Prova P1 **Disciplina: Organização de Computadores**

Professor: Ney Laert Vilar Calazans

Aluno: 14/setembro/2012

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 8 triplas ??? a substituir). 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: Prestem muita atenção ao tratamento de endereços e rótulos.
   Dica 2: Muito cuidado com a mistura de representações numéricas: hexa, binário, complemento de 2, etc.
   Obrigatório: Mostre os desenvolvimentos para obter os resultados nas folhas anexas, justificando este desenvolvimento.
   Obrigatório: Mostre os desenvolvimentos para obter os resultados, justificando.

```
[1]   Endereço  Cód. Objeto Cód. Intermediário          Cód. Fonte
[2]  0x00400000  0x001f8021  addu $16,$0,$31     13     move  $s0, $ra
[3]  0x00400004  0x00004021  addu $8,$0,$0      14     move  $t0, $zero
[4]  0x00400008  0x00004821  addu $9,$0,$0      15     move  $t1, $zero
[5]  0x0040000c      ???          ???           17     mul   $t2, $t0, $t0
[6]  0x00400010  0x012a4821  addu $9,$9,$10     18     addu  $t1, $t1, $t2
[7]  0x00400014  0x25080001  addiu $8,$8,1      19     addiu $t0, $t0, 1
[8]  0x00400018  0x2101ffff  addi $1,$8,-1      20     ble   $t0, 100, loop
[9]  0x0040001c  0x28210064       ???
[10] 0x00400020  0x1420fffa  bne $1,$0,-6
[11] 0x00400024  0x3c011001  lui $1,4097        21     la    $a0, str
[12] 0x00400028  0x34240000  ori $4,$1,0
[13] 0x0040002c  0x0c100011       ???           22         ???
[14] 0x00400030  0x0010f821  addu $31,$0,$16     23     move  $ra, $s0
[15] 0x00400034  0x00092021  addu $4,$0,$9      24     move  $a0, $t1
[16] 0x00400038  0x0c100014  jal 4194384        25     jal   printInt
[17] 0x0040003c      ???          ???           26     move  $ra, $s0
[18] 0x00400040  0x03e00008  jr $31             27     jr    $ra
[19] 0x00400044  0x24020004  addiu $2,$0,4      30     li    $v0, 4
[20] 0x00400048  0x0000000c  syscall            31     syscall
[21] 0x0040004c  0x03e00008  jr $31             32     jr    $ra
[22] 0x00400050  0x24020001  addiu $2,$0,1      35     li    $v0, 1
[23] 0x00400054  0x0000000c  syscall            36     syscall
[24] 0x00400058      ???     jr $31             37     jr    $ra
```

2. (2,0 pontos) Um dos problemas a resolver durante o projeto de uma arquitetura de um processador é definir a codificação de suas instruções. Esta tarefa tem forte influência sobre o tamanho de programas, bem como sobre o desempenho e sobre o projeto do hardware do processador. No MIPS-I que usamos, por exemplo, optou-se por determinar que todas as instruções ocupem exatamente 32 bits. Assim, existem 2^{32} códigos possíveis de instrução. Claro, existem bem menos instruções no processador, pois, por exemplo, um salto pseudo-absoluto (J) possui um operando de 26 bits, gastando 2^{26} dos 2^{32} códigos distintos só para esta instrução. Questão a resolver: "Uma máquina possui instruções de exatamente 16 bits e operandos sempre de 4 bits. Do conjunto total de instruções 15 usam 3 operandos, 14 usam 2 operandos e 31 usam 1 operando. Qual é o número máximo de instruções sem operandos (com 0 operandos) que esta máquina pode ter?" A resposta é uma das opções abaixo. **Não adianta "chutar", pois a resposta não vale nada sem o desenvolvimento**. As opções são dadas apenas como dicas de em que faixa de valores a resposta correta se encontra. **Obs:** Esta questão baseou-se em um problema de uma prova do ENADE de Ciência da Computação na parte de Organização e Arquitetura.

