UNIVERSIDADE DE SÃO PAULO

EACH – Bacharelado em Sistemas de Informação

DISCIPLINA – Organização de Computadores Digitais

1º Semestre - 2017

Relatório MIPS

Docente - Profa. Dra. Gisele da Silva Craveiro

São Paulo

2017

Índice

[Seção 1 – Organização e Arquitetura MIPS 3](#__RefHeading__731_2031764727)

[1.1 – Visão Geral 3](#__RefHeading__733_2031764727)

[1.2 – Registradores 3](#__RefHeading__735_2031764727)

[1.3 – Instruções 4](#__RefHeading__737_2031764727)

[Seção 2 – Explicação detalhada das instruções utilizadas no código feito pela equipe 11](#__RefHeading__739_2031764727)

[Seção 3 – Descrição do problema e código alto nível da solução 13](#__RefHeading__741_2031764727)

[3.1 – Problema sugerido (3.b) 13](#__RefHeading__743_2031764727)

[3.2 – Código em linguagem de alto nível (Java) 13](#__RefHeading__745_2031764727)

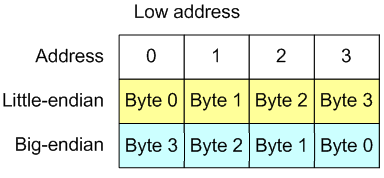
[Seção 4 – Código em Assembly desenvolvido 16](#__RefHeading__747_2031764727)

[Seção 5 – Referências: 20](#__RefHeading__749_2031764727)

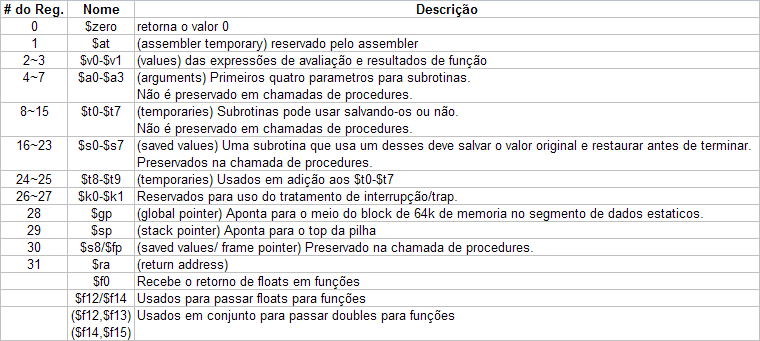
Seção 1 – Organização e Arquitetura MIPS

1.1 – Visão Geral

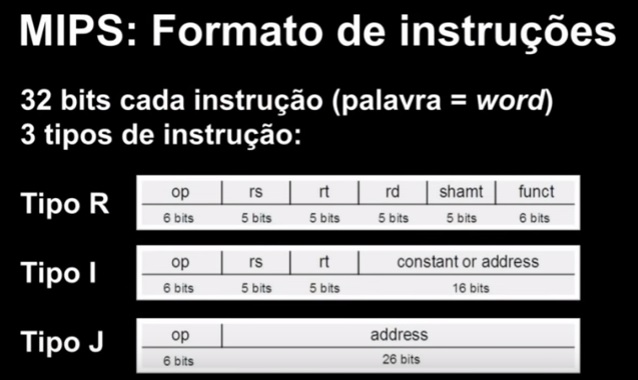
* Dados e endereços na arquitetura MIPS são de 32 bits (Palavra de 32 bits)
* Barramento único de 32 bits para dados e instruções
* Barramento separado de 32 bits para endereçamento
* Barramento único de 32 bits para dados e instruções
* Barramento separado de 32 bits para endereçamento
* Memória endereçada por byte
  + Endereços precisam ser múltiplos de 4
* Tres operandos -> duas origens, um destino
  + add a, b, c  a = b + c
* Big Endian - byte mais significativo no endereço está no byte menos significativo da palavra [tabela]



1.2 – Registradores



* $t0 a $t9 – valores temporários
* $s0 a $s7 – variáveis a serem salvas
* Registrador ZERO ($zero) – é a constante zero / não pode ser sobrescrito (muito usado – for)
  + Também usado para mover conteúdo – add $t2, $s1, $zero (soma zero ao valor em $s1 e armazena em $t2, ou seja, move o valor de $s1 para $t2)
* MIPS **não** tem instrução de movimentação

1.3 – Instruções

* **Leitura/Escrita**
  + Acesso à memória com instruções de escrita e leitura
  + As outras instruções geralmente utilizam registradores como operando
* **Leitura/Escrita de endereçamento direto**
  + lw - registrador, end\_na\_mem
    - copia word da posição da ram dada, para o registrador dado.
  + Lb - registrador, end\_na\_mem
    - copia byte da posição da ram dada, para a parte baixa do registrador dado.
  + li - registrador, valor
    - carrega o valor para o registrador de destino.

