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

Professores: Fabiano Hessel e Ney Calazans

Aluno 22/abril/2009

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

1. O fragmento de código abaixo processa um array unidimensional de palavras e retorna 2 valores importantes em $v0 e $v1. Assuma que o array tem armazenado 5.000 palavras indexadas de 0 a 4.999, sabendo que o endereço base do array está armazenado em $a0 e o tamanho do array (5.000) está inicialmente armazenado em $a1. Descreva o que faz o programa e o que é retornado 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
```

2. Escreva um programa em linguagem de montagem (*assembly language*) do processador MIPS que processa um vetor de números inteiros (VET), transformando cada elemento do vetor em seu valor absoluto. Por exemplo, o valor absoluto do número -43 é +43 e do número +55 é +55. O programa deve obrigatoriamente chamar uma sub-rotina **calc_abs** que recebe como parâmetro um número e calcula o valor absoluto deste, retornando-o segundo as convenções do MIPS. O valor retornado pela sub-rotina deve ser armazenado, pelo programa principal, na mesma posição do array que continha o elemento original. Utilize a área de dados abaixo para escrever o seu programa.

   Dica: Como a subrotina **calc_abs** é claramente folha (não chama nenhuma outra), e como não se especifica como deve ser feita a passagem de parâmetros, não é necessário usar a pilha nesta questão.

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

3. 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 [6], [11] e [12] do trecho de programa e (2) Desmonte os códigos-objeto das linhas [8] e [12] 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.

```
[1]    main:        addu        $s1, $zero, $sp
[2]                 addu        $s2, $zero, $zero
[3]                 la          $s3, valor
[4]                 lw          $s3, 0($s3)
[5]    loop:        addiu       $sp,$sp,-4
[6]                 sw          $s3,0($sp)
```

```
[7]                 addiu       $s3,$s3,-1
[8]                 0x1A600001
[9]                 j           loop
[10]    loop2:      lw          $s4, 0($sp)
[11]                addu        $s2, $s2, $s4
[12]                0x27BD0004
[13]                slt         $s5, $sp, $s1
[14]                blez        $s5, fim
[15]                j           loop2
[16]    fim:        li          $v0,10
[17]                syscall
[18]                .data
[19]    valor:      .word       322
```

Gabarito da P1 de 22/abril/2009

1. O fragmento de código abaixo processa um array de palavras e retorna 2 valores importantes em $v0 e $v1. Assuma que o array tem armazenado 5.000 palavras indexadas de 0 a 4.999, sabendo que o endereço base do array está armazenado em $a0 e o tamanho do array (5.000) em $a1. Descreva em uma frase o que faz o programa e o que é retornado 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
```

Resposta: O código determina qual palavra aparece no array com maior freqüência, retornando a palavra em $v1 e o número de vezes que ela aparece em $v0.

2. Escreva um programa em linguagem de montagem (*assembly language*) do processador MIPS que processa um vetor de números inteiros (VET), transformando cada elemento do vetor em seu valor absoluto. Por exemplo, o valor absoluto do número -43 é +43 e do número +55 é +55. O programa deve obrigatoriamente chamar uma sub-rotina **calc_abs** que recebe como parâmetro um número e calcula o valor absoluto deste, retornando-o segundo as convenções do MIPS. O valor retornado pela sub-rotina deve ser armazenado, pelo programa principal, na mesma posição do array que continha o elemento original. Utilize a área de dados abaixo para escrever o seu programa.

   Dica: Como a subrotina **calc_abs** é claramente folha (não chama nenhuma outra), e como não se especifica como deve ser feita a passagem de parâmetros, não é necessário usar a pilha nesta questão.

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

Resposta:

Programa Principal

```
main:
      la    $s0,VET           # gera ponteiro para VET em $t0
      la    $s1,TAM           #
      lw    $s1,0($s1)        # carrega $s1 com tam de VET (num de elementos a processar)
loop:
      beq   $s1,$zero,fim     # fim se processou todos os elementos
      lw    $a0,0($s0)        # senão, pega elemento de VET e põe como arg. de calc_abs
      jal   calc_abs          # chama subrotina de cálculo de valor absoluto
      sw    $v0,($s0)         # escreve valor absoluto sobre valor anterior do elemento
      addiu $s1,$s1,-1        # decrementa contador de elementos a processar
      addiu $s0,$s0,4         # incrementa ponteiro para próximo elemento de VET
      j     loop              # inicia nova iteração do laço
fim:
      li    $v0,10            # cai fora do programa
```

```
      syscall
```

Subrotina de cálculo do valor absoluto

```
calc_abs:
      move  $v0,$a0           # copia argumento para registrador de valor de retorno
      bgez  $v0,fim_ca        # se argumento maior ou igual a zero, terminou
      subu  $v0,$zero,$v0     # senão, inverte o sinal do argumento
fim_ca:
      jr    $ra         # retorna

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

3. 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 [6], [11] e [12] do trecho de programa e (2) Desmonte os códigos-objeto das linhas [8] e [12] 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.

