Trabalho 2 de Organização e Arquitetura de Computadores

Simulador do MIPS

Professor: Ricardo Jacobi

Componentes:

Ingrid Santana Lopes (14/0083065)

1. Descrição do Problema

O trabalho proposto foi um programa feito em linguagem C/C++ ou Java de forma a reutilizando-se as funções lw, lh, lhu, lb, lbu, sw, sh e sb do trabalho anterior de forma que, dessa vez, fosse possível simular outras características do Assembly MIPS. Instruções adicionais foram implementadas conforme requisitado pelo documento em pdf de forma que o simulador tivesse uma maior capacidade de realização de tarefas do que previamente.

Dessa forma, ao se escolher a linguagem de programação como sendo C, foi feito um programa que seria capaz de ler o .bin que continha as instruções que seriam simuladas através do C. Demais funções foram implementadas além das simuladoras das requeridas funções de forma a auxiliar o comportamento das funções para que as mesmas estivessem dentro do esperado caso estivessem sendo rodadas através do Mars 4.5. Foi necessário também considerar a memória e o trabalho com a mesma uma vez que o MIPS tem a opção de exibir em hexadecimal os valores.

2. Descrição das funções implementadas

a. Funções em simulador.h

i. *int32_t lw(uint32_t address, int16_t kte)* = simula a função load word do MIPS onde considerando a tabela de funções do tipo I, address representa o valor de RS e kte o valor de Imm. Assim, para se executar a função lw $t0, k, uma construção ideal seria reg[8] = lw (0,8192) considerando que k está na posição 8192 (em valor decimal) da memória. Correspondente a 0x2000. Dessa forma, a função lerá o que está na posição 8192 da memória e colocará em reg[8] que simula a posição do registrador t0.

ii. *void sw(uint32_t address, int16_t kte, int32_t dado)* = simula a função store word do MIPS onde considerando a tabela de funções do tipo I, address representa o valor de reg[address], kte o valor de Imm e dado o valor presente no endereço que ele irá checar. Assim, para se executar a função sw $t1, k, uma

construção ideal seria sw (0,8192, reg[9]) considerando que k está na posição 8192 (em valor decimal) da memória. Correspondente a 0x2000 e que reg[9] corresponde ao valor alocado no registrador $t1.

iii. *int32_t lh (uint32_t address, int16_t kte)* = simula a função load half word do MIPS onde considerando a tabela de funções do tipo I, address representa o valor de RS e kte o valor de Imm. Assim, para se executar a função lh $t0, k, uma construção ideal seria reg[8] = lh (0,8192) considerando que k está na posição 8192 (em valor decimal) da memória. Correspondente a 0x2000. Dessa forma, a função lerá o que está na posição 8192 da memória e colocará em reg[8] que simula a posição do registrador t0.

iv. *int32_t lhu (uint32_t address, int16_t kte)* = simula a função load half word unsigned do MIPS onde considerando a tabela de funções do tipo I, address representa o valor de RS e kte o valor de Imm. Assim, para se executar a função lhu $t0, k, uma construção ideal seria reg[8] = lhu (0,8192) considerando que k está na posição 8192 (em valor decimal) da memória. Correspondente a 0x2000. Dessa forma, a função lerá o que está na posição 8192 da memória e colocará em reg[8] que simula a posição do registrador t0.

v. *void sh(uint32_t address, int16_t kte, int16_t dado)* = simula a função store word do MIPS onde considerando a tabela de funções do tipo I, address representa o valor de reg[address], kte o valor de Imm e dado o valor presente no endereço que ele irá checar. Assim, para se executar a função sh $s1, 8232($zero), uma construção ideal seria sh (0,8232, reg[17]), 8232 em decimal, Correspondente a 0x2028 e que reg[17] corresponde ao valor alocado no registrador $s1.

vi. *int32_t lb (uint32_t address, int16_t kte)* = simula a função load byte do MIPS onde considerando a tabela de funções do tipo I, address representa o valor de RS e kte o valor de Imm. Assim, para se executar a função lb $t2, i, uma construção ideal seria reg[10] = lb (0,8193) considerando que k está na posição 8193 (em valor decimal) da memória. Correspondente a 0x2004. Dessa forma, a função lerá o que está na posição 8193 da memória e colocará em reg[10] que simula a posição do registrador t2.

