Centro de Ciências Tecnológicas da Terra e do Mar
Universidade do Vale do Itajaí (UNIVALI)
Itajaí – SC – Brasil

Ciência da Computação

Arquitetura e Organização de Computadores

Prof. Cesar Albenes Zeferino

Avaliação 03 - Simulador MIPS

André Luiz da Silva, Ivan Carlos dos Santos
{silva.andre,ivancsantos}@edu.univali.br

10/05/16

1. Código fonte em Assembly - MIPS

```
.data
       vet    : .word        1,2,3,4,5,6,7,8
       size   : .word        8
       sum    : .word        0


.text
main:
       la $a0, size           # ponteiro para size
       la $a1, vet            # ponteiro para o vetor
       la $a2, sum            # ponteiro para sum

       lw $t0, ($a0)          # carrega o valor de $a0 (size) em $t0
       la $t1, ($a1)          # ponteiro para $a1 (vet) em $t1
       li $t2, 0              # carrega 0 em $t2 (registrador temporario para i)
       li $v0, 0              # carrega 0 em $v0 (resultados parciais de sum)
       jal proc_sum           # jump para soma, guardando pc+4 em $ra

       sw $v0, ($a2)          # guarda o resultado da soma na memória

       j fim                  # encerra o programa


proc_sum:                     # procedimento para o somatorio (laco)

       beq $t0, $t2, saida_laco      # desvia se $t2 (i) = $t0 (size)

                              # calcula o endereco de vet[i]
       sll $t4, $t2, 2        # armazena em $t4 o produto de i * 4
       add $t1, $a1, $t4      # endereço atual = endereço base do vetor + i * 4

       lw $t3, ($t1)          # carrega em $t3 o elemento A[i] apontado por $t1

       add $v0, $v0, $t3      # soma = soma + elemento atual do vetor

       addi $t2, $t2, 1       # i++
       j proc_sum             # retorna para o começo do procedimento proc_sum


saida_laco:
       jr $ra                 # retorna para instrução em $ra


fim:
       nop                    # fim do programa
```

2. Código fonte do simulador em C++

```
//R-format
    if (mips_ins.op_code == RTYPEOP) {
        switch (mips_ins.funct) {
        case SLL:
            this->RegFile[mips_ins.rd] = this->RegFile[mips_ins.rt] << mips_ins.shamt;
            break;
        case JR:
            return this->RegFile[mips_ins.rs];
        case ADD:
            this->RegFile[mips_ins.rd] = this->RegFile[mips_ins.rs] + this-
>RegFile[mips_ins.rt];
            break;
        default:
            printf("Instrucao nao encontrada!\n");
            exit(1);
            break;
        }


    //I or J-format
    }else{
        switch (mips_ins.op_code) {
        case J:
            this->pc += 4;
            this->pc = (this->pc & 0xf0000000) | (mips_ins.immediate_26bits << 2);
            return this->pc;
        case JAL:
            this->RegFile[31] = this->pc + 4;
            this->pc += 4;
            this->pc = (this->pc & 0xf0000000) | (mips_ins.immediate_26bits << 2);
            return this->pc;
        case BEQ:
               if(this->RegFile[mips_ins.rt] == this->RegFile[mips_ins.rs]){
                this->pc += 4;
                return this->pc + (mips_ins.immediate_16bits << 2);
            }
            break;
        case ADDI:
            this->RegFile[mips_ins.rt] = this->RegFile[mips_ins.rs] +
mips_ins.immediate_16bits; // RT = RS + IMMEDIATE
            break;
        case ADDIU:
            this->RegFile[mips_ins.rt] = this->RegFile[mips_ins.rs] +
mips_ins.immediate_16bits;
            break;
        case ORI:
            this->RegFile[mips_ins.rt] = this->RegFile[mips_ins.rs] |
mips_ins.immediate_16bits;
            break;
        case LUI:
            this->RegFile[mips_ins.rt] = mips_ins.immediate_16bits << 16;
            break;
        case LW:
            this->RegFile[mips_ins.rt] = get_DataMem(this->RegFile[mips_ins.rs] +
mips_ins.immediate_16bits);
            break;
        case SW:
            set_DataMem(this->RegFile[mips_ins.rs] + mips_ins.immediate_16bits, this-
>RegFile[mips_ins.rt]);
            break;
        default:
            printf("Instrucao nao encontrada!\n");
            exit(1);
            break;
        }
    }
    return this->pc+4;
}
```

3. Descrição da implementação do simulador

Desenvolvemos em linguagem assembly do processador MIPS um programa que calcula a soma de 8 elementos de um vetor, e ao final armazena esse valor na memória.

O programa inicializa a memória com os dados iniciais vet={1,2,3,4,5,6,7,8}, size = 8, e sum = 0. Para leitura do programa pelo simulador, gravamos esses valores em um arquivo que aqui representa a memória principal – “ram.hex”.

Da mesma maneira, as instruções que serão lidas pelo simulador foram salvas em formato hexadecimal em um arquivo “rom.hex”, que guarda nosso programa.

Ao ser executado o simulador, o mesmo lê os registros dos arquivos ram.hex e rom.hex e armazena os valores nos vetores InsMem[] e DataMem[].

O simulador então interpreta os códigos hexadecimal armazenados em InsMem, identificando o tipo de instrução a ser executada, e então desvia para o código apropriado à execução da instrução do processador MIPS. Durante toda a execução do programa são impressos a instrução que está sendo executada, valores armazenados na memória, e valores carregados nos registradores.

Na figura abaixo, ilustraremos como exemplo a execução do código assembly lw $8, 0($4).