```
[1]   main:           addu        $s1, $zero, $sp
[2]                   addu        $s2, $zero, $zero
[3]                   la          $s3, valor
[4]                   lw          $s3, 0($s3)
[5]   loop:           addiu       $sp,$sp,-4
[6]                   sw          $s3,0($sp)
[7]                   addiu       $s3,$s3,-1
[8]                   0x1A600001
[9]                   j           loop
[10]  loop2:          lw          $s4, 0($sp)
[11]                  addu        $s2, $s2, $s4
[12]                  0x27BD0004
[13]                  slt         $s5, $sp, $s1
[14]                  blez        $s5, fim
[15]                  j           loop2
[16]  fim:            li          $v0,10
[17]                  syscall
[18]                  .data
[19]  valor:          .word       322
```

Respostas:

```
Linha [6]       sw          $s3,0($sp)
```

A instrução sw utiliza o formato I com três operandos, e tem o seguinte formato:

```
                      sw   rt,Offset(rs)
                     0x2B  rs  rt Offset
Número de bits/campo:  6    5   5   16
```

Dado este formato, tem-se o seguinte código binário: 101011 (constante 0x2B em 6 bits), concatenado com 11101 ($sp=$29 em 5 bits), concatenado com 10011 ($s3=$19 em 5 bits), concatenado com 0000000000000000 (o offset, que é 0, em 16 bits). O resultado é o vetor 1010 1111 1011 0011 0000 0000 0000 0000 em binário, ou 0xAFB30000 em hexadecimal. Este é o código de 32 bits da instrução sw em questão.

```
Linha [11]      addu        $s2, $s2, $s4
```

A instrução addiu com três operandos tem o seguinte formato:

```
                     addu rd,rs,rt
                     0x0  rs rt rd 0 0x21
Número de bits/campo:  6   5  5   5 5  6
```

(2) Dado este formato, tem-se o seguinte código binário: 000000 (constante 0 em 6 bits), concatenado com 10010 ($s2=$18 em 5 bits), concatenado com 10100 ($s4=20 em 5 bits), concatenado com 10010 ($s2=$18 em 5 bits), concatenado com 00000 (a constante 0 em 5 bits) concatenado com 100001 (a constante 0x21 em 6 bits). O resultado é o vetor 0000 0010 0101 0100 1001 0000 0010 0001 em binário, ou 0x02549021 em hexadecimal. Este é o código de 32 bits da instrução addu em questão.

```
Linha [15]      j           loop2
```

A instrução addiu com três operandos tem o seguinte formato:

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

(3) Dado este formato, tem-se o seguinte código binário: 000010 (constante 2 em 6 bits), que deve ser concatenado com um valor de 26 bits gerado a partir do endereço do rótulo loop2. Tal endereço pode ser calculado como sendo 0x00400028. Chega-se a este valor da seguinte maneira: o rótulo loop2 está na 10ª linha do programa que inicia no endereço 0x00400000. Nas 9 linhas anteriores, existem 8 instruções que ocupam 8*4=32 posições de memória e uma pseudo-instrução (la na linha 3) que ocupa 8 bytes, totalizando 40 posições antes da linha contendo o rótulo loop2. Traduzindo 40 para hexadecimal, obtém-se 0x28, que somado a 0x00400000 dá 0x00400028. O endereço de memória onde inicia a instrução da linha com rótulo loop2. Para obter os 26 bits do segundo campo da instrução j loop2, traduz-se o endereço 0x00400028 para binário (0000 0000 0100 0000 0000 0000 0010 1000) e descarta-se os 4 bits mais significativos e os dois bits menos significativos. O código objeto final da instrução fica 0000 1000 0001 0000 0000 0000 0000 1010 em binário, ou 0x0810000A em hexadecimal. Este é o código de 32 bits da instrução j em questão.

(4) Dado o código hexadecimal 0x1A600001, sabe-se que os 6 primeiros bits definem o código da instrução, ou pelo menos a classe desta. Neste caso, o valor dos 6 primeiros bits em binário é 000110, ou 0x06 em hexadecimal. Assim, consultando-se a documentação do ISA do MIPS, descobre-se que se trata da instrução blez. Esta instrução possui o formato (6 Rs 0 Offset), onde os campos possuem, da esquerda para a direita os tamanhos 6, 5, 5 e 16, respectivamente. Nota-se que Rs é neste caso o registrador especificado pelos 5 bits seguintes ao código da operação, ou seja, 10011. Assim, trata-se do registrador $19, também denominado $s3. Resta ainda extrair o lugar para onde se salta. O offset de 16 bits é 0000 0000 0000 0001. Assim, trata-se de 1 número positivo, que, convertido para decimal corresponde a 1. Assim, quando o salto é tomado salta-se para a instrução que se encontra uma linha depois da instrução seguinte ao blez, ou seja, aquela que contém o rótulo **loop2**. Assim a instrução que gerou este código objeto resultou da tradução da linha em linguagem de montagem blez $s3, loop2, ou qualquer texto equivalente.

(5) Dado o código hexadecimal 0x27BD0004, sabe-se que os 6 primeiros bits definem o código da instrução, ou pelo menos a classe desta. Neste caso, o valor dos 6 primeiros bits em binário é 001001, ou 0x09 em hexadecimal. Assim, consultando-se a documentação do ISA do MIPS, descobre-se que se trata da instrução addiu. Esta instrução possui formato em linguagem de montagem que é addiu Rt, Rs, Imm, e formato binário (9 Rs Rt Imm), onde os campos possuem, da esquerda para a direita, os tamanhos 6, 5, 5 e 16, respectivamente. Nota-se que Rs é neste caso o registrador especificado pelos 5 bits seguintes ao código da operação, ou seja, 11101. Assim, trata-se do registrador $29, também denominado $sp, o apontador do topo da pilha. Os cinco bits seguintes, que especificam o registrador Rt são 11101, ou seja, o mesmo registrador $29/$sp. O dado imediato corresponde ao valor dos últimos 16 bits, ou seja, o número 4. Assim, a instrução que gerou este código objeto resultou da tradução da linha em linguagem de montagem addiu $sp,$sp,4, ou qualquer texto equivalente.