**Escrita:**

* sw registrador, end\_na\_mem
  + escreve a word do registrador dado na posição da ram dada.
* sb registrador, end\_na\_mem
  + escreve o byte da parte mais baixa do registrador dado para a posição da ram

**Leitura/Escrita de endereçamento indireto e por base**

**Leitura**

* la - registrador, label
  + copia o endereço do label na memória para o registrador dado

Endereçamento indireto

* lw - registrador1, (registrador2)
  + carrega a word que está no endereço dado pelo registrador2, para o registrador1

Endereçamento por base

* lw - registrador1, offset(registrador2)
* carrega a word que está no endereço (registrador2 + offset) para o registrador1

**Escrita**

Endereçamento indireto

* sw - registrador1, (registrador2)
  + copia a word no registrador1 para posição de memória de endereço dado pelo registrador2

Endereçamento por base

* sw - registrador1, offset(registrador2)
  + copia a word no registrador1 para posição de memória de endereço dado por (registrador2+offset)

**Movimentação**

* move - registrador0, registrador1
  + copia o valor do registrador1 para o registrador0
* mfhi - registrador
  + copia o valor do registrador Hi para o registrador0
* mflo - registrador0
  + copia o valor do registrador Lo para o registrador0

**Aritméticas**

* add - registrador0, registrador1, registrador2
  + salva o resultado da soma do registrador1 com o registrador2 no registrador0
* addi - registrador0, registrador1, imediato
  + salva o resultado da soma do registrador1 com o imediato no registrador0
* addu - registrador0, registrador1, registrador2
  + salva o resultado da soma do registrador1 com o registrador2 no registrador0
* sub - registrador0, registrador1, registrador2
  + salva o resultado da subtração do registrador1 c/ o registrador2 no registrador0
* subu - registrador0, registrador1, registrador2
  + salva o resultado da subtração do registrador1 c/ o registrador2 no registrador0
* mul - registrador0, registrador1, registrador2
  + multiplica o registrador1 pelo registrador2 e guarda no registrador0
* mulo - registrador0, registrador1, registrador2
  + multiplica o registrador1 pelo registrador2 e guarda no registrador0
* mulou - registrador0, registrador1, registrador2
  + multiplica o registrador1 pelo registrador2 e guarda no registrador0
* mult - registrador1, registrador2
  + multiplica o registrador1 pelo registrador2 e guarda nos registradores especiais
* multu - registrador1, registrador2
  + multiplica o registrador1 pelo registrador2 e guarda nos registradores especiais
* div - registrador0, registrador1, registrador2
  + guarda o resultado da divisão inteira do reg1 pelo reg2 no registrador0
* divu - registrador0, registrador1, registrador2
  + guarda o resultado da divisão inteira do reg1 pelo reg2 no registrador0
* div - registrador1, registrador2
  + divide o registrador1 pelo registrador2 e guarda nos registradores especiais
* divu - registrador1, registrador2
  + divide o registrador1 pelo registrador2 e guarda nos registradores especiais
* rem - registrador0, registrador1, registrador2
  + guarda o resto da divisão do registrador1 pelo registrador2 no registrador0
* remu - registrador0, registrador1, registrador2
  + guarda o resto da divisão do registrador1 pelo registrador2 no registrador0

