Prova P1 **Disciplina: Organização e Arquitetura de Computadores II**

Professores: Fabiano Hessel e Ney Calazans

Aluno 07/outubro/2009

Valor das questões: 1) 4 pontos 2) 3 pontos 3) 3 pontos

1. Montagem/Desmontagem de código. Considere o trecho de programa abaixo, escrito na linguagem de montagem do MIPS, entremeado com algumas linhas contendo código-objeto. Assuma que este trecho é montado a partir do endereço de memória 0x00400000. (1) Gere o código objeto hexadecimal correspondente às instruções das linhas 10, 12 e 22 do trecho de programa e (2) Desmonte os códigos-objeto das linhas 21 e 23 para obter a representação em linguagem de montagem das instruções que deveriam estar nestas linhas. Justifique cada uma de suas ações durante a geração de código, com base no seu conhecimento da arquitetura MIPS e na documentação consultada.

Dica 1: Prestem muita atenção ao tratamento de endereços.

Dica 2: Cuidado com a mistura de números: hexa, binário, complemento de 2, etc.

Obrigatório: Mostre os desenvolvimentos para obter os resultados, justificando.

Número de Linha	Rótulo	Instrução / Especificação de dado	Endereço inicial do código objeto	Código objeto
1		.data		
2	str:	.asciiz "Hi Mom!"	0x10010000 0x10010004	0x4D206948 0x00216D6F
3	case:	.word 0x0		
4		.text		
5		.globl main		
6	main:	la $t0,str	0x00400000 0x00400004	0x3C011001 0x34280000
7		lw $t1,0($t0)		
8		andi $t6,$t1,0xFF		
9		la $t3,case		
10		lw $t4,0($t3)		
11		addiu $t5,$zero,4		
12	loop:	blez $t6,end		
13		addiu $t5,$t5,-1		
14		sltiu $t2,$t6,65		
15		bne $t2,$zero,nxt_ch		
16		sltiu $t2,$t6,91		
17		beq $t2,$zero,nxt_ch		
18		addiu $t4,$t4,1		
19	nxt_ch:	beq $t5,$zero,endw		
20		addu $t6,$t1,$zero		
21				0x000E7202
22		addu $t1,$t6,$zero		
23				0x31CE00FF
24		j loop		
25	endw:	addiu $t0,$t0,4		
26		lw $t1,0($t0)		
27		andi $t6,$t1,0xFF		
28		addiu $t5,$zero,4		
29		j loop		
30	end:	sw $t4,0($t3)		

2. Analise o código do programa abaixo e explique o que este programa faz. Inicie mapeando que registradores são utilizados, dizendo como cada um é inicializado e conclua descrevendo a função destes no código do programa. Comente semanticamente o código. Com os dados fornecidos, que valor é escrito na memória de dados ao final da execução do programa?

```
[1]          .text
[2]          .globl      main
[3]    main: addu        $s0,$zero,$zero
[4]          la          $s1,SOMA
[5]          la          $t0,N
[6]          lw          $t0,0($t0)
[7]          la          $t1,TI
[8]          lw          $t1,0($t1)
[9]          la          $t2,R
[10]         lw          $t2,0($t2)
[11]  prx_T: blez        $t0,fim
[12]         addu        $s0,$s0,$t1
[13]         addu        $t1,$t1,$t2
[14]         addiu       $t0,$t0,-1
[15]         j           prx_T
[16]  fim:   sw          $s0,0($s1)
[17]  end:   li          $v0,10
[18]         syscall
[19]
[20]         .data
[21]  TI:    .word 12
[22]  R:     .word 3
[23]  N:     .word 6
[24]  SOMA:  .word 0
```