[1] 15
[2] 16
[3] 31
[4] 63
[5] 128

3. (4,0 pontos) Dado o programa abaixo, escrito em linguagem de montagem do MIPS, responda às questões que seguem sobre o mesmo.

```
1.          .data
2.  TI:     .word  12
3.  R:      .word  3
4.  N:      .word  6
5.  SN:     .word  0
6.          .text
7.          .globl main
8.  main:
9.          la     $t0,TI
10.         lw     $t0,0($t0)
11.         la     $t1,R
12.         lw     $t1,0($t1)
13.         la     $t2,N
14.         lw     $t2,0($t2)
15.         la     $t3,SN
16.         beq    $t2,$zero,fim
17.         sw     $t0,0($t3)
18.         addiu  $t2,$t2,-1
19. loop:   beq    $t2,$zero,fim
20.         add    $t0,$t0,$t1
21.         lw     $t4,0($t3)
22.         addu   $t4,$t4,$t0
23.         sw     $t4,0($t3)
24.         addiu  $t2,$t2,-1
25.         j      loop
26. fim:    li     $v0,10
27.         syscall
```

Pede-se:

- (1,5 pontos) Diga o que faz o programa acima (do ponto de vista semântico), detalhando seu funcionamento, e diga o conteúdo final de posições de memória alteradas, se for o caso. Comente semanticamente o texto do programa.
- (1,0 ponto) O programa escreve algo na memória externa ao processador? Qual ou quais registradores possuem resultado relevante ao fim da execução?
- (1,5 pontos) Calcule qual o tamanho do programa, em bytes. Qual o tamanho da área de dados, em bytes?

Prova P1 **Disciplina: Organização de Computadores**

Gabarito **Professor: Ney Laert Vilar Calazans**

18/abril/2012

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 8 triplas ??? a substituir). 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: Prestem muita atenção ao tratamento de endereços e rótulos.
   Dica 2: Muito cuidado com a mistura de representações numéricas: hexa, binário, complemento de 2, etc.
   Obrigatório: Mostre os desenvolvimentos para obter os resultados nas folhas anexas, justificando este desenvolvimento.
   Obrigatório: Mostre os desenvolvimentos para obter os resultados, justificando.

```
[1]   Endereço  Cód. Objeto Cód. Intermediário          Cód. Fonte
[2]  0x00400000  0x001f8021  addu $16,$0,$31     13     move  $s0, $ra
[3]  0x00400004  0x00004021  addu $8,$0,$0      14     move  $t0, $zero
[4]  0x00400008  0x00004821  addu $9,$0,$0      15     move  $t1, $zero
[5]  0x0040000c      ???          ???           17     mul   $t2, $t0, $t0
[6]  0x00400010  0x012a4821  addu $9,$9,$10     18     addu  $t1, $t1, $t2
[7]  0x00400014  0x25080001  addiu $8,$8,1      19     addiu $t0, $t0, 1
[8]  0x00400018  0x2101ffff  addi $1,$8,-1      20     ble   $t0, 100, loop
[9]  0x0040001c  0x28210064       ???
[10] 0x00400020  0x1420fffa  bne $1,$0,-6
[11] 0x00400024  0x3c011001  lui $1,4097        21     la    $a0, str
[12] 0x00400028  0x34240000  ori $4,$1,0
[13] 0x0040002c  0x0c100011       ???           22         ???
[14] 0x00400030  0x0010f821  addu $31,$0,$16     23     move  $ra, $s0
[15] 0x00400034  0x00092021  addu $4,$0,$9      24     move  $a0, $t1
[16] 0x00400038  0x0c100014  jal 4194384        25     jal   printInt
[17] 0x0040003c      ???          ???           26     move  $ra, $s0
[18] 0x00400040  0x03e00008  jr $31             27     jr    $ra
[19] 0x00400044  0x24020004  addiu $2,$0,4      30     li    $v0, 4
[20] 0x00400048  0x0000000c  syscall            31     syscall
[21] 0x0040004c  0x03e00008  jr $31             32     jr    $ra
[22] 0x00400050  0x24020001  addiu $2,$0,1      35     li    $v0, 1
[23] 0x00400054  0x0000000c  syscall            36     syscall
[24] 0x00400058      ???     jr $31             37     jr    $ra
```

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

```
[5]  0x0040000c      ???          ???           17     mul   $t2, $t0, $t0
```

Obviamente, a questão consiste em demonstrar conhecimento de como se monta códigos intermediário e objeto de uma instrução. Note-se que se trata de uma instrução descrita apenas na versão mais recente do Apêndice A, aquela disponível em Português na página da disciplina.