**Lógicas**

* and - - registrador0, registrador1, registrador2
  + guarda o resultado da operação lógica AND entre reg1 e reg2 no registrador0
* andi - registrador0, registrador1, imediato
  + guarda o resultado da operação lógica AND entre reg1 e imed no registrador0
* neg - registrador0, registrador1
  + guarda o inverso do valor do registrador1 no registrador0
* negu - registrador0, registrador1
  + guarda o inverso do valor do registrador1 no registrador0
* nor -registrador0, registrador1, registrador2
  + guarda o resultado da operação lógica NOR entre reg1 e reg2 no registrador0
* not - registrador0, registrador1
  + guarda o resultado da negação binária do valor do registrador1 no registrador0
* or - registrador0, registrador1, registrador2
  + guarda o resultado da operação lógica OR entre reg1 e reg2 no registrador0
* ori - registrador0, registrador1, imediato
  + guarda o resultado da operação lógica OR entre reg1 e imed no registrador0
* rol - registrador0, registrador1, imediato
  + guarda o resultado da rotação para esquerda, de distancia dada pelo valor do imediato, de bits do valor do registrador1 no registrador0
* ror - registrador0, registrador1, imediato
  + guarda o resultado da rotação para direita, de distancia dada pelo valor do imediato, de bits do valor do registrador1 no registrador0
* sll - registrador0, registrador1, imediato
  + guarda o resultado do deslocamento lógico para esquerda, de distancia dada pelo valor do imediato, de bits do valor do registrador1 no registrador0
* sla - registrador0, registrador1, imediato
  + guarda o resultado do deslocamento aritmético para esquerda, de distancia dada pelo valor do imediato, de bits do valor do registrador1 no registrador0
* srl - registrador0, registrador1, imediato
  + guarda o resultado do deslocamento lógico para direita, de distancia dada pelo valor do imediato, de bits do valor do registrador1 no registrador0
* sra - registrador0, registrador1, imediato
  + guarda o resultado do deslocamento aritmético para direita, de distancia dada pelo valor do imediato, de bits do valor do registrador1 no registrador0
* xor - - registrador0, registrador1, registrador2
  + guarda o resultado da operação lógica XOR entre reg1 e reg2 no registrador0
* xori - registrador0, registrador1, imediato
  + guarda o resultado da operação lógica XOR entre reg1 e imed no registrador0

**Desvio**

**Incondicional**

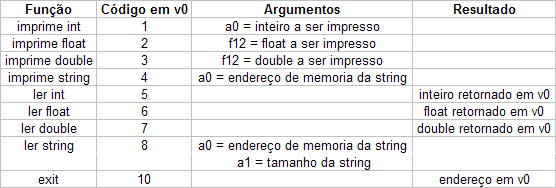
* b - label
  + muda o registrador PC(registrador que guarda o endereço da próxima instrução a ser executada) para o valor do label
* j - label
  + muda o registrador PC(registrador que guarda o endereço da próxima instrução a ser executada) para o valor do label
* jr - registrador
  + muda o registrador PC(registrador que guarda o endereço da próxima instrução a ser executada) para o endereço contido no registrador

**Condicional**

* beq - registrador0, registrador1, label
  + desvia para o label, se: registrador0 = registrador1
* blt - registrador0, registrador1, label
  + desvia para o label, se: registrador0 < registrador1
* ble - registrador0, registrador1, label
  + desvia para o label, se: registrador0 <= registrador1
* bgt - registrador0, registrador1, label
  + desvia para o label, se: registrador0 > registrador1
* bge - registrador0, registrador1, label
  + desvia para o label, se: registrador0 >= registrador1
* bne - registrador0, registrador1, label
  + desvia para o label, se: registrador0 != registrador1

**Entrada/Saída e chamadas de sistema**

* Usado para ler ou imprimir valores ou strings da Entrada/Saída, e indicar o fim de programa
* Usa a chamada de rotina **syscall**
* Deve-se informar os valores necessários nos registradores v0, a0 e a1
* Se houver um retorno, ele será no registrador v0



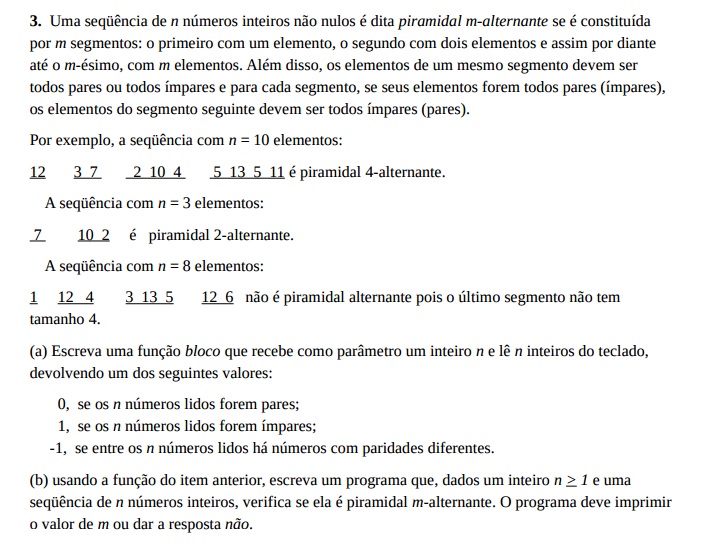
* A função de imprimir string espera uma string terminada com caractere nulo. A diretiva .asciiz cria uma string terminada em caracter nulo.
* A leitura de inteiros, floats e doubles lêem um liha inteira, inclusive o caractere de nova linha.
* A função exit finaliza o programa.