3. Escreva um programa em linguagem de montagem (assembly language) do processador MIPS que processa um vetor de números inteiros (VET), contando quantos elementos pares e ímpares contém o vetor. Os resultados devem ser armazenados nas variáveis PAR e IMPAR em memória. Utilize a área de dados abaixo para escrever o seu programa. Implemente sua funcionalidade com instruções reais da arquitetura.

```
.data
VET:     .word 23 -43 55 -9 -7 21 -76 12 -45 -10
TAM:     .word 10
PAR:     .word 0
IMPAR:   .word 0
```

Gabarito

1. Montagem/Desmontagem de código. Considere o trecho de programa abaixo, escrito na linguagem de montagem do MIPS, entremeado com algumas linhas contendo código-objeto. Assuma que este trecho é montado a partir do endereço de memória 0x00400000. (1) Gere o código objeto hexadecimal correspondente às instruções das linhas 10, 12 e 22 do trecho de programa e (2) Desmonte os códigos-objeto das linhas 21 e 23 para obter a representação em linguagem de montagem das instruções que deveriam estar nestas linhas. Justifique cada uma de suas ações durante a geração de código, com base no seu conhecimento da arquitetura MIPS e na documentação consultada.

Dica 1: Prestem muita atenção ao tratamento de endereços.
Dica 2: Cuidado com a mistura de números: hexa, binário, complemento de 2, etc.
Obrigatório: Mostre os desenvolvimentos para obter os resultados, justificando.

Solução:

A primeira instrução (linha 10) é `lw $t4,0($t3)`. O registrador $t4 tem código decimal 12 e o registrador $t3 tem código decimal 11. O formato da instrução em linguagem de montagem é lw rt, address, onde address é expresso com um par de valores deslocamento(base=Rs). O formato binário, com campos de respectivamente 6, 5, 5, e 16 bits é 0x23 rs rt Offset. Logo, tem-se os seguintes 32 bits de código para a instrução: 100011 01011 01100 0000000000000000, que agrupando de 4 em 4 bits fica 1000 1101 0110 1100 0000 0000 0000 0000. Em hexadecimal, tem-se então o código objeto 0x8D6C0000.

A segunda instrução (linha 12) é `blez $t6,end`. O registrador $t6 tem código decimal 14. O formato da instrução em linguagem de montagem é blez rs, Rótulo. O formato binário, com campos de respectivamente 6, 5, 5 e 16 bits é 0x6 Rs 0 Deslocamento. O Deslocamento, conforme dito em aula pode ser computado calculando quantas linhas separam a instrução seguinte à blez da linha da instrução com rótulo end. Ora a linha seguinte ao blez é a linha 13 e a linha que contém o rótulo end é a linha 30. Como todas as linhas entre a 13 e a 30 possuem exatamente uma instrução do MIPS, o Deslocamento é 30-13=17. Então, tem-se os seguintes 32 bits de código para a instrução: 000110 01110 00000 0000000000010001, que agrupando de 4 em 4 bits fica 0001 1001 1100 0000 0000 0000 0001 0001. Em hexadecimal, tem-se então o código objeto 0x19C00011.

O terceiro código a montar (linha 22) é `addu $t1,$t6,$zero`. O registrador $t1 tem código decimal 9, o registrador $t6 tem código decimal 14, e $zero tem código decimal 0. O formato da instrução em linguagem de montagem é addu Rd,Rs,Rt. O formato binário, com campos de respectivamente 6, 5, 5, 5 e 6 bits é 0x0 Rs Rt Rd 0x0 0x21. Logo, tem-se os seguintes 32 bits de código para a instrução: 000000 01110 00000 01001 00000 100001, que agrupando de 4 em 4 bits fica 0000 0001 1100 0000 0100 1000 0010 0001. Em hexadecimal, tem-se então o código objeto 0x01C04821.