Para produzir o código objeto, inicia-se consultando este Apêndice A, para obter o formato de instrução, o que fornece:

```
                          mul rd, rs, rt
                          0x1c rs rt rd 0 2
     Número de bits/campo: 6    5  5  5 5 6
```

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 011100 (0x1c em 6 bits), 01000 ($t0=$8, logo 8 em binário 5 bits, Rs), 01000 (idem para Rt), 01010 ($t2=$10, logo 10 em binário 5 bits, Rd), 00000 (0 em binário em 5 bits) e 000010 (2 em binário, 6 bits). Juntando os 6 campos, obtém-se 0111 0001 0000 1000 0101 0000 0000 0010, ou, em hexadecimal: **0x71085002**, que é o código objeto da instrução.

Resposta final:

```
[5] 0x0040000c 0x71085002  mul $10,$8,$8     17          mul    $t2,$t0,$t0

[9]  0x0040001c  0x28210064       ???
```

Neste item, o objetivo é gerar o código objeto da segunda das instruções produzidas pela pseudo-instrução `ble  $t0, 100, loop`, dado seu código objeto, ou seja, desmontar o código dado. Parte-se como sempre neste caso da observação dos 6 primeiros bits do código, ou seja, 001010, o que corresponde a 0xa em 6 bits. De posse deste valor entra-se com ele na Figura A.10.2 do Apêndice A (em Português) do livro-texto e descobre-se que se trata de uma instância da instrução SLTI, cujo formato é:

```
                          slti  rt, rs, imm
                          0xa   rs rt  imm
     Número de bits/campo:  6    5  5   16
```

De posse deste formato, basta desmontar o código com base nele, obtendo os códigos intermediário e fonte (001010 00001 00001 0000000001100100).

Resposta final:

```
[9]  0x0040001c  0x28210064       slti $1,$1,0x0064            não há código fonte

[13] 0x0040002c  0x0c100011       ???            22         ???
```

Trata-se de desmontagem de uma instrução regular. Expressando os 32 bits do código objeto em binário e separando-se os seis primeiros, tem-se 000011 00000100000000000000010001: Isto corresponde a uma instrução JAL, cujo formato é

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

Para montar os códigos intermediários e fonte, toma-se s 26 bits do campo target, acrescenta-se dois bits 00 a sua direita, e os quatro bits mais significativos do endereço da linha seguinte ao JAL (linha [14], 0x00400030) são acrescentados a sua esquerda. O resultado é 0000 0000 0100 0000 0000 0000 0100 0100, que corresponde em hexadecimal ao endereço 0x00400044. Como não há rótulo conhecido para esta linha os códigos intermediário e fonte permanecem idênticos:

Resposta Final:

```
[13] 0x0040002c  0x0c100011       jal 0x00400044    22    jal   0x00400044

[17] 0x0040003c      ???          ???            26     move  $ra, $s0
```

Neste item, o objetivo é a montagem dos códigos intermediário e objeto para da pseudo-instrução move. Como visto nas linhas [1], [2]. [3] e [14] do programa, esta pseudo é traduzida para uma instrução addu, que soma o registrador fonte com o registrador $zero e coloca o resultado no registrador de destino. Assim move é traduzida para addu $ra,$zero,$s0, que produz o código intermediário addu $31,$0,$16. Para obter o código objeto consulta-se o Apêndice A onde consta o formato da instrução addu, qual seja:

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

A partir deste formato e do código intermediário monta-se facilmente o código objeto. Em binário tem-se: 000000 00000 10000 11111 00000 100001. Reagrupando de 4 em 4 bits e traduzindo para hexadecimal tem-se o código objeto, que é 0x0010f821.