Seção 2 – Explicação detalhada das instruções utilizadas no código feito pela equipe

|  |  |
| --- | --- |
| LI / Load Immediate – Params: $R e número | Recebe um parâmetro numérico imediato e coloca-o no registrador alvo.  Coloca no IR o OpCode de Soma, o endereço alvo, o endereço do registrador $zero e o valor imediato entrado. Instrução no formato I |
| Syscall | Busca o que está armazenado no registrador $v0, executa, e coloca no registrador $v0 o potencial resultado. |
| Jal / Jump and Link – Params: $R | Coloca o que estiver presente no ProgramCounter em $ra e executa uma função de salto para o endereço de parâmetro |
| LA / Load Address – Params $alvo, ($buscado) | Coloca no IR o OpCode de soma, o endereço alvo, e o valor que está armazenado no endereço buscado, é então executado e o valor da soma é colocado no endereço alvo.  Instrução no formato I |
| BGE / Branch if Greater or Equal – Params $a, $b, end. destino | Verifica se $a < $b, e coloca num registrador temporário o valor 1 caso verdadeiro, e o valor 0 caso contrário. Depois disso verifica o valor colocado no registrador temporário e pula para o destino caso seja 0 |
| ADD – Params $r0, $r1, $r2 | Coloca em $r0 o valor resultado da soma $r1 e $r2 |
| ADDI – Add Imediato – Params $r0, $r1, imm | Executa a instrução Add com os parâmetros $r0, $zero, e o valor imediato |
| BEQ – Branch if Equals – Params $r0, $r1, end | Pula para o endereço end se o valor em $r0 for igual ao valor em $r1 |
| BEQZ – Branch if Equals Zero – Params $r0, end | Executa a instrução BEQ com os parâmetros $r0, $zero, end |
| BNE – Branch if Not Equal – Params $a, $b e end | Verifica se $a < $b ou $a > $b, e coloca num registrador temporário o valor 1 caso qualquer um seja verdadeiro, e o valor 0 caso contrário. Depois disso verifica o valor colocado no registrador temporário e pula para o destino caso seja 0 |
| J / Jump – Params end | Pula para o endereço end |
| Mul / Multiply – Params $r0, $r1, $r2 | Coloca em $r0 o valor resultante da multiplicação $r1 \* $r2 |
| BLEZ / Brench if Less or Equal Zero – Params $a, $b, end | Verifica se $a0 for menor que 0 e $a0 não for maior que 0. Coloca num registrador temporário 1 caso verdadeiro para os dois e 0 caso contrário. Se for verdadeiro, pula para end |
| SUBI / Subtract Immediate – Params $a, $b, $c | Executa a função ADDI com os parâmetros $a, $b e valor $c com o bit de sinal invertido |
| REM / Remainder – Params $a, $b, $c | Coloca no registrador $a o valor do resto da divisão $b por $c. Possui um registrador exclusivo para fazer operações de divisão, o *hi* e o *lo,* que guardam os valores intermediários da divisão. |
| BNEZ – Branch if not equal 0 – Params $a0, end | Executa a função BNE com os parâmetros $a0, $zero, end |
| JR – Jump Register – Params $a | Recupera o endereço presente no registrador $a e executa a instrução Jump para ele |
| SUB / Subtract – Params $a, $b, $c | Executa a função ADD com os parâmetros $a, $b, e valor $c com o bit de sinal invertido |

Seção 3 – Descrição do problema e código alto nível da solução

3.1 – Problema sugerido (3.b)



3.2 – Código em linguagem de alto nível (Java)

import java.util.Scanner;

public class EpOCD {

private static int bloco(int n) {

Scanner s = new Scanner(System.in);

int total = 2;

while (n > 0) {

n--;

int next = s.nextInt();

if (total == 2)

total = next % 2;

else if (total == 1) {

if (next % 2 != 0) {

} else {

total = -1;

}

} else if (total == 0) {

if (next % 2 == 0) {

} else {

total = -1;

}

}

}

return total;

}

public static boolean verificaPiramidavel(int elementos) {

int i = elementos;

for (int j = 0; j < elementos; j++) {

i -= j + 1;

if (i == 0)return true;

}

return false;

}

public static void main(String[] args) {

Scanner s = new Scanner(System.in);

int blocosALer = s.nextInt();

if (!verificaPiramidavel(blocosALer)){

System.out.println("não");

return;