vii. *int32_t lbu (uint32_t address, int16_t kte)* = simula a função load half byte do MIPS onde considerando a tabela de funções do tipo I, address representa o valor de RS e kte o valor de Imm. Assim, para se executar a função lbu $t2, i, uma construção ideal seria reg[10] = lbu (0,8193) considerando que k está na posição 8193 (em valor decimal) da memória. Correspondente a 0x2004. Dessa

forma, a função lerá o que está na posição 8193 da memória e colocará em reg[10] que simula a posição do registrador t2.

viii. *void sb(uint32_t address, int16_t kte, int8_t dado)* = simula a função store word do MIPS onde considerando a tabela de funções do tipo I, address representa o valor de reg[address], kte o valor de Imm e dado o valor presente no endereço que ele irá checar. Assim, para se executar a função sb $s1, 8232($zero), uma construção ideal seria sb (0,8232, reg[17]), 8232 em decimal, Correspondente a 0x2028 e que reg[17] corresponde ao valor alocado no registrador $s1.

ix. *void addi, void add, void sub, void or, void ori, void and, void andi, void Div, void mult, void xor, void nor, void xori, void mflo, void mfhi, voir jr, void j, void jal, void bne, void beq, void slt, void slti, void lui, void bgtz, void blez, void sltiu, void srl, void sll* = Instruções adicionais requeridas pelo pdf. Foram separadas em funções de forma a visar uma melhor organização e a possibilidade de ocorrer uma necessidade de reutilização de alguma das funções. Assim, nesse caso, ficaria mais fácil recorrer a elas. Atenta-se ao fato de a função Div começar com letra maiúscula devido a problemas de pseudônimo com uma função de uma das bibliotecas utilizadas.

x. *int syscall ()*= função de syscall implementada considerando-se apenas a impressão de inteiros e strings e também o encerramento do programa de acordo com o que está presente no registrador $V0.

xi. *int execute()*=função responsável por determinar qual função será executada de acordo com o opcode extraído do hexadecimal presente em ri.

xii. *int step()* = função responsável por executar uma única instrução do MIPS

xiii. *void decode ()* = função responsável por decodificar a instrução presente em ri extraindo do registrador as informações necessárias para identificação da função e execução da mesma posteriormente.

xiv. *void fetch ()* = função responsável por pegar um número em hexadecimal presente na memória e o armazenar em ri, representado assim a próxima função que será executada.

xv. *void run()* = função responsável por, ao chamar a função step, executar todas as intruções armazenadas na área de código considerando as alterações que as mesmas fazem na área de data.

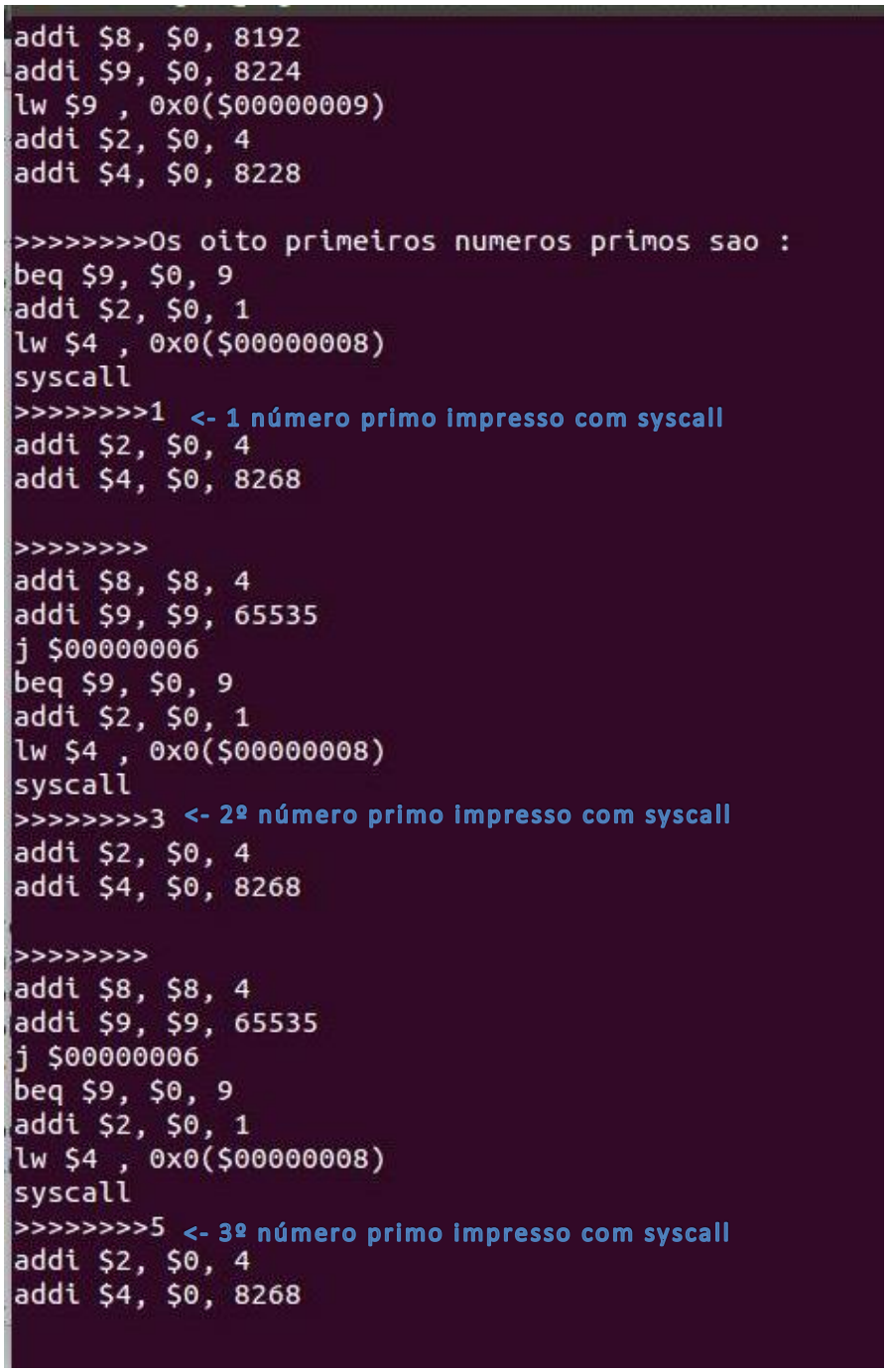
xvi. *void pegaHexa()* = função responsável por abrir data.bin e text.bin para armazenar os hexadecimais presentes nos mesmos na memória do simulador para que a simulação possa ser iniciada.

