registrador $sp em decimal e binário 5 bits é $sp=29=11101. Resta apenas determinar o valor do campo immed que é -16 em 16 bits. Sabe-se que +16 em 16 bits é 0000000000010000. Invertendo todos os bits e somando 1 obtém-se 1111 1111 1111 0000, que é -16. Em hexadecimal este valor fica 0xFFF0. O código objeto completo é então 001000 11101 11101 1111 1111 1111 0000, ou em hexadecimal 0x23BDFFF0:

Resposta Final:

```
 [5] 0x00400068  0x23bdfff0      addi $29,$29,0xfff0       addi   $sp,$sp,-16

[11] 0x004000a4  0x00004012      ???                       ???
```

Neste item, o objetivo é a desmontagem do código objeto. Para descobrir que instrução é esta, usa-se a Tabela A.10.2 do Apêndice A. Separando-se os 6 bits mais significativos do código objeto (bits 31 a 26) tem-se 000000 (gerados a partir da conversão dos dois primeiros dígitos hexa do código objeto, 0x00, e tomando os seis bits mais à esquerda deste, ou seja 0000 0000 => 000000), o número 0, que identifica a classe de instruções tipo R. Logo é necessário olhar os seis últimos bits do código objeto para identificar univocamente a instrução que se tem. Separando-se os 6 bits menos significativos do código objeto (bits 5 a 0) tem-se 010010 (gerados a partir da conversão dos dois últimos dígitos hexa do código objeto, 0x12, e tomando os seis bits mais à

esquerda deste, ou seja 0001 0010 => 010010), o que fornece o decimal 18. Consultando a tabela, nota-se que 18 nestes bits para uma instrução tipo R identificam a instrução MFLO. Para continuar a desmontagem, consulta-se o formato da instrução MFLO no mesmo documento, o que fornece:

```
                                mflo   rd
                                  0     0     rd   0   0x12
      Número de bits/campo:       6     10    5    5    6
```

Os bits 15-11 do código objeto definem o valor de rd, que é 01000 (8 em decimal representado em binário 5 bits, o número que corresponde ao registrador $t0). Assim, tem-se:

Resposta Final:

```
[11] 0x004000a4  0x00004012     mflo   $8                  mflo   $t0


[17] 0x004000bc    ???        lw $31,0x00000000($29)       lw     $ra,0($sp)
```

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

```
                              lw      rt offset(rs)
                               0x23   rs rt   offset
      Número de bits/campo:    6      5  5     16
```

Do código intermediário obtém-se os valores em decimal dos registradores, rs=29=11101, rt=31=11111. O offset também está pronto no código intermediário, e é 0x0000. Logo, o código objeto é 100011 11101 11111 0000 0000 0000 0000, ou em hexadecimal 0x8FBF0000.

Resposta Final:

```
[17] 0x004000bc  0x8fbf0000          lw $31,0x00000000($29)       lw     $ra,0($sp)

[26] 0x004000e0     ???        j 0x00400008                 j      while
```

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

```
                                 j    target
                                0x2    target
      Número de bits/campo:      6       26
```

O endereço de salto já está calculado no código intermediário, donde se conclui que while é o rótulo associado ao endereço 0x00400008. Usando a definição do modo de endereçamento pseudo-absoluto que a instrução J usa, neste código de 32 bits os bits 27 a 2 são os 26 bits que correspondem ao campo target, ou seja, 0000 **00000100000000000000000010** 00. Juntando os 26 bits mostrados em negrito com o opcode (número 2 em decimal), obtém-se o código objeto desejado, que é 0000 10**00 0001 0000 0000 0000 0010**, ou 0x08100002.

Resposta Final:

```
[26] 0x004000e0  0x08100002      j 0x00400008                 j      while
```

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

2. (3,0 pontos) O fragmento de código do MIPS a seguir processa uma cadeia de caracteres Descreva em uma frase o que este trecho de código faz, mencionando o que ele armazena em memória ao final de sua execução. Comente o programa semanticamente.

```
1               .text
2               .globl   main
3    main:      li       $a0,'M'       # Laço Principal - carrega em $a0 caracter a procurar
4               xor      $v0,$v0,$v0   # zera $v0, o contador de caracteres M encontrados
5               la       $s0,dados     # gera ponteiro para cadeia de caracteres dados
6    w:         lw       $t1,0($s0)    # busca próximo caracter da cadeia e coloca em $t1
7               addu     $t3,$zero,$zero      # zera contador de bytes em palavra $t3
8    f:         bgt      $t3,3,ff      # se contador de bytes em palavra é 4, acabou de tratar palavra
9               andi     $t0,$t1,0xff  #       senão, isola em $t0 o byte inferior da palavra
10              beq      $t0,$zero,fc  # se byte inferior é caracter fim de cadeia (NULL), fim do programa
11              bne      $t0,$a0,na    #       senão, testa se caracter da cadeia é o procurado (M)
12              addiu    $v0,$v0,1     #       se teste deu verdadeiro, incrementa contador de M´s
13   na:        srl      $t1,$t1,8     # desloca palavra para colocar próximo caracter no byte inferior
14              addiu    $t3,$t3,1     # incrementa contador de bytes tratados na palavra
15              j        f             # e volta para testar o novo caracter
16   ff:        addiu    $s0,$s0,4     # quando aqui, terminou de tratar uma palavra, incrementa ponteiro para
17                                     #       apontar para a próxima palavra da cadeia
18              j        w             # e volta a testar caracteres desta nova palavra
```

```
19  fc:         li        $v0,10          # quando aqui, chegou ao fim do tratamento da cadeia, antes
20              syscall                   #       disso $v0 contém o número de Ms encontrados na cadeia
21              .data
22  dados:      .asciiz   "Minha mamãe me AMA!!!"    # Cadeia a tratar
```