}

int proxBlocoDeveSer = 0;

int j = 0;

for (int i = 0; i < blocosALer;) {

int numerosNoBloco = j + 1;

j++;

int blocoTotal = bloco(numerosNoBloco);

if (blocoTotal == -1){

System.out.println("não");

return;

}

if (i == 0) {

proxBlocoDeveSer = inverteBit(blocoTotal);

}else {

if (blocoTotal != proxBlocoDeveSer) {

System.out.println("não");

return;

}

proxBlocoDeveSer = inverteBit(blocoTotal);

}

i = numerosNoBloco + i;

}

System.out.println(j);

}

private static int inverteBit(int bit) {

return bit \* -1 + 1;

}

}

Seção 4 – Código em Assembly desenvolvido

.text

main:

li $v0, 5

syscall

jal verificapiramide

li $t9, 0 #proxBlocoDeveSer

li $t8, 0 #j

li $t7, 0 #i do for

la $t6, ($v0) # blocos a ler

loopMain:

bge $t7, $t6, printaJeSai

li $t5, 1 #numeros no bloco, inicia em 1 pra somar j na linha de baixo

add $t5, $t5, $t8 # coloca no $t5 / numeros no bloco o valor j + 1

addi $t8, $t8, 1 # j++

la $a0, ($t5) #Argumento de bloco: numero no blocos

la $s2 ($a0)

jal bloco #entrando em bloco com argumento #a0

#Resposta do lboco no $s0

beq $s0, -1, falsoPiramide

la $a0, ($t5)

beqz $t7, inverteBit #proxBlocoDeveSer = inverteBit(blocoTotal)

bne $s0, $t9, falsoPiramide

j inverteBit #proxBlocoDeveSer = inverteBit(blocoTotal)

#Inverte o valor binario de $s0

inverteBit:

mul $t9, $s0, -1

addi $t9, $t9, 1

add $t7, $t7, $s2

# add $t7, $t7, ($s2)

j loopMain

printaJeSai:

la $a0, ($t8)

li $v0, 1

syscall

li $v0, 10

syscall

#coloca no $s0 o valor 1, 0, ou -1

#puxa do valor a0

bloco:

#a0 == n

la $a1, ($a0) # a1 == n

li $t4, 2 #TOTAL

loopBloco:

blez $a1, voltaMain

subi $a1, $a1, 1

li $v0, 5

syscall # V0 = next

beq $t4, 2, ifTotalEDois

beq $t4, 1, totalEUm

beq $t4, 0, totalEZero

ifTotalEDois: #if total ==2

rem $t4, $v0, 2

j loopBloco

totalEUm:

rem $t3, $v0, 2

bnez $t3, loopBloco

j falsoPiramide

totalEZero:

rem $t3, $v0, 2

beqz $t3, loopBloco

j falsoPiramide

voltaMain:

la $s0, ($t4)

jr $ra

#Verifica numero inserido, e volta caso seja possivel piramidar

#caso nao seja possivel encerra com 'nao'

verificapiramide:

li $t1, 0 #j do for

la $t0, ($v0) #elementos = valor lido

la $t2, ($t0) #i = elementos

looppiramide:

bge $t1, $t0, falsoPiramide

sub $t2, $t2, 1

sub $t2, $t2, $t1

beqz $t2, verdadeiroPiramide

addi $t1, $t1, 1

j looppiramide

verdadeiroPiramide:

jr $ra

falsoPiramide:

li $v1, 0

la $a0, $nao

li $v0, 4

syscall

li $v0, 10

syscall

teste:

li $v1, 0

la $a0, $sim

li $v0, 4

syscall

li $v0, 10

syscall

.data

$nao:

.asciiz "não"

$sim:

.asciiz "sim"

Seção 5 – Referências:

* Patterson, D.A.; Hennessy, J.L. **Organização e Projeto de Computadores – A Interface Hardware/Software.** Campus, 2013, 4ª Ed, Capítulo 2.
* Stallings, W. **Arquitetura e Oragnização de Computadores.** Pearson, 2010, 8ª Ed., Capítulo10.
* http://msdn.microsoft.com/en-us/library/ms253512%28VS.80%29.aspx **Acesso em: 23/06/2017**
* http://www.doc.ic.ac.uk/lab/secondyear/spim/node13.html **Acesso em: 23/06/2017**
* http://logos.cs.uic.edu/366/notes/MIPS%20Quick%20Tutorial.htm **Acesso em: 23/06/2017**