Nesta instrução, será armazenado no registrador $8 o valor armazenado na memória apontada pelo registrador $4 com deslocamento 0.

a. Instrução sendo executada:
>lw $8, 0(4)

b. Ins 0x8c880000
>Instrução em hexadecimal

c. pc 0x00400018
>Contador de programa

d. Registradores

e. next pc
>pc+4

f. .DATA
>valores armazenados na memória princial

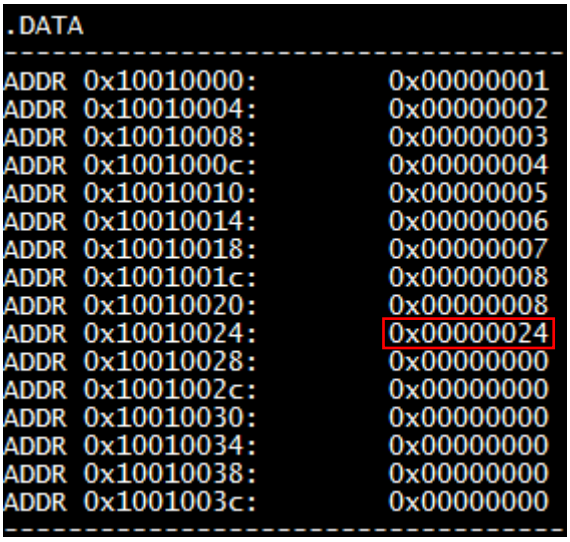
```
INSTRUCAO: lw $8, 0($4)
-----------------------------------
Ins                         0x8c880000
-----------------------------------
pc                          0x00400018
-----------------------------------
$zero   0                   0x00000000
$at     1                   0x10010000
$v0     2                   0x00000000
$v1     3                   0x00000000
$a0     4                   0x10010020
$a1     5                   0x10010000
$a2     6                   0x10010024
$a3     7                   0x00000000
$t0     8                   0x00000008
$t1     9                   0x00000000
$t2     10                  0x00000000
$t3     11                  0x00000000
$t4     12                  0x00000000
$t5     13                  0x00000000
$t6     14                  0x00000000
$t7     15                  0x00000000
$s0     16                  0x00000000
$s1     17                  0x00000000
$s2     18                  0x00000000
$s3     19                  0x00000000
$s4     20                  0x00000000
$s5     21                  0x00000000
$s6     22                  0x00000000
$s7     23                  0x00000000
$t8     24                  0x00000000
$t9     25                  0x00000000
$k0     26                  0x00000000
$k1     27                  0x00000000
$gp     28                  0x00000000
$sp     29                  0x00000000
$fp     30                  0x00000000
$ra     31                  0x00000000
-----------------------------------
next pc                     0x0040001c

-----------------------------------

.DATA
-----------------------------------
ADDR 0x10010000:            0x00000001
ADDR 0x10010004:            0x00000002
ADDR 0x10010008:            0x00000003
ADDR 0x1001000c:            0x00000004
ADDR 0x10010010:            0x00000005
ADDR 0x10010014:            0x00000006
ADDR 0x10010018:            0x00000007
ADDR 0x1001001c:            0x00000008
ADDR 0x10010020:            0x00000008
ADDR 0x10010024:            0x00000000
ADDR 0x10010028:            0x00000000
ADDR 0x1001002c:            0x00000000
ADDR 0x10010030:            0x00000000
ADDR 0x10010034:            0x00000000
ADDR 0x10010038:            0x00000000
ADDR 0x1001003c:            0x00000000
-----------------------------------
```

4. Capturas de telas – comparativo do resultado final

.DATA – simulador MARS

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000001	0x00000002	0x00000003	0x00000004	0x00000005	0x00000006	0x00000007	0x00000008
0x10010020	0x00000008	0x00000024	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

.DATA – Simulador do MIPS implementado em C++



Registradores – simulador MARS

Name	Number	Value
$zero	0	0x00000000
$at	1	0x10010000
$v0	2	0x00000024
$v1	3	0x00000000
$a0	4	0x10010020
$a1	5	0x10010000
$a2	6	0x10010024
$a3	7	0x00000000
$t0	8	0x00000008
$t1	9	0x1001001c
$t2	10	0x00000008
$t3	11	0x00000008
$t4	12	0x0000001c
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000000
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x0040002c
pc		0x00400058
hi		0x00000000
lo		0x00000000

Registradores – simulador do MIPS implementado em C++

```
-----------------------------------
Ins                         0x00000000
-----------------------------------
pc                          0x00400054
-----------------------------------
$zero   0                   0x00000000
$at     1                   0x10010000
$v0     2                   0x00000024
$v1     3                   0x00000000
$a0     4                   0x10010020
$a1     5                   0x10010000
$a2     6                   0x10010024
$a3     7                   0x00000000
$t0     8                   0x00000008
$t1     9                   0x1001001c
$t2     10                  0x00000008
$t3     11                  0x00000008
$t4     12                  0x0000001c
$t5     13                  0x00000000
$t6     14                  0x00000000
$t7     15                  0x00000000
$s0     16                  0x00000000
$s1     17                  0x00000000
$s2     18                  0x00000000
$s3     19                  0x00000000
$s4     20                  0x00000000
$s5     21                  0x00000000
$s6     22                  0x00000000
$s7     23                  0x00000000
$t8     24                  0x00000000
$t9     25                  0x00000000
$k0     26                  0x00000000
$k1     27                  0x00000000
$gp     28                  0x00000000
$sp     29                  0x00000000
$fp     30                  0x00000000
$ra     31                  0x0040002c
-----------------------------------
```