Resposta Final:

```
[17] 0x0040003c      0x0010f821  addu $31,$0,$16       26     move  $ra, $s0
[24] 0x00400058      ???     jr $31              37     jr    $ra
```

O que se deseja aqui é apenas gerar o código objeto da instrução, dado os códigos fonte e intermediário. O formato da instrução jr é:

```
                              jr   rs
                              0    rs   0   8
      Número de bits/campo:   6    5    1   6
```

A partir deste formato e do código intermediário monta-se facilmente o código objeto. Em binário tem-se: 000000 11111 000000000000000 001000. Reagrupando de 4 em 4 bits e traduzindo para hexadecimal tem-se o código objeto, que é 0x03e00008.

Resposta Final:

```
[24] 0x00400058      0x03e00008  jr $31          37     jr  $ra
```

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

2. (2,0 pontos) Um dos problemas a resolver durante o projeto de uma arquitetura de um processador é definir a codificação de suas instruções. Esta tarefa tem forte influência sobre o tamanho de programas, bem como sobre o desempenho e sobre o projeto do hardware do processador. No MIPS-I que usamos, por exemplo, optou-se por determinar que todas as instruções ocupem exatamente 32 bits. Assim, existem 2^{32} códigos possíveis de instrução. Claro, existem bem menos instruções no processador, pois, por exemplo, um salto pseudo-absoluto (J) possui um operando de 26 bits, gastando 2^{26} dos 2^{32} códigos distintos só para esta instrução. Questão a resolver: "Uma máquina possui instruções de exatamente 16 bits e operandos sempre de 4 bits. Do conjunto total de instruções 15 usam 3 operandos, 14 usam 2 operandos e 31 usam 1 operando. Qual é o número máximo de instruções sem operandos (com 0 operandos) que esta máquina pode ter?" A resposta é uma das opções abaixo. **Não adianta "chutar", pois a resposta não vale nada sem o desenvolvimento**. As opções são dadas apenas como dicas de em que faixa de valores a resposta correta se encontra. **Obs:** Esta questão baseou-se em um problema de uma prova do ENADE de Ciência da Computação na parte de Organização e Arquitetura.

```
[1]    15
[2]    16
[3]    31
[4]    63
[5]    128
```

Solução da Questão 2 (2,0 pontos)

O problema consiste em determinar o número total de códigos disponíveis para especificar instruções sem operandos, em uma máquina que possui instruções de 3, 2, 1 e 0 operandos. Assume-se, sem perda da generalidade que os 4 bits mais significativos das instruções de 3 operandos identificam a instrução, e os 12 menos significativos são os três operandos. Para ter 15 instruções de três operandos, gastam-se 15 códigos nos quatro bits mais significativos, sobrando apenas 1 código, que será usado para identificar as instruções que não são de três operandos (são de 2, 1 ou de 0 operando). Por exemplo:
0000 op1 op2 op3 a 1110 op1 op2 op3 são as 15 instruções de 3 operandos, sobrando o código de 4 bits 1111 para identificar todas as instruções de 2 operandos ou menos.

Procede-se de forma similar para as instruções de 2 operandos. Continuando o exemplo anterior (ainda sem perda da generalidade nmo raciocínio):
1111 0000 op1 op2 a 1111 1101 op1 ou op2 (14 instruções). Sobram os códigos 11111110 e 11111111 nos 8 bits mais significativos para identificar instruções de 1 ou menos operandos.

Para as instruções de 1 operando tem-se, continuando o exemplo:
111111100000 op1 a 111111111110 op1 são as 31 instruções de 1 operando. Resta apenas o código 111111111111 nos 12 bits mais significativos para identificar instruções de 0 operando. Como se pode variar os últimos quatro bits para identificar instruções de 0 operando o número máximo destas é o número máximo de códigos distintos de quatro bits, ou seja 16, que corresponde à opção [2].