O primeiro código a desmontar (linha 21) é `0x000E7202`. Isolando-se os seis bits mais significativos do código obtém-se 000000, o que indica tratar-se de uma instrução do tipo R. Conforme as regras da arquitetura, deve-se então observar o valor dos seis bits menos significativos do código, que são 000010, correspondendo à constante 2. Entrando com estes dois valores na tabela da Figura A.19 do Apêndice A, descobre-se que se trata da instrução `srl`. Esta instrução tem o formato `srl rd, rt, shamt` em linguagem de montagem e formato binário 0 0 rt rd shamt 2, com campos de 6, 5, 5, 5, 5 e 6, respectivamente. Traduzindo o código objeto para binário agrupado de 4 em 4 bits

tem-se 0000 0000 0000 1110 0111 0010 0000 0010. Separando-se estes nos campos definidos no formato, obtém-se 000000 00000 01110 01110 01000 000010. Daí se retira que rt=01110=14=>$t6, rd=01110=14=>$t6, shamt=01000=8. Logo, o código fonte original era `srl $t6, $t6, 8`.

O segundo código a desmontar (linha 23) é `0x31CE00FF`. Isolando-se os seis bits mais significativos do código obtém-se 001100=0xC=12. Entrando com este valor na tabela da Figura A.19 do Apêndice A, descobre-se que se trata da instrução `andi`. Esta instrução tem o formato `andi rt, rs, imm` em linguagem de montagem e formato binário 0xc rs rt imm, com campos de 6, 5, 5, e 16, respectivamente. Traduzindo o código objeto para binário agrupado de 4 em 4 bits tem-se 0011 0001 1100 1110 0000 0000 1111 1111. Reagrupando estes nos campos definidos no formato, obtém-se 001100 01110 01110 0000000011111111. Daí se retira que rs=01110=14=>$t6, rt=01110=14=>$t6, e imm=0xFF. Logo, o código fonte original era `andi $t6, $t6, 0xFF`.

Número de Linha	Rótulo	Instrução / Especificação de dado		Endereço inicial do código objeto	Código objeto
1		.data			
2	str:	.asciiz	"Hi Mom!"	0x10010000 0x10010004	0x4D206948 0x00216D6F
3	case:	.word 0x0			
4		.text			
5		.globl	main		
6	main:	la	$t0,str	0x00400000 0x00400004	0x3C011001 0x34280000
7		lw	$t1,0($t0)	0x00400008	
8		andi	$t6,$t1,0xFF	0x0040000C	
9		la	$t3,case	0x00400010 0x00400014	
10		lw	$t4,0($t3)	0x00400018	0x8D6C0000
11		addiu	$t5,$zero,4	0x0040001C	
12	loop:	blez	$t6,end	0x00400020	0x19C00011
13		addiu	$t5,$t5,-1	0x00400024	
14		sltiu	$t2,$t6,65	0x00400028	
15		bne	$t2,$zero,nxt_ch	0x0040002C	
16		sltiu	$t2,$t6,91	0x00400030	
17		beq	$t2,$zero,nxt_ch	0x00400034	
18		addiu	$t4,$t4,1	0x00400038	
19	nxt_ch:	beq	$t5,$zero,endw	0x0040003C	
20		addu	$t6,$t1,$zero	0x00400040	
21		srl	$t6,$t6,8	0x00400044	0x000E7202
22		addu	$t1,$t6,$zero	0x00400048	0x01C04821
23		andi	$t6,$t6,0xFF	0x0040004C	0x31CE00FF
24		j	loop	0x00400050	0x08100008
25	endw:	addiu	$t0,$t0,4		
26		lw	$t1,0($t0)		
27		andi	$t6,$t1,0xFF		
28		addiu	$t5,$zero,4		
29		j	loop		
30	end:	sw	$t4,0($t3)		

2. Analise o código do programa abaixo e explique o que este programa faz. Inicie mapeando que registradores são utilizados, dizendo como cada um é inicializado e conclua descrevendo a função destes no código do programa. Comente semanticamente o código. Com os dados fornecidos, que valor é escrito na memória de dados ao final da execução do programa?.