Solução da Questão 2 (3,0 pontos)

O trecho em questão varre uma cadeia de caracteres terminada pelo caracter NULL (0x00 em hexa) procurando pelo caracter ASCIIE M, cujo código é armazenado inicialmente no registrador $a0. Cada vez que este é achado na cadeia, incrementa-se o contador, para o que se usa o registrador $v0, zerado no início da execução do programa. Uma particularidade deste programa é que ele lê os caracteres da cadeia em memória palavra a palavra e não byte a byte. Assim, a cada palavra lida há um laço interno, executado exatamente 4 vezes sobre esta, tratando caracter a caracter da cadeia e a cada volta do laço posicionando o novo caracter a tratar no byte inferior do registrador de trabalho usado (via uso da instrução SRL na linha 13). O programa acumula o número de caracteres no registrador $v0, mas este é destruído ao final da execução, ao preparar a chamada do sistema que sai do programa. O programa não escreve nada na memória externa.

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

3. (3,0 pontos) Suponha que você possui uma descrição completa do processador MIPS, conforme descrito no Apêndice A do livro texto, e que é necessário estender esta arquitetura acrescentando uma nova instrução. Esta nova instrução chama-se ADDU3, é similar à instrução ADDU mas, ao invés de somar **dois** valores contidos em registradores e colocar o resultado em um terceiro, ela soma **três** valores contidos em registradores e coloca o resultado em um quarto registrador. Deve-se gerar para a nova instrução um formato que não entre em conflito com qualquer instrução existente. Proponha este novo formato e mostre pelo menos um exemplo de linha de programa com esta nova instrução. Gere código objeto para a linha exemplo, em hexadecimal. Você deve seguir os pressupostos da arquitetura. Logo, o código objeto da nova instrução deve ocupar exatamente 32 bits, e cada campo associado a um registrador (devem existir 4 destes) deve ser de 5 bits.

Solução da Questão 3 (3,0 pontos)

A instrução ADDU do MIPS é uma instrução tipo R, com o seguinte formato

	addu	rd	rs	rt		
	0	rs	rt	rd	0	0x21
Número de bits/campo:	6	5	5	5	5	6

Ora, note-se que existem quatro campos de 5 bits, sendo 3 deles usados para especificar operandos do tipo registrador (modo de endereçamento direto a registrador) e um com o valor constante 0. A nova instrução requer 4 operandos do tipo registrador. Assim, se usarmos o campo não usado na ADDU para conter o código do quarto operando registrador (nos bits 6-10 do código objeto), resta apenas achar um par de códigos que identifique univocamente a instrução ADDU3, diferenciando-a de qualquer código de uma instrução já existente. A chave para isto é a Tabela A.10.2 do Apêndice A. Notem que esta tabela mostra que para as instruções tipo R (aquelas com bits 31-25 igual a 000000) existem vários códigos de função (bits 5-0 do código) ainda não usados. Basta escolher um destes para codificar o ADDU3. Dos 64 códigos possíveis deste campo, 27 não usados. Estes códigos não usados são, em decimal(hexa): 5(0x5), 14(0xE), 20 a 23(0x14 a 0x17), 28 a 31(0x1C a 0x1F), 40(0x28), 41(0x29), 44 a 47(0x2C a 0x2F), 53(0x35) e todos entre 55 e 63(0x37 a 0x3F). Tomemos (aleatoriamente) o primeiro destes maior que o código do ADDU, 40(0x28). Com esta escolha, podemos definir o formato da nova instrução da seguinte maneira, notando que o decimal 40 corresponde ao hexadecimal 0x28:

	addu3	rd	rs	rt	rx	
	0	rs	rt	rd	rx	0x28
Número de bits/campo:	6	5	5	5	5	6

É possível verificar que nenhuma instrução do MIPS pode gerar um código objeto que siga este formato. Um exemplo de uso do ADDU3 seria:

```
ADDU3    $t0,$t1,$t2,$t3    # exemplo  de addu3
```

O seu código objeto seria 000000 01001 01010 01000 01011 101000, ou em hexadecimal , 0x012A42E8.

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