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

3. (4,0 pontos) Dado o programa abaixo, escrito em linguagem de montagem do MIPS, responda às questões que seguem sobre o mesmo.

```
[1]             .data                   #
[2]     TI:     .word   12              #       Termo inicial da progressão aritmética (PA)
[3]     R:      .word   3               #       Razão da PA
[4]     N:      .word   6               #       Número de termos da PA a somar
[5]     SN:     .word   0               #       Local onde armazenar a soma dos termos da PA
[6]             .text                   #
[7]             .globl main             #       Área de dados do programa
[8]     main:                           # bytes
[9]             la      $t0,TI          # 8
[10]            lw      $t0,0($t0)      # 4     Carrega valor do termo inicial da PA em $t0
[11]            la      $t1,R           # 8
[12]            lw      $t1,0($t1)      # 4     Carrega valor da razão da PA em $t1
[13]            la      $t2,N           # 8
[14]            lw      $t2,0($t2)      # 4     Carrega número de termos a somar em $t2
[15]            la      $t3,SN          # 8     Carrega endereço onde guardar soma em $t3
[16]            beq     $t2,$zero,fim   # 4     Se N=0 nada a fazer, fim
[17]            sw      $t0,0($t3)      # 4     Senão guarda valor do termo inicial em SN
[18]            addiu   $t2,$t2,-1      # 4     Decrementa o contador de número de termos a somar
[19]    loop:   beq     $t2,$zero,fim   # 4     Se contador chegou a 0, fim
[20]            add     $t0,$t0,$t1     # 4     Senão, adiciona razão ao termo atual, gerando novo termo
[21]            lw      $t4,0($t3)      # 4     Lê de SN valor da soma dos termos atual
[22]            addu    $t4,$t4,$t0     # 4     Soma o termo seguinte, recém gerado em $t4
[23]            sw      $t4,0($t3)      # 4     Escreve soma atualizada em SN
[24]            addiu   $t2,$t2,-1      # 4     Decrementa contador de termos somados
[25]            j       loop            # 4     Volta a testar a chegada ao fim do laço
[26]    fim:    li      $v0,10          # 4     Fim, sai fora do programa
[27]            syscall                 # 4     chamada do sistema
```

Pede-se:

- (1,5 pontos) Diga o que faz o programa acima (do ponto de vista semântico), detalhando seu funcionamento, e diga o conteúdo final de posições de memória alteradas, se for o caso. Comente semanticamente o texto do programa.
- (1,0 ponto) O programa escreve algo na memória externa ao processador? Qual ou quais registradores possuem resultado relevante ao fim da execução?
- (1,5 pontos) Calcule qual o tamanho do programa, em bytes. Qual o tamanho da área de dados, em bytes?

Solução da Questão 3 (4,0 pontos)

- Este programa computa a soma dos N primeiros termos de uma progressão aritmética de termo inicial TI e razão R, armazenando o resultado da soma na variável SN. Comentários no texto do programa acima.
- Sim, o programa escreve valores em memória. Para ser exato, a cada volta do laço existente nas linhas [19] a [25] a posição de memória SN é atualizada com a soma de termos da PA calculada até o momento. A escrita se dá na linha [23] com a instrução SW. Antes de o laço começar, o termo inicial da PA é escrito em SN (linha [17]).
- Todas as instruções do MIPS possuem tamanho fixo de 4 bytes. No programa, todas as linhas da área de programa (linhas [9] a [27]) contêm uma instrução, exceto as linhas [9], [11], [13], [15] e [26], que contém pseudo-instruções. Destas exceções, as 4 primeiras são a pseudo la, que é traduzida para duas instruções (lui e ori) e a última é li, traduzida para um instrução (addiu). Assim, o tamanho da área de programa é Tam= 4*8+(27-8-4)*4=92bytes. A área de dados ocupa exatamente 4 palavras de memória, ou seja, 4*4=16bytes. O programa como um todo ocupa assim 108bytes.

Fim da Solução da Questão 3 (4,0 pontos)