xvii. *void dump_mem(int start, int end, char format)* = responsável por imprimir o conteúdo da memória em um .bin considerando duas posições, uma de início e outra de término, e um formato de impressão. Atenta-se ao fato que start e end devem ser um número > 0 e < MEM_SIZE.

xviii. *void dump_reg(char format)* = responsável por imprimir o conteúdo dos registradores em um .bin considerando um formato de impressão desejado pelo usuário.

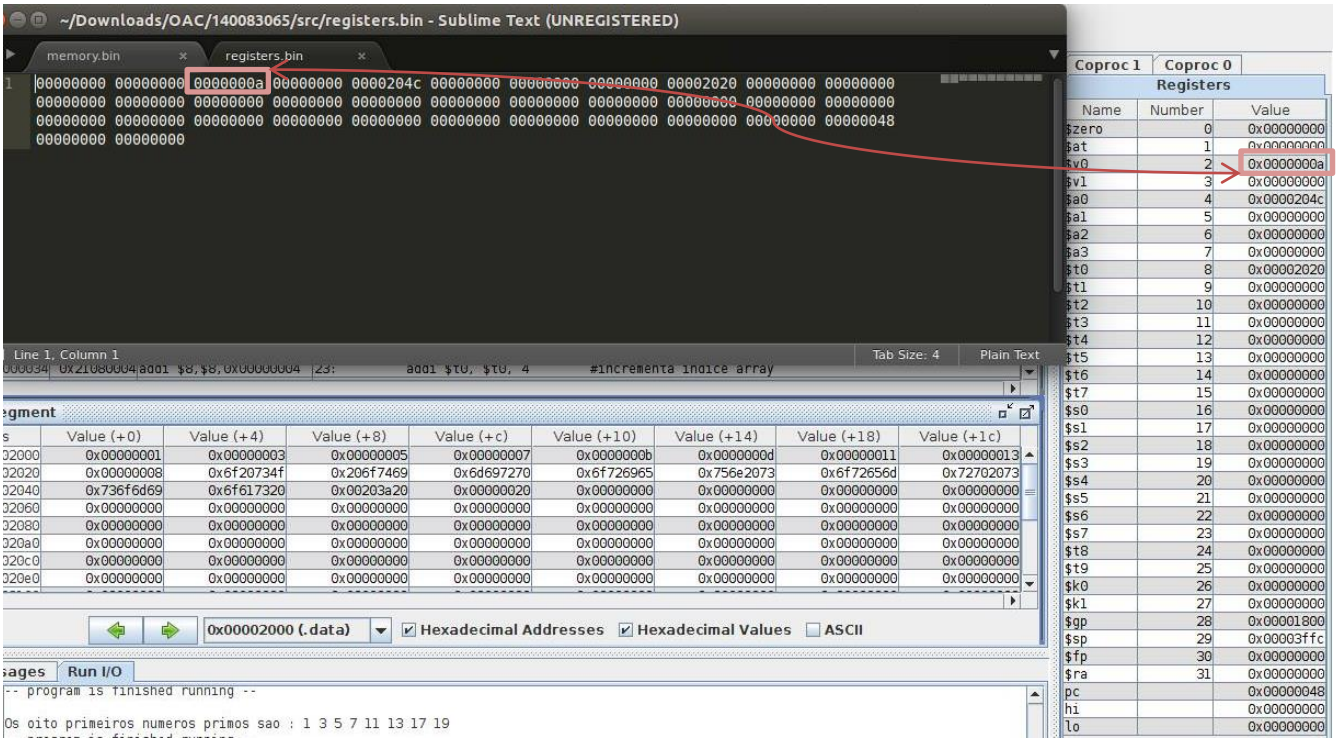
3. Testes e Resultados

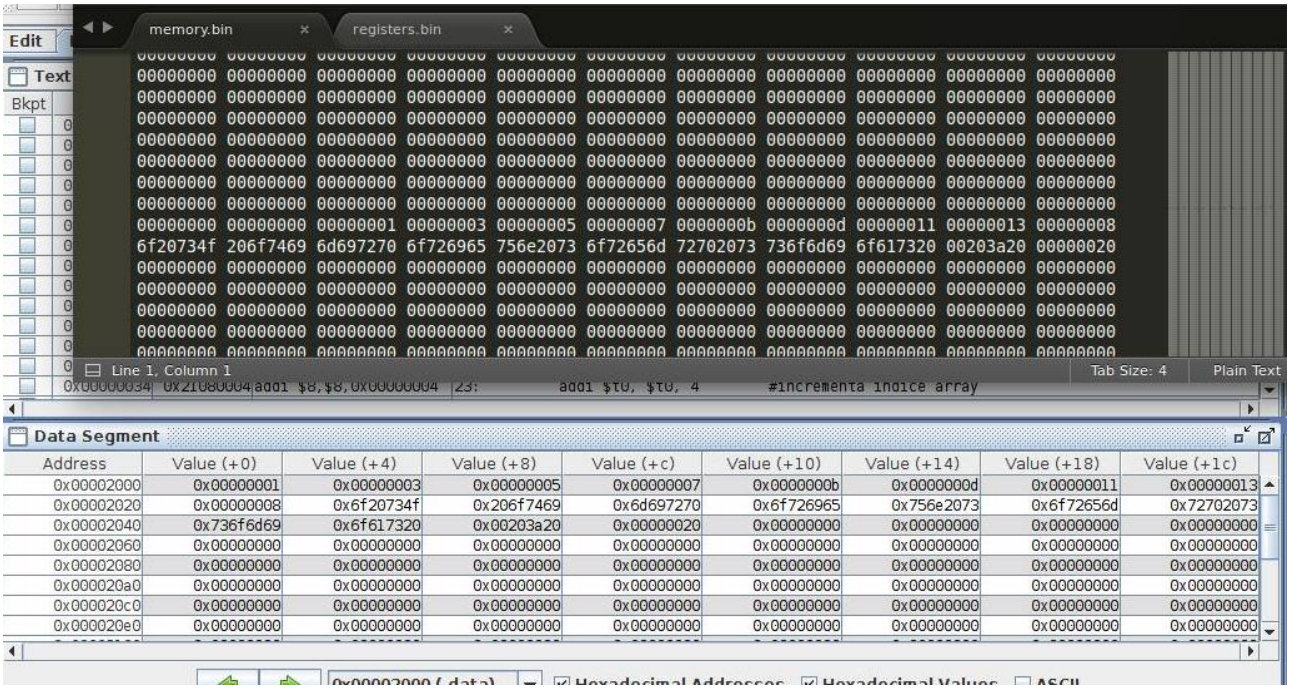
Os testes realizados com o simulador são mostrados a seguir:

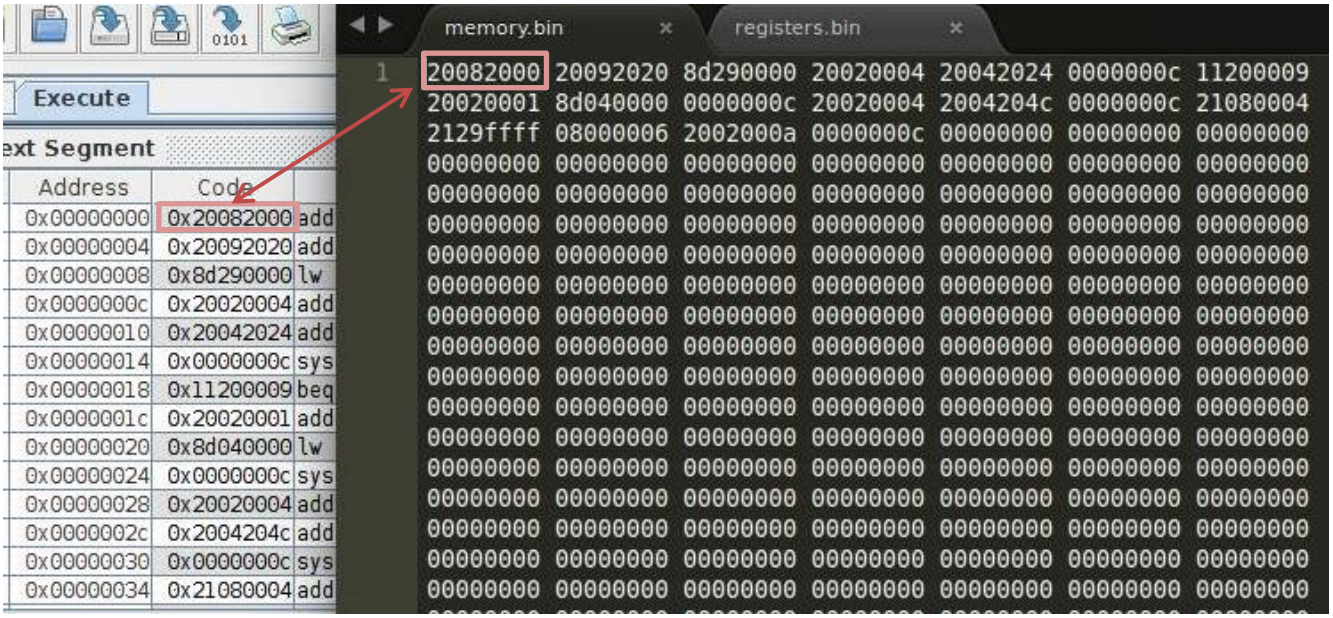


```
addi $2, $0, 4
addi $4, $0, 8268

>>>>>>>>
addi $8, $8, 4
addi $9, $9, 65535
j $00000006
beq $9, $0, 9
addi $2, $0, 1
lw $4 , 0x0($00000008)
syscall
>>>>>>>>7
addi $2, $0, 4
addi $4, $0, 8268

>>>>>>>>
addi $8, $8, 4
addi $9, $9, 65535
j $00000006
beq $9, $0, 9
addi $2, $0, 1
lw $4 , 0x0($00000008)
syscall
>>>>>>>>11
addi $2, $0, 4
addi $4, $0, 8268

>>>>>>>>
addi $8, $8, 4
addi $9, $9, 65535
j $00000006
beq $9, $0, 9
addi $2, $0, 1
lw $4 , 0x0($00000008)
syscall
>>>>>>>>13
addi $2, $0, 4
addi $4, $0, 8268
```

```
addi $2, $0, 1
lw $4 , 0x0($00000008)
syscall
>>>>>>>>13
addi $2, $0, 4
addi $4, $0, 8268

>>>>>>>>
addi $8, $8, 4
addi $9, $9, 65535
j $00000006
beq $9, $0, 9
addi $2, $0, 1
lw $4 , 0x0($00000008)
syscall
>>>>>>>>17
addi $2, $0, 4
addi $4, $0, 8268

>>>>>>>>
addi $8, $8, 4
addi $9, $9, 65535
j $00000006
beq $9, $0, 9
addi $2, $0, 1
lw $4 , 0x0($00000008)
syscall
>>>>>>>>19
addi $2, $0, 4
addi $4, $0, 8268

>>>>>>>>
addi $8, $8, 4
addi $9, $9, 65535
j $00000006
beq $9, $0, 9
addi $2, $0, 10
syscall
>>>>>>>>Terminate program.
```







4. Demais descrições

Linguagem Utilizada: C

Programas utilizados: Mars4.5, Sublime e Gedit

Compilador: GCC 4.8.4

Sistema Operacional: Ubuntu 14.04