Solução:

```
[1]  # Programa que calcula a soma dos termos de uma Progressão Aritmética (PA)
[2]  # Author:  Ney Calazans
[3]  #
[4]
[5]         .text
[6]         .globl      main
[7]  main:
[8]         xor    $s0,$zero,$zero   # $s0=reg que guarda a soma dos termos da PA
[9]         la     $s1,SOMA     # Carrega em $s1 o endereço de SOMA
[10]        la     $t0,N        # Carrega em $t0 o endereço de N
[11]        lw     $t0,0($t0)   # t0=N, inicialmente o núm de termos a somar da PA
[12]        la     $t1,TI       # Carrega em $t1 o endereço de TI
[13]        lw     $t1,0($t1)   # $t1=TI, o Termo Inicial da PA
[14]        la     $t2,R        # Carrega em $t2 o endereço de R
[15]        lw     $t2,0($t2)   # $t2=R, a razão da PA
[16] prox_T:blez   $t0,fim      # Se contador zerado, fim
[17]        addu   $s0,$s0,$t1 # Senão, adiciona termo à soma
[18]        addu   $t1,$t1,$t2 # Gera o próximo termo, somando R ao termo anterior
[19]        addiu $t0,$t0,-1   # Decrementa o contador
[20]        j      prox_T      # Continue a executar o laço
[21] fim:   sw     $s0,0($s1)  # Estoca resultado da soma dos termos da PA em SOMA
[22] end:   li     $v0,10
[23]        syscall            # E cai fora do programa
[24]
[25]        .data              # segmento de dados
[26] TI:    .word 12           # Termo Inicial da PA
[27] R:     .word 3            # Razão da PA
[28] N:     .word 6            # Número de termos a somar da PA
[29] SOMA:  .word 0            # Repositório da soma dos N primeiros termos da PA
```

Com os dados fornecidos o resultado escrito na variável soma é 12+15+18+21+24+27=117.

3. Escreva um programa em linguagem de montagem (assembly language) do processador MIPS que processa um vetor de números inteiros (VET), contando quantos elementos pares e ímpares contém o vetor. Os resultados devem ser armazenados nas variáveis PAR e IMPAR em memória. Utilize a área de dados abaixo para escrever o seu programa. Implemente sua funcionalidade com instruções reais da arquitetura.

Solução:

```
            .text
            .globl      main
main:       la          $t0,VET             # Gera ponteiro para vetor em $t0
            la          $t1,TAM
            lw          $t1,0($t1)          # Inicializa $t1 com o tamanho do vetor
            add         $t2,$zero,$zero     # $t2=contador de pares, no início 0
            add         $t3,$zero,$zero     # $t3=contador de ímpares, no início 0

loop:       beq         $t1,$zer0,end       # Qdo cont. de elementos chega a 0, fim
            lw          $t4,0($t0)          # Lê elemento do vetor
            andi        $t4,$t4,1           # Este AND escreve 0 em $t4 se número
                                            # é par, senão e escreve 1 em $t4
            beq         $t4,$zero, ehpar    # Salta se for par
            addi        $t3,$t3,1           # Senão, incrementa contador de ímpares
            j           loopend             # E vai finalizar laço
ehpar:      addi        $t2,$t2,1           # Incrementa contador de pares
loopend:    addi        $t0,$t0,4           # Avança ponteiro para próximo elemento
            addi        $t1,$t1,-1          # Decr contador de elementos a tratar
            j           loop                # Executa novo teste-iteração

end:        la          $t0,PAR             # Obtém ponteiro para PAR
            sw          $t2,0($t0)          # Escr. no. de pares na variável PAR
            la          $t0,IMPAR           # Obtém ponteiro para IMPAR
            sw          $t3,0($t0)          # Escr. no. de ímpares na variável IMPAR

            li          $v0,10
            syscall                         # E cai fora do programa

            .data
   VET:     .word       23 -43 55 -9 -7 21 -76 12 -45 -10
   TAM:     .word       10
   PAR:     .word       0
   IMPAR:   .word       0
```